# Global Migration Dynamics

An Exploratory Study Integrating Multi-Resolution Modeling Techniques with Semiautomatic Data Acquisition

S.B.J. Wigman

Technische Universiteit Delft

TU Delft

# Global Migration Dynamics

## An Exploratory Study Integrating Multi-Resolution Modeling Techniques with Semiautomatic Data Acquisition

by

S.B.J. Wigman

in partial fulfillment of the requirements for the degree of

**Master of Science**
in Engineering and Policy Analysis

at the Delft University of Technology,
to be defended publicly on Wednesday August 29, 2018 at 15:30 PM.


| | | |
|---|---|---|
| Student number: | 4016246 | |
| Supervisor: | Dr. E. Pruyt, | TU Delft |
| Chair: | Prof. dr. M. J. van den Hoven, | TU Delft |


*This thesis is confidential and cannot be shared until August 29, 2018.*



TU Delft Delft University of Technology

# Summary

This study approached migration-related problems from a global perspective. The interconnected nature of migration means that migration problems accumulate and intensify throughout the world. To capture these effects, a data-rich model component was developed for global migration dynamics.

To that extent, this study has developed the following innovations on a methodological level: a technique to comprehensively implement spatial phenomena into System Dynamics, a method to implement multi-scale System Dynamics through subscript mapping, and a semi-automated data acquisition process to obtain and structure a great variety of data sets on a country level. The nature of the developed model component is generic. That is, the component is scalable, and resolution-independent, as it is initialized through external databases. Therefore, a significant part of this study comprised the development of a semiautomated data acquisition process. The model component was then connected to a root-causes model component to analyze future migration scenarios on a global scale. It was found that migration is expected to increase in line with population growth.

Policies were implemented and tested on a global, regional, and national scale. In the global policy case, the location of additional shelter and coping capabilities was found to greatly influence migration dynamics across the world. Additional shelters in low-income countries greatly reduced the number of migrants traveling towards the more wealthy countries. The regional policy—closure of the European borders when a certain threshold is reached—had significant effects on the countries near the border, but failed to significantly improve the situation in the rest of Europe. This can be attributed to the delayed effects of such policies in these countries. National policies were tested for the Netherlands. Effects of these policies were insignificant, due to the inertia of the migration system. The number of migrants in Germany—a neighboring country—did not significantly change, and even the effect on the number of migrants in the Netherlands was negligible.

Therefore, the first advice from this study towards policy-makers is that it might be more effective to address the root causes of migration, rather than just mitigating its detrimental effects. In addition, the inertia of the migration system requires globally aligned, large-scale policy implementation to prevent and mitigate migration-related problems. The delay in these policies should not be neglected, so it is important to identify potential problems as early as possible and take appropriate action.

# Acknowledgements

The choice to switch to Engineering and Policy Analysis after my Bachelor's degree in Applied Earth Sciences was one of the best decisions I ever made. It gave me the opportunity to work on intriguing projects and problems, meet awesome new people, and make lots of new (international) friends. Although it was tough at times, and my stress level during the past two years has reached new heights, I was always able to put things in perspective and see the bigger picture.

In particular, I would like to express my uttermost gratitude to Erik Pruyt for his *continuous* support and guidance and for giving me the opportunity to experiment and develop new stuff in his courses, his (our?) projects, and during my own thesis. I would also like to express my appreciation to the chair of my committee: Jeroen van den Hoven.

Thanks to everyone who has made my student life worth it, my fellow students at 'Mijnbouw', all my fellow students at EPA, and of course my friends in Rotterdam for countless 'nights to remember'. Thank you all!

Of course I would also like to thank Esther, without whom I would probably not have gotten to where I am. You have given me the support on moments when I needed it the most.

And last, but definitely not least, I would like to thank my parents Ben and Annemie for their encouragements and unconditional support for the entire duration of my studies.

*Stefan Wigman, 15 August 2018.*

# Contents

# List of Figures

# List of Tables

# 1

# Introduction

This study aims to develop a data-rich model in order to explore possible future migration flows—and their underlying root causes—from a global perspective. The developed model will then be used to assess how such migratory flows develop in various plausible scenarios, and how they respond to national, regional, and global migration policies. The goal of this study is twofold: to encourage and stimulate the debate about migration, and to illustrate that a more extensive integration of data science with modeling techniques can make both more useful. The remainder of this chapter will further introduce the topic and scope of this study.

## 1.1. A Pressing Problem?

Over the last few decades, the debate in Europe about migration and immigrants has become increasingly fierce (Geddes and Scholten, 2016; Lucassen, 2018). The migrant problem in Europe reached a climax in 2015, when—according to the most conservative estimates—over one million refugees and migrants crossed the Mediterranean Sea into Europe (BBC News, 2018). A schematic overview of the asylum applications per European country and the migrants' country of origin during this European migrant crisis can be found in Figure A.1. The arrows in this figure depict the main routes used by migrants to migrate into Europe. These inflows caused a ”European migrant crisis” which is still ongoing, despite the low numbers of refugees and migrants currently arriving in Europe (The New York Times, 2018).

The term ”European migrant crisis” is arguable, as it only refers to the destination of the migrants. Suppose that the European borders were closed during the time of the crisis, and that no migrants or refugees would have been able to make the crossing towards the Southern European countries. In this case, the migrant crisis in Europe might have been averted, but would this also mean that there is no migration crisis in general? Presumably not, since the actual problems that were faced in 2015 did not start at the European borders. The conditions in various Middle-Eastern, Asian, African, and even some European countries caused a massive outflow of refugees and economic migrants. This—more extensive—spatial aspect of the 2015 migration problem is depicted in Figure 1.2. The map shows the widespread variety of countries that were dealing with an excessive outflow of migrants during 2015. It is well known that numerous refugees fled from Syria and Afghanistan, but also roughly 50 thousand Kosovars left their home during the migration crisis—albeit for economic reasons rather than persecution, war, or violence (Friedrich Ebert Stiftung, 2015).

Consequently, this can lead to some interesting questions: where are the actual problems regarding migration that need to be solved located? On the European borders? Or do we need to look further? And what actually caused these massive migration waves in the first place? And, as such, could it be more effective to address the problem(s) in the country of origin of migrants, rather than to wait until their migration starts to cause problems in Europe? Is it possible to negate some of the detrimental effects of migration early on in the migration process, for instance by supplying neighboring countries with the means to provide

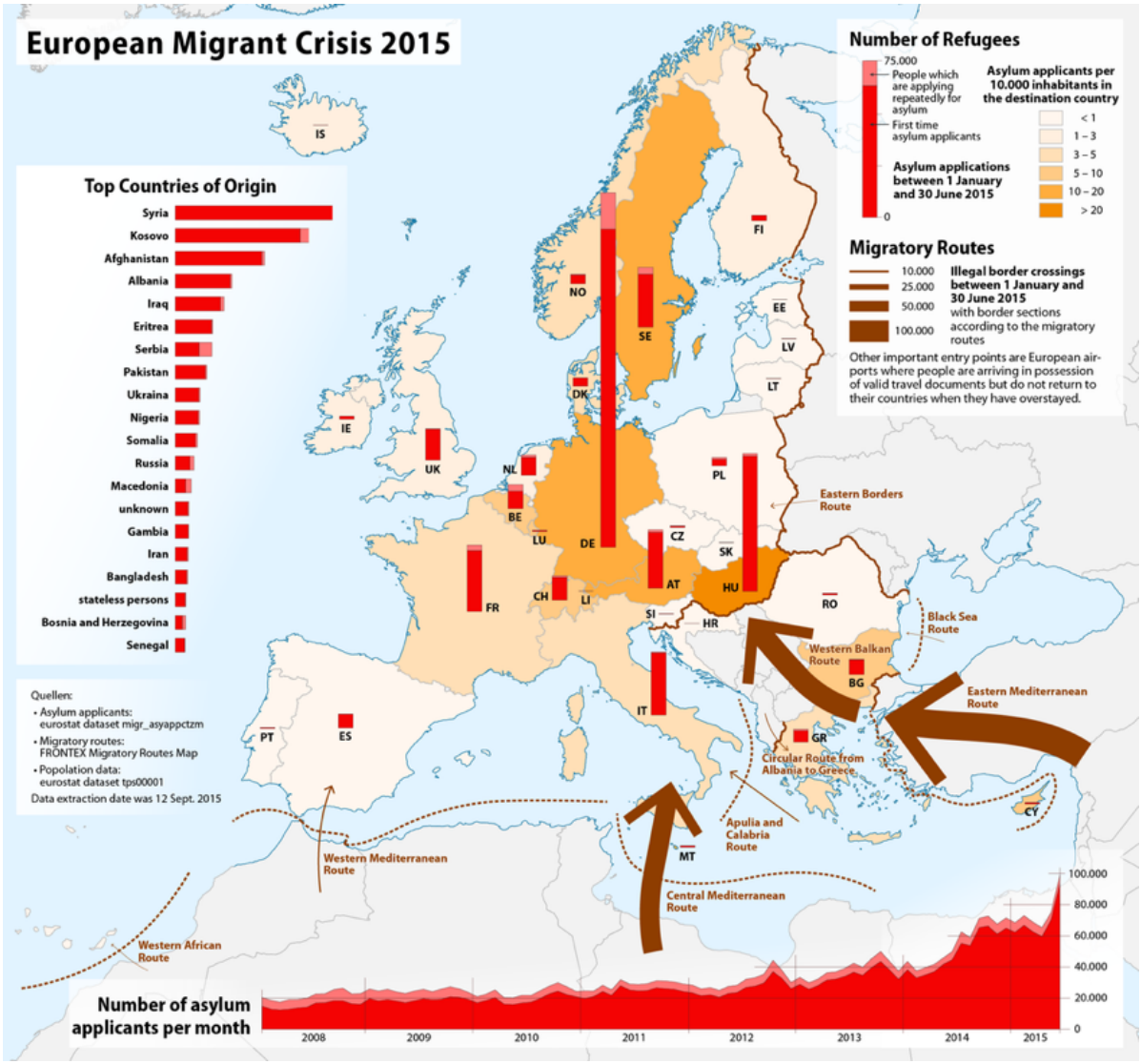



Figure 1.1: Asylum applications during the European migrant crisis in the European Union (EU) and European Free Trade Association (EFTA) states between January 1 and June 30 2015 (Dörrbecker, 2015a).

adequate shelter? The answers to these questions are not trivial. Essentially, this means that migration problems—and the migration process in general—should not be considered in an excessively isolated manner, since it is not just a local, national, or regional process. It is a global phenomenon that links all continents, countries, and even municipalities or neighborhoods, directly or indirectly into one immense network. As such, policy-makers, politicians, but also scientists should always take into consideration that this is in fact a global problem that does not start at the border of their region or country.

The notion that migration is not just confined to certain regions or countries, is supported by the almost simultaneous uprising of multiple migration crises in addition to the European version. For instance, the Venezuela-Colombia migrant crisis mid-2015, where Colombians, living in Venezuela, emigrated massively after some violent incidents (Alvarez and Marcaletti, 2018; Symmes Cobb, 2015), and the Rohingya refugee crisis, where mass migration occurred among “stateless entities” in Myanmar (Bandopadhyay, 2017).

Furthermore, it should be taken into consideration that a significant part of (forced) migration takes place within the borders of a certain region or country. Within Africa, estimates indicate that almost 13 million people lived in internal displacement at the end of 2016 (Internal Displacement Monitoring Centre and Norwegian Refugee Council, 2017). Internally

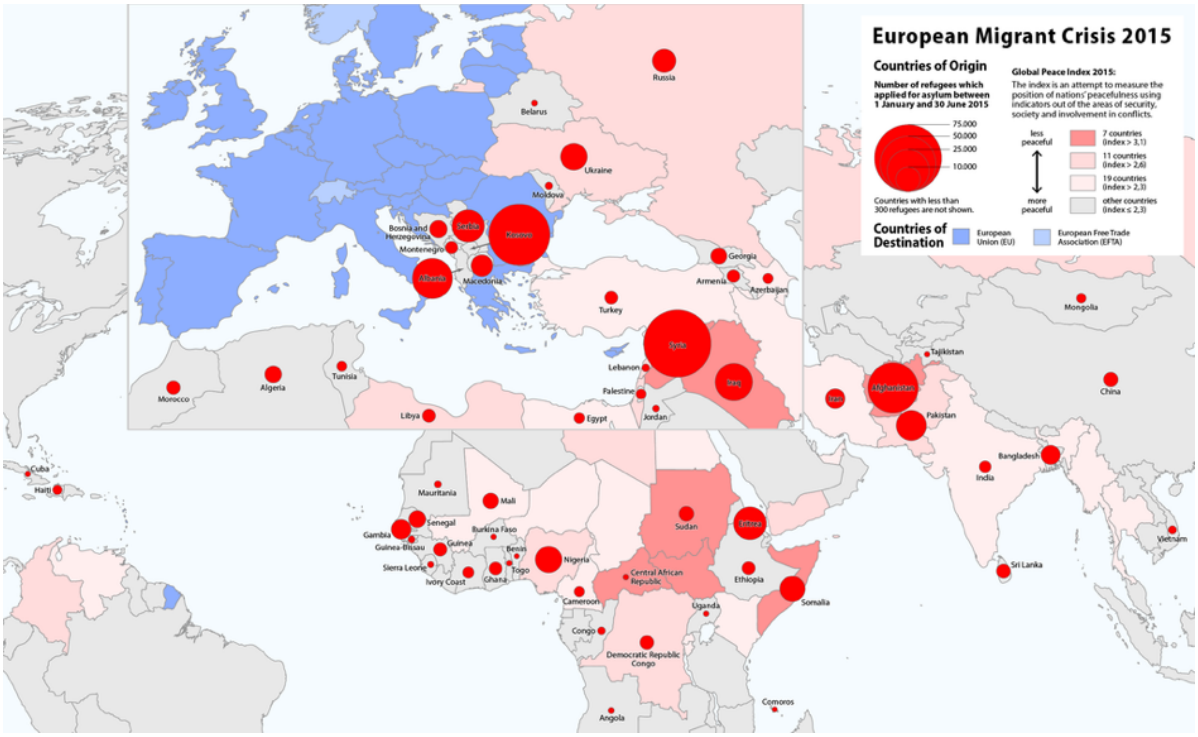



Figure 1.2: Countries of origin for the European migration crisis in 2015 (Dörrbecker, 2015b).

displaced persons (IDPs) are forced to move due to conflict and violence, disasters, or other root causes—but do not cross international borders. Therefore, these migrants often do not classify as refugees—persons that are forced to leave their country in order to escape war, violence, persecution, or natural disaster—and are consequently not entitled to protection by international law (International Committee of the Red Cross, 2017).

An additional issue concerning these recent migration crises, is the polarization of nations that are faced with a significant rise in migrants crossing their borders (Williams, 2009). Over the years, migration seems be accompanied more and more by a negative connotation, but it is in fact migration that has allowed human society as a whole to evolve to where we are now. That is, the massive spread of technology, innovations, knowledge, and science throughout history can be largely attributed to migration (Skeldon, 2014). This is an important realization for anyone dealing with migration problems. Social remittances can lead to cultural diffusion (Levitt, 1998), which can lead to significant—arguably positive—national or regional change (Levitt and Lamba-Nieves, 2011). Therefore, when considering migration from a positive point of view, it could also be seen as a process that distributes wealth, means, and technology more equally across the world. Looking at it from this perspective, it could even be argued that there has not been enough migration (and consequently mixing of cultures) over the last few centuries.

So, what would be the correct perspective to look at migration? Unfortunately, this is a very complex question that is difficult to answer. If we look at this phenomenon from a global perspective, we see that humans have divided the world into continents, regions, countries, states, provinces, cities, villages etc. However, resources are not distributed equally between all of these places, and as a consequence people migrate between these places for various reasons: conflict, persecution, repression, suppression, economic factors, scarcity, natural disasters, a lack of opportunities, family reunification, and undoubtedly many more. This migration takes place in an increasingly complex and interconnected world due to improved transportation, globalization, and digitalization. Solving all or even part of the problems that are inevitably intertwined with migration, is next to impossible. But while addressing these problems, it is important to realize that migration is—fundamentally—advantageous to mankind.

Actually, migration is one of the most important aspects in modern society, as it is a vital mechanism in the progress of societies. It is therefore crucial that any problems caused by this phenomenon are addressed appropriately and adequately. In order to tackle future issues related to migration, it first needs to be assessed how migration might evolve in the future on a global scale. To that extent, this thesis will employ a systems modeling approach to study migration across the globe.

## 1.2. Scale and Computational Limits

Having argued in the previous section that migration is in fact a global problem, which involves all ”bounded areas” in the world, this study approaches it as such. In order to tackle migration-related challenges on a micro-scale, it is crucial to first gain insight into what we can expect on a global scale—since this can provide a context within which these smaller problems should be solved. A systematic approach (i.e. modeling), can be very useful in this regard, as it allows for a systematical description of global migration processes within certain spatial and temporal boundaries. However, in tackling such a huge—and almost infinitely complex—phenomenon with any modeling approach, computing power might be a major limiting factor.

Modeling software and computing power have both developed significantly over the last few decades. Large, data-rich models have become the benchmark of modern science. Still, consider the following simple calculation: If one would like to model migration between continents, the first thing that one would like to keep track of is the population in each one of them: Africa, Asia, Europe, North America, South America, Antarctica, and Australia/Oceania. In total, this sums up to seven continents. Now suppose that people would be free to travel to any continent directly. From each continent, one has the option to travel to 6 other continents. This would lead to $7 \cdot 6 = 42$ migration flows. Now also suppose that the origin of each migrant needs to be tracked. For each continent, this adds another $7 \cdot 6 = 42$ variables. The total number of variables—solely to track where migrants are traveling to and where they come from—is then 91. This number of variables is perfectly manageable, but the rough estimates provided in Table 1.1 show that the complexity increases extremely fast with the number of unique divisions.

| Type | Total number of type | Number of flows | Origin tracking | Total number of variables |
|---|---|---|---|---|
| Continents | 7 | 42 | 42 | 91 |
| Countries | 217 | ~50 thousand | ~50 thousand | ~100 thousand |
| Provinces | ~1720+ | ~3 million+ | ~3 million+ | ~6 million+ |
| Cities | ~50 thousand+ | ~2.5 billion+ | ~2.5 billion+ | ~5 billion+ |

Table 1.1: A simple calculation performed for the number of variables needed to track migrants and their origins on various scales.

But this is only the number of variables needed to track migrants across the world in each case. Since such a model would also require a representation of the environment in each area, what that environment means for people currently residing in that area, and some sort of a decision process for migrants on if and where they travel, it is evident that implementation of migration dynamics on a global scale might not be straightforward.

For instance, suppose a high-level programming language like Python would be used to make a model for migration dynamics. Even the simplest attempt to keep track of migrant distribution in a Python model for all countries could already become problematic quite fast. The maximum size of a Python list (a list is a common data structure in programming languages) on a 32-bit computer system is approximately 537 million. This means that—when using Python lists to store results—100 thousand variables can be saved for roughly 5 thousand time steps. Therefore, modeling migration on this scale can rapidly approach the edge of computational limits, and extreme caution should be taken in constructing and defining models on this scale.

As such, an important aspect of this study will be to constantly evaluate the trade-off between the added value of elements that are taken into account, and the added complexity they

bring with them. The simple demonstration of how fast complexity increases, will have to be taken into consideration at all times. Presumably, any significant element that is added will cause a snowball-effect that might lead to severe, unworkable complications. Consequently, this study will also need to address the viability of the quantitative implementation regarding the various elements of the migration concept.

## 1.3. Project Scope and Research Objectives

Section 1.1 addressed the relevance and scope of the migration phenomenon, and its corresponding challenges. It was illustrated that it is in fact a complex, spatial phenomenon on a global scale. Problems and situations in certain regions resonate throughout the world and cause additional problems and challenges in other, far-away places. To engage this topic, this study will therefore approach the subject from a top-down, global perspective. It will also entail a constant trade-off between completeness (in capturing the migration phenomenon) and computational limits, as was elaborated on in Section 1.2.

Various modeling techniques could be used for addressing this issue. System Dynamics (SD) is one of these techniques. It is a modeling method that is commonly used to model nonlinear behavior of complicated systems over time. This continuous-time method uses stocks, flows, time-delays, and feedback effects to gain better understanding of complex issues and problems (Pruyt, 2013). Its high level of aggregation makes SD an appropriate modeling method to model global issues, and this has been done on multiple occasions (e.g. Forrester, 1971; Meadows et al., 1972; Simonovic, 2002). Therefore, to model the underlying root causes for migration (economic situation, conflict, drought, scarcity, etc.) SD is very well suited. In addition, since SD modeling can be combined with exploratory modeling and analysis, it is a very versatile approach to tackle societal grand challenges. It allows for the systematic exploration of different hypotheses related to model formulation and model parameterization, and their effect on the kinds of behavioral dynamics that can occur (Kwakkel and Pruyt, 2015). However, for problems with a significant spatial factor, such as migration dynamics, SD is not an obvious choice. Although developments in this area are taking place, e.g. by coupling SD software to a Geographic Information System (GIS) (Neuwirth et al., 2015), these are usually addressing relatively small-scale problems. In addition, these developments are mostly aimed at combining SD with other modeling methods, and often do not involve—or allow for—flows between modeled entities. Coupling of software or modeling methods brings about a considerable level of complexity in terms of implementation, while also significantly decreasing model manageability in terms of simulation time. Instead of integrating SD with another method, it might be more beneficial to explore the possibilities of implementing a spatial factor in SD itself.

Another consideration to take into account is that a global migration model should also incorporate the differences between regions, countries, or any areas between which people can migrate. To that extent, one of the essential aspects—and also limitations—of this study is the significant dependence on data. The recent digitalization and globalization has unquestionably increased data availability, but this does not mean that actually employing this data in a System Dynamics model is trivial. In particular, although the combination of SD modeling and data science is promising, data science and machine learning techniques should first be further developed to provide useful inputs for simulation models (Pruyt et al., 2014). The data problem is amplified by additional issues that emerge when dealing with data from multiple databases. Problems that quickly arise are: unavailability of data, poor data quality, integration of conflicting and/or redundant data, algorithm incompatibility across datasets, poorly structured data, and many more. For example, the World Bank specifies numerous indicators for 217 countries, while the United Nations only provide data for 193. In order to correct for this difference, either countries need to be dropped from the analysis, or missing data needs to be corrected for. This study also aims to address these issues thoroughly.

Together, these considerations lead to the following research objectives:

1. Explore the extent to which System Dynamics can be used to forecast potential future migration flows.

   (a) Develop a System Dynamics model component for migration dynamics that tracks the origin of migrants, and can—consequently—distinguish between them.
   (b) Couple this model component to a global root causes model.

2. Develop a significantly automated data acquisition process in order to extract data from a multitude of databases.
   (a) Establish a versatile, accessible, and adaptable data mining method with the purpose of effortlessly converting between, and combining, multiple datasets.
   (b) Assess available data sources for their utility and employ the developed data mining method to assemble and structure the required data.

3. Explore the future of migration using the coupled model for migration dynamics and root causes of migration.
   (a) Define a set of plausible future scenarios, and their uncertainty ranges.
   (b) Define a set of plausible future policies on a global, regional, and national scale.
   (c) Determine the impact of the various root causes, policies, and other uncertainties on future migration dynamics.

## 1.4. Outline

This thesis starts by providing a historical overview of migration, demarcating the concept of migration for this study, an assessment of its complexity, and a brief discussion on migration ethics in Chapter 2. This chapter will lead to some important considerations that should be taken into account when addressing this topic. Chapter 3 looks at migration from a systems perspective, and provides an overview of the elements and principles that served as a basis for the model development in this study. The actual implementation is further elaborated on in Chapter 4, where the developed model is described in detail. Chapter 5 presents the results that were obtained by simulating the model in its uncertainty space, and the effect of global, regional, and national policies on the behavior of variables of interest. Chapter 6 discusses the results, and reflects on how these should be interpreted in light of the considerations from Chapter 2. It also addresses the limitations of the current model implementation and its dependence on data. The chapter concludes with the resulting policy advice. Finally, in Chapter 7, the scientific contributions of this study are discussed, and recommendations for future research are made.

# 2

# Migration

This chapter will gradually introduce the topic of migration. It will start off in Section 2.1 by exploring the relevance of migration throughout the past, in order to illustrate that migration is a phenomenon that is inseparably linked with human history—and undoubtedly with the future. Then, in Section 2.2, the concept of migration is demarcated for this study by answering the following question: what classifies as migration? In Section 2.3, the complexity of migration is assessed on an abstract level, by considering some relatively simple thought experiments. Section 2.4 briefly discusses some of the ethical difficulties that arise in addressing migration-related problems. Throughout the chapter, several considerations will be formulated. These considerations will help to place this study in the correct perspective, and provide an outset for the discussion in Chapter 6.

## 2.1. A Historical Perspective

> History in its broadest aspect is a record of man's migrations from one environment to another.
>
> *Ellsworth Huntington*
> *1876-1947*

In search for an alternative route to the Indies, Christopher Columbus—unintentionally—"discovered" the Americas in 1492 (Rickey, 1992). Subsequently, this new land was colonized by Europeans over the course of a few centuries, laying the foundation for the American countries as they are known at present. One of these countries, the United States of America, is now generally considered as one of the most powerful countries in the world (Wilson III, 2008), and this is a prime example of the considerable influence of migration on human history. But even long before the Europeans were able to cross the Atlantic Ocean, humans had already moved to the Americas across the Bering land bridge (Hopkins, 1959). Figure 2.1 gives a schematic overview of prehistoric migration—including migration across the Bering land bridge, which is estimated to have happened after 15,000 B.C. (Reich et al., 2012).

Subsequently, the invention of clothes and housing enabled humans to break through climatic and geographical barriers, resulting in an almost unrestrained population growth until the end of the great migration in ca. 8000 B.C. (McNeill, 1984). In addition, agricultural developments facilitated further human population growth by enabling an increase in food supply (Vasey, 2002). After harvesting the crop, early farmers burned the field and moved onwards to find new fertile soil elsewhere. As a consequence of this nomadic behavior, crops like wheat and barley spread around all of Eurasia within a few millennia. This type of migration can be defined as whole-community migration (Manning, 2012), and it mainly transpired among early human communities.

From about 4000 B.C., maritime developments made it possible to sail the seas, and populate offshore islands (Gray and Jordan, 2000). In addition, biological changes gave

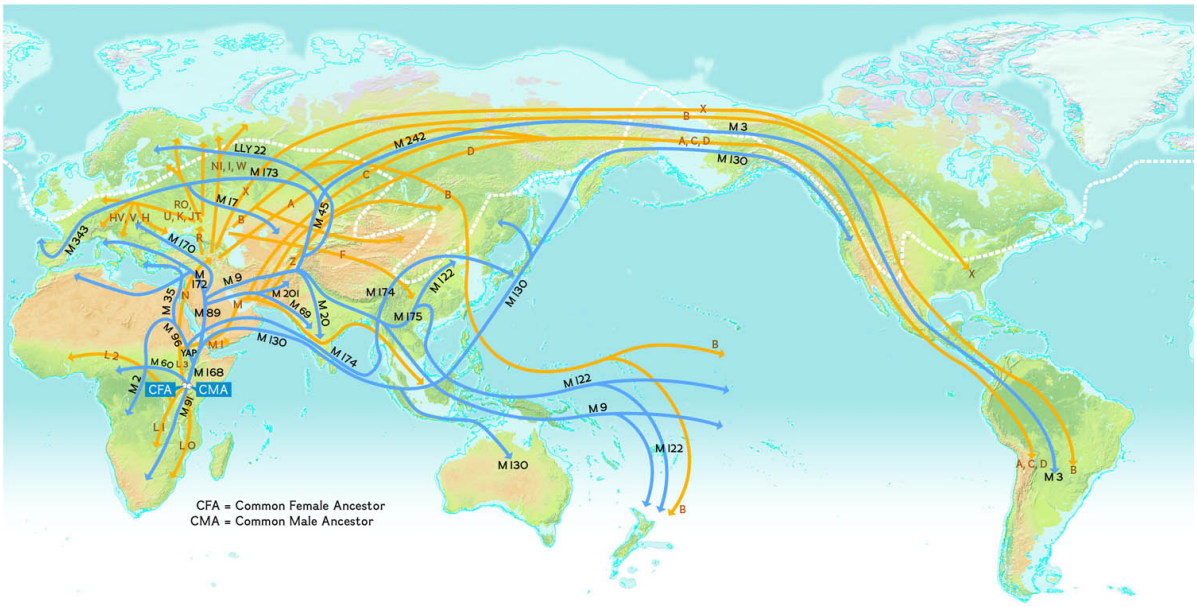



Figure 2.1: Three distinct waves of prehistoric migrants from Asia to North America, based on genetic analyses (National Geographic Society, 2008).

humans the capability to consume animal products, leading to nomadic communities on the plains in Eurasia, Africa, and the Middle East. The emergence of these mobile communities ensured that any useful innovations could spread relatively fast throughout the continent, and the first civilizations ensued shortly after (McNeill, 1984).

These early civilizations are generally regarded as more complex societies, with a slowly developing social stratification as key element. Community members were no longer limited to hunting or farming as main occupation, but could choose alternative specializations (e.g. military or craft). These specializations resulted in a military dominance by civilized communities in most encounters with their barbaric counterparts. As a consequence, communities were forced to evolve and—to that extent—adopt the civilized approach in order to safeguard themselves from adjacent, more developed communities.

At this point in time (ca. 2000 B.C.) new forms of migration started to emerge. Diversification on a social level meant that societies could now benefit from adopting outsiders. These new forms of migration included the merging of communities by conquest, (in)voluntary incorporation of individuals from one community into another, and slavery. It is around this time that the first towns and cities arose (Bairoch, 1991).

Towns and cities struggled with numerous infectious diseases and periodic food shortages, which meant that they were dependent on an inflow of rural people and their supplies. This means that rural communities were in fact the main driver for survival for these early civilizations, since cities themselves were unable to prevent a long-term population decay. A net excess of children in the rural areas in the long-term, provided cities the opportunity to preserve and pass along artisan skills to next generations. On the other hand, innovations in agricultural methods that were developed in the more civilized parts of societies sometimes led to migration from urban to new—previously uncultivated—rural areas. This expansion of cultivated land both extended the border of the civilized society and enabled further population growth. These opposite migratory patterns were an essential element required for the further development of civilizations (McNeill, 1984).

In addition to these inner-society migrations, military and merchant migration patterns contributed further to the spreading of civilized principles over even longer distances. Knowledge, skills, and religions, as well as diseases, were brought from one society to another by traveling merchants and religious preachers (McNeill, 1984). While this facilitated the spread of innovation and technology, the spread of diseases across regions by these travelers caused severe problems in various ancient empires (e.g. the Roman and Han empires) and was prob-

ably the accelerating factor in their downfall. Although these epidemics killed the major part of the population, survivors developed a resistance to some of these diseases, ensuring the survival of at least part of the urban population. During this time, humans slowly advanced to spread across the entire globe, populating essentially all inhabitable regions during several large migration waves: e.g. the Austronesian migration (Figure 2.2) (Gray and Jordan, 2000; Matisoo-Smith, 2015), the Indo-Aryan migration (Bamshad et al., 2001), and the Greek and Roman empire expansions (Boardman et al., 2001).

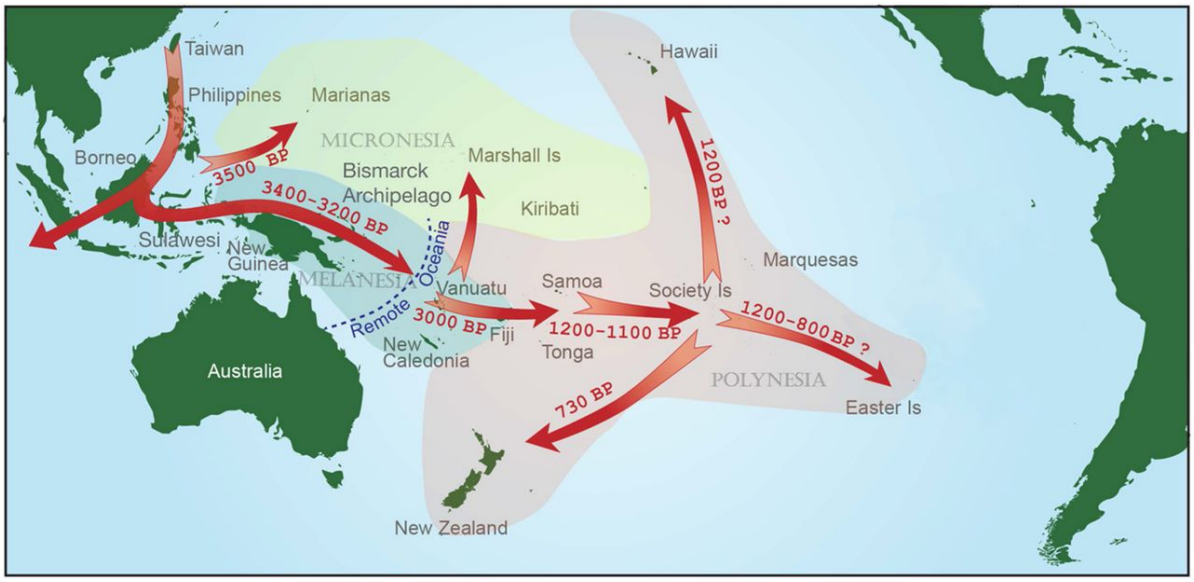



Figure 2.2: Schematic overview of Austronesian expansion. Dates are expressed in years before present (BP) (Matisoo-Smith, 2015).

During the next millennium, various other types of migration made their entry: colonization, various forms of (regional) slave practices, and a slowly intensifying form of labor migration. These types of steady, relatively low-paced, migration processes continued until the Age of Exploration and European colonialism. By that time, the industrial revolution facilitated migration through the development of more accessible, faster, and safer transportation from the mid-19th century onwards. Combined with stacked European cities, this led to massive migration towards oversea, less-populated territories. During the 19th century, over 50 million people departed from Europe to the Americas (Eltis, 1987).

This mass migration was accompanied by a gradual global increase in urbanization. Another important development is the genesis of Romantic nationalism, which is generally assumed to originate in 1848. This development led to a change in perspective in a significant number of countries, where ethnocentrism became the norm (Omohundro, 2008). It also further stimulated migration, as a distinction was made between ethnicities, causing people to move to more tolerant countries. In addition, migration rates increased significantly due to a rise in transnational labor migration until the early 20th century.

The next major drivers of migration are the First and Second World Wars. The dissolution of the Ottoman Empire during World War I resulted in a migration flow of Muslims from the Balkan into Turkey, while Christians moved the opposite way (Şeker, 2013). On the other side of the Black Sea, the Russian Civil War resulted in the migration of roughly 3 million Russians, Poles and Germans out of the Soviet Union to its Western neighbors. During World War II, a variety of (mostly forced) migration phenomena were predominantly present: deportation, forced displacement, mass evacuation, and expulsion (Castles et al., 2013). In 1943, this led to the origination of the United Nations Relief and Rehabilitation Administration, which was the predecessor of the presently well-known United Nations High Commissioner for Refugees (UNHCR). This was the first large-scale international relief agency, and provided aid to more than 8 million refugees. The aftermath of World War II led to the expulsion of 16.5 million Germans from Eastern Europe, and the expelling of several millions of Eastern-

Europeans out of the Soviet Union (Hoffmann-Nowotny, 1978).

After the Second World War, migration has slowly evolved into the form as we are familiar with in the present day. In this form of migration, individuals move from one community to another, and it can be defined as cross-community migration (Manning, 2012). It has turned into a topical issue due to a recent occurrence of massive migration waves throughout the world. For example, nearly 60 thousand individuals registered in the Netherlands as asylum seekers in 2015, while this number was below 15 thousand in 2010 (Centraal Bureau voor de Statistiek, 2018). This increase is typical for the trend that can be seen in the past five years, where migrant flows to Europe have risen exponentially since 2010 (Figure 2.3). In 2015 and 2016, the yearly number of asylum applications was over 1.2 million in the European Union.

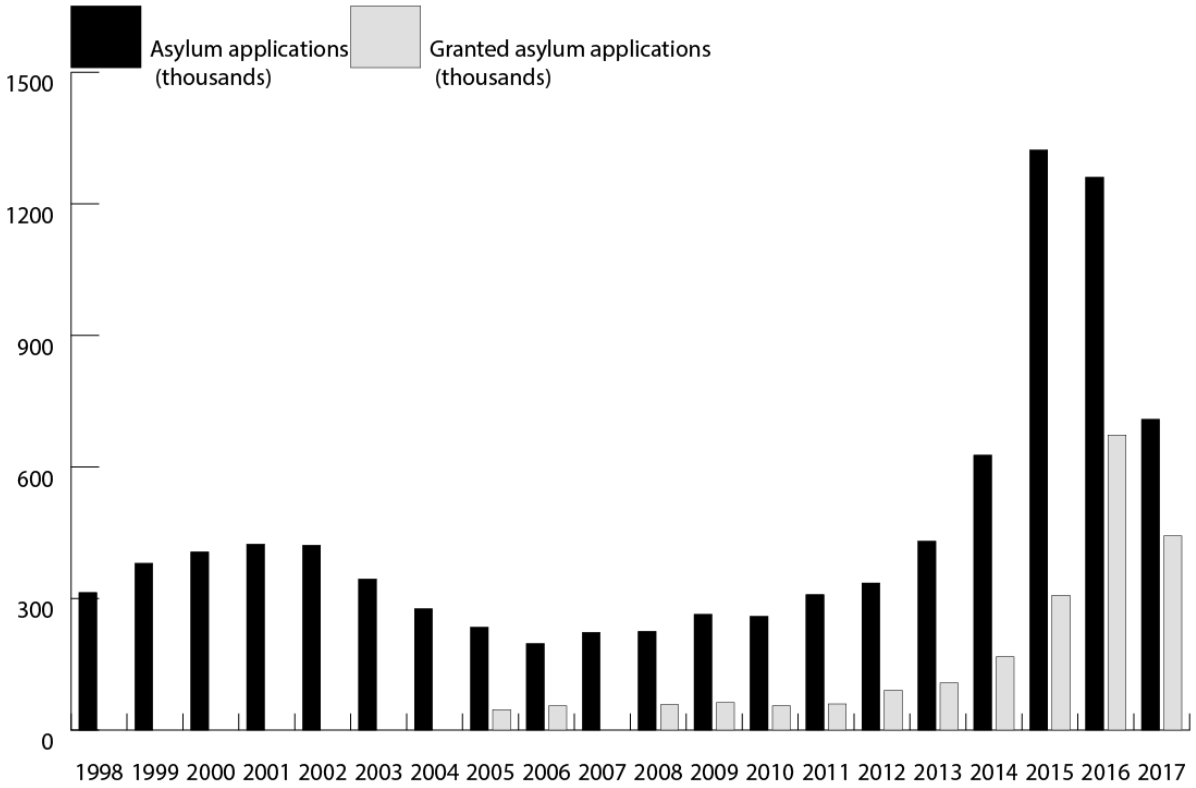



Figure 2.3: Total asylum applications in the European Union and the number of residence permits granted. Data on granted residence permits was unavailable for 1998–2006, and 2007. Data from (Eurostat, nd).

These significant migrant waves were accompanied by a variety of problems. European border countries became overwhelmed, and were unable to cope with the large number of migrants and refugees. Europe as a whole was unprepared, and was consequently forced to operate and respond in a reactive manner. This resulted in severe political chaos, polarization in society, and—most importantly—numerous migrant deaths. The European migrant crisis has put migration high on the agenda amongst politicians, policy-makers, the media, and scientists—even though the current migrant numbers are back to their pre-crisis levels (The New York Times, 2018).

Several important aspects can be identified in this short historical overview. The essence of these aspects is represented in the following considerations:

**Consideration 2.1.1** *Migration is a natural phenomenon that is inseparably intertwined with human history. Therefore, it might also be an important aspect regarding the future development of human society.*

**Consideration 2.1.2** *Remains of ethnocentrism ideologies may be the underlying cause for current polarization anomalies in societies.*

## 2.2. Concept Demarcation

The previous section provided a brief overview of global migration throughout history, and it became evident that there are different types of migration. In this section, the concept migration and its elements are briefly discussed to get a clear demarcation for what is meant by migration in the remainder of this study. To this extent, a comparison is made between the definitions of migration by several well-known institutions:

- *If people migrate, they travel in large numbers to a new place to live temporarily.* (Cambridge Dictionary Online, nd)
- *(Of a person) move to a new area or country in order to find work or better living conditions.* (Oxford Dictionaries Online, nd)
- *To move from one country, place, or locality to another.* (Merriam-Webster Online, nd)

Looking at these definitions, it appears that there is no clear consensus on what precisely classifies as migration. According to the Cambridge definition, it involves large numbers of people, and they only move somewhere for a limited period of time. Every year in July and August, large numbers of North- and West-Europeans travel to Southern Europe to enjoy the warmer climate. After a week or two, whether or not with a deep tan, they travel back home and return to their regular daily business for another year. According to the Cambridge dictionary this would classify as migration, but obviously this phenomenon is not what caused the European migration crisis in 2015.

The Oxford Dictionaries require migrants to move to a new area or country, but only because they want to find work or better living conditions. However, this would exclude voluntary migration to, for example, explore foreign cultures (Madison, 2006). Another doubtful case following this definition is people migrating to less-developed parts of the world to help the local population by means of volunteer work. Their intrinsic motivation for migration is not that they want to find work or better living conditions, since they would probably find better living conditions or work more easily in their country of origin.

Merriam-Webster does not include any minimum group size or time restrictions, and it does not specify any underlying reason for migration. This definition is very broad, and is satisfied if one moves from one country, place, or locality to another. However, it is not clear where the boundary of such localities or places should be placed. Another province? Another city? Another neighborhood? Or would moving to the house across the street also be classified as migration? This definition is still unclear regarding this aspect.

These three different definitions do give a clear view on unclear and ambiguous aspects of migration. Basically, three main elements need to be further specified to get a clear demarcation of migration for this study:

1. Community-boundaries between which migration can take place
2. Underlying reasons for moving
3. Time

For element (1), the boundaries can be specified through already existing boundaries: continents, regions, countries, provinces, states, municipalities, cities and villages, or even neighborhoods. But another, less obvious choice, would be language regions. However, languages tend to transition slowly from one language into another over a certain distance. This study will address migration on a country level. That is, the spatial factor of internal migration within countries is not taken into account.

Element (2) is an important factor, because it allows the distinction between migrants and 'normal' travel. A person that works in a neighboring country can cross a border twice a day, but should not be considered a migrant. In addition, various other migratory-like manifestations need not be classified as migration in this study (e.g. tourism, family visits, study). Therefore, this study considers migration only for a selected group of root causes.

With respect to element (3), this study will not artificially impose any time limits. As long as (2) is satisfied, the amount of time that a migrant moves to an alternative location is not deemed relevant.

This leads to the following consideration:

**Consideration 2.2.1** *Migration is an ambiguous concept. That is, it is scale-, location-, and time-dependent, and can be cause-specific. Therefore, it can be studied from multiple perspectives (e.g. local, national, regional, global).*

## 2.3. Migration: A Complex Topic

In this section, the complexity of migration is explored on an abstract level, by considering some relatively simple thought experiments. In the present day, the only types of human migration that still occur are home-community mobility, and cross-community migration (Manning, 2012). Home-community mobility is comprised of movement within the community, and while this is certainly an interesting topic, it is not within the scope of this study. Cross-community migration is the most complex form of migration, and it entails the migration of humans across linguistic and cultural boundaries. The remainder of this section will address the phenomenon of cross-community migration.

The first questions that might be asked are 'why?' and 'how?'. Why do people decide to migrate? How do they decide where they want to go? Or—more generally—what is the decision– or thought process that any individual goes through before he or she decides to migrate? Given the fact that some areas generate significantly more migrants that other areas, this decision process is probably heavily dependent on the environment an individual is in. As a consequence, the circumstances in an area could be somehow related to the number of migrants that decide to leave or enter that area.

Suppose that this is indeed the case, and consider the (very) simple system description of the migration decision process for an individual in Figure 2.4. An individual receives (continuous) input from his or her environment, undergoes an internal decision process, and makes a decision whether or not to migrate. Generally, the individual will decide to migrate when a certain (unique) threshold is reached. The relevance of this simple system description lies in the complexity that it actually represents. As of 2017, the world population is estimated to have reached almost 7.6 billion people (United Nations, 2017a). As every person is unique, every one of these 7.6 billion people has his or her own internal decision process and external environment, and (in principle) they are free to move anywhere.

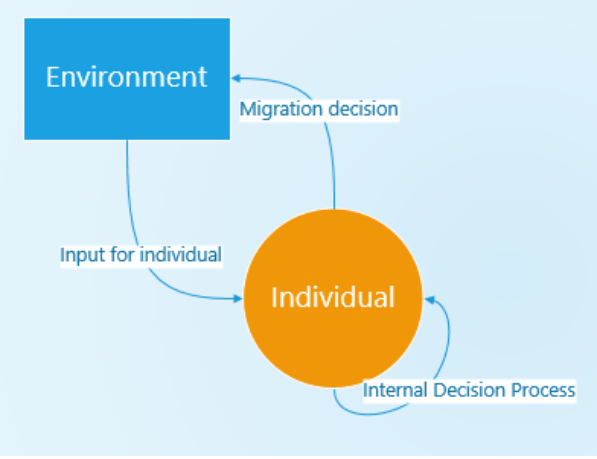



Figure 2.4: A simple overview of the migration decision process for a single individual.

Even if this decision process could be defined for every single individual, this would still not suffice to adequately capture—let alone predict—the complete phenomenon of migration. As time progresses, not only the environment in each region changes, but also the internal

decision processes of all individuals will most likely be subject to change. That is, the same person might make a different decision tomorrow than he or she makes today. In addition, over 300 thousand babies are born each day, while roughly 150 thousand people die.

Of course, this complexity can be reduced by means of generalizations and simplifications. But although this would reduce complexity to a certain extent, this would consequently also greatly reduce the extent to which the migration process is captured. Even when approaching this problem on a country level, differences between these countries would still have to be accounted for. The people of each country are different culturally, socially, and have different norms and values—all of which could affect their country-specific decision process.

Another complicating factor is that people leaving an area at the exact same time might have different reasons for migrating. These root causes for migration should somehow be distinguished from one another. Identifying the underlying root causes and their effect on migration dynamics could provide useful insights in the coherence of the migration system. Additionally, root causes may overlap; maybe it is the combination of various factors that has convinced a migrant to leave his home.

What should also be realized, is that problematic circumstances rising in certain areas as a result of migration, can have its origin in areas thousands of kilometers away. The solution to problems in an area, should therefore not necessarily be sought within its own proximity. Moreover, the solution to problems in one area might actually be the cause of problems in another. For example, building a wall to prevent immigrants from entering a country would immediately negatively affect its neighboring countries.

This hypothetical wall also shows that addressing migration problems is a matter of perspective. If no migrants would be able to enter Europe anymore due to a giant wall, this could potentially solve the problem from a European point of view. However, from a migrant's perspective this only leads to additional problems, and one would tend to a completely different solution when addressing this problem from their point of view.

This complexity should be taken into account when dealing with migration:

**Consideration 2.3.1** *The migration process is extremely complex, and any attempt to capture it requires severe generalizations and simplifications.*

## 2.4. The Ethical Aspect

The previous section illustrated the complexity of migration from a quantitative perspective. Arguably even more complex is the ethical aspect of migration, while migration policies involve highly contested normative judgments in all phases (Bader, 2005). Since the focus of this study is to analyze how global, regional, and national policies affect migration, it is important to identify the ethical issues that accompany them. Therefore, this section will briefly discuss some considerations that should be taken into account regarding the ethics of migration.

While policy-makers tend to focus on how migration can be controlled, it is also disputable whether migration should be controlled—and, if it should, by whom. Do national and/or international governments have the right to interfere with the migratory preferences of individuals? Can this interference be ethically justified? In the current global situation, an individual's place of birth greatly affects his or her migratory opportunities and limitations. From an egalitarianism point of view, governments should not make distinctions based on nationality or underlying reasons for migration—but it would be quite complex to design migration policies that satisfy this criterion.

Another important aspect is related to the responsibility of legal authorities towards migrants. Do national governments have an obligation to take care of migrants—including those in transit—in their country? And what is their responsibility towards individuals that migrated to another country, but retain their nationality? The spatial aspect of migration makes these questions exceedingly complex, and it not only involves authority-to-migrant interactions, but also cooperation between various authorities (i.e. between governments of countries, regional institutions, and global initiatives). It is often not clear where the responsibility of one legal authority starts or ends. For instance, the European Union is currently

planning to build migrant processing centers in Africa (The Guardian, 2018), which is well outside of their geographical boundaries.

Whether a migrant is granted asylum often depends on his or her underlying reason for migrating. Migrants that flee from e.g. war, violence, or persecution, are classified as refugees and receive a special status in most receiving countries. However, specifying the criteria regarding for who is and who is not classified as a refugee is an ambiguous and perspective-dependent affair. That is, these criteria would be heavily dependent on the interests of the one making the decision. For instance, a policy-maker or politician might tend more towards pleasing the citizens of the nation he or she represents, than ensuring the well-being of migrants from abroad. In contrast, a civil servant who meets migrants personally on a daily basis may have a significantly different perspective. For example, a difference in perspectives led to a heated debate in the Netherlands in 2011, in the case of Mauro Manuel (The Guardian, 2011).

Since each migrant case is unique, generalizing policies may lead to distressing individual cases. As a consequence, the discussions and debates about such cases are—most often—emotionally intense. It is hard to quantify this emotional aspect, which should be taken into account when analyzing any model related to migration.

The issues mentioned above illustrate the ethically complex character of the migration phenomenon and its problems, but it is still only a small selection of the ethical aspects of migration. Summarizing, this leads to the following consideration:

**Consideration 2.4.1** *Addressing migration-related problems is ethically complex and perspective-dependent.*

# 3

# A Systems Perspective

In Chapter 2 it was discussed that migration is an almost infinitely complex topic. In order to pinpoint the aspects that can be quantified and modeled, it is helpful to first have a systematic description of processes related to migration. The definition of a system, and the corresponding scientific approach is briefly summarized in Section 3.1. After that, Section 3.2 will describe migration as a system. This system description served as a starting point for the development of the model that is presented in Chapter 4.

## 3.1. What is a System?

Systems theory is an approach to comprehensively describe natural or artificial phenomena. A system is a set of connected elements or parts that function together within certain spatial limits or time constraints. Every system is surrounded by its environment (which could also be interpreted as a set of systems), with which it can interact (Von Bertalanffy, 1973). Due to the interrelations between a system's components, a change in one element can affect all other elements in the system as well.

Most—if not all—modeling methods utilize a systems thinking approach to formulate knowledge in a structured way. It is basically a systematic technique to organize and represent ideas regarding the processes and mechanisms of a system. This allows for quantitative analysis, future exploration, assessment of element interdependence, and opens the door for many more advanced research techniques.

In this case, describing migration as a system is a prerequisite to develop a migration model. Essential migration elements and processes will need to be identified, defined, and described in detail. Therefore, the next section will provide an overview of relevant aspects of migration for this study. It will do so by assessing a select number of historical attempts to model migration, and identifying useful aspects of these previous studies.

## 3.2. Migration Mechanisms

A structured overview of the migration system and its elements is essential before actually quantifying and modeling the relevant processes. To that extent, this section will provide an overview of the selected academic perspectives on migration processes that will be used for this study. To gain insight in the process of migration from a systems perspective, two important aspects should be taken into account: root causes for migration and migration dynamics.

Root causes of migration are the principles that drive people to migrate from one region to another. They can either provide people with a reason to leave a certain area, or provide people with a reason to move towards a certain area. Therefore, these root causes can be subdivided in push factors and pull factors. Push factors are defined as circumstances in regions or countries that repel individuals from staying at that place (Zimmermann, 1996). Examples of push factors are lack of services, lack of safety, high crime rates, crop failure, droughts, flooding, poverty, political instability, and war. On the other hand, regions or

countries can attract individuals to move there by pull factors (Zimmermann, 1996). Pull factors are—for instance—higher employment, more wealth, better services, good climate, low crime rates, political stability, more fertile land, and lower risk from natural hazards. It is generally the difference in factors between places that drives individuals to migrate from one region to another.

It should be taken into account that these root causes should not be considered individually for a certain area. The meaning of these root causes lies in the spatial and temporal distribution of these factors, where the relative difference between areas is the major driver for migration processes. Also, the distance and mutual 'ease of accessibility' should not be underestimated. A more difficult travel route might actually have a significant negative effect on the magnitude of a certain migration flow. An alternative, more comprehensive approach to root causes would be to distribute them into predisposing, proximate, precipitating, and mediating drivers (Van Hear et al., 2018). Predisposing drivers are economic, political, and environmental disparities between areas. Proximate drivers in this framework represent the previously mentioned spatial factors. Precipitating drivers correspond to push factors, as was discussed in the push-pull approach, but also include the actual departure thresholds for individuals. Mediating drivers are factors that constrain or facilitate migration, such as policies, transportation quality, or information availability.

In terms of root causes, this study will take the following attributes into account: spatial distribution of environment characteristics and their dynamics, departure thresholds, limiting or facilitating geographical factors, and drivers that facilitate or constrain migration. The translation of all these elements into a migration model is challenging, but should lead to better insight regarding the extent to which each of these factors influences migration.

The second aspect, migration dynamics, corresponds more to the manner in which (groups of) individuals respond to these changing root causes. When do they decide to migrate? How do they decide where they will migrate to? How will they travel there? How long do they stay in in-between countries? Will they reach their destination? All these elements could be gathered under the previously introduced term migration dynamics. Therefore, in the remainder of this study, migration dynamics refers to the spatial behavior and distribution of migrants in response to external factors, while root causes refer to the dynamics of these external factors.

Throughout academic history, migration modeling has developed significantly, and multiple approaches have been suggested. The first substantial attempt at defining the migration process dates back to two articles: Ravenstein (1885) and Ravenstein (1889), which was the outset to define the following Migration Laws:

1. Every migration flow generates a return or counter-migration.
2. The majority of migrants move a short distance.
3. Migrants who move longer distances tend to choose big-city destinations.
4. Urban residents are often less migratory than inhabitants of rural areas.
5. Families are less likely to make international moves than young adults.
6. Most migrants are adults.
7. Large towns grow by migration rather than natural population growth.

The first four laws are particularly interesting, as they describe migration as a process with gravity-like properties. More than fifty years later, in 1958, John Q. Stewart proposed the idea of applying a physics law to population movement, which resulted in the gravity law of population migration as it is commonly applied:

$$M_i^j = G\frac{P_i^\alpha P_i^\beta}{D_{ij}^\gamma}$$

Although this formula is relatively simple, the gravity model has proven to fit migration data exceptionally well, and has been used successfully in forecasting human displacement on

a sub-national level (Poot et al., 2016). A more recent application is the adaptation of the gravity model towards a radiation model (Simini et al., 2012), which significantly improved the accuracy of the human displacement estimations.

At first glance this seems promising, but the gravity approach to migration still faces significant challenges in some fundamental aspects. For instance, on a micro-level, the migration flow from one country to another is partly dependent on the origin country j. This origin country could be defined in multiple ways: (a) the country of birth, (b) the country of citizenship, or (c) the country of last residence of the migrant (Beine et al., 2016). In the gravity model, the difference between these criteria proves hard to implement—partly due to the level of complexity this would add, and partly due to a lack of data. Another difficulty that is argued by Beine et al. (2016), is found in the multilateral resistance to migration. The attractiveness of alternative destinations exercises influence on the determinants of bilateral migration rates, but the attractiveness of each country is likely to be correlated with the attractiveness of its peer countries—which should be accounted for. An example for how to control multilateral resistance to migration in the gravitation model can be found through the elaboration of the CCE estimator in Bertoli and Moraga (2013).

The exact internal feedback mechanisms and their corresponding migration dynamics is a much debated topic in migration literature. Positive feedback effects may have a reinforcing effect on migration, which would manifest itself in the formation of migration networks. However, these theories are often neglecting indirect feedback effects (e.g. remittances), are unclear about the exact causes of network migration (e.g. the underlying reasons for migration), and do not account for counteracting migration mechanisms (e.g. dynamic policies and polarization of societies) (De Haas, 2010). As such, these shortcomings should be taken into account in this study, and addressed or clarified where necessary.

Several of these attributes related to migration dynamics were already covered in the selection of root causes, but the gravity model is a first step towards connecting and quantifying the actual processes. That is useful, because a migration model requires some sort of distribution process that quantifies when, how, and where migrants move. To that extent, the following attributes will be taken into account with regard to migration dynamics: the spatial distribution of migrants over time, the feedback effects of migration on its own processes, network migration, and indirect feedback effects of migration on the environment.

# 4

# The Migration Dynamics Model

The previous chapter discussed the mechanisms and drivers of migration that are relevant for this study. This resulted in a clear demarcation of concepts that the model in this study should encompass. In this chapter, the constructed model for global migration is described in detail. As was already mentioned before, the spatial character of this complex problem requires integration of data science with modeling techniques. Part of this chapter is therefore dedicated to the data acquisition process that was developed for this study.

## 4.1. Data-Rich Modeling Approach

Recent digital developments and computational progress—particularly in terms of data availability and the origination of the big data concept—have initiated an interesting split in modeling approaches. Whereas model-driven approaches rely heavily on the experience and knowledge of the modeler or experts, data-driven models are more reliant on the actual insights that can be obtained from big data sets. However, data does not answer questions or solve problems. The size of a dataset is not necessarily representative for its potential to address issues. As a result, data is only useful when it is used appropriately to address a related question. Still, data can definitely provide useful insights if it is embedded in a good theoretical framework.

Therefore, this study enhances a data- and theory-rich modeling approach. Data is used to initialize the model, and to gain insight and understanding of the relation between variables and parameter of interest. But how the data is used in the model—the actual mechanics of the system and its interrelations—is based on scientific theories and assumptions.

## 4.2. Underlying Principles

This section provides a brief description of the scope, structure and underlying ideas of the proposed data-rich migration model component. To that extent, several principles are identified which served as a starting point. These principles are mainly related to the model component for migration dynamics, but other elements of the model will also need to adhere to these principles.

### 4.2.1. Principle of Freedom

The underlying idea of the model is that, like in the “real world”, migrants are (relatively) free to travel between countries. That is, there are no pre-made routes forcing migrants to travel in a certain direction. However, topographical hurdles, (partly) closed or difficult border crossings, and other spatial obstacles will be represented in the model. These could be implemented by means of a distribution matrix, where these elements are factored in. A contiguity matrix is used in the model to define possible border crossings and the type of border crossing (i.e. land or water contiguity). These implementations should not interfere with the open nature of the model. That is, an open model structure should be preserved at

all times.

### 4.2.2. Origin Tracking

The country of origin of migrants and refugees is tracked in the model. This means that, no matter how far or long migrants travel, they retain their nationality. This implementation leads to some interesting possibilities for the model:

- The attractiveness (a term used to indicate the tendency of migrants to relocate to a certain region, area, or country) can be unique for each origin-destination pair of countries. As a consequence, diverging migratory preferences of migrants with different nationalities can be taken into account. These "preferences" are determined by various factors that can be both voluntary (e.g. level of similarity between country of origin and destination country, welfare, or number of migrants of the same nationality already in a certain country) and involuntary (e.g. likelihood to get a residence permit, forced redistribution of migrants in a certain region, or conflict). The underlying reason for migrating (e.g. conflict, economic circumstances, or food/water scarcity) could also influence migratory choices of migrants. Assuming that most migrants from a certain country migrate for the same reason, this can also be taken into account.

- It allows for a more detailed mapping of spatial migrant flows than in traditional SD migration models, where nationality is mostly not taken into account on this scale. By tracking the nationality of migrants in all countries at all times, the migration routes per nationality can be acquired and analyzed.

- In reception countries, a distinction can be made more easily between distressing cases (e.g. conflict/political refugees) and less distressing cases (e.g. economic migrants).

### 4.2.3. Behavioral Principles for Migrant Distribution

Given the large number of countries that are taken into account, as well as the dynamic character of migration and the situation in each country, it is not feasible to manually create and/or check the distribution matrix. Therefore, a set of behavioral principles (rules) should be devised that automatically translates the available data (at any point in time) into a dynamically generated distribution matrix. One example of such a principle/rule would be that it is relatively easier for a citizen of the European Union to travel across the EU than for a non-EU citizen. Another example of such a principle is a language barrier, where people would be more attracted to countries that have a similar main language as their native language. A third example is mode of transport, where migrants from wealthy countries can generally afford the fastest means of transport, but migrants from less wealthy countries are forced into more cumbersome modes of transportation.

A good indication for attractiveness of each country for migrants should—at minimum—include the following two important aspects. Firstly, it requires the specification of the mutual differences between country of origin of migrants, and their destination countries. Indicative elements that could be compared to this extent are e.g. wealth, educational or professional opportunities, life expectancy, or language similarities. Secondly, migrants (i.e. from countries with low income) do not necessarily travel to a country with the intent of settling in that particular country. It could be that they only travel there because they were forced to leave their country of origin to the most accessible neighboring country. Another option is that migrants are traveling through countries with the intent of reaching other, more appealing countries. In this respect, centrally placed countries with an advanced transportation infrastructure, might be more attractive for migrants than more isolated countries or countries with a bad transportation infrastructure.

### 4.2.4. Multiple Migration Theories

Various theories exist regarding migration dynamics (e.g. the gravity model, the radiation model, neoclassical theory of migration, network theory). The distribution matrix that determines the migrant flows in the model could be adapted to fit several of these theories.

This would provide the opportunity to compare the difference in model behavior given certain migration theories. The substantial difference between most of these theories lies in the definition of the pull factors for migration (e.g. economics, labor markets, income differences, or race and ethnicity). These pull factors could be modeled separately in an SD model, and used to quantify the mechanisms of the migration model component.

### 4.2.5. Model Overview

The previous sub-sections introduced and discussed the main aspects and underlying ideas that formed the basis for the model development in this study. Summarizing, this leads to the following conditions and aspects that should be implemented and maintained in the model:

- Open model structure
- Spatial subdivision of environments
- Migrant flows
- Origin tracking
- Origin-specific migratory preferences
- Behavioral rules for migrant distribution
- Option for integration of multiple migration theories
- Multiple levels to define variables (multi-resolution model)
- Tolerable level of model manageability

## 4.3. Detailed Model Description

This section will provide a comprehensive description of the model component for migration dynamics. It will discuss the implementation of each of the underlying ideas that were introduced in the last section. Another important aspect that will be discussed here, is the trade-off between these implementations and their corresponding computational limits. The model was built in Vensim DSS, a software package for developing and analyzing System Dynamics models that allows for external data connections. A simplified version of the migration model component can be found in Appendix A.

### 4.3.1. Open Model Structure

The open model structure manifests itself mainly in the implementation that—in principle—it is possible for migrants to travel anywhere in the world. Of course, spatial constraints and certain time limits will be used to reinforce or diminish certain types of migratory behavior. However, the interconnectedness of the model still preserves the fact that, theoretically, migrants have the option to travel to any country in the world by any route they would prefer.

This open model structure is established by ensuring that all countries are interconnected in one giant network. That is, by mapping all countries and their contiguity to other countries, the resultant product enables all combinations of migrant origins, migrant destinations, and migrant travel routes.

### 4.3.2. Spatial Subdivision of Environments

The spatial subdivision of environments entails the representation of differentiation in characteristics between countries. Because it is not feasible to develop a specific model for each country, this was implemented by developing one generic model for a country's environment and migration processes. A technique called subscripting allows for the simultaneous simulation of such a generic model for multiple entities. Each entity—in this case country—can then be initialized with different values or processes. It should be noted that, while each entity represents a unique spatial area, they do not possess a property or element that actually defines their spatial location.

### 4.3.3. Migrant Flows

This lack of inherent spatial information is resolved by representing contiguity between countries in 2-dimensional matrix-form—and the size of this matrix is thus equal to *the number of countries x the number of countries*. This matrix provides information on whether a border exists between each unique pair of countries. The most elementary implementation would be a simple 0 and 1 distinction, where a 1 represents a border, and a 0 represents no border. However, in this study, the distinction is made more elaborately. Borders not only vary in size, difficulty of traverse, and visa requirements, but can also include the crossing of large bodies of water. In addition, the difficulty of border crossings can be nationality-dependent. An simple example is the use of planes: migrants from wealthier countries would not be forced to travel by land, but simply fly to whatever country they want to migrate to. Therefore, various indicators are utilized and combined to estimate the contiguity between countries in more detail. The model takes the following aspects into account: type of border crossing (land, water, or none), difficulty of traverse (number of highways between countries), and it distinguishes between modes of transport (based on the origin of countries). The last aspect is implemented by opening the borders directly to all countries for wealthier migrants.

The model uses the matrix to convert the outflow of each country to the respective inflows for its neighboring countries. Since this matrix is used to specify border crossings, it allows for the implementation of various policies related to e.g. (partial) border closings, accelerated migrant transport, and artificial migrant redistribution by authorities. It is also possible to distinguish between border crossings in different directions. That is, borders may be specified to be more or less difficult to cross in opposite directions. The actual mechanisms and their workings are discussed in more detail in Section 4.3.6.

### 4.3.4. Origin Tracking

Origin tracking is also implemented by means of a two-dimensional matrix. This matrix allows for the tracking of number of migrants in each country by nationality. The size of this matrix is thus equal to *the number of countries x the number of countries*. In addition, a distinction is made between migrants that are in transit (on their way to another country), and migrants that have reached their destination. The latter will no longer be able to migrate to neighboring countries.

### 4.3.5. Origin-Specific Migratory Preferences

The attractiveness of countries can be different for migrants from different countries of origin. This country-specific attractiveness can be dependent on, for instance, visa-requirements, country similarity, number of countrymen in destination countries, and distance from country of origin. The origin-tracking functionality also brings the additional benefit of being able to express these differences in the model.

In the model, this is currently only implemented by means of a two-dimensional language similarity matrix, but it is straightforward to implement additional indicators. The languages spoken in each country have been analyzed and transformed into a language similarity matrix. Another factor that might be origin-specific is distance between country of origin and reception country. However, since this is inherent to the model structure, this was not modeled explicitly.

### 4.3.6. Migrant Distribution

The model distributes migrants across countries by utilizing a generalized decision process. Even though this decision process leads to unique values for each country of origin, the process itself is not country-specific. This migrant (re-)distribution takes into account both pull- and push factors. Push factors affect the unique migration rate for each country. The level of severity of root causes determines the fraction of people migrating because of each root cause. On the other hand, pull factors determine the relative attractiveness of each country for each nationality. Most of these pull factors are in fact inversely proportional to the aforementioned root causes for migration. These factors are defined as *residence attractiveness*. However, some pull factors are related to how useful a country is for migrants to reach their country of destination. These factors are implemented as *transit attractiveness*,

and they are dependent on the centrality and connectedness of each country within the global country network. Network analysis was combined with several data sets (i.e. contiguity, number of roads between country, and total connectedness of each country), to assign a transit value to each country. The weights for transit- and residence attractiveness are part of the uncertainty space of the model.

The implementation of migrant distribution is quite complex, due to the tracking of the origin of migrants throughout the model. This means that a three-dimensional matrix is required for migrant distribution, which depends on the attractiveness of each country for each nationality, and the relative attractiveness of neighboring countries of the current country of residence. The actual inflow into a country of migrants from each nationality is the sum of all the outflows of those migrants out of its neighboring countries—to that specific country.

The inflow of migrants into a country leads to societal stress within that country, if a certain threshold is reached in terms of migrant numbers. This societal stress will then decrease the attractiveness of that country, leading to less migrants traveling to that country. However, the distribution is calculated on a relative scale, meaning that if the attractiveness of neighboring countries also decreases due to societal stress, the actual number of new migrants does not necessarily decrease.

### 4.3.7. Integration of Multiple Migration Theories

With the basic tracking and distribution mechanisms in place, the model allows for the implementation of multiple migration theories. This can be achieved by adapting either or both the distribution matrix and/or the country attractiveness matrix. Although the original idea included the actual implementation of multiple theories, this has not been achieved due to time constraints. Therefore, apart from noting that this implementation is definitely possible and promising, this study does not further assess this element.

### 4.3.8. Multi-Resolution Model: Subscript Mapping

Model variables can be defined on multiple scales through the use of subscript mapping. This technique allows for the definition of variables on a regional scale. Countries within that region—or any subset of countries—then inherit the value of that variable. This technique can also be used to implement policies for any set of countries. An example is the regional policy that will be implemented in Chapter 5, where subscript mapping allows the European Union to close its borders when a certain threshold of migrants is reached. This requires the mapping of the *region* European Union onto the *subrange* of European Union countries, and the mapping of the *reception region* of the European Union on the the *reception subrange* of European Union countries. A step-wise overview of subscript mapping can be found in Appendix B.

### 4.3.9. Model Manageability

As was already discussed thoroughly in the introduction, computational limits are an important aspect of this study and need to be taken into account. The model in its current form is the result of a thorough testing process, where multiple implementations were compared in terms of simulation time, versatility, and manageability. In addition, several techniques that greatly decrease simulation time have been employed. This section will present the main findings of this process.

The first dilemma in this process was the choice of modeling software. The two main options that were considered were a Python implementation and a Vensim implementation of the migration model component. Both have several advantages and disadvantages.

A Python implementation would provide significantly more freedom in terms of model development than a Vensim counterpart, but Python is a programming language that is not specifically created for modeling and simulation. In addition, coupling a Python model to data sources is straightforward, as multiple packages exist specifically for the purpose of data processing and analysis. The model could then also more easily be combined with various other modeling approaches, such as Agent-Based Modeling or Discrete Event Simulation, since a variety of connectors are available for this purpose. However, due the scale and size of the intended model, the viability of such a connection is doubtful.

A Vensim implementation on the other hand provides much less freedom, as such a model is bound to operate with the limits of the modeling program. But, since this software is specifically developed for this purpose, this would definitely be the preferred choice in terms of simulation time. An additional advantage of Vensim is that it also has a great variety of models and model components available, which can easily be coupled to the migration dynamics component. Input data for Vensim can be generated in Python (albeit in a very specific manner), and output data from Vensim can be exported and analyzed in Python as well.

Having said that, both implementations have been attempted in this study, and only the latter was successful. The Python model that was developed in this study generated the correct behavior, but its simulation time was simply too long. The developed model component in Vensim was more successful, but to that extent it required some specific equation definitions and simulation techniques. The remainder of this section will shortly discuss these in more detail.

By far the most important gain in simulation time was reached by using the compiled simulation option that Vensim offers. This is a technique that automatically translates a Vensim model to a C-program, which greatly increases simulation speed. Although this technique is generally only used for optimization purposes or sensitivity analyses, which includes numerous simulations, the migration model component already benefits significantly in a single run. It is safe to say that without compiled simulation the actual application of this model component would have been impossible.

Another aspect that greatly affects simulation time is the used integration method. Vensim offers several of these, and the method determines how the program numerically integrates the model's differential equations. The trade-off between accuracy and simulation time that this entails, is reinforced as models get bigger. In the case of the migration dynamics component, some integration methods (i.e. Runge Kutta 4 Auto) do not even allow to simulate the model, while others (i.e. Runge Kutta 4 fixed and Euler) simulate relatively fast.

The last aspect in terms of modeling techniques relates to the choice of functions that Vensim offers. Some variables in the migration model component can be specified with various Vensim functions, which result in the same outcomes. However, the number and nature of underlying calculations that need to be made are significantly different. This difference is caused by the fact that certain 'smart' functions, first determine which equations actually need to be solved. Since the migration model does not need to calculate migrant flows between countries that are not contiguous, these elements can be neglected. The most important gain in this respect was using the VECTOR SELECT() function, rather than the SUM() function.

An additional aspect that can significantly affect simulation time is number of countries taken into account. This means that even when the model component would have proven unable to take all countries into account, it could always be simulated for a smaller number of countries. Since the model component initializes from external databases, the number of unique entities taken into account does not affect the model mechanisms. This illustrates the generic character of the migration dynamics model component, meaning that it is both scale- and resolution-independent.

## 4.4. Data Acquisition

A number of the model elements that were discussed in the last section require a differentiation between the various countries in terms of their state. For such a significant number of countries this differentiation can only be made by utilizing various indicators in terms of the conditions in those countries. Therefore, a crucial part of this study is acquiring these indicators. The extraction and combination of various data sources is not a straightforward process. Data sources may be incomplete, provide conflicting data, or can be formatted in problematic ways. Several automated techniques were developed and compiled in Python scripts to smoothen this procedure (Appendix D). This section will provide a short overview of the developed data acquisition process and the data sources that were consulted.

### 4.4.1. Country Name Database

The developed model component requires data input to be very specific. All data should be in the exact same order and format. To this extent, an automated process is vital to obtain an adequately sized dataset for the purpose of this study. Since the model mainly requires data on a country level, data on that level should be easily translatable into a required format. One of the major problems in acquiring data from different sources is that country names may not be identical. For this reason, the first step in the data acquisition process is the development of a country name database, that enables automatic translation of alternative country names into the generically used names in this study.

This country name database was constructed in CountryNameDatabase.R (Appendix E), and consecutively expanded in 01 - Country Names Database.ipynb (Appendix D). This database was used throughout the remainder of the data acquisition process, and proved essential for the quick extraction of multiple data sources. Figure 4.1 presents a small part of this database.

| | ar5 | continent | country.name.de | country.name.de.regex | country.name.en | country.name.en.regex | cow.name | cowc | cown | ecb | ... | cldr.variant.yav |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASIA | Asia | Afghanistan | afghan | Afghanistan | afghan | Afghanistan | AFG | 700.0 | AF | ... | Afkanistáŋ |
| 2 | OECD1990 | Europe | Aland Islands | åland | Åland Islands | åland | NaN | NaN | NaN | NaN | ... | AX |
| 3 | EIT | Europe | Albanien | albanien | Albania | albania | Albania | ALB | 339.0 | AL | ... | Alpaní |
| 4 | MAF | Africa | Algerien | algerien | Algeria | algeria | Algeria | ALG | 615.0 | DZ | ... | Alselí |
| 5 | ASIA | Oceania | Amerikanisch-Samoa | ^(?=.*amerik).*samoa | American Samoa | ^(?=.*americ).*samoa | NaN | NaN | NaN | AS | ... | Sámua u Amelíka |

5 rows × 677 columns

Figure 4.1: The top part of the constructed Country Name Database (Appendix D and E).

### 4.4.2. Country Selection

Another important step in the data acquisition process is the selection of countries that will be taken into account. To get an idea about data availability for various countries, this selection process was initiated by comparing three major datasets that were assessed to have significant relevance in this study. To this extent, the members of the United Nations General Assembly were taken as a starting point.

The first source of which the included countries were assessed is the Correlates of War Direct Contiguity dataset (v3.2). The use of this dataset is highly desirable in this study, given its detailed character regarding the nature of country borders. It classifies country borders on a scale from 0 to 5, where a 0 means no border, a 1 means direct border via land, and 2 to 5 are used for water contiguity up to a certain distance (400 miles).

The second dataset that was taken into account for the selection of countries are the World Development Indicators (WDI) of the World Bank. This is a very extensive dataset (both in number of indicators and in number of countries for which these indicators are available), and therefore it lends itself to assess which countries are absent in the other two datasets.

Figures 4.2, 4.3, 4.4, and 4.5 show the discrepancy in the Correlates of War- and World Development Indicators datasets on comparison with the members of the United Nations General Assembly. Correlates of War lacks Iceland, New Zealand, and Serbia. The first two are absent because they simply have no contiguity to other countries within the specified water contiguity boundary of 400 miles. The case of Serbia is interesting, because the Correlates of War does include Yugoslavia. Since Serbia emanated from former Yugoslavia (and the specified contiguity for Yugoslavia matches that of Serbia), it is assumed that the Correlates of War entry of Yugoslavia is actually Serbia.

The World Development Indicators apparently distinguish between 24 more countries in comparison with the UN General Assembly, and no countries are missing from the World Bank data according to this comparison. The additional countries are mainly small islands, and states of which its existence is not recognized by all countries in the world. This poses

```
1 Yugoslavia
2 Kosovo
3 Taiwan
Number of countries in CoW, but not in UN: 3
```

Figure 4.2: Countries that are in the Correlates of War dataset, but not part of the United Nations General Assembly.

```
1 Iceland
2 New Zealand
3 Serbia
Number of countries in UN, but not in CoW: 3
```

Figure 4.3: Countries that are a member of the United Nations General Assembly, but are missing in the Correlates of War dataset.

a dilemma: should these countries be taken into account? Due to the *open model* approach in this study—and the potential future migration from contested or threatened areas—it is deemed essential that the list of countries taken into account is as complete as possible. The reason for this is twofold. Firstly, missing countries might compromise migratory behavior since there are less options available than in reality. Secondly, several of these regions, in particular the contested ones, are just those countries that can be expected to nurture significant root causes that lead to significant migration waves.

```
1 American Samoa
2 Aruba
3 Bermuda
4 British Virgin Islands
5 Cayman Islands
6 Channel Islands
7 Curaçao
8 Faroe Islands
9 French Polynesia
10 Gibraltar
11 Greenland
12 Guam
13 Hong Kong SAR China
14 Isle of Man
15 Kosovo
16 Macau SAR China
17 New Caledonia
18 Northern Mariana Islands
19 Puerto Rico
20 Sint Maarten
21 Saint Martin (French part)
22 Turks & Caicos Islands
23 U.S. Virgin Islands
24 Palestinian Territories
Number of countries in WB, but not in UN: 24
```

Figure 4.4: Countries that are in the World Bank dataset, but are not a member of the United Nations General Assembly.

```
Number of countries in UN, but not in WB: 0
```

Figure 4.5: No members of the United Nations are missing in the World Bank dataset.

All in all, this leads to the selection of countries represented in the World Development Indicators as a base list of countries that will be taken into account. Hence, missing data in other datasets will have to be dealt with accordingly. This data will be either filled by data cleaning algorithms, or it will be accounted for in the construction of the model.

### 4.4.3. Overview of Data Sources

The country name database and the list of selected countries provided the starting point for the remainder of the data gathering process. The databases that were used to initialize and calibrate the model, and assess the correlation between variables are listed in Table 4.1. A complete list of variables that were extracted from each data source can be found in the notebooks in Appendix D, which contains the Python scripts that were used to extract the data from the various sources. The World Bank data was extracted using the Python package wbdata (Sherouse, 2014).

| Dataset | Source | Purpose(s) |
|---|---|---|
| Direct Contiguity (V3.2) | (The Correlates of War Project, nd) | Contiguity data |
| Border Crossings | (Center for Geographic Analysis, nd) | Contiguity data, transit attractiveness |
| International Migrant Stock: The 2017 Revision | (United Nations, 2017b) | Initialization, correlation, calibration |
| World Development Indicators (WDI) | (The World Bank, ndb) | Initialization, correlation, residence attractiveness |
| World Factbook | (Central Intelligence Agency, nd) | Initialization, residence attractiveness, language data |
| Fragile State Index (FSI) | (The Fund for Peace, nd) | Root causes, correlation |
| World Risk Index (Natural Disasters) | (Bündnis Entwicklung Hilft, nd) | Root causes, correlation |
| Exploring Climate and Development | (The World Bank, nda) | Root causes, correlation |
| Freedom in the World | (Freedom House, nd) | Root causes, correlation |

Table 4.1: Overview of the data sources used for this study. The purpose(s) for which each dataset was used is given in the rightmost column.

## 4.5. Coupling to Root Causes Model

The root causes model component that is used in this study was made available by courtesy of dr. E. Pruyt. This model employs several of the datasets that were discussed in the previous section to generate a range of future scenarios for root causes for migration (Fragile State Index, World Risk Index). This model is connected to the migration dynamics component through the migration rate for each country.

## 4.6. Model Verification and Calibration

Verification of the model requires an assessment of the correct implementation of the conceptual model (Robinson, 1997). Two main verification tests have been conducted: a mass balance on the entire model, and a mass balance on the leaving and entering country flows.

The mass balance test for the entire model was executed by setting both birth rate and death rate to zero, meaning that total model population should be constant. This confirms correct implementation of the model equations, and that no migrants are 'lost' when traveling to another country. The result of the mass balance test is presented in Figure 4.6.

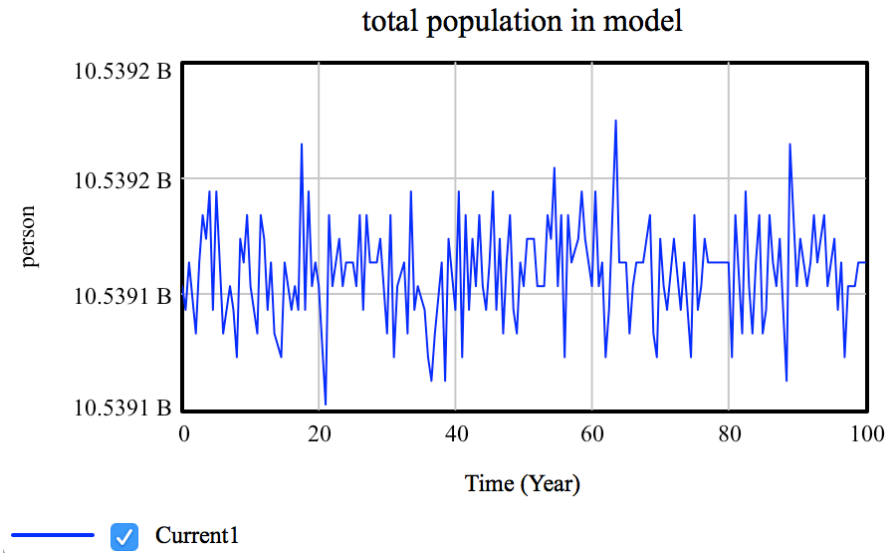



Figure 4.6: The mass balance test for the entire model. There are no births and deaths during simulation time, so the total population in the model does not change. The limited fluctuation is due to small integration errors.

The mass balance test for the leaving and entering country flows is depicted in Figure 4.7. The sum of the inflows is equal to the sum of the outflows for all countries of origin.

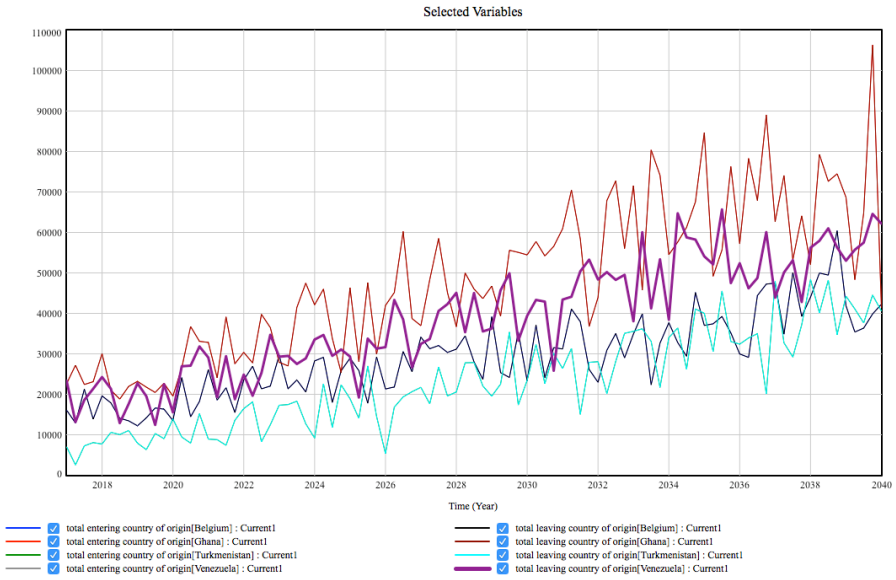



Figure 4.7: A mass balance test for the inflows and outflows of various countries of origin. The inflows and outflows match: the test is successful.

In addition, the constructed contiguity matrix was assessed visually in Gephi to identify any inconsistencies in terms of country borders. A visualization of the transit matrix can be found in Figure 4.8. No anomalies were found.

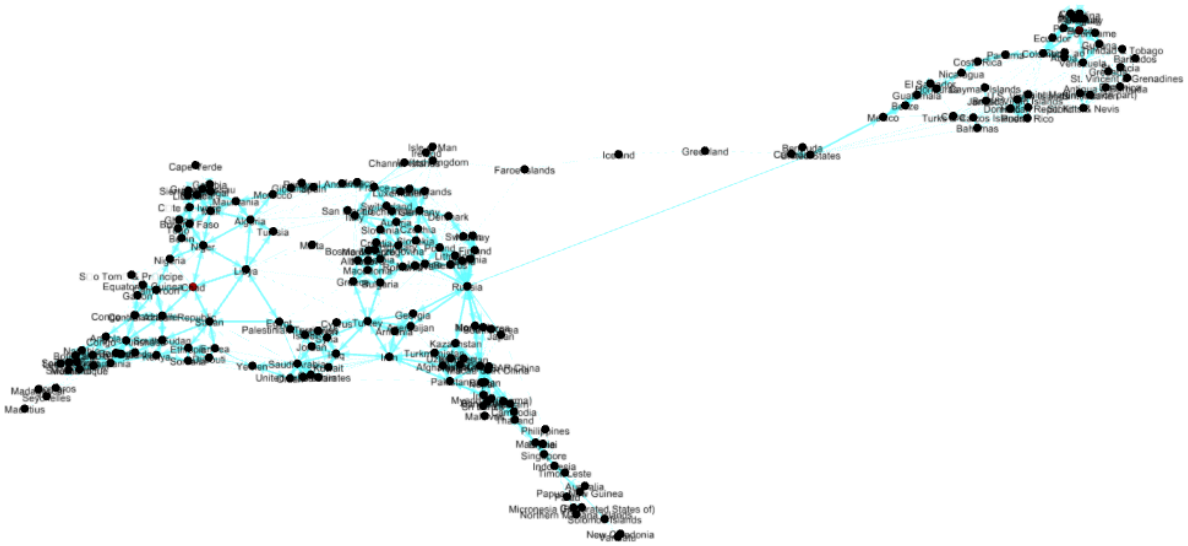
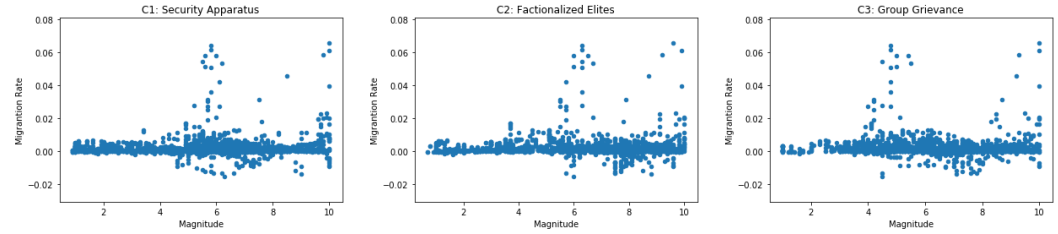
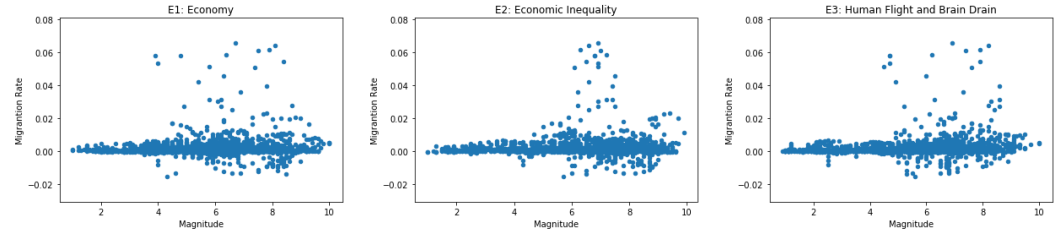


Figure 4.8: A visualization of the contiguity matrix in Gephi.

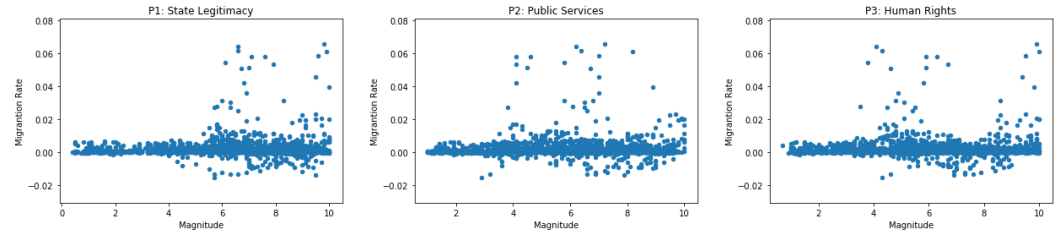
To provide accurate results, the model requires thorough calibration. Fine-tuning of the mechanisms and relations between root causes and migration dynamics by comparing the model behavior to historic data, is a vital prerequisite for realistic results. There are two main aspects for which the model should be calibrated: the relation between out-of-country migration and root causes, and the relation between attractiveness of countries and the number of migrants that travel there. However, data availability is a major issue in this regard. For instance, only the net migration rate is available for the majority of countries. For proper model calibration, the gross numbers are required. Nonetheless, the correlation between net migration rate and the Fragile State Indices is shown in Figure 4.9. From this correlation, the lower bound for out-of-country migration could be derived—but for determining the actual relationship either more information is required, or more advanced data analysis techniques need to be employed. The same holds true for the relation between attractiveness of countries and the number of migrants that travel there. Since this is outside of the scope of this study, this issue is not further addressed. Still, even if the magnitude of the model output is not correct, it does lend itself for a relative comparison of scenarios and policies—which can provide useful insights.



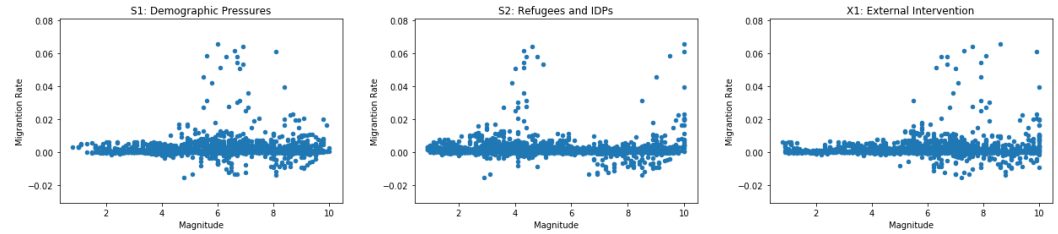

(a) C1: Security Apparatus (b) C2: Factionalized Elites (c) C3: Group Grievance




(d) E1: Economy (e) E2: Economic Inequality (f) E3: Human Flight and Brain Drain




(g) P1: State Legitimacy (h) P2: Public Services (i) P3: Human Rights




(j) S1: Demographic Pressures (k) S2: Refugees and IDPs (l) X1: External Intervention

Figure 4.9: Correlation between the Fragile State Indices and net migration rate on data from 2006 to 2016 for all countries.

# 5

# Results

This chapter will present the obtained model results. Section 5.1 will identify the variables of interest and briefly discuss the uncertainty space of the model. Section 5.2 will explore this uncertainty space and the corresponding outcomes to gain insight into the basic behavior of the model. In Sections 5.3, 5.4, and 5.5, the results are presented for various policy implementations on a global, regional, and national scale.

## 5.1. Uncertainty and Variables of Interest

This study uses Exploratory Modeling and Analysis to analyze the migration system. Exploratory Modeling and Analysis uses computational experiments to analyze complex systems while taking uncertainties into account (Bankes, 1993). To this extent, this study used the Python library EMA Workbench (Kwakkel, nd).

The uncertainties that were taken into account for this study fall in three major categories: uncertainties regarding the root causes, uncertainties regarding the extent to which people respond to these root causes in terms of migration, and uncertainties regarding the migratory preferences of migrants. A complete list of uncertainties—and policy settings—can be found in Appendix C.

The main variables of interest are the number and location of migrants across the world over time. This should allow for the identification of problematic areas, and the consequential potential problems in transit and/or destination countries. The results will be presented for sixteen spatially distributed countries, in order to assess the impact of policies on a global scale. These countries are: Afghanistan, Aruba, Australia, Colombia, Germany, South Korea, Morocco, Myanmar, Netherlands, Nigeria, Russia, Saudi Arabia, Spain, Turkey, United States, and Zimbabwe. A visual representation of the set of countries is given in Figure 5.1.

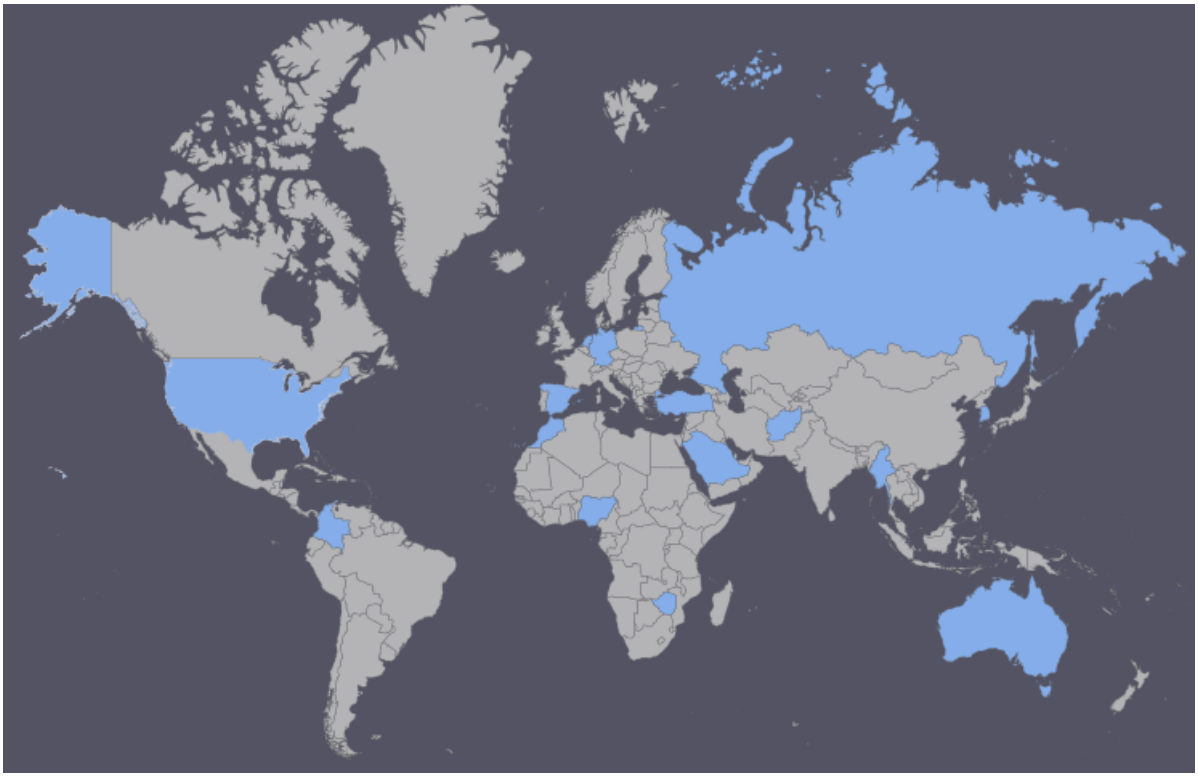


Figure 5.1: The set of countries for which the results will be presented.

## 5.2. Open Exploration

This section provides an overview of a selection of the results that were generated through open exploration of the defined uncertainty space. A complete overview of these uncertainties can be found in Appendix C. In Figure 5.2, a feature scoring for a select number of variables is presented. This gives an overview of the influence of the input variables on the outcomes of interest.

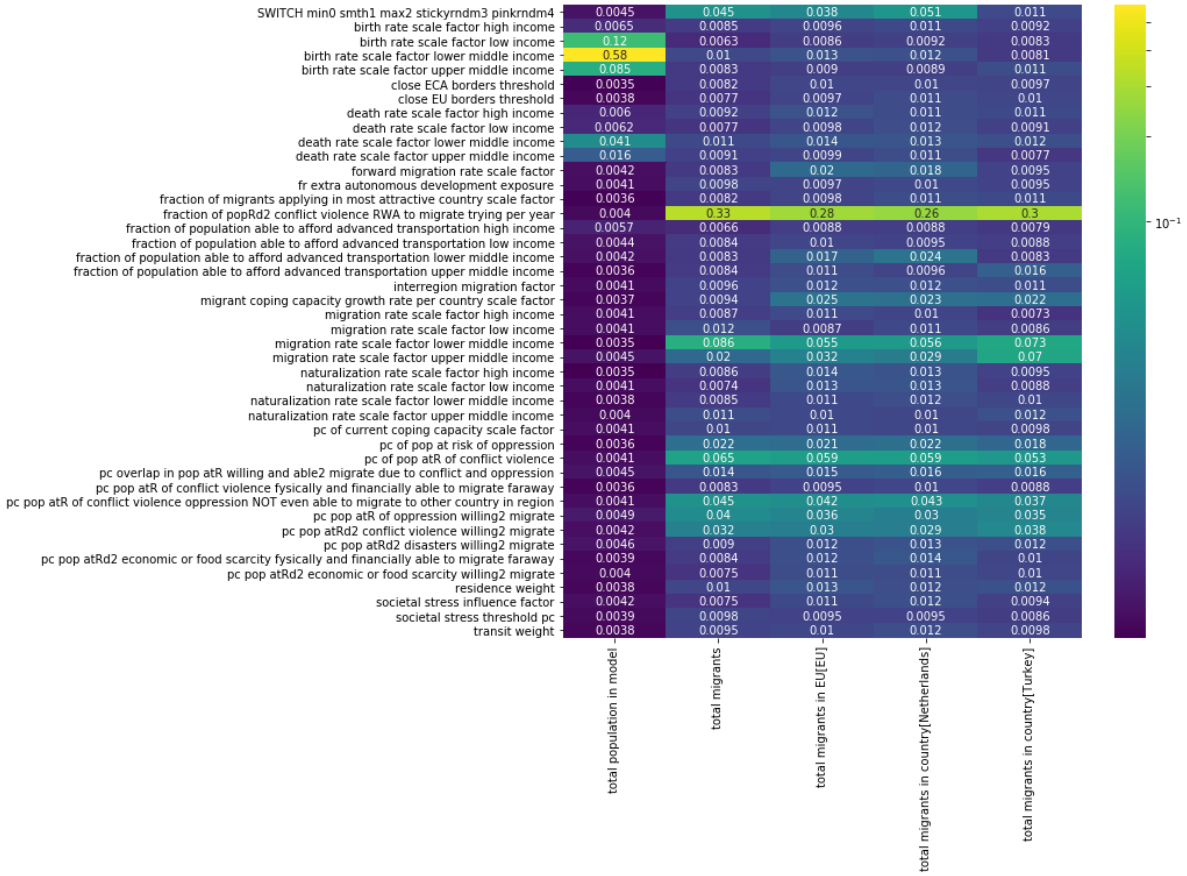
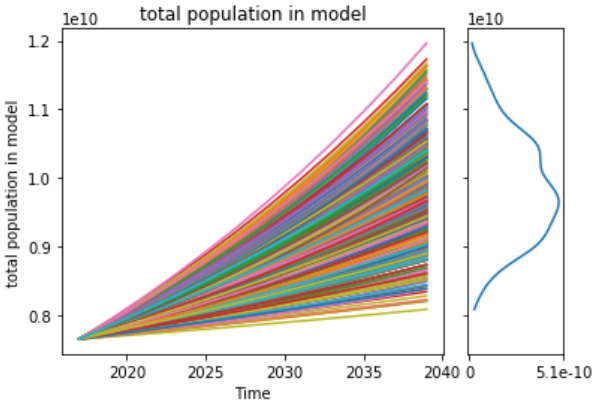
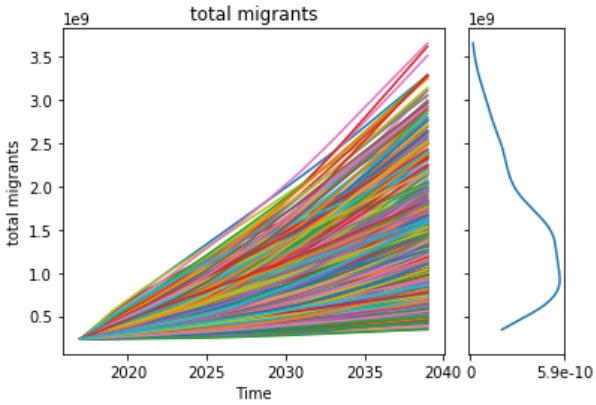


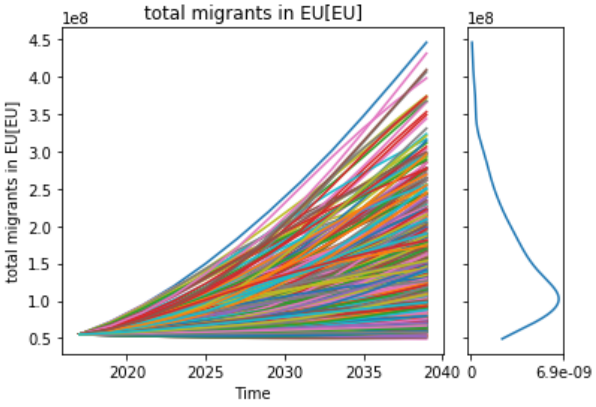

Figure 5.2: A feature scoring heat map based on the scenarios in the base ensemble. The total population is primarily influenced by the birth rates, while the number of migrants is highly dependent on the fraction of the population in each country trying to migrate due to conflict and violence. Note: the color scale is logarithmic.



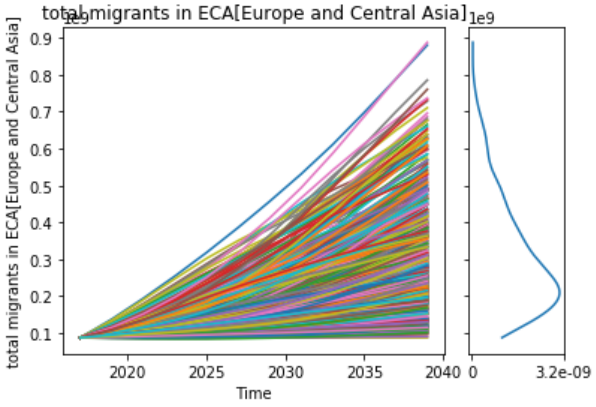

(a) Total population




(b) Total number of migrants

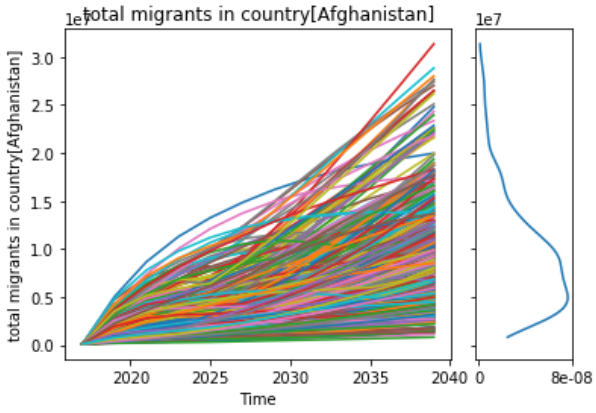



(c) Total number of migrants in the European Union

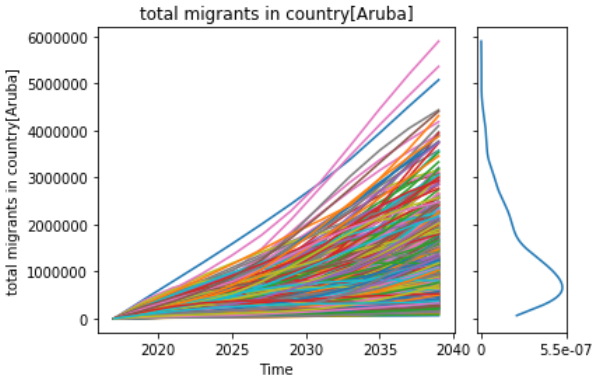



(d) Total number of migrants in Europe and Central Asia

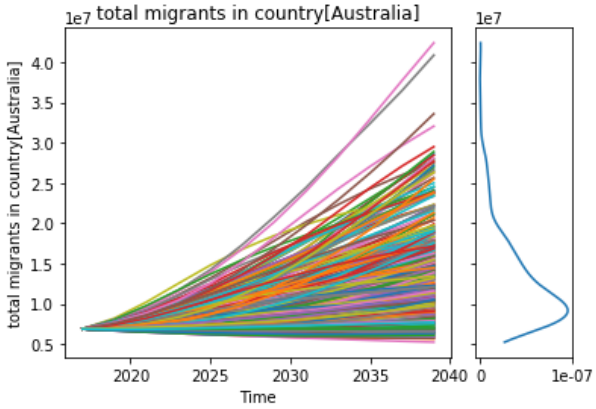
Figure 5.3: Base ensemble of results. The outcomes for total population, number of migrants, and number of migrants in the European Union and Europe and Central Asia for 500 distinct scenarios. There is significant variation in model outcomes. The number of total migrants rises significantly in most scenarios, as does total population.



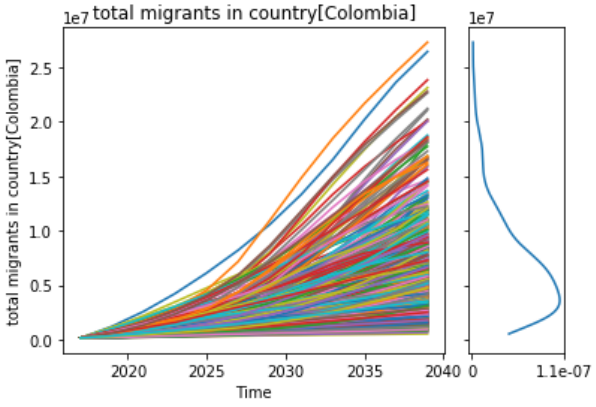

(a) Afghanistan



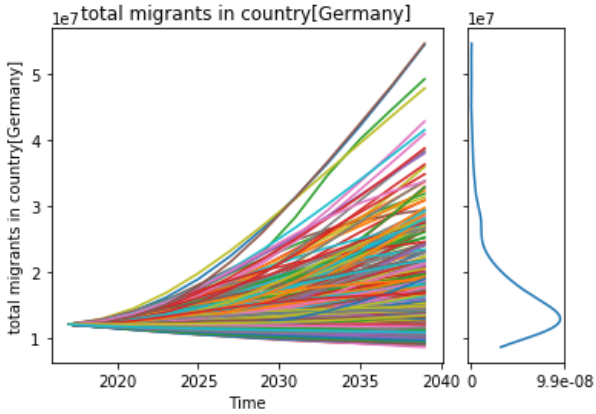

(b) Aruba



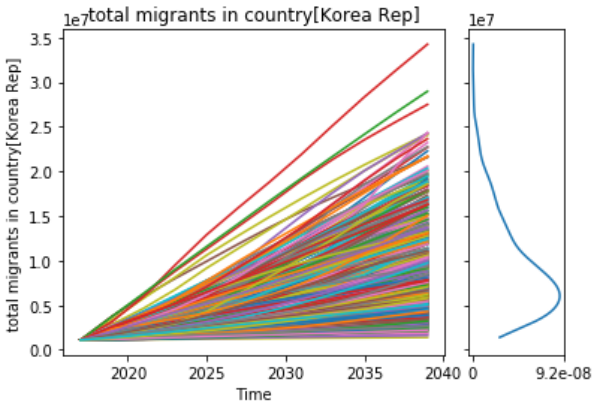

(c) Australia



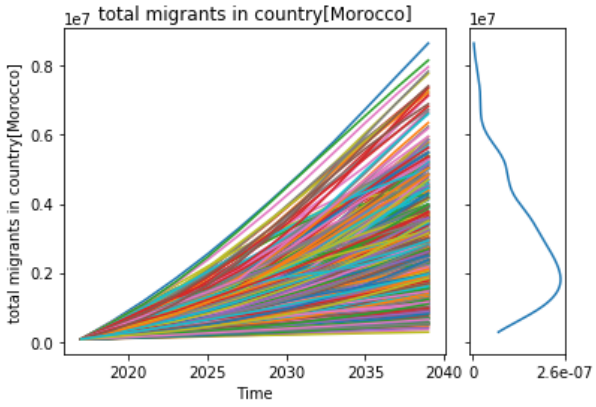

(d) Colombia



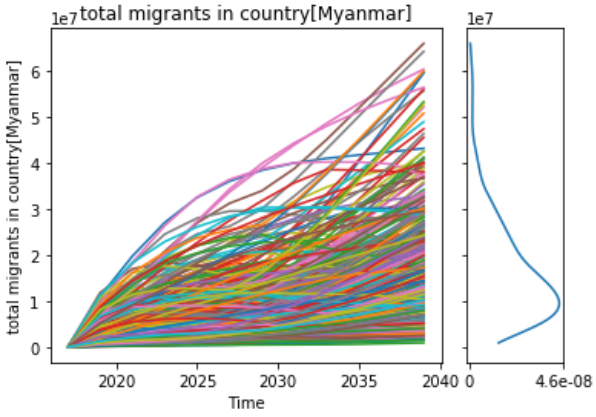

(e) Germany




(f) South Korea

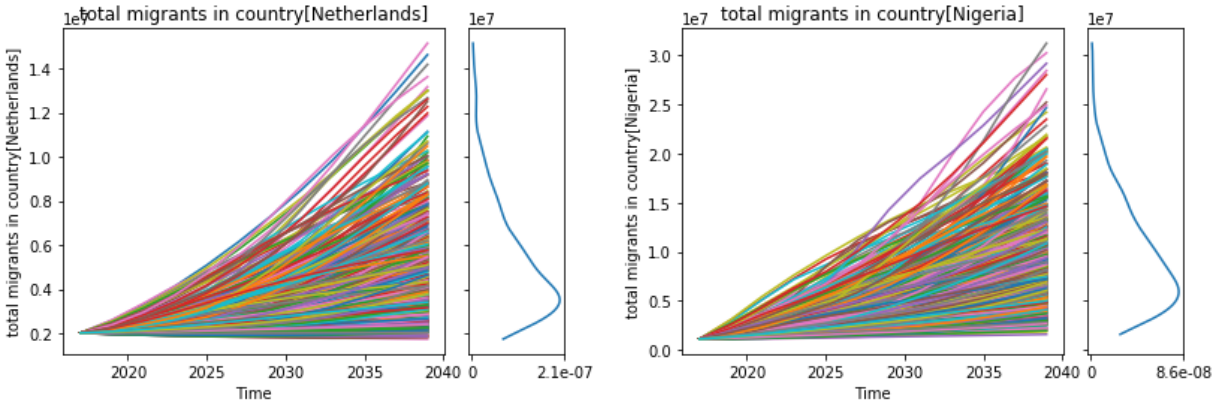



(g) Morocco



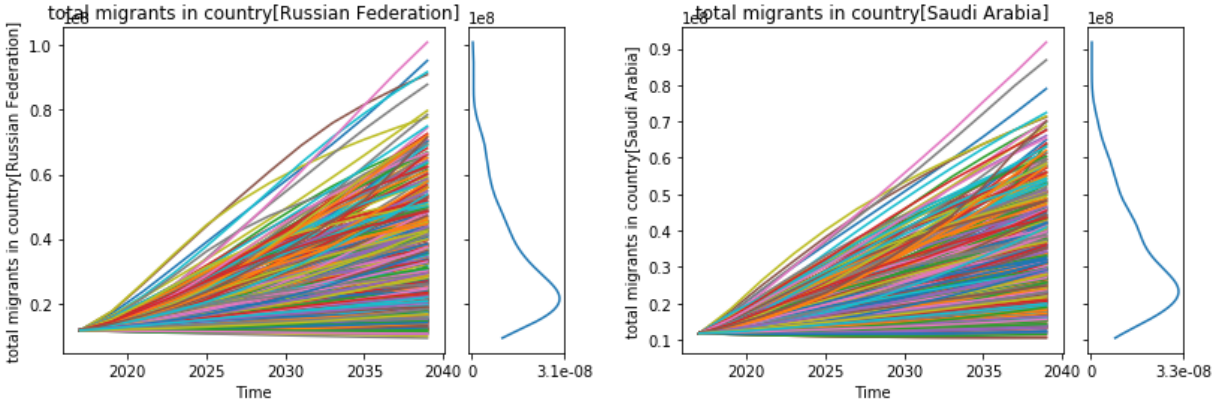

(h) Myanmar

Figure 5.4: Base ensemble of results. The outcomes for total number of migrants in various countries for 500 distinct scenarios. Most countries show similar behavior to the total number of migrants that was presented in Figure 5.3



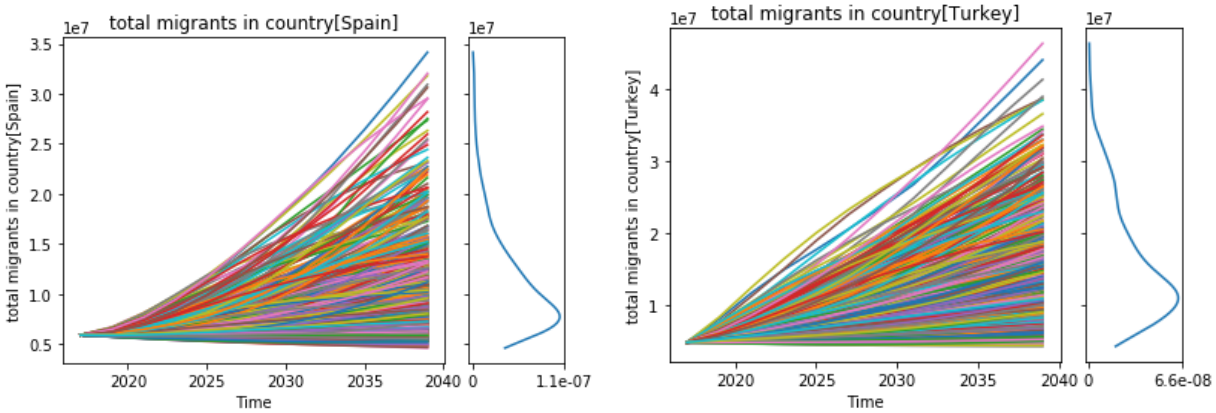

(a) Netherlands

(b) Nigeria



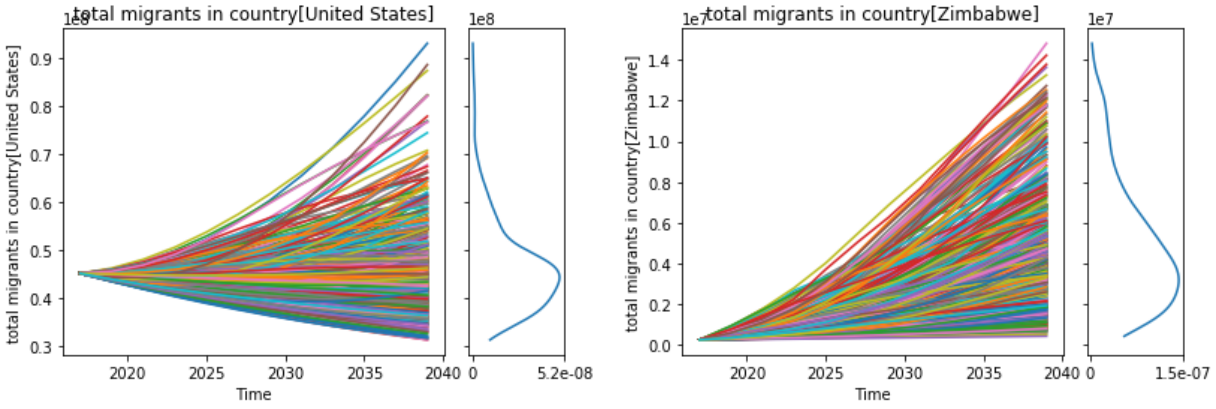

(c) Russia

(d) Saudi Arabia




(e) Spain

(f) Turkey




(g) United States

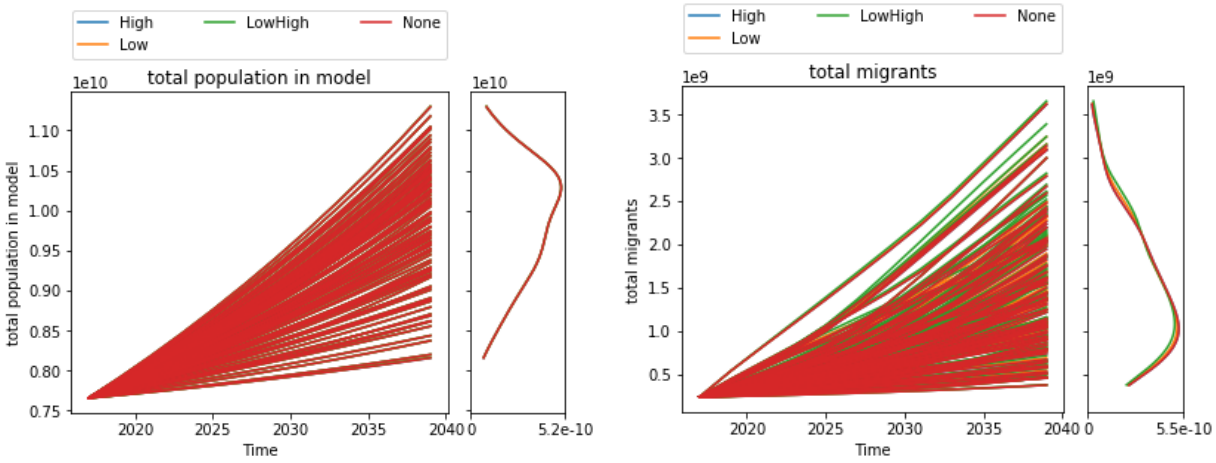
(h) Zimbabwe

Figure 5.5: Base ensemble of results. The outcomes for total number of migrants in various countries for 500 distinct scenarios. Most countries show similar behavior to the total number of migrants that was presented in Figure 5.3

## 5.3. Global Policy: Building Additional Shelters

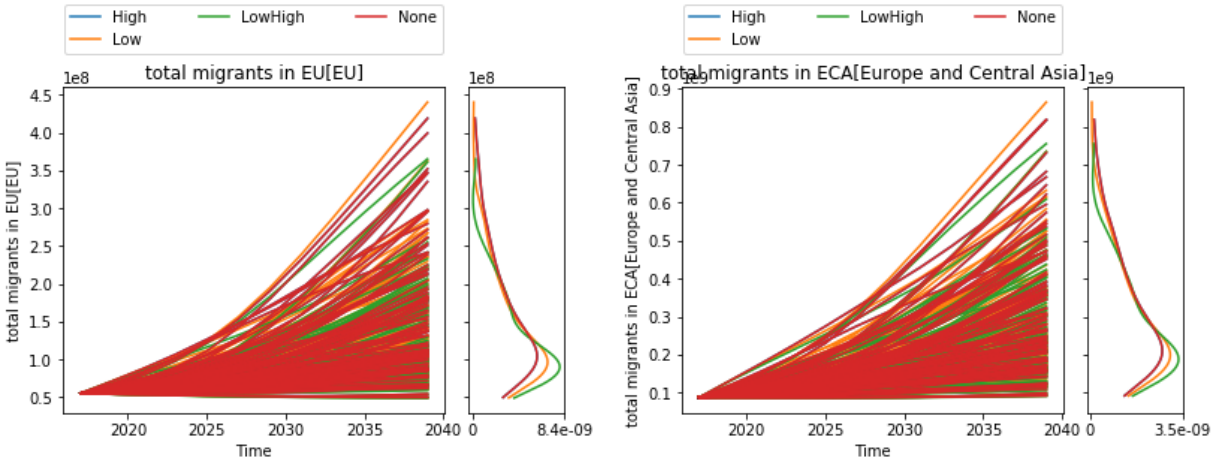
This section provides an overview of the results that were obtained for various global policies. These policies correspond to the location of shelters, either in low income countries, high income countries, or both—to see whether it significantly affects migration behavior globally, and in the previously selected countries. The following policies were implemented:

- **Policy 'None'**: No improvement in shelters and migrant coping capabilities.
- **Policy 'High'**: Shelters and migrant coping capabilities are improved in high- and middle-high income countries
- **Policy 'Low'**: Shelters and migrant coping capabilities are improved in low- and middle-low income countries.
- **Policy 'LowHigh'**: Shelters and migrant coping capabilities are improved in all countries.




(a) Total population

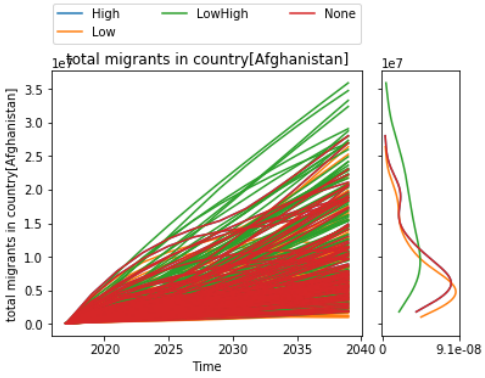
(b) Total number of migrants



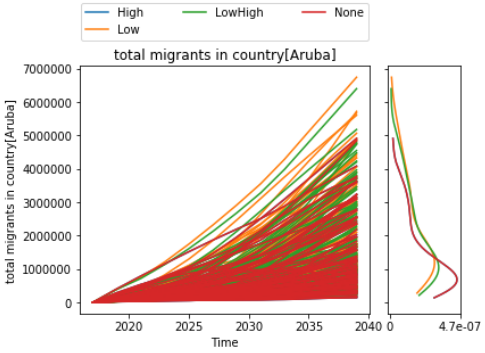

(c) Total number of migrants in the European Union

(d) Total number of migrants in Europe and Central Asia

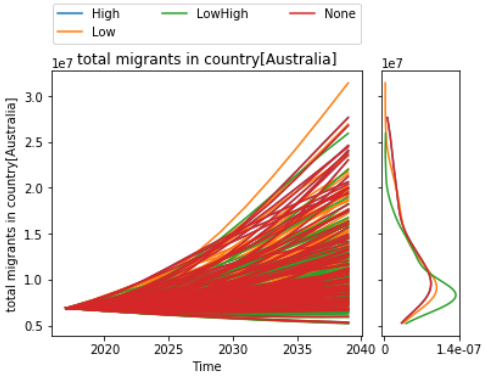
Figure 5.6: Results for global policy model runs. The outcomes for total population, number of migrants, and number of migrants in the European Union and Europe and Central Asia for 100 distinct scenarios. The number of total migrants in the European Union, and in Europe and Central Asia are slightly lowered by the 'Low' and the 'LowHigh' policies.



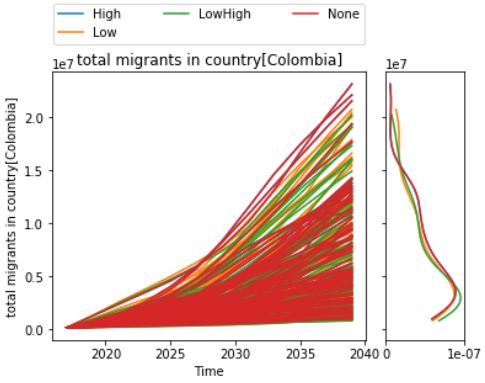

(a) Afghanistan



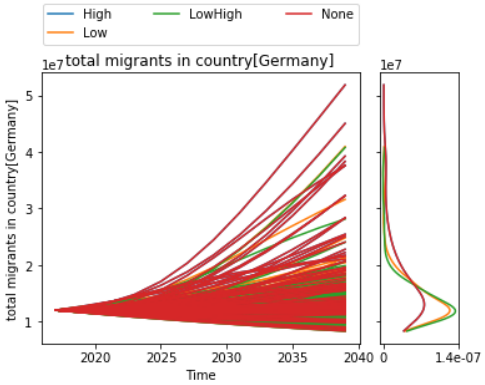

(b) Aruba



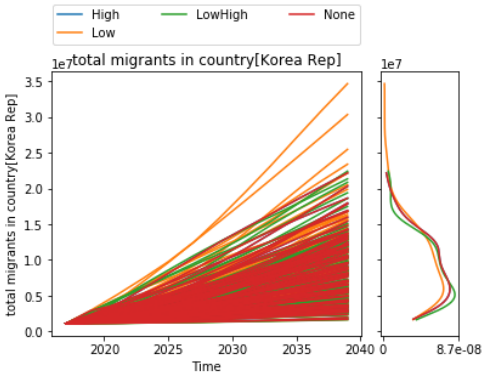

(c) Australia



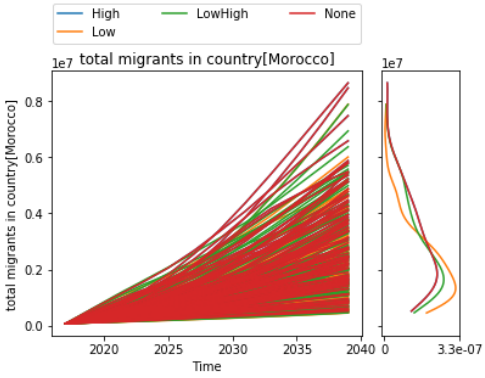

(d) Colombia



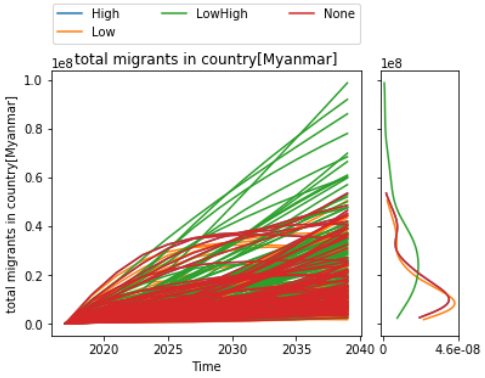

(e) Germany




(f) South Korea

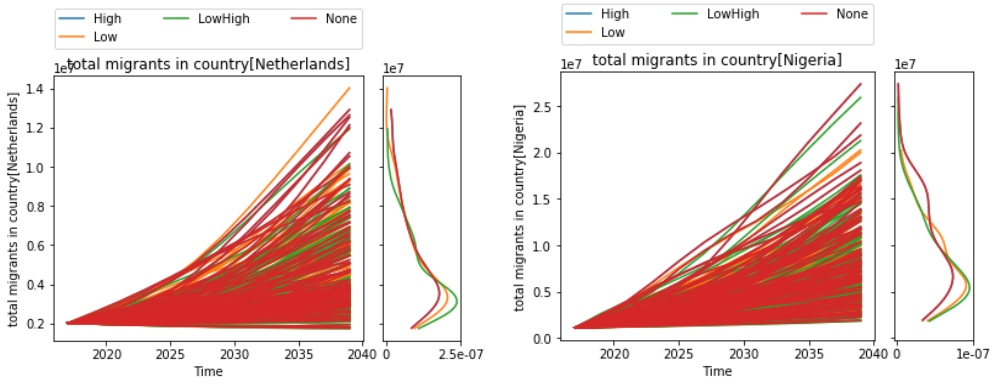



(g) Morocco




(h) Myanmar

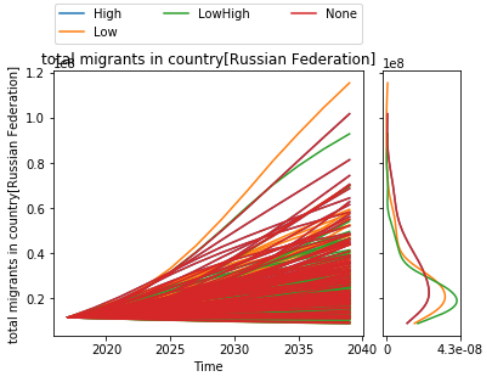
Figure 5.7: Global policy results. The outcomes for total number of migrants in various countries for 100 distinct scenarios. It can be observed that the 'LowHigh' policy is not beneficial for Afghanistan and Myanmar. Both the 'Low' and 'LowHigh' policies decrease migrant numbers in Germany, while Aruba is negatively affected by these. The 'High' policy does not seem to significantly deviate from the 'None' case.



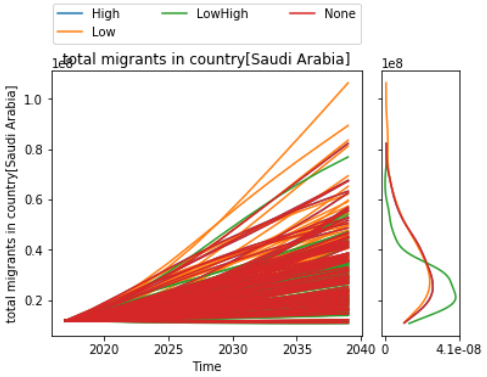

(a) Netherlands



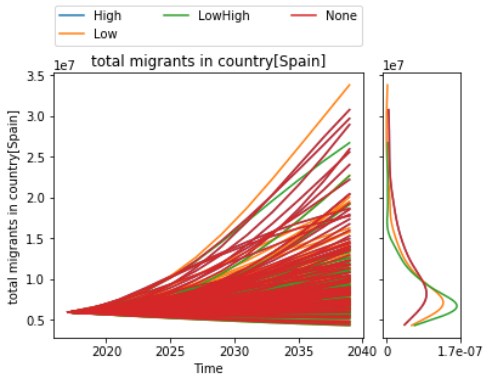

(b) Nigeria



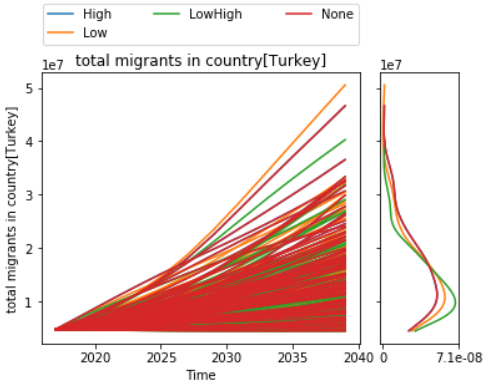

(c) Russia



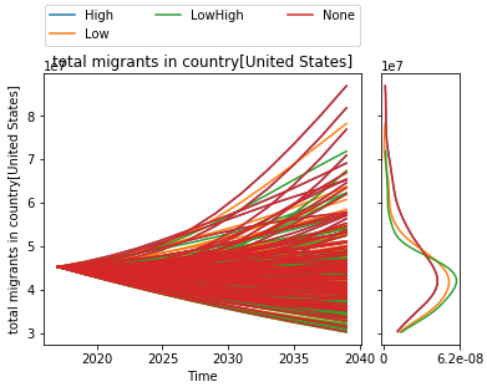

(d) Saudi Arabia



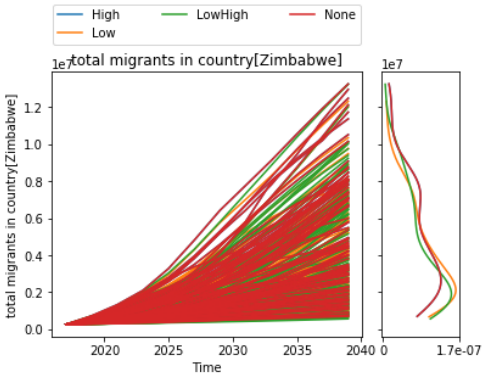

(e) Spain




(f) Turkey




(g) United States



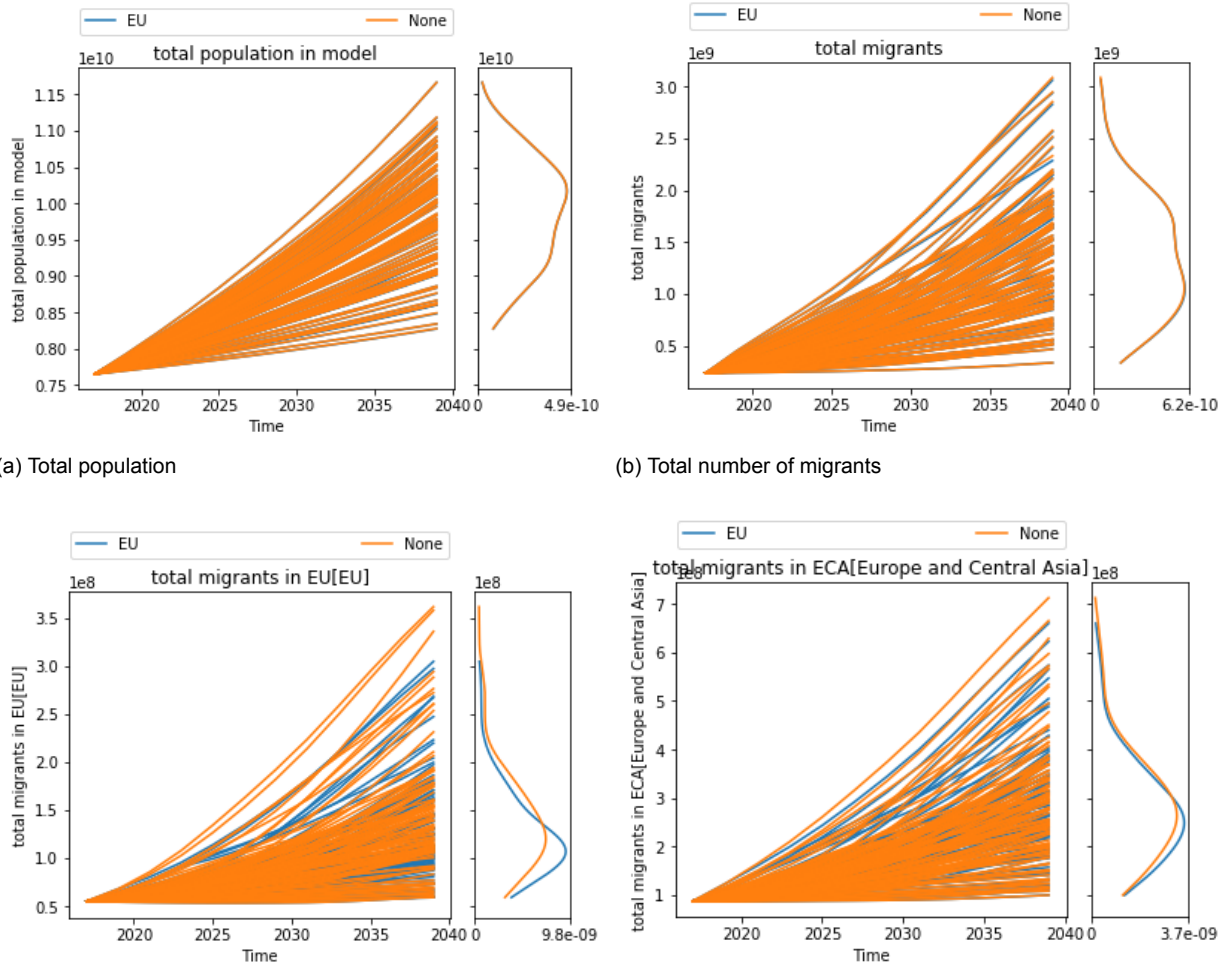

(h) Zimbabwe

Figure 5.8: Global policy results. The outcomes for total number of migrants in various countries for 100 distinct scenarios. Again, there is no significant difference between no policy and the 'High' scenario. This makes sense, as migrants are already attracted to these countries without policy. The 'LowHigh' policy significantly reduces migrant numbers in Russia, Saudi Arabia, and Spain.

## 5.4. Regional Policy: Closing the European Union

In this section, the influence of a regional policy in the European Union is assessed. In this case, the European Union decides to close all of its borders when a certain threshold of migrants is reached. This policy was implemented through subscript mapping (Appendix B), which allows for changing variables dynamically for entire regions.

- **Policy 'None'**: No closed borders.
- **Policy 'EU'**: The European Union closes off its borders when a certain threshold of migrants is exceeded. This threshold is random between 50 million and 100 million.



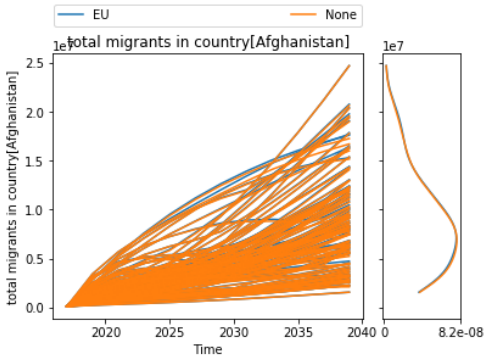

(a) Total population

(b) Total number of migrants

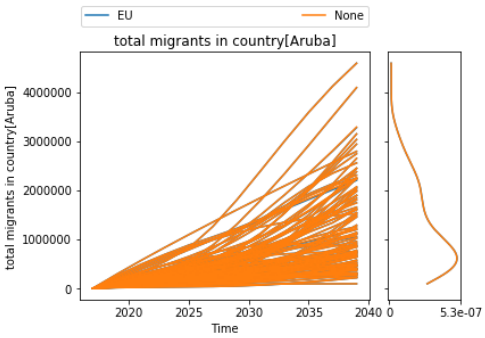
(c) Total number of migrants in the European Union

(d) Total number of migrants in Europe and Central Asia

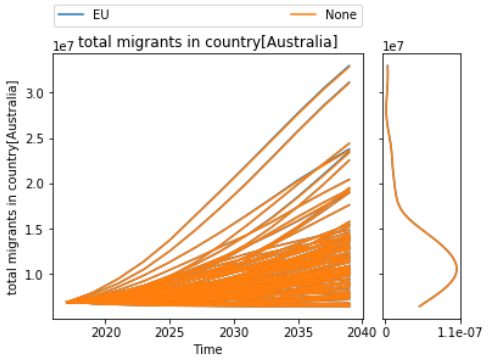
Figure 5.9: Results for regional policy model runs. The outcomes for total population, number of migrants, and number of migrants in the European Union and Europe and Central Asia for 100 distinct scenarios. The total number of migrants in the European Union in the 'EU' policy case is slightly lowered.



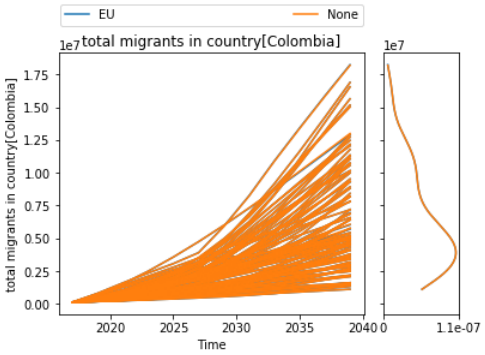

(a) Afghanistan



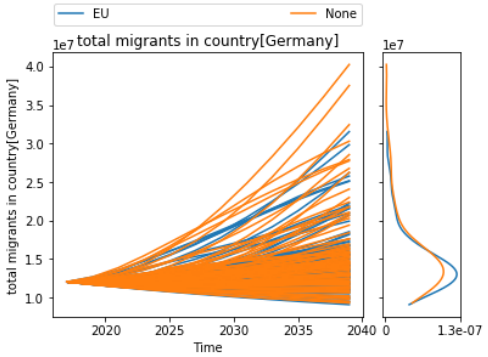

(b) Aruba



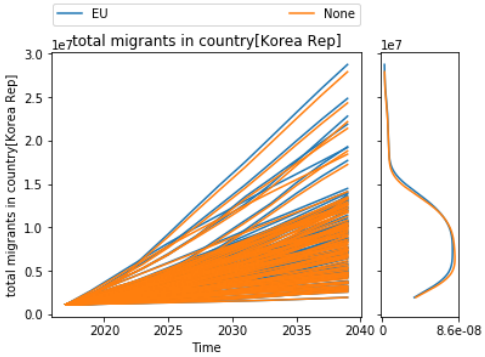

(c) Australia



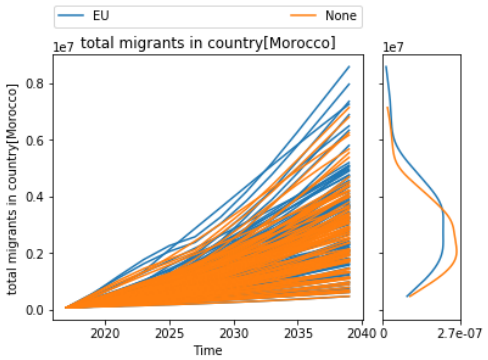

(d) Colombia



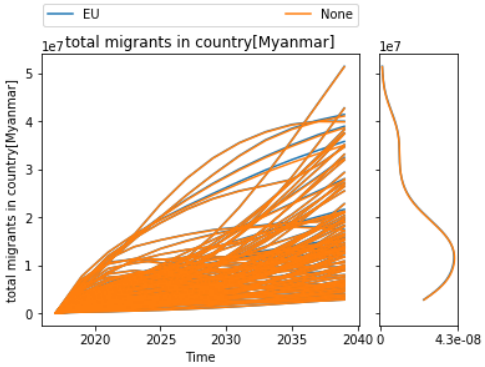

(e) Germany




(f) South Korea

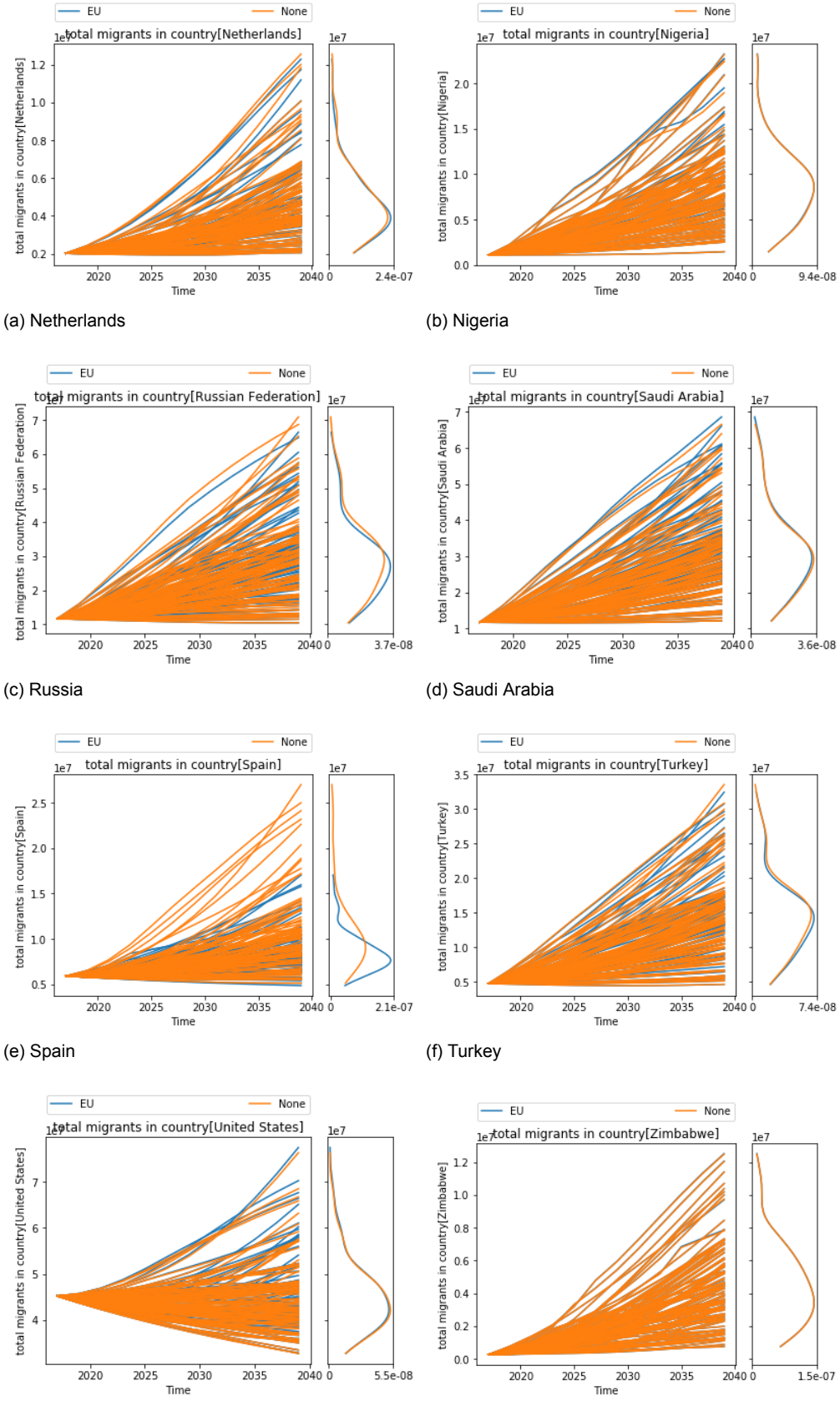



(g) Morocco




(h) Myanmar

Figure 5.10: Regional policy results. The outcomes for total number of migrants in various countries for 100 distinct scenarios. Two countries are interesting: Germany and Morocco. The density of the number of migrants in Germany is affected slightly, which is probably due to the travel time between the borders of the European Union and the German border. Morocco, being at the other side of the European Union borders, is definitely affected by this policy. The density of the simulation outcomes shifts upwards.




(a) Netherlands

(b) Nigeria

(c) Russia

(d) Saudi Arabia

(e) Spain

(f) Turkey

(g) United States

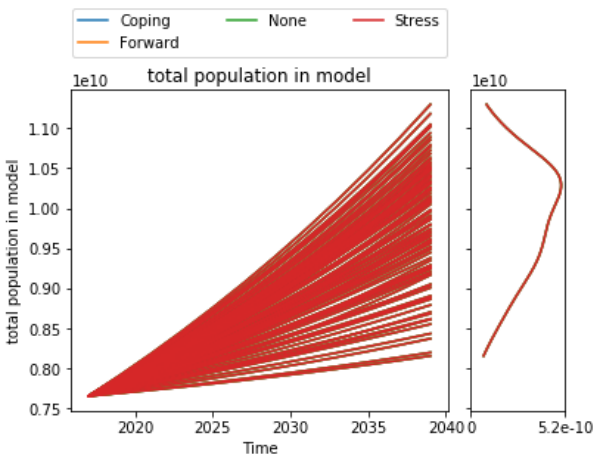
(h) Zimbabwe

Figure 5.11: Regional policy results. The outcomes for total number of migrants in various countries for 100 distinct scenarios. The 'EU' policy seems very beneficial for Spain, as significantly less migrants travel there.

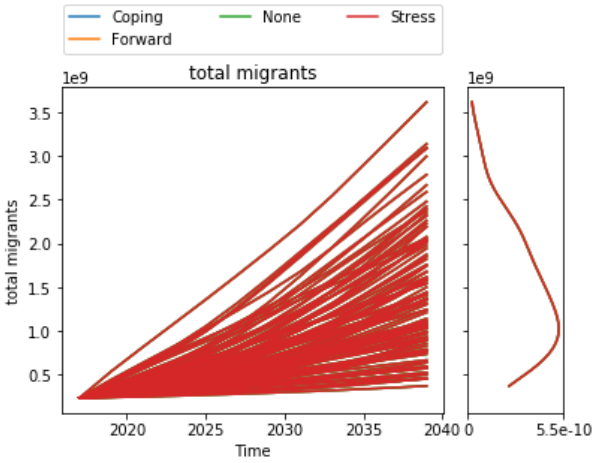
## 5.5. National Policy: Managing Societal Stress, Improving Coping Capacity, and Stimulating Transit

After global and regional policies in the previous sections, in this section the effect of national policies will be tested. Four different policies are implemented for the Netherlands and simulated across 100 scenarios. This allows for analyzing the effect of national policies, both in the country itself, and across the list of countries introduced in Section 5.2.

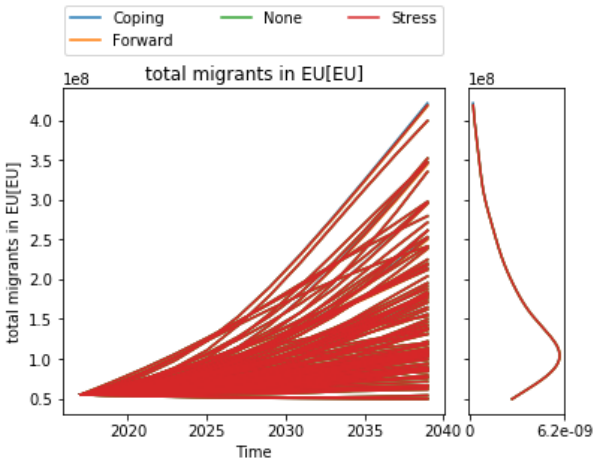
- **Policy 'None'**: No policy implemented.
- **Policy 'Stress'**: Societal stress is managed in the Netherlands. This means that there will be less polarization and negative effects towards migrants. Therefore, the attractiveness of the country is less influenced by the number of migrants in the country.
- **Policy 'Coping'**: The Netherlands increases its coping capabilities, meaning that there is more place for migrants. This increases the attractiveness of the country.
- **Policy 'Forward'**: This policy stimulates migrants to travel onwards to other countries.



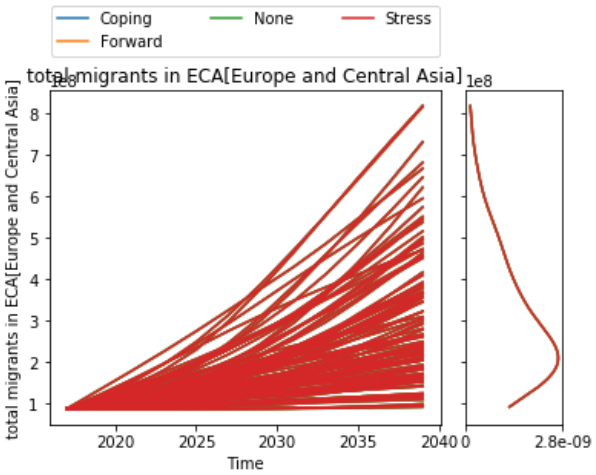

(a) Total population




(b) Total number of migrants

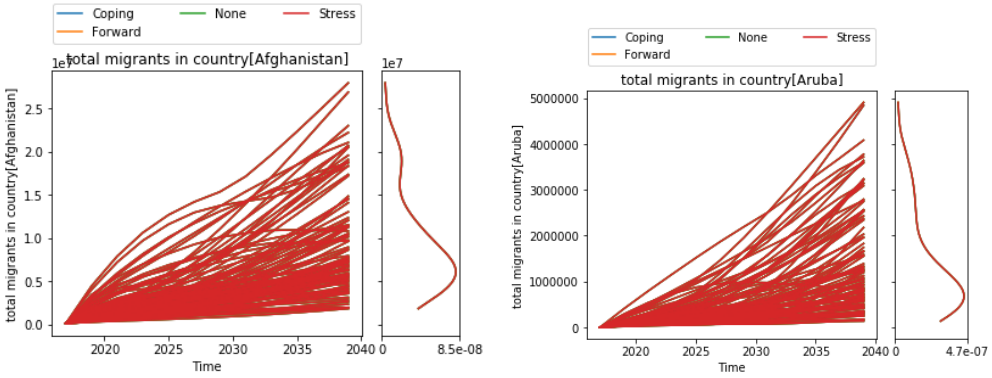



(c) Total number of migrants in the European Union




(d) Total number of migrants in Europe and Central Asia

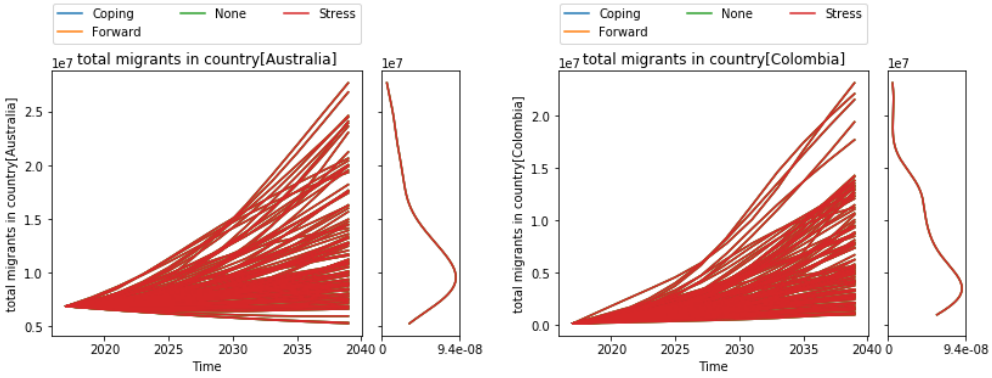
Figure 5.12: National policy results. The outcomes for total population, number of migrants, and number of migrants in the European Union and Europe and Central Asia for 100 distinct scenarios. All outcomes are unaffected by the Dutch policies.




(a) Afghanistan



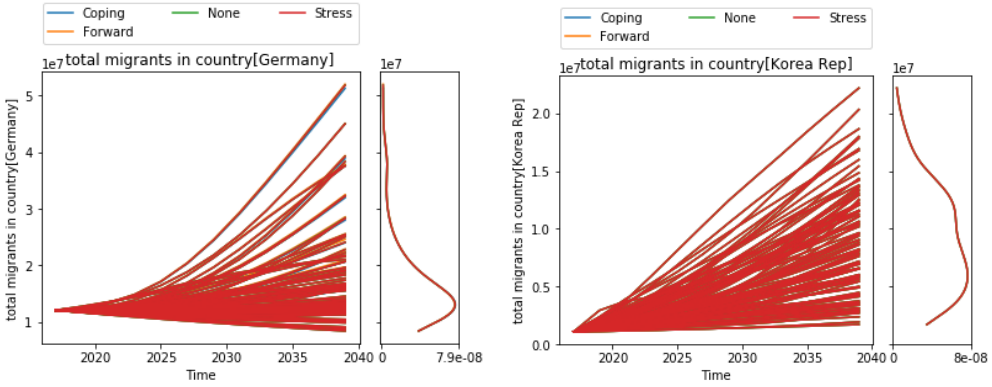

(b) Aruba




(c) Australia



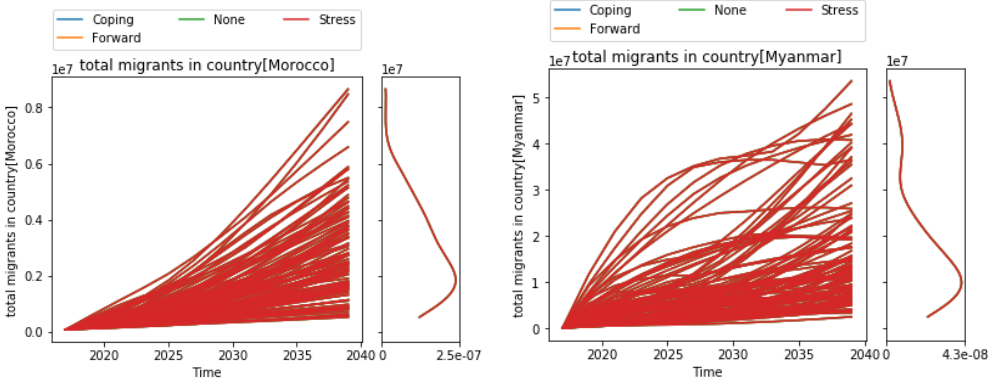

(d) Colombia




(e) Germany




(f) South Korea

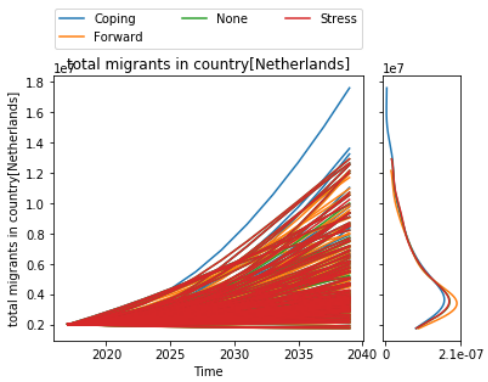



(g) Morocco

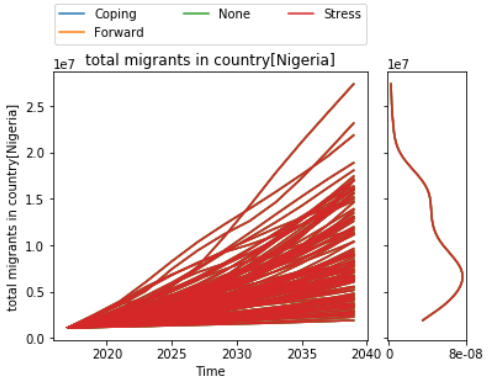



(h) Myanmar

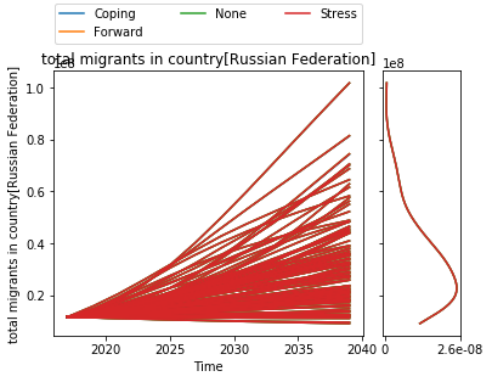
Figure 5.13: National policy results. The outcomes for total number of migrants in various countries for 100 distinct scenarios. It is clear that all countries are unaffected by the Dutch policies, even its direct neighboring country Germany.



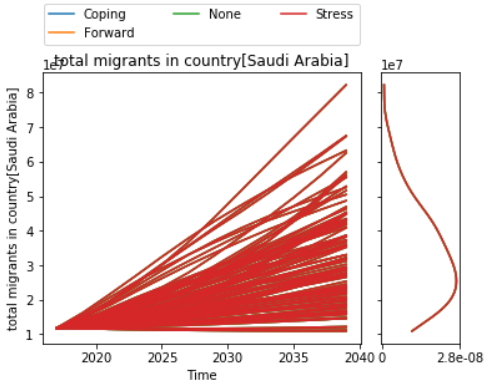

(a) Netherlands



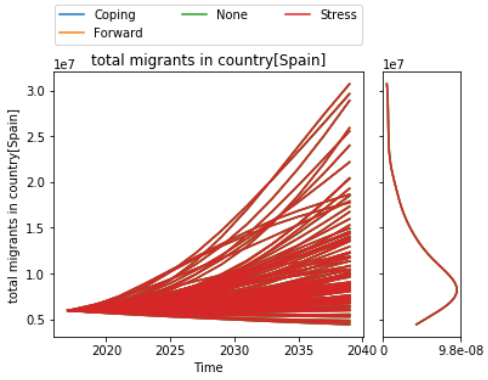

(b) Nigeria



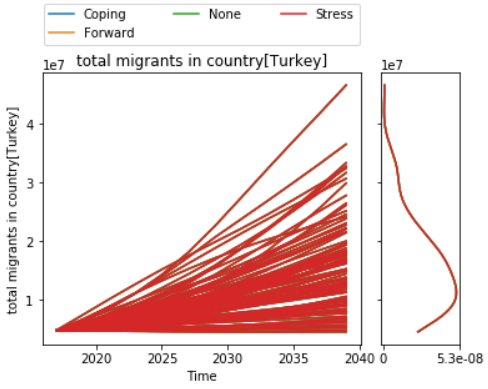

(c) Russia



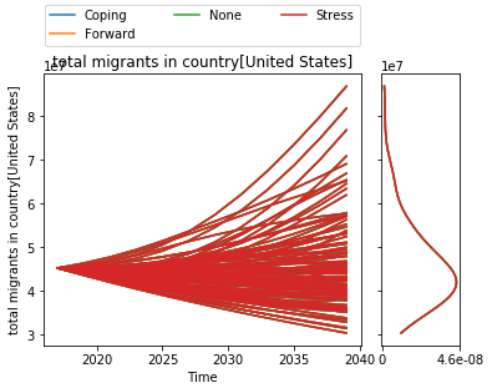

(d) Saudi Arabia



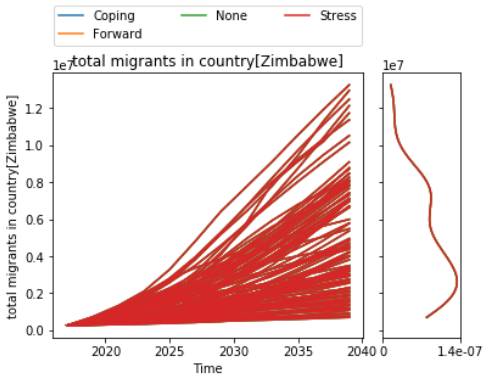

(e) Spain




(f) Turkey




(g) United States




(h) Zimbabwe

Figure 5.14: National policy results. The outcomes for total number of migrants in various countries for 100 distinct scenarios. All countries except the Netherlands are unaffected by the policies. The effects on the Netherlands itself are slight.

# 6

# Discussion

This chapter will first discuss results that were presented in Chapter 5 in Section 6.1. After that, it will reflect on the model and identify its shortcomings in Section 6.2. Subsequently, the data acquisition process will be subjected to a critical assessment in Section 6.3. Then, Section 6.4 will shortly discuss the model in relation to migration in general. Lastly, Section 6.5 will combine and convert the results and considerations into a policy advice.

## 6.1. Results

How useful are these results, and what do they actually indicate? This is one of the most important considerations in this regard. The model output is extremely large, which tends to a significant result overload. As such, it is difficult to analyze and interpret all results simultaneously. This quickly leads to a selective approach in search of both positive and negative extremes. Therefore, it is essential to reflect on the interpretation of the results.

Having said that, the results that were presented in the previous chapter are just a small selection of the total model output. Still, significant differences in migratory behavior occurred throughout various policies.

Upon implementation of the global policies, additional shelter and coping capabilities in low-income countries resulted in more migrants in these countries throughout all scenarios, while the same policy in high-income countries had no significant effect. The cause for this may be that the general trend is that the origin of migrants are primarily these low-income countries, and their preferred destinations are the higher income countries. Additional shelters and coping capabilities in high-income countries, would therefore have no significant effect on the route and destination of migrants. Moreover, since it was assumed that these additional coping capabilities were distributed evenly across all countries of a certain type, the relative difference in attractiveness of countries was not significantly affected. Another interesting observation can be made with respect to the effect of the 'LowHigh' policy. In this case, all countries improve their coping capabilities, which would basically result in a better migration infrastructure, and less societal stress throughout the world. According to the results, this would be beneficial for countries that play a central role in migrant transit from low- to high income regions, such as Turkey and Thailand. Since the total number of migrants are the same throughout the policies, this policy could very well lead to a more even distribution of migrants across countries.

The regional policy that closed the European Union after a certain migrant threshold had been reached resulted in significant differences for countries on both sides of the border (i.e. Spain and Morocco). Surprisingly, this does not hold true for Turkey. That the effect was far less for European countries further from the border, like Germany and the Netherlands, was in line with the expectation. This can be explained by the significant delay for migrants to reach those countries. When the threshold is reached, most of these additional migrants have not yet traveled to their final destination—and most of these migrants tend to travel towards the Western European countries. The delay effect in regional policy implementation

should therefore not be underestimated.

The implementation of national policies for the Netherlands clearly showed the inertia of the migration system. National policies—especially for smaller countries—hardly have any effect on the global migration system. It is interesting to see that the number of migrants in Germany is unaffected, but even the effects on the number of migrants in the Netherlands itself are limited. In that regard, national policies are like trying to empty the ocean with a thimble. Therefore, the results for the national policies support the notion that migration is a global phenomenon and large-scale policies are required to regulate migrant flows.

Concluding, in terms of results, the model definitely provided useful insights. It was shown that location of coping capabilities definitely affects migrant flows significantly, and, at the very least, this shows the interdependence of countries in terms of addressing this problem. That is, policies and decisions in one country can greatly affect the number and origin of migrants in other countries. Delay effects for policies should be taken into account as well, since it was shown that reactive policies may not give desirable results for all involved countries. National policies proved to be relatively ineffective within the migration system, even for the country itself.

## 6.2. Reflection on the Model

Although the model already led to some interesting insights in its current form, there is still a significant number of improvements and adaptations that can be made. Admitting that this indicates that the model is far from complete, the number—and nature—of these improvements could also be seen as an indicator that shows the versatility of the model component. This section will first discuss several of these additional implementations.

The most important aspect that should be addressed is model calibration. Due to time constraints, the current form of the model required a significant number of assumptions in terms of root causes, their effect on the migration mechanisms, and the interrelation and feedback effects of the internal migration mechanisms themselves. More thorough model calibration could severely decrease part of the uncertainty under which the model was simulated in this stage.

One of the other major considerations that should also be taken into account, is that this model operates on a country level. By adhering to this resolution, the model does not account for smaller, more delicate mechanisms and phenomena. However, these types of generalizations and simplifications are inherent to using System Dynamics as a modeling technique. Any complexification in terms of scope or resolution would rapidly lead to substantial problems in simulation and in result interpretation. On a lower level, it would be interesting to add spatial internal migration mechanisms (IDPs), to identify population and migrant density within countries.

Another aspect that could be taken into account is the distance that needs to be traveled in each country, which could greatly affect the time migrants take to transit through a certain country. This is especially true for countries with a less-developed infrastructure. The difficulty of border crossings has been implemented to some extent, but this could have been done more thoroughly by taking more indicators into account (e.g. visa requirements).

In addition, the model does not take ethnicities into account, while this might be an important factor in terms of migratory decisions. In some countries, only people with a certain ethnicity are likely to migrate due to for instance oppression or political/social persecution. On top of that, it might influence decisions these migrants make later on in the migration process in terms of travel decisions.

Lastly, the distribution mechanisms required some severe simplifications. At present, the residence attractiveness, transit attractiveness, and accessibility of each country are each represented in a single number. Of course, this means that the weight that is given to each of these factors is of utmost importance. Therefore, a proper identification and classification of their actual effects—or a better distinction between them—could lead to significant gains in terms of model utility. In the current model, the magnitude and weight of these factors were part of the uncertainty space.

## 6.3. Data

The model that was developed in this study relied heavily on data. While this definitely provides benefits, there are also some issues that need to be taken into account. The first issue that should be taken into account is data accuracy. While some of the indicators that were used are hard, countable numbers (e.g. population, country borders, or GDP), others—estimated indicators—are the result of complex calculations and data combinations (e.g. the Fragile State Index, and the Freedom in the World dataset). The latter are based on estimates, and it is sometimes hard to establish what they actually represent.

The second major issue concerning data is its completeness. Mostly, data sets can diverge in terms of number of countries, but also within data sets not all data might be available. To account for this lack of data, several methods were employed. These methods however, do certainly not improve model reliability. Especially in the case of the previously mentioned estimated indicators, this leads to an estimation of an estimation, and it should be considered that these numbers may be highly inaccurate.

A specific case in terms of data is the language data. The languages spoken in each country—and the percentage of the population that speaks it—were assessed and compared to other countries, which resulted in a certain overlap between them. However, not all languages are fit for this purpose. For instance, while the official language in various countries in South America is Spanish, these forms are significantly different to the version of Spanish spoken in Spain. The current implementation does not take these differences in to account.

## 6.4. Migration in General

In this study, the global nature of the migration phenomenon has been emphasized on multiple occasions. The results have more than confirmed this belief, since effects of national policies were hardly visible anywhere throughout the world.

This highlights the interconnectedness and complexity of this model. A solution to migration problems should not only be sought on a national or regional level, but should also focus on the effects on a global scale. In addition, positive effects of migration on a global scale should be taken into account. Migration might lead to significant progress and substantially equalize the distribution of resources and knowledge throughout the world. In fact, one could even argue that migration should be fostered and stimulated for human progression. As such, migration could provide the solution to its own causes and problems: better (economic) circumstances throughout the world, and more cultural and social understanding.

## 6.5. Policy Advice

European countries are at present trying to solve the European migration problem, but they are not addressing the actual causes of this problem. In fact, their focus lies on mitigating its effects. The sustainability of this approach might be questioned, since such policies lead to migrant overflows in relatively low-income countries. These policies may lead—in due time—to even more migration, because it is unlikely that these countries are able to cope with these numbers.

Therefore, the first advice from this study towards policy-makers is that it might be more effective to address the root causes of migration, rather than just mitigating its detrimental effects. At the very least, it should receive the same amount of attention and resources, since these root causes form the basis of the problems that arise with migration.

In addition, the results showed the significant interdependence of countries regarding migration problems. The inertia of the migration system requires globally aligned, large-scale policy implementations to prevent and mitigate migration-related problems. The delay in these policies should not be neglected, so it is important to identify potential problems as early as possible and take appropriate action.

# 7

# Conclusions and Recommendations

This study has developed the following innovations on a methodological level: a technique to comprehensively implement spatial phenomena into System Dynamics, a method to create multi-scale System Dynamics models through subscript mapping, and a semi-automated data acquisition process to obtain and structure a great variety of data sets on a country level. The nature of the developed model component is generic. That is, it can be used on any scale and on any resolution. The number of unique entities taken into account does not affect the basic model mechanisms. For instance, it could also be used to gain insight in regional or national migration flows by simply changing the initialization parameters.

To improve the value of the model, several important aspects could be addressed. Firstly, the biggest gain would result from more thorough model calibration. An improved assessment of the actual relations between model parameters and variables could lead to more realistic results. However, the actual relation between root causes and their corresponding migration rates is hard to determine, since this requires a substantial amount of data that is not trivially available. An example that could help with this dilemma is to use geospatial data (for instance from Twitter), to track and quantify actual migrant flows. Secondly, to more accurately assess the true spectrum of plausible future migration flows, several types of migration mechanisms should be implemented. Such an approach would also allow for the quantitative comparison of various theories regarding migration, which would also be a quite interesting addition in itself.

Another area where there is definitely room for improvement is in the data acquisition process. More advanced data analysis and processing techniques could greatly improve the added value for the current model component.

In terms of policy advice, the model can be subjected to a more extensive exploration of scenarios and the effect of a multitude of policies. A more country- or region-specific approach could lend itself well for this purpose. As such, on the one hand it could be explored what the effect is of regional or national policies on the global scale, but on the other hand it could also be employed to determine the effects of events throughout the world specifically for a country or region. The latter case would allow for the identification and testing of various mitigation policies for e.g. immigration and naturalization services, national governments, regional institutions, or NGOs.

# Bibliography


Alvarez, G. and Marcaletti, M. (2018). Venezuela is forcing these people to flee the country with their furniture on their backs. Buzzfeed News. Retrieved from https://www.buzzfeed.com/gretaalvarez/deportan-colombianos-de-venezuela-282?utm_term=.xp8WPPyvXy#.furynn3ej3. Last accessed on June 27, 2018.

Bader, V. (2005). The ethics of immigration. *Constellations*, 12(3):331–361.

Bairoch, P. (1991). *Cities and economic development: from the dawn of history to the present*. University of Chicago Press.

Bamshad, M., Kivisild, T., Watkins, W. S., Dixon, M. E., Ricker, C. E., Rao, B. B., Naidu, J. M., Prasad, B. R., Reddy, P. G., Rasanayagam, A., et al. (2001). Genetic evidence on the origins of indian caste populations. *Genome research*, 11(6):994–1004.

Bandopadhyay, D. (2017). Reflection on the rohingya crisis. Politics Now. Retrieved from http://politicsnow.in/rohingya-crisis/. Last accessed on June 27, 2018.

Bankes, S. (1993). Exploratory modeling for policy analysis. *Operations research*, 41(3):435–449.

BBC News (2018). Migrant crisis: Migration to europe in historical perspective. Retrieved from https://bbc.com/news/world-europe-34131911. Last accessed on June 27, 2018.

Beine, M., Bertoli, S., and Fernández-Huertas Moraga, J. (2016). A practitioners' guide to gravity models of international migration. *The World Economy*, 39(4):496–512.

Bertoli, S. and Moraga, J. F.-H. (2013). Multilateral resistance to migration. *Journal of Development Economics*, 102:79–100.

Boardman, J., Griffin, J., and Murray, O. (2001). *The Oxford illustrated history of the Roman world*, volume 2. Oxford Paperbacks.

Bündnis Entwicklung Hilft (n.d.). World risk index. Retrieved from https://weltrisikobericht.de/english-2/. Last accessed on May 28, 2018.

Cambridge Dictionary Online (n.d.). Migrate [def. 2]. Retrieved from https://dictionary.cambridge.org/english/migrate. Last accessed on June 25, 2018.

Castles, S., De Haas, H., and Miller, M. J. (2013). *The age of migration: International population movements in the modern world*. Macmillan International Higher Education.

Center for Geographic Analysis (n.d.). Border crossings. Harvard University. Retrieved from https://worldmap.harvard.edu/data/geonode:border_crossings_phv. Last accessed on June 12, 2018.

Centraal Bureau voor de Statistiek (2018). Asielverzoeken; nationaliteit, vanaf 1975. Retrieved from https://opendata.cbs.nl/statline/#/CBS/nl/dataset/80059ned/table?ts=1532856653802. Last accessed on July 29, 2018.

Central Intelligence Agency (n.d.). The world factbook. Retrieved from https://www.cia.gov/library/publications/the-world-factbook/. Last accessed on April 15, 2018.

De Haas, H. (2010). The internal dynamics of migration processes: A theoretical inquiry. *Journal of ethnic and migration studies*, 36(10):1587–1617.

Dörrbecker, M. (2015a). Map of the european migrant crisis 2015. Retrieved from https://commons.wikimedia.org/wiki/File:Map_of_the_European_Migrant_Crisis_2015.png. Last accessed on June 27, 2018.

Dörrbecker, M. (2015b). Map of the european migrant crisis 2015 - asylum applicants. Retrieved from https://commons.wikimedia.org/wiki/File:Map_of_the_European_Migrant_Crisis_2015_-_Asylum_applicants%27_countries_of_origin.png. Last accessed on June 27, 2018.

Eltis, D. (1987). *Economic growth and the ending of the transatlantic slave trade*. New York, NY: Oxford University Press.

Eurostat (n.d.). Asylum and managed migration. Retrieved from http://ec.europa.eu/eurostat/web/asylum-and-managed-migration/data/database. Last accessed on July 29, 2018.

Forrester, J. W. (1971). *World dynamics*, volume 59. Wright-Allen Press Cambridge, MA.

Freedom House (n.d.). Freedom in the world. Retrieved from https://freedomhouse.org/report-types/freedom-world. Last accessed on May 29, 2018.

Friedrich Ebert Stiftung (2015). The kosovo torrent to eu: People, reasons and ways. Retrieved from http://library.fes.de/pdf-files/bueros/kosovo/13857.pdf. Last accessed on June 27, 2018.

Geddes, A. and Scholten, P. (2016). *The politics of migration and immigration in Europe*. Sage.

Gray, R. and Jordan, F. (2000). Language trees support the express-train sequence of austronesian expansion. *Nature*, 405:1052.

Hoffmann-Nowotny, J. (1978). European migration after world war ii.

Hopkins, D. M. (1959). Cenozoic history of the bering land bridge. *Science*, 129(3362):1519–1528.

Internal Displacement Monitoring Centre and Norwegian Refugee Council (2017). Africa report on internal displacement. Retrieved from https://reliefweb.int/sites/reliefweb.int/files/resources/20171206-Africa-report-2017-web.pdf. Last accessed on July 25, 2018.

International Committee of the Red Cross (2017). Internally displaced persons and international humanitarian law. Retrieved from https://www.icrc.org/en/download/file/62432/internally_displaces_persons_2017.pdf [*sic*]. Last accessed on July 27, 2018.

Kwakkel, J. (n.d.). Ema workbench [python package]. Retrieved from https://github.com/quaquel/EMAworkbench. Last accessed on August 5, 2018.

Kwakkel, J. H. and Pruyt, E. (2015). Using system dynamics for grand challenges: the esdma approach. *Systems Research and Behavioral Science*, 32(3):358–375.

Levitt, P. (1998). Social remittances: Migration driven local-level forms of cultural diffusion. *International migration review*, 32(4):926–948.

Levitt, P. and Lamba-Nieves, D. (2011). Social remittances revisited. *Journal of Ethnic and Migration Studies*, 37(1):1–22.

Lucassen, L. (2018). Peeling an onion: the “refugee crisis” from a historical perspective. *Ethnic and Racial Studies*, 41(3):383–410.

Madison, G. (2006). Existential migration. *Existential analysis*, 17(2):238–260.

Manning, P. (2012). *Migration in world history*. Routledge.

Matisoo-Smith, E. A. (2015). Tracking austronesian expansion into the pacific via the paper mulberry plant. *Proceedings of the National Academy of Sciences*, 112(44):13432–13433.

McNeill, W. H. (1984). Human migration in historical perspective. *Population and Development Review*, pages 1–18.

Meadows, D. H., Meadows, D. H., Randers, J., and Behrens III, W. W. (1972). The limits to growth: a report to the club of rome (1972). *Google Scholar*.

Merriam-Webster Online (n.d.). Migrate [def. 1]. Retrieved from https://www.merriam-webster.com/dictionary/migrate. Last accessed on June 25, 2018.

National Geographic Society (2008). Genetic analyses indicate that there were three distinct waves of prehistoric migrants from asia to north america. Retrieved from https://natgeoeducationblog.files.wordpress.com/2016/05/human-migration-2008.jpg . Last accessed on July 28, 2018.

Neuwirth, C., Peck, A., and Simonović, S. (2015). Modeling structural change in spatial system dynamics: A daisyworld example. *Environmental Modelling & Software*, 65:30–40.

Omohundro, J. T. (2008). *Thinking like an anthropologist: A practical introduction to cultural anthropology*. McGraw Hill.

Oxford Dictionaries Online (n.d.). Migrate [def. 1]. Retrieved from https://en.oxforddictionaries.com/definition/migrate. Last accessed on June 25, 2018.

Poot, J., Alimi, O., Cameron, M. P., and Maré, D. C. (2016). The gravity model of migration: the successful comeback of an ageing superstar in regional science.

Pruyt, E. (2013). Small system dynamics models for big issues: Triple jump towards real-world complexity.

Pruyt, E., Cunningham, S., Kwakkel, J., and De Bruijn, J. (2014). From data-poor to data-rich: system dynamics in the era of big data. In *32nd International Conference of the System Dynamics Society, Delft, The Netherlands, 20-24 July 2014; Authors version*. The System Dynamics Society.

Ravenstein, E. G. (1885). The laws of migration, part 1. *Journal of the statistical society of London*, 48(2):167–235.

Ravenstein, E. G. (1889). The laws of migration, part 2. *Journal of the royal statistical society of London*, 52(2):241–305.

Reich, D., Patterson, N., Campbell, D., Tandon, A., Mazieres, S., Ray, N., Parra, M. V., Rojas, W., Duque, C., Mesa, N., et al. (2012). Reconstructing native american population history. *Nature*, 488(7411):370.

Rickey, V. F. (1992). How columbus encountered america. *Mathematics Magazine*, 65(4):219–225.

Robinson, S. (1997). Simulation model verification and validation: increasing the users' confidence. In *Proceedings of the 29th conference on Winter simulation*, pages 53–59. IEEE Computer Society.

Şeker, N. (2013). Forced population movements in the ottoman empire and the early turkish republic: An attempt at reassessment through demographic engineering. *European Journal of Turkish Studies. Social Sciences on Contemporary Turkey*, (16).

Sherouse, O. (2014). Wbdata. Arlington, VA. Retrieved from http://github.com/OliverSherouse/wbdata. Last accessed on July 29, 2018.

Simini, F., González, M. C., Maritan, A., and Barabási, A.-L. (2012). A universal model for mobility and migration patterns. *Nature*, 484(7392):96.

Simonovic, S. P. (2002). World water dynamics: global modeling of water resources. *Journal of Environmental Management*, 66(3):249–267.

Skeldon, R. (2014). *Migration and development: A global perspective*. Routledge.

Symmes Cobb, J. (2015). Brazil, argentina seek to resolve colombia-venezuela border spat. Reuters. Retrieved from https://www.reuters.com/article/us-venezuela-colombia/brazil-argentina-seek-to-resolve-colombia-venezuela-border-spat-idUSKCN0R42CJ20150904. Last accessed on June 27, 2018.

The Correlates of War Project (n.d.). Direct contiguity (v3.2). Retrieved from http://correlatesofwar.org/data-sets/direct-contiguity. Last accessed on April 1, 2018.

The Fund for Peace (n.d.). Fragile states index. Retrieved from http://fundforpeace.org/fsi/data. Last accessed on May 29, 2018.

The Guardian (2011). Angolan teenager mauro manuel loses fight to stay in the netherlands. Retrieved from https://www.theguardian.com/world/2011/nov/01/angola-teenager-netherlands-asylum?INTCMP=ILCNETTXT3487. Last accessed on August 15, 2018.

The Guardian (2018). Eu to consider plans for migrant processing centres in north africa. Retrieved from https://www.theguardian.com/world/2018/jun/19/eu-migrant-processing-centres-north-africa-refugees. Last accessed on August 15, 2018.

The New York Times (2018). Migration to europe is down sharply. so is it still a 'crisis'? Retrieved from https://www.nytimes.com/interactive/2018/06/27/world/europe/europe-migrant-crisis-change.html.

The World Bank (n.d.a). Exploring climate and development. Retrieved from http://climate4development.worldbank.org/open/. Last accessed on June 6, 2018.

The World Bank (n.d.b). World development indicators. Retrieved from https://data.worldbank.org/products/wdi and through Python API. Last accessed on April 5, 2018.

United Nations (2017a). World population prospects. Retrieved from https://esa.un.org/unpd/wpp/. Last accessed on June 22, 2018.

United Nations, D. (2017b). International migrant stock. Retrieved from http://un.org/en/development/desa/population/migration/data/estimates2/estimates17.shtml. Last accessed on April 15, 2018.

Van Hear, N., Bakewell, O., and Long, K. (2018). Push-pull plus: reconsidering the drivers of migration. *Journal of Ethnic and Migration Studies*, 44(6):927–944.

Vasey, D. E. (2002). *An ecological history of agriculture 10,000 BC-AD 10,000*. Purdue University Press.

Von Bertalanffy, L. (1973). The meaning of general system theory. *General system theory: Foundations, development, applications*, pages 30–53.

Williams, A. M. (2009). International migration, uneven regional development and polarization. *European Urban and Regional Studies*, 16(3):309–322.

Wilson III, E. J. (2008). Hard power, soft power, smart power. *The Annals of the American Academy of Political and Social Science*, 616(1):110–124.

Zimmermann, K. F. (1996). European migration: Push and pull. *International Regional Science Review*, 19(1-2):95–128.

# A

# Migration Dynamics Model Component

The figure below shows a simplified version of the migration dynamics model component. This model component can be connected to any other Vensim model by specifying the variables depicted in red. The variables depicted in cyan are initial values, and require data input.

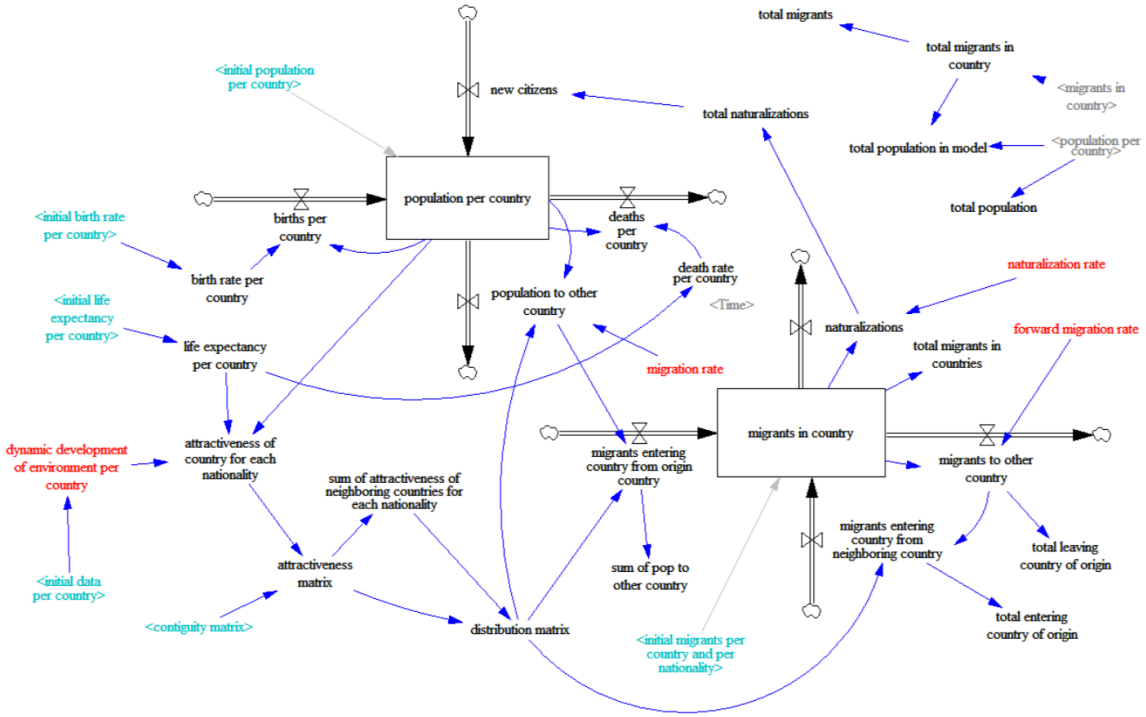



Figure A.1: Model component for migration dynamics that tracks country of origin of migrants, and allows for separate migration mechanisms for each nationality.

# B

# Subscript Mapping

**Subscripting with regions**

- **Step 1: Country subscript**
  - In Vensim: Click subscript click New.. (or Edit... when it already exists) enter name of subscript add equation: GET XLS SUBSCRIPT('*name of datafile*', '*name of sheet*' , '*named range of countries*' , " )
  - This will load all the countries into the Vensim model within the subscript *country*.
  - Example: GET XLS SUBSCRIPT('Data.xlsx', 'Country Data' , 'A2' , 'A218' , " )
- **Step 2: Creating subranges for countries in each region**
  - Make sure the countries in the Excel file are sorted by region, or that the named ranges are correctly implemented. For each region, we have to define in Vensim which countries belong to it.
  - In Vensim: Click subscript click New.. (or Edit... when it already exists) enter the name of the region add equation: GET XLS SUBSCRIPT('*name of datafile*', '*name of sheet*' , '*named range of countries*' , " )
  - Vensim now automatically creates a subrange containing the countries in the selected cells. Repeat this process for each region.
  - Example: GET XLS SUBSCRIPT('Data.xlsx', 'Country Data' , 'A2' , 'A218' , " )
- **Step 3: Mapping the regions onto the subranges**
  - The regions are now defined as subranges of country. However, suppose we have a value for a variable for an entire region [e.g. sheep birth rate is the same throughout each region], we must first create another subscript range called regions.
  - We retrieve the regions from the Excel-sheet and map them onto the subranges of country like follows: GET XLS SUBSCRIPT('*name of datafile*', '*name of sheet*' , '*named range of regions*' , "*specify a prefix*" ) -> (*subcript countries: 'region subrange 1', 'region subrange 2', '...'* )
  - Example: GET XLS SUBSCRIPT('Data.xlsx', 'Regional Data' , 'A2' , 'A8' , 'region' ) -> (country: "East Asia and Pacific","Europe and Central Asia","Latin America and Caribbean","Middle East and North Africa", North America , South Asia ,"Sub Saharan Africa")
- **Step 4: Defining the variables**
  - Now, to define a variable for an entire region, use the subscript range *[region]*, and to define a variable on country level, use *[country]*.

- **Step 5: Multiple subranges**
  - If multiple subranges exist, for instance origin and reception countries, a separate regional subrange needs to be mapped on each corresponding subrange. If this is done correctly, Vensim can make the distinction between these subranges.
  - Example:
  - Example: GET XLS SUBSCRIPT('Data.xlsx', 'Regional Data' , 'A2' , 'A8' , 'region' ) -> (countries: "East Asia and Pacific","Europe and Central Asia","Latin America and Caribbean","Middle East and North Africa", North America , South Asia ,"Sub Saharan Africa")
  - Example: GET XLS SUBSCRIPT('Data.xlsx', 'Regional Data' , 'A2' , 'A8' , 'receptionregion' ) -> (receptioncountries: "East Asia and Pacific","Europe and Central Asia","Latin America and Caribbean","Middle East and North Africa", North America , South Asia ,"Sub Saharan Africa")

# C

# Uncertainties and Policies

| Variable | Lower Range | Upper Range |
|---|---|---|
| pc of pop atR of conflict violence | 0.2 | 0.95 |
| pc of pop at risk of oppression | 0.25 | 0.95 |
| pop atRd2 conflict violence willing2 migrate | 0.2 | 0.8 |
| pc pop atR of oppression willing2 migrate | 0.1 | 0.6 |
| pc pop atRd2 disasters willing2 migrate | 0.1 | 0.6 |
| pc pop atRd2 economic or food scarcity willing2 migrate | 0.1 | 0.6 |
| pc pop atR of oppression willing2 migrate | 0.1 | 0.6 |
| fraction of popRd2 conflict violence RWA to migrate trying per year | 0.1 | 0.6 |
| pc overlap in pop atR willing and able2 migrate due to conflict and oppression | 0.1 | 0.8 |
| pc pop atR of conflict violence fysically and financially able to migrate faraway | 0.05 | 0.65 |
| pc pop atR of conflict violence oppression NOT even able to migrate to other country in region | 0.05 | 0.6 |
| pc pop atRd2 economic or food scarcity fysically and financially able to migrate faraway | 0.05 | 0.75 |
| fr extra autonomous development exposure | 0.001 | 0.1 |
| fraction of population able to afford advanced transportation high income | 0.9 | 1 |
| fraction of population able to afford advanced transportation upper middle income | 0.4 | 0.7 |
| fraction of population able to afford advanced transportation lower middle income | 0.1 | 0.4 |
| fraction of population able to afford advanced transportation low income | 0 | 0.2 |
| birth rate scale factor high income | 0.5 | 1.5 |
| birth rate scale factor upper middle income | 0.5 | 1.5 |
| birth rate scale factor lower middle income | 0.5 | 1.5 |
| birth rate scale factor low income | 0.5 | 1.5 |
| death rate scale factor high income | 0.5 | 1.5 |
| death rate scale factor upper middle income | 0.5 | 1.5 |
| death rate scale factor lower middle income | 0.5 | 1.5 |
| death rate scale factor low income | 0.5 | 1.5 |
| migration rate scale factor high income | 0.2 | 1.1 |
| migration rate scale factor upper middle income | 0.2 | 1.1 |
| migration rate scale factor lower middle income | 0.2 | 1.1 |
| migration rate scale factor low income | 0.2 | 1.1 |
| naturalization rate scale factor high income | 0.001 | 0.02 |
| naturalization rate scale factor upper middle income | 0.001 | 0.02 |
| naturalization rate scale factor lower middle income | 0.001 | 0.02 |
| naturalization rate scale factor low income | 0.001 | 0.02 |
| societal stress influence factor | 1 | 10 |
| societal stress threshold pc | 0.4 | 0.9 |
| transit weight | 1 | 10 |
| residence weight | 1 | 10 |
| interregion migration factor | 0.5 | 5 |
| migrant coping capacity growth rate per country scale factor | 0.01 | 0.6 |
| pc of current coping capacity scale factor | 0.3 | 0.9 |
| forward migration rate scale factor | 0.1 | 0.9 |
| fraction of migrants applying in most attractive country scale factor | 0.3 | 1 |
| close EU borders threshold | 50000000 | 100000000 |

Table C.1: Overview of the uncertainty space for the model.

| Policy | None | Low | High | LowHigh |
|---|---|---|---|---|
| SWITCH low income shelter increase policy | 0 | 1 | 0 | 1 |
| SWITCH high income shelter increase policy | 0 | 0 | 1 | 1 |

Table C.2: Overview of global policy model settings.

| Policy | None | EU |
|---|---|---|
| SWITCH close borders EU | 0 | 1 |

Table C.3: Overview of regional policy model settings.

| Policy | None | Stress | Coping | Forward |
|---|---|---|---|---|
| forward migration rate Netherlands | 0.01 | 0.01 | 0.01 | 0.9 |
| migrant coping capacity growth rate Netherlands | 0 | 0 | 0.1 | 0 |
| manage societal stress Netherlands | 1 | 0.2 | 1 | 1 |

Table C.4: Overview of national policy model settings.

# D

# Python Scripts

## D.1. Country Names Database

# 01 - Country Names Database

August 15, 2018

## 1 Notebook 1 - Country Names Database

*@author: Stefan Wigman, 2018*

### 1.1 1.1 - Introduction

This notebook converts the country name database that was generated in the R script * Convert-ForLoop.R *to a directly importable CSV-file for Python. This CSV-file can then be imported as a* dictionary* in future Python notebooks.

### 1.2 1.2 - Importing the required Python packages

This notebook uses the Python packages Pandas, and csv.

```
In [1]: import pandas as pd
        import csv
```

### 1.3 1.3 - Importing the Country Name Database

The first step is to import the 'alternative name' database to a Pandas DataFrame:

```
In [2]: country_name_database = pd.read_csv("processed_data/CountryCodes/codelist.csv")
```

```
In [3]: country_name_database.head()
```

```
Out[3]:          ar5 continent     country.name.de country.name.de.regex  \
        1       ASIA      Asia         Afghanistan                afghan
        2   OECD1990    Europe       Aland Islands                 åland
        3        EIT    Europe            Albanien              albanien
        4        MAF    Africa            Algerien              algerien
        5       ASIA   Oceania  Amerikanisch-Samoa  ^(?=.*amerik).*samoa

          country.name.en country.name.en.regex     cow.name cowc   cown  ecb  \
        1     Afghanistan                afghan  Afghanistan  AFG  700.0   AF
        2   Åland Islands                 åland          NaN  NaN    NaN  NaN
        3         Albania               albania      Albania  ALB  339.0   AL
        4         Algeria               algeria      Algeria  ALG  615.0   DZ
        5  American Samoa  ^(?=.*americ).*samoa          NaN  NaN    NaN   AS
```

```
          ...        cldr.variant.yav  cldr.variant.yi  \
1         ...               Afkanistá
2         ...                      AX               AX
3         ...                  Alpaní
4         ...                  Alselí               DZ
5         ...           Sámua u Amelíka               AS

               cldr.variant.yo            cldr.variant.yo_bj  \
1        Orílède Àfùgànístánì        Orílède Àfùgànístánì
2                          AX                             AX
3          Orílède Àlùbàníánì          Orílède Àlùbàníánì
4          Orílède Àlùgèríánì          Orílède Àlùgèríánì
5  Sámóánì ti Orílède Àmérikà  Sámóánì ti Orílède Àmérikà

  cldr.variant.yue   cldr.variant.zgh  cldr.variant.zh cldr.variant.zh_hant  \
1
2                             AX
3
4
5

  cldr.variant.zh_hant_hk   cldr.variant.zu
1                        i-Afghanistan
2                    i-Åland Islands
3                         i-Albania
4                         i-Algeria
5                   i-American Samoa

[5 rows x 677 columns]
```

Each row in the resulting dataframe corresponds to a country. The dataframe has 677 columns, which means that for each country, there are 677 alternative names. Apparently it also contains more general information, such as 'continent' and 'ar5'. These should be filtered out at some point.

### 1.4 1.4 Database: Conversion to Dictionary

In order to efficiently convert alternative country names into the 'generic' country names used in this model, a dictionary will have to be generated. To this extent, we first convert the **country name database** to a *dictionary of dictionaries*:

```
In [4]: country_name_dic = country_name_database.to_dict('index')
```

Each country now has a dictionary of its own with all alternative names for that country:

```
In [5]: country_name_dic[167].values()
```

```
Out[5]: dict_values(['OECD1990', 'Europe', 'Niederlande', 'niederlande', 'Netherlands', '^(?!.
```

```
In [6]: len(country_name_dic)

Out[6]: 272

In [7]: len(country_name_dic[1])

Out[7]: 677
```

The *dictionary of dictionaries* now contains 272 countries with 677 alternative names. However, as mentioned earlier, some alternative names direct towards regions or organizations that countries are part of. These entries have been listed below, as the result of a *Trial and Error approach*.

```
In [8]: not_country_names = [
        'continent',
        'eurocontrol_pru',
        'eurocontrol_statfor',
        'region',
        'ar5',
        'icao_region',
        'eu28',
        # 'icao',
        # 'ecb',
        # 'fips',
        ]
```

In addition, a *generic* country name list should be selected to convert the country names to. We'll go with the country name in English:

```
In [9]: generic_name = 'country.name.en'
```

The code below converts the *dictionary of dictionaries* to a single dictionary that translates all alternative names to the *generic country name* given above. Whenever it finds a conflicting entry (that is, a name or abbrevation is already in the dictionary), it assesses whether the given alternative name should direct to multiple countries. If so, it adds it to the dictionary entry in the format "MANUAL: Country X or Country Y".

```
In [10]: complete_dic = {}
         counter = 0
         for key in country_name_dic.keys():
             country_dic = country_name_dic[key]
             for key2 in country_dic.keys():
                 if (type(country_dic[key2]) == str
                     and key2 not in not_country_names
                    ):
                     if country_dic[key2] in complete_dic and complete_dic[country_dic[key2]]!=

                         old_country = complete_dic[country_dic[key2]]
                         new_country = country_dic[generic_name]
                         counter += 1
```

```
                        if new_country in old_country:
                            counter -= 1
                            continue
                        elif 'MANUAL:' in old_country:
                            complete_dic[country_dic[key2]] = old_country + ' or ' + new_coun
                        else:
                            complete_dic[country_dic[key2]] = "MANUAL: "+ old_country + ' or
                        print(counter, country_dic[key2], complete_dic[country_dic[key2]], '(
                    else:
                        complete_dic[country_dic[key2]] =country_dic[generic_name]

                else:
                    continue
        print("There are", counter, "alternative country names with multiple options.")
        print("This is", round(counter/len(complete_dic)*100, 2), "%.")
        print("Size of database:", len(complete_dic), "entries")
```

```
1 AQ MANUAL: American Samoa or Antarctica (ecb)
2 AG MANUAL: Algeria or Antigua & Barbuda (ecb)
3 AS MANUAL: American Samoa or Australia (fips)
4 AUS MANUAL: Australia or Austria (cowc)
5 AU MANUAL: Australia or Austria (fips)
6 BAH MANUAL: Bahamas or Bahrain (cowc)
7 BH MANUAL: Bahrain or Belize (fips)
8 BD MANUAL: Bangladesh or Bermuda (fips)
9 BO MANUAL: Belarus or Bolivia (ecb)
10 TN MANUAL: Aruba or Caribbean Netherlands (icao)
11 BA MANUAL: Bahrain or Bosnia & Herzegovina (ecb)
12 BN MANUAL: Benin or Brunei (ecb)
13 BRN MANUAL: Bahrain or Brunei (genc3c)
14 BG MANUAL: Bangladesh or Bulgaria (ecb)
15 BF MANUAL: Bahamas or Burkina Faso (ecb)
16 BY MANUAL: Belarus or Burundi (fips)
17 CHI MANUAL: Channel Islands or Chile (ioc)
18 Y MANUAL: Australia or Christmas Island (icao)
19 Y MANUAL: Australia or Christmas Island or Cocos (Keeling) Islands (icao)
20 CN MANUAL: China or Comoros (fips)
21 CF MANUAL: Central African Republic or Congo - Brazzaville (fips)
22 CK MANUAL: Cocos (Keeling) Islands or Cook Islands (ecb)
23 CI MANUAL: Chile or Côte dIvoire (ecb)
24 CW MANUAL: Cook Islands or Curaçao (ecb)
25 TN MANUAL: Aruba or Caribbean Netherlands or Curaçao (icao)
26 CZE MANUAL: Czechoslovakia or Czechia (genc3c)
27 CD MANUAL: Chad or Congo - Kinshasa (ecb)
28 CG MANUAL: Congo - Brazzaville or Congo - Kinshasa (fips)
29  MANUAL: Congo - Brazzaville or Congo - Kinshasa (cldr.name.kk)
30 DA MANUAL: Algeria or Denmark (fips)
31 TD MANUAL: Chad or Dominica (icao)
```

```
32 DO MANUAL: Dominica or Dominican Republic (ecb)
33 EK MANUAL: Denmark or Equatorial Guinea (fips)
34 FK MANUAL: Cameroon or Falkland Islands (ecb)
35 EK MANUAL: Denmark or Equatorial Guinea or Faroe Islands (icao)
36 EF MANUAL: Åland Islands or Finland (icao)
37 FG MANUAL: Equatorial Guinea or French Guiana (fips)
38 FO MANUAL: Faroe Islands or Gabon (icao)
39 GA MANUAL: Gabon or Gambia (fips)
40 GB MANUAL: Gabon or Gambia (icao)
41 ET MANUAL: Ethiopia or German Democratic Republic (icao)
42 GM MANUAL: Gambia or Germany (fips)
43 BG MANUAL: Bangladesh or Bulgaria or Greenland (icao)
44 TF MANUAL: French Southern Territories or Guadeloupe (icao)
45 GQ MANUAL: Equatorial Guinea or Guam (fips)
46 GG MANUAL: Georgia or Guernsey (ecb)
47 EG MANUAL: Egypt or Guernsey (icao)
48 GV MANUAL: Cape Verde or Guinea (fips)
49 GU MANUAL: Guam or Guinea (icao)
50 GG MANUAL: Georgia or Guernsey or Guinea-Bissau (icao)
51 HA MANUAL: Ethiopia or Haiti (fips)
52 BI MANUAL: Burundi or Iceland (icao)
53 EG MANUAL: Egypt or Guernsey or Isle of Man (icao)
54 IS MANUAL: Iceland or Israel (fips)
55 EG MANUAL: Egypt or Guernsey or Isle of Man or Jersey (icao)
56 HK MANUAL: Hong Kong SAR China or Kenya (icao)
57 UA MANUAL: Kazakhstan or Kyrgyzstan (icao)
58 LA MANUAL: Albania or Laos (ecb)
59 LG MANUAL: Greece or Latvia (fips)
60 LB MANUAL: Bulgaria or Lebanon (ecb)
61 LI MANUAL: Italy or Liberia (fips)
62 GL MANUAL: Greenland or Liberia (icao)
63 LI MANUAL: Italy or Liberia or Liechtenstein (ecb)
64 LS MANUAL: Lesotho or Liechtenstein (fips)
65 LT MANUAL: Lesotho or Lithuania (ecb)
66 LH MANUAL: Hungary or Lithuania (fips)
67  MANUAL: Latvia or Lithuania (cldr.name.mzn)
68 EL MANUAL: Greece or Luxembourg (icao)
69 MG MANUAL: Guatemala or Madagascar (ecb)
70 FM MANUAL: Comoros or Madagascar (icao)
71 MW MANUAL: Cayman Islands or Malawi (ecb)
72 MY MANUAL: Bahamas or Malaysia (ecb)
73 MAD MANUAL: Madagascar or Maldives (cowc)
74 GA MANUAL: Gabon or Gambia or Mali (icao)
75 MT MANUAL: Haiti or Malta (ecb)
76 MH MANUAL: Honduras or Marshall Islands (ecb)
77 TF MANUAL: French Southern Territories or Guadeloupe or Martinique (icao)
78 MR MANUAL: Costa Rica or Mauritania (ecb)
79 GQ MANUAL: Equatorial Guinea or Guam or Mauritania (icao)
```

80 MAS MANUAL: Malaysia or Mauritius (cowc)
81 MU MANUAL: Cuba or Mauritius (ecb)
82 FI MANUAL: Finland or Mauritius (icao)
83 FM MANUAL: Comoros or Madagascar or Mayotte (icao)
84 FM MANUAL: Comoros or Madagascar or Mayotte or Micronesia (Federated States of) (ecb)
85 MC MANUAL: Macau SAR China or Monaco (ecb)
86 MON MANUAL: Monaco or Mongolia (cowc)
87 MN MANUAL: Monaco or Mongolia (ecb)
88 MG MANUAL: Guatemala or Madagascar or Mongolia (fips)
89 MNG MANUAL: Mongolia or Montenegro (cowc)
90 LY MANUAL: Libya or Montenegro (icao)
91 MS MANUAL: El Salvador or Montserrat (ecb)
92 MH MANUAL: Honduras or Marshall Islands or Montserrat (fips)
93 MA MANUAL: Madagascar or Morocco (ecb)
94 MO MANUAL: Macau SAR China or Morocco (fips)
95 GM MANUAL: Gambia or Germany or Morocco (icao)
96 MZ MANUAL: Belize or Mozambique (ecb)
97 MM MANUAL: Mexico or Myanmar (Burma) (ecb)
98 BM MANUAL: Bermuda or Myanmar (Burma) (fips)
99 AN MANUAL: Andorra or Nauru (icao)
100 AN MANUAL: Andorra or Nauru or Netherlands Antilles (ecb)
101 NT MANUAL: French Polynesia or Netherlands Antilles (fips)
102 TN MANUAL: Aruba or Caribbean Netherlands or Curaçao or Netherlands Antilles (icao)
103 NC MANUAL: Cook Islands or New Caledonia (ecb)
104 MN MANUAL: Monaco or Mongolia or Nicaragua (icao)
105 NG MANUAL: Kiribati or Niger (fips)
106 DR MANUAL: Dominican Republic or Niger (icao)
107 NIG MANUAL: Niger or Nigeria (cowc)
108 NG MANUAL: Kiribati or Niger or Nigeria (ecb)
109 NI MANUAL: Nicaragua or Nigeria (fips)
110 Nigeri MANUAL: Niger or Nigeria (cldr.name.sq)
111 NU MANUAL: Nicaragua or Niue (ecb)
112 NE MANUAL: Niger or Niue (fips)
113 NI MANUAL: Nicaragua or Nigeria or Niue (icao)
114 NF MANUAL: Fiji or Norfolk Island (ecb)
115 Y MANUAL: Australia or Christmas Island or Cocos (Keeling) Islands or Norfolk Island (icao)
116 MP MANUAL: Mauritius or Northern Mariana Islands (ecb)
117 PG MANUAL: Guam or Northern Mariana Islands (icao)
118 EN MANUAL: Estonia or Norway (icao)
119 MU MANUAL: Cuba or Mauritius or Oman (fips)
120 PK MANUAL: Marshall Islands or Pakistan (ecb)
121 PT MANUAL: Micronesia (Federated States of) or Palau (icao)
122 PS MANUAL: Palau or Palestinian Territories (ecb)
123 LV MANUAL: Latvia or Palestinian Territories (icao)
124 MP MANUAL: Mauritius or Northern Mariana Islands or Panama (icao)
125 PG MANUAL: Guam or Northern Mariana Islands or Papua New Guinea (ecb)
126 AY MANUAL: Antarctica or Papua New Guinea (icao)
127 PA MANUAL: Panama or Paraguay (fips)

```
128 PT MANUAL: Micronesia (Federated States of) or Palau or Portugal (ecb)
129 KR MANUAL: Kiribati or South Korea (ecb)
130 MD MANUAL: Dominican Republic or Moldova (ecb)
131 LU MANUAL: Luxembourg or Moldova (icao)
132 FM MANUAL: Comoros or Madagascar or Mayotte or Micronesia (Federated States of) or Réunion
133 LR MANUAL: Liberia or Romania (icao)
134 HR MANUAL: Croatia or Rwanda (icao)
135 BL MANUAL: Bolivia or St. Barthélemy (ecb)
136 TB MANUAL: Barbados or St. Barthélemy (fips)
137 TF MANUAL: French Southern Territories or Guadeloupe or Martinique or St. Barthélemy (icao)
138 KN MANUAL: North Korea or St. Kitts & Nevis (ecb)
139 LC MANUAL: Cyprus or St. Lucia (ecb)
140 MF MANUAL: Mayotte or Saint Martin (French part) (ecb)
141 TF MANUAL: French Southern Territories or Guadeloupe or Martinique or St. Barthélemy or Sa
142 PM MANUAL: Panama or St. Pierre & Miquelon (ecb)
143 LF MANUAL: France or St. Pierre & Miquelon (icao)
144 ST MANUAL: St. Lucia or São Tomé & Príncipe (ecb)
145 FP MANUAL: French Polynesia or São Tomé & Príncipe (icao)
146 SA MANUAL: Argentina or Saudi Arabia (ecb)
147 SG MANUAL: Paraguay or Senegal (fips)
148 RS MANUAL: Russia or Serbia (ecb)
149 LY MANUAL: Libya or Montenegro or Serbia (icao)
150 SC MANUAL: St. Kitts & Nevis or Seychelles (ecb)
151 SE MANUAL: Ecuador or Seychelles (fips)
152 FS MANUAL: French Southern Territories or Seychelles (icao)
153 SL MANUAL: Bolivia or Sierra Leone (ecb)
154 GF MANUAL: French Guiana or Sierra Leone (icao)
155 SG MANUAL: Paraguay or Senegal or Singapore (ecb)
156 SN MANUAL: Senegal or Singapore (fips)
157 WS MANUAL: Samoa or Singapore (icao)
158 TN MANUAL: Aruba or Caribbean Netherlands or Curaçao or Netherlands Antilles or Sint Maarte
159 SK MANUAL: Colombia or Slovakia (ecb)
160 LO MANUAL: Austria or Slovakia (fips)
161 SLV MANUAL: El Salvador or Slovenia (cowc)
162 SLO MANUAL: Slovakia or Slovenia (ioc)
163 SB MANUAL: St. Pierre & Miquelon or Solomon Islands (ecb)
164 AG MANUAL: Algeria or Antigua & Barbuda or Solomon Islands (icao)
165 SO MANUAL: French Guiana or Somalia (ecb)
166 SF MANUAL: Falkland Islands or South Africa (fips)
167 SX MANUAL: Sint Maarten or South Georgia & South Sandwich Islands (fips)
168 ES MANUAL: El Salvador or Spain (ecb)
169 SP MANUAL: Peru or Spain (fips)
170 LE MANUAL: Lebanon or Spain (icao)
171 LK MANUAL: Czechia or Sri Lanka (ecb)
172 VC MANUAL: St. Vincent & Grenadines or Sri Lanka (icao)
173 NS MANUAL: American Samoa or Suriname (fips)
174 SM MANUAL: San Marino or Suriname (icao)
175 SV MANUAL: El Salvador or Svalbard & Jan Mayen (fips)
```

```
176 EN MANUAL: Estonia or Norway or Svalbard & Jan Mayen (icao)
177 SE MANUAL: Ecuador or Seychelles or Sweden (ecb)
178 ES MANUAL: El Salvador or Spain or Sweden (icao)
179 SWZ MANUAL: Swaziland or Switzerland (cowc)
180 CH MANUAL: China or Switzerland (ecb)
181 SZ MANUAL: Swaziland or Switzerland (fips)
182 LS MANUAL: Lesotho or Liechtenstein or Switzerland (icao)
183 SY MANUAL: Guyana or Syria (ecb)
184 TJ MANUAL: Puerto Rico or Tajikistan (ecb)
185 VT MANUAL: Vatican City or Thailand (icao)
186 MAC MANUAL: Macau SAR China or Macedonia (cowc)
187 MK MANUAL: Jamaica or Macedonia (ecb)
188 TL MANUAL: St. Lucia or Timor-Leste (ecb)
189 TG MANUAL: Grenada or Togo (ecb)
190 TK MANUAL: St. Kitts & Nevis or Tokelau (ecb)
191 TL MANUAL: St. Lucia or Timor-Leste or Tokelau (fips)
192 TO MANUAL: Togo or Tonga (ecb)
193 TN MANUAL: Aruba or Caribbean Netherlands or Curaçao or Netherlands Antilles or Sint Maarte
194 NF MANUAL: Fiji or Norfolk Island or Tonga (icao)
195 TT MANUAL: Timor-Leste or Trinidad & Tobago (ecb)
196 TD MANUAL: Chad or Dominica or Trinidad & Tobago (fips)
197 TN MANUAL: Aruba or Caribbean Netherlands or Curaçao or Netherlands Antilles or Sint Maarte
198 TR MANUAL: Montserrat or Turkey (ecb)
199 LT MANUAL: Lesotho or Lithuania or Turkey (icao)
200 TX MANUAL: Bermuda or Turkmenistan (fips)
201 UT MANUAL: Tajikistan or Turkmenistan (icao)
202 TK MANUAL: St. Kitts & Nevis or Tokelau or Turks & Caicos Islands (fips)
203 MB MANUAL: Martinique or Turks & Caicos Islands (icao)
204 TV MANUAL: St. Vincent & Grenadines or Tuvalu (ecb)
205 NG MANUAL: Kiribati or Niger or Nigeria or Tuvalu (icao)
206 UG MANUAL: Georgia or Uganda (ecb)
207 HU MANUAL: Hungary or Uganda (icao)
208 UA MANUAL: Kazakhstan or Kyrgyzstan or Ukraine (ecb)
209 OM MANUAL: Oman or United Arab Emirates (icao)
210 GB MANUAL: Gabon or Gambia or United Kingdom (ecb)
211 UK MANUAL: Ukraine or United Kingdom (eurostat)
212 EG MANUAL: Egypt or Guernsey or Isle of Man or Jersey or United Kingdom (icao)
213 BK MANUAL: Bosnia & Herzegovina or United Kingdom (cldr.short.az)
214 RU MANUAL: Russia or United Kingdom (cldr.short.br)
215 EB MANUAL: Belgium or United Kingdom (cldr.short.eu)
216 ZK MANUAL: North Korea or United Kingdom (cldr.short.sl)
217 OK MANUAL: Kuwait or United Kingdom (cldr.short.smn)
218 MB MANUAL: Martinique or Turks & Caicos Islands or United Kingdom (cldr.short.sq)
219 HT MANUAL: Haiti or Tanzania (icao)
220 SU MANUAL: Sudan or United States (cldr.short.br)
221 SAM MANUAL: Samoa or United States (cldr.short.ga)
222 SA MANUAL: Argentina or Saudi Arabia or United States (cldr.short.gd)
223 UM MANUAL: Belarus or United States Minor Outlying Islands (the) (ecb)
```

```
224 SU MANUAL: Sudan or United States or Uruguay (icao)
225 UT MANUAL: Tajikistan or Turkmenistan or Uzbekistan (icao)
226 SV MANUAL: El Salvador or Svalbard & Jan Mayen or Venezuela (icao)
227 VN MANUAL: Nepal or Vietnam (ecb)
228 VM MANUAL: Macau SAR China or Vietnam (fips)
229 VG MANUAL: Bangladesh or British Virgin Islands (ecb)
230 TU MANUAL: Turkey or British Virgin Islands (icao)
231 VI MANUAL: British Virgin Islands or U.S. Virgin Islands (ecb)
232 VQ MANUAL: Bhutan or U.S. Virgin Islands (fips)
233 TI MANUAL: Tajikistan or U.S. Virgin Islands (icao)
234 NL MANUAL: Netherlands or Wallis & Futuna (icao)
235 EH MANUAL: Netherlands or Western Sahara (ecb)
236 GS MANUAL: South Georgia & South Sandwich Islands or Western Sahara (icao)
237 ZM MANUAL: Mongolia or Zambia (ecb)
238 ZA MANUAL: South Africa or Zambia (fips)
There are 238 alternative country names with multiple options.
This is 0.87 %.
Size of database: 27339 entries
```

The database now contains over 27,000 alternative country names! In addition, it is clear that the conflicts mainly occur for two-letter and three-letter abbrevations. These will have to be assessed manually.

Below is a list of manually added country conversions, that were not in the original database, but proved to be necessary later on in the data preparation process.

```
In [11]: complete_dic['Libya']

Out[11]: 'Libya'

In [12]: complete_dic['DRV'] = 'Vietnam' # CoW
         complete_dic["CÃ‰te d'Ivoire"]='Côte dIvoire' # WB
         complete_dic["Korea, Dem. Peoples Rep."] = 'North Korea' # WB
         complete_dic["Antigua And Barbuda"] = "Antigua & Barbuda" # CIA
         complete_dic["Bosnia And Herzegovina"] = "Bosnia & Herzegovina" # CIA
         complete_dic["Cote D'Ivoire"] = "Côte dIvoire" # CIA
         complete_dic["Congo, Republic Of The"] = "Congo - Brazzaville" # CIA
         complete_dic["Congo, Democratic Republic Of The"] = "Congo - Kinshasa" # CIA
         complete_dic["Micronesia, Federated States Of"] = "Micronesia (Federated States of)"
         complete_dic["Saint Kitts And Nevis"] = "St. Kitts & Nevis" # CIA
         complete_dic["Saint Vincent And The Grenadines"] = "St. Vincent & Grenadines" # CIA
         complete_dic["Sao Tome And Principe"] = "São Tomé & Príncipe" # CIA
         complete_dic["Trinidad And Tobago"] = "Trinidad & Tobago" # CIA
         complete_dic["Holy See (Vatican City)"] = "Vatican City" # CIA
         complete_dic["Isle Of Man"] = "Isle of Man" # CIA
         complete_dic["Turks And Caicos Islands"] = "Turks & Caicos Islands" # CIA
         complete_dic["Saint Helena, Ascension, And Tristan Da Cunha"] = "St. Helena" # CIA
         complete_dic["Saint Martin"] = "Sint Maarten" # CIA
         complete_dic["Wallis And Futuna"] = "Wallis & Futuna" # CIA
```

```
complete_dic["Congo Democratic Republic"] = "Congo - Kinshasa" # FSI
complete_dic["Congo Republic"] = "Congo - Brazzaville" # FSI
complete_dic["Israel and West Bank"] = "Israel" # FSI
complete_dic["Micronesia"] = "Micronesia (Federated States of)" # FSI

complete_dic["Congo (Kinshasa)"] = "Congo - Kinshasa" # FSI
complete_dic["Congo (Brazzaville)"] = "Congo - Brazzaville" # FSI
complete_dic["The Gambia"] = 'Gambia' # FSI

complete_dic["Congo, DRC"] = 'Congo - Kinshasa' # Road Crossings
complete_dic["West Bank"] = "Palestinian Territories" # Road Crossings
complete_dic["Gaza Strip"] = "Palestinian Territories" # Road Crossings

complete_dic['Libyan Arab Jamahiriya'] = 'Libya' # World Risk Index

complete_dic['The Bahamas'] = 'Bahamas'
complete_dic['French Southern and Antarctic Lands'] = 'French Southern Territories'
complete_dic['Republic of the Congo'] = "Congo - Brazzaville"
complete_dic['Republic of Serbia'] = 'Serbia'

complete_dic["Cote d Ivoire"] = "Côte dIvoire"
complete_dic["Congo Republic"] = "Congo - Brazzaville"
complete_dic["Congo Democratic Republic"] = "Congo - Kinshasa"
complete_dic["Micronesia"] = "Micronesia (Federated States of)"
complete_dic['Bahamas The'] = 'Bahamas'
complete_dic['Macedonia FYR'] = 'Macedonia'
complete_dic['Korea Republic'] = 'South Korea'
complete_dic['Korea DPR'] = 'North Korea'
complete_dic['Macao SAR China'] = 'Macao SAR China'
complete_dic['St Lucia'] = 'St. Lucia'
complete_dic['St Kitts and Nevis'] = 'St. Kitts & Nevis'
# complete_dic[""] = ""
```

A significant number of conflicting entries involved the *Correlates of War* abbrevations. Since this is a key dataset in this study, it is probably worthwile to create a dictionary specifically for this dataset.

```
In [13]: cow_entries = [
             'cow.name',
             'cowc',
             'cown'
              ]

In [14]: cow_dic = {}
         for key in country_name_dic.keys():
             country_dic = country_name_dic[key]
             for key2 in country_dic.keys():
                 if (type(country_dic[key2]) == str
```

```
                    and (key2 in cow_entries
                   )):
                    if country_dic[key2] in cow_dic and cow_dic[country_dic[key2]]!=country_di
                        print(country_dic[key2], 'is already in the dictionary.')
                        print(key, key2, cow_dic[country_dic[key2]], country_dic[key2])
                    cow_dic[country_dic[key2]]=country_dic['country.name.en']
                else:
                    continue
```

There is one abbrevation missing in this dictionary, so we add *Vietnam* manually:

```
In [15]: cow_dic['DRV'] = 'Vietnam'

In [16]: iso2_entry = ['iso2c']

In [17]: iso2_dic = {}
         for key in country_name_dic.keys():
             country_dic = country_name_dic[key]
             for key2 in country_dic.keys():
                 if (type(country_dic[key2]) == str
                     and (key2 in iso2_entry
                    )):
                     if country_dic[key2] in iso2_dic and iso2_dic[country_dic[key2]]!=country_
                         print(country_dic[key2], 'is already in the dictionary.')
                         print(key, key2, iso2_dic[country_dic[key2]], country_dic[key2])
                     iso2_dic[country_dic[key2]]=country_dic['country.name.en']
                 else:
                     continue
```

### 1.5 1.5 Saving the Dictionaries to CSV

Now all that's left is to save the dictionaries to csv files for later use.

```
In [18]: with open('processed_data/cow_name_dic.csv', 'w') as output_file:
             fieldnames = ['cow', 'name']
             dict_writer = csv.DictWriter(output_file, fieldnames)
             dict_writer.writeheader()
             data = [dict(zip(fieldnames, [k, v])) for k, v in cow_dic.items()]
             dict_writer.writerows(data)

In [19]: with open('processed_data/country_name_dic.csv', 'w') as output_file:
             fieldnames = ['alt_name', 'name']
             dict_writer = csv.DictWriter(output_file, fieldnames)
             dict_writer.writeheader()
             data = [dict(zip(fieldnames, [k, v])) for k, v in complete_dic.items()]
             dict_writer.writerows(data)

In [20]: with open('processed_data/iso2_name_dic.csv', 'w') as output_file:
             fieldnames = ['iso2code', 'name']
```

```
            dict_writer = csv.DictWriter(output_file, fieldnames)
            dict_writer.writeheader()
            data = [dict(zip(fieldnames, [k, v])) for k, v in iso2_dic.items()]
            dict_writer.writerows(data)
```

```
In [21]: energy = pd.read_excel("energydata-2.xlsx")
```

```
In [22]: energy = energy.set_index('country')
```

```
In [23]: for country in energy.index:
            if country not in complete_dic.keys():
                print(country)
```

```
St Martin
Virgin Islands
```

## D.2. Country Selection

# 02 - Country Selection

August 15, 2018

## 1 Notebook 2 - Country Selection

*@author: Stefan Wigman*

### 1.1 2.1 - Introduction

This notebook is used to generate an initial list of countries that will be taken into account in this study. The available countries in the major datasets that are used are compared and assessed to select an initial set of countries.

In order to efficiently compare available countries in different datasets, the 'country name database' that was generated in Notebook 1 is employed.

### 1.2 2.2 - Importing the required Python packages

This notebook uses Pandas, NumPy, and csv.

```
In [1]: import pandas as pd
        import numpy as np
        import csv
```

### 1.3 2.3 - Importing the country name database

We start by importing the previously generated *alternative country name* database.

```
In [2]: with open('processed_data/country_name_dic.csv') as csvfile:
            reader = csv.DictReader(csvfile, delimiter = ',')
            country_name_dict = {rows['alt_name']:rows['name'] for rows in reader}
```

For the *Correlates of War dataset*, we import the dictionary separately.

```
In [3]: with open('processed_data/cow_codes.csv') as csvfile:
            reader = csv.DictReader(csvfile, delimiter = ',')
            cow_dict = {rows['cow']:rows['name'] for rows in reader}
```

### 1.4 2.4 - United Nations: Members of General Assembly

As an initial list of countries, we take all members of the *General Assembly of the United Nations*.

This data was acquired from http://data.un.org/Data.aspx?q=membership&d=SDGs&f=series%3aSG_INT_M

```
In [4]: un_data = pd.read_csv("raw data/UNdata_MembershipOfGeneralAssembly.txt", sep = ';')

In [5]: un_data.head()

Out[5]:              Reference Area  Time Period Unit of measurement  \
        0  Developing regions (MDG)         2016             Percent
        1  Developing regions (MDG)         2015             Percent
        2  Developing regions (MDG)         2010             Percent
        3  Developing regions (MDG)         2005             Percent
        4  Developing regions (MDG)         2000             Percent

                          Nature      Value
        0  Global monitoring data  74.093264
        1  Global monitoring data  74.093264
        2  Global monitoring data  73.958333
        3  Global monitoring data  74.345550
        4  Global monitoring data  74.603175
```

This dataset also contains the proportion of developing countries currently in the general UN assembly. This entry will have to be removed manually. A full list of unique countries in this dataset is shown below:

```
In [6]: pd.DataFrame(un_data['Reference Area'].unique())

Out[6]:                                                  0
        0                          Developing regions (MDG)
        1                                       Afghanistan
        2                                           Albania
        3                                           Algeria
        4                                           Andorra
        5                                            Angola
        6                                Antigua and Barbuda
        7                                         Argentina
        8                                           Armenia
        9                                         Australia
        10                                          Austria
        11                                       Azerbaijan
        12                                          Bahamas
        13                                          Bahrain
        14                                       Bangladesh
        15                                         Barbados
        16                                          Belarus
        17                                          Belgium
        18                                           Belize
```

```
19                                          Benin
20                                         Bhutan
21           Bolivia (Plurinational State of)
22                     Bosnia and Herzegovina
23                                       Botswana
24                                         Brazil
25                                 Brunei Darussalam
26                                       Bulgaria
27                                   Burkina Faso
28                                        Burundi
29                                  CÃ‡te d'Ivoire
..                                            ...
164                                     Suriname
165                                    Swaziland
166                                       Sweden
167                                  Switzerland
168                           Syrian Arab Republic
169                                    Tajikistan
170                                     Thailand
171    The former Yugoslav Republic of Macedonia
172                                  Timor-Leste
173                                         Togo
174                                        Tonga
175                          Trinidad and Tobago
176                                      Tunisia
177                                       Turkey
178                                  Turkmenistan
179                                       Tuvalu
180                                       Uganda
181                                      Ukraine
182                          United Arab Emirates
183  United Kingdom of Great Britain and Northern I...
184                  United Republic of Tanzania
185                     United States of America
186                                      Uruguay
187                                   Uzbekistan
188                                      Vanuatu
189           Venezuela (Bolivarian Republic of)
190                                    Viet Nam
191                                       Yemen
192                                      Zambia
193                                    Zimbabwe

[194 rows x 1 columns]
```

For comparison with other data sets, the list of countries is extracted from this Data Frame. In addition, the *'Developing regions (MDG)* is removed.

```
In [7]: un_data=un_data.set_index('Reference Area')
```

```
In [8]: un_country_list = list(un_data.index.unique())

In [9]: un_country_list.remove('Developing regions (MDG)')
```

The variable *un_country_list* is now a list of all countries that are a member of the *General Assembly of the United Nations*. According to Wikipedia (https://en.wikipedia.org/wiki/United_Nations_General_Assembly) the UN has 193 members. Therefore, the length of the list should be equal to 193.

```
In [10]: if len(un_country_list) == 193:
             print('\x1b[1;31m'+"OK!"+'\x1b[0m'+" There are 193 countries in the list. You may
         else:
             print('\x1b[1;31m'+"Uh oh!"+'\x1b[0m'+ "Something is not right. The list does not

OK! There are 193 countries in the list. You may proceed.
```

Another check that needs to be executed is whether all countries in *un_country_list* are also in the **Country Name Database**.

```
In [11]: counter = 0
         for item in un_country_list:              # If this cell does not create output,
             if item not in country_name_dict:     # all countries are accounted for in the cou
                 print(item)                       # All output should be added to the country
                 counter += 1
         if counter == 0:
             print('\x1b[1;31m'+"OK!"+'\x1b[0m'+ " All (alternative) country names are accounte
         else:
             print('\x1b[1;31m'+"Uh oh!"+'\x1b[0m'+ "Some countries are not in the "+'\x1b[1;3

OK! All (alternative) country names are accounted for in the Country Name Database.
```

This concludes the checks for the *United Nations General Assembly* country list. For comparison with other datasets, we now convert the *UN names* to the *English country names*.

```
In [12]: un_country_list_english = [ country_name_dict.get(item,item) for item in un_country_li
```

### 1.5 2.5 - Correlates of War: Contiguity Data

The second source of which the included countries will be assessed is the *Correlates of War Direct Contiguity dataset (v3.2)*. The use of this dataset is highly desirable in this study, given its detailed character regarding country borders. It classifies country borders on a scale from 0 to 5, where a 0 means *no border*, a 1 means *direct border via land*, and 2 to 5 are used for *water contiguity up to a certain distance* (400 miles).

Countries that are not included in this dataset, will have to be added manually (or the contiguity of these countries will have to be added using another method (i.e. from a shapefile).

The dataset can be found here: http://www.correlatesofwar.org/data-sets/direct-contiguity.

The first step is to import the dataset using Pandas:

```
In [13]: contiguity_df=pd.read_csv('raw data/correlates_of_war/DirectContiguity320/contdird.csv
```

```
In [14]: contiguity_df.head()
```

Out[14]:

| | dyad | state1no | state1ab | state2no | state2ab | year | conttype | version |
|---|---|---|---|---|---|---|---|---|
| 0 | 2020 | 2 | USA | 20 | CAN | 1920 | 1 | 3.2 |
| 1 | 2020 | 2 | USA | 20 | CAN | 1921 | 1 | 3.2 |
| 2 | 2020 | 2 | USA | 20 | CAN | 1922 | 1 | 3.2 |
| 3 | 2020 | 2 | USA | 20 | CAN | 1923 | 1 | 3.2 |
| 4 | 2020 | 2 | USA | 20 | CAN | 1924 | 1 | 3.2 |

For each existing border, there is a separate row in the dataset. There is a *from* country and a *to* country, respectively in columns *state1ab* and *state2ab*. Let's first check if the countries contained in these two columns are the same, and how many countries are represented in this dataset.

```
In [15]: if list(sorted(contiguity_df.state1ab.unique()))==list(sorted(contiguity_df.state2ab.u
             print('\x1b[1;31m'+"OK!"+'\x1b[0m'+ " The countries in columns state1ab and state2
         else:
             print('\x1b[1;31m'+"Uh oh!"+'\x1b[0m'+ " The countries in columns state1ab and sta
```

```
OK! The countries in columns state1ab and state2ab match. Please continue.
```

```
In [16]: print("This dataset contains", len(contiguity_df.state2ab.unique()), "unique countries
```

```
This dataset contains 215 unique countries.
```

Apparently, there are 215 countries countries represented in this dataset. However, it should be taken into account that this data is given over time. Therefore, there might be *"old"* countries in the dataset. Let's first find out for which years we have data.

```
In [17]: print("This dataset contains data for the following years:", sorted(contiguity_df.year
```

```
This dataset contains data for the following years: [1816, 1817, 1818, 1819, 1820, 1821, 1822,
```

At this point, we are only interested in the latest contiguity data. Therefore, we select only the most recent year (2016):

```
In [18]: contiguity_df_2016 = contiguity_df[contiguity_df.year==2016]
```

Let's see how many countries are left:

```
In [19]: print("The dataset contains", len(contiguity_df_2016.state2ab.unique()), "unique coun
```

```
The dataset contains 193 unique countries for 2016.
```

193 countries seems promising. In fact, it is exactly the same number of countries as in the *UN General Assembly* data set. The countries in both data sets will be compared in section **2.7**. First, the country names will have to be converted to their *English name*. To that extent, we create a list containing all the countries in the **Correlates of War data set**.

```
In [20]: cow_country_list = list(contiguity_df_2016.state1ab.unique())
```

Then we run the check to see whether all countries in the **Correlates of War data set** are accounted for in the **Country Name Database**:

```
In [21]: counter = 0
         for item in cow_country_list:            # If this cell does not create output,
             if item not in cow_dict:             # all countries are accounted for in the cow
                 print(item)                      # All output should be added to the cow dict
                 counter += 1
         if counter == 0:
             print('\x1b[1;31m'+"OK!"+'\x1b[0m'+ " All (alternative) country names are accounte
         else:
             print('\x1b[1;31m'+"Uh oh!"+'\x1b[0m'+ "Some countries are not in the "+'\x1b[1;3
```

```
OK! All (alternative) country names are accounted for in the Country Name Database.
```

This concludes the checks for the *Correlates of War* country list. For comparison with other datasets, we now convert the *Correlates of War names* to the corresponding *English country names*.

```
In [22]: cow_country_list_english =  [cow_dict.get(item,item) for item in cow_country_list]
```

### 1.6 2.6 - World Bank Data (World Development Indicators)

The last data set that will be taken into account for the selection of countries are the *World Development Indicators (WDI)* of the World Bank. This is a very extensive data set (both in number of indicators *and* in number of countries for which these indicators are available), and therefore it lends itself exellently to assess which countries are absent in the other two data sets.

A list of countries for which the *WDI* are available is contained in the Excel file *"Regions.xlsx"*, so the first step is to import that file to a Pandas Data Frame:

```
In [23]: countries_wb=pd.read_excel("raw data/Regions.xlsx", sheetname='Countries', header=0, s
                                   names=['Country Data','Region','IncomeGroup'],
                                     parse_cols=None, parse_dates=False, date_parser=None, na_valu
```

```
In [24]: countries_wb.head()
```

Out[24]:

| country | Country Data | Region | IncomeGroup |
|---|---|---|---|
| Afghanistan | AFG | South Asia | Low income |
| Albania | ALB | Europe & Central Asia | Upper middle income |
| Algeria | DZA | Middle East & North Africa | Upper middle income |
| American Samoa | ASM | East Asia & Pacific | Upper middle income |
| Andorra | AND | Europe & Central Asia | High income |

The countries for which *WDI* data is available are in the *country* column. The number of countries is shown below:

```
In [25]: print("The World Development Indicators data set contains", len(countries_wb.index.uni

The World Development Indicators data set contains 217 unique countries.
```

Apparently, the *WDI* data set is a bit more comprehensive and includes more countries than the previous two data sets. We will assess the difference in available countries in section **2.7 Comparison of Different Datasets**. For now we create a list containing all unique countries contained in the data set.

```
In [26]: wb_country_list = list(countries_wb.index.unique())
```

Next, we run the fast check to see whether all *World Bank* names are accounted for in the **Country Name Database**:

```
In [27]: counter = 0
         for item in wb_country_list:                # If this cell does not create output,
             if item not in country_name_dict:       # all countries are accounted for in the cou
                 print(item)                         # All output should be added to the country
                 counter += 1
         if counter == 0:
             print('\x1b[1;31m'+"OK!"+'\x1b[0m'+ " All (alternative) country names are accounte
         else:
             print('\x1b[1;31m'+"Uh oh!"+'\x1b[0m'+ "Some countries are not in the "+'\x1b[1;3

OK! All (alternative) country names are accounted for in the Country Name Database.
```

This concludes the checks for the *World Development Indicators* country list. For comparison with the other datasets, we now convert the *World Bank country names* to the corresponding *English country names*.

```
In [28]: wb_country_list_english = [ country_name_dict.get(item,item) for item in wb_country_li
```

## 1.7 2.7 Comparison of Countries in Different Datasets

All country entries should now have the same name throughout the lists, which makes it possible to compare the lists.

The countries that are present in the **Correlates of War dataset**, but are *not* a member of the **United Nations General Assembly**:

```
In [29]: counter = 0
         for item in cow_country_list_english:
             if item not in un_country_list_english:
                 counter += 1
                 print(counter, '\x1b[1;31m'+item+'\x1b[0m')
         print("Number of countries in CoW, but not in UN:", '\x1b[1;31m'+str(counter)+'\x1b[0
```

```
1 Yugoslavia
2 Kosovo
3 Taiwan
Number of countries in CoW, but not in UN: 3
```

The countries that are a member of the **United Nations General Assembly**, but are *not* in the **Correlates of War dataset**:

```
In [30]: counter = 0
         for item in un_country_list_english:
             if item not in cow_country_list_english:
                 counter += 1
                 print(counter, '\x1b[1;31m'+item+'\x1b[0m')
         print("Number of countries in UN, but not in CoW:", '\x1b[1;31m'+str(counter)+'\x1b[0
```

```
1 Iceland
2 New Zealand
3 Serbia
Number of countries in UN, but not in CoW: 3
```

Apparently, three countries are not included in the *Correlates of War* data set: *Iceland, New Zealand, and Serbia*. However, these are countries of substantial size and importance and cannot be omitted in this study.

As we saw previously, Yugoslavia *is* in the *Correlates of War* data set, and this could well be the name used for Serbia in this data set. Iceland and New Zealand are both considerably isolated in the middle of the ocean. As such, they might not be contiguous to any other countries in the data set. These issues will be further adressed in Notebook *03a - Contiguity and Adjacency, Correlates of War*.

The countries that are present in the **World Bank dataset**, but are *not* a member of the **United Nations General Assembly**:

```
In [31]: counter = 0
         for item in wb_country_list_english:
             if item not in un_country_list_english:
                 counter += 1
                 print(counter, '\x1b[1;31m'+item+'\x1b[0m')
         print("Number of countries in WB, but not in UN:", '\x1b[1;31m'+str(counter)+'\x1b[0m
```

```
1 American Samoa
2 Aruba
3 Bermuda
4 British Virgin Islands
5 Cayman Islands
6 Channel Islands
7 Curaçao
8 Faroe Islands
9 French Polynesia
```

```
10 Gibraltar
11 Greenland
12 Guam
13 Hong Kong SAR China
14 Isle of Man
15 Kosovo
16 Macau SAR China
17 New Caledonia
18 Northern Mariana Islands
19 Puerto Rico
20 Sint Maarten
21 Saint Martin (French part)
22 Turks & Caicos Islands
23 U.S. Virgin Islands
24 Palestinian Territories
Number of countries in WB, but not in UN: 24
```

Vice versa, the countries that are members of the **United Nations General Assembly**, but are *not* in the **World Bank dataset**:

```
In [32]: counter = 0
         for item in un_country_list_english:
             if item not in wb_country_list_english:
                 counter += 1
                 print(counter, '\x1b[1;31m'+item+'\x1b[0m')
         print("Number of countries in UN, but not in WB:", '\x1b[1;31m'+str(counter)+'\x1b[0m
```

```
Number of countries in UN, but not in WB: 0
```

### 1.8 2.8 - Country Selection

This section will discuss the country selection process.

The criteria for country selection are: - Data availability - (Additional) Complexity - Model Completeness

```
In [33]: country_list_final = un_country_list_english

In [34]: countries = pd.DataFrame(country_list_final)

In [35]: countries=countries.set_index(0)

In [36]: countries.index.rename('country', inplace=True)

In [37]: writer = pd.ExcelWriter('processed_data/country_list.xlsx')
         countries.to_excel(writer,'country')
         writer.save()
```

## D.3. Contiguity and Adjacency

# 03a - Contiguity and Adjacency, Correlates of War

August 15, 2018

## 1 Notebook 3a - Contiguity and Adjacency, Correlates of War

*@author: Stefan Wigman*

### 1.1 3a.1 - Introduction

### 1.2 3a.2 - Importing the required Python packages

```
In [1]: import pandas as pd
        import numpy as np
        import csv
        import os

In [2]: os.getcwd()

Out[2]: '/Users/stefanwigman/Desktop/Migration Thesis/03 - Data'

In [3]: contiguity_df=pd.read_csv('raw data/correlates_of_war/DirectContiguity320/contdird.csv

In [4]: contiguity_df_2016=contiguity_df[contiguity_df['year']==2016]

In [5]: set(contiguity_df_2016['state1ab'].unique())==set(contiguity_df_2016['state2ab'].uniqu

Out[5]: True

In [6]: contiguity_df_2016.loc[96]

Out[6]: dyad        2020
        state1no       2
        state1ab     USA
        state2no      20
        state2ab     CAN
        year        2016
        conttype       1
        version      3.2
        Name: 96, dtype: object

In [7]: contiguity_matrix=pd.DataFrame(columns=set(contiguity_df_2016['state1ab']), index=set(
```

```
In [8]: for row in contiguity_df_2016.index:
        #     print(contiguity_df_2016.loc[row]['state1ab'])
            contiguity_matrix[contiguity_df_2016.loc[row]['state1ab']][contiguity_df_2016.loc[

In [9]: filled_matrix = contiguity_matrix.fillna(0)

In [11]: filled_matrix['YUG']['DEN']

Out[11]: 0

In [12]: countries=pd.read_excel('processed_data/country_list.xlsx', index_col = 'country')

In [13]: with open('processed_data/cow_codes.csv') as csvfile:
             reader = csv.DictReader(csvfile, delimiter = ',')
             cow_dict = {rows['cow']:rows['name'] for rows in reader}

In [14]: filled_matrix = filled_matrix.rename(columns=cow_dict, index=cow_dict)

In [15]: filled_matrix = filled_matrix.rename(columns = {"Yugoslavia":"Serbia"}, index = {"Yug

In [16]: filled_matrix[filled_matrix['Serbia']!=0].index.values.astype(str)

Out[16]: array(['Bulgaria', 'Hungary', 'Montenegro', 'Macedonia',
                'Bosnia & Herzegovina', 'Romania', 'Croatia', 'Kosovo'],
               dtype='<U20')

In [17]: filled_matrix['Iceland']=0
         filled_matrix['New Zealand']=0
         filled_matrix.loc['Iceland']=0
         filled_matrix.loc['New Zealand']=0

In [18]: filled_matrix.drop(labels=['Kosovo', 'Taiwan'], inplace=True)

In [19]: filled_matrix.drop(['Kosovo', 'Taiwan'], 1, inplace=True)

In [20]: filled_matrix['New Zealand']['Australia']=1/5
         filled_matrix['Australia']['New Zealand']=1/5
         filled_matrix['Iceland']['United Kingdom']=1/5
         filled_matrix['United Kingdom']['Iceland']=1/5

In [21]: writer = pd.ExcelWriter('contiguity_data.xlsx')
         # writer.define_name('countries', '=Sheet1!$A$2:$A$194')
         workbook=writer.book
         workbook.define_name('countries', '=Sheet1!$A$2:$A$194')
         filled_matrix.to_excel(writer)
         writer.save()

In [22]: for country in filled_matrix.index:
             if country not in countries.index:
                 print(country)
```

```
In [23]: filled_matrix = filled_matrix.reindex_axis(countries.index, axis=0)
         filled_matrix = filled_matrix.reindex_axis(countries.index, axis=1)

In [26]: replace_value_dic={5:1/5,
                            4:1/4,
                            3:1/3,
                            2:1/2}

In [27]: filled_matrix.replace(to_replace=replace_value_dic, value=None, inplace=True)

In [56]: # filled_matrix

In [29]: writer = pd.ExcelWriter('processed_data/contiguity_data.xlsx')
         # writer.define_name('countries', '=Sheet1!$A$2:$A$194')
         workbook=writer.book
         workbook.define_name('countries', '=Sheet1!$A$2:$A$194')
         filled_matrix.to_excel(writer)
         writer.save()

In [30]: countries_wb=pd.read_excel("raw data/Regions.xlsx", sheetname='Countries', header=0, s
                                names=['Country Data','Region','IncomeGroup'],
                                  parse_cols=None, parse_dates=False, date_parser=None, na_valu

In [31]: with open('processed_data/country_name_dic.csv', encoding='UTF-8') as csvfile:
             reader = csv.DictReader(csvfile, delimiter = ',')
             country_name_dict = {rows['alt_name']:rows['name'] for rows in reader}

In [32]: countries_wb_sort = countries_wb.sort_values(['Region', 'Country Data'], axis=0)
         countries_wb_sort = countries_wb_sort.rename(index=country_name_dict)

In [33]: wb_contiguity_df = pd.DataFrame(index=countries_wb_sort.index, columns=countries_wb_s

In [36]: for country in filled_matrix.index:
             if country not in wb_contiguity_df.index:
                 print(country)

In [37]: filled_matrix = filled_matrix.rename(index={'CÃ‰te dâIvoire': 'Côte dIvoire',
                                                     'SÃ£o TomÃ‰ & PrÃŋncipe': 'São Tomé & Prín

In [57]: # for country in wb_contiguity_df.index:
         #     if country not in filled_matrix.index:
         #         print(country)

In [40]: for column in wb_contiguity_df.columns:
             try:
                 wb_contiguity_df[column]=filled_matrix[column]
             except KeyError:
                 continue

In [58]: # wb_contiguity_df
```

In [42]:
```
# American Samoa
wb_contiguity_df['American Samoa'] = wb_contiguity_df['Samoa']
wb_contiguity_df.loc['American Samoa'] = wb_contiguity_df.loc['Samoa']

wb_contiguity_df.at['Samoa', 'American Samoa'] = 1/2
wb_contiguity_df.at['American Samoa', 'Samoa'] = 1/2

# Guam
wb_contiguity_df['Guam'] = wb_contiguity_df['Micronesia (Federated States of)']
wb_contiguity_df.loc['Guam'] = wb_contiguity_df.loc['Micronesia (Federated States of)

wb_contiguity_df.at['Micronesia (Federated States of)', 'Guam'] = 1/2
wb_contiguity_df.at['Guam', 'Micronesia (Federated States of)'] = 1/2

# Hong Kong SAR China
wb_contiguity_df.at['Hong Kong SAR China', 'China'] = 1
wb_contiguity_df.at['China', 'Hong Kong SAR China'] = 1

# Macau SAR China
wb_contiguity_df.at['Macau SAR China', 'China'] = 1
wb_contiguity_df.at['China', 'Macau SAR China'] = 1

wb_contiguity_df.at['Macau SAR China', 'Hong Kong SAR China'] = 1/2
wb_contiguity_df.at['Hong Kong SAR China', 'Macau SAR China'] = 1/2

# Northern Mariana Islands
wb_contiguity_df['Northern Mariana Islands'] = wb_contiguity_df['Micronesia (Federate
wb_contiguity_df.loc['Northern Mariana Islands'] = wb_contiguity_df.loc['Micronesia (

wb_contiguity_df.at['Micronesia (Federated States of)', 'Northern Mariana Islands'] =
wb_contiguity_df.at['Northern Mariana Islands', 'Micronesia (Federated States of)'] =

wb_contiguity_df.at['Guam', 'Northern Mariana Islands'] = 1/2
wb_contiguity_df.at['Northern Mariana Islands', 'Guam'] = 1/2

# New Caledonia
wb_contiguity_df['New Caledonia'] = wb_contiguity_df['Vanuatu']
wb_contiguity_df.loc['New Caledonia'] = wb_contiguity_df.loc['Vanuatu']

wb_contiguity_df.at['New Caledonia', 'Vanuatu'] = 1/2
wb_contiguity_df.at['Vanuatu', 'New Caledonia'] = 1/2

# French Polynesia
wb_contiguity_df['French Polynesia'] = wb_contiguity_df['American Samoa']
wb_contiguity_df.loc['French Polynesia'] = wb_contiguity_df.loc['American Samoa']

wb_contiguity_df.at['French Polynesia', 'American Samoa'] = 1/2
wb_contiguity_df.at['American Samoa', 'French Polynesia'] = 1/2
```

```
# Channel Islands
wb_contiguity_df.at['Channel Islands', 'France'] = 1/2
wb_contiguity_df.at['France', 'Channel Islands'] = 1/2

wb_contiguity_df.at['Channel Islands', 'United Kingdom'] = 1/2
wb_contiguity_df.at['United Kingdom', 'Channel Islands'] = 1/2

# Faroe Islands
wb_contiguity_df.at['Faroe Islands', 'United Kingdom'] = 1/5
wb_contiguity_df.at['United Kingdom', 'Faroe Islands'] = 1/5

wb_contiguity_df.at['Faroe Islands', 'Iceland'] = 1/5
wb_contiguity_df.at['Iceland', 'Faroe Islands'] = 1/5

wb_contiguity_df.at['Faroe Islands', 'Norway'] = 1/5
wb_contiguity_df.at['Norway', 'Faroe Islands'] = 1/5

# Gibraltar
wb_contiguity_df.at['Gibraltar', 'Spain'] = 1
wb_contiguity_df.at['Spain', 'Gibraltar'] = 1

wb_contiguity_df.at['Gibraltar', 'Morocco'] = wb_contiguity_df.at['Spain', 'Morocco']
wb_contiguity_df.at['Morocco', 'Gibraltar'] = wb_contiguity_df.at['Morocco', 'Spain']

# Greenland
wb_contiguity_df.at['Greenland', 'Canada'] = 1/5
wb_contiguity_df.at['Canada', 'Greenland'] = 1/5

wb_contiguity_df.at['Greenland', 'Iceland'] = 1/5
wb_contiguity_df.at['Iceland', 'Greenland'] = 1/5

# Isle of Man
wb_contiguity_df.at['Isle of Man', 'United Kingdom'] = 1/2
wb_contiguity_df.at['United Kingdom', 'Isle of Man'] = 1/2

wb_contiguity_df.at['Isle of Man', 'Ireland'] = 1/2
wb_contiguity_df.at['Ireland', 'Isle of Man'] = 1/2

# Kosovo
wb_contiguity_df.at['Kosovo', 'Serbia'] = 1
wb_contiguity_df.at['Serbia', 'Kosovo'] = 1

wb_contiguity_df.at['Kosovo', 'Albania'] = 1
wb_contiguity_df.at['Albania', 'Kosovo'] = 1

wb_contiguity_df.at['Kosovo', 'Montenegro'] = 1
wb_contiguity_df.at['Montenegro', 'Kosovo'] = 1
```

```
wb_contiguity_df.at['Kosovo', 'Macedonia'] = 1
wb_contiguity_df.at['Macedonia', 'Kosovo'] = 1

# Aruba
wb_contiguity_df.at['Aruba', 'Venezuela'] = 1/2
wb_contiguity_df.at['Venezuela', 'Aruba'] = 1/2

wb_contiguity_df.at['Aruba', 'Colombia'] = 1/2
wb_contiguity_df.at['Colombia', 'Aruba'] = 1/2

# Curaçao
wb_contiguity_df.at['Curaçao', 'Venezuela'] = 1/2
wb_contiguity_df.at['Venezuela', 'Curaçao'] = 1/2

wb_contiguity_df.at['Curaçao', 'Colombia'] = 1/2
wb_contiguity_df.at['Colombia', 'Curaçao'] = 1/2

# Cayman Islands
wb_contiguity_df['Cayman Islands'] = wb_contiguity_df['Jamaica']
wb_contiguity_df.loc['Cayman Islands'] = wb_contiguity_df.loc['Jamaica']

wb_contiguity_df.at['Cayman Islands', 'Jamaica'] = 1/2
wb_contiguity_df.at['Jamaica', 'Cayman Islands'] = 1/2

# Saint Martin (French part)
wb_contiguity_df['Saint Martin (French part)'] = wb_contiguity_df['St. Kitts & Nevis']
wb_contiguity_df.loc['Saint Martin (French part)'] = wb_contiguity_df.loc['St. Kitts &

wb_contiguity_df.at['Saint Martin (French part)', 'St. Kitts & Nevis'] = 1/2
wb_contiguity_df.at['St. Kitts & Nevis', 'Saint Martin (French part)'] = 1/2

# Puerto Rico
wb_contiguity_df['Puerto Rico'] = wb_contiguity_df['Dominican Republic']
wb_contiguity_df.loc['Puerto Rico'] = wb_contiguity_df.loc['Dominican Republic']

wb_contiguity_df.at['Puerto Rico', 'Dominican Republic'] = 1/2
wb_contiguity_df.at['Dominican Republic', 'Puerto Rico'] = 1/2

# Sint Maarten
wb_contiguity_df['Sint Maarten'] = wb_contiguity_df['Saint Martin (French part)']
wb_contiguity_df.loc['Sint Maarten'] = wb_contiguity_df.loc['Saint Martin (French part

wb_contiguity_df.at['Sint Maarten', 'Saint Martin (French part)'] = 1
wb_contiguity_df.at['Saint Martin (French part)', 'Sint Maarten'] = 1

# Turks & Caicos Islands
wb_contiguity_df['Turks & Caicos Islands'] = wb_contiguity_df['Bahamas']
```

```
wb_contiguity_df.loc['Turks & Caicos Islands'] = wb_contiguity_df.loc['Bahamas']

wb_contiguity_df.at['Turks & Caicos Islands', 'Bahamas'] = 1/2
wb_contiguity_df.at['Bahamas', 'Turks & Caicos Islands'] = 1/2

# British Virgin Islands
wb_contiguity_df['British Virgin Islands'] = wb_contiguity_df['Puerto Rico']
wb_contiguity_df.loc['British Virgin Islands'] = wb_contiguity_df.loc['Puerto Rico']

wb_contiguity_df.at['British Virgin Islands', 'Puerto Rico'] = 1/2
wb_contiguity_df.at['Puerto Rico', 'British Virgin Islands'] = 1/2

# U.S. Virgin Islands
wb_contiguity_df['U.S. Virgin Islands'] = wb_contiguity_df['Puerto Rico']
wb_contiguity_df.loc['U.S. Virgin Islands'] = wb_contiguity_df.loc['Puerto Rico']

wb_contiguity_df.at['U.S. Virgin Islands', 'Puerto Rico'] = 1/2
wb_contiguity_df.at['Puerto Rico', 'U.S. Virgin Islands'] = 1/2

# Palestinian Territories
wb_contiguity_df.at['Palestinian Territories', 'Israel'] = 1
wb_contiguity_df.at['Israel', 'Palestinian Territories'] = 1

wb_contiguity_df.at['Palestinian Territories', 'Egypt'] = 1
wb_contiguity_df.at['Egypt', 'Palestinian Territories'] = 1

# Bermuda
wb_contiguity_df.at['Bermuda', 'United States'] = 1/5
wb_contiguity_df.at['United States', 'Bermuda'] = 1/5
```

In [59]:
```
# wb_contiguity_df.sum(axis=1)
```

In [60]:
```
# wb_contiguity_df.sum()
```

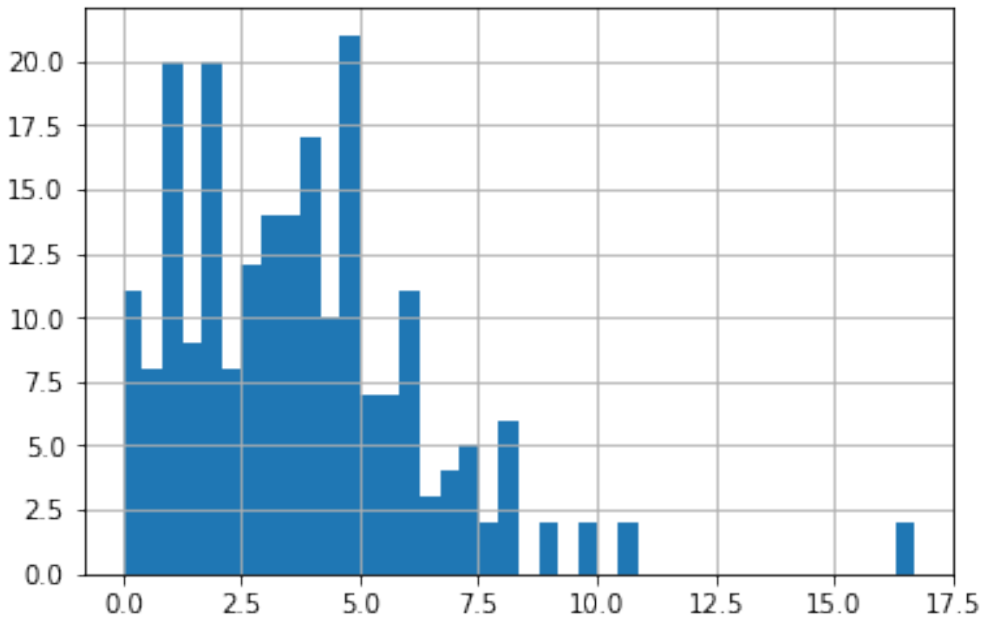
In [45]:
```
x=pd.DataFrame(wb_contiguity_df.sum())
```

In [61]:
```
# x
```

In [47]:
```
%matplotlib inline
```

In [48]:
```
wb_contiguity_df.sum().hist(bins=40)
```

Out[48]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x116dfdb00>
```




In [49]:
```
from plotly import __version__
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
```

In [50]:
```
import plotly.plotly as py
import plotly.graph_objs as go
import pandas as pd
import numpy as np # for generating random data


data = [go.Histogram(x=x[0])]

layout = go.Layout(
    title='Contiguity Histogram',
    xaxis=dict(
        title='Value'
    ),
    yaxis=dict(
        title='Count'
    ),
    bargap=0.2,
    bargroupgap=0.1
)
fig = go.Figure(data=data, layout=layout)
# IPython notebook
py.iplot(fig)
```

```
Out[50]: <plotly.tools.PlotlyDisplay object>

In [52]: wb_contiguity_df.at['Iceland', 'United Kingdom']

Out[52]: 0.0

In [53]: wb_contiguity_df.fillna(0, inplace=True)

In [54]: writer = pd.ExcelWriter('processed_data/contiguity_data_wb.xlsx')
         # writer.define_name('countries', '=Sheet1!$A$2:$A$194')
         workbook=writer.book
         workbook.define_name('countries', '=Sheet1!$A$2:$A$'+str(len(wb_contiguity_df)+1))
         wb_contiguity_df.to_excel(writer)
         writer.save()
```

## D.4. United Nations: International Migrant Stock

# 04 - United Nations Migrant Data

August 15, 2018

## 1 Notebook 4 - United Nations Migrant Data

*@author: Stefan Wigman*

### 1.1 4.1 - Introduction

```
In [1]: import pandas as pd
        import numpy as np
        import csv
```

```
In [2]: xl = pd.ExcelFile("raw data/UN Migrant Data/UN_MigrantStockByOriginAndDestination_2017
```

```
In [3]: countries = pd.ExcelFile("country_list.xlsx")
        countries = countries.parse()
        countries.set_index('country', inplace=True)
```

```
In [4]: df = xl.parse("Table 1", skiprows=15)
```

```
In [5]: df.columns
```

```
Out[5]: Index(['Unnamed: 0', 'Unnamed: 1', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4',
               'Unnamed: 5', 'Total', 'Other North', 'Other South', 'Afghanistan',
               ...
               'Uruguay', 'Uzbekistan', 'Vanuatu',
               'Venezuela (Bolivarian Republic of)', 'Viet Nam',
               'Wallis and Futuna Islands', 'Western Sahara', 'Yemen', 'Zambia',
               'Zimbabwe'],
              dtype='object', length=241)
```

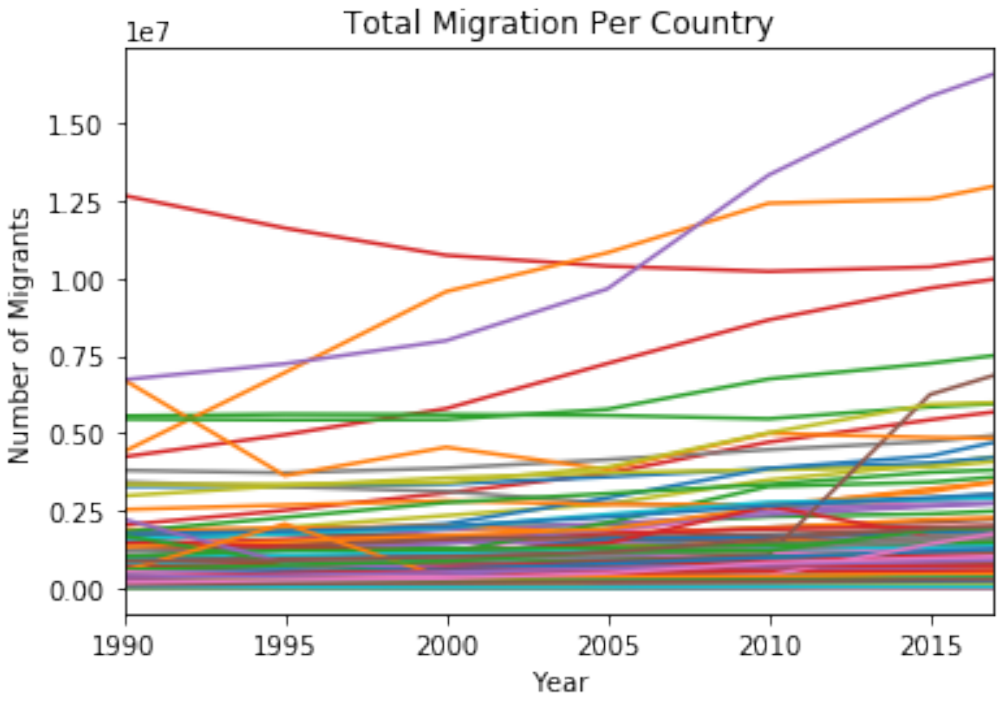
```
In [6]: df.rename(columns = {'Unnamed: 0':'Year',
                            'Unnamed: 1':'SortOrder',
                            'Unnamed: 2': 'Country',
                            'Unnamed: 3': 'Notes',
                            'Unnamed: 4': 'Code',
                            'Unnamed: 5': 'TypeOfData'
                            }, inplace = True)
```

```
In [7]: with open('country_name_dic.csv') as csvfile:
            reader = csv.DictReader(csvfile, delimiter = ',')
            country_name_dict = {rows['alt_name']:rows['name'] for rows in reader}

In [8]: df.set_index('Country', inplace = True)

In [9]: countries_list = list(countries)

In [10]: df = df.rename(columns=country_name_dict, index=country_name_dict)

In [11]: df_2017 = df[df['Year']==2017]

In [12]: df_2017 = df_2017.loc[list(countries.index), list(countries.index)]

In [13]: df_2017 = df_2017.replace('..', 0).fillna(0)

In [14]: df_2017.loc['Switzerland', 'Portugal']

Out[14]: 213555

In [15]: countries=pd.read_excel('processed_data/country_list.xlsx', index_col = 'country')

In [16]: countries_wb=pd.read_excel("raw data/Regions.xlsx", sheetname='Countries', header=0, s
                                names=['Country Data','Region','IncomeGroup'],
                                  parse_cols=None, parse_dates=False, date_parser=None, na_valu

In [17]: countries_wb.rename(index = country_name_dict, inplace = True)

In [18]: countries_wb = countries_wb.sort_values(["Region", "Country Data"])

In [19]: df_2017_wb = df_2017.loc[list(countries_wb.index), list(countries_wb.index)]

In [20]: df_2017_wb = df_2017_wb.replace('..', 0).fillna(0)

In [21]: writer = pd.ExcelWriter('migrant_stock_data.xlsx')
         # writer.define_name('countries', '=Sheet1!$A$2:$A$194')
         workbook=writer.book
         workbook.define_name('countries', '=Sheet1!$A$2:$A$194')
         df_2017.to_excel(writer)
         writer.save()

In [22]: writer = pd.ExcelWriter('processed_data/migrant_stock_data.xlsx')
         # writer.define_name('countries', '=Sheet1!$A$2:$A$194')
         workbook=writer.book
         workbook.define_name('countries', '=Sheet1!$A$2:$A$194')
         df_2017_wb.to_excel(writer)
         writer.save()
```

```
In [23]: from bokeh.palettes import Spectral11
         from bokeh.plotting import figure, show, output_file
         output_file('temp.html')


         numlines=len(df.loc['WORLD'].columns)
         mypalette=Spectral11[0:numlines]

         p = figure(width=500, height=300, x_axis_type="datetime")
         p.multi_line(xs=[df.index.values]*numlines,
                      ys=[df[name].values for name in df],
                      line_color=mypalette,
                      line_width=5)
         show(p)
```

```
/anaconda3/lib/python3.6/site-packages/bokeh/models/sources.py:137: BokehUserWarning: ColumnDa
  "Current lengths: %s" % ", ".join(sorted(str((k, len(v))) for k, v in data.items())), BokehU
```

```
In [25]: df.loc['WORLD'].T.plot()

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x116a9ca58>

In [28]: plotlist=list(countries_wb.index)

In [29]: plotlist.append('Year')

In [30]: %matplotlib inline
         ax = df.loc['WORLD', plotlist].plot(x='Year', legend=False, title='Total Migration Pe
         ax.set_xlabel("Year")
         ax.set_ylabel("Number of Migrants")

Out[30]: <matplotlib.text.Text at 0x113d8b630>
```




```
In [31]: df_ned = df.loc['Netherlands', plotlist]

         df_ned = df_ned[(df_ned.T != '..').any()]

In [32]: df_ned = df_ned.T.dropna(how='all')

In [37]: # df_ned[(df_ned.T > 100000).all()]

In [38]: # df_ned.T>=100

In [39]: # df.loc['WORLD', list(countries_wb.index)]

In [36]: countries_wb.index

Out[36]: Index(['American Samoa', 'Australia', 'Brunei', 'China', 'Fiji',
                'Micronesia (Federated States of)', 'Guam', 'Hong Kong SAR China',
                'Indonesia', 'Japan',
                ...
                'São Tomé & Príncipe', 'Swaziland', 'Seychelles', 'Chad', 'Togo',
                'Tanzania', 'Uganda', 'South Africa', 'Zambia', 'Zimbabwe'],
               dtype='object', name='country', length=217)
```

## D.5. World Bank

# 05 - World Bank Data

August 15, 2018

## 1 5 - World Bank Data

*@author: Stefan Wigman*

### 1.1 5.1 - Introduction

```
In [1]: from DataFunctions import *
        import datetime
        import pandas as pd
        import wbdata
        import numpy as np
        import csv
```

```
In [32]: # wbdata.get_source()
```

```
In [33]: # x=wbdata.get_indicator(source=2)
```

```
In [4]: type(x)
```

```
Out[4]: NoneType
```

```
In [5]: wbdata.search_indicators('urban', source=32)
```

```
In [6]: countries=pd.read_excel("Regions.xlsx", sheetname='Countries', header=0, skiprows=None
                               names=['Country Data','Region','IncomeGroup'],
                               parse_cols=None, parse_dates=False, date_parser=None, na_values
```

```
In [7]: indicator_dataframe, indicators, tabnames=GetIndicatorsWB(file='Selected_Indicators.xls
```

```
In [8]: tabnames['Country Data']='Country Data'
        tabnames['IncomeGroup']='IncomeGroup'
        tabnames['Region']='Region'
```

```
In [9]: indicators = {
         'NY.GDP.MKTP.CD': 'GDP (current US$)',
         'SP.DYN.CBRT.IN': 'Birth rate, crude (per 1,000 people)',
         'SP.DYN.LE00.IN': 'Life expectancy at birth, total (years)',
         'SP.DYN.TFRT.IN': 'Fertility rate, total (births per woman)',
         'SP.POP.TOTL': 'Population, total',
```

```
        'SP.POP.TOTL.FE.IN': 'Population, female',
        'SP.POP.TOTL.MA.IN': 'Population, male',
        'SP.URB.GROW': 'Urban population growth (annual %)',
        'SP.URB.TOTL.IN.ZS': 'Urban population (% of total)'}
```

```
In [10]: wbdata = GetDataWB(indicators, 2010, 2016)
```

```
/Users/stefanwigman/Desktop/Migration Thesis/03 - Data/DataFunctions.py:98: SettingWithCopyWarn
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  df_filled[column_source][df_filled[column].notnull()] = 'WB data ' + str(year2)
```

```
In [34]: # wbdata.head()
```

```
In [12]: source_cols=[col for col in wbdata.columns if 'source' in col]
         cols=wbdata.columns[~wbdata.columns.isin(source_cols)]
```

```
In [13]: (1-(wbdata[cols].isnull().sum().sum())/(264*74))*100
```

```
Out[13]: 99.3140868140868
```

```
In [14]: regions=GetRegionIncomeDataWB()
```

```
In [15]: wb_data_regions = regions.join(wbdata, how='inner')
```

```
In [16]: countries = pd.ExcelFile("country_list.xlsx")
         countries = countries.parse()
         countries.set_index('country', inplace=True)
```

```
In [17]: with open('processed_data/country_name_dic.csv') as csvfile:
             reader = csv.DictReader(csvfile, delimiter = ',')
             country_name_dict = {rows['alt_name']:rows['name'] for rows in reader}
```

```
In [18]: wb_data_regions = wb_data_regions.rename(index=country_name_dict)
```

```
In [19]: wb_data_regions = countries.join(wb_data_regions, how='inner')
```

```
In [20]: for country in countries.index:
             if country not in wb_data_regions.index:
                 print(country)
```

```
In [21]: countries.index
```

```
Out[21]: Index(['Afghanistan', 'Albania', 'Algeria', 'Andorra', 'Angola',
                'Antigua & Barbuda', 'Argentina', 'Armenia', 'Australia', 'Austria',
                ...
                'Tanzania', 'United States', 'Uruguay', 'Uzbekistan', 'Vanuatu',
                'Venezuela', 'Vietnam', 'Yemen', 'Zambia', 'Zimbabwe'],
               dtype='object', name='country', length=193)
```

```
In [22]: wb_data_regions.index

Out[22]: Index(['Afghanistan', 'Albania', 'Algeria', 'Andorra', 'Angola',
                'Antigua & Barbuda', 'Argentina', 'Armenia', 'Australia', 'Austria',
                ...
                'Tanzania', 'United States', 'Uruguay', 'Uzbekistan', 'Vanuatu',
                'Venezuela', 'Vietnam', 'Yemen', 'Zambia', 'Zimbabwe'],
               dtype='object', name='country', length=193)

In [23]: region_income_data=FillByRegionAndIncomeWB(wb_data_regions)

In [24]: income_data=FillByIncomeWB(region_income_data)

In [25]: region_data=FillByRegionWB(income_data)

In [26]: mean_data=FillWithMeanWB(region_data)

In [28]: def WriteToExcelWB(dataframe, tabnames, filename='testdata1.xlsx'):
             """
             Function that writes the (complete) dataframe to Excel in the correct
             format, with a separate tab for each indicator.


             ------
             Inputs
             ------
             dataframe:  The dataframe to write to Excel
             tabnames:   The dictionary that contains the tabnames of the indicators
             filename:   The name of the resulting Excel file

             -------
             Outputs
             -------
             An Excelfile with a separate sheet for each indicator.
             Each sheet contains a list of the countries with their values, and
             a column containing the source of each value.

             """

             writer = pd.ExcelWriter(filename)
             dont_include = ['Country Data', 'Region', 'IncomeGroup', 'Population0to14',
                             'Population15to34','Population35to64','PopulationOver65']

             source_cols=[col for col in dataframe.columns if 'source' in col]
             dataframe1=dataframe.drop(source_cols, axis=1)

             region_list=["East Asia and Pacific",
             'Europe and Central Asia',
             "Latin America and Caribbean",
```

```
            "Middle East and North Africa",
            "North America",
            "South Asia",
            "Sub Saharan Africa"]

            regional_data=pd.DataFrame(region_list)
            regional_data.set_index(0)

            regional_data.index.rename("Region")
            print(regional_data)



            regional_data.to_excel(writer, 'Regional Data')

            for column in dataframe1.columns:
                if column in dont_include:
                    df_to_write=pd.DataFrame(dataframe1[column])
                    df_to_write.to_excel(writer, tabnames[column][:31])
                else:
                    df_to_write = dataframe[[column,column+" source"]]
                    df_to_write.to_excel(writer, tabnames[column][:31])
```

```
In [30]: WriteToExcelWB(mean_data, tabnames, filename='processed_data/WB_data.xlsx')
```

```
                              0
0         East Asia and Pacific
1       Europe and Central Asia
2   Latin America and Caribbean
3  Middle East and North Africa
4                 North America
5                    South Asia
6            Sub Saharan Africa
```

## D.6. CIA World Factbook

# 06 - CIA World Factbook

August 15, 2018

In [1]:
```
import json
import pandas as pd
import csv
import numpy as np
```

In [2]:
```
countries = pd.ExcelFile("processed_data/country_list.xlsx")
countries = countries.parse()
countries.set_index('country', inplace=True)
```

In [3]:
```
countries_wb=pd.read_excel("raw data/Regions.xlsx", sheetname='Countries', header=0, sk
                           names=['Country Data','Region','IncomeGroup'],
                             parse_cols=None, parse_dates=False, date_parser=None, na_values
```

In [4]:
```
with open('processed_data/country_name_dic.csv', encoding='UTF-8') as csvfile:
    reader = csv.DictReader(csvfile, delimiter = ',')
    country_name_dict = {rows['alt_name']:rows['name'] for rows in reader}
```

In [5]:
```
json1_file = open('raw data/CIA World Factbook/2018-04-30_factbook.json')
json1_str = json1_file.read()
```

In [6]:
```
json1_data = json.loads(json1_str)
```

In [7]:
```
type(json1_data)
```

Out[7]:
```
dict
```

In [8]:
```
len(json1_data)
```

Out[8]:
```
2
```

In [83]:
```
# print(json1_data["countries"]["zambia"]["data"]["people"].keys())
```

In [82]:
```
# print(json1_data["countries"]["curacao"]["data"]["economy"]['gdp'])
```

In [11]:
```
print(json1_data["countries"]["germany"]["data"]["people"]["population"])
```

```
{'total': 80594017, 'global_rank': 19, 'date': '2017-07-01'}
```

```
In [12]: df = pd.DataFrame.from_dict({(i,j): json1_data[i][j]
                                      for i in json1_data.keys()
                                      for j in json1_data[i].keys()})

In [13]: df["countries"]["afghanistan"]

Out[13]: data        {'name': 'Afghanistan', 'introduction': {'back...
         metadata    {'date': '2018-04-24', 'source': 'https://web...
         Name: afghanistan, dtype: object

In [14]: df.loc["data"]["countries"]["afghanistan"]["name"]

Out[14]: 'Afghanistan'

In [15]: df1=df.loc["data"]["countries"]

In [80]: # data_dict = {}
         # for country in df1.index:
         # #     print(df1[country]["name"])
         # #     print(country)
         #     df1.rename(index={country : df1.loc[country]["name"]}, inplace=True)

In [20]: df1 = pd.DataFrame(df1.rename(index=country_name_dict))

In [21]: for country in countries.index:
             if country not in df1.index:
                 print(country)

In [81]: # for country in df1.index:
         #     if country not in country_name_dict.keys():
         #         print(country)

In [23]: df1.loc[country]['data']['people']['youth_unemployment']['total']['value']

Out[23]: 16.5

In [24]: data_dic = {

                     # GEOGRAPHY
                     "area total" :
                     "df1.loc[country]['data']['geography']['area']['total']['value']",

                     "area land" :
                     "df1.loc[country]['data']['geography']['area']['land']['value']",

                     "area water" :
                     "df1.loc[country]['data']['geography']['area']['land']['value']",

                     "latitude" :
                     "df1.loc[country]['data']['geography']['geographic_coordinates']['latitud
```

```
"longitude" :
"df1.loc[country]['data']['geography']['geographic_coordinates']['longitu

"total length of land boundaries":
"df1.loc[country]['data']['geography']['land_boundaries']['total']['value

"total length of coastline" :
"df1.loc[country]['data']['geography']['coastline']['value']",

"highest point elevation" :
"df1.loc[country]['data']['geography']['elevation']['highest_point']['ele

"lowest point elevation" :
"df1.loc[country]['data']['geography']['elevation']['lowest_point']['eleva

"mean elevation" :
"df1.loc[country]['data']['geography']['elevation']['mean_elevation']['va

"land use agricultural land arable" :
"df1.loc[country]['data']['geography']['land_use']['by_sector']['agricultu

"land use agricultural land permananent crops" :
"df1.loc[country]['data']['geography']['land_use']['by_sector']['agricultu

"land use agricultural land permanent pasture" :
"df1.loc[country]['data']['geography']['land_use']['by_sector']['agricultu

"land use agricultural land total" :
"df1.loc[country]['data']['geography']['land_use']['by_sector']['agricultu

"land use forest land" :
"df1.loc[country]['data']['geography']['land_use']['by_sector']['forest']

"land use other land" :
"df1.loc[country]['data']['geography']['land_use']['by_sector']['other'][

"land use source year":
"df1.loc[country]['data']['geography']['land_use']['date']",

"irrigated land" :
"df1.loc[country]['data']['geography']['irrigated_land']['value']",

"irrigated land source year" :
"df1.loc[country]['data']['geography']['irrigated_land']['date']",

# PEOPLE AND SOCIETY
```

```
"population":
"df1.loc[country]['data']['people']['population']['total']",

"population source year":
"df1.loc[country]['data']['people']['population']['date']",

"population 0 to 14 female" :
"df1.loc[country]['data']['people']['age_structure']['0_to_14']['females']

"population 0 to 14 male" :
"df1.loc[country]['data']['people']['age_structure']['0_to_14']['males']"

"population 0 to 14 percent" :
"df1.loc[country]['data']['people']['age_structure']['0_to_14']['percent']

"population 15 to 24 female" :
"df1.loc[country]['data']['people']['age_structure']['15_to_24']['females

"population 15 to 24 male" :
"df1.loc[country]['data']['people']['age_structure']['15_to_24']['males']

"population 15 to 24 percent" :
"df1.loc[country]['data']['people']['age_structure']['15_to_24']['percent

"population 25 to 54 female" :
"df1.loc[country]['data']['people']['age_structure']['25_to_54']['females

"population 25 to 54 male" :
"df1.loc[country]['data']['people']['age_structure']['25_to_54']['males']

"population 25 to 54 percent" :
"df1.loc[country]['data']['people']['age_structure']['25_to_54']['percent

"population 55 to 64 female" :
"df1.loc[country]['data']['people']['age_structure']['55_to_64']['females

"population 55 to 64 male" :
"df1.loc[country]['data']['people']['age_structure']['55_to_64']['males']

"population 55 to 64 percent" :
"df1.loc[country]['data']['people']['age_structure']['55_to_64']['percent

"population 65 and over female" :
"df1.loc[country]['data']['people']['age_structure']['65_and_over']['femal

"population 65 and over male" :
"df1.loc[country]['data']['people']['age_structure']['65_and_over']['males
```

```
"population 65 and over percent" :
"df1.loc[country]['data']['people']['age_structure']['65_and_over']['perc

"population distribution source year" :
"df1.loc[country]['data']['people']['age_structure']['date']",

"elderly dependency ratio" :
"df1.loc[country]['data']['people']['dependency_ratios']['ratios']['elder

"potential support ratio" :
"df1.loc[country]['data']['people']['dependency_ratios']['ratios']['poten

"total dependency ratio" :
"df1.loc[country]['data']['people']['dependency_ratios']['ratios']['total

"youth dependency ratio" :
"df1.loc[country]['data']['people']['dependency_ratios']['ratios']['youth

"dependency ratios source year":
"df1.loc[country]['data']['people']['dependency_ratios']['date']",

"median age female" :
"df1.loc[country]['data']['people']['median_age']['female']['value']",

"median age male" :
"df1.loc[country]['data']['people']['median_age']['male']['value']",

"median age total" :
"df1.loc[country]['data']['people']['median_age']['total']['value']",

"median age source year" :
"df1.loc[country]['data']['people']['median_age']['date']",

"population growth rate" :
"df1.loc[country]['data']['people']['population_growth_rate']['growth_rat

"population growth rate source year" :
"df1.loc[country]['data']['people']['population_growth_rate']['date']",

"birth rate per 1000 population" :
"df1.loc[country]['data']['people']['birth_rate']['births_per_1000_popula

"birth rate per 1000 population source year" :
"df1.loc[country]['data']['people']['birth_rate']['date']",

"death rate per 1000 population" :
"df1.loc[country]['data']['people']['death_rate']['deaths_per_1000_popula
```

```
"death rate per 1000 population source year" :
"df1.loc[country]['data']['people']['death_rate']['date']",

"net migration rate per 1000" :
"df1.loc[country]['data']['people']['net_migration_rate']['migrants_per_1

"net migration rate per 1000 source year" :
"df1.loc[country]['data']['people']['date']",

"urbanization rate" :
"df1.loc[country]['data']['people']['urbanization']['rate_of_urbanization

"urban population pc" :
"df1.loc[country]['data']['people']['urbanization']['urban_population']['

"urban population pc source year" :
"df1.loc[country]['data']['people']['urbanization']['urban_population']['

"mothers mean age at first birth" :
"df1.loc[country]['data']['people']['mothers_mean_age_at_first_birth']['ag

"mothers mean age at first birth source year" :
"df1.loc[country]['data']['people']['mothers_mean_age_at_first_birth']['da

"maternal mortality rate (per 100k)" :
"df1.loc[country]['data']['people']['maternal_mortality_rate']['deaths_per

"maternal mortality rate (per 100k) source year" :
"df1.loc[country]['data']['people']['maternal_mortality_rate']['date']",

"infant mortality rate per 1000 births total" :
"df1.loc[country]['data']['people']['infant_mortality_rate']['total']['val

 "infant mortality rate per 1000 births male" :
"df1.loc[country]['data']['people']['infant_mortality_rate']['male']['valu

 "infant mortality rate per 1000 births female" :
"df1.loc[country]['data']['people']['infant_mortality_rate']['female']['va

"infant mortality rate per 1000 births source year" :
"df1.loc[country]['data']['people']['infant_mortality_rate']['date']",

"life expectancy at birth":
"df1.loc[country]['data']['people']['life_expectancy_at_birth']['total_pop

"life expectancy at birth male":
"df1.loc[country]['data']['people']['life_expectancy_at_birth']['male']['
```

```
"life expectancy at birth female":
"df1.loc[country]['data']['people']['life_expectancy_at_birth']['female']

"life expectancy at birth source year" :
"df1.loc[country]['data']['people']['life_expectancy_at_birth']['date']",

"total fertility rate (children born per woman)" :
"df1.loc[country]['data']['people']['total_fertility_rate']['children_bor

"total fertility rate (children born per woman) source year" :
"df1.loc[country]['data']['people']['total_fertility_rate']['date']",

"contraceptive prevalence rate pc" :
"df1.loc[country]['data']['people']['contraceptive_prevalence_rate']['valu

"contraceptive prevalence rate pc source year" :
"df1.loc[country]['data']['people']['contraceptive_prevalence_rate']['date

"health expenditures pc of gdp" :
"df1.loc[country]['data']['people']['health_expenditures']['percent_of_gdp

"health expenditures pc of gdp (source year)" :
"df1.loc[country]['data']['people']['health_expenditures']['date']",

"physicians density per 1000" :
"df1.loc[country]['data']['people']['physicians_density']['physicians_per_

"physicians density per 1000 source year" :
"df1.loc[country]['data']['people']['physicians_density']['date']",

"hospital beds density per 1000" :
"df1.loc[country]['data']['people']['hospital_bed_density']['beds_per_1000

"hospital beds density per 1000 source year" :
"df1.loc[country]['data']['people']['hospital_bed_density']['date']",

"drinking water source improved rural" :
"df1.loc[country]['data']['people']['drinking_water_source']['improved']['

"drinking water source improved urban" :
"df1.loc[country]['data']['people']['drinking_water_source']['improved']['

"drinking water source improved total" :
"df1.loc[country]['data']['people']['drinking_water_source']['improved']['

"drinking water source unimproved rural" :
"df1.loc[country]['data']['people']['drinking_water_source']['unimproved']
```

```
"drinking water source unimproved urban" :
"df1.loc[country]['data']['people']['drinking_water_source']['unimproved'

"drinking water source unimproved total" :
"df1.loc[country]['data']['people']['drinking_water_source']['unimproved'

"drinking water source source year" :
"df1.loc[country]['data']['people']['drinking_water_source']['date']",

"sanitation facility access improved rural" :
"df1.loc[country]['data']['people']['sanitation_facility_access']['improv

"sanitation facility access improved urban" :
"df1.loc[country]['data']['people']['sanitation_facility_access']['improv

"sanitation facility access improved total" :
"df1.loc[country]['data']['people']['sanitation_facility_access']['improv

"sanitation facility access unimproved rural" :
"df1.loc[country]['data']['people']['sanitation_facility_access']['unimpr

"sanitation facility access unimproved urban" :
"df1.loc[country]['data']['people']['sanitation_facility_access']['unimpr

"sanitation facility access unimproved total" :
"df1.loc[country]['data']['people']['sanitation_facility_access']['unimpr

"sanitation facility access source year" :
"df1.loc[country]['data']['people']['sanitation_facility_access']['date']

"hiv aids adult prevalence rate pc of adults" :
"df1.loc[country]['data']['people']['hiv_aids']['adult_prevalence_rate']['

"hiv aids adult prevalence rate pc of adults source year" :
"df1.loc[country]['data']['people']['hiv_aids']['adult_prevalence_rate']['

"hiv aids deaths" :
"df1.loc[country]['data']['people']['hiv_aids']['deaths']['total']",

"hiv aids deaths source year" :
"df1.loc[country]['data']['people']['hiv_aids']['deaths']['date']",

"people living with hiv aids" :
"df1.loc[country]['data']['people']['hiv_aids']['people_living_with_hiv_a

"people living with hiv aids source year" :
"df1.loc[country]['data']['people']['hiv_aids']['people_living_with_hiv_a
```

```
"obesity pc of adults" :
"df1.loc[country]['data']['people']['adult_obesity']['percent_of_adults']"

"obesity pc of adults source year" :
"df1.loc[country]['data']['people']['adult_obesity']['date']",

"underweight children under age of five pc" :
"df1.loc[country]['data']['people']['underweight_children']['percent_of_c

"underweight children under age of five pc source year" :
"df1.loc[country]['data']['people']['underweight_children']['date']",

"education expenditures pc of gdp" :
"df1.loc[country]['data']['people']['education_expenditures']['percent_of_

"education expenditures pc of gdp (source year)" :
"df1.loc[country]['data']['people']['education_expenditures']['date']",

"school life expectancy total":
"df1.loc[country]['data']['people']['school_life_expectancy']['total']['va

"school life expectancy male":
"df1.loc[country]['data']['people']['school_life_expectancy']['male']['val

"school life expectancy female":
"df1.loc[country]['data']['people']['school_life_expectancy']['female']['

"school life expectancy source year":
"df1.loc[country]['data']['people']['school_life_expectancy']['date']",

"youth unemployment total":
"df1.loc[country]['data']['people']['youth_unemployment']['total']['value

"youth unemployment male":
"df1.loc[country]['data']['people']['youth_unemployment']['male']['value']

"youth unemployment female":
"df1.loc[country]['data']['people']['youth_unemployment']['female']['valu

"youth unemployment source year":
"df1.loc[country]['data']['people']['youth_unemployment']['date']",

# GOVERNMENT



"suffrage (age)":
"df1.loc[country]['data']['government']['suffrage']['age']",
```

```
"suffrage (compulsory)":
"df1.loc[country]['data']['government']['suffrage']['compulsory']",


# ECONOMY

"gdp purchasing power parity" :
"df1.loc[country]['data']['economy']['gdp']['purchasing_power_parity']['a

"gdp purchasing power source year" :
"df1.loc[country]['data']['economy']['gdp']['purchasing_power_parity']['a

"gdp official exchange rate usd" :
"df1.loc[country]['data']['economy']['gdp']['official_exchange_rate']['US

"gdp official exchange rate usd source year" :
"df1.loc[country]['data']['economy']['gdp']['official_exchange_rate']['da

"gdp real growth rate pc" :
"df1.loc[country]['data']['economy']['gdp']['real_growth_rate']['annual_v

"gdp real growth rate pc source year" :
"df1.loc[country]['data']['economy']['gdp']['real_growth_rate']['annual_v

"gdp per capita purchasing power parity usd" :
"df1.loc[country]['data']['economy']['gdp']['per_capita_purchasing_power_

"gdp per capita purchasing power parity usd source year" :
"df1.loc[country]['data']['economy']['gdp']['per_capita_purchasing_power_

"gross national saving pc of gdp" :
"df1.loc[country]['data']['economy']['gross_national_saving']['annual_valu

"gross national saving pc of gdp source year" :
"df1.loc[country]['data']['economy']['gross_national_saving']['annual_valu

"gdp composition by end use exports of goods and services" :
"df1.loc[country]['data']['economy']['gdp']['composition']['by_end_use']['

"gdp composition by end use government consumption" :
"df1.loc[country]['data']['economy']['gdp']['composition']['by_end_use']['

"gdp composition by end use household consumption" :
"df1.loc[country]['data']['economy']['gdp']['composition']['by_end_use']['

"gdp composition by end use imports of goods and services" :
```

```
"df1.loc[country]['data']['economy']['gdp']['composition']['by_end_use'][

"gdp composition by end use investment in fixed capital" :
"df1.loc[country]['data']['economy']['gdp']['composition']['by_end_use'][

"gdp composition by end use investment in inventories" :
"df1.loc[country]['data']['economy']['gdp']['composition']['by_end_use'][

"gdp composition by end use source year" :
"df1.loc[country]['data']['economy']['gdp']['composition']['by_end_use'][

"gdp composition by sector of origin agriculture" :
"df1.loc[country]['data']['economy']['gdp']['composition']['by_sector_of_

"gdp composition by sector of origin industry" :
"df1.loc[country]['data']['economy']['gdp']['composition']['by_sector_of_

"gdp composition by sector of origin services" :
"df1.loc[country]['data']['economy']['gdp']['composition']['by_sector_of_

"gdp composition by sector of origin source year" :
"df1.loc[country]['data']['economy']['gdp']['composition']['by_sector_of_

"industrial production growth rate annual pc" :
"df1.loc[country]['data']['economy']['industrial_production_growth_rate']

"industrial production growth rate annual pc source year" :
"df1.loc[country]['data']['economy']['industrial_production_growth_rate']

"labor force total size" :
"df1.loc[country]['data']['economy']['labor_force']['total_size']['total_

"labor force total size source year" :
"df1.loc[country]['data']['economy']['labor_force']['total_size']['date']

"unemployment rate pc" :
"df1.loc[country]['data']['economy']['unemployment_rate']['annual_values'

"unemployment rate pc source year" :
"df1.loc[country]['data']['economy']['unemployment_rate']['annual_values'

"population below poverty line pc" :
"df1.loc[country]['data']['economy']['population_below_poverty_line']['va

"population below poverty line pc source year" :
"df1.loc[country]['data']['economy']['population_below_poverty_line']['da

"pc household income or consumption by highest 10 pc" :
```

```
"df1.loc[country]['data']['economy']['household_income_by_percentage_share

"pc household income or consumption by lowest 10 pc" :
"df1.loc[country]['data']['economy']['household_income_by_percentage_share

"pc household income or consumption by lowest 10 pc source year" :
"df1.loc[country]['data']['economy']['household_income_by_percentage_share

"distribution of family income gini index" :
"df1.loc[country]['data']['economy']['distribution_of_family_income']['an

"distribution of family income gini index source year" :
"df1.loc[country]['data']['economy']['distribution_of_family_income']['an

"budget expenditures usd" :
"df1.loc[country]['data']['economy']['budget']['expenditures']['value']",

"budget revenues usd" :
"df1.loc[country]['data']['economy']['budget']['revenues']['value']",

"budget usd source year" :
"df1.loc[country]['data']['economy']['budget']['date']",

"taxes and other revenues pc of gdp" :
"df1.loc[country]['data']['economy']['taxes_and_other_revenues']['percent

"taxes and other revenues pc of gdp source year" :
"df1.loc[country]['data']['economy']['taxes_and_other_revenues']['date']"

"budget surplus or deficit pc of gdp" :
"df1.loc[country]['data']['economy']['budget_surplus_or_deficit']['percen

"budget surplus or deficit pc of gdp source year" :
"df1.loc[country]['data']['economy']['budget_surplus_or_deficit']['date']

"public debt pc of gdp" :
"df1.loc[country]['data']['economy']['public_debt']['annual_values'][0]['

"public debt pc of gdp source year" :
"df1.loc[country]['data']['economy']['public_debt']['annual_values'][0]['

"inflation rate consumer prices" :
"df1.loc[country]['data']['economy']['inflation_rate']['annual_values'][0

"inflation rate consumer prices source year" :
"df1.loc[country]['data']['economy']['inflation_rate']['annual_values'][0

"central bank discount rate" :
```

```
"df1.loc[country]['data']['economy']['central_bank_discount_rate']['annual

"central bank discount rate source year" :
"df1.loc[country]['data']['economy']['central_bank_discount_rate']['annual

"commercial bank prime lending rate" :
"df1.loc[country]['data']['economy']['commercial_bank_prime_lending_rate']

"commercial bank prime lending rate source year" :
"df1.loc[country]['data']['economy']['commercial_bank_prime_lending_rate']

"stock of narrow money" :
"df1.loc[country]['data']['economy']['stock_of_narrow_money']['annual_valu

"stock of narrow money source year" :
"df1.loc[country]['data']['economy']['stock_of_narrow_money']['annual_valu

"stock of broad money" :
"df1.loc[country]['data']['economy']['stock_of_broad_money']['annual_value

"stock of broad money source year" :
"df1.loc[country]['data']['economy']['stock_of_broad_money']['annual_value

"stock of domestic credit" :
"df1.loc[country]['data']['economy']['stock_of_domestic_credit']['annual_

"stock of domestic credit source year" :
"df1.loc[country]['data']['economy']['stock_of_domestic_credit']['annual_

"market value of publicly traded shares" :
"df1.loc[country]['data']['economy']['market_value_of_publicly_traded_shar

"market value of publicly traded shares source year" :
"df1.loc[country]['data']['economy']['market_value_of_publicly_traded_shar

"current account balance" :
"df1.loc[country]['data']['economy']['current_account_balance']['annual_va

"current account balance source year" :
"df1.loc[country]['data']['economy']['current_account_balance']['annual_va

"exports total value" :
"df1.loc[country]['data']['economy']['exports']['total_value']['annual_val

"exports total value source year" :
"df1.loc[country]['data']['economy']['exports']['total_value']['annual_val

"imports total value" :
```

```
"df1.loc[country]['data']['economy']['imports']['total_value']['annual_val

"imports total value source year" :
"df1.loc[country]['data']['economy']['imports']['total_value']['annual_val

"reserves of foreign exchange and gold" :
"df1.loc[country]['data']['economy']['reserves_of_foreign_exchange_and_gol

"reserves of foreign exchange and gold source year" :
"df1.loc[country]['data']['economy']['reserves_of_foreign_exchange_and_gol

"external debt" :
"df1.loc[country]['data']['economy']['external_debt']['annual_values'][0]

"external debt source year" :
"df1.loc[country]['data']['economy']['external_debt']['annual_values'][0]

"stock of direct foreign investment abroad usd" :
"df1.loc[country]['data']['economy']['stock_of_direct_foreign_investment'

"stock of direct foreign investment abroad usd source year" :
"df1.loc[country]['data']['economy']['stock_of_direct_foreign_investment'

"stock of direct foreign investment at home usd" :
"df1.loc[country]['data']['economy']['stock_of_direct_foreign_investment'

"stock of direct foreign investment at home usd source year" :
"df1.loc[country]['data']['economy']['stock_of_direct_foreign_investment'


# ENERGY - ELECTRICITY

"total electrification pc of pop" :
"df1.loc[country]['data']['energy']['electricity']['access']['total_electr

"rural electrification pc of rural pop" :
"df1.loc[country]['data']['energy']['electricity']['access']['rural_electr

"urban electrification pc of urban pop" :
"df1.loc[country]['data']['energy']['electricity']['access']['urban_electr

"electrification pc of pop source year" :
"df1.loc[country]['data']['energy']['electricity']['access']['date']",

"electricity production kWh" :
"df1.loc[country]['data']['energy']['electricity']['production']['kWh']",

"electricity production kWh source year":
```

```
"df1.loc[country]['data']['energy']['electricity']['production']['date']"

"electricity consumption kWh" :
"df1.loc[country]['data']['energy']['electricity']['consumption']['kWh']"

"electricity consumption kWh source year":
"df1.loc[country]['data']['energy']['electricity']['consumption']['date']

"electricity exports kWh" :
"df1.loc[country]['data']['energy']['electricity']['exports']['kWh']",

"electricity exports kWh source year":
"df1.loc[country]['data']['energy']['electricity']['exports']['date']",

"electricity imports kWh" :
"df1.loc[country]['data']['energy']['electricity']['imports']['kWh']",

"electricity imports kWh source year":
"df1.loc[country]['data']['energy']['electricity']['imports']['date']",

"electricity installed generating capacity kW" :
"df1.loc[country]['data']['energy']['electricity']['installed_generating_

"electricity installed generating capacity kW source year":
"df1.loc[country]['data']['energy']['installed_generating_capacity']['exp

"electricity from fossil fuels pc of total installed capacity":
"df1.loc[country]['data']['energy']['electricity']['by_source']['fossil_fu

"electricity from fossil fuels pc of total installed capacity source year
"df1.loc[country]['data']['energy']['electricity']['by_source']['fossil_fu

"electricity from nuclear fuels pc of total installed capacity":
"df1.loc[country]['data']['energy']['electricity']['by_source']['nuclear_

"electricity from nuclear fuels pc of total installed capacity source yea
"df1.loc[country]['data']['energy']['electricity']['by_source']['nuclear_

"electricity from hydroelectric plants pc of total installed capacity":
"df1.loc[country]['data']['energy']['electricity']['by_source']['hydroele

"electricity from hydroelectric plants pc of total installed capacity sou
"df1.loc[country]['data']['energy']['electricity']['by_source']['hydroele

"electricity from other renewable sources pc of total installed capacity"
"df1.loc[country]['data']['energy']['electricity']['by_source']['other_re

"electricity from other renewable sources pc of total installed capacity
```

```
"df1.loc[country]['data']['energy']['electricity']['by_source']['other_re

# ENERGY - OIL

"crude oil production bbl per day" :
"df1.loc[country]['data']['energy']['crude_oil']['production']['bbl_per_da

"crude oil production bbl per day source year" :
"df1.loc[country]['data']['energy']['crude_oil']['production']['date']",

"crude oil exports bbl per day" :
"df1.loc[country]['data']['energy']['crude_oil']['exports']['bbl_per_day']

"crude oil exports bbl per day source year" :
"df1.loc[country]['data']['energy']['crude_oil']['exports']['date']",

"crude oil imports bbl per day" :
"df1.loc[country]['data']['energy']['crude_oil']['imports']['bbl_per_day']

"crude oil imports bbl per day source year" :
"df1.loc[country]['data']['energy']['crude_oil']['imports']['date']",

"crude oil proved reserves bbl" :
"df1.loc[country]['data']['energy']['crude_oil']['proved_reserves']['bbl']

"crude proved reserves bbl source year" :
"df1.loc[country]['data']['energy']['crude_oil']['proved_reserves']['date

# ENERGY - REFINED PETROLEUM PRODUCTS

"refined petroleum products production bbl per day":
"df1.loc[country]['data']['energy']['refined_petroleum_products']['produc

"refined petroleum products production bbl per day source year":
"df1.loc[country]['data']['energy']['refined_petroleum_products']['produc

"refined petroleum products consumption bbl per day":
"df1.loc[country]['data']['energy']['refined_petroleum_products']['consum

"refined petroleum products consumption bbl per day source year":
"df1.loc[country]['data']['energy']['refined_petroleum_products']['consum

"refined petroleum products exports bbl per day":
"df1.loc[country]['data']['energy']['refined_petroleum_products']['export

"refined petroleum products exports bbl per day source year":
"df1.loc[country]['data']['energy']['refined_petroleum_products']['export
```

```
"refined petroleum products imports bbl per day":
"df1.loc[country]['data']['energy']['refined_petroleum_products']['imports

"refined petroleum products imports bbl per day source year":
"df1.loc[country]['data']['energy']['refined_petroleum_products']['imports

# ENERGY - NATURAL GAS

"natural gas production cubic meters" :
"df1.loc[country]['data']['energy']['natural_gas']['production']['cubic_me

"natural gas production cubic meters source year" :
"df1.loc[country]['data']['energy']['natural_gas']['production']['date']"

"natural gas consumption cubic meters" :
"df1.loc[country]['data']['energy']['natural_gas']['consumption']['cubic_m

"natural gas consumption cubic meters source year" :
"df1.loc[country]['data']['energy']['natural_gas']['consumption']['date']

"natural gas exports cubic meters" :
"df1.loc[country]['data']['energy']['natural_gas']['exports']['cubic_metre

"natural gas exports cubic meters source year" :
"df1.loc[country]['data']['energy']['natural_gas']['exports']['date']",

"natural gas imports cubic meters" :
"df1.loc[country]['data']['energy']['natural_gas']['imports']['cubic_metre

"natural gas imports cubic meters source year" :
"df1.loc[country]['data']['energy']['natural_gas']['imports']['date']",

"natural gas proved reserves cubic meters" :
"df1.loc[country]['data']['energy']['natural_gas']['proved_reserves']['cub

"natural gas proved reserves cubic meters source year" :
"df1.loc[country]['data']['energy']['natural_gas']['proved_reserves']['dat

# ENERGY - CARBON DIOXIDE EMISSIONS

"carbon dioxide emissions from energy consumption Mt":
"df1.loc[country]['data']['energy']['carbon_dioxide_emissions_from_consump

"carbon dioxide emissions from energy consumption Mt source year":
"df1.loc[country]['data']['energy']['carbon_dioxide_emissions_from_consump

# COMMUNICATIONS
```

```
"telephones fixed lines total subscriptions" :
"df1.loc[country]['data']['communications']['telephones']['fixed_lines']['

"telephones fixed lines subscriptions per one hundred inhabitants" :
"df1.loc[country]['data']['communications']['telephones']['fixed_lines']['

"telephones fixed lines source year" :
"df1.loc[country]['data']['communications']['telephones']['fixed_lines']['

"telephones mobile cellular total subscriptions" :
"df1.loc[country]['data']['communications']['telephones']['mobile_cellula

"telephones mobile cellular subscriptions per one hundred inhabitants" :
"df1.loc[country]['data']['communications']['telephones']['mobile_cellula

"telephones mobile cellular source year" :
"df1.loc[country]['data']['communications']['telephones']['mobile_cellula

"internet users total":
"df1.loc[country]['data']['communications']['internet']['users']['total']

"internet users pc of pop":
"df1.loc[country]['data']['communications']['internet']['users']['percent

"internet users source year":
"df1.loc[country]['data']['communications']['internet']['users']['date']"

# TRANSPORTATION

"national air transport nr of registered air carriers":
"df1.loc[country]['data']['transportation']['air_transport']['national_sy

"national air transport inventory of registered air carriers":
"df1.loc[country]['data']['transportation']['air_transport']['national_sy

"national air transport annual passenger traffic":
"df1.loc[country]['data']['transportation']['air_transport']['national_sy

"national air transport annual freight traffic":
"df1.loc[country]['data']['transportation']['air_transport']['national_sy

"total number of airports":
"df1.loc[country]['data']['transportation']['air_transport']['airports']['

"total number of airports source year":
"df1.loc[country]['data']['transportation']['air_transport']['airports']['

"number of airports with paved runways":
```

```
                    "df1.loc[country]['data']['transportation']['air_transport']['airports']['

                    "number of airports with unpaved runways":
                    "df1.loc[country]['data']['transportation']['air_transport']['airports']['

                    "total length of railways km":
                    "df1.loc[country]['data']['transportation']['railways']['total']['length']

                    "total length of railways km source year":
                    "df1.loc[country]['data']['transportation']['railways']['date']",

                    "total length of roadways km":
                    "df1.loc[country]['data']['transportation']['roadways']['total']['value']

                    "total length of roadways paved km":
                    "df1.loc[country]['data']['transportation']['roadways']['paved']['value']

                    "total length of roadways unpaved km":
                    "df1.loc[country]['data']['transportation']['roadways']['unpaved']['value

                    "total length of roadways km source year":
                    "df1.loc[country]['data']['transportation']['roadways']['date']",

                    "total length of waterways km":
                    "df1.loc[country]['data']['transportation']['waterways']['value']",

                    "total length of waterways km source year":
                    "df1.loc[country]['data']['transportation']['waterways']['date']",

                    "merchant marine total":
                    "df1.loc[country]['data']['transportation']['merchant_marine']['total']",

                    "merchant marine total source year":
                    "df1.loc[country]['data']['transportation']['merchant_marine']['date']",

                    # MILITARY EXPENDITURES

                    "military expenditures pc of gdp":
                    "df1.loc[country]['data']['military_and_security']['expenditures']['annual

                    "military expenditures pc of gdp source year":
                    "df1.loc[country]['data']['military_and_security']['expenditures']['annual

                   }

In [25]: str_data_dic = {
                    "residency requirement for naturalization in years":
                    "df1.loc[country]['data']['government']['citizenship']['residency_requirem
```

```
                    "suffrage (universal)":
                    "df1.loc[country]['data']['government']['suffrage']['universal']",
            }
```

```
In [26]: df1.loc[country]['data']['government']['suffrage']
```

```
Out[26]: {'age': 18, 'compulsory': False, 'universal': True}
```

```
In [27]: df1.loc[country]['data']['communications']['telephones']['mobile_cellular']
```

```
Out[27]: {'date': '2016-07-01',
          'global_rank': 72,
          'subscriptions_per_one_hundred_inhabitants': 89,
          'total': 12878926}
```

```
In [28]: df1.loc['United States']['data']['transportation']['merchant_marine']
```

```
Out[28]: {'by_type': [{'count': 5, 'type': 'bulk carrier'},
           {'count': 61, 'type': 'container ship'},
           {'count': 114, 'type': 'general cargo'},
           {'count': 66, 'type': 'oil tanker'},
           {'count': 3365, 'type': 'other'}],
          'date': '2017',
          'global_rank': 5,
          'total': 3611}
```

```
In [29]: eval("df1.loc[country]['data']['government']['citizenship']['residency_requirement_fo
```

```
Out[29]: '5 years'
```

```
In [30]: df1.loc[country]['data']['people']['maternal_mortality_rate']['deaths_per_100k_live_b
```

```
Out[30]: 443
```

```
In [31]: df1.loc['United States']['data']['people']['birth_rate']
```

```
Out[31]: {'births_per_1000_population': 12.5, 'date': '2017', 'global_rank': 158}
```

```
In [70]: # data_dic.keys()
```

```
In [33]: counter=0
         missing_data_df = pd.DataFrame(columns=data_dic.keys())
         missing_data_df.loc["number of countries without data"]=0
         for data in data_dic.keys():
             counter+=1
             if counter%20==0:
                 print(round(counter/len(data_dic.keys())*100,1), "%")
             missing_data_df.loc["number of countries without data"][data]=0

             countries[data]=np.nan
```

```
            for country in countries.index:
                try:
                    value = eval(data_dic[data])
                    countries.loc[country][data]=value

                except (KeyError, TypeError) as e:
                    missing_data_df[data] += 1

        print("100%")
```

```
7.5 %
15.1 %
22.6 %
30.2 %
37.7 %
45.3 %
52.8 %
60.4 %
67.9 %
75.5 %
83.0 %
90.6 %
98.1 %
100%
```

In [34]:
```
counter=0
missing_data_df = pd.DataFrame(columns=str_data_dic.keys())
missing_data_df.loc["number of countries without data"]=0
for data in str_data_dic.keys():
    counter+=1
    if counter%20==0:
        print(round(counter/len(data_dic.keys())*100,1), "%")
    missing_data_df.loc["number of countries without data"][data]=0

    countries[data]=np.dtype('|S1')
    for country in countries.index:
        try:
            value = eval(str_data_dic[data])
            countries.loc[country][data]=value

        except (KeyError, TypeError) as e:
            countries.loc[country][data]=None
            missing_data_df[data] += 1
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html

/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

```
In [35]: countries_wb.rename(index=country_name_dict, inplace=True)
```

```
In [36]: wb_df = pd.DataFrame(index=countries_wb.index)
```

```
In [37]: counter=0
         missing_data_df = pd.DataFrame(columns=data_dic.keys())
         missing_data_df.loc["number of countries without data"]=0
         for data in data_dic.keys():
             counter+=1
             if counter%20==0:
                 print(round(counter/len(data_dic.keys())*100,1), "%")
             missing_data_df.loc["number of countries without data"][data]=0

             wb_df[data]=np.nan
             for country in wb_df.index:
                 try:
                     value = eval(data_dic[data])
                     wb_df.loc[country][data]=value

                 except (KeyError, TypeError) as e:
                     missing_data_df[data] += 1

         print("100%")
```

```
7.5 %
15.1 %
22.6 %
30.2 %
37.7 %
45.3 %
52.8 %
60.4 %
67.9 %
75.5 %
83.0 %
90.6 %
98.1 %
100%
```

```
In [38]: countries_wb.head()
```

Out[38]:

| country | Country Data | Region | IncomeGroup |
|---|---|---|---|
| Afghanistan | AFG | South Asia | Low income |
| Albania | ALB | Europe & Central Asia | Upper middle income |
| Algeria | DZA | Middle East & North Africa | Upper middle income |
| American Samoa | ASM | East Asia & Pacific | Upper middle income |
| Andorra | AND | Europe & Central Asia | High income |

In [39]:
```
wb_df['Region'] = countries_wb['Region']
wb_df['IncomeGroup'] = countries_wb['IncomeGroup']
wb_df['Country Code'] = countries_wb['Country Data']
```

In [40]:
```
wb_df = wb_df.sort_values(['Region', 'Country Code'], axis=0)
```

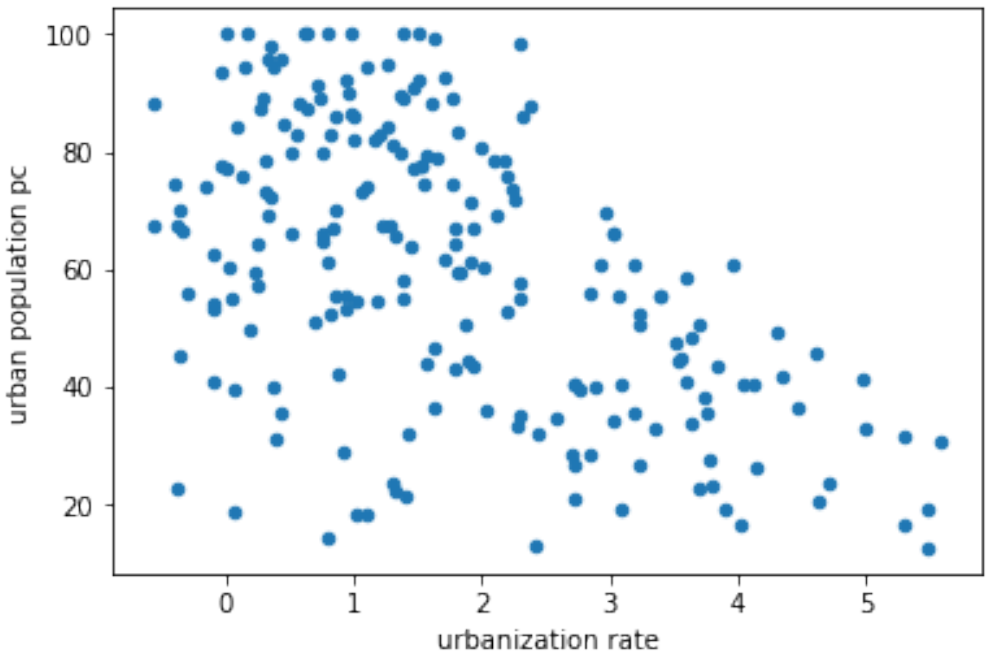
In [71]:
```
# wb_df.corr()['net migration rate per 1000'].sort_values()
```

In [73]:
```
# for column in wb_df.columns:
#     print(column)
```

In [43]:
```
%matplotlib inline

indicator1 = 'urbanization rate'
indicator2 = 'urban population pc'

df_to_plot =wb_df[np.abs(wb_df[indicator1]-wb_df[indicator1].mean())<=(3*wb_df[indica
df_to_plot = df_to_plot[np.abs(df_to_plot[indicator2]-df_to_plot[indicator2].mean())<

corr = df_to_plot[indicator1].corr(df_to_plot[indicator2])
print(corr)


df_to_plot.plot.scatter(x=indicator1 , y=indicator2,
                        #c='internet users total',
                        #colormap='viridis'
                        )
```

```
-0.517487147171
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x113d7f978>




```
In [44]: def get_corrs(df):
             col_correlations = df.corr()
             col_correlations.loc[:, :] = np.tril(col_correlations, k=-1)
             cor_pairs = col_correlations.stack()
             return cor_pairs.to_dict()

         my_corrs = get_corrs(wb_df)

In [74]: # my_corrs

In [46]: for country in countries.index:
             if country not in countries_wb.index:
                 print(country)

In [75]: # for country in countries_wb.index:
         #     if country not in countries.index:
         #         print(country)

In [48]: countries['Region'] = countries_wb['Region']
         countries['IncomeGroup'] = countries_wb['IncomeGroup']
         countries['Country Data'] = countries_wb['Country Data']

In [49]: countries.sort_values(['Region', 'Country Data'], axis=0, inplace=True)

In [51]: (1-countries.isnull().sum().sum()/(193*267))*100
```

```
Out[51]: 86.39265684733462

In [76]: # countries.head()

In [77]: # countries.tail()

In [55]: import xlsxwriter

In [56]: source_cols = [col for col in countries.columns if 'source year' in col]

In [78]: # countries.columns[~countries.columns.isin(source_cols)]

In [58]: LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

         def colToExcel(col): # col is 1 based
             excelCol = str()
             div = col
             while div:
                 (div, mod) = divmod(div-1, 26) # will return (x, 0 .. 25)
                 excelCol = chr(mod + 65) + excelCol
             return excelCol

In [59]: wb_df=wb_df.fillna(0.4242)

In [60]: writer = pd.ExcelWriter('processed_data\cia_world_factbook_data_wb_countries.xlsx', e
         workbook = writer.book

         source_cols = [col for col in wb_df.columns if 'source year' in col]

         wb_df[wb_df.columns[~wb_df.columns.isin(source_cols)]].to_excel(writer, sheet_name='d

         wb_df[source_cols].to_excel(writer, sheet_name='source')

         counter = 1
         for column in wb_df.columns[~wb_df.columns.isin(source_cols)]:
                 counter+=1
                 excel_col=colToExcel(counter)
                 excel_range = '=data!$'+excel_col+'$2:$'+excel_col+'$'+str(len(wb_df)+1)
                 column_name = column.replace(" ", "_").replace("(", "").replace(")","")
                 workbook.define_name(column_name , excel_range)


         # # Get the xlsxwriter objects from the dataframe writer object.
         # workbook  = writer.book
         # worksheet = writer.sheets['data']

In [61]: writer.close()
         workbook.close()

In [62]: countries = countries.fillna(0.4242)
```

```
In [64]: # writer = pd.ExcelWriter('processed_data/cia_world_factbook_data.xlsx', engine='xlsx
         # workbook = writer.book

         # source_cols = [col for col in countries.columns if 'source year' in col]

         # countries[countries.columns[~countries.columns.isin(source_cols)]].to_excel(writer,

         # countries[source_cols].to_excel(writer, sheet_name='source')

         # counter = 1
         # for column in countries.columns[~countries.columns.isin(source_cols)]:
         #         counter+=1
         #         excel_col=colToExcel(counter)
         #         excel_range = '=data!$'+excel_col+'$2:$'+excel_col+'$'+str(len(countries)+1
         #         column_name = column.replace(" ", "_").replace("(", "").replace(")","")
         #         workbook.define_name(column_name , excel_range)


         # # Get the xlsxwriter objects from the dataframe writer object.
         # workbook  = writer.book
         # worksheet = writer.sheets['data']

In [65]: missing_data_df.loc["number of countries without data"]=0

In [66]: %matplotlib inline

In [79]: # missing_data_df.T

In [68]: max(missing_data_df)

Out[68]: 'youth unemployment total'

In [69]: from bokeh.charts import Histogram, output_file, show
         from bokeh.sampledata.autompg import autompg as df

         p = Histogram(missing_data_df.T['number of countries without data'], title="Missing Da

         output_file("histogram.html",)

         show(p)

/anaconda3/lib/python3.6/site-packages/bokeh/util/deprecation.py:34: BokehDeprecationWarning:
The bokeh.charts API has moved to a separate 'bkcharts' package.

This compatibility shim will remain until Bokeh 1.0 is released.
After that, if you want to use this API you will have to install
the bkcharts package explicitly.

  warn(message)
```

```
E-1010 (CDSVIEW_SOURCE_DOESNT_MATCH): CDSView used by Glyph renderer must have a source that ma
E-1010 (CDSVIEW_SOURCE_DOESNT_MATCH): CDSView used by Glyph renderer must have a source that ma
E-1010 (CDSVIEW_SOURCE_DOESNT_MATCH): CDSView used by Glyph renderer must have a source that ma
```

## D.7. Fragile State Index

# 07 - Fragile State Index

August 15, 2018

In [1]:
```
import json
import pandas as pd
import csv
import numpy as np
import sys
```

In [2]:
```
countries = pd.ExcelFile("processed_data/country_list.xlsx")
countries = countries.parse()
countries.set_index('country', inplace=True)
```

In [3]:
```
countries_wb=pd.read_excel("raw data/Regions.xlsx", sheetname='Countries', header=0, sk
                           names=['Country Data','Region','IncomeGroup'],
                             parse_cols=None, parse_dates=False, date_parser=None, na_values
```

In [4]:
```
with open('processed_data/country_name_dic.csv', encoding = "UTF-8") as csvfile:
    reader = csv.DictReader(csvfile, delimiter = ',')
    country_name_dict = {rows['alt_name']:rows['name'] for rows in reader}
```

In [5]:
```
countries_wb_sort = countries_wb.sort_values(['Region', 'Country Data'], axis=0)
countries_wb_sort = countries_wb_sort.rename(index=country_name_dict)
```

In [6]:
```
fsi_data = countries_wb_sort
```

In [7]:
```
for year in range(2006, 2019, 1):
    file_loc = "raw data/Fragile State Index/fsi-"+str(year)+".xlsx"
    year_df = pd.ExcelFile(file_loc)
    year_df = year_df.parse()
    year_df.set_index('Country', inplace=True)
    col_list = year_df.columns
    year_dict={}
    for column in year_df.columns:
        year_dict[column]=column + " "+str(year)
    year_df = year_df.rename(index=country_name_dict, columns=year_dict)
    fsi_data = pd.concat([fsi_data, year_df], axis=1, join_axes=[fsi_data.index])
```

In [8]:
```
def func(x):
    if x.values[0] is None:
        return None
    else:
        return index_df.loc[x.name, x.values[0]]
```

In [9]:
```
index_dict = {}

        for column in col_list:
            index_df=None
            index_cols = [col for col in fsi_data.columns if column in col]

            index_df = fsi_data[fsi_data.columns[fsi_data.columns.isin(index_cols)]]
            index_df = index_df.T.drop_duplicates().T


            most_recent_value_colname = column + " Most Recent Value"

            index_df[most_recent_value_colname] = pd.DataFrame(index_df.apply(lambda x: x.last


            index_df['Region'] = fsi_data['Region']
            index_df['Country Code'] = fsi_data['Country Data']

            index_df = index_df.sort_values(['Region', 'Country Code'], axis=0)

            index_df.fillna(0.4242, inplace=True)

            cols = index_df.columns.tolist()
            cols.insert(0, cols.pop(cols.index('Region')))
            cols.insert(0, cols.pop(cols.index('Country Code')))
            cols.insert(0, cols.pop(cols.index(most_recent_value_colname)))
            index_df = index_df.reindex(columns= cols)

            index_dict[column] = index_df
```

In [10]:
```
# index_dict['C1: Security Apparatus']
```

In [11]:
```
index_dict.keys()
```

Out[11]:
```
dict_keys(['Year', 'Rank', 'Total', 'C1: Security Apparatus', 'C2: Factionalized Elit
```

In [12]:
```
for country in year_df.index:
            if country not in country_name_dict.keys():
                print(country)
```

In [13]:
```
for country in countries.index:
            if country not in year_df.index:
                print(country)
```

```
Andorra
Dominica
Kiribati
Liechtenstein
```

```
Marshall Islands
Monaco
Nauru
Palau
St. Kitts & Nevis
St. Lucia
St. Vincent & Grenadines
San Marino
Tonga
Tuvalu
Vanuatu
```

```
In [14]: for country in year_df.index:
             if country not in countries.index:
                 print(country)

In [15]: (1-fsi_data.isnull().sum().sum()/(193*195))*100

Out[15]: 76.40494220805103

In [16]: import xlsxwriter

In [17]: LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

         def colToExcel(col): # col is 1 based
             excelCol = str()
             div = col
             while div:
                 (div, mod) = divmod(div-1, 26) # will return (x, 0 .. 25)
                 excelCol = chr(mod + 65) + excelCol
             return excelCol

In [18]: index_dict=None
         for year in range(2006, 2019, 1):
             file_loc = "raw data/Fragile State Index/fsi-"+str(year)+".xlsx"

             year_df = pd.ExcelFile(file_loc)
             year_df = year_df.parse()
             year_df.set_index('Country', inplace=True)

             if index_dict!=None:
                 pass
             else:
                 index_dict={}
                 for column in year_df.columns:
                     index_dict[column]=pd.DataFrame(index=year_df.index)

             year_dict={}
```

```
            for column in year_df.columns:
                year_dict[column]=column + " "+str(year)

                new_df = pd.concat([index_dict[column], year_df], axis=1, join_axes=[index_di
                index_dict[column] = new_df
            year_df = year_df.rename(index=country_name_dict, columns=year_dict)
            fsi_data = pd.concat([fsi_data, year_df], axis=1, join_axes=[fsi_data.index])
```

In [19]:
```
index_dict.keys()
```

Out[19]:
```
dict_keys(['Year', 'Rank', 'Total', 'C1: Security Apparatus', 'C2: Factionalized Elit
```

In [20]:
```
def whatisthis(s):
    if isinstance(s, str):
        print("ordinary string")
    elif isinstance(s, unicode):
        print("unicode string")
    else:
        print("not a string")
```

In [28]:
```
try:
    writer.close()
except (NameError, AttributeError):
    pass
writer = pd.ExcelWriter('processed_data/fragile_state_index_data.xlsx', engine='xlsxw
workbook = writer.book

for index in index_dict.keys():

    sheet_name=_name = index.replace(" ", "_").replace("(", "").replace(")","").repla
#     print(sheet_name)
#     whatisthis(sheet_name)
#     print(sheet_name)

    df_to_write = index_dict[index].copy()
    df_to_write.rename(columns=lambda x: int(x[-4:]) if x[-4]=='2' else x , inplace=T
    df_to_write.to_excel(writer, sheet_name=sheet_name)

    counter = 1+3

    for column in index_dict[index].columns:
        if (column == 'Region' or column == 'Country Code'):
            continue
        elif " Most Recent Value" in column:
            excel_range = '='+sheet_name+'!$B$2:$B$'+str(len(index_dict[index])+1)
#             whatisthis(excel_range)
            column_name = column.replace(" ", "_").replace("(", "").replace(")","").re
```

```
            #             whatisthis(column_name)
            #             print(column_name, excel_range)
                        workbook.define_name(column_name, excel_range)
                    else:
                        counter+=1
                        excel_col = colToExcel(counter)
                        excel_range = '='+sheet_name+'!$'+excel_col+'$2:$'+excel_col+'$'+str(len(
            #             whatisthis(excel_range)
                        column_name = column.replace(" ", "_").replace("(", "").replace(")","").re
            #             whatisthis(excel_range)
            #             print(column_name, excel_range)
                        workbook.define_name(column_name, excel_range)



            # fsi_data.to_excel(writer, sheet_name='data')


            writer.save()

            # counter = 1
            # for column in fsi_data.columns:
            #         counter+=1
            #         excel_col=colToExcel(counter)
            #         excel_range = '=data!$'+excel_col+'$2:$'+excel_col+'$'+str(len(fsi_data)+1)
            #         column_name = column.replace(" ", "_").replace("(", "").replace(")","").rep
            #         print(column_name)
            #         workbook.define_name(column_name , excel_range)
```

In [29]:
```
# index_dict["C1: Security Apparatus"].head()
```

In [30]:
```
index_dict.keys()
```

Out[30]:
```
dict_keys(['Year', 'Rank', 'Total', 'C1: Security Apparatus', 'C2: Factionalized Elite
```

In [31]:
```
writer.close()
workbook.close()
writer = pd.ExcelWriter('processed_data/fragile_state_index_data.xlsx', engine='xlsxwr
workbook = writer.book
index_dict['C1: Security Apparatus'].to_excel(writer, sheet_name='1_Security_Apparatus

for column in index_dict['C1: Security Apparatus'].columns:
        if (column == 'Region' or column == 'Country Code'):
            continue
        if " Most Recent Value" in column:
            excel_range = '='+sheet_name+'!$B$2:$B$'+str(len(index_dict[index])+1)
            column_name = column.replace(" ", "_").replace("(", "").replace(")","").re
            workbook.define_name(column_name, excel_range)
```

```
                continue
            counter+=1
            excel_col = colToExcel(counter)
#           print(excel_col)
            excel_range = '='+'1_Security_Apparatus'+'!$'+excel_col+'$2:$'+excel_col+'$'+s
            column_name = column.replace(" ", "_").replace("(", "").replace(")","").repla
#           print(column_name, excel_range)
            workbook.define_name(column_name, excel_range)

    writer.save()
```

In [32]:
```
try:
    writer.close()
    workbook.close()
except (NameError, AttributeError):
    pass
writer = pd.ExcelWriter('processed_data/fragile_state_index_data.xlsx', engine='xlsxw
workbook = writer.book

for index in ['C1: Security Apparatus', 'C2: Factionalized Elites', 'C3: Group Grieva

    sheet_name=_name = index.replace(" ", "_").replace("(", "").replace(")","").repla
#     print(sheet_name)
    counter = 1+3

    for column in index_dict[index].columns:
        if (column == 'Region' or column == 'Country Code'):
            continue
        if " Most Recent Value" in column:
            excel_range = '='+sheet_name+'!$B$2:$B$'+str(len(index_dict[index])+1)
            column_name = column.replace(" ", "_").replace("(", "").replace(")","").re
            workbook.define_name(column_name, excel_range)
            continue
        counter+=1
        excel_col = colToExcel(counter)
#         print(excel_col)
        excel_range = '='+sheet_name+'!$'+excel_col+'$2:$'+excel_col+'$'+str(len(inde
        column_name = column.replace(" ", "_").replace("(", "").replace(")","").repla
#         print(column_name, excel_range)
        workbook.define_name(column_name, excel_range)

for index in ['C1: Security Apparatus', 'C2: Factionalized Elites', 'C3: Group Grieva
    sheet_name=_name = index.replace(" ", "_").replace("(", "").replace(")","").repla
    index_dict[index].to_excel(writer, sheet_name=sheet_name )



# fsi_data.to_excel(writer, sheet_name='data')
```

```
workbook.close()
writer.save()
```

## D.8. Political Instability Task Force

# 08 - Political Instability Task Force

August 15, 2018

In [1]:
```
import json
import pandas as pd
import csv
import numpy as np
```

In [2]:
```
countries = pd.ExcelFile("country_list.xlsx")
countries = countries.parse()
countries.set_index('country', inplace=True)
```

In [3]:
```
with open('processed_data/country_name_dic.csv') as csvfile:
    reader = csv.DictReader(csvfile, delimiter = ',')
    country_name_dict = {rows['alt_name']:rows['name'] for rows in reader}
```

In [4]:
```
pitf_data = pd.ExcelFile("raw data/Political Instability Task Force/pitf.world.2016010
pitf_data = pitf_data.parse()
```

In [5]:
```
pitf_data.head()
```

Out[5]:

| | "Disclaimer: This research was conducted for the Political Instability Task Force (P |
|---|---|
| 0 | EVENT TYPE AND REPORTING |
| 1 | Event Type |
| 2 | Incident |
| 3 | Incident |
| 4 | Incident |

| | Unnamed: 1 | Unnamed: 2 | Unnamed: 3 | Unnamed: 4 | \ |
|---|---|---|---|---|---|
| 0 | NaN | NaN | EVENT DATE | NaN | |
| 1 | Campaign Identifier | Event Reporting | Start Day | Start Month | |
| 2 | NaN | Eyewitness Account | 6 | 1 | |
| 3 | NaN | Eyewitness Account | 17 | 1 | |
| 4 | NaN | Eyewitness Account | 20 | 1 | |

| | Unnamed: 5 | Unnamed: 6 | Unnamed: 7 | Unnamed: 8 | Unnamed: 9 | ... | \ |
|---|---|---|---|---|---|---|---|
| 0 | NaN | NaN | NaN | NaN | EVENT LOCATION | ... | |
| 1 | Start Year | End Day | End Month | End Year | Country | ... | |
| 2 | 2016 | 99 | 99 | 9999 | AFG | ... | |
| 3 | 2016 | 99 | 99 | 9999 | AFG | ... | |
| 4 | 2016 | 99 | 99 | 9999 | AFG | ... | |

```
  Unnamed: 63             Unnamed: 64     Unnamed: 65  \
0        LINK             DATA SOURCE             NaN
1        Link     Primary Source Type  Primary Source
2         NaN                     BBC   International
3         NaN  BBC, AFP, Reuters, NYT   International
4         NaN   BBC, AFP, Reuters, AP   International

                                      Unnamed: 66       Unnamed: 67  \
0                                             NaN               NaN
1                          Secondary Source Type  Secondary Source
2        Afghan Islamic Press news agency, Peshawar             Local
3  Shamshad TV, Kabul, Afghan Channel One (1TV), ...             Local
4  Tolo TV, Kabul, Tolo News, Kabul, Voice of Jih...             Local

             Unnamed: 68        Unnamed: 69  \
0                    NaN                NaN
1  Contesting Source Type  Contesting Source
2                    NaN               None
3                    NaN               None
4                    NaN               None

                                      Unnamed: 70 Unnamed: 71 Unnamed: 72
0                                             NaN    COMMENTS         NaN
1                                        Citation    Comments       Coder
2                          BBCSAP0020160107ec170025t         NaN         PAS
3  BBCMNF0020160117ec1h000rv, AFPR000020160117ec1...         NaN         PAS
4  BBCMNF0020160120ec1k003e9, BBCMNF0020160120ec1...         NaN         PAS

[5 rows x 73 columns]
```

# D.9. Climate Data: Climate Research Unit

# 09 - Climate Data

August 15, 2018

In [ ]:
```
import json
import pandas as pd
import csv
import numpy as np
import io
import requests
from bs4 import BeautifulSoup
from io import StringIO
import re
```

In [ ]:
```
countries = pd.ExcelFile("country_list.xlsx")
countries = countries.parse()
countries.set_index('country', inplace=True)
```

In [ ]:
```
with open('processed_data/country_name_dic.csv') as csvfile:
    reader = csv.DictReader(csvfile, delimiter = ',')
    country_name_dict = {rows['alt_name']:rows['name'] for rows in reader}
```

In [ ]:
```
countries_wb=pd.read_excel("raw data/Regions.xlsx", sheetname='Countries', header=0, sk
                           names=['Country Data','Region','IncomeGroup'],
                             parse_cols=None, parse_dates=False, date_parser=None, na_values
```

In [ ]:
```
climate_data_df = pd.ExcelFile("raw data/Climate Data/Climate_country_table.xlsx")
climate_data_df = climate_data_df.parse(header=None)
```

In [ ]:
```
climate_data_df.columns = ['country','mean','tmp','pre','cld','dtr','vap']
```

In [ ]:
```
climate_data_df.set_index('country', inplace=True)
```

In [ ]:
```
climate_data_dic = {}

for country in climate_data_df.index:
#     print(country)
    country_url = country.replace(" ", "_").replace(".","")
    url="https://crudata.uea.ac.uk/~timm/cty/obs/data/obs."+country_url+".htm"

    res = requests.get(url)
    urlData = requests.get(url).content
```

```
            soup = BeautifulSoup(urlData, 'html.parser')
            texts = soup.find_all(text=True)

            for t in texts:
                newtext = t.replace(" ", "")
                newtext = t.replace("\n","")
                newtext = t.replace("\xa0","")
                newtext = t.replace("\r","")
                newtext = re.sub("\s+", ",", t.strip())
                t.replace_with(newtext)
            texts = soup.find_all(text=True)

            list_text = list(texts)
            new_list = []

            for i in range(len(list_text)):
                my_list = list_text[i].split(",")
                new_list.append(my_list)
            for i in range(4, 8, 1):
                new_list[i][0:2] = [''.join(new_list[i][0:2])]
            new_list[3]=['var',
                         'ann',
                         'MAM',
                         'JJA',
                         'SON',
                         'DJF' ,
                         'Jan', 'Feb','Mar','Apr','May','Jun','Jul', 'Aug', 'Sep', 'Oct', 'Nov', '

            df = pd.DataFrame(new_list[4:len(new_list)], columns = new_list[3])
            df.set_index('var', inplace=True)

            str_country = str(country)

            create_dic_expression = 'climate_data_dic["'+str_country+'"]=df'
            exec(create_dic_expression)
```

In [ ]:
```
climate_data_dic['Actaeon Group'].loc['Tmean']
```

In [ ]:
```
climate_data_df.head()
```

In [ ]:
```
tmp_dic = {}
for country in climate_data_df.index:
#     print(country)
    country_url = country.replace(" ", "_").replace(".","")
    url = "https://crudata.uea.ac.uk/~timm/cty/scen/data/tyn_cy_3_0."+country_url+".tm
#     print(url)
    res = requests.get(url)
    urlData = requests.get(url).content
```

```
        soup = BeautifulSoup(urlData, 'html.parser')
        texts = soup.find_all(text=True)
        for t in texts:
            newtext = t.replace(" ", "")
            newtext = t.replace("\n","")
            newtext = t.replace("\xa0","")
            newtext = t.replace("\r","")
            newtext = re.sub("\s+", ",", t.strip())
            t.replace_with(newtext)
        texts = soup.find_all(text=True)

        new_list = []


        for i in range(len(texts)):
            my_list = texts[0].split(",")
            new_list.append(my_list)

        scen_index = new_list[0].index('SCEN')
        ann_index = new_list[0].index('ANN')

        nr_columns = int(len(new_list[0][scen_index:ann_index+1]))
        nr_rows = int(len(new_list[0][ann_index+1:])/nr_columns)

        tmp = pd.DataFrame(np.array(new_list[0][ann_index+1:]).reshape(nr_rows, nr_columns)

        tmp.set_index('SCEN', inplace=True)

        str_country = str(country)

        create_dic_expression = 'tmp_dic["'+str_country+'"]=tmp'
        exec(create_dic_expression)


In [ ]: pre_dic = {}
        for country in climate_data_df.index:
        #     print(country)
            country_url = country.replace(" ", "_").replace(".","")
            url = "https://crudata.uea.ac.uk/~timm/cty/scen/data/tyn_cy_3_0."+country_url+".pr
        #     print(url)
            res = requests.get(url)
            urlData = requests.get(url).content
            soup = BeautifulSoup(urlData, 'html.parser')
            texts = soup.find_all(text=True)
            for t in texts:
                newtext = t.replace(" ", "")
                newtext = t.replace("\n","")
                newtext = t.replace("\xa0","")
```

```
        newtext = t.replace("\r","")
        newtext = re.sub("\s+", ",", t.strip())
        t.replace_with(newtext)
    texts = soup.find_all(text=True)

    new_list = []


    for i in range(len(texts)):
        my_list = texts[0].split(",")
        new_list.append(my_list)

    scen_index = new_list[0].index('SCEN')
    ann_index = new_list[0].index('ANN')

    nr_columns = int(len(new_list[0][scen_index:ann_index+1]))
    nr_rows = int(len(new_list[0][ann_index+1:])/nr_columns)

    pre = pd.DataFrame(np.array(new_list[0][ann_index+1:]).reshape(nr_rows, nr_columns)

    pre.set_index('SCEN', inplace=True)

    str_country = str(country)

    create_dic_expression = 'pre_dic["'+str_country+'"]=pre'
    exec(create_dic_expression)
```

In [ ]:
```
cld_dic = {}
for country in climate_data_df.index:
#     print(country)
    country_url = country.replace(" ", "_").replace(".","")
    url = "https://crudata.uea.ac.uk/~timm/cty/scen/data/tyn_cy_3_0."+country_url+".cl
#     print(url)
    res = requests.get(url)
    urlData = requests.get(url).content
    soup = BeautifulSoup(urlData, 'html.parser')
    texts = soup.find_all(text=True)
    for t in texts:
        newtext = t.replace(" ", "")
        newtext = t.replace("\n","")
        newtext = t.replace("\xa0","")
        newtext = t.replace("\r","")
        newtext = re.sub("\s+", ",", t.strip())
        t.replace_with(newtext)
    texts = soup.find_all(text=True)

    new_list = []
```

```
        for i in range(len(texts)):
            my_list = texts[0].split(",")
            new_list.append(my_list)

        scen_index = new_list[0].index('SCEN')
        ann_index = new_list[0].index('ANN')

        nr_columns = int(len(new_list[0][scen_index:ann_index+1]))
        nr_rows = int(len(new_list[0][ann_index+1:])/nr_columns)

        cld = pd.DataFrame(np.array(new_list[0][ann_index+1:]).reshape(nr_rows, nr_columns

        cld.set_index('SCEN', inplace=True)

        str_country = str(country)

        create_dic_expression = 'cld_dic["'+str_country+'"]=cld'
        exec(create_dic_expression)
```

In [ ]:
```
dtr_dic = {}
for country in climate_data_df.index:
#     print(country)
    country_url = country.replace(" ", "_").replace(".","")
    url = "https://crudata.uea.ac.uk/~timm/cty/scen/data/tyn_cy_3_0."+country_url+".dt
#     print(url)
    res = requests.get(url)
    urlData = requests.get(url).content
    soup = BeautifulSoup(urlData, 'html.parser')
    texts = soup.find_all(text=True)
    for t in texts:
        newtext = t.replace(" ", "")
        newtext = t.replace("\n","")
        newtext = t.replace("\xa0","")
        newtext = t.replace("\r","")
        newtext = re.sub("\s+", ",", t.strip())
        t.replace_with(newtext)
    texts = soup.find_all(text=True)

    new_list = []


    for i in range(len(texts)):
        my_list = texts[0].split(",")
        new_list.append(my_list)
```

```
        scen_index = new_list[0].index('SCEN')
        ann_index = new_list[0].index('ANN')

        nr_columns = int(len(new_list[0][scen_index:ann_index+1]))
        nr_rows = int(len(new_list[0][ann_index+1:])/nr_columns)

        dtr = pd.DataFrame(np.array(new_list[0][ann_index+1:]).reshape(nr_rows, nr_columns

        dtr.set_index('SCEN', inplace=True)

        str_country = str(country)

        create_dic_expression = 'dtr_dic["'+str_country+'"]=dtr'
        exec(create_dic_expression)
```

In [ ]:
```
vap_dic = {}
for country in climate_data_df.index:
#     print(country)
    country_url = country.replace(" ", "_").replace(".","")
    url = "https://crudata.uea.ac.uk/~timm/cty/scen/data/tyn_cy_3_0."+country_url+".vap
#     print(url)
    res = requests.get(url)
    urlData = requests.get(url).content
    soup = BeautifulSoup(urlData, 'html.parser')
    texts = soup.find_all(text=True)
    for t in texts:
        newtext = t.replace(" ", "")
        newtext = t.replace("\n","")
        newtext = t.replace("\xa0","")
        newtext = t.replace("\r","")
        newtext = re.sub("\s+", ",", t.strip())
        t.replace_with(newtext)
    texts = soup.find_all(text=True)

    new_list = []


    for i in range(len(texts)):
        my_list = texts[0].split(",")
        new_list.append(my_list)

    scen_index = new_list[0].index('SCEN')
    ann_index = new_list[0].index('ANN')

    nr_columns = int(len(new_list[0][scen_index:ann_index+1]))
    nr_rows = int(len(new_list[0][ann_index+1:])/nr_columns)
```

```
            vap = pd.DataFrame(np.array(new_list[0][ann_index+1:]).reshape(nr_rows, nr_columns)

            vap.set_index('SCEN', inplace=True)

            str_country = str(country)

            create_dic_expression = 'vap_dic["'+str_country+'"]=vap'
            exec(create_dic_expression)
```

In [ ]:
```
print(nr_rows, nr_columns)
```

In [ ]:
```
288/16
```

In [ ]:
```
tmp = pd.DataFrame(np.array(new_list[0][ann_index+1:]).reshape(nr_rows, nr_columns), c
```

In [ ]:
```
tmp.head()
```

In [ ]:
```
url="https://crudata.uea.ac.uk/~timm/cty/scen/scaler.ann"
```

In [ ]:
```
res = requests.get(url)
```

In [ ]:
```
urlData = requests.get(url).content
```

In [ ]:
```
soup = BeautifulSoup(urlData, 'html.parser')
```

In [ ]:
```
texts = soup.find_all(text=True)
```

In [ ]:
```
for t in texts:
            newtext = t.replace(" ", "")
            newtext = t.replace("\n","")
            newtext = t.replace("\xa0","")
            newtext = t.replace("\r","")
            newtext = re.sub("\s+", ",", t.strip())
            t.replace_with(newtext)
        texts = soup.find_all(text=True)
```

In [ ]:
```
new_list = []

        for i in range(len(texts)):
            my_list = texts[i].split(",")
            new_list.append(my_list)
```

In [ ]:
```
nr_columns = int(len(new_list[0][36:53]))
        nr_rows = int(len(new_list[0][53:])/len(new_list[0][36:53]))
```

In [ ]:
```
print(nr_columns, nr_rows)
```

In [ ]:
```
scalers = pd.DataFrame(np.array(new_list[0][53:]).reshape(nr_rows, nr_columns), column
        scalers.set_index("YEAR", inplace=True)
```

```
In [ ]: scalers
```

```
In [ ]: GCM_SRES_scenario_dic = {
            "CGCM2a1" : 1,
            "CGCM2a2" : 2,
            "CGCM2b1" : 4,
            "CGCM2b2" : 3,
            "CSIRO2a1" : 5,
            "CSIRO2a2" : 6,
            "CSIRO2b1" : 8,
            "CSIRO2b2" : 7,
            "HadCM3a1" : 9,
            "HadCM3a2" : 10,
            "HadCM3b1" : 12,
            "HadCM3b2" : 11,
            "PCMa1" : 13,
            "PCMa2" : 14,
            "PCMb1" : 16,
            "PCMb2" : 15
        }
```

```
In [ ]: scalers['PCMb2']
```

```
In [ ]: test_df = tmp_dic['Albania']
```

```
In [ ]: test_df.loc['1']
```

```
In [ ]: test_df.iloc[GCM_SRES_scenario_dic['PCMb2']]
```

```
In [ ]: scalers['PCMb2']
```

```
In [ ]: # for country in tmp_dic.keys():
        # #     print(country)
        #     country_df = tmp_dic[country]
        #     for scenario in GCM_SRES_scenario_dic.keys():
        #         x, y = None, None
        #         x=country_df.loc[str(GCM_SRES_scenario_dic[scenario])].convert_objects(conve
        #         y=scalers[scenario].convert_objects(convert_numeric=True)
        #         temp_df=None
        #         temp_df = pd.DataFrame(np.multiply.outer(y,x), columns = country_df.loc[str(

        #         z=climate_data_dic[country].loc['Tmean'].convert_objects(convert_numeric=Tru

        #         z.index = z.index.str.upper()
        #         z = z.reindex(index = x.index)

        #         temp_df = temp_df + z

        #         for year in temp_df.index:
        #             for month in temp_df.columns:
        #                 final_df.loc(axis=0)[country, year, month][scenario] = temp_df.loc[y
```

```
In [ ]: scenario = 'CGCM2a1'

In [ ]: %matplotlib inline

In [ ]: # final_df.unstack(level=1)['CGCM2a1'].unstack(level=1)

In [ ]: # df2 = final_df.reset_index()

In [ ]: # new_df = final_df.unstack()['CGCM2a1'].unstack(level=1)

In [ ]: # final_df.index.get_level_values(2)

In [ ]: # final_df.columns

In [ ]: month_cols = ["JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL","AUG","SEP","OCT","NOV",

In [ ]: # df3 = pd.DataFrame(final_df['CGCM2a1']).unstack(level=0).swaplevel(0, 1, axis=0).T[m

In [ ]: # df3.head()

In [ ]: # df4 = df3.reset_index()

In [ ]: # tmp_dic.keys()

In [ ]: for country in tmp_dic.keys():
            if country not in country_name_dict.keys():
                print(country)

In [ ]: df_key = pd.DataFrame(list(tmp_dic.keys()))
        df_key.set_index(0, inplace=True)
        df_key.rename(index=country_name_dict, inplace=True)
        countries_wb.rename(index=country_name_dict, inplace=True)
        for country in countries_wb.index:
            if country not in df_key.index:
                print(country)

In [ ]: countries_wb.index

In [ ]: # df4.index

In [ ]: # df4.rename(index = country_name_dict)

In [ ]: # df4.index

In [ ]: # df4.index

In [ ]: # scenario_dic_dfs = {}

        # for scenario in GCM_SRES_scenario_dic.keys():

        #     df3 = pd.DataFrame(final_df[scenario]).unstack(level=0).swaplevel(0, 1, axis=0).
```

```
        #     df4 = df3.reset_index()
        #     df4["Date"] = df4["Month"] + " " +df4["Year"].map(str)
        #     df4["Month Time"]=df4["Month"]
        #     df4.replace({"Month Time" : model_time_dic}, inplace=True)
        #     df4["Year"]=pd.to_numeric(df4["Year"])
        #     print(type(df4["Year"]), type(df4["Month Time"]))
        #     df4["Model Time"] = df4["Year"]+df4["Month Time"]

        #     df4 = df4.T

        #     df4.rename(index = country_name_dict, inplace=True)


        #     df4['Region'] = countries_wb['Region']
        #     df4['Country Code'] = countries_wb['Country Data']

        #     df4 = df4.sort_values(['Region', 'Country Code'], axis=0)

        #     scenario_dic_dfs[scenario] = df4
```

In [ ]:
```
# df4.sort_values(by='Country')
```

In [ ]:
```
scenario_dic_dfs.keys()
```

In [ ]:
```
country_name_dict['USA']
```

In [ ]:
```
# scenario_dic_dfs['CGCM2a2']
```

In [ ]:
```
# df5 = df4.sort_values('Model Time').T
```

In [ ]:
```
# df5.head()
```

In [ ]:
```
model_time_dic = {"JAN": 0+0.5*(1/12),
                  "FEB":1/12+0.5*(1/12),
                  "MAR":2/12+0.5*(1/12),
                  "APR":3/12+0.5*(1/12),
                  "MAY":4/12+0.5*(1/12),
                  "JUN":5/12+0.5*(1/12),
                  "JUL":6/12+0.5*(1/12),
                  "AUG":7/12+0.5*(1/12),
                  "SEP":8/12+0.5*(1/12),
                  "OCT":9/12+0.5*(1/12),
                  "NOV":10/12+0.5*(1/12),
                  "DEC":11/12+0.5*(1/12),
                   "MAM": 0,
                "JJA":0,
                "SON":0,
                "DJF":0,
                "ANN":0}
```

```
In [ ]: # type(df2["Year"][0])

In [ ]: # df2["Date"] = df2["Month"] + " " +df2["Year"].map(str)
        # df2["Month Time"]=df2["Month"]
        # df2.replace({"Month Time" : model_time_dic}, inplace=True)
        # df2["Year"]=pd.to_numeric(df2["Year"])
        # print(type(df2["Year"]), type(df2["Month Time"]))
        # df2["Model Time"] = df2["Year"]+df2["Month Time"]

In [ ]: # df2.pivot(columns='Country')

In [ ]: # pd.to_numeric(df2["Year"]).values

In [ ]: # df2.head

In [ ]: # plot_df = pd.DataFrame(final_df["CGCM2a1"].loc[:,:,"ANN"])

In [ ]: # plot_df2 = plot_df.unstack(level=0)

In [ ]: # plot_df2.T.head()

In [ ]: # plot_df2['CGCM2a1'].loc['2001'].sort_values(axis=0)

In [ ]: # plot_df2['CGCM2a1'].loc['2100'].sort_values(axis=0)

In [ ]: # plot_df2.plot(x=plot_df2.index, y=plot_df2.columns, legend=False)

In [ ]: # final_df.loc[:,:,slice("AUG")]

In [ ]: # plot_df.loc["JAN"]

In [ ]: import matplotlib as plt
        plt.rcParams['savefig.facecolor'] = "0.8"

In [ ]: # plot_df.plot(x=plot_df.index, y=plot_df.columns, legend=False)

In [ ]: # final_df.loc["Zimbabwe"]

In [ ]: # df_afgh.loc['2001']['JAN']

In [ ]: # country_df = tmp_dic['Afghanistan']

In [ ]: # country_df.iloc[GCM_SRES_scenario_dic[scenario]]["JAN":"DEC"]

In [ ]: # x=country_df.iloc[GCM_SRES_scenario_dic[scenario]].convert_objects(convert_numeric=T

In [ ]: y=scalers[scenario].convert_objects(convert_numeric=True)

In [ ]: z=climate_data_dic['Afghanistan'].loc['Tmean'].convert_objects(convert_numeric=True)

In [ ]: type(z)
```

```
In [ ]: z.index = z.index.str.upper()
        z = z.reindex(index = x.index)

In [ ]: z.head()

In [ ]: df_afgh = pd.DataFrame(np.multiply.outer(y,x), columns = country_df.iloc[GCM_SRES_scena

In [ ]: climate_data_dic['Afghanistan'].loc['Tmean']

In [ ]: (df_afgh+z).head()

In [ ]: %matplotlib inline

In [ ]: df_afgh.head()

In [ ]: df_afgh.plot(x=df_afgh.index, y='AUG')

In [ ]: idx = pd.MultiIndex.from_product([climate_data_df.index,
                                          scalers.index,
                                         country_df.iloc[GCM_SRES_scenario_dic[scenario]].index
                                          names=['Country', 'Year','Month'])
        cols = GCM_SRES_scenario_dic.keys()

In [ ]: country_df.iloc[GCM_SRES_scenario_dic[scenario]].index

In [ ]: final_df = pd.DataFrame(pd.DataFrame(np.nan, idx, cols))

In [ ]: scalers.index

In [ ]: final_df.loc(axis=0)['Afghanistan','2001','JAN']['CGCM2a1'] = -10

In [ ]: final_df.loc(axis=0)['Afghanistan', '2001']

In [ ]: climate_data_dic['Afghanistan'].loc['Tmean']
```

## D.10. Climate Data: World Bank

# 10 - Climate Data World Bank

August 15, 2018

In [1]:
```
import json
import pandas as pd
import csv
import numpy as np
import wbpy
```

In [2]:
```
countries_wb=pd.read_excel("raw data/Regions.xlsx", sheetname='Countries', header=0, sk
                           names=['Country Data','Region','IncomeGroup'],
                             parse_cols=None, parse_dates=False, date_parser=None, na_values
```

In [3]:
```
c_api = wbpy.ClimateAPI()

c_api.ARG_DEFINITIONS["instrumental_types"]
```

Out[3]:
```
{'pr': 'Precipitation (rainfall and assumed water equivalent), in millimeters',
 'tas': 'Temperature, in degrees Celsius'}
```

In [4]:
```
c_api.ARG_DEFINITIONS["instrumental_intervals"]
```

Out[4]:
```
['year', 'month', 'decade']
```

In [5]:
```
c_api.ARG_DEFINITIONS["modelled_types"]
```

Out[5]:
```
{'ppt_days': 'Number of days with precipitation > 0.2mm',
 'ppt_days10': 'Number of days with precipitation > 10mm',
 'ppt_days2': 'Number of days with precipitation > 2mm',
 'ppt_days90th': "Number of days with precipitation > the control period's 90th percen
 'ppt_dryspell': 'Average number of days between precipitation events',
 'ppt_means': 'Average daily precipitation',
 'pr': 'Precipitation (rainfall and assumed water equivalent), in millimeters',
 'tas': 'Temperature, in degrees Celsius',
 'tmax_days10th': "Number of days with max temperature below the control period's 10th
 'tmax_days90th': "Number of days with max temperature above the control period's 90th
 'tmax_means': 'Average daily maximum temperature, Celsius',
 'tmin_days0': 'Number of days with min temperature below 0 degrees Celsius',
 'tmin_days10th': "Number of days with min temperature below the control period's 10th
 'tmin_days90th': "Number of days with min temperature above the control period's 90th
 'tmin_means': 'Average daily minimum temperature, Celsius'}
```

```
In [6]: c_api.ARG_DEFINITIONS["modelled_intervals"]

Out[6]: {'aanom': 'Average annual change (anomaly).',
         'aavg': 'Annual average',
         'annualanom': 'Average annual change (anomaly).',
         'annualavg': 'Annual average',
         'manom': 'Average monthly change (anomaly).',
         'mavg': 'Monthly average'}

In [7]: countries = list(countries_wb['Country Data'].values)

In [8]: countries1=countries[0:10]

In [9]: for country in countries1:
            print(country)
            modelled_dataset = c_api.get_modelled("tas", "mavg", [country])

AFG
ALB
DZA
ASM
AND
AGO
ATG
ARG
ARM
ABW


In [10]: model_list = list(modelled_dataset.as_dict().keys())
         scenario_list = ["a2","b2"]

In [11]: countries1=["AFG","CHI"]

In [14]: wb_dict = countries_wb['Country Data'].to_dict()

         reverse_dict={}
         for key in wb_dict.keys():
             reverse_dict[wb_dict[key]] = key

In [15]: time = np.linspace(1939+1/12/2, 2100-1/12/2, (2100-1939)*12)

         full_results_dic = {}
         for scenario in scenario_list:
             full_results_dic[scenario] = {}
         for scenario in scenario_list:
             for model in model_list:
                 full_results_dic[scenario][model] = pd.DataFrame(columns=time)
```

```
for country in countries1:
    print(country)
    try:
        modelled_dataset = c_api.get_modelled("tas", "mavg", [country])
    except ValueError:
        new_df = pd.DataFrame(index=range(12), columns=range(1939, 2100))
        inter_df = new_df
        inter_df=inter_df.melt().T
        inter_df.columns = time
        inter_df = inter_df.drop('variable')
        inter_df.rename(index={'value': country}, inplace=True)
        inter_df = inter_df.fillna(0.4242)
        for model in model_list:
            for scenario in scenario_list:
                full_results_dic[scenario][model] = full_results_dic[scenario][model]
        continue
    for model in model_list:
        for scenario in scenario_list:
            model_df = pd.DataFrame(pd.DataFrame(modelled_dataset.as_dict(scenario)).
            new_df = pd.DataFrame(index=range(12), columns=range(1939, 2100))

            model_df.columns= model_df.columns.astype(str)
            new_df.columns = new_df.columns.astype(str)
            new_df = pd.merge(new_df,model_df, how = 'right')

            new_df.columns = new_df.columns.astype(float)
            new_df = new_df.astype(float)

            new_df.interpolate(axis=0, method='time')
            trans_df = new_df.T
            inter_df = trans_df.interpolate().T
            inter_df=inter_df.melt().T

            inter_df.columns = time

            inter_df = inter_df.drop('variable')
            inter_df.rename(index={'value': country}, inplace=True)
            full_results_dic[scenario][model] = full_results_dic[scenario][model].appe

for model in model_list:
    for scenario in scenario_list:
        full_results_dic[scenario][model]=full_results_dic[scenario][model].rename(inc
        full_results_dic[scenario][model]['Region'] =countries_wb['Region']
        full_results_dic[scenario][model]['Country Code'] =countries_wb['Country Data
        full_results_dic[scenario][model]['Income Level'] =countries_wb['IncomeGroup']

        cols=list(full_results_dic[scenario][model].columns)
        cols.insert(0, cols.pop(cols.index('Region')))
```

```
                cols.insert(0, cols.pop(cols.index('Country Code')))
                cols.insert(0, cols.pop(cols.index('Income Level')))
                full_results_dic[scenario][model] = full_results_dic[scenario][model].loc[:, c
```

```
AFG
CHI
```

In [17]:
```
# writer.close()

# writer = pd.ExcelWriter('processed_data/climate_data_wb_temp.xlsx')

# for scenario in scenario_list:
#     for model in model_list:
#         print(scenario, model)
#         tabname = scenario +"_"+model
#         df_to_write = full_results_dic[scenario][model]
#         df_to_write.to_excel(writer, tabname)
```

In [18]:
```
# writer.save()
```

In [19]:
```
len(full_results_dic['a2'].keys())
```

Out[19]: 18

In [21]:
```
# full_results_dic['a2']['bccr_bcm2_0']
```

In [22]:
```
#precipitation

time = np.linspace(1939+1/12/2, 2100-1/12/2, (2100-1939)*12)

full_results_dic_precipitation = {}
for scenario in scenario_list:
    full_results_dic_precipitation[scenario] = {}
for scenario in scenario_list:
    for model in model_list:
        full_results_dic_precipitation[scenario][model] = pd.DataFrame(columns=time)

for country in countries1:
    print(country)
    try:
        modelled_dataset = c_api.get_modelled("pr", "mavg", [country])
    except ValueError:
        new_df = pd.DataFrame(index=range(12), columns=range(1939, 2100))
        inter_df = new_df
        inter_df=inter_df.melt().T
```

```
                inter_df.columns = time
                inter_df = inter_df.drop('variable')
                inter_df.rename(index={'value': country}, inplace=True)
                inter_df = inter_df.fillna(0.4242)
                for model in model_list:
                    for scenario in scenario_list:
                        full_results_dic_precipitation[scenario][model] = full_results_dic_pr
                continue
            for model in model_list:
                for scenario in scenario_list:
                    model_df = pd.DataFrame(pd.DataFrame(modelled_dataset.as_dict(scenario)).
                    new_df = pd.DataFrame(index=range(12), columns=range(1939, 2100))

                    model_df.columns= model_df.columns.astype(str)
                    new_df.columns = new_df.columns.astype(str)
                    new_df = pd.merge(new_df,model_df, how = 'right')

                    new_df.columns = new_df.columns.astype(float)
                    new_df = new_df.astype(float)

                    new_df.interpolate(axis=0, method='time')
                    trans_df = new_df.T
                    inter_df = trans_df.interpolate().T
                    inter_df=inter_df.melt().T

                    inter_df.columns = time

                    inter_df = inter_df.drop('variable')
                    inter_df.rename(index={'value': country}, inplace=True)
                    full_results_dic_precipitation[scenario][model] = full_results_dic_precip

        for model in model_list:
            for scenario in scenario_list:
                full_results_dic_precipitation[scenario][model]=full_results_dic_precipitatio
                full_results_dic_precipitation[scenario][model]['Region'] =countries_wb['Regi
                full_results_dic_precipitation[scenario][model]['Country Code'] =countries_wb
                full_results_dic_precipitation[scenario][model]['Income Level'] =countries_wb

                cols=list(full_results_dic_precipitation[scenario][model].columns)
                cols.insert(0, cols.pop(cols.index('Region')))
                cols.insert(0, cols.pop(cols.index('Country Code')))
                cols.insert(0, cols.pop(cols.index('Income Level')))
                full_results_dic_precipitation[scenario][model] = full_results_dic_precipitat


AFG
CHI
```

In [23]:
```
# writer.close()

# writer = pd.ExcelWriter('processed_data/climate_data_wb_precipitation.xlsx')

# for scenario in scenario_list:
#     for model in model_list:
#         print(scenario, model)
#         tabname = scenario +"_"+model
#         df_to_write = full_results_dic_precipitation[scenario][model]
#         df_to_write.to_excel(writer, tabname)
```

# D.11. Ethnic Groups

# 11 - Ethnic Groups

August 15, 2018

In [1]:
```
import json
import pandas as pd
import csv
import numpy as np
import wbpy
import io
import requests
import bs4
from bs4 import BeautifulSoup
import math
from itertools import combinations, product
```

In [2]:
```
countries_wb=pd.read_excel("raw data/Regions.xlsx", sheetname='Countries', header=0, sk
                           names=['Country Data','Region','IncomeGroup'],
                             parse_cols=None, parse_dates=False, date_parser=None, na_values
```

In [3]:
```
with open('processed_data/iso2_name_dic.csv', encoding = "UTF-8") as csvfile:
    reader = csv.DictReader(csvfile, delimiter = ',')
    iso2_name_dict = {rows['iso2code']:rows['name'] for rows in reader}
```

In [4]:
```
with open('processed_data/country_name_dic.csv', encoding = "UTF-8") as csvfile:
    reader = csv.DictReader(csvfile, delimiter = ',')
    country_name_dict = {rows['alt_name']:rows['name'] for rows in reader}
```

In [5]:
```
json1_file = open('raw data/CIA World Factbook/2018-04-30_factbook.json')
json1_str = json1_file.read()
```

In [6]:
```
json1_data = json.loads(json1_str)
```

In [8]:
```
# json1_data["countries"]["Congo"]["data"]['people']['ethnic_groups']['ethnicity']
```

In [9]:
```
df = pd.DataFrame.from_dict({(i,j): json1_data[i][j]
                             for i in json1_data.keys()
                             for j in json1_data[i].keys()})
```

In [10]:
```
df1=df.loc["data"]["countries"]
```

In [11]:
```
for country in df1.index:
    #     print(df1[country]["name"])
    #     print(country)
        df1.rename(index={country : df1.loc[country]["name"]}, inplace=True)
```

```
In [12]: df1 = pd.DataFrame(df1.rename(index=country_name_dict))

In [13]: countries_wb = countries_wb.sort_values(['Region', 'Country Data'])

In [14]: countries_wb.rename(index=country_name_dict, inplace=True)

In [15]: ethnicity_df = pd.DataFrame(index = countries_wb.index)

In [16]: for country in countries_wb.index:
             if country not in df1.index:
                 print(country)

Channel Islands
Saint Martin (French part)
U.S. Virgin Islands


In [17]: df1.loc["Gabon"]['data']['people']['ethnic_groups']['ethnicity']

Out[17]: [{'name': 'Bantu tribes'},
          {'name': 'including four major tribal groupings ; other Africans and Europeans',
           'note': 'Fang, Bapounou, Nzebi, Obamba'},
          {'percent': 154},
          {'percent': 0},
          {'name': 'including', 'percent': 10},
          {'name': 'French and', 'percent': 11},
          {'name': 'persons of dual nationality', 'percent': 0}]

In [18]: df1.loc['Gabon']['data']['people']['ethnic_groups']['ethnicity']

Out[18]: [{'name': 'Bantu tribes'},
          {'name': 'including four major tribal groupings ; other Africans and Europeans',
           'note': 'Fang, Bapounou, Nzebi, Obamba'},
          {'percent': 154},
          {'percent': 0},
          {'name': 'including', 'percent': 10},
          {'name': 'French and', 'percent': 11},
          {'name': 'persons of dual nationality', 'percent': 0}]

In [19]: df1.loc['Gabon']['data']['people']['ethnic_groups']['ethnicity']=[x for x in df1.loc[

In [20]: for country in ethnicity_df.index:
             try:
                 for ethnicity in df1.loc[country]['data']['people']['ethnic_groups']['ethnici

                     if ethnicity['name'] not in ethnicity_df.columns:
                         ethnicity_df[ethnicity['name']] = None


                     if 'percent' in ethnicity:
```

```
                    ethnicity_df.loc[country][ethnicity['name']]=ethnicity['percent']
                    #print(country)
                else:
                    #print(language)
                    ethnicity_df.loc[country][ethnicity['name']] = 100/len(df1.loc[countr
            except KeyError:
                pass

        for country in ethnicity_df.index:

            total_sum = ethnicity_df.loc[country].sum()

            if 95.0 <= total_sum <= 105.0:
                continue

            elif math.isnan(total_sum):
                print("No data for:", country)

            else:
                specified_percentages = 0
                ethnicities_wo_percentage = []

                df1.loc[country]['data']['people']['ethnic_groups']['ethnicity']=[x for x in

                for ethnicity in df1.loc[country]['data']['people']['ethnic_groups']['ethnici
                    if 'percent' in ethnicity:
                        if ethnicity['percent']>100:
                            print(country)
                            print('Value for', ethnicity['name'], 'in', country, 'is over 100
                            ethnicities_wo_percentage.append(ethnicity['name'])
                        else:
                            specified_percentages += ethnicity['percent']

                    else:
                        ethnicities_wo_percentage.append(ethnicity['name'])
        #                 print(country, counter, languages_wo_percentage, specified_percenta

                if (specified_percentages <=95 and len(ethnicities_wo_percentage)==0):
                    for ethnicity in df1.loc[country]['data']['people']['ethnic_groups']['eth
                        ethnicity_df.loc[country][ethnicity['name']] = (ethnicity['percent']/

                for ethnicity in ethnicities_wo_percentage:
                    ethnicity_df.loc[country][ethnicity] = (100-specified_percentages)/len(et
```

```
No data for: Channel Islands
Portugal
Value for 000; since in Portugal is over 100%. Scaling down in process...
```

```
No data for: Saint Martin (French part)
No data for: Sint Maarten
No data for: U.S. Virgin Islands
No data for: Palestinian Territories
Congo - Kinshasa
Value for over African ethnic groups of which the majority are Bantu; the four largest tribes
```

```
In [21]: # ethnicity_df
```

```
In [22]: # No data for: Channel Islands
         # No data for: Saint Martin (French part)
         # No data for: Sint Maarten
         # No data for: U.S. Virgin Islands
         # No data for: Palestinian Territories

         ethnicity_df.loc['Channel Islands']['French'] = 50
         ethnicity_df.loc['Channel Islands']['English'] = 50
         ethnicity_df.loc['Saint Martin (French part)']['French'] = 10
         ethnicity_df.loc['Saint Martin (French part)']['Afro-Caribbean'] = 90
         ethnicity_df.loc['Sint Maarten']['Dutch'] = 10
         ethnicity_df.loc['Sint Maarten']['Afro-Caribbean'] = 90
         ethnicity_df.loc['U.S. Virgin Islands']['black/African American'] = 76
         ethnicity_df.loc['U.S. Virgin Islands']['American'] = 15.7
         ethnicity_df.loc['U.S. Virgin Islands']['Asian'] = 1.4
         ethnicity_df.loc['Palestinian Territories']['Arab'] = 78.5
         ethnicity_df.loc['Palestinian Territories']['Jewish'] = 21.5
```

```
In [23]: [col for col in ethnicity_df.columns if 'Jew' in col]
```

```
Out[23]: ['Jewish', 'Jewish minorities', 'non-Jewish', 'Jewish and other']
```

```
In [24]: ethnicity_df1 = ethnicity_df.copy()
```

```
In [25]: test_list = ['Native', 'Hawaiian', 'or', 'other', 'Pacific', 'Islander']
```

```
In [26]: def merge(x):
             tmp = []
             for i in range(len(x)):
                 if (x[i][0].isupper and x[i+1][0].isupper):
                     print('yes')
                     yield ' '.join(tmp)
                     tmp = []
                     tmp.append(x[i] +" "+ x[i+1])
                     print(tmp)
             if len(tmp):
                 yield ' '.join(tmp)
```

```
In [27]: test_list
```

```
Out[27]: ['Native', 'Hawaiian', 'or', 'other', 'Pacific', 'Islander']

In [29]: # merge_list(test_list)

In [30]: def merge_list(input_list):
             new_list = []
             counter = 0
             for i in input_list[:-1]:
                 index_i = input_list.index(i)
              #   print(i)
                 if index_i != 0:
                #     print(i in new_list[index_i-1-counter])
                     if i in input_list[index_i-1-counter]:
                       #  print('yes')
                         continue
                 if (i[0].isupper() and input_list[index_i+1][0].isupper()):
                     combined_string=input_list[index_i]+" "+input_list[index_i+1]
                     new_list.append(combined_string)
                     counter+=1
                 else:
                     new_list.append(i)
             if len(new_list) == 0:
                 print("now")
                 new_list.append(input_list[-1])

             to_remove=[]
             for i in new_list:
                 if not i[0].isupper():

                     to_remove.append(i)
             for i in to_remove:
                 new_list.remove(i)

             return new_list

In [31]: for column in ethnicity_df1.columns:
             counter = 0
             if len(column.split())>1:
                 column_list = column.split()
         #         print(1, column_list)
                 new_list = merge_list(column_list)
         #         print(2, new_list)

         #         ethnicity_list = []
         #         for item in column_list:
         #             if item[0].isupper() and item in language_df1.columns:
         #                 counter+=1
         #                 language_list.append(item)
```

```
#         for language in language_list:
#             print(language, "-", column)
#             language_df1[language] = language_df1[language]+(language_df1[column]/l

#     if counter!=0:
#         language_df1.drop(column, axis=1, inplace = True)

# for column in language_df1.columns:
#     if column[0].islower():
#         language_df1.drop(column, axis=1, inplace = True)
```

## D.12. Language Similarity

# 12 - Language Data

August 15, 2018

```
In [1]: import json
        import pandas as pd
        import csv
        import numpy as np
        import wbpy
        import io
        import requests
        import bs4
        from bs4 import BeautifulSoup
        import math
        from itertools import combinations, product
```

```
In [2]: countries_wb=pd.read_excel("raw data/Regions.xlsx", sheetname='Countries', header=0, sk
                                  names=['Country Data','Region','IncomeGroup'],
                                    parse_cols=None, parse_dates=False, date_parser=None, na_values
```

```
In [3]: with open('processed_data/iso2_name_dic.csv', encoding = "UTF-8") as csvfile:
            reader = csv.DictReader(csvfile, delimiter = ',')
            iso2_name_dict = {rows['iso2code']:rows['name'] for rows in reader}
```

```
In [4]: with open('processed_data/country_name_dic.csv', encoding = "UTF-8") as csvfile:
            reader = csv.DictReader(csvfile, delimiter = ',')
            country_name_dict = {rows['alt_name']:rows['name'] for rows in reader}
```

```
In [5]: json1_file = open('raw data/CIA World Factbook/2018-04-30_factbook.json')
        json1_str = json1_file.read()
```

```
In [6]: json1_data = json.loads(json1_str)
```

```
In [7]: json1_data["countries"]["zambia"]["data"]['people']['languages']
```

```
Out[7]: {'date': '2010',
         'language': [{'name': 'Bemba', 'percent': 33.4},
          {'name': 'Nyanja', 'percent': 14.7},
          {'name': 'Tonga', 'percent': 11.4},
          {'name': 'Lozi', 'percent': 5.5},
          {'name': 'Chewa', 'percent': 4.5},
          {'name': 'Nsenga', 'percent': 2.9},
```

```
          {'name': 'Tumbuka', 'percent': 2.5},
          {'name': 'Lunda', 'note': 'North Western', 'percent': 1.9},
          {'name': 'Kaonde', 'percent': 1.8},
          {'name': 'Lala', 'percent': 1.8},
          {'name': 'Lamba', 'percent': 1.8},
          {'name': 'English', 'note': 'official', 'percent': 1.7},
          {'name': 'Luvale', 'percent': 1.5},
          {'name': 'Mambwe', 'percent': 1.3},
          {'name': 'Namwanga', 'percent': 1.2},
          {'name': 'Lenje', 'percent': 1.1},
          {'name': 'Bisa', 'percent': 1},
          {'name': 'other', 'percent': 9.7},
          {'name': 'unspecified', 'percent': 0.2}],
         'note': "Zambia is said to have over 70 languages, although many of these may be consi
```

```
In [51]: # json1_data['countries'].keys()
```

```
In [9]: df = pd.DataFrame.from_dict({(i,j): json1_data[i][j]
                                    for i in json1_data.keys()
                                    for j in json1_data[i].keys()})
```

```
In [10]: df1=df.loc["data"]["countries"]
```

```
In [52]: for country in df1.index:
         #     print(df1[country]["name"])
         #     print(country)
             df1.rename(index={country : df1.loc[country]["name"]}, inplace=True)
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

/anaconda3/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_value(self, s
   2482             try:
-> 2483                 return libts.get_value_box(s, key)
   2484             except IndexError:


pandas/_libs/tslib.pyx in pandas._libs.tslib.get_value_box (pandas/_libs/tslib.c:18843)


pandas/_libs/tslib.pyx in pandas._libs.tslib.get_value_box (pandas/_libs/tslib.c:18477)


TypeError: 'str' object cannot be interpreted as an integer


During handling of the above exception, another exception occurred:
```

```
KeyError                                  Traceback (most recent call last)

<ipython-input-52-92ad6499776e> in <module>()
      2 #     print(df1[country]["name"])
      3 #     print(country)
----> 4     df1.rename(index={country : df1.loc[country]["name"]}, inplace=True)


/anaconda3/lib/python3.6/site-packages/pandas/core/series.py in __getitem__(self, key)
    599         key = com._apply_if_callable(key, self)
    600         try:
--> 601             result = self.index.get_value(self, key)
    602
    603             if not is_scalar(result):


/anaconda3/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_value(self, s
   2489                     raise InvalidIndexError(key)
   2490                 else:
-> 2491                     raise e1
   2492             except Exception:  # pragma: no cover
   2493                 raise e1


/anaconda3/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_value(self, s
   2475         try:
   2476             return self._engine.get_value(s, k,
-> 2477                                           tz=getattr(series.dtype, 'tz', None))
   2478         except KeyError as e1:
   2479             if len(self) > 0 and self.inferred_type in ['integer', 'boolean']:


pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_value()


pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_value()


pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.ge


pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.ge
```

```
        KeyError: 'name'
```

```
In [12]: df1 = pd.DataFrame(df1.rename(index=country_name_dict))

In [13]: countries_wb = countries_wb.sort_values(['Region', 'Country Data'])

In [14]: countries_wb.rename(index=country_name_dict, inplace=True)

In [15]: language_df = pd.DataFrame(index = countries_wb.index)

In [16]: for country in countries_wb.index:
             if country not in df1.index:
                 print(country)

Channel Islands
Saint Martin (French part)
U.S. Virgin Islands
```

```
In [17]: for country in language_df.index:
             try:
                 for language in df1.loc[country]['data']['people']['languages']['language']:

                     if language['name'] not in language_df.columns:
                         language_df[language['name']] = None


                     if 'percent' in language:
                         language_df.loc[country][language['name']]=language['percent']
                         #print(country)
                     else:
                         #print(language)
                         language_df.loc[country][language['name']] = 100/len(df1.loc[country]
             except KeyError:
                 pass

         for country in language_df.index:

             total_sum = language_df.loc[country].sum()

             if 95.0 <= total_sum <= 105.0:
                 continue

             elif math.isnan(total_sum):
                 print("No data for:", country)

             else:
```

```
                specified_percentages = 0
                languages_wo_percentage = []
                for language in df1.loc[country]['data']['people']['languages']['language']:

                    if 'percent' in language:
                        if language['percent']>100:
                            print('Value for', language['name'], 'in', country, 'is over 100%
                            languages_wo_percentage.append(language['name'])
                        else:
                            specified_percentages += language['percent']

                    else:
                        languages_wo_percentage.append(language['name'])
            #                 print(country, counter, languages_wo_percentage, specified_percenta

                if (specified_percentages <=95 and len(languages_wo_percentage)==0):
                    for language in df1.loc[country]['data']['people']['languages']['language
                        language_df.loc[country][language['name']] = (language['percent']/spe

                for language in languages_wo_percentage:
                    language_df.loc[country][language] = (100-specified_percentages)/len(langu
```

```
Value for indigenous languages in Solomon Islands is over 100%. Scaling down in process...
No data for: Channel Islands
Value for Aranese  along with Catalan, speakers in Spain is over 100%. Scaling down in process
No data for: Saint Martin (French part)
No data for: Sint Maarten
No data for: U.S. Virgin Islands
No data for: Palestinian Territories
Value for other , Fang, Bubi census in Equatorial Guinea is over 100%. Scaling down in process
Value for over additional indigenous languages in Nigeria is over 100%. Scaling down in proces
Value for more than different languages and dialects in Chad is over 100%. Scaling down in pro
```

```
In [18]: len(df1.loc["Brunei"]['data']['people']['languages']['language'])
```

```
Out[18]: 3
```

```
In [19]: df1.loc["Guam"]['data']['people']['languages']['language']
```

```
Out[19]: [{'name': 'English', 'percent': 43.6},
          {'name': 'Filipino', 'percent': 21.2},
          {'name': 'Chamorro', 'percent': 17.8},
          {'name': 'other Pacific island languages', 'percent': 10},
          {'name': 'Asian languages', 'percent': 6.3},
          {'name': 'other', 'percent': 1.1}]
```

```
In [20]: df1.loc["Somalia"]['data']['people']['languages']['language']
```

```
Out[20]: [{'name': 'Somali',
           'note': 'official, according to the 2012 Transitional Federal Charter'},
          {'name': 'Arabic',
           'note': 'official, according to the 2012 Transitional Federal Charter'},
          {'name': 'Italian'},
          {'name': 'English'}]

In [ ]: # language_df.head()

In [22]: language_df.loc['Zambia'].sum()

Out[22]: 99.9

In [ ]: # language_df.loc['Equatorial Guinea']

In [ ]: # language_df.sum(axis=1)

In [ ]: # language_df.loc["New Caledonia"]

In [26]: # No data for: Channel Islands
         # No data for: Saint Martin (French part)
         # No data for: Sint Maarten
         # No data for: U.S. Virgin Islands
         # No data for: Palestinian Territories

         language_df.loc['Channel Islands']['French'] = 50
         language_df.loc['Channel Islands']['English'] = 50
         language_df.loc['Saint Martin (French part)']['English'] = 100
         language_df.loc['Sint Maarten']['English'] = 100
         language_df.loc['U.S. Virgin Islands']['Spanish'] = 17
         language_df.loc['U.S. Virgin Islands']['English'] = 100-17
         language_df.loc['Palestinian Territories']['Arabic']=100

In [27]: language_matrix = pd.DataFrame(index = language_df.index, columns = language_df.index

In [28]: language_df = language_df.fillna(0)

In [29]: language_df1 = language_df.copy()

In [30]: for column in language_df1.columns:
             counter = 0
             if len(column.split())>1:
                 column_list = column.split()

                 language_list = []
                 for item in column_list:
                     if item[0].isupper() and item in language_df1.columns:
                         counter+=1
                         language_list.append(item)
                 for language in language_list:
```

```
                    print(language, "-", column)
                    language_df1[language] = language_df1[language]+(language_df1[column]/len

            if counter!=0:
                language_df1.drop(column, axis=1, inplace = True)

        for column in language_df1.columns:
            if column[0].islower():
                language_df1.drop(column, axis=1, inplace = True)
```

```
Chinese - Chinese dialects
Chinese - Standard Chinese or Mandarin
Mandarin - Standard Chinese or Mandarin
Chinese - other Chinese dialects
Chinese - Northern Chinese
Filipino - Filipino  and English ; eight major dialects - Tagalog
English - Filipino  and English ; eight major dialects - Tagalog
Tagalog - Filipino  and English ; eight major dialects - Tagalog
English - English and Tongan
Tongan - English and Tongan
French - some French
Khmer - and Khmer
German - German  less than
Castilian - Castilian Spanish
Spanish - Castilian Spanish
German - German widely spoken
Russian - Russian widely used in government and business
French - French patois
Maya - Maya languages
Creole - Guyanese Creole
English - English patois
Spanish - Spanish only
Spanish - Spanish and indigenous languages
English - Panamanian English Creole
Creole - Panamanian English Creole
French - French Creole
Creole - French Creole
Hindustani - Caribbean Hindustani
Creole - Trinidadian Creole English
English - Trinidadian Creole English
Creole - Tobagonian Creole English
English - Tobagonian Creole English
Creole - Trinidadian Creole French
French - Trinidadian Creole French
Berber - Berber or Tamazight ; dialects include Kabyle Berber
Berber - Berber or Tamazight ; dialects include Kabyle Berber
Berber - Shawiya Berber
```

```
Berber - Mzab Berber
Berber - Tuareg Berber
English - English and French widely understood by educated classes
French - English and French widely understood by educated classes
Azeri - Azeri Turkic and Turkic dialects
Turkic - Azeri Turkic and Turkic dialects
Turkic - Azeri Turkic and Turkic dialects
Armenian - and Armenian are official in areas where native speakers of these languages constitu
English - English widely spoken
English - English ; Berber
Berber - English ; Berber
Berber - Berber languages , Tachelhit, Tarifit
English - English commonly used as a second language
Persian - Afghan Persian or Dari
French - French <.5% each
Kirundi - Kirundi and other language
French - French  and French and other language
French - French  and French and other language
Swahili - Swahili and Swahili and other language
Swahili - Swahili and Swahili and other language
English - English  and English and other language
English - English  and English and other language
Yoruba - Fon and Yoruba
Lingala - Lingala and Monokutuba
French - French  <.1
English - English  <.1
Swahili - Swahili  <.1
Creole - Seychellois Creole
Ewe - Ewe and Mina
Dagomba - Kabye  and Dagomba
Kiswahili - Kiswahili or Swahili
Swahili - Kiswahili or Swahili
```

```
In [ ]: # for column in sorted(language_df.columns):
        #     print(column)
```

```
In [32]: (language_df1.loc['Argentina']*language_df1.loc['United States']/(100**2)).sum()
```

```
Out[32]: 0.15333333333333335
```

```
In [33]: for combo in product(language_df1.index, repeat = 2):
             country1, country2 = combo
             language_matrix.at[country1, country2] = ((language_df1.loc[country1]*language_df
```

```
In [53]: # language_matrix
```

```
In [35]: counter = 0
         for country in language_df1.index:
```

```
            counter+=(language_df1.loc["American Samoa"]*language_df1.loc[country]).sum()/(lan
```

```
In [36]: (language_df1.loc["American Samoa"]*language_df1.loc['American Samoa']).sum()/(languag
```

```
Out[36]: 1.0
```

```
In [37]: counter
```

```
Out[37]: 3.6563588461427265
```

```
In [38]: language_matrix['United Kingdom'].sum()
```

```
Out[38]: 94.38853573098004
```

```
In [39]: language_matrix['United States'].sum()
```

```
Out[39]: 78.1116379786224
```

```
In [40]: language_matrix["Spain"].sum()
```

```
Out[40]: 11.179142474701631
```

```
In [54]: # language_matrix['Marshall Islands']
```

```
In [42]: df1.loc["Mali"]['data']['people']['languages']['language']
```

```
Out[42]: [{'name': 'French', 'note': 'official'},
          {'name': 'Bambara', 'percent': 46.3},
          {'name': 'Peul/Foulfoulbe', 'percent': 9.4},
          {'name': 'Dogon', 'percent': 7.2},
          {'name': 'Maraka/Soninke', 'percent': 6.4},
          {'name': 'Malinke', 'percent': 5.6},
          {'name': 'Sonrhai/Djerma', 'percent': 5.6},
          {'name': 'Minianka', 'percent': 4.3},
          {'name': 'Tamacheq', 'percent': 3.5},
          {'name': 'Senoufo', 'percent': 2.6},
          {'name': 'Bobo', 'percent': 2.1},
          {'name': 'unspecified', 'percent': 0.7},
          {'name': 'other', 'percent': 6.3}]
```

```
In [43]: df1.loc["Seychelles"]['data']['people']['languages']['language']
```

```
Out[43]: [{'name': 'Seychellois Creole', 'note': 'official', 'percent': 89.1},
          {'name': 'English', 'note': 'official', 'percent': 5.1},
          {'name': 'French', 'note': 'official', 'percent': 0.7},
          {'name': 'other', 'percent': 3.8},
          {'name': 'unspecified', 'percent': 1.4}]
```

```
In [55]: # language_matrix.sum(axis=1)
```

```
In [56]: # language_matrix.sum(axis=0)

In [46]: language_matrix.loc['Guam']['United States']

Out[46]: 1.2724377800681952

In [57]: # language_matrix

In [48]: for country in language_matrix.index:
             language_matrix.at[country,country]=0

In [58]: # language_matrix

In [50]: writer = pd.ExcelWriter('processed_data/language_data_wb.xlsx')
         # writer.define_name('countries', '=Sheet1!$A$2:$A$194')
         workbook=writer.book
         workbook.define_name('countries', '=Sheet1!$A$2:$A$'+str(len(language_matrix)+1))
         language_matrix.to_excel(writer)
         writer.save()
```

# D.13. Freedom in the World

# 13 - Freedom in the World

August 15, 2018

```
In [1]: import pandas as pd
        import csv
        import numpy as np

In [2]: countries_wb=pd.read_excel("raw data/Regions.xlsx", sheetname='Countries', header=0, sk
                                   names=['Country Data','Region','IncomeGroup'],
                                     parse_cols=None, parse_dates=False, date_parser=None, na_values

In [3]: with open('processed_data/iso2_name_dic.csv', encoding = "UTF-8") as csvfile:
            reader = csv.DictReader(csvfile, delimiter = ',')
            iso2_name_dict = {rows['iso2code']:rows['name'] for rows in reader}

In [4]: with open('processed_data/country_name_dic.csv', encoding = "UTF-8") as csvfile:
            reader = csv.DictReader(csvfile, delimiter = ',')
            country_name_dict = {rows['alt_name']:rows['name'] for rows in reader}

In [5]: freedom_df = pd.read_excel("raw data/Country and Territory Ratings and Statuses FIW1973

In [6]: freedom_df.at[0, 'Year(s) Under Review'] = 'data'

In [7]: columns = list(freedom_df.columns)

In [8]: unnamed_1 = list(range(2, len(freedom_df.columns), 3))
        unnamed_2 = list(range(3, len(freedom_df.columns), 3))

In [9]: for i in range(1,len(columns)):
            if i in unnamed_1:
                columns[i] = str(columns[i-1])+' CL'
            elif i in unnamed_2:
                columns[i] = str(columns[i-2])+' Status'

In [10]: freedom_df.columns = columns

In [11]: freedom_df.set_index('Year(s) Under Review', inplace=True)

In [12]: # freedom_df

In [13]: pr_cols = freedom_df.loc['data']=='PR'

In [14]: pr_df = freedom_df[pr_cols.index[pr_cols]]
```

```
In [15]: # freedom_df[pr_cols.index[pr_cols]]

In [16]: years = list(range(1990, 2018))

In [17]: pr_df = pr_df[years]

In [18]: pr_df = pr_df.rename(index=country_name_dict)

In [19]: pr_df_wb = countries_wb.sort_values(['Region', 'Country Data'])

In [20]: pr_df_wb = pr_df_wb.rename(index=country_name_dict)

In [21]: final_pr_df = pd.concat([pr_df_wb, pr_df], axis=1,join_axes=[pr_df_wb.index])

In [22]: len(final_pr_df)

Out[22]: 217

In [23]: final_pr_df = final_pr_df.fillna(0.4242)

In [24]: def func_pr(x):
             if x.values[0] is None:
                 return None
             else:
                 return final_pr_df.loc[x.name, x.values[0]]

In [25]: most_recent_value_colname = 'Most Recent Value'
         final_pr_df[most_recent_value_colname] = pd.DataFrame(final_pr_df.apply(lambda x: x.la
         cols = final_pr_df.columns.tolist()
         cols.insert(0, cols.pop(cols.index('Region')))
         cols.insert(0, cols.pop(cols.index('Country Data')))
         cols.insert(0, cols.pop(cols.index(most_recent_value_colname)))
         final_pr_df = final_pr_df.reindex(columns= cols)

In [26]: mapping = {'-' : 0.4242}

         final_pr_df = final_pr_df.replace(mapping)

In [27]: results_dic = {}

In [28]: results_dic['PR'] = final_pr_df

In [29]: cl_cols = freedom_df.loc['data']=='CL'

In [30]: cl_df = freedom_df[cl_cols.index[cl_cols]]

In [31]: old_columns = cl_df.columns

In [32]: old_columns
```

```
Out[32]: Index(['2009 CL', '2010 CL', '2011 CL', '2012 CL', '2013 CL', '2014 CL',
                '2015 CL', '2016 CL', '2017 CL'],
               dtype='object')

In [33]: new_cols = [int(s.strip(' CL')) for s in old_columns]

In [34]: cl_df.columns = new_cols

In [35]: cl_df = cl_df.rename(index=country_name_dict)

In [36]: cl_df_wb = countries_wb.sort_values(['Region', 'Country Data'])

In [37]: cl_df_wb = cl_df_wb.rename(index=country_name_dict)

In [38]: final_cl_df = pd.concat([cl_df_wb, cl_df], axis=1,join_axes=[cl_df_wb.index])

In [39]: final_cl_df = final_cl_df.fillna(0.4242)

In [40]: def func_cl(x):
             if x.values[0] is None:
                 return None
             else:
                 return final_cl_df.loc[x.name, x.values[0]]

In [41]: most_recent_value_colname = 'Most Recent Value'
         final_cl_df[most_recent_value_colname] = pd.DataFrame(final_cl_df.apply(lambda x: x.la
         cols = final_cl_df.columns.tolist()
         cols.insert(0, cols.pop(cols.index('Region')))
         cols.insert(0, cols.pop(cols.index('Country Data')))
         cols.insert(0, cols.pop(cols.index(most_recent_value_colname)))
         final_cl_df = final_cl_df.reindex(columns= cols)

In [42]: mapping = {'-' : 0.4242}

         final_cl_df = final_cl_df.replace(mapping)

In [43]: results_dic['CL'] = final_cl_df

In [44]: # results_dic['CL']

In [45]: st_cols = freedom_df.loc['data']=='Status'

In [46]: st_df = freedom_df[st_cols.index[st_cols]]

In [47]: old_columns = st_df.columns

In [48]: new_cols = [int(s.strip(' Status')[-4:]) for s in old_columns]

In [49]: st_df.columns = new_cols

In [50]: st_df = st_df[years]
```

```
In [51]: st_df = st_df.rename(index=country_name_dict)

In [52]: st_df_wb = countries_wb.sort_values(['Region', 'Country Data'])

In [53]: st_df_wb = st_df_wb.rename(index=country_name_dict)

In [54]: final_st_df = pd.concat([st_df_wb, st_df], axis=1,join_axes=[st_df_wb.index])

In [55]: final_st_df = final_st_df.fillna(0.4242)

In [56]: def func_st(x):
             if x.values[0] is None:
                 return None
             else:
                 return final_st_df.loc[x.name, x.values[0]]

In [57]: most_recent_value_colname = 'Most Recent Value'
         final_st_df[most_recent_value_colname] = pd.DataFrame(final_st_df.apply(lambda x: x.la
         cols = final_st_df.columns.tolist()
         cols.insert(0, cols.pop(cols.index('Region')))
         cols.insert(0, cols.pop(cols.index('Country Data')))
         cols.insert(0, cols.pop(cols.index(most_recent_value_colname)))
         final_st_df = final_st_df.reindex(columns= cols)

In [58]: mapping = {'F': 1, 'PF': 2, 'NF': 3 , '-' : 0.4242}

         final_st_df = final_st_df.replace(mapping)
         final_st_df = final_st_df.fillna(0.4242)

In [59]: final_st_df.loc['Micronesia (Federated States of)'][1990]

Out[59]: 0.42420000000000002

In [60]: # final_st_df

In [61]: results_dic['ST'] = final_st_df

In [62]: # results_dic['ST']

In [63]: for country in pr_df.index:
             if country not in pr_df_wb.index:
                 print(country)

data
Czechoslovakia
Germany, E.
Germany, W.
Northern Cyprus
Taiwan
USSR
```

```
Vietnam, N.
Vietnam, S.
Yemen, N.
Yemen, S.
Yugoslavia
Yugoslavia (Serbia & Montenegro)
```

```
In [64]: for country in pr_df_wb.index:
             if country not in pr_df.index:
                 print(country)
```

```
American Samoa
Guam
Hong Kong SAR China
Macau SAR China
Northern Mariana Islands
New Caledonia
French Polynesia
Channel Islands
Faroe Islands
Gibraltar
Greenland
Isle of Man
Aruba
Curaçao
Cayman Islands
Saint Martin (French part)
Puerto Rico
Sint Maarten
Turks & Caicos Islands
British Virgin Islands
U.S. Virgin Islands
Palestinian Territories
Bermuda
```

```
In [65]: LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

         def colToExcel(col): # col is 1 based
             excelCol = str()
             div = col
             while div:
                 (div, mod) = divmod(div-1, 26) # will return (x, 0 .. 25)
                 excelCol = chr(mod + 65) + excelCol
             return excelCol
```

```
In [66]: try:
             writer.close()
```

```
        except (NameError, AttributeError):
            pass
        writer = pd.ExcelWriter('processed_data/freedom_in_the_world.xlsx', engine='xlsxwrite
        workbook = writer.book

        for index in results_dic.keys():
            print(index)
            sheet_name = index.replace(" ", "_").replace("(", "").replace(")","").replace(":"

            df_to_write = results_dic[index].copy()
            df_to_write.to_excel(writer, sheet_name=sheet_name)

            counter = 1+3

            for column in results_dic[index].columns:
                if (column == 'Region' or column == 'Country Data'):
                    continue
                elif column == "Most Recent Value":
                    excel_range = '='+sheet_name+'!$B$2:$B$'+str(len(results_dic[index])+1)
                    column_name = index +"_"+ column.replace(" ", "_").replace("(", "").repla
                    print(column_name, excel_range)
                    workbook.define_name(column_name, excel_range)
                else:
                    counter+=1
                    excel_col = colToExcel(counter)
                    excel_range = '='+sheet_name+'!$'+excel_col+'$2:$'+excel_col+'$'+str(len(

                    column_name = index + "_" + str(column)
                    workbook.define_name(column_name, excel_range)

        writer.save()
```

```
PR
PR_Most_Recent_Value =PR!$B$2:$B$218
CL
CL_Most_Recent_Value =CL!$B$2:$B$218
ST
ST_Most_Recent_Value =ST!$B$2:$B$218
```

## D.14. Number of Border Crossings Between Countries

# 14 - Number of Border Crossings Between Countries

August 15, 2018

In [1]:
```
import pandas as pd
import numpy as np
import csv
import os
```

In [2]:
```
border_crossings_df = pd.read_excel('raw data/border_crossings_phv-2.xls')
```

In [3]:
```
# border_crossings_df
```

In [4]:
```
countries_wb=pd.read_excel("raw data/Regions.xlsx", sheetname='Countries', header=0, sk
                           names=['Country Data','Region','IncomeGroup'],
                             parse_cols=None, parse_dates=False, date_parser=None, na_values
```

In [5]:
```
with open('processed_data/country_name_dic.csv', encoding='UTF-8') as csvfile:
    reader = csv.DictReader(csvfile, delimiter = ',')
    country_name_dict = {rows['alt_name']:rows['name'] for rows in reader}
```

In [6]:
```
countries_wb_sort = countries_wb.sort_values(['Region', 'Country Data'], axis=0)
countries_wb_sort = countries_wb_sort.rename(index=country_name_dict)
```

In [7]:
```
border_crossings_wb = pd.DataFrame(index = countries_wb_sort.index, columns = countries
```

In [8]:
```
border_crossings_df['CNTRY_NAME'] = border_crossings_df.CNTRY_NAME.replace(country_name
```

In [9]:
```
border_crossings_df['CNTRY_BORD'] = border_crossings_df.CNTRY_BORD.replace(country_name
```

In [10]:
```
border_crossings_wb.fillna(0, inplace=True)
```

In [11]:
```
for row in border_crossings_df.index:
    country1 = border_crossings_df.loc[row]['CNTRY_NAME']
    country2 = border_crossings_df.loc[row]['CNTRY_BORD']

    if (country1 in border_crossings_wb.index) and (country2 in border_crossings_wb.in
        border_crossings_wb.at[country1, country2] += 1
    else:
        print(country1, country2)
```

```
Morocco Western Sahara
Western Sahara Morocco
Italy Vatican City
Italy Vatican City
Vatican City Italy
Vatican City Italy
```

```
In [12]: border_crossings_wb.loc['India']['Nepal']
```

```
Out[12]: 6
```

```
In [13]: # (border_crossings_wb/(border_crossings_wb.max()))
```

```
In [14]: dfmax = border_crossings_wb.max()
```

```
In [15]: border_crossing_wb = border_crossings_wb.divide(dfmax, axis=0)
```

```
In [16]: border_crossing_wb.fillna(0, inplace=True)
```

```
In [17]: writer = pd.ExcelWriter('processed_data/road_contiguity_data_wb.xlsx')
         # writer.define_name('countries', '=Sheet1!$A$2:$A$194')
         workbook=writer.book
         workbook.define_name('countries', '=Sheet1!$A$2:$A$'+str(len(border_crossings_wb)+1))
         border_crossing_wb.to_excel(writer)
         writer.save()
```

# D.15. World Risk Index: Natural Disasters

# 15 - World Risk Index (Natural Disaster)

August 15, 2018

```
In [1]: import pandas as pd
        import csv
        import numpy as np

In [2]: with open('processed_data/country_name_dic.csv') as csvfile:
            reader = csv.DictReader(csvfile, delimiter = ',')
            country_name_dict = {rows['alt_name']:rows['name'] for rows in reader}

In [3]: countries_wb=pd.read_excel("raw data/Regions.xlsx", sheetname='Countries', header=0, sk
                                   names=['Country Data','Region','IncomeGroup'],
                                     parse_cols=None, parse_dates=False, date_parser=None, na_values

In [4]: countries_wb.rename(index = country_name_dict, inplace = True)

In [5]: countries_wb = countries_wb.sort_values(["Region", "Country Data"])

In [6]: risk_data=pd.read_excel("raw data/Table_WorldRiskIndex-2012-2016-Average.xlsx", sheetna

In [7]: risk_data.set_index('Country', inplace = True)

In [8]: for country in risk_data.index:
            if country not in country_name_dict:
                print(country)

In [9]: risk_data.rename(index = country_name_dict, inplace = True)

In [10]: risk_data_wb = pd.DataFrame(index=countries_wb.index)

In [11]: for column in risk_data.columns:
             risk_data_wb[column] = risk_data[column]

In [12]: risk_data_wb.fillna(0.4242, inplace = True)

In [13]: risk_data_wb.columns

Out[13]: Index(['Rank', 'Risk Ø', 'Exposure 2012-2016', 'Vulnerability Ø',
                'Susceptibility Ø', 'Lack of coping capacities Ø',
                'Lack of adaptive capacities Ø'],
               dtype='object')
```

In [14]:
```
import xlsxwriter
```

In [15]:
```
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def colToExcel(col): # col is 1 based
    excelCol = str()
    div = col
    while div:
        (div, mod) = divmod(div-1, 26) # will return (x, 0 .. 25)
        excelCol = chr(mod + 65) + excelCol
    return excelCol
```

In [16]:
```
writer = pd.ExcelWriter('processed_data/world_risk_index.xlsx', engine='xlsxwriter')
workbook = writer.book


risk_data_wb.to_excel(writer, sheet_name='data')

counter = 1
for column in risk_data_wb.columns:
        counter+=1
        excel_col=colToExcel(counter)
        excel_range = '=data!$'+excel_col+'$2:$'+excel_col+'$'+str(len(risk_data_wb)+
        column_name = column.replace(" Ø", "").replace("-", "_").replace(" ", "_")
        workbook.define_name(column_name , excel_range)
```

In [17]:
```
risk_data_wb.columns
```

Out[17]:
```
Index(['Rank', 'Risk Ø', 'Exposure 2012-2016', 'Vulnerability Ø',
       'Susceptibility Ø', 'Lack of coping capacities Ø',
       'Lack of adaptive capacities Ø'],
      dtype='object')
```

## D.16. Exploratory Modeling and Analysis

# MigrationModel_Multiprocessing_EMA

August 15, 2018

## 1 EMA Experimentation Multiprocessing

This notebook allows for experimentation with EMA workbench.

The model also requires the data files to be in the folder of this notebook and the model.

### 1.1 1. Importing the required Python packages

```
In [1]: from ema_workbench import(Model, RealParameter,Constant,IntegerParameter,CategoricalPa
        from ema_workbench.connectors.vensim import VensimModel
        from ema_workbench.em_framework.evaluators import LHS, SOBOL
        import timeit
        from ema_workbench import MultiprocessingEvaluator
        from ema_workbench.analysis.plotting import envelopes

        from ema_workbench.analysis.plotting_util import KDE
        import pandas as pd
        import numpy as np
```

```
C:\Users\LocalAdmin\Anaconda3\lib\site-packages\ema_workbench\em_framework\optimization.py:29:
  warnings.warn("platypus based optimization not available", ImportWarning)
C:\Users\LocalAdmin\Anaconda3\lib\site-packages\ema_workbench\connectors\__init__.py:18: Impor
  warnings.warn("netlogo connector not available", ImportWarning)
C:\Users\LocalAdmin\Anaconda3\lib\site-packages\ema_workbench\connectors\__init__.py:23: Impor
  warnings.warn("pysd connector not available", ImportWarning)
C:\Users\LocalAdmin\Anaconda3\lib\importlib\_bootstrap.py:219: ImportWarning: can't resolve pa
  return f(*args, **kwds)
```

### 1.2 2. Loading the Vensim model

In this step, we specify the working directory (this should be the location of this notebook and the Vensim model file), and import the Vensim model into our workspace. Note that the model file needs to have a .vpm extension, which can be created by using the publish function in Vensim.

In addition, the Vensim dll can be specified if required. This defaults to 'vendll32.dll' for the normal vensimdll, but if double precision is installed and required, change this to 'vdpdll32'. [Note: Since only one model can be loaded per dll file, this also allows for two models to be loaded at the same time.]

We can also specify a name for the .vdf file in which we want Venpy to store the intermediate results. Note that this file should not be in use by any other program. This defaults to 'CurrentRun'.

```
In [2]: wd =r'.\migrationmodel'
        model = VensimModel('WorldMigrationModel', wd = wd , model_file=r'D:\Migration\Migrati

In [3]: ema_logging.log_to_stderr(ema_logging.INFO)    # we want to see what EMA is doing

Out[3]: <Logger EMA (DEBUG)>

In [4]: names = pd.read_excel('contiguity_data.xlsx', 'Sheet1')

In [5]: names.set_index('country', inplace = True)

In [6]: country_list = list(names.index)
```

### 1.3 3. Specify uncertainties and outcomes

Here we will specify the uncertainties and outcomes. The subscript of a variable is simply placed between square brackets.

```
In [7]: uncertainties = [CategoricalParameter('SWITCH min0 smth1 max2 stickyrndm3 pinkrndm4',
                         RealParameter('pc of pop atR of conflict violence', 0.2 , 0.95), # 0.
                         RealParameter('pc of pop at risk of oppression', 0.25, 0.95), #0.038
                         RealParameter('pc pop atRd2 conflict violence willing2 migrate', 0.2,
                         RealParameter('pc pop atR of oppression willing2 migrate', 0.1, 0.6),
                         RealParameter('pc pop atRd2 disasters willing2 migrate', 0.1, 0.6), #
                         RealParameter('pc pop atRd2 economic or food scarcity willing2 migrat
                         RealParameter('pc pop atR of oppression willing2 migrate', 0.1, 0.6),
                         RealParameter('fraction of popRd2 conflict violence RWA to migrate try
                         RealParameter('pc overlap in pop atR willing and able2 migrate due to
                         RealParameter('pc pop atR of conflict violence fysically and financia
                         RealParameter('pc pop atR of conflict violence oppression NOT even abl
                         RealParameter('pc pop atRd2 economic or food scarcity fysically and fi
                         RealParameter('fr extra autonomous development exposure', 0.001, 0.1)

                         RealParameter('fraction of population able to afford advanced transpor
                         RealParameter('fraction of population able to afford advanced transpor
                         RealParameter('fraction of population able to afford advanced transpor
                         RealParameter('fraction of population able to afford advanced transpor

                         RealParameter('birth rate scale factor high income',0.5,1.5),
                         RealParameter('birth rate scale factor upper middle income',0.5,1.5),
                         RealParameter('birth rate scale factor lower middle income',0.5,1.5),
                         RealParameter('birth rate scale factor low income',0.5,1.5),

                         RealParameter('death rate scale factor high income',0.5,1.5),
                         RealParameter('death rate scale factor upper middle income',0.5,1.5),
```

```
                        RealParameter('death rate scale factor lower middle income',0.5,1.5),
                        RealParameter('death rate scale factor low income',0.5,1.5),

                        RealParameter('migration rate scale factor high income',0.2,1.1),
                        RealParameter('migration rate scale factor upper middle income',0.2,1
                        RealParameter('migration rate scale factor lower middle income',0.2,1
                        RealParameter('migration rate scale factor low income',0.2,1.1),

                        RealParameter('naturalization rate scale factor high income',0.001,0.
                        RealParameter('naturalization rate scale factor upper middle income',
                        RealParameter('naturalization rate scale factor lower middle income',
                        RealParameter('naturalization rate scale factor low income',0.001,0.0

                        RealParameter('societal stress influence factor',1,10),
                        RealParameter('societal stress threshold pc', 0.4, 0.9), # 0.0

                        RealParameter('transit weight', 1, 10), # 0.0
                        RealParameter('residence weight', 1, 10), # 0.0

                        RealParameter('interregion migration factor', 0.5, 5), # 0.0

                        RealParameter('migrant coping capacity growth rate per country scale
                        RealParameter('pc of current coping capacity scale factor', 0.3, 0.9)
                        RealParameter('forward migration rate scale factor', 0.1, 0.9), # PER
                        RealParameter('fraction of migrants applying in most attractive counti
                        RealParameter('close EU borders threshold', 50000000,100000000),
                        RealParameter('close ECA borders threshold', 130000000,250000000),
                       ]

In [8]: outcomes = [TimeSeriesOutcome('total population in model'),
                    TimeSeriesOutcome('total migrants'),
                    TimeSeriesOutcome('total migrants in EU[EU]'),
                    TimeSeriesOutcome('total migrants in ECA[Europe and Central Asia]')]

In [9]: for country in country_list:
            outcomes.append(TimeSeriesOutcome('population to other country['+country+']'))
            outcomes.append(TimeSeriesOutcome('total migrants in country['+country+']'))
            outcomes.append(TimeSeriesOutcome('population per country['+country+']'))
            outcomes.append(TimeSeriesOutcome('total naturalizations['+country+']'))
            outcomes.append(TimeSeriesOutcome('returnees['+country+']'))
            outcomes.append(TimeSeriesOutcome('internal migration total['+country+']'))
            outcomes.append(TimeSeriesOutcome('societal stress in country due to migrants['+cou

In [10]: model.uncertainties = uncertainties
         model.outcomes = outcomes
```

### 1.4 4.1 Base Ensemble

```
In [11]: nr_scenarios = 500
         seed = 50
         np.random.seed(seed) # fix seed

In [12]: start_time = timeit.default_timer()

         with MultiprocessingEvaluator(model) as evaluator:
             policy_results = evaluator.perform_experiments(scenarios=nr_scenarios)# policies=

         elapsed = timeit.default_timer() - start_time

         print("Total time in minutes:", elapsed/60, "-- Time per run in seconds:", elapsed/(n
```

```
[MainProcess/INFO] pool started
[MainProcess/INFO] performing 500 scenarios * 1 policies * 1 model(s) = 500 experiments
[MainProcess/INFO] 50 cases completed
[MainProcess/INFO] 100 cases completed
[MainProcess/INFO] 150 cases completed
[MainProcess/INFO] 200 cases completed
[MainProcess/INFO] 250 cases completed
[MainProcess/INFO] 300 cases completed
[MainProcess/INFO] 350 cases completed
[MainProcess/INFO] 400 cases completed
[MainProcess/INFO] 450 cases completed
[MainProcess/INFO] 500 cases completed
[MainProcess/INFO] experiments finished
[MainProcess/INFO] terminating pool
[SpawnPoolWorker-4/INFO] finalizing
[SpawnPoolWorker-2/INFO] finalizing
[SpawnPoolWorker-3/INFO] finalizing
[SpawnPoolWorker-1/INFO] finalizing
```

```
Total time in minutes: 111.57994722596125 -- Time per run in seconds: 13.389593667115351
```

```
In [13]: ## In case you want to save the outputs in a file:

         save_results(policy_results, r'D:\Migration\Results\BaseEnsemble500Scenarios.tar.gz')
```

```
[MainProcess/INFO] results saved successfully to D:\Migration\Results\BaseEnsemble500Scenarios
```

```
In [ ]: policy_results = load_results( r'D:\Migration\Results\BaseEnsemble500Scenarios.tar.gz'

In [16]: ## Import specific plotting commands:
         import matplotlib.pyplot as plt
```

```
from ema_workbench.analysis.plotting import lines, plot_lines_with_envelopes
from ema_workbench.analysis.plotting_util import KDE, HIST, VIOLIN, BOXPLOT
import seaborn as sns
import ema_workbench.analysis.pairs_plotting as pairs
import ema_workbench.analysis.plotting as emaplt
```

In [ ]:
```
%matplotlib inline
```

```
fig = lines(policy_results, outcomes_to_show=('total population in model'),  density=KD
fig[0].savefig(r'Plots\BaseEnsemble\total_population_base_ensemble.png')
fig = lines(policy_results, outcomes_to_show=('total migrants'),  density=KDE) #group_
fig[0].savefig(r'Plots\BaseEnsemble\total_migrants_base_ensemble.png')
fig = lines(policy_results, outcomes_to_show=('total migrants in ECA[Europe and Centra
fig[0].savefig(r'Plots\BaseEnsemble\total_migrantsECA_base_ensemble.png')
fig = lines(policy_results, outcomes_to_show=('total migrants in EU[EU]'),  density=KDE
fig[0].savefig(r'Plots\BaseEnsemble\total_migrantsEU_base_ensemble.png')

for country in country_list:
    outcome = 'total migrants in country['+country+']'
    fig = lines(policy_results, outcomes_to_show=outcome,  density=KDE)
    fig[0].savefig(r"Plots\BaseEnsemble\base_"+outcome+".png")

    outcome = 'population to other country['+country+']'
    fig = lines(policy_results, outcomes_to_show=outcome,  density=KDE)
    fig[0].savefig(r"Plots\BaseEnsemble\base_"+outcome+'.png')

    outcome='population per country['+country+']'
    fig = lines(policy_results, outcomes_to_show=outcome,  density=KDE)
    fig[0].savefig(r"Plots\BaseEnsemble\base_"+outcome+'.png')


    outcome='total naturalizations['+country+']'
    fig = lines(policy_results, outcomes_to_show=outcome,  density=KDE)
    fig[0].savefig(r"Plots\BaseEnsemble\base_"+outcome+'.png')

    outcome='returnees['+country+']'
    fig = lines(policy_results, outcomes_to_show=outcome,  density=KDE)
    fig[0].savefig(r"Plots\BaseEnsemble\base_"+outcome+'.png')

    outcome='internal migration total['+country+']'
    fig = lines(policy_results, outcomes_to_show=outcome,  density=KDE)
    fig[0].savefig(r"Plots\BaseEnsemble\base_"+outcome+'.png')

    outcome='societal stress in country due to migrants['+country+']'
    fig = lines(policy_results, outcomes_to_show=outcome,  density=KDE)
    fig[0].savefig(r"Plots\BaseEnsemble\base_"+outcome+'.png')
    plt.close('all')
```

In [ ]:
```
fig[0]
```

```
In [ ]: from ema_workbench.analysis import pairs_plotting

In [ ]: # fig, axes = pairs_plotting.pairs_scatter(policy_results, legend=False) # don't run

In [ ]: !pip install mpldatacursor

In [ ]: experiments, outcomes = policy_results

In [ ]: outcomes_list = ['total population in model', 'total migrants', 'total migrants in EU[

In [ ]: outcomes.keys()

In [ ]: outcomes_feature_scoring = {}

        for i in outcomes_list:
            outcomes_feature_scoring[i]=outcomes[i]

In [ ]: from ema_workbench.analysis import feature_scoring
        from matplotlib.colors import LogNorm
        import matplotlib.pyplot as plt
        import seaborn as sns

        fig,ax = plt.subplots(figsize=(10,10))


        x=experiments
        y=outcomes_feature_scoring

        fs = feature_scoring.get_feature_scores_all(x,y)
        fs=fs.drop('model').drop('policy')
        sns.heatmap(fs, cmap='viridis', norm=LogNorm(vmin=0, vmax=1), annot=True)
        plt.savefig('GlobalPoliciesFeatureScoring.png', bbox_inches='tight')
```

### 1.5 4.2 Global Policies

```
In [ ]: policies = [Policy('None', **{"SWITCH low income shelter increase policy":0,
                                       'SWITCH high income shelter increase policy':0}),
                    Policy('Low', **{'SWITCH low income shelter increase policy':1,
                                       'SWITCH high income shelter increase policy':0}),
                    Policy('High', **{'SWITCH low income shelter increase policy':0,
                                       'SWITCH high income shelter increase policy':1}),
                    Policy('LowHigh', **{'SWITCH low income shelter increase policy':1,
                                       'SWITCH high income shelter increase policy':1})]

In [ ]: nr_scenarios = 100
        seed = 50
        np.random.seed(seed) # fix seed
```

In [ ]:
```
start_time = timeit.default_timer()

with MultiprocessingEvaluator(model) as evaluator:
    policy_results = evaluator.perform_experiments(scenarios=nr_scenarios, policies = p

elapsed = timeit.default_timer() - start_time

print("Total time in minutes:", elapsed/60, "-- Time per run in seconds:", elapsed/(nr
```

In [ ]:
```
## In case you want to save the outputs in a file:

save_results(policy_results, r'D:\Migration\Results\GlobalPolicies100scenarios.tar.gz')
```

In [ ]:
```
fig = lines(policy_results, outcomes_to_show=('total population in model'),  density=KD
fig[0].savefig(r'Plots\GlobalPolicies\total_population_global.png')
fig = lines(policy_results, outcomes_to_show=('total migrants'),  density=KDE,group_by=
fig[0].savefig(r'Plots\GlobalPolicies\total_migrants_global.png')
fig = lines(policy_results, outcomes_to_show=('total migrants in ECA[Europe and Central
fig[0].savefig(r'Plots\GlobalPolicies\total_migrantsECA_global.png')
fig = lines(policy_results, outcomes_to_show=('total migrants in EU[EU]'),  density=KDE
fig[0].savefig(r'Plots\GlobalPolicies\total_migrantsEU_global.png')
```

In [ ]:
```
%matplotlib inline

fig1 = lines(policy_results, outcomes_to_show=('total population in model'),  density=K
fig2 = lines(policy_results, outcomes_to_show=('total migrants'),  density=KDE, group_b
fig1 = lines(policy_results, outcomes_to_show=('total migrants in ECA[Europe and Centra
fig1 = lines(policy_results, outcomes_to_show=('total migrants in EU[EU]'),  density=KD
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Germany]'), c
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Turkey]'), de
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Morocco]'), c
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Zimbabwe]'),
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Guinea]'), de
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Greece]'), de
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Spain]'), der
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Netherlands]
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Luxembourg]')
```

In [ ]:
```
# policy_results = load_results(r'D:\Migration\Results\RegionalPolicies.tar.gz')
```

In [ ]:
```
import tarfile
tar = tarfile.open(r'D:\Migration\Results\GlobalPolicies.tar.gz', "r:gz")
for member in tar.getmembers():
    f = tar.extractfile(member)
    if f is not None:
        content = f.read()
```

In [ ]:
```
content
```

## 1.6 4.3 Regional Policies

In [ ]:
```
policy_results = None
```

In [ ]:
```
policies = [Policy('None', **{"SWITCH close borders":0,
                               'SWITCH close borders EU':0,
                               'SWITCH close borders Europe and Central Asia':0 }),
            Policy('EU', **{"SWITCH close borders":1,
                            'SWITCH close borders EU':1,
                            'SWITCH close borders Europe and Central Asia':0 }),
           ]

# Policy('ECA', **{"SWITCH close borders":1,
#                               'SWITCH close borders EU':0,
#                               'SWITCH close borders Europe and Central Asia':1 }
```

In [ ]:
```
nr_scenarios = 100
seed = 50
np.random.seed(seed) # fix seed
```

In [ ]:
```
start_time = timeit.default_timer()

with MultiprocessingEvaluator(model) as evaluator:
    policy_results = evaluator.perform_experiments(scenarios=nr_scenarios, policies = p

elapsed = timeit.default_timer() - start_time

print("Total time in minutes:", elapsed/60, "-- Time per run in seconds:", elapsed/(nr
```

In [ ]:
```
## In case you want to save the outputs in a file:

save_results(policy_results, r'D:\Migration\Results\RegionalPolicies10Scenarios.tar.gz
```

In [ ]:
```
%matplotlib inline

fig1 = lines(policy_results, outcomes_to_show=('total population in model'),  density=K
fig2 = lines(policy_results, outcomes_to_show=('total migrants'),  density=KDE, group_l
fig1 = lines(policy_results, outcomes_to_show=('total migrants in ECA[Europe and Centra
fig1 = lines(policy_results, outcomes_to_show=('total migrants in EU[EU]'),  density=KD
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Germany]'), c
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Turkey]'), de
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Morocco]'), c
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Zimbabwe]'),
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Guinea]'), de
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Greece]'), de
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Spain]'), der
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Netherlands]
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Luxembourg]'
```

## 1.7 4.4 National Policies

```
In [11]: policies = [Policy('None', **{"COUNTRY POLICY forward migration rate scale factor[Net
                                       'COUNTRY POLICY migrant coping capacity growth rate[N
                                      'COUNTRY POLICY manage societal stress[Netherlands]':1})
                     Policy('Stress', **{"COUNTRY POLICY forward migration rate scale factor[N
                                       'COUNTRY POLICY migrant coping capacity growth rate[N
                                      'COUNTRY POLICY manage societal stress[Netherlands]':0.2
                     Policy('Coping', **{"COUNTRY POLICY forward migration rate scale factor[N
                                       'COUNTRY POLICY migrant coping capacity growth rate[N
                                      'COUNTRY POLICY manage societal stress[Netherlands]':1})
                     Policy('Forward', **{"COUNTRY POLICY forward migration rate scale factor[
                                       'COUNTRY POLICY migrant coping capacity growth rate[N
                                      'COUNTRY POLICY manage societal stress[Netherlands]':1})]

In [12]: nr_scenarios = 100
         seed = 50
         np.random.seed(seed) # fix seed

In [13]: start_time = timeit.default_timer()

         with MultiprocessingEvaluator(model) as evaluator:
             policy_results = evaluator.perform_experiments(scenarios=nr_scenarios, policies =

         elapsed = timeit.default_timer() - start_time

         print("Total time in minutes:", elapsed/60, "-- Time per run in seconds:", elapsed/(n
```

```
[MainProcess/INFO] pool started
[MainProcess/INFO] performing 100 scenarios * 4 policies * 1 model(s) = 400 experiments
[MainProcess/INFO] 40 cases completed
[MainProcess/INFO] 80 cases completed
[MainProcess/INFO] 120 cases completed
[MainProcess/INFO] 160 cases completed
[MainProcess/INFO] 200 cases completed
[MainProcess/INFO] 240 cases completed
[MainProcess/INFO] 280 cases completed
[MainProcess/INFO] 320 cases completed
[MainProcess/INFO] 360 cases completed
[MainProcess/INFO] 400 cases completed
[MainProcess/INFO] experiments finished
[MainProcess/INFO] terminating pool
[SpawnPoolWorker-1/INFO] finalizing
[SpawnPoolWorker-3/INFO] finalizing
[SpawnPoolWorker-4/INFO] finalizing
[SpawnPoolWorker-2/INFO] finalizing


Total time in minutes: 95.08389314211 -- Time per run in seconds: 14.2625839713165
```

```
In [14]: ## In case you want to save the outputs in a file:

         save_results(policy_results, r'D:\Migration\Results\NationalPolicies10Scenarios.tar.gz

[MainProcess/INFO] results saved successfully to D:\Migration\Results\NationalPolicies10Scenari


In [17]: fig = lines(policy_results, outcomes_to_show=('total population in model'),  density=K
         fig[0].savefig(r'Plots\NationalPolicies\total_population_national.png')
         fig = lines(policy_results, outcomes_to_show=('total migrants'),  density=KDE,group_by
         fig[0].savefig(r'Plots\NationalPolicies\total_migrants_national.png')
         fig = lines(policy_results, outcomes_to_show=('total migrants in ECA[Europe and Centra
         fig[0].savefig(r'Plots\NationalPolicies\total_migrantsECA_national.png')
         fig = lines(policy_results, outcomes_to_show=('total migrants in EU[EU]'),  density=KD
         fig[0].savefig(r'Plots\NationalPolicies\total_migrantsEU_national.png')

In [ ]: %matplotlib inline

        fig1 = lines(policy_results, outcomes_to_show=('total population in model'),  density=K
        fig2 = lines(policy_results, outcomes_to_show=('total migrants'),  density=KDE, group_b
        fig1 = lines(policy_results, outcomes_to_show=('total migrants in ECA[Europe and Centra
        fig1 = lines(policy_results, outcomes_to_show=('total migrants in EU[EU]'),  density=KD
        fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Germany]'), d
        fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Turkey]'), de
        fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Morocco]'), d
        fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Zimbabwe]'),
        fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Guinea]'), de
        fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Greece]'), de
        fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Spain]'), der
        fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Netherlands]
        fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Luxembourg]'
```

### 1.8 5. Visualization of the results

```
In [ ]: ## Import specific plotting commands:
        import matplotlib.pyplot as plt
        from ema_workbench.analysis.plotting import lines, plot_lines_with_envelopes
        from ema_workbench.analysis.plotting_util import KDE, HIST, VIOLIN, BOXPLOT
        import seaborn as sns
        import ema_workbench.analysis.pairs_plotting as pairs
        import ema_workbench.analysis.plotting as emaplt

In [ ]: experiments, outcomes = policy_results

In [ ]: for outcome in outcomes.keys():
            print(outcome)
```

### 1.8.1 Specified Policies

In [ ]:
```
%matplotlib inline
```

```
fig1 = lines(policy_results, outcomes_to_show=('total population in model'), group_by=
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Germany]'),gr
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Turkey]'),gro
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Morocco]'),gr
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Zimbabwe]'),g
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Guinea]'),gro
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Greece]'),gro
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Spain]'),grou
fig1 = lines(policy_results, outcomes_to_show=('total migrants in country[Netherlands]
fig2 = lines(policy_results, outcomes_to_show=('total migrants'),group_by='policy',  de

# fig2 = lines(policy_results, outcomes_to_show=('pop per region[regionEast Asia and P
```

In [ ]:
```
fig2 = lines(policy_results, outcomes_to_show=('total naturalizations[Turkey]'),group_
fig2 = lines(policy_results, outcomes_to_show=('total naturalizations[Japan]'),group_by
fig2 = lines(policy_results, outcomes_to_show=('total naturalizations[Australia]'),grou
fig2 = lines(policy_results, outcomes_to_show=('total naturalizations[South Africa]'),g
```

In [ ]:
```
fig2 = lines(policy_results, outcomes_to_show=('total naturalizations[Guinea]'),group_
fig2 = lines(policy_results, outcomes_to_show=('total naturalizations[Ethiopia]'),grou
fig2 = lines(policy_results, outcomes_to_show=('total naturalizations[India]'),group_by
fig2 = lines(policy_results, outcomes_to_show=('total naturalizations[Thailand]'),grou
fig2 = lines(policy_results, outcomes_to_show=('total naturalizations[Bolivia]'),group_
```

In [ ]:
```
experiments, outcomes = policy_results
```

In [ ]:
```
germany_outcomes = {k:v for k,v in outcomes.items() if 'migrants in destination country
```

In [ ]:
```
germany_outcomes.keys()
```

In [ ]:
```
country_list=['Germany', 'Turkey', 'Uganda', 'United States', 'Argentina', 'China', 'Au
variable = 'migrants in destination country'

for country in country_list:
    x=pd.DataFrame(index=[country])
    country_outcomes = {k:v for k,v in outcomes.items() if variable+'['+country+',' in
    for origin in country_outcomes.keys():
        if country_outcomes[origin].mean()>50000:
            origin_country = origin.strip('migrants in destination country['+country+'
            x[origin_country]=country_outcomes[origin].mean()
    x.plot.bar(title = 'Migrants in '+country+' by Country of Origin', colormap='viridi
```

In [ ]:
```
x.plot.bar(title = 'Migrants in Germany by Country of Origin').legend(bbox_to_anchor=(
```

In [ ]:
```
list(x.columns)
```

In [ ]:
```
help(plotly.offline.iplot)
```

In [ ]:
```
outcomes.keys()
```

# E

# R Scripts

## E.1. Conversion of Country Name Data

```
library(readxl)
cowcdata <- read_excel("/Users/stefanwigman/Desktop/
Migration Thesis/03 - Data/CountryCode Conversion R
/COW_Countries.xlsx", 1)

countrydata <- data.frame(country.name =
countrycode(cowcdata$cow_code, "cowc", "country.name"))
countrydata$country.name.de <-
countrycode(countrydata$country.name, "country.name", "country.name.de")
countrydata$cowc <-
countrycode(countrydata$country.name, "country.name", "cowc")
countrydata$cown <-
countrycode(countrydata$country.name, "country.name", "cown")
countrydata$ecb <-
countrycode(countrydata$country.name, "country.name", "ecb")
countrydata$eurostat <-
countrycode(countrydata$country.name, "country.name", "eurostat")
countrydata$fao <-
countrycode(countrydata$country.name, "country.name", "fao")
countrydata$fips <-
countrycode(countrydata$country.name, "country.name", "fips")
countrydata$gaul <-
countrycode(countrydata$country.name, "country.name", "gaul")
countrydata$genc2c <-
countrycode(countrydata$country.name, "country.name", "genc2c")
countrydata$genc3c <-
countrycode(countrydata$country.name, "country.name", "genc3c")
countrydata$genc3n <-
countrycode(countrydata$country.name, "country.name", "genc3n")
countrydata$imf <-
countrycode(countrydata$country.name, "country.name", "imf")
countrydata$ioc <-
countrycode(countrydata$country.name, "country.name", "ioc")
countrydata$iso2c <-
countrycode(countrydata$country.name, "country.name", "iso2c")
countrydata$ico3c <-
countrycode(countrydata$country.name, "country.name", "iso3c")
countrydata$iso2n <-
```

```
countrycode(countrydata$country.name, "country.name", "iso2n")
countrydata$ico3n <-
countrycode(countrydata$country.name, "country.name", "iso3n")
countrydata$p4n <-
countrycode(countrydata$country.name, "country.name", "p4n")
countrydata$p4c <-
countrycode(countrydata$country.name, "country.name", "p4c")
countrydata$un <-
countrycode(countrydata$country.name, "country.name", "un")
countrydata$unpd <-
countrycode(countrydata$country.name, "country.name", "unpd")
countrydata$wb <-
countrycode(countrydata$country.name, "country.name", "wb")
countrydata$wvs <-
countrycode(countrydata$country.name, "country.name", "wvs")

countrydata$ar5 <-
countrycode(countrydata$country.name, "country.name", "ar5")
countrydata$continent <-
countrycode(countrydata$country.name, "country.name", "continent")
countrydata$cow.name <-
countrycode(countrydata$country.name, "cowc", "cow.name")
countrydata$ecb.name <-
countrycode(countrydata$country.name, "country.name", "ecb.name")
countrydata$eurocontrol_pru <-
countrycode(countrydata$country.name, "country.name", "eurocontrol_pru")
countrydata$eurocontrol_statfor <-
countrycode(countrydata$country.name, "country.name", "eurocontrol_statfor")
countrydata$eu28 <-
countrycode(countrydata$country.name, "country.name", "eu28")
countrydata$fao.name <-
countrycode(countrydata$country.name, "country.name", "fao.name")
countrydata$fips.name <-
countrycode(countrydata$country.name, "country.name", "fips.name")
countrydata$genc.name <-
countrycode(countrydata$country.name, "country.name", "genc.name")
countrydata$icao.name <-
countrycode(countrydata$country.name, "country.name", "icao.name")
countrydata$icao_region <-
countrycode(countrydata$country.name, "country.name", "icao_region")
countrydata$ioc.name <-
countrycode(countrydata$country.name, "country.name", "ioc.name")
countrydata$iso.name.en <-
countrycode(countrydata$country.name, "country.name", "iso.name.en")
countrydata$iso.name.fr <-
countrycode(countrydata$country.name, "country.name", "iso.name.fr")
countrydata$p4.name <-
countrycode(countrydata$country.name, "country.name", "p4.name")
countrydata$region <-
countrycode(countrydata$country.name, "country.name", "region.name")
countrydata$un.name.ar <-
countrycode(countrydata$country.name, "country.name", "un.name.ar")
countrydata$un.name.en <-
countrycode(countrydata$country.name, "country.name", "un.name.en")
countrydata$un.name.fr <-
countrycode(countrydata$country.name, "country.name", "un.name.fr")
```

```
countrydata$un.name.ru <-
countrycode(countrydata$country.name, "country.name", "un.name.ru")
countrydata$un.name.zh <-
countrycode(countrydata$country.name, "country.name", "un.name.zh")
countrydata$unpd.name <-
countrycode(countrydata$country.name, "country.name", "unpd.name")
countrydata$wvs.name <-
countrycode(countrydata$country.name, "country.name", "wvs.name")
# countrydata$cldr.* <-
countrycode(countrydata$cldr, "country.name", "cldr.*")

codelist1 <- codelist

library(readxl)
cowcdata <- read_excel("/Users/stefanwigman/Desktop/Migration Thesis
/03 - Data/CountryCode Conversion R/COW_Countries.xlsx", 1)

countrydata <-
data.frame(country.name = countrycode(cowcdata$cow_code, "cowc", "country.name"))

for (code in names(codelist)){countrydata$code <-
countrycode(countrydata$country.name, "country.name", code)}

library(countrycode)
codelist2<-codelist
write.table(codelist2, "codelist.txt", sep=",")
```