Distinguishing 'Sugar', 'Gravel, 'Flowers' and 'Fish' mesoscale cloud patterns in the trades using cluster analysis

Gyan Luchmun (4845889)

Bachelor Final Project
TU Delft, Faculty of EEMCS,
BSc program Applied Mathematics,
TU Delft, Faculty of Applied Sciences,
BSc program Applied Physics

Delft, December 11, 2022

Supervisors:
Dr. Franziska Glassmeier
Dr. Maarten van Hoven
Dr. Aurèle Adam
Dr. Paul Visser

Abstract

In this thesis we are interested in distinguishing patterns of mesoscale cloud patterns in the trades. Specifically, whether Sugar, Gravel, Fish and Flowers patterns can objectively be identified using physical quantities. For this purpose, we use cloud fraction data attained by the CORAL Ka-Band cloud radar at the Barbados Cloud Observatory during the boreal winter seasons of 2018, 2019 and 2020. These cloud fraction data represents the curves up until a height of 4 km for a given 6-hour interval of time. We do this to see if these clusters match up with the labels assigned to each cloud fraction curve obtained from a classification model used in Schulz (2021). Firstly, we map the cloud fraction curves onto points on a finite dimensional space using functional principal component analysis. We subsequently apply K-means, Gaussian Mixture Models and Mean Shift clustering onto the pre-processed dataset to identify any robust clusters. We have been able to attain robust Sugar-like clusters for K-means for 3 and 4 partitions and Mean Shift with bandwidth $\lambda \approx 585$. This provides evidence that we are able to use cloud fraction data to distinguish Sugar. However, the same can not be said for Gravel, Fish and Flowers as we have not been able to identify them in our analysis. It is suggested for future research to do sensitivity analysis in the height interval of the cloud fraction data, that outliers are omitted and that the labeled data from Schulz (2021) are used instead of the mean and spread of the data pertaining to those labels.

Contents

Part I
Introduction

> Humans are good at discerning subtle patterns that are really there, but equally so at imagining them when they are altogether absent.
>
> *Carl Sagan*

Imagine you are relaxing in the scorching Aruban sun off its beautiful coast on its easternmost point. You are completely in trance from the environment and as you look up at the sky you notice something peculiar that brings you back to reality and makes you wonder: 'Why are these clouds waiting in line?'. What you end up seeing looks something like figure 1. You might shrug it off as a coincidence until you see it occur for a second, third or fourth time. On satellite imagery, a possible cloud field that corresponds with this phenomenon is shown in figure 2.

Figure 1: This is a panoramic view of one of the beaches near the easternmost point of the island. We see some row-like formation of clouds towards the south of the island between the coasts of Aruba and Venezuela. From ref. [1].

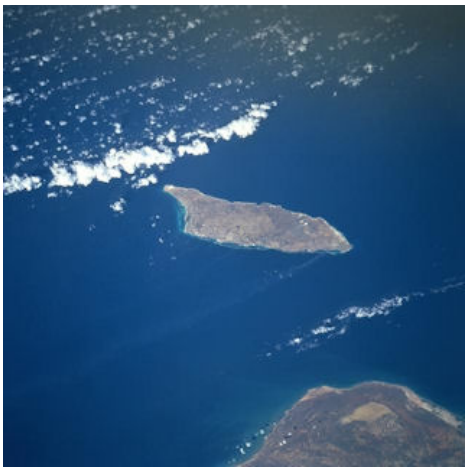


Figure 2: This figure illustrates a satellite image with cloud streets that can be seen above and below the island of Aruba. The cloud street seen in figure 1 has a similar location and orientation as the cloud street that can be seen to be between Aruba and Venezuela. From ref. [2].

.

This is an example of what it means to have spatial organization of clouds and these occur under specific conditions. However, the processes that drive the organization of clouds under such conditions tend to not be fully understood. It is not clear whether the clouds organise themselves and/or organise due to changes in the large-scale environment. This makes it difficult to incorporate it into weather models. This gives us an incentive to investigate these systems to further implement them into climate models [3].
Cloud streets are not the only example of mesoscale cloud organization. In the trades, there happen to be four additional ways that cloud organise themselves in the order of 2-2000 km (mesoscale) throughout the year. These additional ways of cloud organisation are denoted to be Sugar, Gravel, Fish and Flowers; and are shown briefly in figure 3.

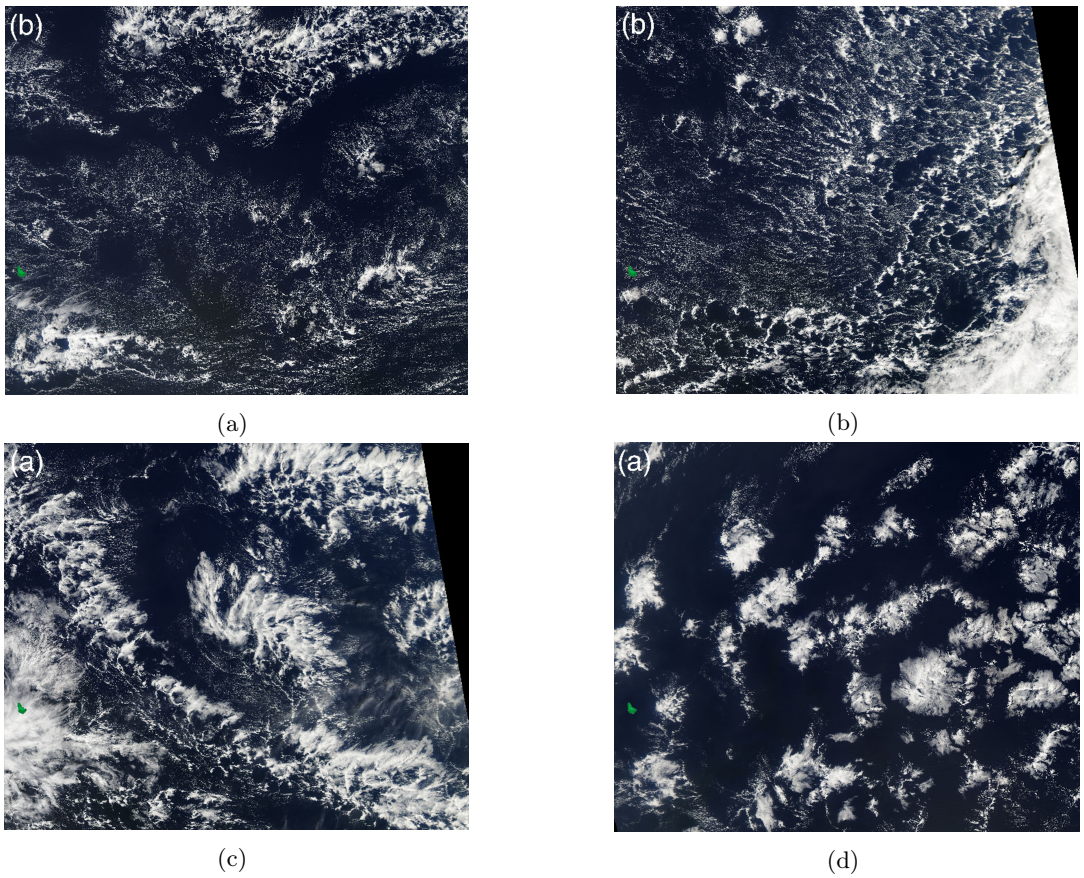


Figure 3: Here we show the four different cloud patterns that can occur in the trades near the island of Barbados. The subfigures a., b., c. and d. represent Sugar, Gravel, Fish and Flowers respectively. The greenly lit island in all of the figures represents Barbados. From ref. [4].

Broadly speaking, one can associate these patterns with varying levels of associated precipitation and cloudiness. It must also be noted that according to literature that climate change will effect the frequency of recurrence of these cloud patterns [4]. As the climate changes, conditions will favor Sugar and disfavor Fish and Flowers [4]. Because Sugar is a 'sunny' pattern, we therefore expect clouds, on average, to reflect less sunlight back to space. This shifts the planetary energy balance and causes the climate to therefore become even warmer. Understanding these patterns will help us improve climate models and also help us to understand climate change and its effects [3].
For the past years, researchers have been investigating these different cloud patterns. Due to the visual nature of this phenomenon, researchers in the field have conducted classification studies. Given we are able to label satellite images based off some classification model, we can find physical quantities associated with that label. However, it should be noted that these physical quantities that pertain to some label are not necessarily distinct. Distinct in the sense that physical quantities

belonging to the same label are more similar with respect to physical quantities coming from different labels. This touches upon what is known as clustering; where physical quantities belonging to clusters have the aforementioned property. The epigraph in the beginning of this chapter tells us it is not at all obvious whether clusters might be visually distinct. To truly find and determine clusters we use a combination of mathematics and an all-around understanding of these cloud patterns to base our judgement upon. In this thesis we will investigate whether we are able to find reasonable clusters that pertain to labels that have been found using a classification model. As for the labeled physical quantity, we use cloud fraction as a function of height up to 4 km. This physical quantity, the cloud fraction, is the closest related quantity to these purely visually defined cloud patterns and will hopefully allow us to answer our research question [4].
In chapter 2 we the necessary theory that should be understood for our thesis. In chapter 3 we present our results and discuss these. Chapter 4 gives the conclusions drawn by this analysis. The thesis is part of the Bachelor Final Project that is part of the double bachelor Applied Mathematics and Applied Physics degree at the Delft University of Technology.

Part II
Theory

In this chapter we go through the theory that is relevant for having a proper understanding regarding the intricacies of this thesis. We start by talking about the instrument involved in measuring the data that we use. Then, we give some context regarding the literature of these four patterns. Subsequently, we explain how we pre-process the data before we move onto the mathematical topic of clustering and finish this chapter off with the methods involved in answering our research question.

1 Instrumentation and Data

In this section we elaborate on the relevant physical quantities involved in this research and the instrument involved in measuring those.

1.1 The Radar Equation

Detection at a distance using radars are done by comparing the transmitted EM-wave, with known properties, alongside the reflected EM-wave that has bounced off some target we are interested in investigating. These targets are denoted as 'point targets' and are of negligible spatial extent. This negligibility is due to the sheer distance between the target and the antenna of the radar [5]. To understand what is happening, one must first consider the an antenna radiating its energy isotropically into space. Meaning: radiation from a point source without preferred direction. Suppose we are interested in the power P_σ that pours through an area σ of little radial extent at a distance of r from an antenna which generates power P_t. To relate these quantities, we invoke the inverse quadratic relationship of power at a distance from an isotropically radiating source as shown in equation 1.

$$P_\sigma = \frac{A_\sigma}{4\pi r^2} P_t. \tag{1}$$

What a radar is able to do with this antenna, is it is able to control the amount of power that it pours through A_σ. It does this by focusing its energy into a beam onto the point target instead of simply allowing the antenna to radiate the energy isotropically. Then, we can describe the power flowing through A_σ in this scenario to be proportional to the isotropic case. This constant of proportionality G is denoted to be the antenna axial-gain. The modified equation for the power flowing through σ is shown in equation 2.

$$P_\sigma = \frac{GA_\sigma}{4\pi r^2} P_t. \tag{2}$$

G happens to be related with the aperture area A_e and is given in equation 3 [5].

$$G = \frac{4\pi A_e}{\lambda^2}. \tag{3}$$

Now that we know the power flowing through the target point due to a beam of wavelength λ, we must now ask ourselves what the measured power will be due to reflection. If we measure using antenna with aperture A_e, which again is far enough from the source to be described as spatially negligible, we are eventually able to describe the reflected power P_r as:

$$P_r = \frac{A_e}{4\pi r^2} P_\sigma = \frac{GA_\sigma A_e}{(4\pi r^2)^2} P_t. \tag{4}$$

By writing our aperture area A_e in terms of gain G and the beam wavelength λ in equation 4, we eventually arrive at equation 5.

$$P_r = \frac{G^2\lambda^2}{(4\pi)^3 r^4} P_t A_\sigma. \tag{5}$$

Because this equation unfortunately relies upon the assumption that our target radiates isotropically, we use a slightly different expression. We, instead, use the radar equation for a single target, which is shown in equation 6 [5].

$$P_r = \frac{G^2\lambda^2}{(4\pi)^3 r^4} P_t \sigma. \tag{6}$$

The σ is known as the backscatter cross-section. This is not A_σ, rather some correlatory value that matches observations tied to the target and is given in equation 7.

$$\sigma = \frac{\pi^5}{\lambda^4} |K|^2 D^6. \tag{7}$$

D represents the diameter of the spherical target and K is a coefficient depending of the refractive index and absorption coefficient of the target. $|K| \approx 0.93$ for water [5].

1.2 The Reflectivity Z

Rain and cloud droplets belong to an important class of radar targets knows as distributed targets. To be able to measure a distribution of these, confined to a small space, we take measured averages of timescales greater than 10^{-2} s. We do this to average-out any high frequency power fluctuation that the radar might be able to pick up. This is shown in equation 8.

$$\bar{P}_r = \frac{G^2\lambda^2}{(4\pi)^3 r^4} P_t \sum_i \sigma_i. \tag{8}$$

$\sum_i \sigma_i$ indicates the sum of backscatter cross-sections pertaining to the distribution of targets reflecting the generated EM-waves [5]. By combining equations 8 and 7 we are eventually able to arrive at the measured reflected power average by a number of spherical scatterers as,

$$\bar{P}_r = \frac{G^2\pi^5|K|^2}{(4\pi)^3 r^4 \lambda^2} P_t \sum_i D_i^6. \tag{9}$$

This is where the motivation behind the quantity 'reflectivity' Z becomes clear. We express the reflectivity Z as $\sum_i D_i^6$. Supposing we know the distribution of the diameter of these objects $N(D)$ we are able to write Z as,

$$Z = \sum_i D_i^6 = \int_0^\infty N(D) D^6 \mathrm{d}D. \tag{10}$$

$N(D)$ are associated with the distribution of the size of these scatterers, whether it may be raindrops or snowflakes [5]. Another related quantity is the logarithmic reflectivity. Figure 4 illustrates how this logarithmic reflectivity might be used.

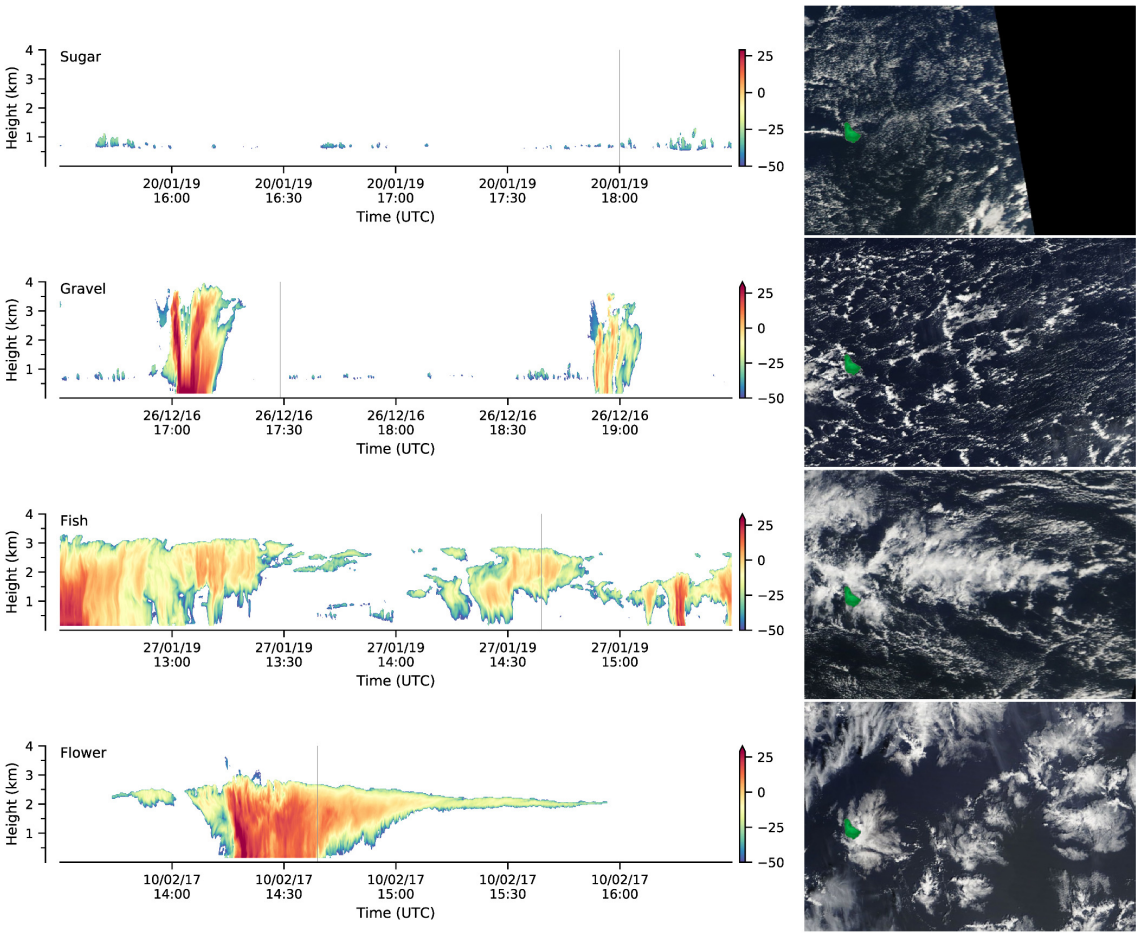


Figure 4: This figure shows logarithmic reflectivity as a function of height and time values for Sugar, Gravel, Fish and Flower patterns measured at the Barbados Cloud Observatory. The greenly lit island in the right-hand side of the figure represents Barbados. From ref. [6].

This will become useful for calculating cloud fractions. To summarize, reflectivity can be measured by parameters that both depend on the radar and the target.

1.3 Cloud Fraction

The need to be able to quantify cloudiness at a location during some period of time motivates the use of a quantity which is known as the cloud fraction. The cloud fraction is essentially the percentage of 'cloudy time' during some time spent measuring whether a location was 'cloudy'. Although the latter might be an intuitive definition, cloud fraction is more often defined geometrically as the fraction of a scene that is cloud filled. Suppose we want to measure the cloud fraction over some time interval T. If we have conducted, say, N reflectivity measurements during this time, we are able to make a statement on this average 'cloudiness'. 'Cloudiness' at a location is ascertained by checking if the reflectivity at the location is high enough. This can mathematically be described as a function C that takes on logarithmic reflectivity. This is shown in the following equation,

$$C(Z) = \begin{cases} 1 & \text{for } \log(Z) > -50 \text{ dB}, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

Observe in equation 11 that for logarithmic reflectivities -50 dB and higher we assign $C \mapsto 1$ and $C \mapsto 0$ otherwise. We choose -50 dB to be our threshold to not detect sea-salt aerosols [4]. The cloud fraction is estimated by calculating the mean of these one-zero realizations with respect to time.

1.4 CORAL Ka-Band Cloud Radar Characteristics

The cloud radar whose data we use in this thesis comes from the Barbados Cloud Observatory (BCO). A picture of this radar is shown in figure 5. The features of this radar are show in table 1. For the purposes of this thesis, the reflectivity measurements of this device will be of relevance.

Figure 5: This figure shows the CORAL Ka-Band Cloud Radar at the Barbados Cloud Observatory. We use its measurements for the purposes of this thesis. From ref. [7].

Radar Type:	Mono static, pulsed, magnetron
Frequency (related to λ):	35.5 Ghz $\pm$ 150 MHz
Diameter of Antenna (related to A_e):	2 m
Peak Power:	30 kW
Pulse Width:	200 ns for 30 m range resolution
Anatenna Beam Width:	0.3 deg x 0.3 deg
Sensitivity:	-48 dBz at 5 km, -70 dBz at 500 m

Table 1: The properties of the CORAL Ka-Band Cloud Radar at the Barbados Cloud Observatory relevant for this thesis. From ref. [7].

2 Mesoscale Cloud Organization: Sugar, Flower, Fish and Gravel

Within the tropics we have some recurrence regarding the ways clouds might organize themselves in the trades. Sugar, Flower, Fish and Gravel happen to be the names of these different types of recurring cloud organization. These clouds organize themselves in the order of 20 to 2000 km and are therefore denoted as mesoscale cloud organization [6].

2.1 Complex Systems

It is useful to consider the hypothesis that the visual properties of these patterns are emergent properties coming from an underlying complex system. Arising due to the internal interactions from the complex system itself. We therefore often speak about self-organization because this describes the relationship between internal interaction and the resulting visual properties. Complex systems are systems, naturally evolving through time, that can be described on a potential landscape. The potential landscape can be described using hills and valleys. Regions respectively of local stability and instability. As for the remaining part of what influences a system's destiny are external influences. These are all grounded in the deterministic laws which are naturally able to model

this non-linear behaviour. When attempting to describe measurable properties arising from such complex systems, a probabilistic approach is preferred over a traditional deterministic one [8].

2.2 Visual Identification

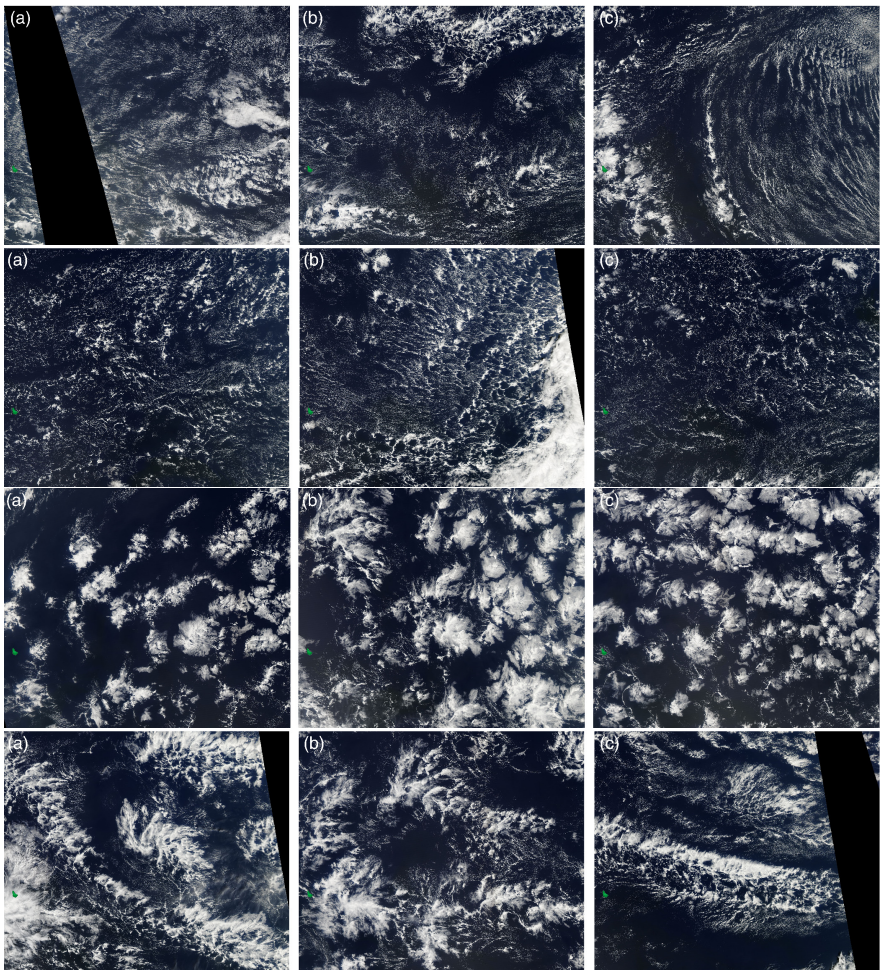


Figure 6: This figure illustrates a collection of satellite images. The rows of these shows how clouds might organize themselves during what is classified to be 'Sugar', 'Gravel', 'Flower' and 'Fish' respectively. The columns represent different realizations of the same pattern. From ref. [6].

The first row of figure 6 shows an example of what is classified to be Sugar. Sugar is associated with fine dust-like clouds with no precipitation and shows little to no self-organization [6]. Gravel is associated with low precipitation. This pattern is characterized by structures the size of 20 to 100 km. Cell-like cloud patterns can be seen if one closely observes the second row of satellite imagery in figure 6. These cell-like cloud patterns are also known cold pools [6]. These cells, which are visible in figure 6, are formed where gust-fronts collide that accompany these cold pools. Leading to brighter clouds than Sugar. These deeper tower-like clouds precipitate and are illustrated in figure 4. Flower is associated precipitation. This state is characteristical due to the cloud-decks and tower-like clouds which are shown in the third row in figure 6. The size of these flower like cloud structures are in the scale of 20 km to 200 km and are often well separated from each other due to the presence of defined cloud-free areas. Fish can be identified from the fish-bone like cloud structure that can be seen in the final row in figure 6. These fish-bone like structure span 200 to 2000 km and are clearly separated from each other. Lastly, it is important to note that this pattern is associated with precipitation.

2.3 Classification

The ability of being able to distinguish these patterns visually has motivated people to conduct classification studies. Schulz (2021), among which, have been able to classify 6-hour cloud fraction data from the Barbados Clouds Observatory spanning the boreal winter seasons from 2018 (January-February-March), 2019 (November-December-January-February-March) and 2020 (November-December-January-February-March). These also happen to be considered dry seasons for Barbados. Dry season is generally associated with warm days, cool nights and relatively little rainfall [9]. Roughly 90% of the 6-hour time windows are available and the remaining 10% are missing time windows [4].
Schulz (2021) has been able to label these time windows using the object detection algorithm Keras RetinaNet. They had trained this neural network using 49,000 manually created labels from 10 years of satellite imagery as input data [4]. Some main results from this classification study are given in figures 7 and 8.

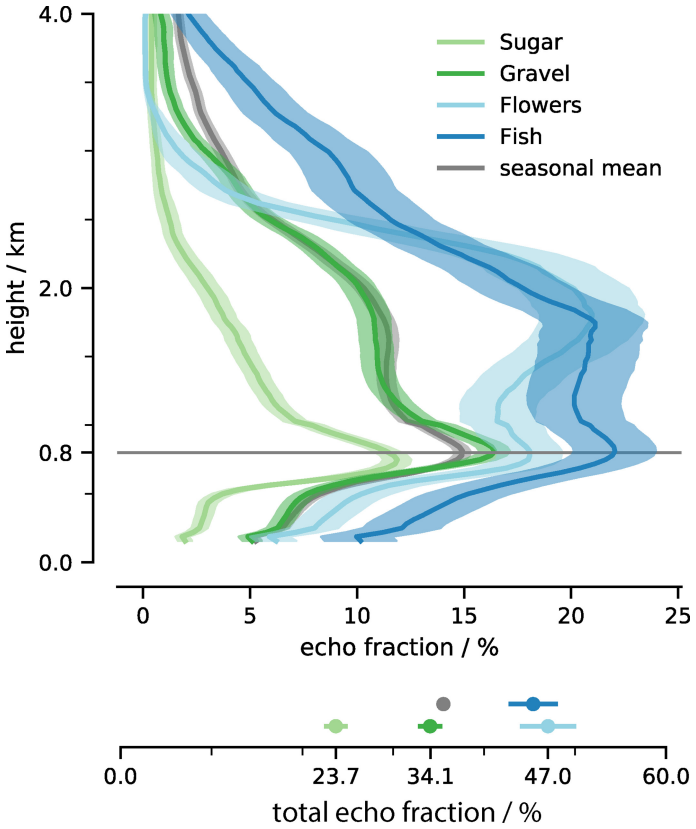


Figure 7: This figure illustrates the Schulz (2021) classification means alongside their standard deviation. The data corresponds with classified cloud fraction data that has been captured using the Ka-band cloud radar at BCO. From ref. [4].

pattern	# of 6 h windows	% of total	% of robust patterns
Sugar	145	9	22
Gravel	305	19	46
Flowers	77	5	12
Fish	141	9	21
Others	846	58	N/A
mixed	567	36	
no pattern	337	21	

Figure 8: This table shows the frequency of each classification label from Schulz (2021). The frequency is given in terms of 6 hours windows, percentage of the total windows and percentage of robust patterns. From ref. [4].

Although Sugar has been labeled for 9% of all the 6-hour time windows according to figure 8, it must be remarked that this classification study underemphasizes Sugar [4]. Sugar, the state that shows little to no self-organization with randomly distributed grain-sized clouds, appears very often. However, it is picked out by their analysis due to not often being dominant on a satellite image when classifying these 49,000 satellite images [4]. These satellite images represent too large of a domain on which Sugar is underrepresented through the classification bias towards larger scale phenomenon.

3 Functional Principal Component Analysis

Functional data analysis entails that we have observations in the form of smooth functions whose discrete counterpart we are able to capture using, say, n observations with d attributes. In our thesis, we model cloud fraction X as the realization of a stochastic process on a height interval $[0, H]$. Such data may be described as realizations of smooth random functions $X_1(h), \ldots, X_n(h) \in L^2[0, H]$; denoting the n sampled discrete observations as $X_{ij} = X_i(h_j)$ for $(i, j) \in [1, n] \times [1, d]$. For the purposes of clustering, we are able to go about this problem in different ways as shown in figure 9.

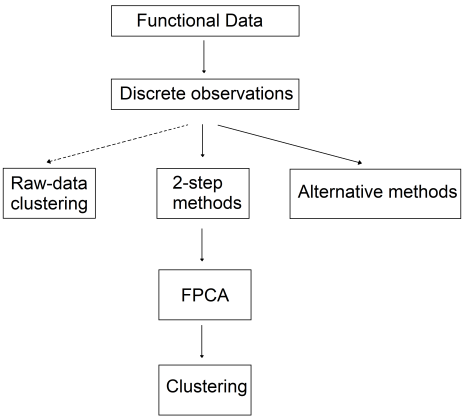


Figure 9: This figure illustrates a tree diagram which indicates the different approaches when it comes to clustering functional data according to literature. The node of the tree diagram representing FPCA denotes Functional Principal Component Analysis [10].

As shown in the most left branch of figure 9, one might consider capturing each raw observation X_i in the form of a d-dimensional vector $(X_{ij})_{1\leq j\leq d}$ on which we apply the clustering technique of interest. However, d is usually too large to reasonably be able to apply any statistical analysis due to the curse of dimensionality. For increasing dimension d we can expect the meaningfulness of any distance function to worsen as it's ability to discriminate between points becomes hampered [11]. To remedy this issue, we instead try to find an appropriate projection of our data onto a lower dimensional subspace; on which we can then apply our clustering techniques. This way of going about clustering functional data is illustrated in the 2-step methods branch of figure 9. This branch in the illustration tells us that we start by applying FPCA to reduce a discrete representation of a curve in L^2 to a point in $\mathbb{R}^{d_*}$, to thereafter be able to appropriately apply clustering techniques. The natural number d_* represents the minimum amount of attributes necessary to capture the exact same observation in L^2. Hence, this is called a 2-step method. In what follows, we explain the underlying concepts of FPCA. We do this by explaining how the Karhunen-Loève expansion is used to be able to help us arrive at our dimensionally reduced dataset. This is shown in the following theorem:

Theorem 3.1. *(Karhunen-Loève expansion) A stochastic process X in $L^2(\mathcal{I})$ can be expressed as:*

$$X(h) = \mu(h) + \sum_{k=1}^{\infty} \sqrt{\lambda_k}\xi_k\phi_k(h). \tag{12}$$

Where (λ_k) is a non-increasing sequence of positive numbers, (ϕ_k) an orthogonal sequence on $L^2(\mathcal{I})$ and $\xi_k = 1/\sqrt{\lambda_k}\int_{\mathcal{I}} X^c(v)\phi_k(v)\mathrm{d}v$ is an uncorrelated random variables with zero mean and unit variance for all k [12].

Infer how we are able to write our observation as the sum of the mean observation and a linear combination modes of which any further variation is composed of according to theorem 3.1. A sum whose sequence is ordered in importance; starting from the most dominant mode. Note that $(\lambda_k)_{k\geq 1}$ represents a monotonically decreasing sequence of eigenvalues corresponding to the variance of each mode. If we are interested in finding the least amount of attributes to reasonable be able to model the same observations, our problem boils down to trying to find a suitable cut-off point for our monotonically increasing converging sum. A useful tool for the purposes of choosing this cut-off point is the explained variance $(EV)_i = \lambda_i/\sum_k \lambda_k$. This tells us, mode-wise, the quality of how well we are able to reconstruct observations on a lower dimensional subspace. By taking an informed decision using the $((EV)_k)_{k\geq 1}$ we arrive at, we are able to choose a suitable cut-off within our infinite sum; thus representing our data as follows:

$$X_i(h) = \mu(h) + \sum_{k=1}^{d_*} c_{ik}\phi_k(h). \tag{13}$$

$\boldsymbol{c}_i = (c_{ik})_{1\leq k\leq d_*} \in \mathbb{R}^{d_*}$ is a sequence containing the ordered FPCA scores that are associated with some observed cloud fraction curve X_i. This is the essence of FPCA. We minimize the number of attributes necessary to quantitatively describe the exact same observation. In this thesis, the FPCA scores corresponding to our observed cloud fraction curves alongside its explained variances will be computed using Scikit-FDA package [13][14]. An example application this method is presented in figure 10.

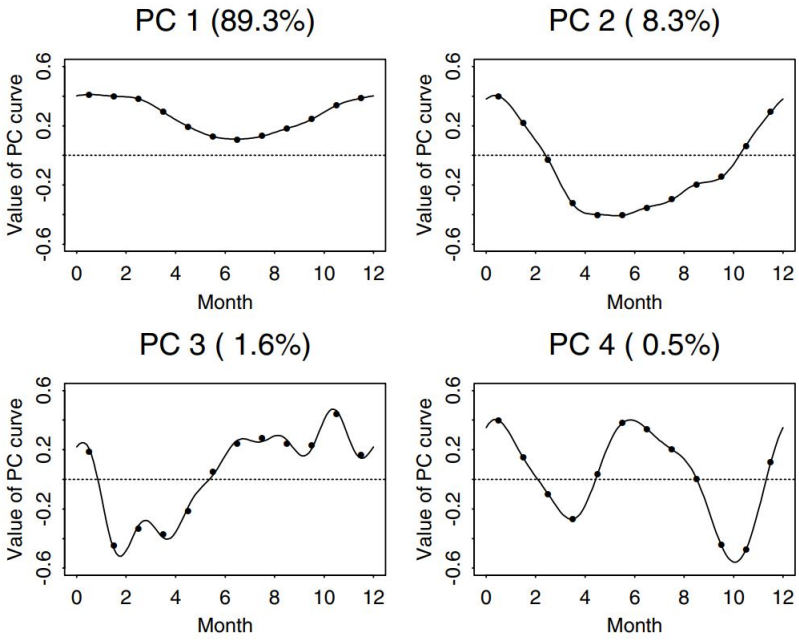


Figure 10: These graphs represent the functional principal components of some Canadian Temperature data. The percentage represents the amounts of variation that can be explained from its corresponding functional principal component. From ref. [14].

Apparently more than 95% of any further deviation from the mean can be explained using the first two functional principal components according to figure 10. Therefore, we are able to justify a change of basis on which we express observations in terms of linear combination of these two functional principal components.

4 Clustering

Clustering is about finding groups of observations such that the observations within a group are more 'similar' with respect to observations pertaining to other groups. Before we are able to talk about the clustering of observations with $d \in \mathbb{N}$ attributes, that we may represent as vectors $\boldsymbol{x} \in \mathbb{R}^d$, it becomes of importance to us to understand what defines 'closeness' or 'remoteness' between observations within our data set. In other words: How do we define the distance between two observations? An interpretation of such a distance is called a proximity measure. To give an example, a commonly used proximity measure is the Euclidean metric [15].
Due to the generality and subjective nature of the concept of clustering, exists countless of ways to cluster a given data set [15][16]. Thus, if we want to meaningfully cluster any data set, it is important to make choices based off knowledge regarding the system from which we are making our observations from. Or maybe upon rather the lack of which. Different clustering algorithms yield different interpretations on what the optimal grouping of observations should be. A comparison of the clustering algorithms that are available in the Scikit-learn python package are illustrated in figure 11.

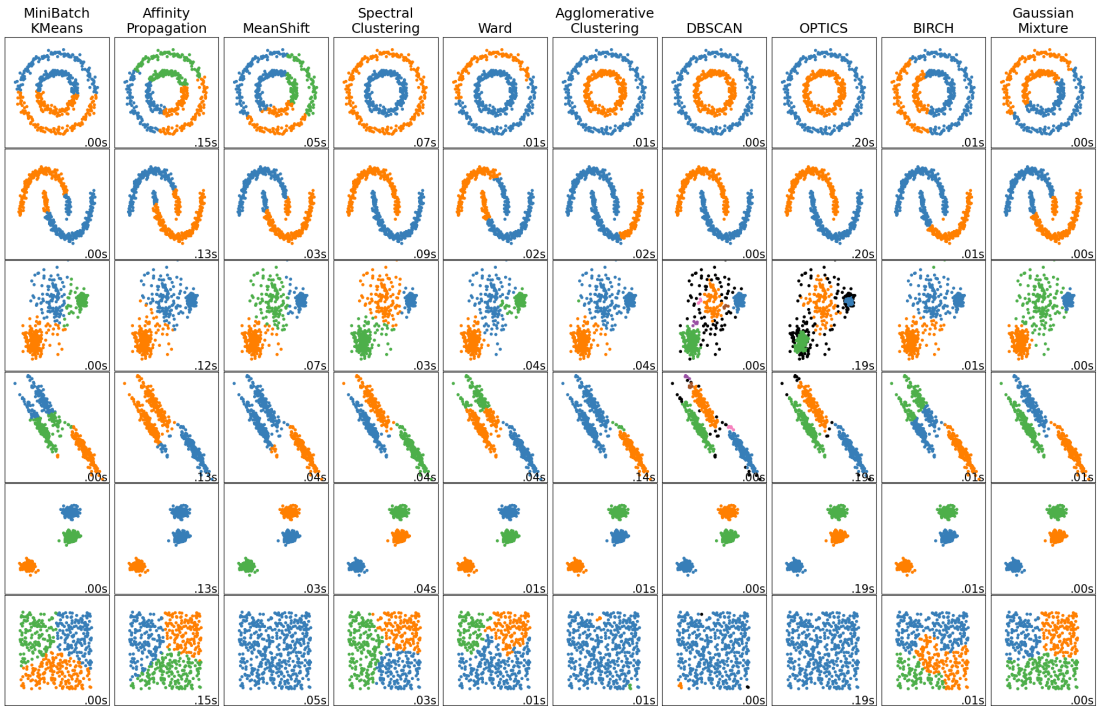
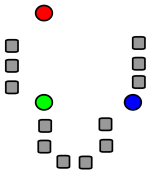


Figure 11: This figure illustrates the application of various clustering algorithms onto two-dimensional toy datasets using the Scikit-learn package. Connectivity-based algorithms: Ward, Agglomerative Clustering and BIRCH. Centroid-based algorithms: MiniBatch KMeans and Affinity Propagation. Distribution-based algorithms: Gaussian Mixture. Density-based algorithms: MeanShift, DBSCAN and OPTICS. Graph-based algorithm: Spectral Clustering. From ref. [17].

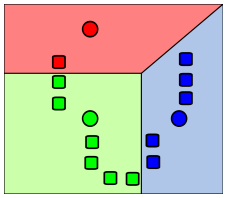
The clustering algorithms that are illustrated in figure 11 can broadly be categorized into connectivity-, centroid-, distribution- and density-based clustering algorithms. For the purposes of this thesis, we are interested in choosing methods that are able to take into account that our observations can be seen as the realizations of some random variable. Therefore, we are interested in applying centroid-based, distribution-based and density-based methods to find meaningful clusters. Examples of executing these algorithms onto different two-dimensional toy datasets are shown in figure 11. We use these techniques in section III to arrive at a substantiated answer for our research question. In the following subsections we are going to sketch the general idea behind these clustering algorithms alongside their strengths and weaknesses.

4.1 Centroid-based Clustering: K-means

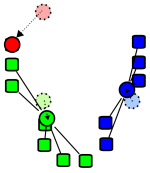
When going about clustering with K-means, we attempt to find a partition of the data such that the average intra-cluster variance is minimized. An example execution of this algorithm along with this solution is sketched in figure 12.



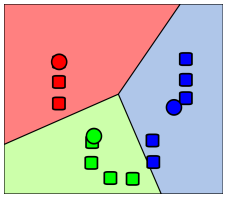
(a) We start by initializing the centroids. (https://commons.wikimedia.org/wiki/File:K_Means_Example_Step_1.svg).



(b) In this step we assign all observations to the closest centroid. (https://commons.wikimedia.org/wiki/File:K_Means_Example_Step_2.svg).



(c) We assign new centroids. We keep assigning new centroids until the algorithm converges. (https://commons.wikimedia.org/wiki/File:K_Means_Example_Step_3.svg).



(d) The algorithm arrives at an optimal solution. (https://commons.wikimedia.org/wiki/File:K_Means_Example_Step_4.svg).

Figure 12: This figure shows the general steps one encounters when executing the K-means algorithm to find clusters. The circles are centroids and the blocks represents the data. The shaded regions in figures b. and d. displays the points closer to a certain centroid than any other. From "Wikimedia Commons," by P. Weston, 2007. Licensed under CC BY-SA 3.0.

Consider n collected samples of data to be vectors $\boldsymbol{x}_i \in \mathbb{R}^d$, for $1 \leq i \leq n$ and denote some partition of the data as $\boldsymbol{C} = \{C_1, C_2, \ldots, C_m\}$ with $m \leq n$. Taking $\boldsymbol{\mu}_i$ to be the mean for our partition C_i, for $1 \leq i \leq m$, allows us to describe the optimization problem at hand:

$$\underset{\boldsymbol{C}}{\operatorname{argmin}} \sum_{i=1}^{m} \sum_{\boldsymbol{x} \in C_i} || \boldsymbol{x} - \boldsymbol{\mu}_i ||^2 . \tag{14}$$

To obtain a solution, we go about initializing candidate means and choosing an algorithm to allows these means to converge to an optimal solution. Initialization entails make an educated guess towards our optimal partition. An initial starting position that we are able to feed to our algorithm. Please take note that this starting position is not obvious and should be chosen with care as different initial positions might lead to different optimal solutions [15]. Therefore, we start with what is called initialization and is shown in subfigure 12a. In this thesis we initialize using K-means++ and apply Lloyd's algorithm to find our optimal partition. These algorithms are implemented using Scikit-learn and are written in pseudo code in Algorithm 1 and Algorithm 2 respectively within appendix A [18].

4.2 Distribution-based Clustering: Gaussian Mixed Models (GMM)

The philosophy behind distribution-based clustering is that we assume that our clusters to be realizations of some random variable whose parameters are fitted to model the clusters that might be apparent in our observations. A well known example of such a distribution-based clustering method is Gaussian Mixed Models. To assume that our clusters are realizations of some Gaussian with mean μ and covariance matrix Σ is not a stretch in many cases. We are able to capture, not only, the location of these clusters, but also the correlation and dependence of the attributes involved within a cluster. This model happens to be excellent for e.g. modelling measurement error.
An obvious drawback of this method, despite its applicability, suffers from its underlying assumption. Assuming that all clusters follow some Gaussian distribution is a very strong assumption.

Any additional structure and complexity that might be present in the data will therefore completely be lost [19]. Another drawback is that, just like K-means, it is sensitive to its input parameters [15].
Distribution-based clustering using Gaussian Mixed Models boils down to assuming that all clusters are realizations of some Gaussian of mean μ and covariance matrix Σ. The complete model for some observation x_i can be expressed as follows:

$$p(x_i) = \sum_g \phi_g N(x_i|\mu_g, \Sigma_g). \tag{15}$$

In other words, the probability of x_i occurring can be expressed as the sum of membership probabilities multiplied by their associated probability. Obtaining μ_g and Σ_g for all g components comes down to finding candidate solutions and subsequently optimizing them to fit the observations. After applying K-means++ to initialize the candidate means for the Gaussian models, we find the optimal parameters belonging to our data using the Expectation-Maximization algorithm. This algorithm is illustrated as pseudo code in Algorithm 3 within appendix A and can be implemented using the Scikit-learn python package [18].

4.3 Density-based Clustering: Mean Shift

The Mean Shift algorithm makes no model assumptions, is able to capture complex-shaped clusters, has a physically interpretable input parameter, has no local minima and thus is also robust with respect to its input parameter; and is relatively insensitive to outliers [20]. These properties distinguishes itself from K-means and GMM as they are, however: sensitive to outliers, have the number of clusters as an input parameter and have local minima when it comes to finding optimal solutions [15]. Mean Shift is able to circumvent all of these issues with the benefit that clusters can be realizations of some random variable.
The Mean Shift clustering algorithm is also know as a 'mode-seeking' algorithm. When applying this algorithm we are trying to seek out 'modes' that might be present within the data. For each iteration t we try to find a centroid x^{t+1} whose neighbourhood $N(x_i^{t+1})$ is denser than the previous iteration $N(x_i^t)$. This is done through estimation of the direction where this increase in density is maximum. For this purpose we use a kernel $K(.)$; a function that effectively help assign weights onto neighbouring points. In other words, we continuously 'shift' our candidate 'mean' until convergence and subsequently occupies what it deems to be the most the most dense region. This iterative process where we update our centroid $x_i^t \rightarrow x_i^{t+1}$ can be shown as follows:

$$x^{t+1} = m(x^t) = \sum_{x_j \in N(x^t)} \frac{K(x_j - x^t)x_j}{\sum_{x_j \in N(x^t)} K(x_j - x^t)}. \tag{16}$$

This kernel depends the bandwidth λ, which is a parameter that indicates the relative size of the neighbourhood which contain useful data points. This bandwidth, λ, is estimated by ordering all the pairwise distances available in our data and by subsequently picking its median [18]. For the purposes of this thesis we will use what is called the flat-kernel for equation 16. An example sketch of a run of the Mean Shift algorithm using a flat-kernel onto an ideal geometry is given in figure 13.

$$K(x) = \begin{cases} 1 & \text{for } x < \lambda, \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

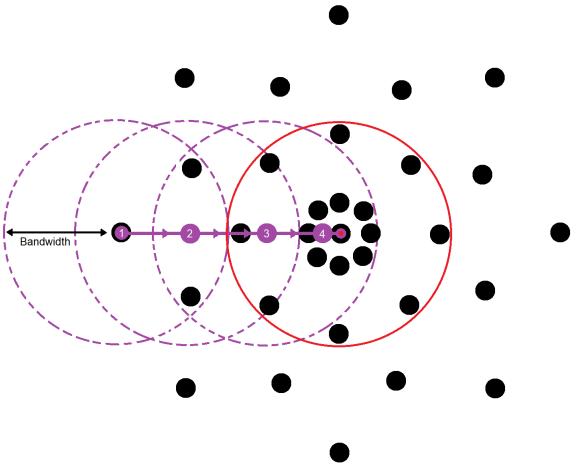


Figure 13: This is a sketch of some data whose density drops smoothly as a function of the distance away from its center. It also illustrates how the Mean Shift algorithm iterates to achieve a region where the density is optimal. The purple enumerated dotted line shows how the Mean Shift algorithm uses a flat-kernel to find another centroid whose neighbourhood is denser. These neighbourhoods are represented by circles that surround its centroid whose radius is defined as the pre-chosen parameter: the bandwidth. The red centroid represents the converged result.

This algorithm is also written as pseudo code in Algorithm 4 within appendix A and is implemented using Scikit-learn [21][18]. As seen in Algorithm 4, clusters are constructed by continuously placing a 'walker' until it converges which is shown in figure 13. Then the algorithm places all data that are at most a bandwidth distance away from each other in the same cluster. This is done until all observations are assigned to some cluster. However, it can occur that you have isolated points of data whose nearest neighbour is further than the bandwidth itself. These isolated points, or modes, are denoted to be orphans and can be omitted for the purposes of clustering.

4.4 Internal Validation

The power behind the application of these algorithms is that they rest upon the general assumption that we are able to extract distinct patterns from our data. To assess the quality of clusters, we apply silhouette plots [16]. An example of such a plot is shown in figure 14. Silhouette plots illustrate the ordered Silhouette scores of the observations pertaining to each cluster.

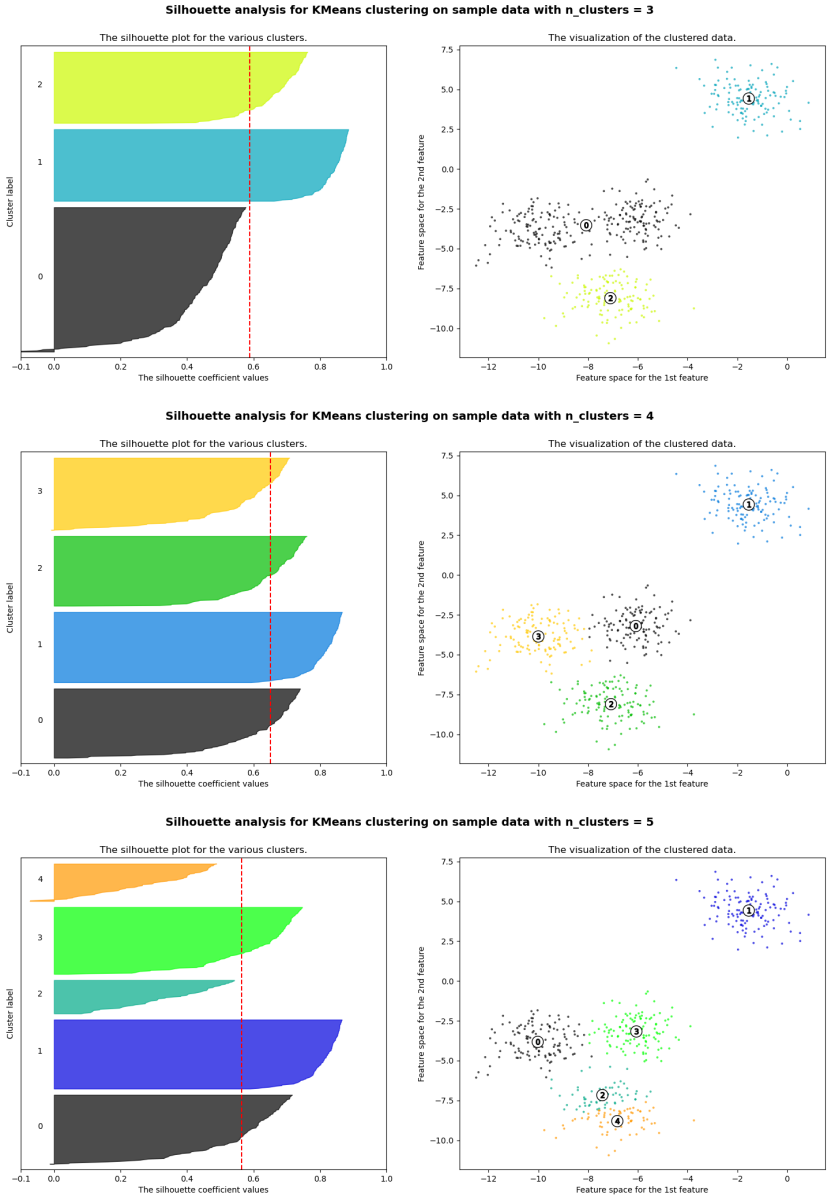


Figure 14: In this figure we show Silhouette plots for some data that has three distinct clusters. The Silhouette plots are given for three, four and five clusters respectively. From ref. [22].

The Silhouette score for some observation i can be expressed as the following:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}. \tag{18}$$

In equation 18, $a(i)$ represents the mean distance from observation i to the observations pertaining to its belonging cluster and $b(i)$ expresses the mean distance from observation i to observations pertaining to clusters that it does not belong to [23]. This value has the property that $-1 \leq s(i) \leq -1$ for any observation i and can be computed using the Scikit-learn python package [18].
Observe what happens when $s(i) \approx 1$, $s(i) \approx -1$ and $s(i) \approx 0$. For the first case we see that $b(i) >> a(i)$. This indicates strong evidence for belonging to its assigned cluster. As for the second case we have that $b(i) \approx a(i)$. This tells us that it is not clear whether the observation belongs to its assigned cluster. The final case corresponds with having $a(i) >> b(i)$. This, as opposed to the first case, provides strong evidence against belonging to its assigned cluster.

4.5 The Hopkins Statistic

We can test 'clusterability' in our data using the Hopkins statistic. This checks for clustering tendency by testing the null hypothesis, which in most cases is some random position hypothesis [24].
Suppose we have some data $X = (x_i)_{1 \leq i \leq n} \in \mathbb{R}^d$. We now generate $m << n$ samples $Y = (y_i)_{1 \leq i \leq m} \in \mathbb{R}^d$ that are randomly placed in a subspace of the d-dimensional space. Let us define u_j^d to be the distance of $y_j \in Y$ from its nearest neighbour in X and w_j^d to be the minimum distance between m randomly selected data $x \in X$ and their nearest neighbours. The Hopkins statistic in d-dimensions is shown in equation 19.

$$H = \frac{\sum_{j=0}^{m} u_j^d}{\sum_{j=0}^{m} u_j^d + \sum_{j=0}^{m} w_j^d}. \tag{19}$$

We test against the null hypothesis H_0: Our data is generated by some randomly uniform distribution. Note how it follows that, under H_0, the mean distance $\bar{u}^d$ and the mean $\bar{w}^d$ should roughly be the same. Therefore, $\sum_{j=0}^{m} u_j^d \approx \sum_{j=0}^{m} w_j^d$ in equation 19 and we arrive at $H = 0.5$. When $\bar{w}^d << \bar{u}^d$ or $\bar{w}^d >> \bar{u}^d$, we have $H = 0$ and $H = 1$ respectively. $H \neq 0.5$ apparently builds evidence against the random position hypothesis. When $H = 0$ holds, we have that the data is uniformly distributed and $H = 1$ tells us that our data is highly clustered. To estimate p-values using this statistic we make use of the fact that the test statistic H follows a Beta(m, m) distribution [24].

4.6 Determining the Number of Clusters

In this thesis we will use the Gap statistic, the Elbow method and Silhouette scores to arrive at an informed decision regarding what number of clusters might be present in the data.

4.6.1 The Elbow Method

The Elbow method presents a heuristic for determining the number of clusters available in the data. This is done by analyzing an average within-sum squares plot against the number of optimal partitions and by subsequently choosing its 'elbow'. This reflects the point at which the cost of adding an additional cluster becomes too much when considering the diminishing returns involved [16]. It can be computed using K-means through the Scikit-learn python package and its 'elbow' can be detected using an elbow detector which is referred to in the references [25]. An example how such an average within-sum squares plot might look like is given in subplot b. of figure 15. According to the Elbow method we infer the data in subplot a. of figure 15 to have two clusters because subplot b. indicates the presence of an elbow at $k = 2$.

4.6.2 Gap Statistic

The Gap statistic returns a value that reflects the contrast of the optimized cost functions of our algorithm between our data and some uniformly distributed data [26]. This method can be seen as a generalized version of the Elbow method and demonstrates good performance according to simulation study [16][26]. Before we find an expression of the Gap statistic it is important to understand that W_k expresses the sum of the average within-pairwise distances of all the clusters. For clusters $r \in \{1, 2, \ldots, K\}$ that we define as C_r and euclidean distance $d_{ii'}$ between observations i and i' helps us express W_k as:

$$W_k = \sum_r \frac{1}{n_r} \left(\frac{1}{2} \sum_{i,i' \in C_r} d_{ii'} \right). \tag{20}$$

The Gap statistic is then defined as,

$$\text{Gap}_k(n) = \mathbb{E}_n^*(\log(W_k)) - \log(W_k). \tag{21}$$

Where $\mathbb{E}_n^*$ defines the expected W_k using n samples generated using some reference distribution. An example of computing the Gap statistic for some data containing two distinct clusters is given in figure 15. The number of clusters we expect the data to have is the smallest k such that $\text{Gap}_k(n) \geq \text{Gap}_{k+1}(n) - s_{k+1}$ where s_{k+1} is the standard deviation of the Gap statistic at $k+1$ [26]. As one can see, this corresponds with $k = 2$ in figure 15.

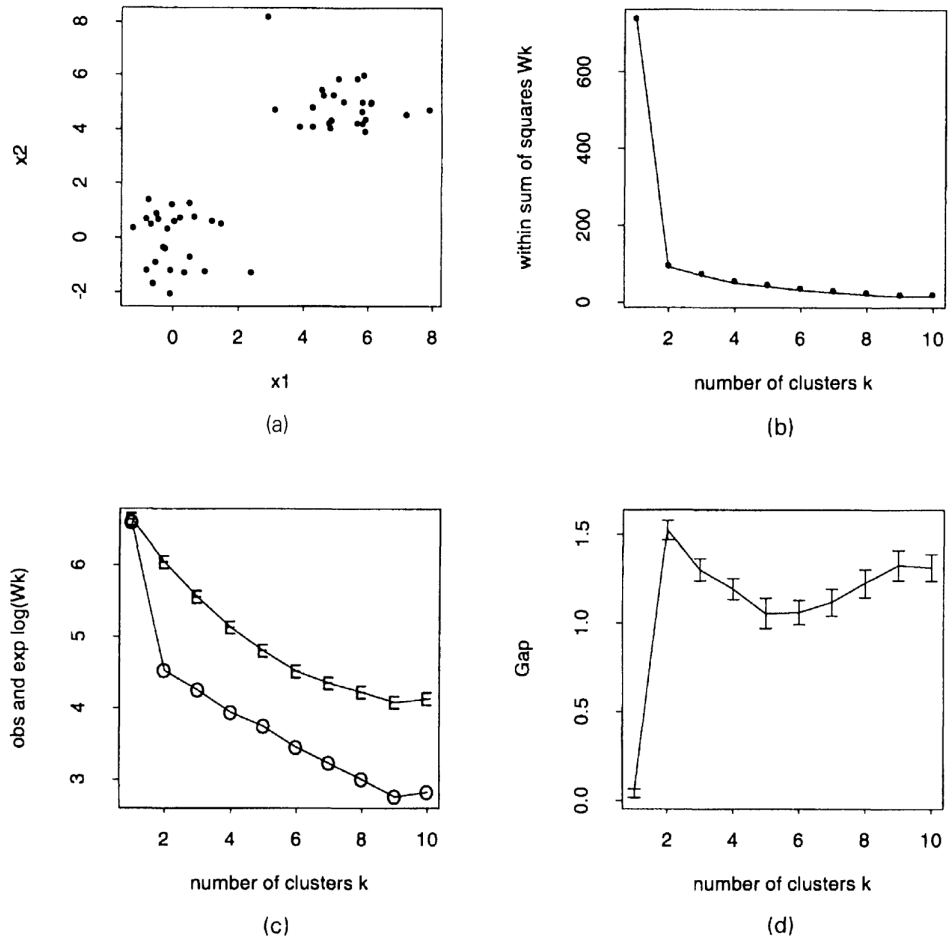


Figure 15: This figure illustrates the data in subplot a., the within-sum squares plot in subplot b., the graphs $\log(\mathbb{E}_n^*(W_k))$ and $\log(W_k)$ in subplot c.; and finally the difference of these graphs are shown in subplot d. This difference shows the Gap statistic for varying k. From ref. [26].

To give us some motivation as to why this 'peak contrast' between the expected value of the reference distribution and the observations, which is shown in figure 15 subplot c., tells us the optimal number of clusters; consider clustering n uniform data points in p dimensions with k centres. If there are actually K clusters we expect W_k to decrease faster than $\mathbb{E}^*(W_k)$ for $k < K$. For $k > K$ it is as if we are adding unnecessary cluster centres in the middle of an approximately uniform cloud and thus W_k should decrease slower than its expected rate. Hence, the gap statistic should be largest when $k = K$ [26].

4.7 Methodology

Due to the great degree of choice that is left upon the researcher to find or even verify clusters one must be careful as how these choices are made. For this purpose, we follow a set of guidelines

one must follow according to Milligan (1996) [27]. This framework illustrates what constitutes a cluster analysis that is grounded in the literature and is shown in table 2.

Steps: Milligan (1996) [27]	Remarks:
1. Objects to cluser	The randomly sampled observations should be members of the cluster structure believed to be present [16].
2. Variables to be used	The variables that are chosen must all justifiably define the clusters [16].
3. Missing Values	
4. Variable standardization	
5. Proximity measure	It is important to choose an interpretation that is able to discriminate between objects the best [16].
6. Clustering method	Clustering methods should be chosen such that it is robust and effecting at recovering the clusters that are suspected in the data.
7. Number of clusters	
8. Replication and testing	
9. Interpretation	

Table 2: Important remarks from a combination of Milligan's original suggestions and the book Cluster Analysis 5th edition [27][16].

5 Methods

We start by modeling the possible structure involved behind cloud fraction data as realizations arising from a unimodal probability distribution in FPCA space. By doing this, we are then able to explore the hypothesis of there being any reliable evidence that these labels correlate with any possible clusters. By comparing the classified observations with any obtained clusters, we are able to arrive at some statement about the possible presence of an underlying complex system associated with the labeled observations.

5.1 Model Selection

When trying to identify any structure with the occurrence of these patterns, attention must be paid to the choice of clustering model. By assuming that these patterns are realizations of some complex system, we have reasonable justification to model these patterns as realizations coming from some unimodal probability distribution in FPCA space. This has been expanded on in chapter 2.1.
The chosen clustering algorithms that make room for this 'unimodal density assumption' are the Gaussian Mixed Models, Mean Shift and the K-means algorithms.

5.2 Data Extraction

We use the exact same data that Schulz (2021) has used for classification. We will, instead, use this data for the sake of clustering analysis. The .nc files that are associated with these 6-hour time windows were obtained from BCO's RAMADDA Data Repository [28].
For further analysis, we extract the Schulz (2021) classification means using a WebPlotDigitizer [29]. After having extracted sufficient points from the graphs shown in figure 7, we apply a linear interpolation scheme to estimate the cloud fraction values at exactly the height values that the radar measures the height values at.

5.3 Cluster Analysis

On top of finding and evaluating these clusters we also have show in table 3 which gives a short summary regarding how Milligan's methodology applies to our thesis. This goes into constituting what is as defined meaningful cluster analysis that is grounded in literature.

Steps: Milligan (1996) [27]	Remarks with respect to our research:
1. Objects to cluster	We cluster cloud fraction data. Specifically, we make observations between an interval of 0 to 4000 meters. It must be noted that this motivates sensitivity analysis for discovering clusters for varying intervals to investigate possible robustness of clusters as a choice of interval. See section 1.3.
2. Variables to be used	We use FPCA scores to be able to model our observations using the least amount of features. See section 3.
3. Missing Values	We neglect them. See section 2.
4. Variable standardization	We don't standardize FPCA scores.
5. Proximity measure	The Euclidean norm is a reasonable measure of distance in this situation and there is no obvious motivation for choosing another. For simplicity, we use the Euclidean proximity measure.
6. Clustering method	We use K-means, Mean Shift and GMM. Their resulting clusters will be compared and interpreted to check robustness for any suspected clusters. See section 4
7. Number of clusters	See section 4.6.
8. Replication and testing	We don't have direct access to the labels associated with each specific observation. Otherwise, it would have been interesting to use the Rand measure to be able to quantify correspondence between the labels assigned through clustering and classification.
9. Interpretation	Finally we interpret the results. This is done through internal validation , see section 4.4, and through external validation by seeing as to how far the existing literature is able to match our results. We mainly use the means associated with the classified cloud fraction patterns from Schulz (2021). We compare any suspected cluster means with the classified means extracted from Schulz (2021).

Table 3: This table gives a short summary as to how Milligan's (1996) methodology applied to our cluster analysis.

Part III
Results and Discussion

In this chapter we present our results that we thoroughly discuss in the following subsections. We start by investigating clustering tendency and subsequently move onto data pre-processing before we do our cluster analysis.

6 Testing for Clustering Tendency

Testing for clustering tendency, using 1% of the complete sample size as randomly generated samples, the Hopkins statistic results in a p-value of $p = 0.00 < 0.05$. Thus we may reject the null hypothesis and assume that the data is clusterable under a confidence interval of 95%.

7 Data Pre-processing: FPCA

The first two FPCA components estimated using Schulz's (2021) data are presented in figure 16.

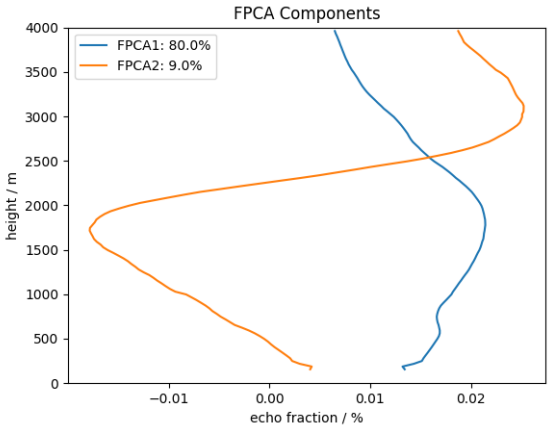


Figure 16: Here we illustrate a height versus cloud fraction plot where the first two estimated principal components corresponding to Schulz's (2021) data are displayed. The explained variance corresponding to each functional principal component are shown in the legend.

These two principal components are able to explain 89% of all variation in the data with respect to the mean observation. It therefore suffices to use these two components if we are willing to sacrifice roughly 10% of explained variance. Any additional component will yield diminishing returns, come at a cost of visualization and allows the curse of dimensionality to take more of an affect.
Our first principal component is able to explain 80.0% of the variation in our data. Any observation from our data can be described using the mean observation plus a sum of functional principal components as shown in equation 3.1. If we approximate our observations using only the first principal component; it entails that our observations are modelled using scalar multiples of the first mode of variation ϕ_1 plus the mean observation μ. Infer how this is expressed in equation 22.

$$X(h) = \mu(h) + c_1\phi_1(h) \tag{22}$$

The first mode of variation's shape can be described as hump-like as seen in figure 16. This mode is shown to be greatest around 1800 km. Using equation 22, the absolute difference ΔX_{ij} between two observations with FPCA scores c_1^i and c_1^j can be written as:

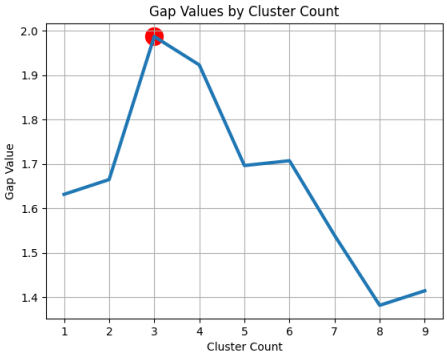
$$\Delta X_{ij} = |c_1^i - c_1^j|\phi_1. \tag{23}$$

Observe, for fixed FPCA scores in equation 23, how the hump-like structure in from figure 16 transfers to ΔX_{ij}. Therefore, we can argue that our data has the greatest variability around 1800 km. This makes sense when we look at the classification means in figure 7. According to Schulz (2021); the largest inter-pattern variability is indeed maximum between 1.5 km and 2.5 km [4]. We are therefore able to corroborate the descriptive capability our applied FPCA. Any further functional principal component becomes too difficult to interpret due to the diminishing negligible contributions towards the total explained variance.

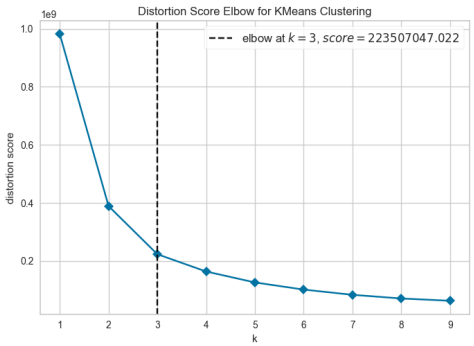
8 K-means

8.1 Determining the Number of Clusters

According to the Elbow method we infer that the optimal number of clusters to be 3 as seen in figure 17b. The Gap statistic also tells us that the optimal number is 3 according to figure 17a. Because we are interested in finding at most four possible patterns through clusters, we have motivation to conduct our cluster analysis for 2,3 and 4 clusters.



(a) This plot presents a blue graph of the Gap statistic against cluster number. The red dot indicates what is considered to be the optimal number of clusters using K-means.



(b) This plot shows a blue graph of the total sum-squares against the cluster number. This 'total sum-squares' is denoted as the distortion score in this figure. The optimal number of clusters is indicated by the dashed line.

Figure 17: In this figure we illustrated the plots involved in estimating the optimal number of clusters using the Gap statistic and the Elbow method.

Lastly, we start by making any inferences about the mean Silhouette scores as shown in the figures in Appendix B. For 2, 3 and 4 clusters we have mean Silhouette scores of 0.75, 0.65 and 0.59 respectively. These are shown in figures B.20, B.21 and B.22. This gives us an impression that for 2, 3 and 4 clusters we are, on average, able to find distinguishable clusters. However, we do see that this decline in mean Silhouette scores tell us that the distinguishability between clusters start to suffer as we increase the number of optimal partitions that we allow K-means to find.

8.2 Internal & External Validation

In this subsection we investigate the quality of our obtained clusters from the shown in Appendix B. We now check the Silhouette plots for 2, 3 and 4 clusters using K-means. Look at the clusters labeled 0, 2 and 0 respectively in the Silhouette plots of figures B.20, B.21 and B.22. Observe how we are able to see that our analysis is able to constantly pick out one 'thick' band of Silhouette scores that point towards the good quality of the cluster in question. These 'thick' bands provide

evidence that the related clusters are of good quality in the sense that a great chunk of the observations have Silhouette value, say, greater than 0.5. In contrast to the remaining sliver of observations that represent Silhouette scores smaller than 0.5 thus drawing out a 'thick' band-like structure in our plots. Any cluster we consider of having a reasonably high distinguishability, we will further deem as 'quality clusters'. Apparently, we find exactly one quality cluster for each number of partitions that we ask K-means to find.
Furthermore, by looking at the remaining estimated clusters: They tend to, for all number of estimated partitions, represent less than 32% of all the data. These clusters, in contrast to the quality clusters, tends to not be 'band-like' and tend to have a greater percentage of observations that do not obviously belong to its assigned cluster. We must therefore interpret the remaining estimated clusters with a grain of salt.
Placing our focus upon the quality clusters for each number of partitions as shown in figures B.20, B.21 and B.22; we will now elaborate how they match up with existing literature. In figures B.21 and B.22 we see that these quality cluster means are relatively close to the Sugar classification means. In the plots pertaining to height against cloud fraction data, we are able to see the quality cluster means reflecting a great deal of similarity when compared with the Sugar classification mean. This great degree of similarity is no coincidence when we look at their associated FPCA plots. The quality cluster mean can visually be seen to be much closer to the Sugar classification mean as opposed to the other classification means, i.e. Gravel, Flowers and Fish.
For K-means with two partitions, we see that it has not been able to capture the suspected structure; although it has been capable of doing so for three and four partitions. This might be because the number of partitions are too low. We have been able to find 3 as the optimal number of cluster and any lower number of partitions might merge our suspected cluster with peripherally located data. This leads the membership towards the suspected cluster to become over represented.

9 Gaussian Mixed Models

9.1 Determining the Number of Clusters

The Gap statistic according to figure 18 tells us that the optimal amount of clusters should be 3. The Elbow method can only be seen as a correlatory value for the optimal number of clusters using GMM. This is because, the Elbow method uses partitions coming from the K-means algorithm and not the GMM algorithm. The Elbow method tells us that the optimal number of partition is also 3; so we therefore have motivation to investigate this algorithm's outputs for 2,3 and 4 clusters.

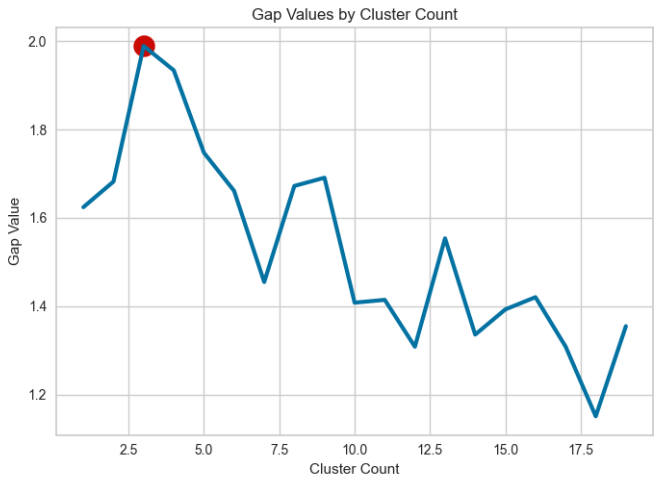


Figure 18: This plot presents a blue graph of the Gap statistic against cluster number. The red dot indicates what is considered to be the optimal number of clusters using GMM.

Lastly, we start by making any inferences about the mean Silhouette scores as shown in the figures in Appendix C. For 2, 3 and 4 components we have mean Silhouette scores of 0.45, 0.38 and 0.34 respectively. These are shown in figures C.23, C.24 and C.25. This gives us an impression that for 2, 3 and 4 clusters we are, on average, able to find somewhat distinguishable clusters as they are not negative but are at the same time less than 0.5. The distinguishability, judging by the mean Silhouette scores, tend to worsen for higher components.

9.2 Internal & External Validation

In this subsection we investigate the quality of our obtained clusters from the shown in Appendix C. We now check the Silhouette plots for 2, 3 and 4 clusters that has been estimated using GMM. In the Silhouette plots of figures C.23, C.24 and C.25 it has been deduced that the quality clusters are that of the labels 0, 2 and 0 respectively. However, it can be argued that these quality clusters have no meaningful interpretation. All the remaining clusters in the Silhouette plots show are of arguably bad quality. These clusters contain a significant percentage of observations with negative Silhouette scores. This is also reflected in the low Silhouette means.
Because the quality of clusters can come at a cost of the quality of others, we will interpret these quality clusters with a grain of salt. Despite this, we are able to see that the quality cluster corresponding with two components in figure C.23 has strong similarities with the Sugar classification mean.

10 Mean Shift

10.1 Bandwidth

We have been able to find a bandwidth $\lambda \approx 585$.

10.2 Internal & External Validation

The plot relating with the Mean Shift analysis is shown in figure D.26 of Appendix D. Starting with the mean Silhouette score, we arrive at a value of 0.63. The Silhouette plot in figure D.26 shows one quality cluster alongside five relatively minuscule clusters. These isolated modes can be interpreted to be artifacts from the Mean Shift algorithm so we do no tie any meaningful interpretation to these.
An interesting observation is that the quality cluster's mean shares a strong similarity with the Sugar classification mean according to figure D.26.

11 Investigating Robust Clusters

For the sake of brevity and clarity, let us denote the quality clusters that are interestingly close to the Sugar classification mean to be denoted as the 'robust clusters'. We discovered robust clusters when we use K-means for 3 and 4 partitions and Mean Shift with bandwidth $\lambda \approx 585$. We denote the cluster obtained by GMM with 2 components as the 'semi-robust cluster' as shown in figure C.23. We take the semi-robust cluster with a grain of salt considering we do not have strong overall clustering for two components.
We essentially have arrived at robust clusters that visually occupy different, yet greatly overlapping regions in FPCA space. When we apply a reverse transformation onto these robust clusters using the Scikit-FDA python package, we arrive at a collection of curves. What we are able to infer, is that the mean and spread of these collections are insensitive to the varying diameter of the sets that represent these robust clusters that are used to subsequently estimate these collections. An important question one must ask, despite the varying diameter in the sizes of these sets, are: Do our observation corroborate with our assumption that we are measuring the realizations of some Sugar unimodal density?
The answer is yes. These robust clusters indeed corroborate with the assumption that our robust cluster follows some and the same unimodal probability density in FPCA space. This manifests

itself in the aforementioned insensitivity. This insensitivity implies that all of these different sets do contain the 'core' observations representative to Sugar. To explain this we have a look at the following figure:

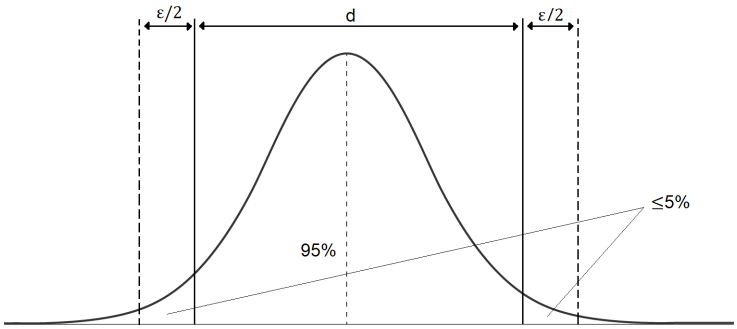


Figure 19: In this figure we see some unimodal distribution with two regions that are defined by the sets whose boundaries represent the dashed and solid lines. The set of observations with diameter d contains 95% of all observations including the mode. Observe that the set with diameter $d + \epsilon$ is at most able to include 5% of all possible observations.

Suppose a region is carved out containing, say, 95%, of all observations pertaining to some unimodal density as shown in figure 19. Any positive perturbation in the diameter of this region will at most contribute to 5% extra observations. According to our results, the smallest set containing information regarding the Sugar-like cluster is the semi-robust cluster. Despite this set being small in diameter, it is able to capture 61% of all observations. This is strongly indicating of the presence of a mode and 'core-like' observations tied to some unimodal density.
The robust cluster of smallest diameter is the robust cluster seen in using K-means with four partitions as seen in figure B.22. The robust cluster representing the largest diameter in our results is attained through Mean Shift as shown in figure D.26. If we make the assumption that the core of our observations are all contained within our robust clusters; we can amount the difference in diameter to an artifact of the clustering algorithm. This assumption is justified from the insensitivity and how at least 61% of the data present can describe Sugar-like curves.
Finally, we briefly elaborate on how we are able to explain our results through any artifacts that the algorithms we have used, to estimate the semi-robust cluster and robust clusters, are prone to generate.
For the semi-robust cluster, found using GMM, we infer that it has not been able to properly find clusters possibly due to the algorithm's assumption that these clusters must be Gaussian distributed. This might have been too harsh of a restriction considering that the robust clusters attained by K-means and Mean shift have been able to carve out clusters that do not seem Gaussian. As for our robust clusters, we naturally expect Mean Shift to make a much larger estimate than our K-means. K-means are bound by the number of predetermined clusters and whose data are hard partitioned into regions where the variance is minimum. Mean Shift, on the other hand, tries to find blobs of smoothly varying density that is highest around their modes. This explains the much larger region that Mean Shift is able to carve out opposed to K-means. Now we have reasonable corroboration that there might be some probabilistic structure behind a subset of our observations. This motivates further research regarding the true unimodal probability density that is associated to Sugar realizations in FPCA space. Leading us to a better understanding of the random function generating Sugar realizations. Doing this will help us be able to give a much more defined and accurate description regarding the possible cloud fraction curves that can denote a Sugar realization.

11.1 Overall Remarks

A peculiarity in our results are that we see our robust clusters dominate in terms of the percentage of observations that it contains. Although it is in strong contradiction of the proportion that Schulz (2021) has found, as seen in table 8, we have to keep in mind that Schulz (2021) underemphasizes

Sugar [4]. Lastly, we observe that we continually obtain clusters that are significantly high in terms of cloud fraction. It might be useful to remove outliers that hinder our clustering algorithms to find meaningful patterns.

Part IV
Conclusion

We have been able to attain robust Sugar-like clusters for K-means for 3 and 4 partitions and Mean Shift with bandwidth $\lambda \approx 585$. This hints at the existence of some unimodal probability distribution whose realizations are related with the Sugar pattern in FPCA space. However, the same can not be said for Gravel, Fish and Flowers as we have not been able to identify them in our analysis. For future research it might be interesting to investigate the true random function that is able to model Sugar realizations. Furthermore, the choice of using cloud fraction curves that represent a height interval up until 4 km is relatively arbitrary. It might be interesting to pursue some type of sensitivity analysis to check whether our results corroborate for varying height interval. A major improvement that can be made on this thesis is to use individual Sugar-labeled observations from Schulz (2021) to evaluate whether clusters are 'robust clusters'. This will yield clearer results as opposed to solely using the mean and spread of the observations labeled as Sugar. Lastly, it might be useful to remove outliers that hinder our clustering algorithms to find meaningful patterns.

References

[1] Marcus Kock. Google Maps. https://www.google.com/maps/@12.4423006,-69.8753425,3a,75y,20h,110t/data=!3m8!1e1!3m6!1sAF1QipOlALWyDsXZPsUcEAOWtQZ5xLSsaCR-G0BO7vnV!2e10!3e11!6shttps:%2F%2Flh5.googleusercontent.com%2Fp%2FAF1QipOlALWyDsXZPsUcEAOWtQZ5xLSsaCR-G0BO7vnV%3Dw203-h100-k-no-pi-20-ya16.534597-ro-0-fo100!7i5472!8i2736. [Online; accessed 2022-11-23].

[2] Aruba Vacations. http://www.arubaanswers.com/Maps-of-Aruba.html. [Online; accessed 2022-11-23].

[3] A. Pier Siebesma, Sandrine Bony, Christian Jakob, and Bjorn Stevens. *Clouds and Climate: Climate Science's Greatest Challenge*. Cambridge University Press, aug 31 2020. [Online; accessed 2022-11-23].

[4] Hauke Schulz, Ryan Eastman, and Bjorn Stevens. Characterization and evolution of organized shallow convection in the downstream north atlantic trades. *Journal of Geophysical Research: Atmospheres*, 126(17):e2021JD034575, 2021.

[5] Roddy Rhodes Rogers and Man Kong Yau. *A Short Course in Cloud Physics*. jan 1 1989. [Online; accessed 2022-11-01].

[6] Bjorn Stevens, Sandrine Bony, Hélène Brogniez, Laureline Hentgen, Cathy Hohenegger, Christoph Kiemle, Tristan S L'Ecuyer, Ann Kristin Naumann, Hauke Schulz, Pier A Siebesma, et al. Sugar, gravel, fish and flowers: Mesoscale cloud patterns in the trade winds. *Quarterly Journal of the Royal Meteorological Society*, 146(726):141–152, 2020.

[7] Coral Ka-Band Cloud Radar [MPI Wiki]. https://wiki.mpimet.mpg.de/doku.php id=observations:bco:cloudradars:coralradar. [Online; accessed 2022-11-03].

[8] G. Nicolis and C. Rouvas-Nicolis. Complex systems. *Scholarpedia*, 2(11):1473, 2007. revision #91143.

[9] World Bank Climate Change Knowledge Portal. https://climateknowledgeportal.worldbank.org/country/barbados/climate-data-historical. [Online; accessed 2022-11-06].

[10] Julien Jacques and Cristian Preda. Functional data clustering: a survey. *Advances in Data Analysis and Classification*, 8(3):231–255, 2014.

[11] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *International conference on database theory*, pages 217–235. Springer, 1999.

[12] Han Lin Shang. A survey of functional principal component analysis. *AStA Advances in Statistical Analysis*, 98(2):121–142, 2014.

[13] GAA-UAM. Github - GAA-UAM/scikit-fda: Functional Data Analysis Python package. https://github.com/GAA-UAM/scikit-fda. [Online; accessed 2022-10-21].

[14] JO Ramsay and BW Silverman. Principal components analysis for functional data. *Functional data analysis*, pages 147–172, 2005.

[15] Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193, 2015.

[16] Brian S. Everitt, Sabine Landau, Morven Leese, and Daniel Stahl. *Cluster Analysis*. John Wiley Sons, jan 14 2011.

[17] Comparing different clustering algorithms on toy datasets. https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html. [Online; accessed 2022-10-21].

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] Charles Bouveyron, Gilles Celeux, T. Brendan Murphy, and Adrian E. Raftery. *Model-Based Clustering and Classification for Data Science: With Applications in R*. Cambridge University Press, jul 25 2019.

[20] Miguel A Carreira-Perpinán. A review of mean-shift algorithms for clustering. *arXiv preprint arXiv:1503.00687*, 2015.

[21] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5):603–619, 2002.

[22] Selecting the number of clusters with silhouette analysis on KMeans clustering. https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html. [Online; accessed 2022-11-03].

[23] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[24] Amit Banerjee and Rajesh N Dave. Validating clusters using the hopkins statistic. In *2004 IEEE International conference on fuzzy systems (IEEE Cat. No. 04CH37542)*, volume 1, pages 149–153. IEEE, 2004.

[25] arvkevi. Github - arvkevi/kneed: Knee point detection in Python. https://github.com/arvkevi/kneed. [Online; accessed 2022-10-21].

[26] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[27] Glenn W Milligan. Clustering validation: results and implications for applied analyses. In *Clustering and classification*, pages 341–375. World Scientific, 1996.

[28] Ramadda Data Repository. https://observations.mpimet.mpg.de/repository/. [Online; accessed 2022-11-03].

[29] Webplotdigitizer. https://automeris.io/WebPlotDigitizer/. [Online; accessed 2022-11-03].

[30] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[31] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[32] Kyu-Won Kim and Gyu-In Jee. Free-resolution probability distributions map-based precise vehicle localization in urban areas. *Sensors*, 20(4):1220, 2020.

[33] Miguel A Carreira-Perpinán. A review of mean-shift algorithms for clustering. *arXiv preprint arXiv:1503.00687*, 2015.

A Algorithms

Algorithm 1 K-means++: Adapted from [30]

Input: Data vectors $(\boldsymbol{x}_n)_{n=1}^N$, number of clusters K
$n \leftarrow \text{RandomInteger}(1, N)$
$\boldsymbol{\mu}_1 \leftarrow \boldsymbol{x}_n$
for $k \leftarrow 2 \dots K$ **do**
    for $n \leftarrow 1 \dots N$ **do**
        $d_n \leftarrow \min_{k'<k} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||_2$
    for $n \leftarrow 1 \dots N$ **do**
        $p_n \leftarrow d_n^2 / \sum_{n'} d_{n'}^2$
    $n \leftarrow \text{Discrete}(p_1, p_2, \dots, p_N)$
    $\boldsymbol{\mu}_k \leftarrow \boldsymbol{x}_n$
Return cluster means $(\boldsymbol{\mu}_k)_{k=1}^K$

Algorithm 2 Lloyd's algorithm: Adapted from [31]

Input: Data vectors $(\boldsymbol{x}_n)_{n=1}^N$, number of clusters K
for $n \leftarrow 1 \dots N$ **do**
    $\boldsymbol{r} \leftarrow [0, 0, \dots, 0]$
    $k' \leftarrow \text{RandomInteger}(1, K)$
    $r_{nk'} = 1$
repeat
    for $k \leftarrow 1 \dots K$ **do**
        $N_k \leftarrow \sum_{n=1}^N r_{nk}$
        $\mu_k \leftarrow \frac{1}{N_k} r_{nk} \boldsymbol{x}_n$
    for $n \leftarrow 1 \dots K$ **do**
        $\boldsymbol{r}_n \leftarrow [0, 0, \dots, 0]$
        $k' \leftarrow \arg\min_k ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2$
        $r_{nk'} = 1$
until none of the $\boldsymbol{r}_n$ change.
Return assignments $(\boldsymbol{r}_n)_{n=1}^N$ for each datum, and cluster means $(\boldsymbol{\mu}_n)_{n=1}^K$

Algorithm 3 Expectation-Maximization algorithm: Adapted from [32]

Input: Data vectors $(\boldsymbol{x}_n)_{n=1}^N$, number of clusters K
Parameter Initialization π, μ, Σ
for $t \leftarrow 1 \dots T$ **do**
    for $n \leftarrow 1 \dots N$ **do**
        for $k \leftarrow 1 \dots K$ **do**
            $\gamma(z_{nk}) = \frac{\pi_k N(p_n|\mu_k, \Sigma_k)}{\sum_i^K \pi_i N(p_n|\mu_i, \Sigma_i)}$
        for $k \leftarrow 1 \dots K$ **do**
            $\mu_k = \frac{\sum_{n=1}^N \gamma(z_{nk}) p_n}{\sum_{n=1}^N \gamma(z_{nk})}$
            $\Sigma_k = \frac{\sum_{n=1}^N \gamma(z_{nk})(p_n - \mu_k)(p_n - \mu_k)^T}{\sum_{n=1}^N \gamma(z_{nk})}$
            $\pi_k = \frac{1}{N} \sum_{n=1}^N \gamma(z_{nk})$
Return π, μ, σ

Algorithm 4 Mean Shift algorithm: Adapted from [33]

Input: Data vectors $(\boldsymbol{x}_n)_{n=1}^N$
for $n \leftarrow 1 \dots N$ **do**
    $\boldsymbol{x} \leftarrow \boldsymbol{x}_n$
    repeat
        $\forall n : p(n|x) \leftarrow \frac{K(||\boldsymbol{x}-\boldsymbol{x}_n||^2)}{\sum_{n=1}^N K(||\boldsymbol{x}-\boldsymbol{x}_{n'}||^2)}$
        $\boldsymbol{x} \leftarrow \sum_{n=1}^N p(n|\boldsymbol{x})\boldsymbol{x}_n$
    until stop
    $\boldsymbol{z}_n \leftarrow \boldsymbol{x}$
connected-component $((\boldsymbol{z}_n)_{n=1}^N, \lambda)$

B Analysis for K-means Clustering with Schulz's (2021) Data

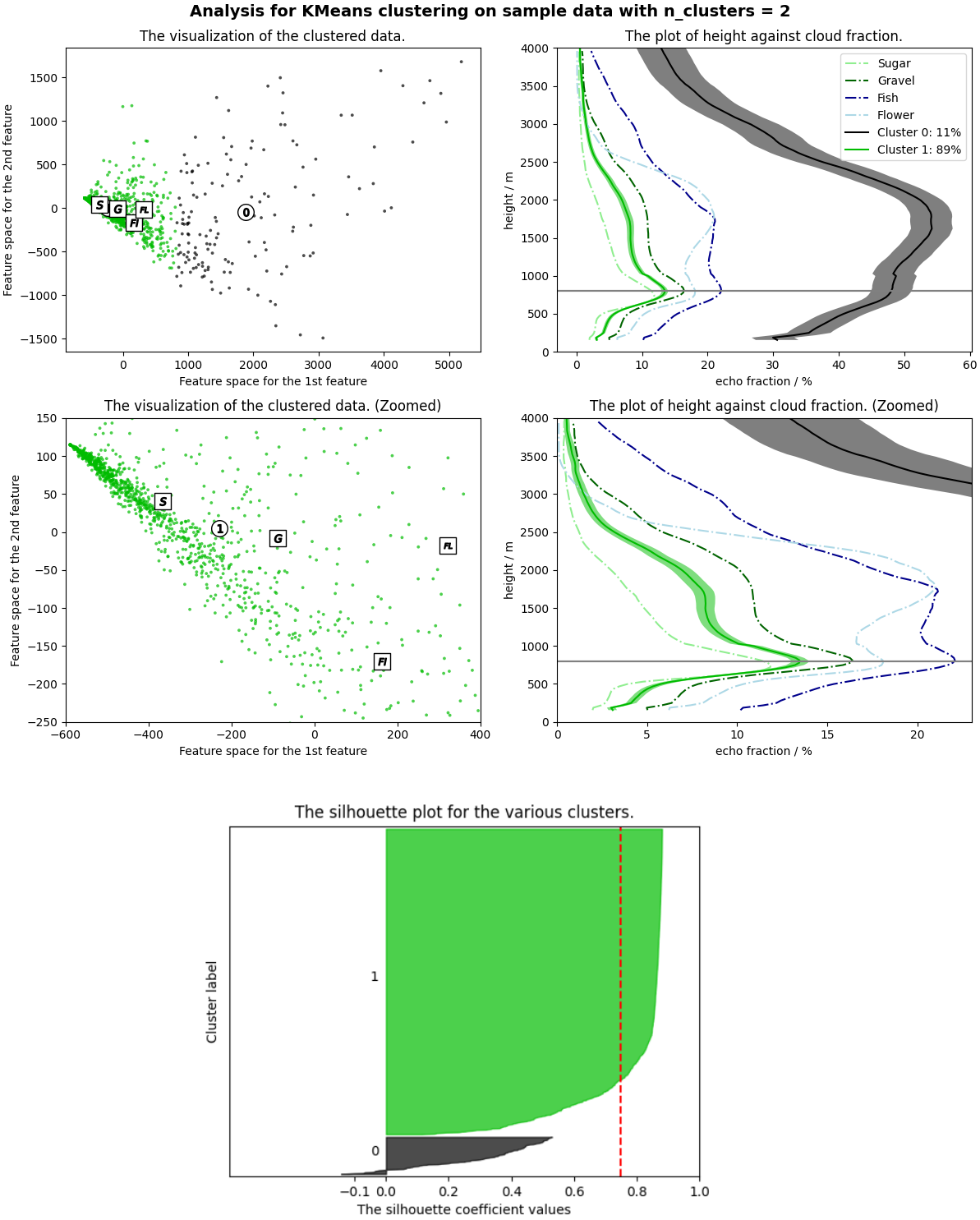


Figure B.20: We show the analysis for K-means with 2 clusters. The clusters are associated with a color and its % of belonging observations are shown in the legend. In the scatter plots, the squared markers represent the classification means and the circled ones represent the cluster means. In the echo fraction plots, the shading represents points up to two standard deviations away from its mean. In the Silhouette plot, the dashed red line represents the mean Silhouette score of 0.75.

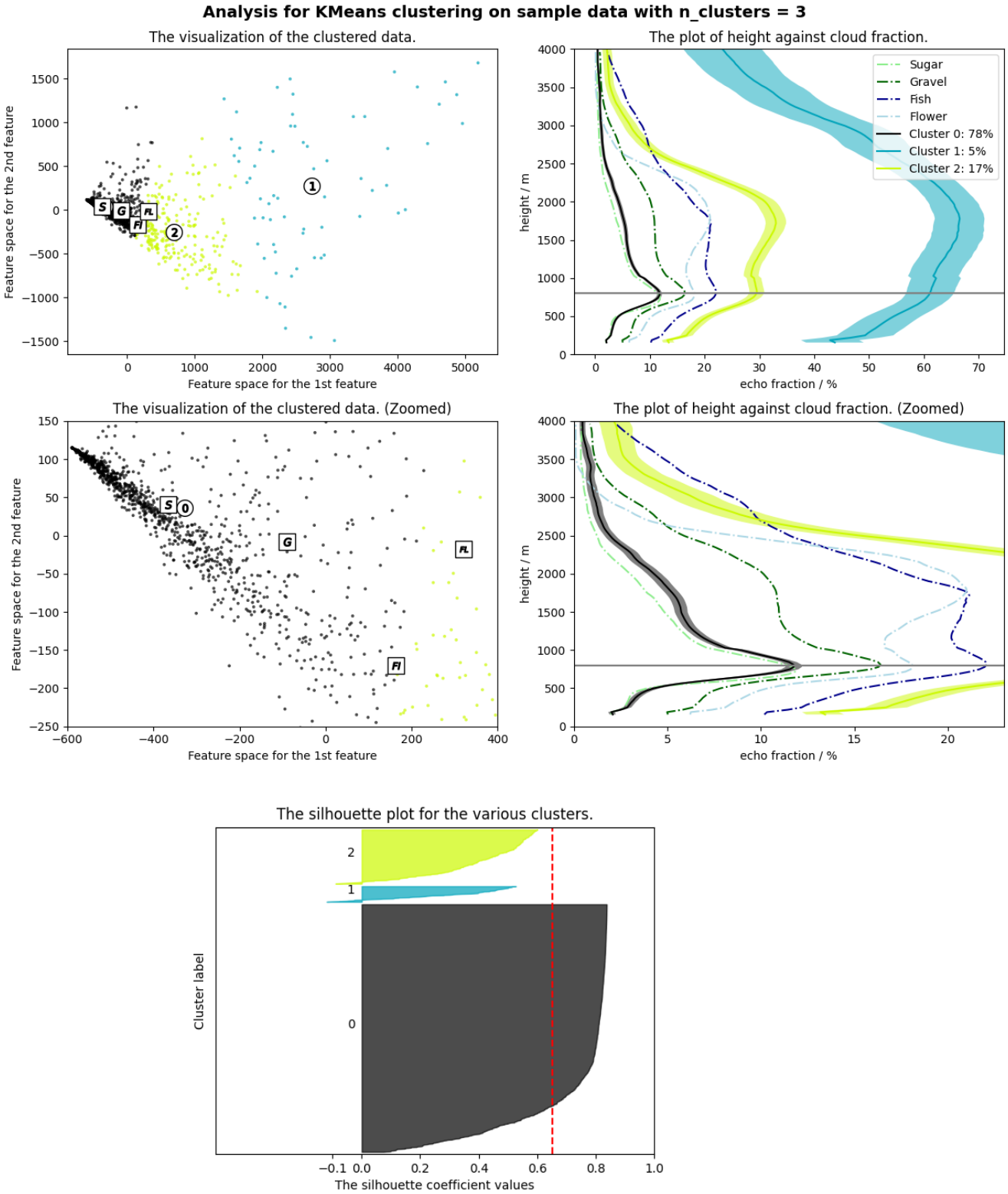


Figure B.21: We show the analysis for K-means with 3 clusters. The clusters are associated with a color and its % of belonging observations are shown in the legend. In the scatter plots, the squared markers represent the classification means and the circled ones represent the cluster means. In the echo fraction plots, the shading represents points up to two standard deviations away from its mean. In the Silhouette plot, the dashed red line represents the mean Silhouette score of 0.65.

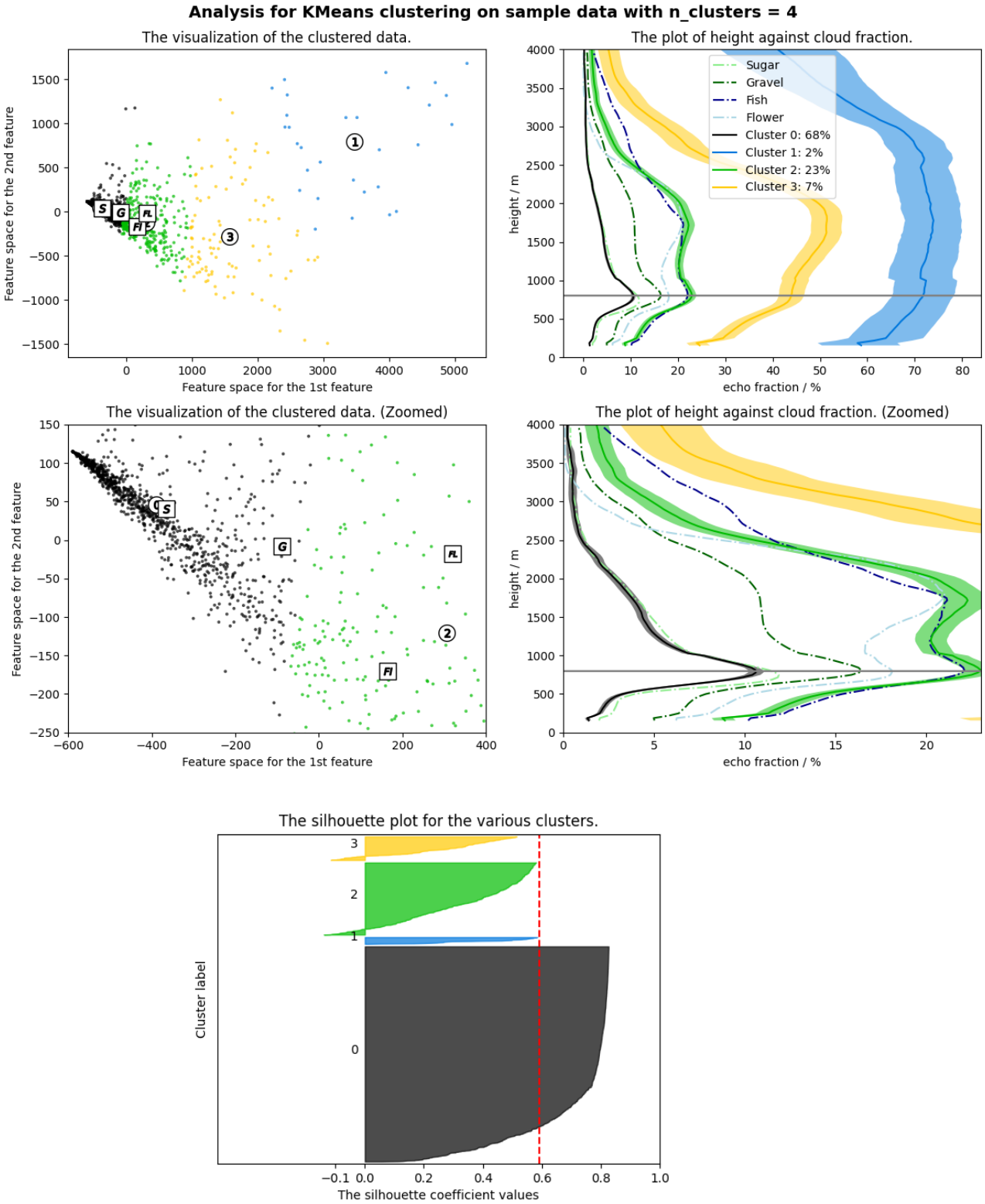


Figure B.22: We show the analysis for K-means with 5 clusters. The clusters are associated with a color and its % of belonging observations are shown in the legend. In the scatter plots, the squared markers represent the classification means and the circled ones represent the cluster means. In the echo fraction plots, the shading represents points up to two standard deviations away from its mean. In the Silhouette plot, the dashed red line represents the mean Silhouette score of 0.59.

C Analysis for GMM Clustering with Schulz's (2021) Data

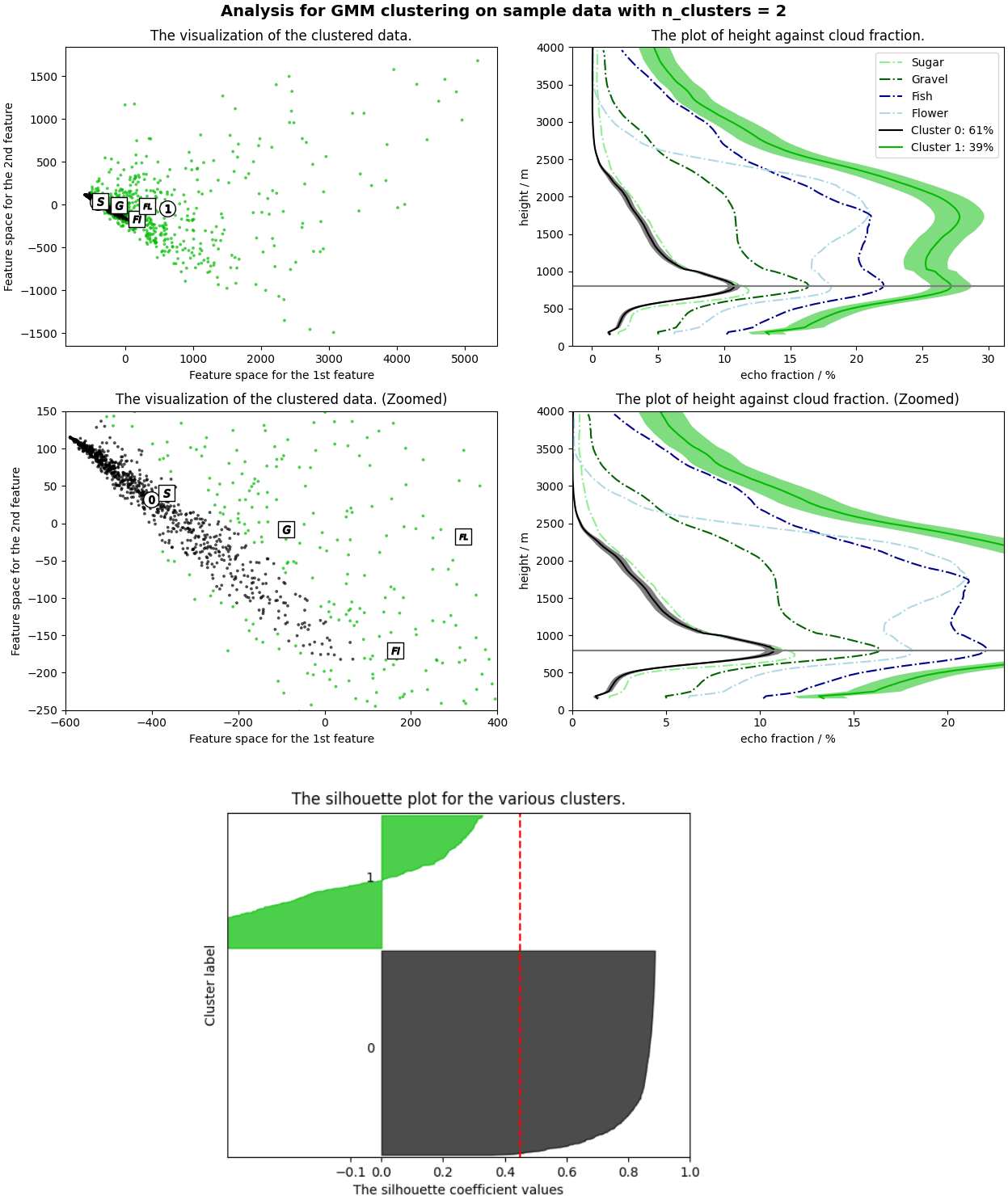


Figure C.23: We show the analysis for GMM with 2 clusters. The clusters are associated with a color and its % of belonging observations are shown in the legend. In the scatter plots, the squared markers represent the classification means and the circled ones represent the cluster means. In the echo fraction plots, the shading represents points up to two standard deviations away from its mean. In the Silhouette plot, the dashed red line represents the mean Silhouette score of 0.45.

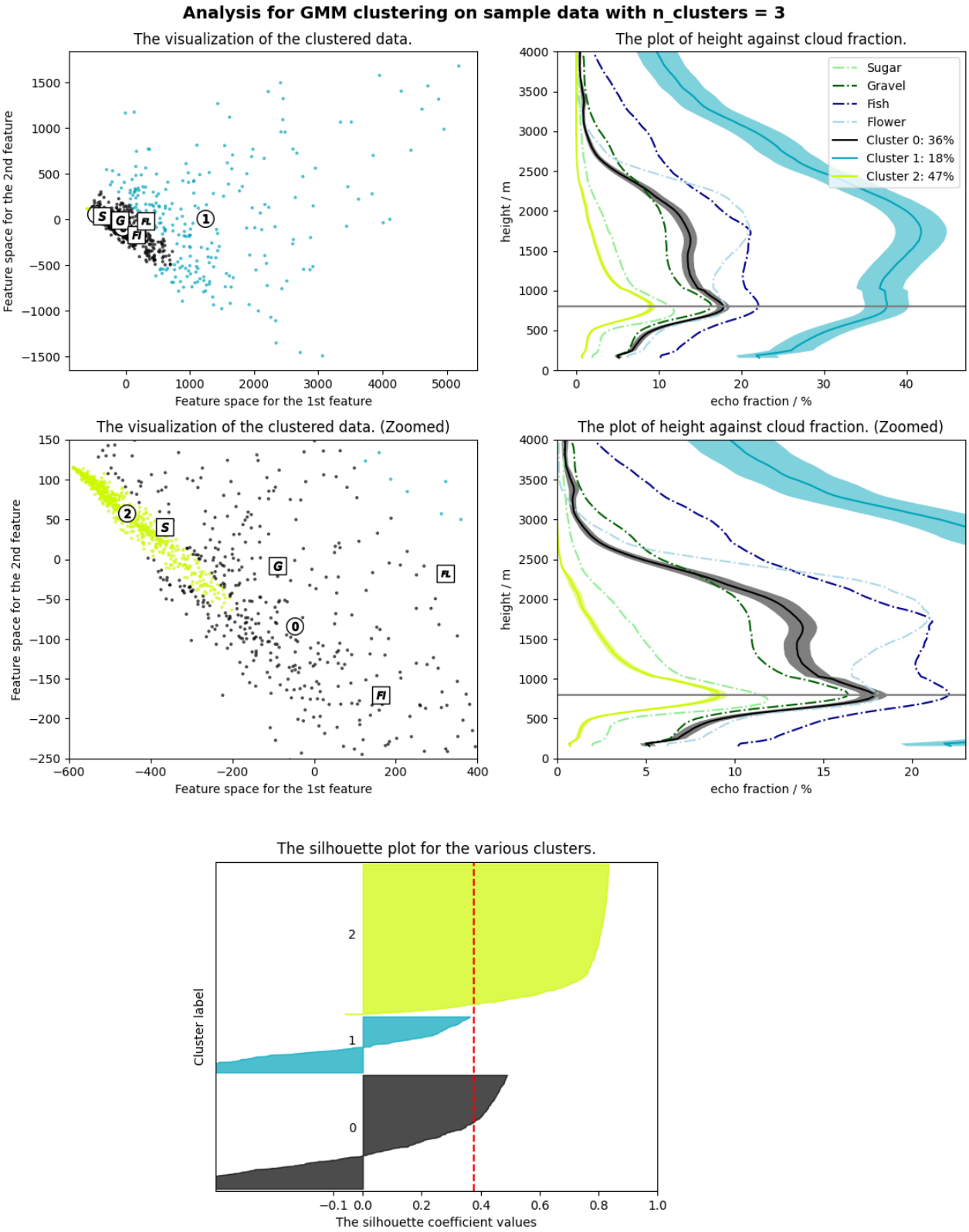


Figure C.24: We show the analysis for GMM with 3 clusters. The clusters are associated with a color and its % of belonging observations are shown in the legend. In the scatter plots, the squared markers represent the classification means and the circled ones represent the cluster means. In echo fraction plot, the shading represents points up to two standard deviations away from its mean. In the Silhouette plot, the dashed red line represents the mean Silhouette score of 0.38.

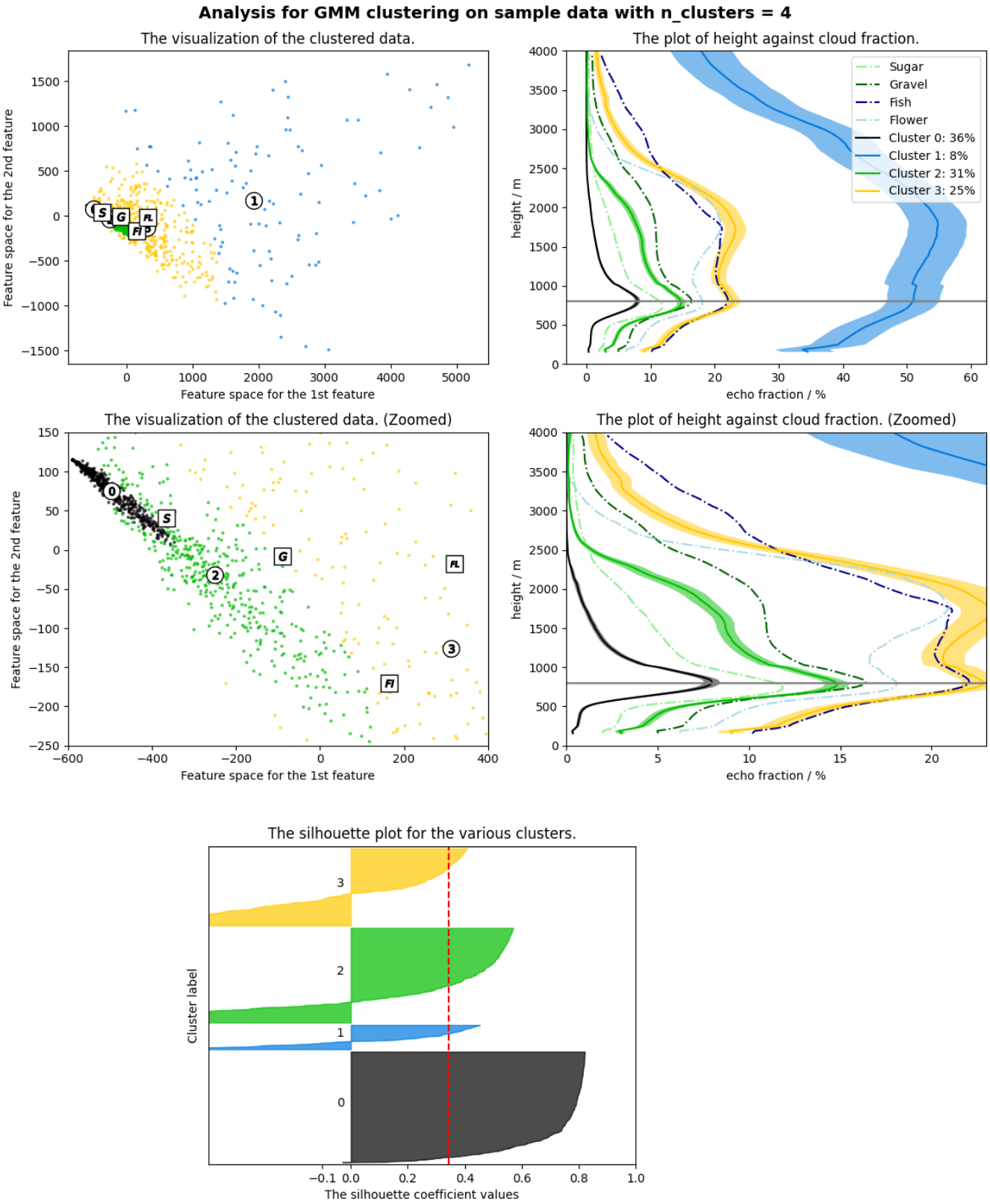


Figure C.25: We show the analysis for GMM with 4 clusters. The clusters are associated with a color and its % of belonging observations are shown in the legend. In the scatter plots, the squared markers represent the classification means and the circled ones represent the cluster means. In echo fraction plot, the shading represents points up to two standard deviations away from its mean. In the Silhouette plot, the dashed red line represents the mean Silhouette score of 0.34.

D Analysis for Mean Shift Clustering with Schulz's (2021) Data

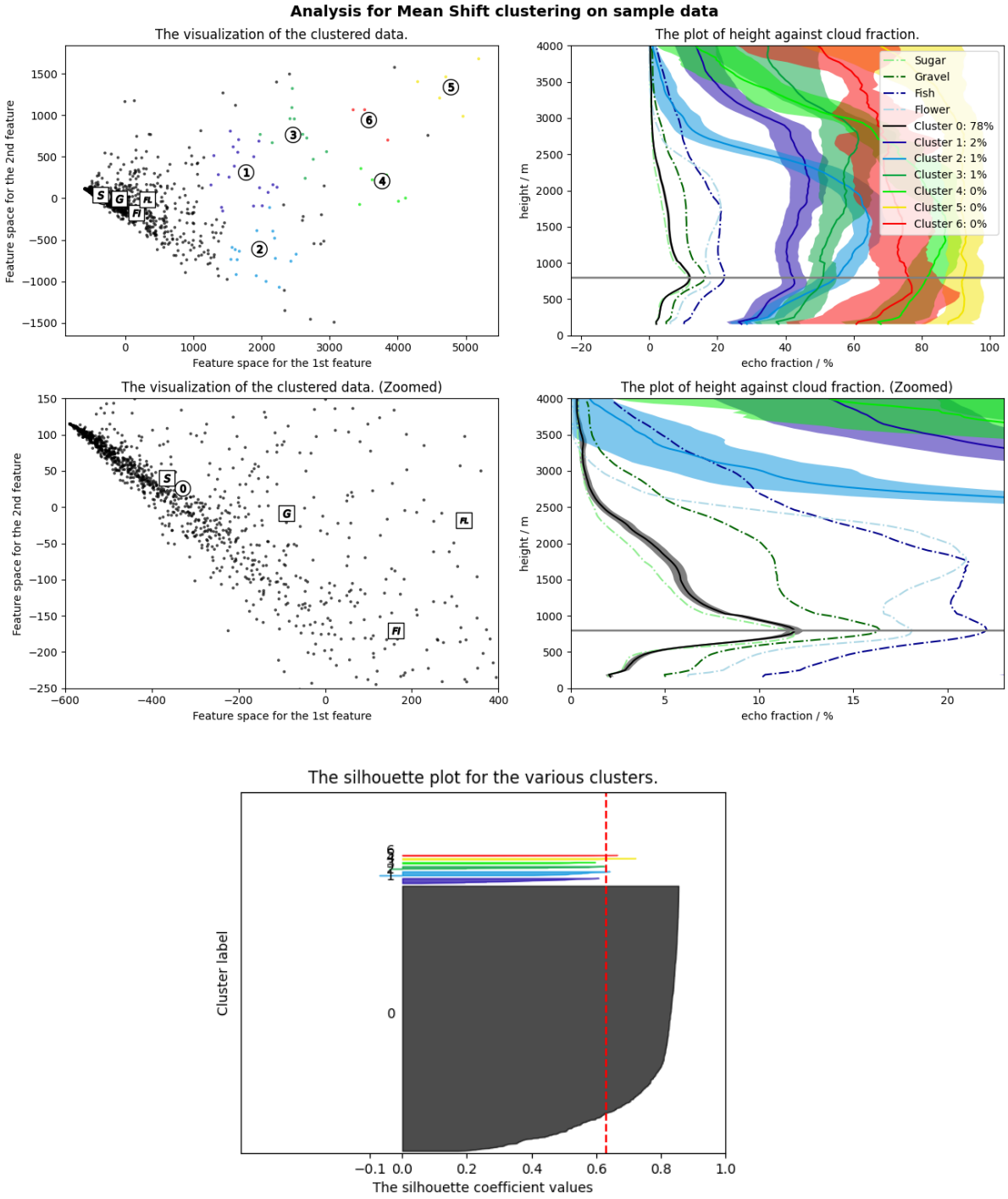


Figure D.26: We show the analysis for Mean Shift clustering. Orphans have been omitted in our analysis. In all subplots, the clusters are associated with a color and its % of belonging observations are shown in the legend. In the scatter plots, the squared markers represent the classification means and the circled ones represent the cluster means. In the echo fraction plots, the shading represents points up to two standard deviations away from its mean. In the Silhouette plot, the dashed red line represents the mean Silhouette score of 0.63.

E Python Programs

E.1 Time Windows

Listing 1: Time Windows.py

```
import numpy as np
import xarray as xr
import datetime as dt
import os
import pandas as pd

def hour_rounder(t):
    # Rounds to nearest hour by adding a timedelta hour if minute >= 30
    return (t.replace(second=0, microsecond=0, minute=0, hour=t.hour)
               +dt.timedelta(hours=t.minute//30))

path = 'C:/Users/Gyan/Desktop/BEP Python/Data'

Data = []
nd_Data = []

for filesA in os.listdir(path):
    for filesB in os.listdir(path + '/' + filesA):

        Radar_File = xr.open_dataset(path + '/' + filesA + '/' + filesB
            )

        Zf = Radar_File.Zf.copy()
        Zf_Cloud = np.where(Zf < -50, np.nan, Zf)

        Time = Radar_File.time.values

        window_hours = 6

        window_width = dt.timedelta(hours=window_hours)

        time_window_starts = pd.date_range(Radar_File.time.values[0],
                                           Radar_File.time.values[-1],
                                               freq='{}H'.format(str(
                                               window_hours)))

        for i in range(len(time_window_starts)):

            Cut_Time = Time[(Time >= hour_rounder(time_window_starts[i
                ])) & (Time <= hour_rounder(time_window_starts[i]) + dt
                .timedelta(hours=window_hours))]
            Cut_Zf_Cloud = Zf_Cloud[(Time >= hour_rounder(
                time_window_starts[i])) & (Time <= hour_rounder(
                time_window_starts[i]) + dt.timedelta(hours=
                window_hours))]


            Range = Radar_File.range.values
            Mask = np.where(np.isnan(Cut_Zf_Cloud)==False,1,np.nan)

            #Calculate cloud fraction
```

```
            CF = np.zeros(shape=(len(Range)))
            for i in range(len(CF)):
                CF[i] = np.nansum(Mask[:,i])

            if len(Cut_Time) > 0:
                CF = CF / len(Cut_Time) * 100
                Data.append(CF[:605])
                if np.sum(CF**2) != 0:
                    nd_Data.append(CF[:605])
                else:
                    pass
            else:
                pass

CSV_path = 'C:/Users/Gyan/Desktop/BEP Python/CSVData/Data.csv'
df_Data = pd.DataFrame(Data)
df_Data.to_csv(CSV_path,index=False)

CSV_nd_path = 'C:/Users/Gyan/Desktop/BEP Python/CSVData/Data_nd.csv'
df_train_Data = pd.DataFrame(nd_Data)
df_train_Data.to_csv(CSV_nd_path,index=False)
```

E.2 Clustering Tendency

Listing 2: Data Reader Clustering Tendency.py

```
from scipy.stats import beta
import numpy as np
import xarray as xr
import pandas as pd
from skfda.preprocessing.dim_reduction.feature_extraction import FPCA
from skfda.representation.grid import FDataGrid
from sklearn.neighbors import NearestNeighbors
from random import sample
from numpy.random import uniform

def find_nearest(array, value):

    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()

    return idx

def hopkins_statistic(X):

    sample_size = int(X.shape[0]*0.05) #0.05 (5%) based on paper by
        Lawson and Jures


    #a uniform random sample in the original data space
    X_uniform_random_sample = uniform(X.min(axis=0), X.max(axis=0) ,(
        sample_size , X.shape[1]))



    #a random sample of size sample_size from the original data X
```

```
29      random_indices=sample(range(0, X.shape[0], 1), sample_size)
30      X_sample = X[random_indices]
31
32
33      #initialise unsupervised learner for implementing neighbor searches
34      neigh = NearestNeighbors(n_neighbors=2)
35      nbrs=neigh.fit(X)
36
37      #u_distances = nearest neighbour distances from uniform random
            sample
38      u_distances , u_indices = nbrs.kneighbors(X_uniform_random_sample ,
             n_neighbors=2)
39      u_distances = u_distances[: , 0] #distance to the first (nearest)
            neighbour
40
41      #w_distances = nearest neighbour distances from a sample of points
            from original data X
42      w_distances , w_indices = nbrs.kneighbors(X_sample , n_neighbors=2)
43      #distance to the second nearest neighbour (as the first neighbour
            will be the point itself, with distance = 0)
44      w_distances = w_distances[: , 1]
45
46      u_sum = np.sum(u_distances)
47      w_sum = np.sum(w_distances)
48
49      #compute and return hopkins' statistic
50      H = u_sum/ (u_sum + w_sum)
51      return H
52
53  root_path = 'C:/Users/Gyan/Desktop/BEP Python'
54
55  radar_file = xr.open_dataset(root_path + '/Data/20201' + '/
        MMCR__MBR2__Spectral_Moments__10s__155m-18km__200112.nc')
56
57  Range = radar_file.range.values
58  dataframe_nd = pd.read_csv(root_path + '/CSVData/Data_nd.csv')
59  data_nd = dataframe_nd.to_numpy()
60
61  lower = 0
62
63  upper = 4000
64
65  train_range = range(find_nearest(Range, lower),find_nearest(Range,
        upper))
66  f_data = FDataGrid(data_matrix = data_nd[:,train_range], grid_points=
        Range[train_range])
67
68  fpca_basis = FPCA(2).fit(f_data)
69  fpca_data = fpca_basis.transform(f_data)
70
71  statistic = hopkins_statistic(fpca_data)
72
73 m = len(data_nd)
74
75  pvalue = 1 - beta.cdf(statistic ,m,m)
76  print(pvalue)
```

E.3 K-means Clustering

E.3.1 K-means Analysis

Listing 3: Data Reader K-Means Analysis.py

```
from scipy import stats
from scipy.stats import pmean
import numpy as np
import matplotlib.pyplot as plt
import xarray as xr
import pandas as pd
from sklearn.cluster import KMeans
from skfda.preprocessing.dim_reduction.feature_extraction import FPCA
from skfda.representation.grid import FDataGrid
from scipy.interpolate import interp1d
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.cm as cm

def find_nearest(array, value):

    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()

    return idx

def WPD_Reader(path_input, train_range, range_data):

    dataframe_input = pd.read_csv(path_input, header = None)
    data_input = dataframe_input.to_numpy()

    x_values_input = data_input[:,0]
    y_values_input = data_input[:,1]

    interpolated_input = interp1d(x_values_input,y_values_input)

    mean_input = interpolated_input(range_data[train_range])
    fmean_input = FDataGrid(data_matrix = mean_input, grid_points=
        range_data[train_range])

    return mean_input, fmean_input

root_path = 'C:/Users/Gyan/Desktop/BEP Python'
sugar_path = 'C:/Users/Gyan/Desktop/BEP Python/Extracted Data/Sugar.csv
    '
gravel_path = 'C:/Users/Gyan/Desktop/BEP Python/Extracted Data/Gravel.
    csv'
flower_path = 'C:/Users/Gyan/Desktop/BEP Python/Extracted Data/Flowers.
    csv'
fish_path = 'C:/Users/Gyan/Desktop/BEP Python/Extracted Data/Fish.csv'

radar_file = xr.open_dataset(root_path + '/Data/20201' + '/
    MMCR__MBR2__Spectral_Moments__10s__155m-18km__200112.nc')

Range = radar_file.range.values
dataframe_nd = pd.read_csv(root_path + '/CSVData/Data_nd.csv')
data_nd = dataframe_nd.to_numpy()
```

```
47
48 lower = 0
49
50 upper = 4000
51
52 train_range = range(find_nearest(Range, lower),find_nearest(Range,
       upper))
53 h_train = Range[train_range]
54
55 f_data = FDataGrid(data_matrix = data_nd[:,train_range], grid_points=
       h_train)
56
57 fpca_basis = FPCA(2).fit(f_data)
58 fpca_data = fpca_basis.transform(f_data)
59
60 fpca_components = fpca_basis.components_.data_matrix
61
62 plt.plot(fpca_components[0][:], h_train, label = 'FPCA1: {}%'.format(
       fpca_basis.explained_variance_ratio_[0].round(2)*100))
63 plt.plot(fpca_components[1][:], h_train, label = 'FPCA2: {}%'.format(
       fpca_basis.explained_variance_ratio_[1].round(2)*100))
64 plt.title('FPCA Components')
65 plt.ylim(0,4000)
66 plt.xlabel('echo fraction / %')
67 plt.ylabel('height / m')
68 plt.legend()
69 plt.show()
70
71 mean_sugar, fmean_sugar = WPD_Reader(sugar_path, train_range, Range)
72 mean_gravel, fmean_gravel = WPD_Reader(gravel_path, train_range, Range)
73 mean_flower, fmean_flower = WPD_Reader(flower_path, train_range, Range)
74 mean_fish, fmean_fish = WPD_Reader(fish_path, train_range, Range)
75
76 mean_gravel_hat = [fpca_basis.transform(fmean_gravel)[0], 'G']
77 mean_sugar_hat = [fpca_basis.transform(fmean_sugar)[0], 'S']
78 mean_flower_hat = [fpca_basis.transform(fmean_flower)[0], 'FI']
79 mean_fish_hat = [fpca_basis.transform(fmean_fish)[0], 'FL']
80
81 mean_hat_list = [mean_gravel_hat,mean_sugar_hat,mean_flower_hat,
       mean_fish_hat]
82
83 n_clusters = 2
84
85 ##################################################################PLOTS
86 ax2 = plt.subplot2grid((2, 2), (0, 0))
87 ax3 = plt.subplot2grid((2, 2), (0, 1))
88 ax4 = plt.subplot2grid((2, 2), (1, 0))
89 ax5 = plt.subplot2grid((2, 2), (1, 1))
90
91 plt.subplots_adjust(wspace = 0.3, hspace = 0.3)
92
93 # Initialize the clusterer with n_clusters value and a random generator
94 # seed of 10 for reproducibility.
95 clusterer = KMeans(n_clusters=n_clusters)
96 cluster_labels = clusterer.fit_predict(fpca_data)
97
```

```
# The silhouette_score gives the average value for all the samples.
# This gives a perspective into the density and separation of the formed
# clusters
silhouette_avg = silhouette_score(fpca_data, cluster_labels)
print(
    "For n_clusters =",
    n_clusters,
    "The average silhouette_score is :",
    silhouette_avg,
)

# ################################################################################################################
# AX2
# 2nd Plot showing the actual clusters formed
colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
ax2.scatter(
    fpca_data[:, 0], fpca_data[:, 1], marker=".", s=30, lw=0, alpha=0.7, c=colors, edgecolor="k"
)

# Labeling the clusters
centers = clusterer.cluster_centers_

# Draw white circles at cluster centers
ax2.scatter(
    centers[:, 0],
    centers[:, 1],
    marker="o",
    c="white",
    alpha=1,
    s=200,
    edgecolor="k",
)

for i, c in enumerate(centers):
    ax2.scatter(c[0], c[1], marker="$%d$" % i, alpha=1, s=50, edgecolor="k")

# Draw classification mean centers
for i in range(len(mean_hat_list)):
    ax2.scatter(
        mean_hat_list[i][0][0],
        mean_hat_list[i][0][1],
        marker="s",
        c="white",
        alpha=1,
        s=200,
        edgecolor="k",
    )
    ax2.scatter(mean_hat_list[i][0][0], mean_hat_list[i][0][1], marker="${}$".format(mean_hat_list[i][1]), alpha=1, s=50, edgecolor="k", c='white')

ax2.set_title("The visualization of the clustered data.")
```

```
147 ax2.set_xlabel("Feature space for the 1st feature")
148 ax2.set_ylabel("Feature space for the 2nd feature")
149
150 # ################################################################################
    AX4
151 # 2nd Plot showing the actual clusters formed
152 colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
153 ax4.scatter(
154     fpca_data[:, 0], fpca_data[:, 1], marker=".", s=30, lw=0, alpha
        =0.7, c=colors, edgecolor="k"
155 )
156 # Draw white circles at cluster centers
157 ax4.scatter(
158     centers[:, 0],
159     centers[:, 1],
160     marker="o",
161     c="white",
162     alpha=1,
163     s=200,
164     edgecolor="k",
165 )
166 ax4.set_xlim([-600,400])
167 ax4.set_ylim([-250,150])
168
169 for i, c in enumerate(centers):
170     ax4.scatter(c[0], c[1], marker="$%d$" % i, alpha=1, s=50, edgecolor
        ="k")
171
172 # Draw classification mean centers
173 for i in range(len(mean_hat_list)):
174     ax4.scatter(
175         mean_hat_list[i][0][0],
176         mean_hat_list[i][0][1],
177         marker="s",
178         c="white",
179         alpha=1,
180         s=200,
181         edgecolor="k",
182     )
183     ax4.scatter(mean_hat_list[i][0][0], mean_hat_list[i][0][1], marker=
        "${}$".format(mean_hat_list[i][1]), alpha=1, s=50, edgecolor="k
        ", c='white')
184
185 ax4.set_title("The visualization of the clustered data. (Zoomed)")
186 ax4.set_xlabel("Feature space for the 1st feature")
187 ax4.set_ylabel("Feature space for the 2nd feature")
188
189 plt.suptitle(
190     "Analysis for KMeans clustering on sample data with n_clusters = %d
        "
191     % n_clusters,
192     fontsize=14,
193     fontweight="bold",
194 )
195
```

```
196 #
    ##########################################################################################
    AX3
197 ax3.plot(mean_sugar, h_train, color = 'lightgreen', label='Sugar',
    linestyle = 'dashdot')
198 ax3.plot(mean_gravel, h_train, color = 'darkgreen', label='Gravel',
    linestyle = 'dashdot')
199 ax3.plot(mean_fish, h_train, color = 'darkblue', label='Fish',
    linestyle = 'dashdot')
200 ax3.plot(mean_flower, h_train, color = 'lightblue', label='Flower',
    linestyle = 'dashdot')
201
202 pattern_dictionary = dict()
203
204 for k in range(n_clusters):
205     pattern_dictionary[k] = list()
206
207
208 for i in range(len(fpca_data)):
209     for k in range(n_clusters):
210         if cluster_labels[i] == k:
211             pattern_dictionary[k].append(data_nd[i])
212         else:
213             pass
214
215 #arrayfication
216 for key in pattern_dictionary:
217     pattern_dictionary[key] = np.array(pattern_dictionary[key])
218
219 #stdev
220 stdev_pattern_dictionary = pattern_dictionary.copy()
221
222 for key in pattern_dictionary:
223     stdev_pattern_dictionary[key] = stats.sem(pattern_dictionary[key])
224
225 #mean
226 mean_pattern_dictionary = pattern_dictionary.copy()
227
228 for key in pattern_dictionary:
229     mean_pattern_dictionary[key] = pmean(pattern_dictionary[key],1)
230
231 #plot patterns
232 for key in pattern_dictionary:
233     Left = mean_pattern_dictionary[key]-2*stdev_pattern_dictionary[key]
234     Right = mean_pattern_dictionary[key]+2*stdev_pattern_dictionary[key
        ]
235     color = cm.nipy_spectral(float(key) / n_clusters)
236
237     ax3.plot(mean_pattern_dictionary[key], Range[:605],
238                         color = color,
239                         label = 'Cluster {}: {}%'.format(key ,round(len
                            (pattern_dictionary[key])*100/len(data_nd))
                            ))
240     ax3.fill_betweenx(Range[:605], Left, Right, facecolor=color, alpha
        =0.5, zorder=-1)
241     ax3.set_ylim(0,4000)
```

```
242     ax3.set_title('The plot of height against cloud fraction.')
243     ax3.axline((0, 800), (1, 800), color = 'grey')
244     ax3.set(xlabel = 'echo fraction / %', ylabel = 'height / m')
245     ax3.legend()
246
247 #   ##########################################################################################
    AX5
248 ax5.plot(mean_sugar, h_train, color = 'lightgreen', label='Sugar',
    linestyle = 'dashdot')
249 ax5.plot(mean_gravel, h_train, color = 'darkgreen', label='Gravel',
    linestyle = 'dashdot')
250 ax5.plot(mean_fish, h_train, color = 'darkblue', label='Fish',
    linestyle = 'dashdot')
251 ax5.plot(mean_flower, h_train, color = 'lightblue', label='Flower',
    linestyle = 'dashdot')
252
253 pattern_dictionary = dict()
254
255 for k in range(n_clusters):
256     pattern_dictionary[k] = list()
257
258
259 for i in range(len(fpca_data)):
260     for k in range(n_clusters):
261         if cluster_labels[i] == k:
262             pattern_dictionary[k].append(data_nd[i])
263         else:
264             pass
265
266 #arrayfication
267 for key in pattern_dictionary:
268     pattern_dictionary[key] = np.array(pattern_dictionary[key])
269
270 #stdev
271 stdev_pattern_dictionary = pattern_dictionary.copy()
272
273 for key in pattern_dictionary:
274     stdev_pattern_dictionary[key] = stats.sem(pattern_dictionary[key])
275
276 #mean
277 mean_pattern_dictionary = pattern_dictionary.copy()
278
279 for key in pattern_dictionary:
280     mean_pattern_dictionary[key] = pmean(pattern_dictionary[key],1)
281
282 #plot patterns
283 for key in pattern_dictionary:
284     Left = mean_pattern_dictionary[key]-2*stdev_pattern_dictionary[key]
285     Right = mean_pattern_dictionary[key]+2*stdev_pattern_dictionary[key
        ]
286     color = cm.nipy_spectral(float(key) / n_clusters)
287
288     ax5.plot(mean_pattern_dictionary[key], Range[:605],
289                         color = color,
```

```
290                             label = 'Cluster {}: {}%'.format(key ,round(len
                                    (pattern_dictionary[key])*100/len(data_nd))
                                    ))
291         ax5.fill_betweenx(Range[:605], Left, Right, facecolor=color, alpha
                =0.5, zorder=-1)
292         ax5.set_xlim(0,23)
293         ax5.set_ylim(0,4000)
294         ax5.set_title('The plot of height against cloud fraction. (Zoomed)'
                )
295         ax5.axline((0, 800), (1, 800), color = 'grey')
296         ax5.set(xlabel = 'echo fraction / %', ylabel = 'height / m')
297
298 #
    ##########################################################################################################
    AX1
299 fig, ax1 = plt.subplots()
300
301 # Compute the silhouette scores for each sample
302 sample_silhouette_values = silhouette_samples(fpca_data, cluster_labels
        )
303
304 ax1.set_xlim([-0.5, 1])
305 # The (n_clusters+1)*10 is for inserting blank space between silhouette
306 # plots of individual clusters, to demarcate them clearly.
307 ax1.set_ylim([0, len(fpca_data) + (n_clusters + 1) * 10])
308
309 y_lower = 10
310 for i in range(n_clusters):
311     # Aggregate the silhouette scores for samples belonging to
312     # cluster i, and sort them
313     ith_cluster_silhouette_values = sample_silhouette_values[
            cluster_labels == i]
314
315     ith_cluster_silhouette_values.sort()
316
317     size_cluster_i = ith_cluster_silhouette_values.shape[0]
318     y_upper = y_lower + size_cluster_i
319
320     color = cm.nipy_spectral(float(i) / n_clusters)
321     ax1.fill_betweenx(
322         np.arange(y_lower, y_upper),
323         0,
324         ith_cluster_silhouette_values,
325         facecolor=color,
326         edgecolor=color,
327         alpha=0.7,
328     )
329
330     # Label the silhouette plots with their cluster numbers at the
            middle
331     ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
332
333     # Compute the new y_lower for next plot
334     y_lower = y_upper + 10  # 10 for the 0 samples
335
336 ax1.set_title("The silhouette plot for the various clusters.")
```

```
337 ax1.set_xlabel("The silhouette coefficient values")
338 ax1.set_ylabel("Cluster label")
339
340 # The vertical line for average silhouette score of all the values
341 ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
342
343 ax1.set_yticks([])  # Clear the yaxis labels / ticks
344 ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
```

E.3.2 K-means Elbow method

Listing 4: Data Reader K-Means Elbow Method.py

```
1  import numpy as np
2  import xarray as xr
3  import pandas as pd
4  from sklearn.cluster import KMeans
5  from skfda.preprocessing.dim_reduction.feature_extraction import FPCA
6  from skfda.representation.grid import FDataGrid
7  from yellowbrick.cluster import KElbowVisualizer
8
9  def find_nearest(array, value):
10
11     array = np.asarray(array)
12     idx = (np.abs(array - value)).argmin()
13
14     return idx
15
16 root_path = 'C:/Users/Gyan/Desktop/BEP Python'
17
18 radar_file = xr.open_dataset(root_path + '/Data/20201' + '/
       MMCR__MBR2__Spectral_Moments__10s__155m-18km__200112.nc')
19
20 Range = radar_file.range.values
21 dataframe_nd = pd.read_csv(root_path + '/CSVData/Data_nd.csv')
22 data_nd = dataframe_nd.to_numpy()
23
24 lower = 0
25
26 upper = 4000
27
28 train_range = range(find_nearest(Range, lower),find_nearest(Range,
       upper))
29 f_data = FDataGrid(data_matrix = data_nd[:,train_range], grid_points=
       Range[train_range])
30
31 fpca_basis = FPCA(2).fit(f_data)
32 fpca_data = fpca_basis.transform(f_data)
33
34 clusterer = KMeans()
35
36 distortion = KElbowVisualizer(clusterer, k=(1,10), locate_elbow = True,
       timings = False, metric = 'distortion')
37
38 # Fit the data to the visualizer
39 distortion.fit(fpca_data)
```

```
40
41 # Finalize and render the figure
42 distortion.show()
```

E.3.3 K-means Gap statistic

Listing 5: Data Reader K-Means Gap Statistic.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import xarray as xr
4  import pandas as pd
5  from skfda.preprocessing.dim_reduction.feature_extraction import FPCA
6  from skfda.representation.grid import FDataGrid
7  from gap_statistic import OptimalK
8
9  def find_nearest(array, value):
10
11     array = np.asarray(array)
12     idx = (np.abs(array - value)).argmin()
13
14     return idx
15
16 root_path = 'C:/Users/Gyan/Desktop/BEP Python'
17
18 radar_file = xr.open_dataset(root_path + '/Data/20201' + '/
       MMCR__MBR2__Spectral_Moments__10s__155m-18km__200112.nc')
19
20 Range = radar_file.range.values
21 dataframe_nd = pd.read_csv(root_path + '/CSVData/Data_nd.csv')
22 data_nd = dataframe_nd.to_numpy()
23
24 lower = 0
25
26 upper = 4000
27
28 train_range = range(find_nearest(Range, lower),find_nearest(Range,
       upper))
29 f_data = FDataGrid(data_matrix = data_nd[:,train_range], grid_points=
       Range[train_range])
30
31 fpca_basis = FPCA(2).fit(f_data)
32 fpca_data = fpca_basis.transform(f_data)
33
34 optimalK = OptimalK()
35
36 n_clusters = optimalK(fpca_data,cluster_array=np.arange(1, 10))
37
38 plt.plot(optimalK.gap_df.n_clusters, optimalK.gap_df.gap_value,
       linewidth=3)
39 plt.scatter(optimalK.gap_df[optimalK.gap_df.n_clusters == n_clusters].
       n_clusters,
40             optimalK.gap_df[optimalK.gap_df.n_clusters == n_clusters].
                   gap_value, s=250, c='r')
41 plt.grid(True)
42 plt.xlabel('Cluster Count')
```

```
plt.ylabel('Gap Value')
plt.title('Gap Values by Cluster Count')
plt.show()
```

E.4 GMM Clustering

E.4.1 GMM Analysis

Listing 6: Data Reader GMM Analysis.py

```
from scipy import stats
from scipy.stats import pmean
import numpy as np
import matplotlib.pyplot as plt
import xarray as xr
import pandas as pd
from skfda.preprocessing.dim_reduction.feature_extraction import FPCA
from skfda.representation.grid import FDataGrid
from scipy.interpolate import interp1d
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.cm as cm
from sklearn.mixture import GaussianMixture

def find_nearest(array, value):

    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()

    return idx

def WPD_Reader(path_input, train_range, range_data):

    dataframe_input = pd.read_csv(path_input, header = None)
    data_input = dataframe_input.to_numpy()

    x_values_input = data_input[:,0]
    y_values_input = data_input[:,1]

    interpolated_input = interp1d(x_values_input,y_values_input)

    mean_input = interpolated_input(range_data[train_range])
    fmean_input = FDataGrid(data_matrix = mean_input, grid_points=
        range_data[train_range])

    return mean_input, fmean_input

root_path = 'C:/Users/Gyan/Desktop/BEP Python'
sugar_path = 'C:/Users/Gyan/Desktop/BEP Python/Extracted Data/Sugar.csv
    '
gravel_path = 'C:/Users/Gyan/Desktop/BEP Python/Extracted Data/Gravel.
    csv'
flower_path = 'C:/Users/Gyan/Desktop/BEP Python/Extracted Data/Flowers.
    csv'
fish_path = 'C:/Users/Gyan/Desktop/BEP Python/Extracted Data/Fish.csv'

```

```
radar_file = xr.open_dataset(root_path + '/Data/20201' + '/
    MMCR__MBR2__Spectral_Moments__10s__155m-18km__200112.nc')

Range = radar_file.range.values
dataframe_nd = pd.read_csv(root_path + '/CSVData/Data_nd.csv')
data_nd = dataframe_nd.to_numpy()

lower = 0

upper = 4000

train_range = range(find_nearest(Range, lower),find_nearest(Range,
    upper))
h_train = Range[train_range]

f_data = FDataGrid(data_matrix = data_nd[:,train_range], grid_points=
    h_train)

fpca_basis = FPCA(2).fit(f_data)
fpca_data = fpca_basis.transform(f_data)

fpca_components = fpca_basis.components_.data_matrix

plt.plot(fpca_components[0][:], h_train, label = 'FPCA1: {}%'.format(
    fpca_basis.explained_variance_ratio_[0].round(2)*100))
plt.plot(fpca_components[1][:], h_train, label = 'FPCA2: {}%'.format(
    fpca_basis.explained_variance_ratio_[1].round(2)*100))
plt.title('FPCA Components')
plt.ylim(0,4000)
plt.xlabel('echo fraction / %')
plt.ylabel('height / m')
plt.legend()
plt.show()

mean_sugar, fmean_sugar = WPD_Reader(sugar_path, train_range, Range)
mean_gravel, fmean_gravel = WPD_Reader(gravel_path, train_range, Range)
mean_flower, fmean_flower = WPD_Reader(flower_path, train_range, Range)
mean_fish, fmean_fish = WPD_Reader(fish_path, train_range, Range)

mean_gravel_hat = [fpca_basis.transform(fmean_gravel)[0], 'G']
mean_sugar_hat = [fpca_basis.transform(fmean_sugar)[0], 'S']
mean_flower_hat = [fpca_basis.transform(fmean_flower)[0], 'FI']
mean_fish_hat = [fpca_basis.transform(fmean_fish)[0], 'FL']

mean_hat_list = [mean_gravel_hat,mean_sugar_hat,mean_flower_hat,
    mean_fish_hat]

n_clusters = 2

##########################################################################PLOT

ax2 = plt.subplot2grid((2, 2), (0, 0))
ax3 = plt.subplot2grid((2, 2), (0, 1))
ax4 = plt.subplot2grid((2, 2), (1, 0))
ax5 = plt.subplot2grid((2, 2), (1, 1))

```

```
92  plt.subplots_adjust(wspace = 0.3, hspace = 0.3)
93
94  # Initialize the clusterer with n_clusters value and a random generator
95  # seed of 10 for reproducibility.
96  clusterer = GaussianMixture(n_components=n_clusters, init_params='k-
        means++')
97  cluster_labels = clusterer.fit_predict(fpca_data)
98
99  #
        ##########################################################################################
        AX2
100 # 2nd Plot showing the actual clusters formed
101 colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
102 ax2.scatter(
103     fpca_data[:, 0], fpca_data[:, 1], marker=".", s=30, lw=0, alpha
        =0.7, c=colors, edgecolor="k"
104 )
105
106 # Labeling the clusters
107 centers = clusterer.means_
108 # Draw white circles at cluster centers
109 ax2.scatter(
110     centers[:, 0],
111     centers[:, 1],
112     marker="o",
113     c="white",
114     alpha=1,
115     s=200,
116     edgecolor="k",
117 )
118
119 for i, c in enumerate(centers):
120     ax2.scatter(c[0], c[1], marker="$%d$" % i, alpha=1, s=50, edgecolor
        ="k")
121
122 # Draw classification mean centers
123 for i in range(len(mean_hat_list)):
124     ax2.scatter(
125         mean_hat_list[i][0][0],
126         mean_hat_list[i][0][1],
127         marker="s",
128         c="white",
129         alpha=1,
130         s=200,
131         edgecolor="k",
132     )
133     ax2.scatter(mean_hat_list[i][0][0], mean_hat_list[i][0][1], marker=
        "${}$".format(mean_hat_list[i][1]), alpha=1, s=50, edgecolor="k
        ", c='white')
134
135 ax2.set_title("The visualization of the clustered data.")
136 ax2.set_xlabel("Feature space for the 1st feature")
137 ax2.set_ylabel("Feature space for the 2nd feature")
138
139 #
        ##########################################################################################
```

```
    AX4
# 2nd Plot showing the actual clusters formed
colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
ax4.scatter(
    fpca_data[:, 0], fpca_data[:, 1], marker=".", s=30, lw=0, alpha
        =0.7, c=colors, edgecolor="k"
)

# Labeling the clusters
centers = clusterer.means_
# Draw white circles at cluster centers
ax4.scatter(
    centers[:, 0],
    centers[:, 1],
    marker="o",
    c="white",
    alpha=1,
    s=200,
    edgecolor="k",
)
ax4.set_xlim([-600,400])
ax4.set_ylim([-250,150])



for i, c in enumerate(centers):
    ax4.scatter(c[0], c[1], marker="$%d$" % i, alpha=1, s=50, edgecolor
        ="k")

# Draw classification mean centers
for i in range(len(mean_hat_list)):
    ax4.scatter(
        mean_hat_list[i][0][0],
        mean_hat_list[i][0][1],
        marker="s",
        c="white",
        alpha=1,
        s=200,
        edgecolor="k",
    )
    ax4.scatter(mean_hat_list[i][0][0], mean_hat_list[i][0][1], marker=
        "${}$".format(mean_hat_list[i][1]), alpha=1, s=50, edgecolor="k
        ", c='white')

ax4.set_title("The visualization of the clustered data. (Zoomed)")
ax4.set_xlabel("Feature space for the 1st feature")
ax4.set_ylabel("Feature space for the 2nd feature")

plt.suptitle(
    "Analysis for GMM clustering on sample data with n_clusters = %d"
    % n_clusters,
    fontsize=14,
    fontweight="bold",
)

```

```
190 #
    ##########################################################################################
    AX3
191 ax3.plot(mean_sugar, h_train, color = 'lightgreen', label='Sugar',
    linestyle = 'dashdot')
192 ax3.plot(mean_gravel, h_train, color = 'darkgreen', label='Gravel',
    linestyle = 'dashdot')
193 ax3.plot(mean_fish, h_train, color = 'darkblue', label='Fish',
    linestyle = 'dashdot')
194 ax3.plot(mean_flower, h_train, color = 'lightblue', label='Flower',
    linestyle = 'dashdot')
195
196 pattern_dictionary = dict()
197
198 for k in range(n_clusters):
199     pattern_dictionary[k] = list()
200
201
202 for i in range(len(fpca_data)):
203     for k in range(n_clusters):
204         if cluster_labels[i] == k:
205             pattern_dictionary[k].append(data_nd[i])
206         else:
207             pass
208
209 #arrayfication
210 for key in pattern_dictionary:
211     pattern_dictionary[key] = np.array(pattern_dictionary[key])
212
213 #stdev
214 stdev_pattern_dictionary = pattern_dictionary.copy()
215
216 for key in pattern_dictionary:
217     stdev_pattern_dictionary[key] = stats.sem(pattern_dictionary[key])
218
219 #mean
220 mean_pattern_dictionary = pattern_dictionary.copy()
221
222 for key in pattern_dictionary:
223     mean_pattern_dictionary[key] = pmean(pattern_dictionary[key],1)
224
225 #plot patterns
226 for key in pattern_dictionary:
227     Left = mean_pattern_dictionary[key]-2*stdev_pattern_dictionary[key]
228     Right = mean_pattern_dictionary[key]+2*stdev_pattern_dictionary[key
        ]
229     color = cm.nipy_spectral(float(key) / n_clusters)
230
231     ax3.plot(mean_pattern_dictionary[key], Range[:605],
232                         color = color,
233                         label = 'Cluster {}: {}%'.format(key ,round(len
                            (pattern_dictionary[key])*100/len(data_nd))
                            ))
234     ax3.fill_betweenx(Range[:605], Left, Right, facecolor=color, alpha
        =0.5, zorder=-1)
235     ax3.set_ylim(0,4000)
```

```
236     ax3.set_title('The plot of height against cloud fraction.')
237     ax3.axline((0, 800), (1, 800), color = 'grey')
238     ax3.set(xlabel = 'echo fraction / %', ylabel = 'height / m')
239     ax3.legend()
240 
241 #   ##########################################################################################
    AX5
242 ax5.plot(mean_sugar, h_train, color = 'lightgreen', label='Sugar',
    linestyle = 'dashdot')
243 ax5.plot(mean_gravel, h_train, color = 'darkgreen', label='Gravel',
    linestyle = 'dashdot')
244 ax5.plot(mean_fish, h_train, color = 'darkblue', label='Fish',
    linestyle = 'dashdot')
245 ax5.plot(mean_flower, h_train, color = 'lightblue', label='Flower',
    linestyle = 'dashdot')
246 
247 pattern_dictionary = dict()
248 
249 for k in range(n_clusters):
250     pattern_dictionary[k] = list()
251 
252 for i in range(len(fpca_data)):
253     for k in range(n_clusters):
254         if cluster_labels[i] == k:
255             pattern_dictionary[k].append(data_nd[i])
256         else:
257             pass
258 
259 #arrayfication
260 for key in pattern_dictionary:
261     pattern_dictionary[key] = np.array(pattern_dictionary[key])
262 
263 #stdev
264 stdev_pattern_dictionary = pattern_dictionary.copy()
265 
266 for key in pattern_dictionary:
267     stdev_pattern_dictionary[key] = stats.sem(pattern_dictionary[key])
268 
269 #mean
270 mean_pattern_dictionary = pattern_dictionary.copy()
271 
272 for key in pattern_dictionary:
273     mean_pattern_dictionary[key] = pmean(pattern_dictionary[key],1)
274 
275 #plot patterns
276 for key in pattern_dictionary:
277     Left = mean_pattern_dictionary[key]-2*stdev_pattern_dictionary[key]
278     Right = mean_pattern_dictionary[key]+2*stdev_pattern_dictionary[key
        ]
279     color = cm.nipy_spectral(float(key) / n_clusters)
280 
281     ax5.plot(mean_pattern_dictionary[key], Range[:605],
282                          color = color,
283                          label = 'Cluster {}: {}%'.format(key ,round(len
                             (pattern_dictionary[key])*100/len(data_nd))
```

```
                                    ))
284     ax5.fill_betweenx(Range[:605], Left, Right, facecolor=color, alpha
            =0.5, zorder=-1)
285     ax5.set_xlim(0,23)
286     ax5.set_ylim(0,4000)
287     ax5.set_title('The plot of height against cloud fraction. (Zoomed)'
            )
288     ax5.axline((0, 800), (1, 800), color = 'grey')
289     ax5.set(xlabel = 'echo fraction / %', ylabel = 'height / m')
290
291 #
    ##########################################################################################
    AX2
292 fig, ax1 = plt.subplots()
293
294 # The 1st subplot is the silhouette plot
295 # The silhouette coefficient can range from -1, 1 but in this example
    all
296 # lie within [-0.1, 1]
297 ax1.set_xlim([-0.5, 1])
298 # The (n_clusters+1)*10 is for inserting blank space between silhouette
299 # plots of individual clusters, to demarcate them clearly.
300 ax1.set_ylim([0, len(fpca_data) + (n_clusters + 1) * 10])
301
302
303 # The silhouette_score gives the average value for all the samples.
304 # This gives a perspective into the density and separation of the
    formed
305 # clusters
306 silhouette_avg = silhouette_score(fpca_data, cluster_labels)
307 print(
308     "For n_clusters =",
309     n_clusters,
310     "The average silhouette_score is :",
311     silhouette_avg,
312 )
313
314 # Compute the silhouette scores for each sample
315 sample_silhouette_values = silhouette_samples(fpca_data, cluster_labels
    )
316
317 y_lower = 10
318 for i in range(n_clusters):
319     # Aggregate the silhouette scores for samples belonging to
320     # cluster i, and sort them
321     ith_cluster_silhouette_values = sample_silhouette_values[
            cluster_labels == i]
322
323     ith_cluster_silhouette_values.sort()
324
325     size_cluster_i = ith_cluster_silhouette_values.shape[0]
326     y_upper = y_lower + size_cluster_i
327
328     color = cm.nipy_spectral(float(i) / n_clusters)
329     ax1.fill_betweenx(
330         np.arange(y_lower, y_upper),
```

```
            0,
            ith_cluster_silhouette_values,
            facecolor=color,
            edgecolor=color,
            alpha=0.7,
        )

        # Label the silhouette plots with their cluster numbers at the
            middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")

# The vertical line for average silhouette score of all the values
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

ax1.set_yticks([])  # Clear the yaxis labels / ticks
ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
```

E.4.2 GMM Elbow method

Listing 7: Data Reader GMM Elbow Method.py

```
import numpy as np
import xarray as xr
import pandas as pd
from sklearn.cluster import KMeans
from skfda.preprocessing.dim_reduction.feature_extraction import FPCA
from skfda.representation.grid import FDataGrid
from yellowbrick.cluster import KElbowVisualizer

def find_nearest(array, value):

    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()

    return idx

root_path = 'C:/Users/Gyan/Desktop/BEP Python'

radar_file = xr.open_dataset(root_path + '/Data/20201' + '/
    MMCR__MBR2__Spectral_Moments__10s__155m-18km__200112.nc')

Range = radar_file.range.values
dataframe_nd = pd.read_csv(root_path + '/CSVData/Data_nd.csv')
data_nd = dataframe_nd.to_numpy()

lower = 0

upper = 4000

```

```
train_range = range(find_nearest(Range, lower),find_nearest(Range,
    upper))
f_data = FDataGrid(data_matrix = data_nd[:,train_range], grid_points=
    Range[train_range])

fpca_basis = FPCA(2).fit(f_data)
fpca_data = fpca_basis.transform(f_data)

clusterer = KMeans()

distortion = KElbowVisualizer(clusterer, k=(1,10), locate_elbow = True,
     timings = False, metric = 'distortion')

# Fit the data to the visualizer
distortion.fit(fpca_data)

# Finalize and render the figure
distortion.show()
```

E.4.3 GMM Gap statistic

Listing 8: Data Reader GMM Gap Statistic.py

```
import numpy as np
import matplotlib.pyplot as plt
import xarray as xr
import pandas as pd
from skfda.preprocessing.dim_reduction.feature_extraction import FPCA
from skfda.representation.grid import FDataGrid
from gap_statistic import OptimalK
from sklearn.mixture import GaussianMixture

def find_nearest(array, value):

    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()

    return idx

root_path = 'C:/Users/Gyan/Desktop/BEP Python'

radar_file = xr.open_dataset(root_path + '/Data/20201' + '/
    MMCR__MBR2__Spectral_Moments__10s__155m-18km__200112.nc')

Range = radar_file.range.values
dataframe_nd = pd.read_csv(root_path + '/CSVData/Data_nd.csv')
data_nd = dataframe_nd.to_numpy()

lower = 0

upper = 4000

train_range = range(find_nearest(Range, lower),find_nearest(Range,
    upper))
f_data = FDataGrid(data_matrix = data_nd[:,train_range], grid_points=
    Range[train_range])
```

```
31
32 fpca_basis = FPCA(2).fit(f_data)
33 fpca_data = fpca_basis.transform(f_data)
34
35 def special_clustering_func(X, k):
36     """
37     Special clustering function which uses the MeanShift
38     model from sklearn.
39
40     These user defined functions *must* take the X and a k
41     and can take an arbitrary number of other kwargs, which can
42     be pass with `clusterer_kwargs` when initializing OptimalK
43     """
44
45     # Here you can do whatever clustering algorithm you heart desires,
46     # but we'll do a simple wrap of the MeanShift model in sklearn.
47
48     m = GaussianMixture()
49     m.fit(X)
50
51     # Return the location of each cluster center,
52     # and the labels for each point.
53     return m.means_, m.predict(X)
54
55 optimalK = OptimalK()
56
57 n_clusters = optimalK(fpca_data,cluster_array=np.arange(1, 20))
58
59 plt.plot(optimalK.gap_df.n_clusters, optimalK.gap_df.gap_value,
       linewidth=3)
60 plt.scatter(optimalK.gap_df[optimalK.gap_df.n_clusters == n_clusters].
       n_clusters,
61             optimalK.gap_df[optimalK.gap_df.n_clusters == n_clusters].
                   gap_value, s=250, c='r')
62 plt.grid(True)
63 plt.xlabel('Cluster Count')
64 plt.ylabel('Gap Value')
65 plt.title('Gap Values by Cluster Count')
66 plt.show()
```

E.5 Mean Shift Clustering

E.5.1 Mean Shift Analysis

Listing 9: Data Reader MS Analysis.py

```
1 from scipy import stats
2 from scipy.stats import pmean
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import xarray as xr
6 import pandas as pd
7 from skfda.preprocessing.dim_reduction.feature_extraction import FPCA
8 from skfda.representation.grid import FDataGrid
9 from scipy.interpolate import interp1d
10 from sklearn.metrics import silhouette_samples, silhouette_score
```

```
11 import matplotlib.cm as cm
12 from sklearn.cluster import MeanShift
13 from sklearn.cluster import estimate_bandwidth
14
15 def find_nearest(array, value):
16
17     array = np.asarray(array)
18     idx = (np.abs(array - value)).argmin()
19
20     return idx
21
22 def WPD_Reader(path_input, train_range, range_data):
23
24     dataframe_input = pd.read_csv(path_input, header = None)
25     data_input = dataframe_input.to_numpy()
26
27     x_values_input = data_input[:,0]
28     y_values_input = data_input[:,1]
29
30     interpolated_input = interp1d(x_values_input,y_values_input)
31
32     mean_input = interpolated_input(range_data[train_range])
33     fmean_input = FDataGrid(data_matrix = mean_input, grid_points=
           range_data[train_range])
34
35     return mean_input, fmean_input
36
37 root_path = 'C:/Users/Gyan/Desktop/BEP Python'
38 sugar_path = 'C:/Users/Gyan/Desktop/BEP Python/Extracted Data/Sugar.csv
       '
39 gravel_path = 'C:/Users/Gyan/Desktop/BEP Python/Extracted Data/Gravel.
       csv'
40 flower_path = 'C:/Users/Gyan/Desktop/BEP Python/Extracted Data/Flowers.
       csv'
41 fish_path = 'C:/Users/Gyan/Desktop/BEP Python/Extracted Data/Fish.csv'
42
43 radar_file = xr.open_dataset(root_path + '/Data/20201' + '/
       MMCR__MBR2__Spectral_Moments__10s__155m-18km__200112.nc')
44
45 Range = radar_file.range.values
46 dataframe_nd = pd.read_csv(root_path + '/CSVData/Data_nd.csv')
47 data_nd = dataframe_nd.to_numpy()
48
49 lower = 0
50
51 upper = 4000
52
53 train_range = range(find_nearest(Range, lower),find_nearest(Range,
       upper))
54 h_train = Range[train_range]
55
56 f_data = FDataGrid(data_matrix = data_nd[:,train_range], grid_points=
       h_train)
57
58 fpca_basis = FPCA(2).fit(f_data)
59 fpca_data = fpca_basis.transform(f_data)
```

```
60
61 fpca_components = fpca_basis.components_.data_matrix
62
63 plt.plot(fpca_components[0][:], h_train, label = 'FPCA1: {}%'.format(
       fpca_basis.explained_variance_ratio_[0].round(2)*100))
64 plt.plot(fpca_components[1][:], h_train, label = 'FPCA2: {}%'.format(
       fpca_basis.explained_variance_ratio_[1].round(2)*100))
65 plt.title('FPCA Components')
66 plt.ylim(0,4000)
67 plt.xlabel('echo fraction / %')
68 plt.ylabel('height / m')
69 plt.legend()
70 plt.show()
71
72 mean_sugar, fmean_sugar = WPD_Reader(sugar_path, train_range, Range)
73 mean_gravel, fmean_gravel = WPD_Reader(gravel_path, train_range, Range)
74 mean_flower, fmean_flower = WPD_Reader(flower_path, train_range, Range)
75 mean_fish, fmean_fish = WPD_Reader(fish_path, train_range, Range)
76
77 mean_gravel_hat = [fpca_basis.transform(fmean_gravel)[0], 'G']
78 mean_sugar_hat = [fpca_basis.transform(fmean_sugar)[0], 'S']
79 mean_flower_hat = [fpca_basis.transform(fmean_flower)[0], 'FI']
80 mean_fish_hat = [fpca_basis.transform(fmean_fish)[0], 'FL']
81
82 mean_hat_list = [mean_gravel_hat,mean_sugar_hat,mean_flower_hat,
       mean_fish_hat]
83
84 n_clusters = 2
85
86 ##################################################################PLOTS
87
88 # Create a subplot with 1 row and 3 columns
89 ax2 = plt.subplot2grid((2, 2), (0, 0))
90 ax3 = plt.subplot2grid((2, 2), (0, 1))
91 ax4 = plt.subplot2grid((2, 2), (1, 0))
92 ax5 = plt.subplot2grid((2, 2), (1, 1))
93
94
95 plt.subplots_adjust(wspace = 0.3, hspace = 0.3)
96
97 # Initialize the clusterer with n_clusters value and a random generator
98 # seed of 10 for reproducibility.
99 bandwidth_est = estimate_bandwidth(fpca_data, quantile=0.5)
100
101 clusterer = MeanShift(cluster_all = False, bandwidth=bandwidth_est)
102 cluster_labels = clusterer.fit_predict(fpca_data)
103 n_clusters = max(cluster_labels) + 1
104
105 #
       ##########################################################################################################
       AX2
106 # 2nd Plot showing the actual clusters formed
107 colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
108 ax2.scatter(
109     fpca_data[:, 0], fpca_data[:, 1], marker=".", s=30, lw=0, alpha
          =0.7, c=colors, edgecolor="k"
```

```
110 )
111
112 # Labeling the clusters
113 centers = clusterer.cluster_centers_
114 # Draw white circles at cluster centers
115 ax2.scatter(
116     centers[:, 0],
117     centers[:, 1],
118     marker="o",
119     c="white",
120     alpha=1,
121     s=200,
122     edgecolor="k",
123 )
124
125 for i, c in enumerate(centers):
126     ax2.scatter(c[0], c[1], marker="$%d$" % i, alpha=1, s=50, edgecolor
        ="k")
127
128 # Draw classification mean centers
129 for i in range(len(mean_hat_list)):
130     ax2.scatter(
131         mean_hat_list[i][0][0],
132         mean_hat_list[i][0][1],
133         marker="s",
134         c="white",
135         alpha=1,
136         s=200,
137         edgecolor="k",
138     )
139     ax2.scatter(mean_hat_list[i][0][0], mean_hat_list[i][0][1], marker=
        "${}$".format(mean_hat_list[i][1]), alpha=1, s=50, edgecolor="k
        ", c='white')
140
141 ax2.set_title("The visualization of the clustered data.")
142 ax2.set_xlabel("Feature space for the 1st feature")
143 ax2.set_ylabel("Feature space for the 2nd feature")
144
145 #
    ##################################################################################################
    AX4
146 # 2nd Plot showing the actual clusters formed
147 colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
148 ax4.scatter(
149     fpca_data[:, 0], fpca_data[:, 1], marker=".", s=30, lw=0, alpha
        =0.7, c=colors, edgecolor="k"
150 )
151 # Draw white circles at cluster centers
152 ax4.scatter(
153     centers[:, 0],
154     centers[:, 1],
155     marker="o",
156     c="white",
157     alpha=1,
158     s=200,
159     edgecolor="k",
```

```
160 )
161 ax4.set_xlim([-600,400])
162 ax4.set_ylim([-250,150])
163
164 for i, c in enumerate(centers):
165     ax4.scatter(c[0], c[1], marker="$%d$" % i, alpha=1, s=50, edgecolor
        ="k")
166
167 # Draw classification mean centers
168 for i in range(len(mean_hat_list)):
169     ax4.scatter(
170         mean_hat_list[i][0][0],
171         mean_hat_list[i][0][1],
172         marker="s",
173         c="white",
174         alpha=1,
175         s=200,
176         edgecolor="k",
177     )
178     ax4.scatter(mean_hat_list[i][0][0], mean_hat_list[i][0][1], marker=
        "${}$".format(mean_hat_list[i][1]), alpha=1, s=50, edgecolor="k
        ", c='white')
179
180 ax4.set_title("The visualization of the clustered data. (Zoomed)")
181 ax4.set_xlabel("Feature space for the 1st feature")
182 ax4.set_ylabel("Feature space for the 2nd feature")
183
184 plt.suptitle(
185     "Analysis for Mean Shift clustering on sample data"
186     % n_clusters,
187     fontsize=14,
188     fontweight="bold",
189 )
190
191 #
    ##########################################################################################################
    AX3
192 ax3.plot(mean_sugar, h_train, color = 'lightgreen', label='Sugar',
    linestyle = 'dashdot')
193 ax3.plot(mean_gravel, h_train, color = 'darkgreen', label='Gravel',
    linestyle = 'dashdot')
194 ax3.plot(mean_fish, h_train, color = 'darkblue', label='Fish',
    linestyle = 'dashdot')
195 ax3.plot(mean_flower, h_train, color = 'lightblue', label='Flower',
    linestyle = 'dashdot')
196
197
198 pattern_dictionary = dict()
199
200 for k in range(n_clusters):
201     pattern_dictionary[k] = list()
202
203
204 for i in range(len(fpca_data)):
205     for k in range(n_clusters):
206         if cluster_labels[i] == k:
```

```
207                 pattern_dictionary[k].append(data_nd[i])
208             else:
209                 pass
210 
211 #arrayfication
212 for key in pattern_dictionary:
213     pattern_dictionary[key] = np.array(pattern_dictionary[key])
214 
215 #stdev
216 stdev_pattern_dictionary = pattern_dictionary.copy()
217 
218 for key in pattern_dictionary:
219     stdev_pattern_dictionary[key] = stats.sem(pattern_dictionary[key])
220 
221 #mean
222 mean_pattern_dictionary = pattern_dictionary.copy()
223 
224 for key in pattern_dictionary:
225     mean_pattern_dictionary[key] = pmean(pattern_dictionary[key],1)
226 
227 #plot patterns
228 for key in pattern_dictionary:
229     Left = mean_pattern_dictionary[key]-2*stdev_pattern_dictionary[key]
230     Right = mean_pattern_dictionary[key]+2*stdev_pattern_dictionary[key
        ]
231     color = cm.nipy_spectral(float(key) / n_clusters)
232 
233     ax3.plot(mean_pattern_dictionary[key], Range[:605],
234                          color = color,
235                          label = 'Cluster {}: {}%'.format(key ,round(len
                                 (pattern_dictionary[key])*100/len(data_nd))
                                 ))
236     ax3.fill_betweenx(Range[:605], Left, Right, facecolor=color, alpha
        =0.5, zorder=-1)
237     ax3.set_ylim(0,4000)
238     ax3.set_title('The plot of height against cloud fraction.')
239     ax3.axline((0, 800), (1, 800), color = 'grey')
240     ax3.set(xlabel = 'echo fraction / %', ylabel = 'height / m')
241     ax3.legend()
242 
243 #
    ##########################################################################################
    AX5
244 ax5.plot(mean_sugar, h_train, color = 'lightgreen', label='Sugar',
    linestyle = 'dashdot')
245 ax5.plot(mean_gravel, h_train, color = 'darkgreen', label='Gravel',
    linestyle = 'dashdot')
246 ax5.plot(mean_fish, h_train, color = 'darkblue', label='Fish',
    linestyle = 'dashdot')
247 ax5.plot(mean_flower, h_train, color = 'lightblue', label='Flower',
    linestyle = 'dashdot')
248 
249 
250 pattern_dictionary = dict()
251 
252 for k in range(n_clusters):
```

```
253     pattern_dictionary[k] = list()
254
255
256 for i in range(len(fpca_data)):
257     for k in range(n_clusters):
258         if cluster_labels[i] == k:
259             pattern_dictionary[k].append(data_nd[i])
260         else:
261             pass
262
263 #arrayfication
264 for key in pattern_dictionary:
265     pattern_dictionary[key] = np.array(pattern_dictionary[key])
266
267 #stdev
268 stdev_pattern_dictionary = pattern_dictionary.copy()
269
270 for key in pattern_dictionary:
271     stdev_pattern_dictionary[key] = stats.sem(pattern_dictionary[key])
272
273 #mean
274 mean_pattern_dictionary = pattern_dictionary.copy()
275
276 for key in pattern_dictionary:
277     mean_pattern_dictionary[key] = pmean(pattern_dictionary[key],1)
278
279 #plot patterns
280 for key in pattern_dictionary:
281     Left = mean_pattern_dictionary[key]-2*stdev_pattern_dictionary[key]
282     Right = mean_pattern_dictionary[key]+2*stdev_pattern_dictionary[key
        ]
283     color = cm.nipy_spectral(float(key) / n_clusters)
284
285     ax5.plot(mean_pattern_dictionary[key], Range[:605],
286                         color = color,
287                         label = 'Cluster {}: {}%'.format(key ,round(len
                            (pattern_dictionary[key])*100/len(data_nd))
                            ))
288     ax5.fill_betweenx(Range[:605], Left, Right, facecolor=color, alpha
        =0.5, zorder=-1)
289     ax5.set_xlim(0,23)
290     ax5.set_ylim(0,4000)
291     ax5.set_title('The plot of height against cloud fraction. (Zoomed)'
        )
292     ax5.axline((0, 800), (1, 800), color = 'grey')
293     ax5.set(xlabel = 'echo fraction / %', ylabel = 'height / m')
294
295 #
    ##############################################################################
    AX1
296 fig, ax1 = plt.subplots()
297
298 # The 1st subplot is the silhouette plot
299 # The silhouette coefficient can range from -1, 1 but in this example
    all
300 # lie within [-0.1, 1]
```

```
301 ax1.set_xlim([-0.5, 1])
302 # The (n_clusters+1)*10 is for inserting blank space between silhouette
303 # plots of individual clusters, to demarcate them clearly.
304 ax1.set_ylim([0, len(fpca_data) + (n_clusters + 1) * 10])
305
306
307 # The silhouette_score gives the average value for all the samples.
308 # This gives a perspective into the density and separation of the
    formed
309 # clusters
310 silhouette_avg = silhouette_score(fpca_data, cluster_labels)
311 print(
312     "For n_clusters =",
313     n_clusters,
314     "The average silhouette_score is :",
315     silhouette_avg,
316 )
317
318 # Compute the silhouette scores for each sample
319 sample_silhouette_values = silhouette_samples(fpca_data, cluster_labels
    )
320
321 y_lower = 10
322 for i in range(n_clusters):
323     # Aggregate the silhouette scores for samples belonging to
324     # cluster i, and sort them
325     ith_cluster_silhouette_values = sample_silhouette_values[
        cluster_labels == i]
326
327     ith_cluster_silhouette_values.sort()
328
329     size_cluster_i = ith_cluster_silhouette_values.shape[0]
330     y_upper = y_lower + size_cluster_i
331
332     color = cm.nipy_spectral(float(i) / n_clusters)
333     ax1.fill_betweenx(
334         np.arange(y_lower, y_upper),
335         0,
336         ith_cluster_silhouette_values,
337         facecolor=color,
338         edgecolor=color,
339         alpha=0.7,
340     )
341
342     # Label the silhouette plots with their cluster numbers at the
        middle
343     ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
344
345     # Compute the new y_lower for next plot
346     y_lower = y_upper + 10  # 10 for the 0 samples
347
348 ax1.set_title("The silhouette plot for the various clusters.")
349 ax1.set_xlabel("The silhouette coefficient values")
350 ax1.set_ylabel("Cluster label")
351
352 # The vertical line for average silhouette score of all the values
```

```
353 ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
354
355 ax1.set_yticks([])  # Clear the yaxis labels / ticks
356 ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
```