Customer Segmentation for Subscriptions

IDS 575 Machine Learning Statistics

Project Idea

To attract more clients to pay a subscription for a dating application company aims to add new subscription tiers based on their age, income, and education. The sales staff has divided all their consumers into four groups in their current market (A, B, C, D). Following that, they conducted segmented outreach and communication for several client segments. They've had a lot of success with this method. They intend to apply the same method to additional markets, and they have found 2627 new potential clients. The main idea is to create a machine learning model to assist the application in predicting the proper type of incoming consumer.

Project Scope

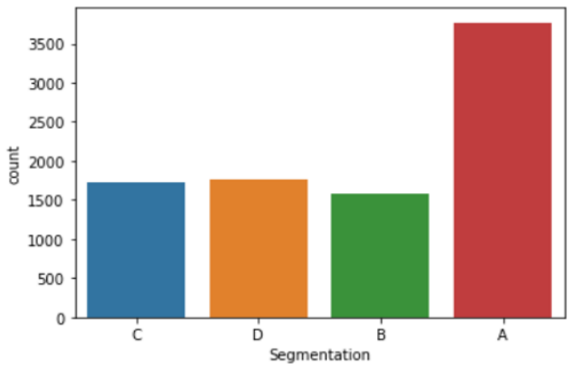
Developing a model based on a clustering algorithm to predict customer segments based on various parameters. We will work on the historic data collected by the company to build and train a machine learning model. Also, we will use various techniques to evaluate the model accuracy. This clustering prediction model with suitable datasets and data features can be used to predict segments in use-cases like the targeted audience.

Data Curation and Exploration

Our dataset contains **10695 rows and 11 columns.** The 11 variables in our dataset are: ID, Gender, Ever_Married, Age, Graduated, Profession, Work_Experience, Spending_Score, Family_Size, Var_1, and Segmentation. Amongst the given features, 7 are categorical and 4 are numerical. We do have missing values in 6 of the features and thus the number of rows decreased to 8819 after we dropped the missing values. Below are given details for the important features.

Segmentation Data:

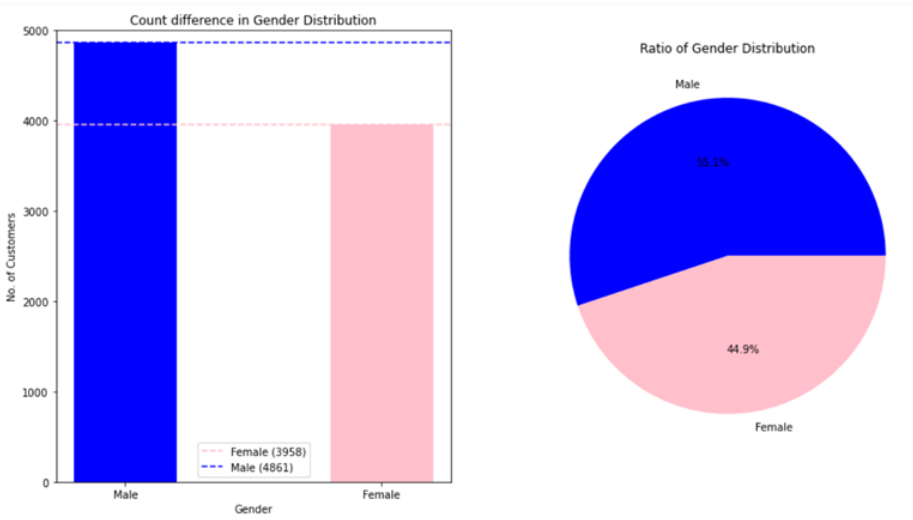
The below data shows the segmentation of each of the customers in four different segments. A, B, C and D where all the variables like Age, Marital Status, Gender, Work experience, etc are considered.




```
A    3770
D    1757
C    1720
B    1572
Name: Segmentation, dtype: int64
```

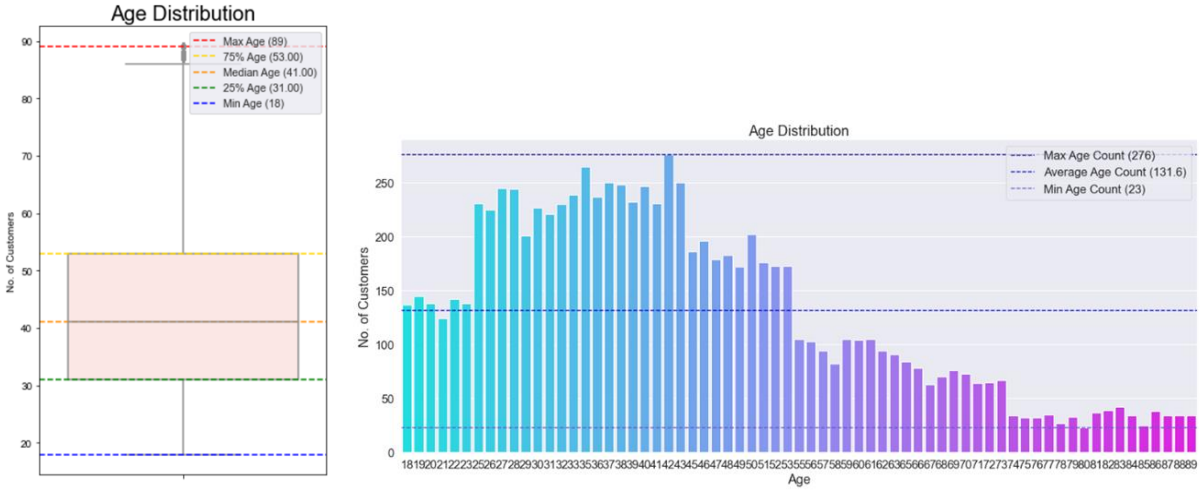
Gender Distribution:

The Gender Distribution reveals 44.9% (3958) of the population are male and 55.1% (4861) are female. The population consists of people of all ages which we shall see in upcoming sections.



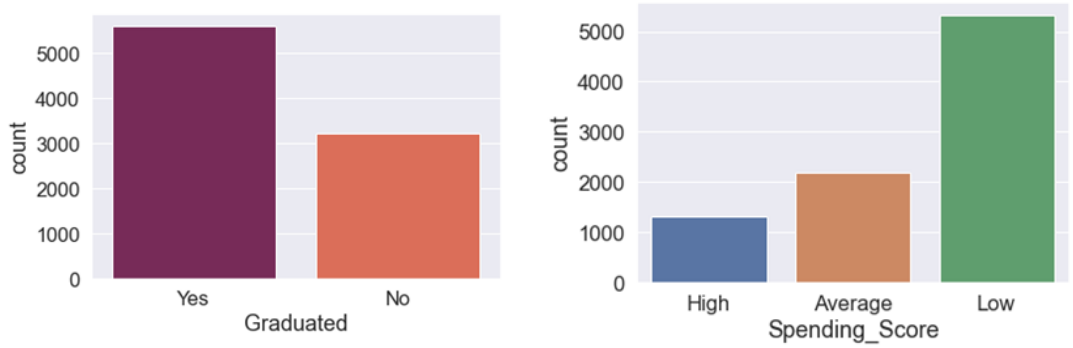
Age Distribution:

The age of the population ranges from 18 to 89 years of age with the concentration between being 25 to 55 years of age. The box plot as can be seen has its minima, first quartile and the median spread only in the range of the 20 years while the rest spread across 50 years. The next graph shows a bell curve which is slightly left skewed and thus depicts that the use of specific dating app is common among younger ages.



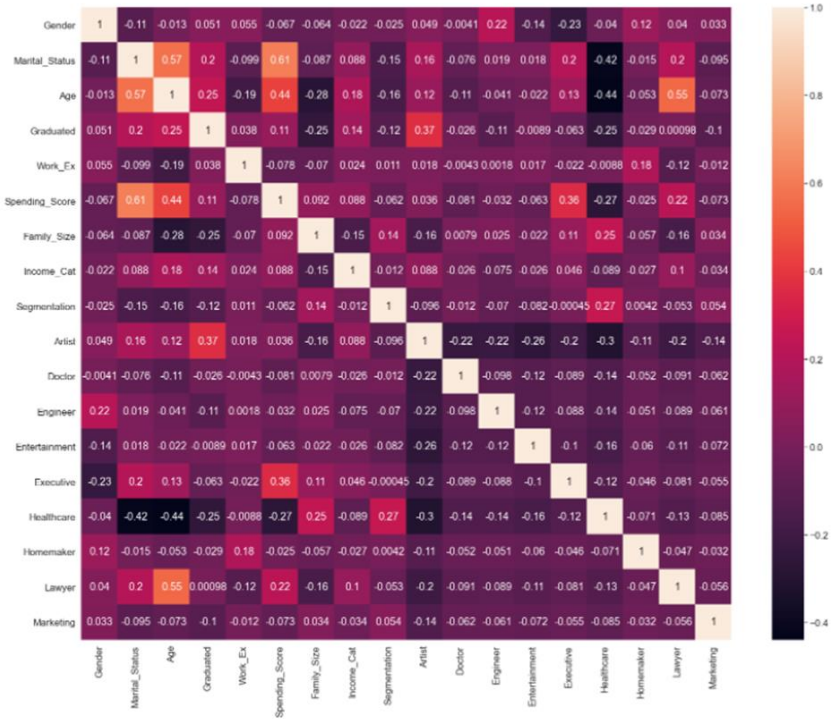
Graduated and Spending Score:

More than 5000 subscribers indicated that they have graduated from college, while 3000 claimed that they ha-ve not. While the spending scores for the subscribers show that most of the population has lower spending scores thus showing that the population might belong to people who believe in saving money given the age distribution.



Correlation Matrix:

After seeing all features and extracting proper visualization for each of those, we thought to check which of the variables interacted at most. Thus, the below heatmap discloses that Age, Graduated, Work_Experience, and Speding_Score have the highest correlation to segmentation classes.



K-means Clustering

Description:

K-means clustering is an unsupervised learning algorithm. As an 'unsupervised model' it only used input vectors without referring to the known labels or outcomes. Hence, the objective of K-means clustering is to group similar data together and discover underlying patterns.

In general, the K-means algorithm identifies 'k' centroids (averaging of the data), and then assigns every data point to the closest cluster, while maintaining the centroids as small as possible.

Procedure:

To process the given data, the K-means algorithm in data mining begins by randomly selecting the centroids, which are then used to perform iterative calculation to optimize the positions of the centroids.

It deters creating and optimizing clusters, when either:

- The centroids have stabilized (no change in their values) because the clustering has been successful.
- The defined number of iterations has been achieved.

Implementation:

Constructed a class with 3 methods - to initialize, fit and predict outcomes for the model.

Initialize: This method takes three parameters and initializes them, as –

- 'k' takes an int value as input, that is the value of desired clusters
- 'tol' takes a float value as input, which is the optimization criteria
- 'max_iter' takes int value as input, which is the maximum number of iterations the algorithm must run in case the centroids are not stabilized.

Fit: The method has the procedure (core logic) to obtain clusters and optimize centroids. Initially, it randomly assigns data points to centroids (number of centroid = number of clusters). In the next steps we use 'Euclidean Distance' to calculate distances and update the centroids value till either maximum iterations are reached or are stabilized. It returns the centroids and classified data into segments for the data (type: pd.DataFrame) passed as the argument.

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Predict: This method has the procedure to predict segmentation labels based on the 'Euclidean Distance' Formula. It returns a list of predicted segmentation labels based on the data (type: pd.DataFrame) passed as the argument.

Independent Functions:

Principal Component Analysis (PCA): With our identified variables impacting the segmentation, it was quite difficult to visualize the resulting data. Therefore, PCA is used to reduce the dimensionality of the data (reduced the dimensionality to 3). This helped us better visualize the k-means outcome.

This method takes data (type: pd.DataFrame) and number of components (reduced dimension variable, type: int) and returns data with reduced dimension.

Cost Function: This method takes data and number of trials. Then we initialize a new kmeans instance for each trial and compute the cost and accuracy score for that particular value of 'k'.

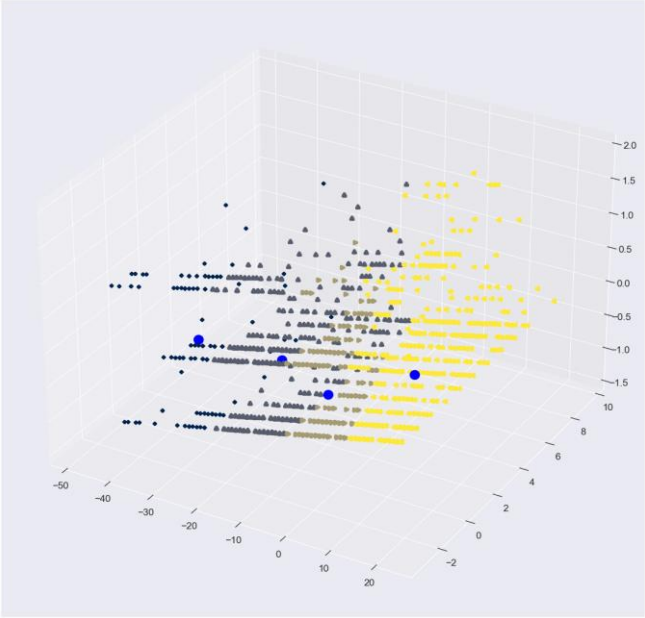
The costs are calculated using 'Elbow method' based on Within-Cluster Sum of Squares (WCSS) and then iteratively store its value corresponding to various values of 'k'.

The accuracy score is calculated using 'Silhouette score' based on the separation between the clusters. We are using the 'scikit-learn' metrics method to evaluate the score. The scores for each iteration are then stored iteratively corresponding to various values of 'k'. The results of the function help analyze the optimal 'k' value to be used for model fitting, visualization and model prediction.

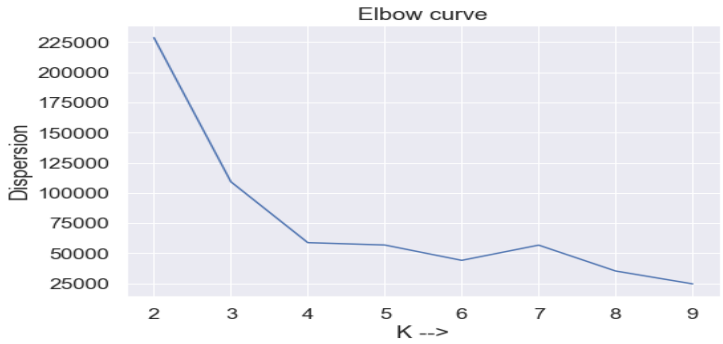
Train and Test Splitting: This method randomly splits dataset into test and train data based on the unequal ration. It accepts data (type: pd.DataFrame) and the fraction (type: float, value: between 0 to 1) and returns two data sets.

Visualization Functions:

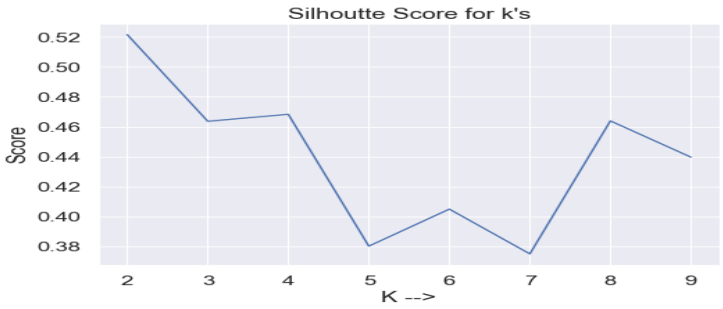
Graph Plot Function: This function takes centroids, cluster data and 'k' as the arguments and then projects the data into a 3-dimensional graph. A reference graph from training data with the optimal value of k (same as used for model fitting) being '4'.



Elbow Curve Plot Function: This function takes the costs (output from cost function) and the number of trials (same as in cost function) and plots the values corresponding to different 'k'.



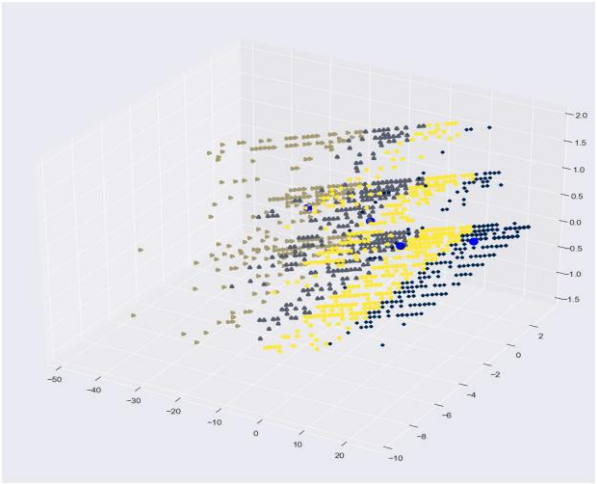
Silhouette Score Plot Function: This function takes the scores (output from cost function) and the number of trials (same as in cost function) and plots the values corresponding to different ‘k’.



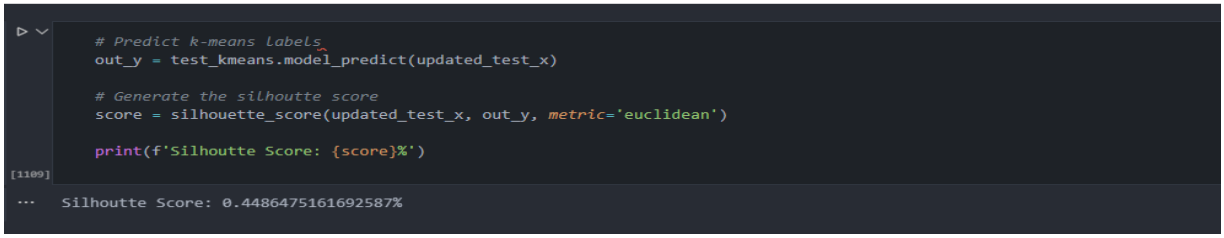
Results:

During the implementation, we are using random methods for test and train data splitting, which are highly impacting the optimal value of ‘k’. After carefully varying the values for the model fitting on the training set. Following are the results obtained for K-means clustering for the value of k as 4, and a dataset containing 4 columns namely ‘Age’, ‘Graduated’, ‘Work Experience’, and ‘Spending Score’.

A cluster visualization for the same can be seen as below.



The accuracy score for the same are attached below:



```
# Predict k-means labels
out_y = test_kmeans.model_predict(updated_test_x)

# Generate the silhoutte score
score = silhouette_score(updated_test_x, out_y, metric='euclidean')

print(f'Silhoutte Score: {score}%')
```

[1109]

```
Silhoutte Score: 0.4486475161692587%
```

Conclusions:

We can use the k-means clustering to classify data, but we will have to use a different method for training and test data split, as we can avoid the repeated changes in the optimal k-value and the accuracy score for the model. Moreover, k-means will help us analyze the overlapping data which brings into attention the dual interests of a data point.

Density-based spatial clustering of application with noise (DBSCAN)

Description:

Unlike K-means which is very sensitive to the number of clusters and outliers, DBScan is a density-based clustering algorithm that does not specify the number of clusters, and it is not sensitive to outliers. DBScan is a form of unsupervised clustering algorithm that can automatically identify clusters based on the data input and parameters. The major difference between DBScan and K-means is that DBScan can find arbitrary shapes on it own. It has amazing performance at handling outliers and noise because anything that is not within reachable distance from the core point is considered an outlier.

Procedures:

- Clean the data by dropping the na.values and transforming any non-numerical values into numerical values.
- Define the eps (epsilon), which is the radius of the circle that will be created around the data points for identifying the density.
- Define the min_samples (minPoints), which is the minimum number of data points inside a circle to complete the neighbor, or to classify the as a core point.
- We use a loop to go over the different values of eps to find the number of clusters, for dbscan, we can only set eps and not the number of clusters, we need to find out at which eps, we can get 4 clusters
- We plot the number of clusters against the Epsilons to identify the number of epsilon which will give us four clusters since we have four customer groups in our customer segmentation data.
- After we identify the epsilon, we put it into the DBScan model, which will return the number of neighbors in each cluster.
- Then we compare the result from the DBScan model to the original customer segmentation from the data to calculate the accuracy of the model.

Implementation:

def __init__: define the radius of the circle and the minimum number of points inside each circle/neighbor.

def find_distances: Calculate the distance between all pairs.

def find_neightbors: Return the indices of neighbors for specific observations.

def fit: find the number of clusters in the dataset by grouping all the neighbors within the density connected point. It also identifies the outliers when they do not satisfy the min_sample and eps requirements.

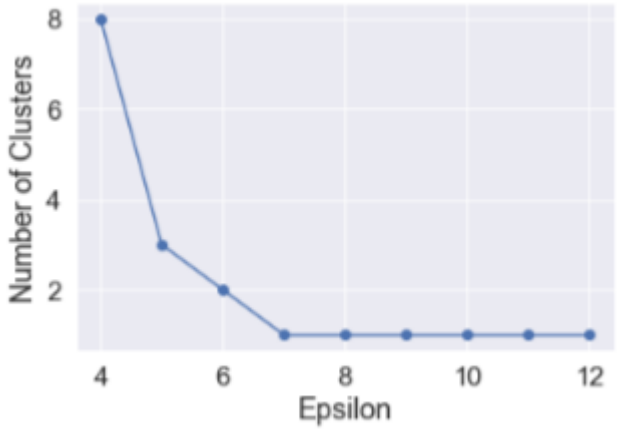
Results:

Accuracy of the predicted segmentation

```
In [313]: 1 adjusted_rand_score(data['Segmentation'], dbscan.labels_)
Out[313]: 0.004704905550590988
```

DBSCAN is not useful because we cannot specify the number of clusters as parameters. We need to change the eps (radius of clusters to find in which radius we can get 4 clusters. Even in this case, there is a chance that observation is part of any cluster. The matching score between customer segmentation and the DBScan cluster is only 0.47%.

It is difficult to find the number of eps that would give us exact four clusters because it is somewhere in between eps of 4 and 5. Therefore, we have to try to input different eps values to check which one will give me four clusters. After several trials, we found that eps set to 4.35 would give us four clusters.



If we set the eps to 4, the model would return 8 clusters and the accuracy would be -0.011%. If we set the eps to 5, it would return only 3 clusters with the accuracy of -0.000245%.

If we set the eps to 4, the model would return 8 clusters and the accuracy would be -0.011%. If we set the eps to 5, it would return only 3 clusters with the accuracy of -0.000245%. So, a higher or lower eps value would give us a very poor result.

Eps 4.35 accuracy

```
In [313]: 1 adjusted_rand_score(data['Segmentation'], dbscan.labels_)
Out[313]: 0.004704905550590988
```

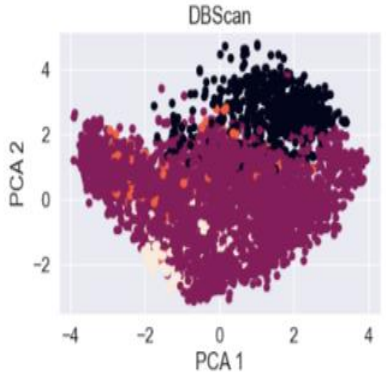
Eps 4 accuracy

```
In [380]: 1 adjusted_rand_score(data['Segmentation'], dbscan.labels_)
Out[380]: -0.011269595374259244
```

Eps 5 accuracy

```
In [375]: 1 adjusted_rand_score(data['Segmentation'], dbscan.labels_)
Out[375]: -0.00024565633673403174
```

Visualization is done with the Principal Component Analysis (PCA) function taken from K-means clustering to reduce the dimensionality to 2 dimensions. A scatter plot function is used to display the data points along with their cluster group. Each color represents a different cluster. Overall, the plot looks very overwhelming and hard to interpret.



K-Nearest Neighbors Algorithm (KNN)

Description:

A **supervised machine learning** algorithm (as opposed to an unsupervised machine learning algorithm) is one that relies on labeled input data to learn a function that produces an appropriate output when given new unlabeled data. A **classification problem** has a discrete value as its output. For example, in our dataset its segmentation "like A, B, C, D".

An **unsupervised machine learning** algorithm makes use of input data without any labels —in other words, no teacher (label) telling the child (computer) when it is right or when it has made a mistake so that it can **self-correct.** Unlike supervised learning that tries to learn a function that will allow us to make predictions given some new unlabeled data and is also called **a lazy learner**, unsupervised learning tries to learn the basic structure of the data to give us more insight into the data.

We implemented KNN to predict the upcoming customer's label to assign them in the segment with closest proximity. In other words, with similar features near each other. The algorithm hinges on the assumption being true enough for the algorithm to be useful. It captures the idea of similarity (called distance, proximity, or closeness) — calculating the distance between points.

Procedure:

Steps for implementing KNN Algorithm:

1. Load the data
2. Initialize K to your chosen number of neighbors
3. For each example in the data
   3.1 Calculate the distance between the query example and the current example from the data.
   3.2 Add the distance and the index of the example to an ordered collection
4. Sort the ordered collection of distances and indices from smallest to largest by the distances
5. Pick the first K entries from the sorted collection
6. Get the labels of the selected K entries
7. If regression, return the mean of the K labels, if classification, return the mode of the K labels

Implementation:

Constructed three methods to implement KNN

- **def nearest_Neighbours:** Calculate the euclidean distance and select top k nearest distances

$$\text{Euclidean distance} = \sqrt{\sum_{i=1}^{n} \left(x_i - y_i\right)^2}$$

- **def knn_prediction:** Predicting the label of the new piece of data based in k-nearest neighbors
- **def knn_getAccuracy:** Accuracy of correctly predicted

K value (here we've considered k =5) indicates the count of the nearest neighbors. We have to compute distances between test points and trained labels points. Updating distance metrics with every iteration is computationally expensive, and that's why **KNN is a lazy learning algorithm**.

Results:

We have calculated the accuracy of Model by splitting data as train and test data

Accuracy of Model

```
In [128]: # Accuracy of predicted species
          output=knn_prediction(X_train,y_train,X_test,100)

          knn_getAccuracy(y_test,output)

Out[128]: 48.36
```

Conclusion:

KNN's main disadvantage of becoming significantly slower as the volume of data increases makes it an impractical choice in environments where predictions need to be made rapidly. Moreover, there are faster algorithms that can produce more accurate classification results.
However, it's easy to implement and understand.

Insights

We have provided three different ways for the company to segment the population:

1. Kmeans – Unsupervised algorithm where the segmentation is based on number of clusters (Advantage: N number of clusters)
2. DBSCAN – Unsupervised algorithm where the segmentation is done by Radial clustering or area of target (Advantage: N number of clusters)
3. KNN – Supervised algorithm where the segmentation would consider the existing customer segmentation logic

All the above methods show the accuracy ranging between 40%-60% which implies that the models are a good fit for new data. The company can choose any option to go forward with depending on the needs of the company. However, we would suggest against KNN since it is a very slow method and does not add any new insight to already existing logic.

References

1. https://towardsdatascience.com/knn-using-scikit-learn-c6bed765be75
2. https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/
3. https://medium.com/analytics-vidhya/easy-knn-algorithm-using-scikit-learn-7f6e256c9453
4. https://python.plainenglish.io/full-implementation-of-knn-k-nearest-neighbors-in-machine-learning-using-scikit-learn-2134168f76a3
5. https://scrunts23.medium.com/dbscan-algorithm-from-scratch-in-python-475b82e0571c
6. https://aihubprojects.com/k-means-clustering-from-scratch-python/
7. https://www.codecademy.com/learn/machine-learning/modules/dspath-clustering/cheatsheet
8. https://www.geeksforgeeks.org/python-dictionary-values-mean/
9. https://hands-on.cloud/implementation-of-k-means-clustering-algorithm-using-python/#h-k-mean-clustering-algorithm-overview
10. https://neptune.ai/blog/k-means-clustering