K-Means Clustering, Mall Customers and Voronoi Diagrams

Ashish Kumar Patra

March 18, 2022

1 Introduction

K-Means Clustering is considered to be a very efficient unsupervised machine learning algorithm, as it seemingly "recognizes" patterns without any form of outer intervention or inputs. It allows us to cluster data points to certain categories by assimilating and evaluating their differences and similarities. Here, we first use the `sklearn` module to perform the clustering and later we try to build a very specific and hard coded K-Means Clustering algorithm from scratch, and see how it works at each step. We use a simple data set[2] from Kaggle containing the data from customers who frequent a mall. We attempt to cluster the customers based on their spending habits.

2 K-Means Clustering

Our aim is to separate the data points into k clusters, $C_1, C_2, ..., C_k$ each of which holds data points which satisfy some metric of similarity amongst the other data points of the same set.

We begin by setting forth the two rules:

1. We make sure that each data point is in at least one of the clusters.($C_1 \cup C_2 \cup ... \cup C_k = 1, ..., n$).
2. No single data point should belong to more than one cluster. $C_k \cap C_{k'} = \emptyset$.

The idea is that for a *good* clustering the variation amongst the data points within the same cluster will be minimum. There are several metrics to measure these intracluster variations, but a popular one is **W**ithin **C**luster **S**um of **S**quares or **W.C.S.S**.

3 WCSS

Why calculate WCSS?

Often times the data points can be arranged in random and seemingly absurd form of distributions and we might have difficulty in determining the k value for the data through exploratory analysis. Then we can calculate the WCSS value for different no. of clusters and see where the value drastically decreases and forms a so-called "Elbow" in the plot WCSS vs k.

We see that if no. of data points = no. of clusters, then WCSS = 0, but if no. of clusters = 1, then WCSS becomes maximum. Thus, a value in between has to be selected using the above mentioned elbow method.

The sum of the squares of all the distances of the data points with respect to the centroid is calculated.

$$WCSS = \frac{1}{|C_k|} \sum_{k=0}^{K} \left(\sum_{n=0}^{N_k} dist(C_k, x_n^k)^2 \right)$$

where:

- K = Total no. of centroids.
- N_k = No. of data points in the cluster labelled k
- C_k = Centroid labelled k
- x_n^k = The n^{th} data point of the cluster labelled k

```
# This is for a single cluster, the same has to be done for all k
    clusters.
def wcss(cluster_x, cluster_y, centroid_x, centroid_y):
     '''Gets as Input Arrays holding the x and y positions of the
         specific cluster, and the associated centroid's x and y
        positions.'''
    wcss_value = 0
    for (px, py) in zip(cluster_x, cluster_y):
        wcss_value += euclidean_distance(centroid_x, centroid_y,
           px, py)
    return wcss_value
```

3.1 Limitation(s) of the Elbow Method

- It only shows the change in WCSS as we increase the no.of clusters. And this is a relative value, hence we might require some form of trial and error to get the k value right.
- We might have to visualize the cluster solutions and determine the suitable no. of clusters accordingly.

4 Algorithm

1. Randomly assign to each data point a cluster between 0 to k - 1 (zero indexing is efficient if Python is used, otherwise go from 1 to k). These serve as the initial cluster assignments for the observations.

2. For each of the k clusters, compute the cluster centroid.

3. Assign each data point or observation to the cluster whose centroid is closest (wherein their *Euclidean distance* is minimum.)

4.1 Euclidean Distance Function

The Euclidian distance between two points (x_1, y_1) and (x_2, y_2) in 2 Dimensional Space is given by:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

```
def euclidean_distance(centroid_x, centroid_y, data_point_x,
    data_point_y):
     return np.sqrt((centroid_x - data_point_x)**2 + (centroid_y -
         data_point_y)**2)
```

4.2 Centroid Calculation Function

The centroid of a group of points is the average of the position of all the data points in the cluster.

```
def centroid(cluster_x, cluster_y):
    centroid_point = np.array([0, 0])
    for (px, py) in zip(cluster_x, cluster_y):
        centroid_point[0] += px
        centroid_point[1] += py
    centroid_point = np.true_divide(centroid_point, len(cluster_x
       ))
    return centroid_point
```

5 Pros and Cons of K-Means

5.1 Pros

- Simple to Understand
- Fast to cluster

- Widely available, usable and scalable.
- Always yields a result, but not necessarily the correct one.

5.2 Cons

- We NEED to pick the no.of clusters beforehand.
- Very sensitive to the initialization of the centroids.
- Also sensitive to the outliers.
- Produces spherical solutions, as the distance metric used is Euclidian.
- Standardization is needed, other wise features with low range might be ignored during the clustering process.

5.3 Remedies

- To determine the no.of clusters, we can use the Elbow method.
- For the initialization problem - k-means++ algorithm is used to determine the initial position of the centroids. Another effective method would be to randomly associate k numbers to each data point, which will allow for non-localization of the centroids formed initially.
- Outliers can be neglected.

6 Types of Clustering

The relationship between features might be established through some form of regression, after one neglects the groups which are not part of the linear relationship.

There are broadly two types of clustering: **Flat** and **Hierarchical**. K-Means Clustering is a good example of Flat Clustering, as here one has to specify the number of clusters beforehand.

In Hierarchical Clustering, the no.of clusters are determined through computation. Hierarchical clustering can be further classified into Agglomerative(Bottom to Up approach) and Divisive(Top to Down approach)

6.1 Dendrogram

A dendrogram is a clustering method which employs the bottom up approach. It starts with N clusters . where N = no. of data points available, and further upon each iteration, clusters are formed with the nearby data points based on their common features. This continues until there is only one final cluster remaining. The process

when followed through, will resemble a tree like structure containing N leaves and subsequently lesser branches at each level and 1 root.

```
seaborn.clustermap(data = the_data_frame)
```

The no.of branches at the node just two or three nodes above the root, can give us an idea of the no.of clusters we can use, for visualizing the natural clusters formed.

7 Execution

7.1 Data set - Features

The data set contains the following features. (This data set was created only for learning purposes.)

- `CustomerID` - A unique number associated with each customer.
- `Gender` - Gender of the Customer
- `Age` - Age of the Customer
- `Annual Income` - The annual income of the individual (in units of thousand US Dollars)
- `Spending Score (1-100)` - Score assigned by the mall based on customer behavior and spending nature.

For our future purposes, we shall neglect the features `CustomerID` and `Gender`, and perform the clustering with the remaining three features.

7.2 Implementation through Scikit-Learn

The implementation of the clustering using the module `sklearn` is pretty straightforward. We additionally calculate the metric **W.C.S.S** for a number of different k values, to produce the so called "Elbow Diagram" and use it to determine the suitable value of k to be 5.

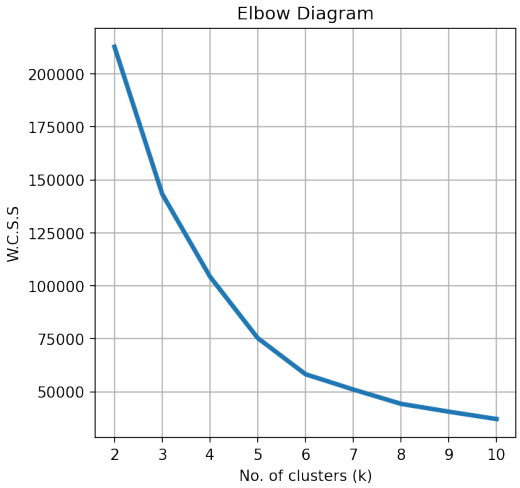


Figure 1: The Elbow Diagram for the Mall Customer Data

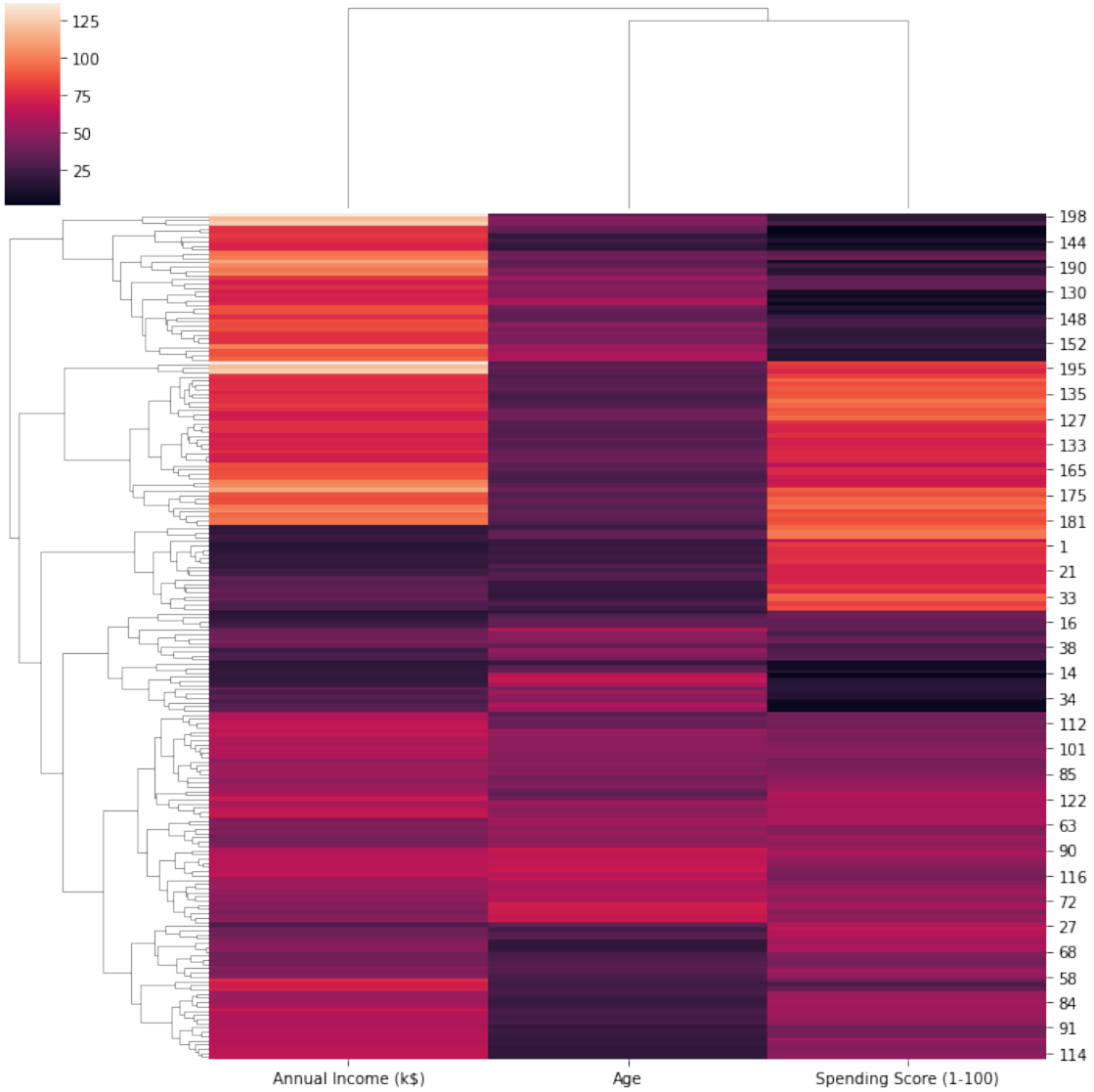
Next we implement the K-Means Clustering and obtain the following form of clustering. Here, we can see the subsequent forms or classes of customers(As the k = 5 was an entirely arbitrary value dependent on this particular data set, we shall give colloquial names to each cluster):

Figure 2: K-Means Clustering for k = 5 in the plot of Spending Score vs. Annual Income

- **Unhappy or Oblivious** The light blue section at the bottom left are people who have less amount of income to spend **and** they have low spending score. These people have low loyalty and expenditure.
- **Diligent** The light green section shows people have a high spending score and low prospect of expenditure. These people are loyal to the mall.

- **Normal** The dark blue center cluster shows people who have an average spending score and are in the centre of the distribution when it comes to annual income.
- **Drifters** The violet cluster at the bottom right shows people who have high prospect for expenditure but for some reason are not as loyal as we'd like.
- **Fans** These people are the ideal form of customers in the top right, who are most happy with the mall, and contribute consistently.

Below is the clustermap or **Dendrogram** of the dataset. One is to notice that, this diagram is somewhat comprehensible because of the low amount of features and data points. Higher amount would lead to entirely undecipherable diagrams.



8 Voronoi Diagrams

Voronoi Diagrams[3] are roughly put, a form of partition of a plane based on certain set of rules, usually based on the closest distance with a set of finite points.

For a metric space X, with an euclidean distance function d and a set of points P_m with m points, the partition or labelling of an individual point x in the space is obtained in the following way:

$$L_x = \{k | d(k, P_k) \leq d(k, P_j)\, for\, all\, j \neq k\}$$

The label k (the index value of the point P_k) is sufficient label for the point x.

Voronoi Diagrams in our case would allow us to visualize and analyse the boundaries or points of classification of our 5 clusters.

8.1 Algorithm

If one is to ignore the concepts of computation complexity and assume that x is a fairly small number, the algorithm is pretty straightforward, although there are many other efficient ways to implement the Voronoi Diagrams.

1. Pick a point in the space and calculate the euclidean distance between the point and each point in the set P_k.
2. Assign the index value or some similar property of the point P_k which has the minimum distance to the point x.
3. Repeat for all the points in the space with a reasonable amount of precision. Higher precision would lead to a lot of computation time.

The function used for the upcoming visualizations is the following:

```
def voronoi(cluster_x, cluster_y, no_of_points = 10000):
    color_in_f = []
    vx = []
    vy = []
    for i in range(no_of_points):
        px = np.random.random() * 140
        py = np.random.random() * 100
        vx.append(px)
        vy.append(py)
        distance_from_centroid = []
        for c_x, c_y in zip(cluster_x, cluster_y):
            distance_from_centroid.append(distance(px, py, c_x,
                c_y))
        color_in_f.append(np.argmin(distance_from_centroid))
    return vx, vy, color_in_f
```

The values 140 and 100 were chosen for a compact form of visualization and are again, entirely arbitrary and dependent on the data and the user's discretion.

The above function takes as input the x-coordinates and y-coordinates of the points P_m which in our case, would be the centroids of the clusters. It returns as a tuple the following three values:

- vx - The x-coordinates of all the points.
- vy - The y-coordinates of all the points.
- color_in_f - The color map containing the labels of the points.

The choice of using individual variables instead of vectorized inputs is conscious and done to keep the implementation as elementary as possible.

8.2 The Mall Customer Voronoi Diagram

9 From Scratch Implementation

For this, we shall use the modules - `numpy`, `matplotlib`, `pandas`. The algorithm used is the same as in Section 4. The frequently used functions are the `distance` function, `wcss` function, `voronoi` function as given above. Along with them the function `check_closest` is also used. This function takes as x-coordinates and the y-coordinates of the centroids and an individual point, and returns a single value

which we consider to be the label of point `(px, py)`. We use this function to create the array containing the labels of all the points.

```
def check_closest(C_X, C_Y, px, py):
    dist = []
    for c_x, c_y in zip(C_X, C_Y):
        dist.append(distance(px, py, c_x, c_y))
    return np.argmin(dist)
```

9.1 8 Steps Visualization

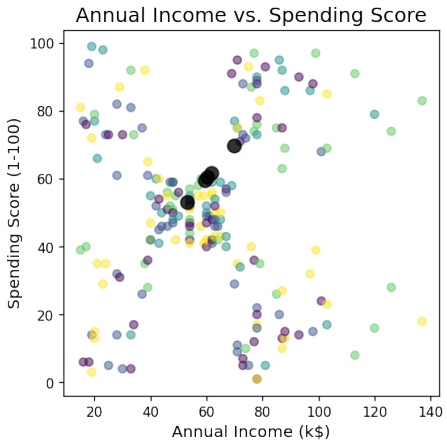


Figure 3: Step 1 - Random Assignment of Labels and Calculation of centroids for the random allocation

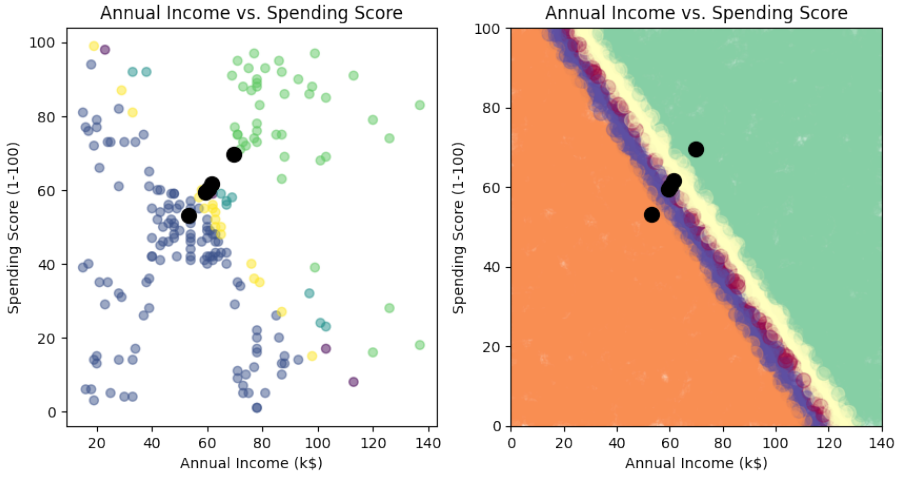


Figure 4: Step 2 - The centroids are re-calibrated and then the relevant Voronoi Diagram for the present position of the centroids.

The remaining steps are but repetitions of the above step.

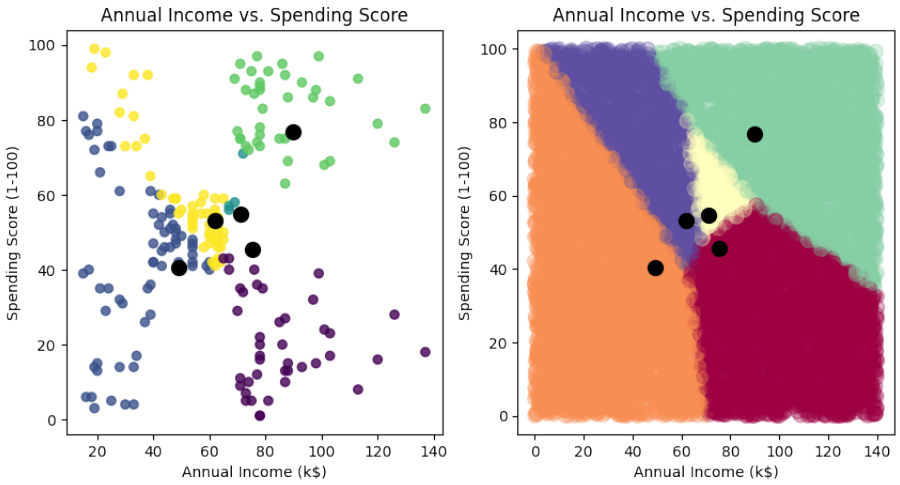


Figure 5: Iteration 3

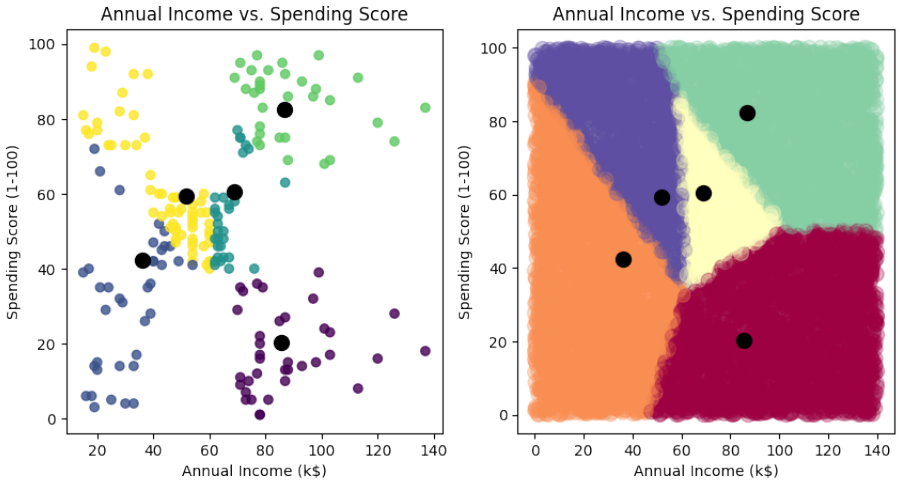


Figure 6: Iteration 4

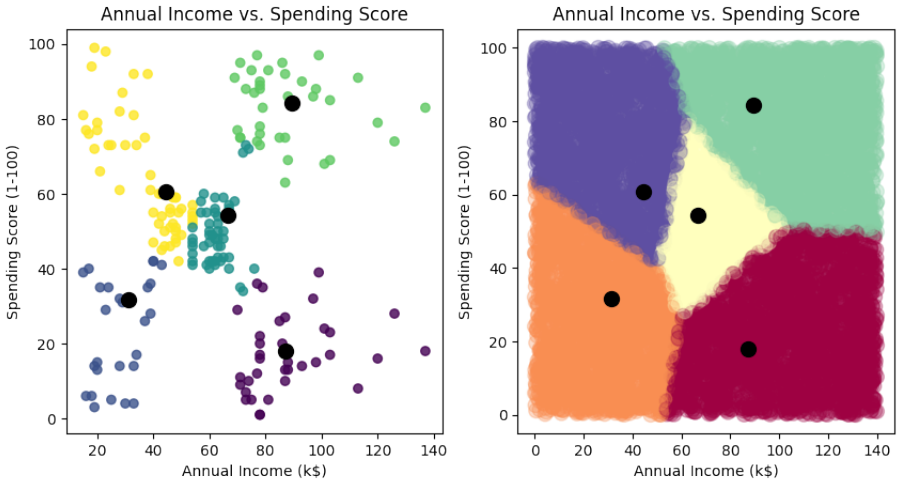


Figure 7: Iteration 5

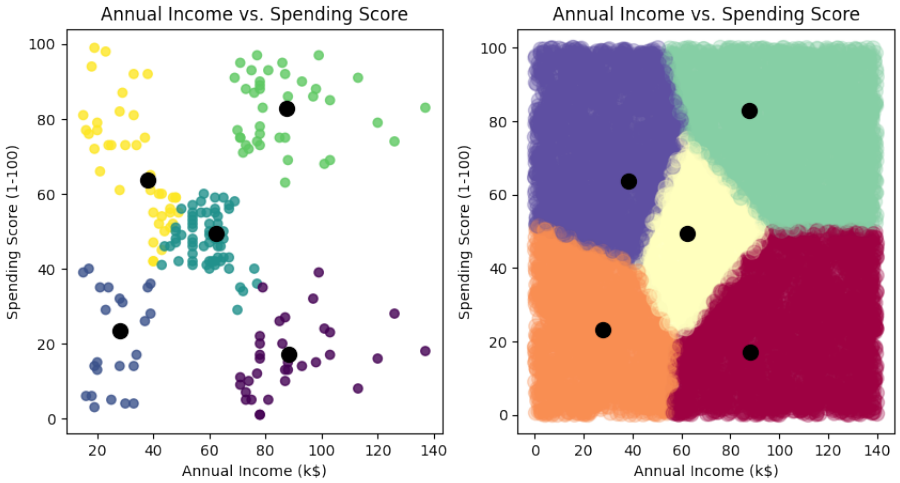


Figure 8: Iteration 6

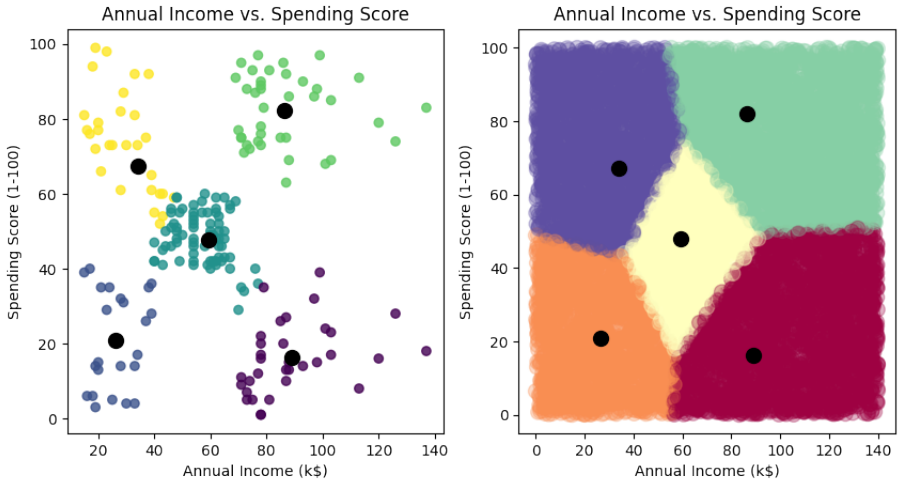


Figure 9: Iteration 7

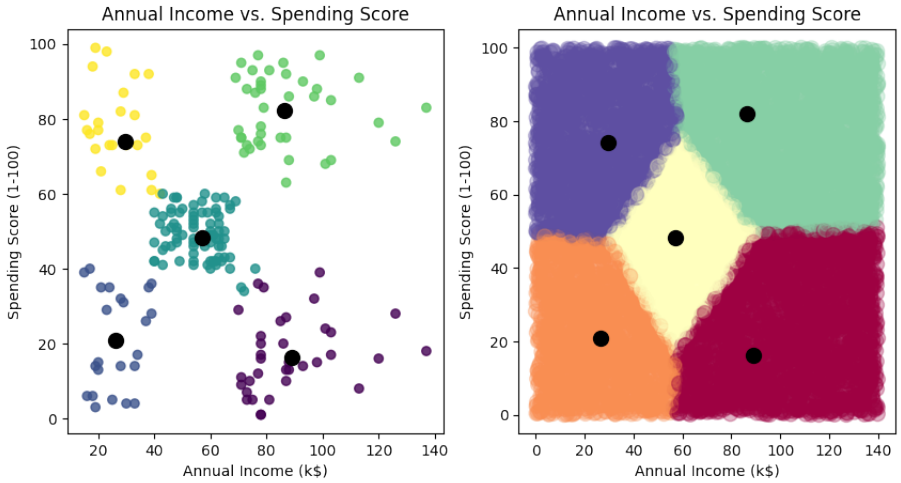


Figure 10: Iteration 8 - Final Step

We have iterated for exactly 8 steps and have received a decent plot comparable to the one we received from the `sklearn` implementation. This shows us that the K-Means Clustering is a rapidly converging form of clustering.

10 Conclusion

The K-Means Clustering method was implemented for the customer basket data set, allowing us to study the types of customers frequenting the mall. Further steps can be be taken to draw in the mid-range customers with discount vouchers or specialized deals.

The inherent beauty and necessity of the Voronoi Diagrams were studied, which showed us the boundaries of the type of customers, which again is food for further thought.

We implemented the K-Means Algorithm again from scratch and iterated for 8 steps and arrived at a reasonably decent clustering.

Reference

1. An Introduction to Statistical Learning with Applications in R.(James, Witten, Hastie, Tibshirani)

2. Mall Customer Segmentation Data. here

3. Voronoi Diagram Wikipedia