CS-614 Project Report

March 2021

Customer Segmentation

Submitted in partial fulfillment of the requirements

for the award of degree of

Bachelor of Technology

In

Computer Science and Engineering

Table 1: Submitted By

Roll No	$NameOfStudent$
39/CSE/18100/411	$UtsavRaj$
39/CSE/18105/416	$VivekKumarYadav$
39/CSE/18036/347	$KunalGupta$

Under the guidance of

Dr. Bhaskar Biswas

Department of Computer Science and Engineering

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY KALYANI

Kalyani, Nadia, West Bengal- 741235

Spring Semester 2021

Contents

Acknowledgement

Firstly, we would like to thank our mentor Dr. Bhaskar Biswas for giving us the opportunity to work in a project that we were looking forward to doing and then giving us the creative freedom to design and work on it how we wished to but simultaneously giving us his constructive criticism and advice on what problems we might face along the way specially his advice on how to plan out a timeline or what we could work on really helped us complete this project in a timely manner. This project would not be possible without his constant support throughout.

1 Introduction

1.1 The Basic Idea Of Unsupervised Learning

Unsupervised learning is a branch of machine learning that learns from test data that has not been labeled, classified or categorized. Instead of responding to feedback, unsupervised learning identifies commonalities in the data and reacts based on the presence or absence of such commonalities in each new piece of data.
The data given to unsupervised algorithm are not labelled, which means only the input variables, i.e. the data points

$$x^(i)$$

are given with no corresponding output variables. So the machine has to learn on its own to "separate data",without any training. This is the first and most important **difference** in comparison to supervised learning.

1.2 The Business Problem

Any company in retail, no matter the industry, ends up collecting, creating, and manipulating data over the course of their lifespan. These data are produced and recorded in a variety of contexts, most notably in the form of shipments, tickets, employee logs, and digital interactions. Each of these in- stances of data describes a small piece of how the company operates, for better or for worse. The more access to data that one has, the better the picture that the data can delineate.

Inorder to get optimize our commercial practices we need to answer: Why is product B purchased more on the first Saturday of every month compared to other weekends?, If a customer bought product B, will they like product C?, What are the defining traits of our customers? Can we predict what customers will want to buy? It is the later half of the last question that will be the broad focus of this paper.

1.3 Problem Statement

While mass marketing tactics are still able to get results, the assumption that simply everyone will be interested in buying what you are selling is a time-consuming, inefficient and expensive strategy.

Instead of a "one-size-fits-all" approach, successful segmentation clusters your customer data into groups sharing the same properties or behavioral characteristics, helping to drive dynamic content and personalization tactics for timelier, relevant and more effective marketing communications.However, for segmentation to be used correctly, it needs to take into account that different customers buy for different reasons, and marketers need to intelligently apply a number of considerations that could affect their purchasing decisions.

There are many reasons why companies fail at targeting the correct customers. It could be an assumption that segmentation is based entirely on demographics, or that segments have been defined too broadly. It could be that there is no strategic goal, like lead acquisition or customer retention. Or, it could be that the lack of a Single Customer View and siloed data has resulted in an inaccurate or misinformed understanding of your consumer base.

1.4 Criteria for Clustering

1. To expedite the clustering process, the new customer data needed to undergo minor data preprocessing. In this step, certain customers were pruned from the dataset if they did not meet certain self-imposed constraints.

2. The customer must have visited at least three times. This is to ensure that there is ample data collection and that time-based features, such as average time between visits, can be meaningful.

1.5 Cluster Analysis

Cluster analysis groups data objects based only on information found in the data that describes the objects and their relationships. The goal is that the objects within a group be similar (or related) to one another and dierent from (or unrelated to) the objects in other groups. The greater the similarity (or homogeneity) within a group and the greater the dierence between groups, the better or more distinct the clustering.
An entire collection of clusters is commonly referred to as a clustering, and in this section, we distinguish various types of clusterings: hierarchical (nested) versus partitional (unnested), exclusive versus overlapping versus fuzzy, and complete versus partial.

A **partitional** clustering is simply a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset. If we permit clusters to have subclusters, then we obtain a **hierarchical** clustering, which is a set of nested clusters that are organized as a tree. Each node (cluster) in the tree (except for the leaf nodes) is the union of its children (subclusters), and the root of the tree is the cluster containing all the objects. **Exclusive** clusters assign each object to a single cluster. There are many situations in which a point could reasonably be placed in more than one cluster, and these situations are better addressed by **non-exclusive** clustering. In the most general sense, an overlapping or non-exclusive clustering is used to reflect the fact that an object can simultaneously belong to more than one group (class). In a **fuzzy** clustering, every object belongs to every cluster with a membership weight that is between 0 (absolutely doesn't belong) and 1 (absolutely belongs). With such a clustering, we address situations where data points could belong to more clusters. Usually, the fuzzy clustering is combined with partitional clustering, by assigning data points to the clusters with the highest probability.

1.6 Target Audience Additional Information

The most common ways in which businesses segment their customer base are:

A.**Demographic information**, such as gender, age, familial and marital status, income, education, and occupation.

B.**Geographical information**, which differs depending on the scope of the company. For localized businesses, this info might pertain to specific towns or counties. For larger companies, it might mean a customer's city, state, or even country of residence.

C.**Psychographics**, such as social class, lifestyle, and personality traits.

D.**Behavioral data**, such as spending and consumption habits, product/service usage, and desired benefits.

1.7 Background

Different methods for customer segmentation :

1.7.1 Hierarchical Clustering

Hierarchical clustering techniques are a second important category of clustering methods. As with K-means, these approaches are relatively old compared to many clustering algorithms, but they still enjoy widespread use. There are two basic approaches for generating a hierarchical clustering:

1.7.2 A.Agglomerative

Start with the points as individual clusters and, at each step, merge the closest pair of clusters. This requires defining a notion of cluster proximity.

1.7.3 Basic Agglomerative Hierarchical Clustering Algorithm

1. Compute the proximity matrix, if necessary.

2. Repeat
3. Merge the closest two clusters.
4. Update the proximity matrix to reflect the proximity between the new cluster and the original clusters.
5. Until Only one cluster remains.

Lets say we have six data points A,B,C,D,E,F.

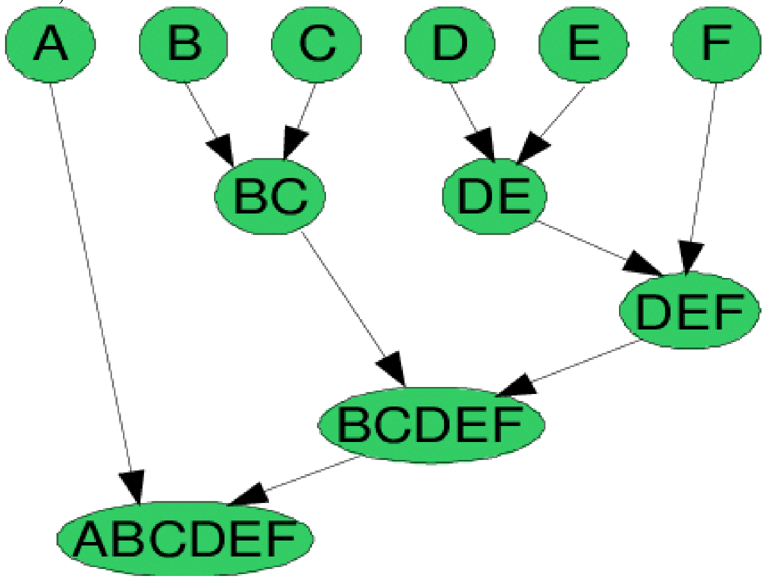
Step- 1: In the initial step, we calculate the proximity of individual points and consider all the six data points as individual clusters as shown in the image below.

Step- 2: In step two, similar clusters are merged together and formed as a single cluster. Let's consider B,C, and D,E are similar clusters that are merged in step two. Now, we are left with four clusters which are A, BC, DE, F.

Step- 3: We again calculate the proximity of new clusters and merge the similar clusters to form new clusters A, BC, DEF.

Step- 4: Calculate the proximity of the new clusters. The clusters DEF and BC are similar and merged together to form a new cluster. We're now left with two clusters A, BCDEF.

Step- 5: Finally, all the clusters are merged together and form a single cluster.
A hierarchical clustering is often displayed graphically using a tree-like diagram called a **dendrogram**, which displays both the cluster relationships and the order in which the clusters were merged (agglomerative view) or split (divisive view).



1.7.4 B.Divisive

Start with one, all-inclusive cluster and, at each step, split a cluster until only singleton clusters of individual points remain. In this case, we need to decide which cluster to split at each step and how to do the splitting.

Since the Divisive Hierarchical clustering Technique is not much used in the real world, we will give a brief of the Divisive Hierarchical clustering Technique.

In simple words, we can say that the Divisive Hierarchical clustering is exactly the opposite of the Agglomerative Hierarchical clustering. In Divisive Hierarchical clustering, we consider all the data points as a single cluster and in each iteration, we separate the data points from the cluster which are not similar. Each data point which is separated is considered as an individual cluster. In the end, we'll be left with n clusters.

As we are dividing the single clusters into n clusters, it is named as Divisive Hierarchical clustering.Divisive is just opposite of agglomerative clustering. So there is no need to draw image for divisive.

1.7.5 Space and Time Complexity

1.7.6 Space complexity

The space required for the Hierarchical clustering Technique is very high when the number of data points are high as we need to store the similarity matrix in the RAM. The space complexity is the order of the square of n. Space complexity = O(n2) where n is the number of data points.

1.7.7 Time complexity

Since we've to perform n iterations and in each iteration, we need to update the similarity matrix and restore the matrix, the time complexity is also very high. The time complexity is the order of the cube of n. points. Time complexity = O(n3) where n is the number of data points.

1.7.8 Limitatons

There is no mathematical objective for Hierarchical clustering.

All the approaches to calculate the similarity between clusters have its own disadvantages.

High space and time complexity for Hierarchical clustering. Hence this clustering algorithm cannot be used when we have huge data.

1.7.9 K-Means Clustering

K - means algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster

1.7.10 K-Means Algorithm Pseudocode

```
1. for k=1 to K do
2. randomly initialize mean for kth cluster
                    $\mu_k$
   = some random location
3. end for
4. repeat
5. for n = 1 to N do
6. E-step : assign n-th data point to closest center
                    $r_n k = argmin_k \|x_n - \mu_k\|^2$
7. end for
8. for k = 1 to K do
9. M-step : re-estimate mean of cluster k with 2.2
                    $\mu_k = MEAN_k(x_n, r_n k)$
10. end for
11. until converged
12. return r return cluster assignments
```

1.7.11 Advantages of K - Means

K-means clustering is relatively simple to implement, and can be implemented without using frameworks – just by using simple programming language.

The algorithm is known to easily adapt to new examples.

It guarantees convergence by trying to minimize the total SSE as an objective function over a number of iterations.

1.7.12 Complexity

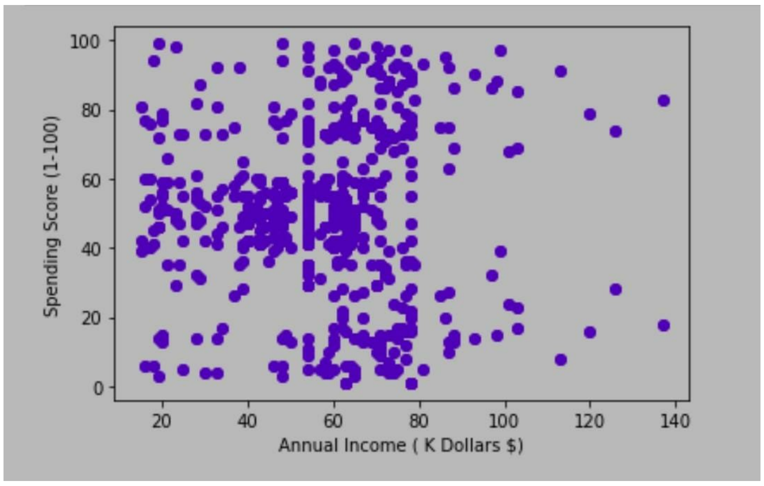
The algorithm is fast and efficient in terms of computational cost which is typically O(K*n*d).It depends on what you call k-means.

The problem of finding the global optimum of the K means objective function.

$$\min_{S} \sum_{i=1}^{K} \sum_{x_j \in S_i} \|x_i - \mu_k\|^2,$$

1.7.13 Analysis of Data

This is the scatterplot of customer dataset between Spending Score and Annual Income and you can see noise in the plot.And we have seen that it is difficult to find similarities if there is noise in the plot.So hierarchical clustering is not a good approach for customer segmentation.
Because of these reasons we are selecting K-means clustering for our project.

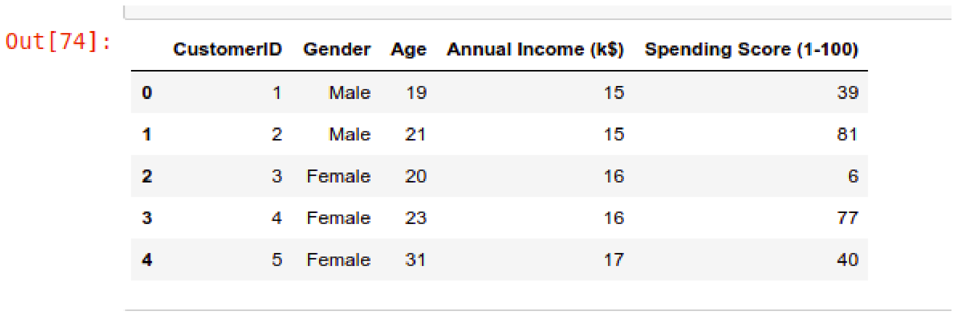


2 Overview

You are owning a supermarket mall and through membership cards, you have some basic data about your customers like Customer ID, age, gender, annual income and spending score. You want to understand the customers like who are the target customers so that the sense can be given to the marketing team and plan the strategy accordingly.
The premise of market segmentation is that to maximize sales to a large population of customers, it is best to divide it into logical subgroups. The assumption is that by dividing one large, amorphous mass into subgroups, you can fine-tune your product, messaging, support, or distribution channels to meet the specific needs of unique customer groups. Thus, the goal is to use a market segmentation model to improve marketing success and optimize marketing.
The way of implementation.

1. Specify number of clusters K.
2. Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.

This file contains the basic information (ID, age, gender, income, spending score) about the customers.


Out[74]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

2.1 Acquisition of Data

Finding readied, usable data for analysis in a business context is a rarity. As such, it is imperative to collect as much data as possible, but also in a format that meets a wide variety of financial, ethical, and computational considerations. But before discussing these, it is first important to describe the ways in which the relevant retail data are stored and utilized across the company.

- **CustomerID** :It contains all information regarding CustomerID on basis of which we are classifying and associated customer ID are relevant for this particular analysis.
- **Gender and Age** :It contains all information regarding gender either male of female those are relevant for this particular analysis.

- **Annual Income (k Dollars)** :It contains all information regarding income of customer annually weather they are profitable of not.
- **Spending Score (1-100)** :It contains all information regarding spending habbits of cutomer.

2.2 Brief Overview of Code

The entirety of the code written for this project was in Python. The following Python packages played pivotal roles in the execution and development of this project:

- pandas, numpy, sklearn.preprocessing, os, datetime, and time were all used for data collection, handling, and manipulation
- seaborn, matplotlib.pyplot, and scipy.cluster.hierarchy were used to create visualizations of data
- sklearn.cluster and sklearn.decomposition were used for clustering the data or decomposing it into three dimensions for plotting.

2.3 Algorithm with steps

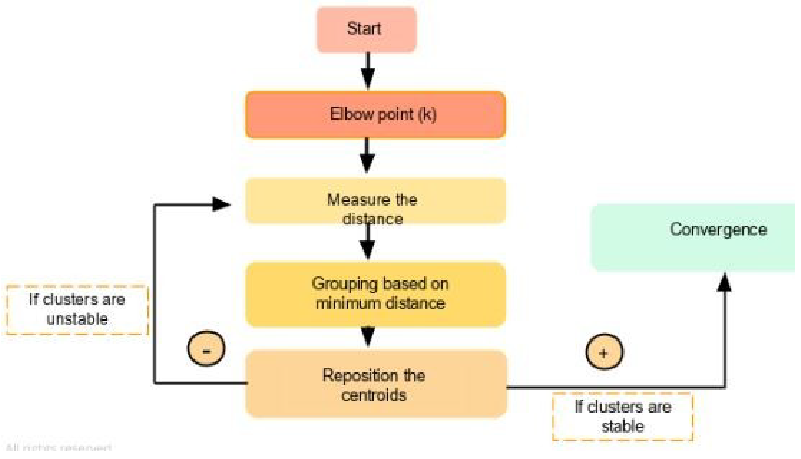
K-Means clustering is one of the most effective clustering methods out there in the market.

This method has the following steps:

1. Specify number of clusters K.
2. Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.
    - Compute the sum of the squared distance between data points and all centroids.
    - Assign each data point to the closest cluster (centroid).
    - Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

2.3.1 Flow of the algorithm

Pictorial representation of flow of K means clustering algorithm.



3 How K-means algorithm Works

3.1 Experimental Setup

Visualization of Data

```
In [1]: #import libraries

        import pandas as pd
        import numpy as np
        import random as rd
        import matplotlib.pyplot as plt
        import seaborn as sns
        from pandas import Series,DataFrame
```

```
In [2]: dframe = pd.read_csv("Project2_CustomerData.csv")
```

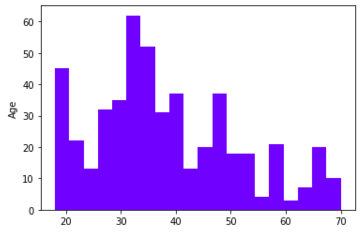
```
In [3]: dframe.head()
Out[3]:
```

	CustomerID	Age	Annual Income	Spending Score	Net Quatity	Returned Item Quantity	Purchased Item Quantity	Gender
0	1	19	15	39	2	0	2	Male
1	2	21	15	81	2	0	2	Male
2	3	20	16	6	0	-2	2	Female
3	4	23	16	77	1	0	1	Female
4	5	31	17	40	1	0	1	Female

3.1.1 Histogram Plots For

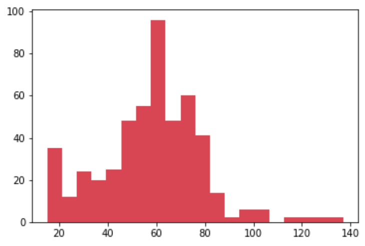
Age

```
In [28]: plt.hist(dframe['Age'],color = 'blue',bins = 20)
         plt.xlabel("Age")

Out[28]: Text(0, 0.5, 'Age')
```



Annual Income

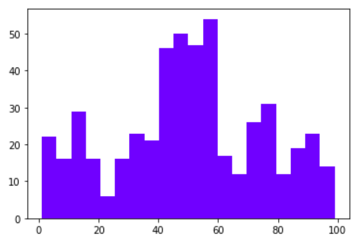
```
n [5]: plt.hist(dframe['Annual Income'],color = 'indianred',bins = 20)

ut[5]: (array([35., 12., 24., 20., 25., 48., 55., 96., 48., 60., 41., 14.,  2.,
                6.,  6.,  0.,  2.,  2.,  2.,  2.]),
        array([ 15. ,  21.1,  27.2,  33.3,  39.4,  45.5,  51.6,  57.7,  63.8,
                69.9,  76. ,  82.1,  88.2,  94.3, 100.4, 106.5, 112.6, 118.7,
               124.8, 130.9, 137. ]),
        <a list of 20 Patch objects>)
```



Spending Score

```
In [6]: plt.hist(dframe['Spending Score'],color = 'blue',bins = 20)
```

```
Out[6]: (array([22., 16., 29., 16.,  6., 16., 23., 21., 46., 50., 47., 54., 17.,
                12., 26., 31., 12., 19., 23., 14.]),
         array([ 1. ,  5.9, 10.8, 15.7, 20.6, 25.5, 30.4, 35.3, 40.2, 45.1, 50. ,
                54.9, 59.8, 64.7, 69.6, 74.5, 79.4, 84.3, 89.2, 94.1, 99. ]),
         <a list of 20 Patch objects>)
```

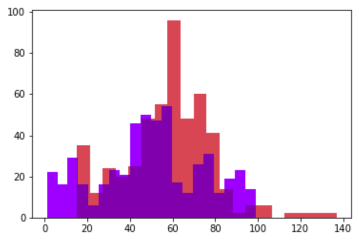


3.1.2 Combined Histogram Plot For Different Attributes

Annual Income and Spending Score

```
In [8]: plt.hist(dframe['Annual Income'],color = 'indianred',bins = 20)
        plt.hist(dframe['Spending Score'],color = 'blue',bins = 20,alpha = 0.5)
```

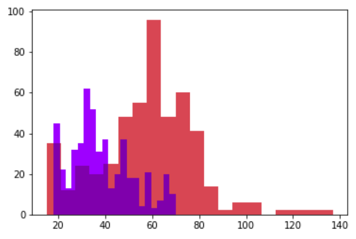
```
Out[8]: (array([22., 16., 29., 16.,  6., 16., 23., 21., 46., 50., 47., 54., 17.,
                12., 26., 31., 12., 19., 23., 14.]),
         array([ 1. ,  5.9, 10.8, 15.7, 20.6, 25.5, 30.4, 35.3, 40.2, 45.1, 50. ,
                54.9, 59.8, 64.7, 69.6, 74.5, 79.4, 84.3, 89.2, 94.1, 99. ]),
         <a list of 20 Patch objects>)
```



Annual Income and Age

```
In [24]: plt.hist(dframe['Annual Income'],color = 'indianred',bins = 20)
         plt.hist(dframe['Age'],color = 'blue',bins = 20,alpha = 0.5)
```

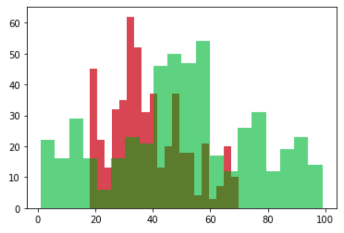
```
Out[24]: (array([45., 22., 13., 32., 35., 62., 52., 31., 37., 13., 20., 37., 18.,
                 18.,  4., 21.,  3.,  7., 20., 10.]),
          array([18. , 20.6, 23.2, 25.8, 28.4, 31. , 33.6, 36.2, 38.8, 41.4, 44. ,
                 46.6, 49.2, 51.8, 54.4, 57. , 59.6, 62.2, 64.8, 67.4, 70. ]),
          <a list of 20 Patch objects>)
```



Age and Spending Score

```
In [23]: plt.hist(dframe['Age'],color = 'indianred',bins = 20)
         plt.hist(dframe['Spending Score'],color = 'green',bins = 20,alpha = 0.5)
```

```
Out[23]: (array([22., 16., 29., 16.,  6., 16., 23., 21., 46., 50., 47., 54., 17.,
                 12., 26., 31., 12., 19., 23., 14.]),
          array([ 1. ,  5.9, 10.8, 15.7, 20.6, 25.5, 30.4, 35.3, 40.2, 45.1, 50. ,
                 54.9, 59.8, 64.7, 69.6, 74.5, 79.4, 84.3, 89.2, 94.1, 99. ]),
          <a list of 20 Patch objects>)
```

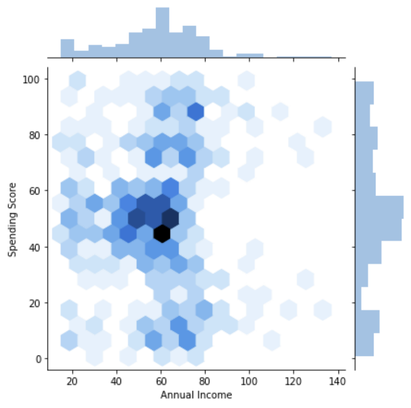


3.1.3 Joint HexPlots

Joint Hexplot for Annual Income and Spending Score

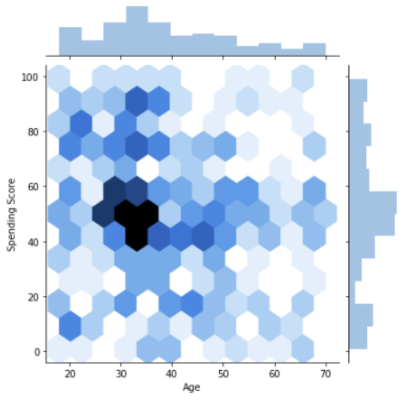
```
In [9]: sns.jointplot(dframe['Annual Income'],dframe['Spending Score'],kind = 'hex')
```

```
Out[9]: <seaborn.axisgrid.JointGrid at 0x7f6a72990a30>
```



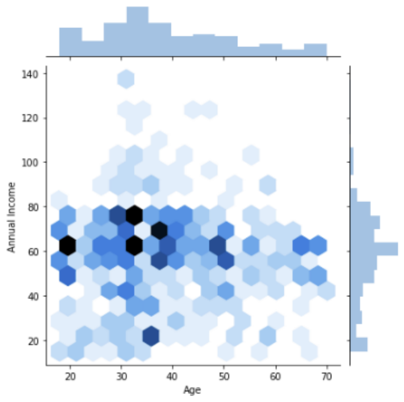
Joint Hexplot for Age and Spending Score

```
In [43]: sns.jointplot(dframe['Age'],dframe['Spending Score'],kind = 'hex')
Out[43]: <seaborn.axisgrid.JointGrid at 0x7f40d7c097c0>
```



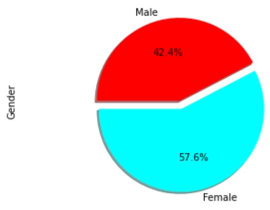
Joint Hexplot for Annual Income and Age

```
In [42]: sns.jointplot(dframe['Age'],dframe['Annual Income'],kind = 'hex')
Out[42]: <seaborn.axisgrid.JointGrid at 0x7f40d7a54460>
```



Pie Chart for number of males and females in dataset

```
In [12]: colors = ['cyan','red']
         explode = (0.1,0)
         plt.axis('equal')
         dframe['Gender'].value_counts().plot.pie(colors = colors , explode = explode ,autopct = '%1.1f%%' , startangle = 18
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6a728a55e0>
```

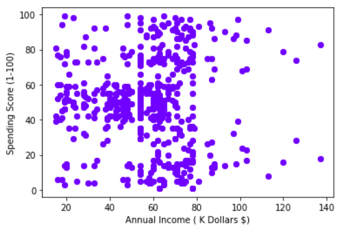


3.1.4 Scatter Plot of Annual Income,Spending Score and Age

Scatter plot can be plotted using two attributes at a time to show how points are distributed.So there are 3 attributes and ,we have to select 2 at a time we can do this by 3C2. We will use Scatter Plot of Annual Income and Spending Score for training and apply K means on these two attributes .Then we will see what is the result using the other two combination. This will be kind of train and test which is for Supervised Learning.

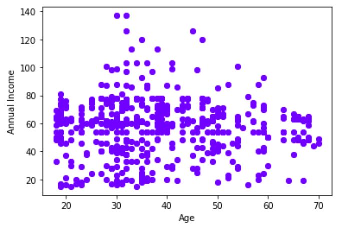
Spending Score and Annual Income

```
In [15]: X = dframe[['Annual Income','Spending Score']]
         #Visualise data points
         plt.scatter(X['Annual Income'],X['Spending Score'],c='blue')
         plt.xlabel('Annual Income ( K Dollars $)')
         plt.ylabel('Spending Score (1-100)')
         plt.show()
```



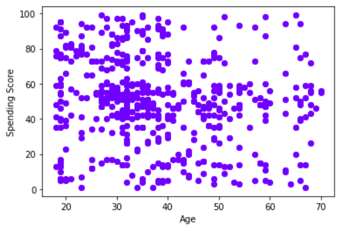
Age and Annual Income

```
In [15]: #Visualise data points
         X = dframe[['Age','Annual Income']]
         plt.scatter(X['Age'],X['Annual Income'],c='blue')
         plt.xlabel('Age')
         plt.ylabel('Annual Income')
         plt.show()
```

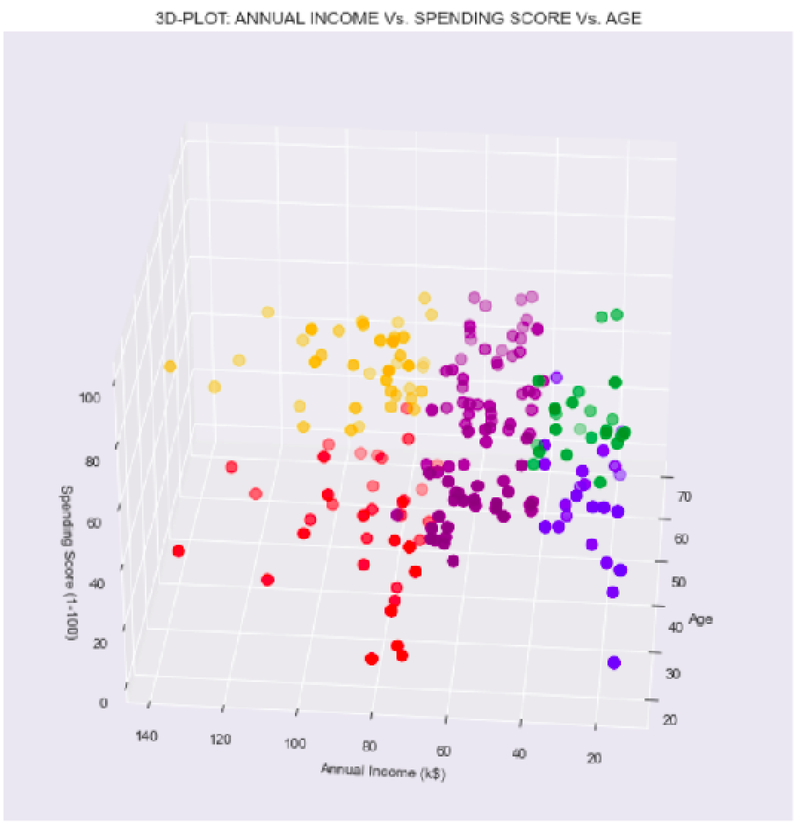


Spending Score and Age

```
In [18]: #Visualise data points
         X = dframe[['Age','Spending Score']]
         plt.scatter(X['Age'],X['Spending Score'],c='blue')
         plt.xlabel('Age')
         plt.ylabel('Spending Score')
         plt.show()
```



3 D Plot between Annual Income , Spending Score and Age



Statistics of data using Standard Scaler and for that we have to remove Gender column

```
In [21]: from sklearn.cluster import KMeans
```

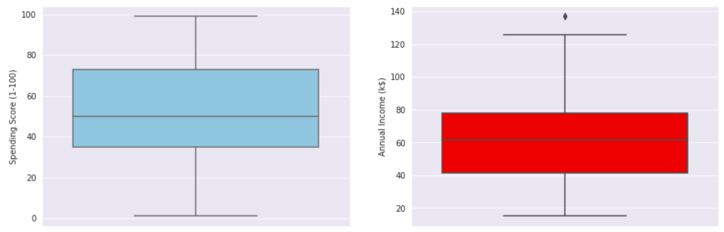
```
In [22]: # statistics of the data
         dframe.describe()
```

Out[22]:

	CustomerID	Age	Annual Income	Spending Score	Net Quatity	Returned Item Quantity	Purchased Item Quantity
count	500.000000	500.000000	500.00000	500.00000	500.000000	500.000000	500.000000
mean	250.500000	38.380000	57.77200	50.10800	1.226000	-0.016000	1.242000
std	144.481833	13.114098	20.59272	24.88604	0.481027	0.140654	0.460282
min	1.000000	18.000000	15.00000	1.00000	0.000000	-2.000000	1.000000
25%	125.750000	29.000000	46.00000	35.00000	1.000000	0.000000	1.000000
50%	250.500000	36.000000	60.00000	50.00000	1.000000	0.000000	1.000000
75%	375.250000	47.000000	71.00000	69.50000	1.000000	0.000000	1.000000
max	500.000000	70.000000	137.00000	99.00000	4.000000	0.000000	4.000000

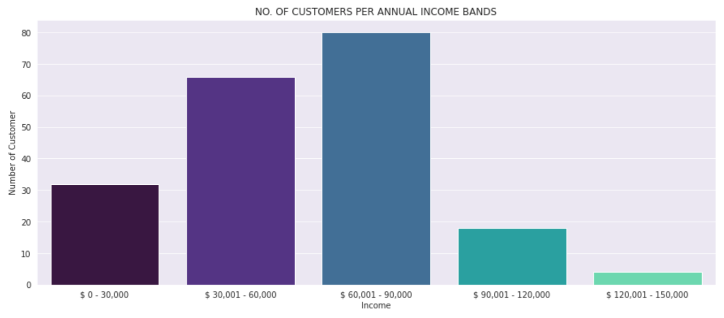
Box Plot Of Annual Income and Spending Score

```
In [76]: #box plot of spending score and annual income to better visualize the distribution range

         #It gives idea about outliers

         #Boxplot of of spending Score
         plt.figure(figsize=(15,5))
         plt.subplot(1,2,1)
         sns.boxplot(y=df["Spending Score (1-100)"], color="skyblue")

         #Boxplot of Annual income

         plt.subplot(1,2,2)
         sns.boxplot(y=df["Annual Income (k$)"],color ="red")
         plt.show()
```



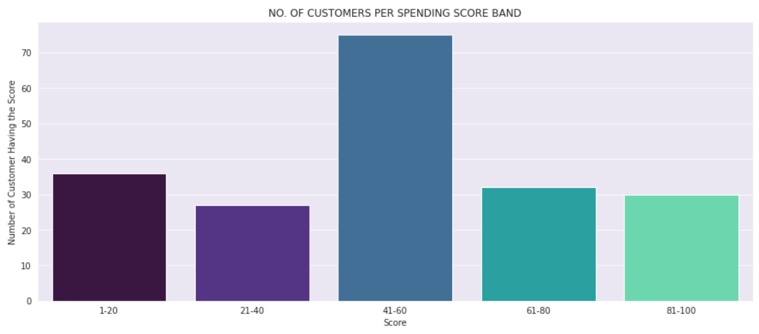
Plot of Annual Income into five consecutive bands

```
In [72]: # visualize the number of customers according to their annual income

         ai0_30 = data["Annual Income (k$)"][(data["Annual Income (k$)"] >= 0) & (data["Annual Income (k$)"] <= 30)]
         ai31_60 = data["Annual Income (k$)"][(data["Annual Income (k$)"] >= 31) & (data["Annual Income (k$)"] <= 60)]
         ai61_90 = data["Annual Income (k$)"][(data["Annual Income (k$)"] >= 61) & (data["Annual Income (k$)"] <= 90)]
         ai91_120 = data["Annual Income (k$)"][(data["Annual Income (k$)"] >= 91) & (data["Annual Income (k$)"] <= 120)]
         ai121_150 = data["Annual Income (k$)"][(data["Annual Income (k$)"] >= 121) & (data["Annual Income (k$)"] <= 150)]

         aix = ["$ 0 - 30,000", "$ 30,001 - 60,000", "$ 60,001 - 90,000", "$ 90,001 - 120,000", "$ 120,001 - 150,000"]
         aiy = [len(ai0_30.values), len(ai31_60.values), len(ai61_90.values), len(ai91_120.values), len(ai121_150.values)]

         plt.figure(figsize=(15,6))
         sns.barplot(x=aix, y=aiy, palette="mako")
         plt.title("NO. OF CUSTOMERS PER ANNUAL INCOME BANDS")
         plt.xlabel("Income")
         plt.ylabel("Number of Customer")
         plt.show()
```



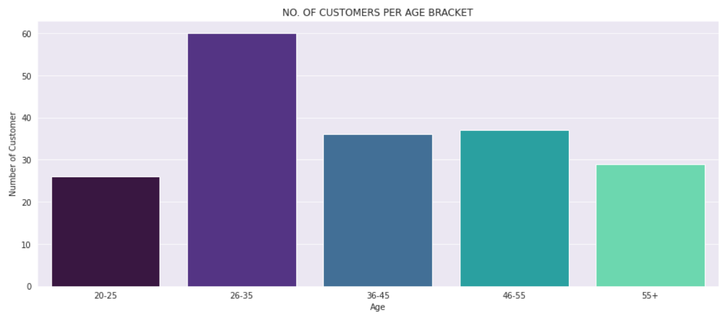
Plot of Spending Score

```
spending_score1_20 = data["Spending Score (1-100)"][(data["Spending Score (1-100)"] >= 1) & (data["Spending Score (

spending_score21_40 = data["Spending Score (1-100)"][(data["Spending Score (1-100)"] >= 21) & (data["Spending Score

spending_score41_60 = data["Spending Score (1-100)"][(data["Spending Score (1-100)"] >= 41) & (data["Spending Score

spending_score61_80 = data["Spending Score (1-100)"][(data["Spending Score (1-100)"] >= 61) & (data["Spending Score

spending_score81_100 = data["Spending Score (1-100)"][(data["Spending Score (1-100)"] >= 81) & (data["Spending Scor

#Values over X and Y axis
spending_score_x = ["1-20", "21-40", "41-60", "61-80", "81-100"]
spending_score_y = [len(spending_score1_20.values), len(spending_score21_40.values), len(spending_score41_60.values

#ploting the data based on Score and NO. OF CUSTOMERS PER SPENDING SCORE
plt.figure(figsize=(15,6))

#importing seaborn libraries to plot
sns.barplot(x=spending_score_x, y=spending_score_y, palette="mako")
plt.title("NO. OF CUSTOMERS PER SPENDING SCORE BAND")
plt.xlabel("Score")
plt.ylabel("Number of Customer Having the Score")
plt.show()
```



Plot of Age per bracket

```
In [40]: # check the distribution of number of customers in each age group
         #Taking age between 18 to 25
         age18_25 = df.Age[(df.Age <= 25) & (df.Age >= 20)]
         #Taking age between 26 to 35
         age26_35 = df.Age[(df.Age <= 35) & (df.Age >= 26)]
         #Taking age between 36 to 45
         age36_45 = df.Age[(df.Age <= 45) & (df.Age >= 36)]
         #Taking age between 46 to 55
         age46_55 = df.Age[(df.Age <= 55) & (df.Age >= 46)]
         #Taking age greater than 56
         age55above = df.Age[df.Age >= 56]

         #label on x axis
         x = ["20-25","26-35","36-45","46-55","55+"]
         #label on y axis
         y = [len(age18_25.values),len(age26_35.values),len(age36_45.values),len(age46_55.values),len(age55above.values)]

         plt.figure(figsize=(15,6))
         sns.barplot(x=x, y=y, palette="mako")
         plt.title("NO. OF CUSTOMERS PER AGE BRACKET")
         plt.xlabel("Age")
         plt.ylabel("Number of Customer")
         plt.show()
```



3.2 Specify number of clusters K.

Honestly, we can have any number of clusters.The maximum possible number of clusters will be equal to the number of observations in the dataset. But then how can we decide the optimum number of clusters? One thing we can do is plot a graph, also known as an elbow curve, where the x-axis will represent the number of clusters and the y-axis will be an evaluation metric(Inertia).

3.2.1 Elbow Method

Inertia tells us how far the points within a cluster are. So, inertia actually calculates the sum of distances of all the points within a cluster from the centroid of that cluster. We calculate this for all the clusters and the final inertial value is the sum of all these distances. This distance within the clusters is known as intracluster distance. So, inertia gives us the sum of intracluster distances:

- There's a popular method known as elbow method,which is used to determine the optimal value of k to perform clustering.
- The basic idea behind this method is that it plots the various values of cost with changing k.
- The point where this distortion declines the most is the elbow point, which works as an optimal value of k.
- Determining the number of Optimal Clusters Using The Elbow Method.
- Goal - maximize number of clusters and limiting case each data point becoming its own cluster centroid.
- Plot Within Cluster Sum Of Squares (WCSS) against number of clusters (K Value).
- WCSS measures sum of distances of observations from their cluster centroids.
- Calculate the Within Cluster Sum of Squared Errors (WSS) for different values of k, and choose the k for which WSS first starts to diminish.

The cluster value where this decrease in **inertia** value becomes constant can be chosen as the right cluster value for our data.

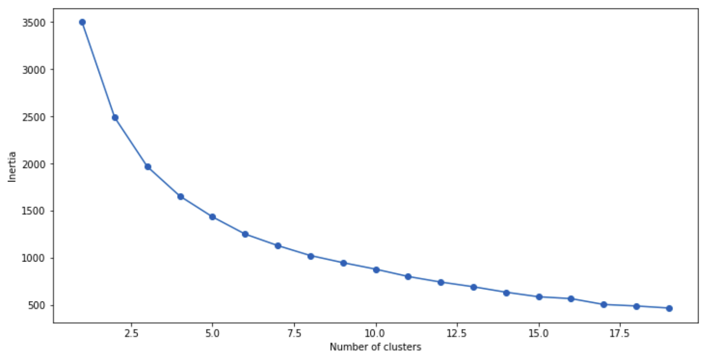
```
In [26]: # defining the kmeans function with initialization as k-means++
         kmeans = KMeans(n_clusters=2, init='k-means++')

         # fitting the k means algorithm on scaled data
         kmeans.fit(dframe)

Out[26]: KMeans(n_clusters=2)
```

```
In [27]: # inertia on the fitted data
         kmeans.inertia_

Out[27]: 3209367.6400000006
```

```
In [28]: # fitting multiple k-means algorithms and storing the values in an empty list
         SSE = []
         for cluster in range(1,20):
             kmeans = KMeans(n_jobs = -1, n_clusters = cluster, init='k-means++')
             kmeans.fit(data_scaled)
             SSE.append(kmeans.inertia_)

         # converting the results into a dataframe and plotting them
         frame = pd.DataFrame({'Cluster':range(1,20), 'SSE':SSE})
         plt.figure(figsize=(12,6))
         plt.plot(frame['Cluster'], frame['SSE'], marker='o')
         plt.xlabel('Number of clusters')
         plt.ylabel('Inertia')
```

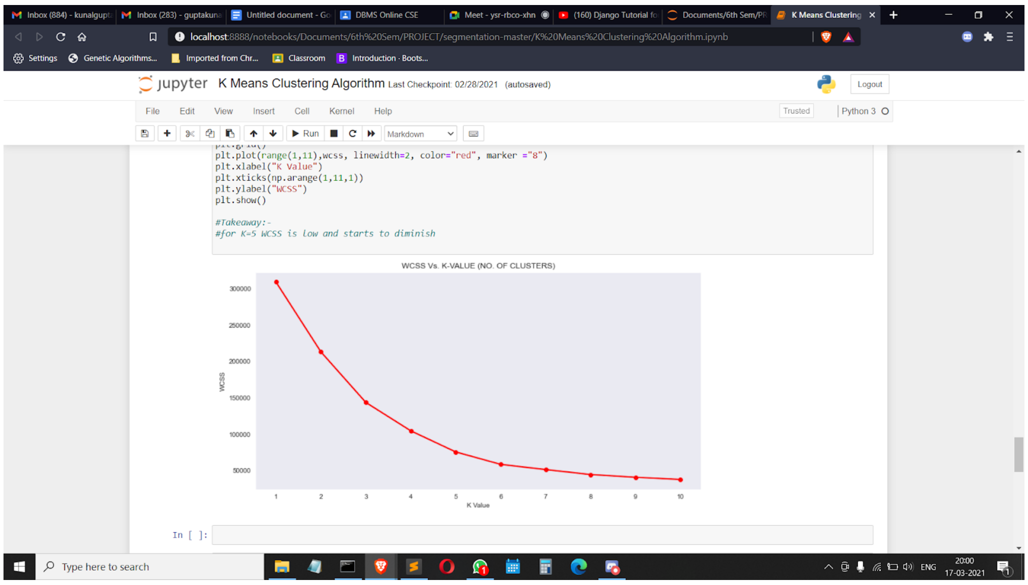
Here, we can choose any number of clusters between 5 and 10. We can have 6, 7, 8, or even 9 clusters. We must also look at the computation cost while deciding the number of clusters. If we increase the number of clusters, the computation cost will also increase. So, if we do not have high computational resources, we will go with lesser number of clusters.Let us take **K = 6**.

```
Out[28]: Text(0, 0.5, 'Inertia')
```



```
In [29]: # k means using 5 clusters and k-means++ initialization
         kmeans = KMeans(n_jobs = -1, n_clusters = 6, init='k-means++')
         kmeans.fit(data_scaled)
         pred = kmeans.predict(data_scaled)

         /home/basilisk/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:792: FutureWarning: 'n_jobs' was d
         eprecated in version 0.23 and will be removed in 1.0 (renaming of 0.25).
           warnings.warn("'n_jobs' was deprecated in version 0.23 and will be"

In [30]: frame = pd.DataFrame(data_scaled)
         frame['cluster'] = pred
         frame['cluster'].value_counts()

Out[30]: 0    125
         1    115
         2    114
         4     80
         5     59
         3      7
         Name: cluster, dtype: int64
```

Optimal Number of Clusters and Number of points in each cluster



```
plt.plot(range(1,11),wcss, linewidth=2, color="red", marker ="8")
plt.xlabel("K Value")
plt.xticks(np.arange(1,11,1))
plt.ylabel("WCSS")
plt.show()

#Takeaway:-
#for K=5 WCSS is low and starts to diminish
```

jupyter K Means Clustering Algorithm Last Checkpoint: 02/28/2021 (autosaved)
File Edit View Insert Cell Kernel Help
Trusted
Python 3
Run
Markdown
WCSS Vs. K-VALUE (NO. OF CLUSTERS)
WCSS
K Value
In [ ]:

Categorizing the data points into number of Optimal Clusters using K Means Clustering

3.3 Selecting K = 6 data points for the centroids and Iterating until there is no change to the centroids

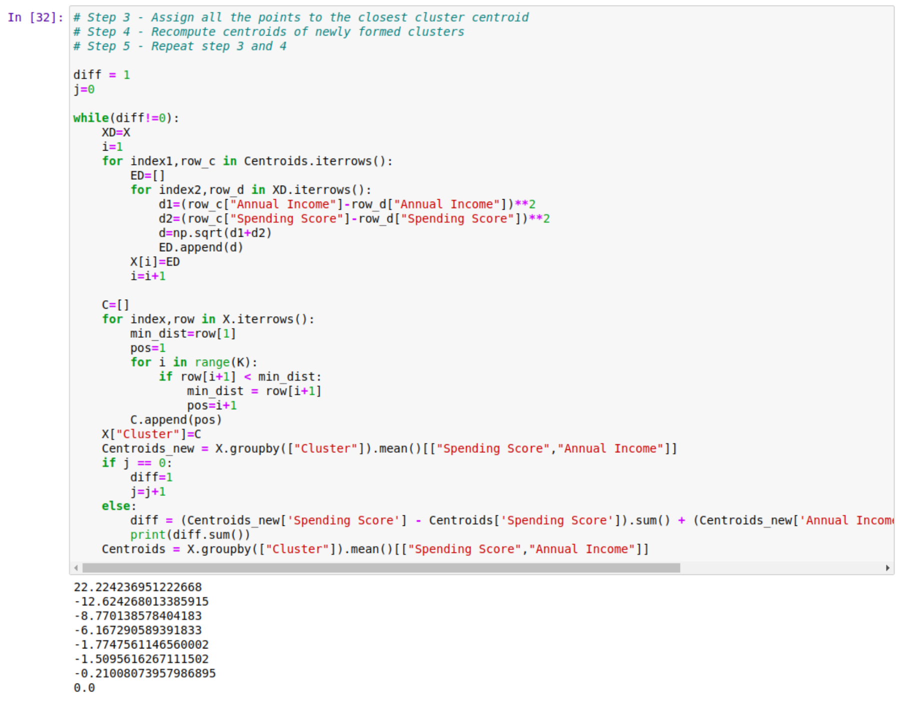
Let's say we have x1, x2, x3......... x(n) as our inputs, and we want to split this into K clusters.

Step 1 : Choose K random points as cluster centers called Centroids.

Step 2 : Assign each x(i) to the closest cluster by implementing euclidean distance (i.e., calculating its distance to each centroid)

Step 3 : Identify new centroids by taking the average of the assigned points.

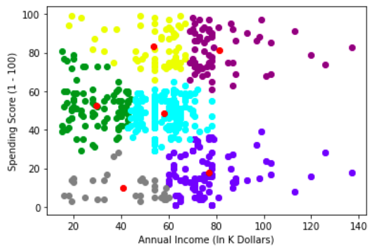
Step 4 : Keep repeating step 2 and step 3 until convergence is Achieved



```
In [32]: # Step 3 - Assign all the points to the closest cluster centroid
         # Step 4 - Recompute centroids of newly formed clusters
         # Step 5 - Repeat step 3 and 4

         diff = 1
         j=0

         while(diff!=0):
             XD=X
             i=1
             for index1,row_c in Centroids.iterrows():
                 ED=[]
                 for index2,row_d in XD.iterrows():
                     d1=(row_c["Annual Income"]-row_d["Annual Income"])**2
                     d2=(row_c["Spending Score"]-row_d["Spending Score"])**2
                     d=np.sqrt(d1+d2)
                     ED.append(d)
                 X[i]=ED
                 i=i+1

             C=[]
             for index,row in X.iterrows():
                 min_dist=row[1]
                 pos=1
                 for i in range(K):
                     if row[i+1] < min_dist:
                         min_dist = row[i+1]
                         pos=i+1
                 C.append(pos)
             X["Cluster"]=C
             Centroids_new = X.groupby(["Cluster"]).mean()[["Spending Score","Annual Income"]]
             if j == 0:
                 diff=1
                 j=j+1
             else:
                 diff = (Centroids_new['Spending Score'] - Centroids['Spending Score']).sum() + (Centroids_new['Annual Incom
                 print(diff.sum())
             Centroids = X.groupby(["Cluster"]).mean()[["Spending Score","Annual Income"]]
```

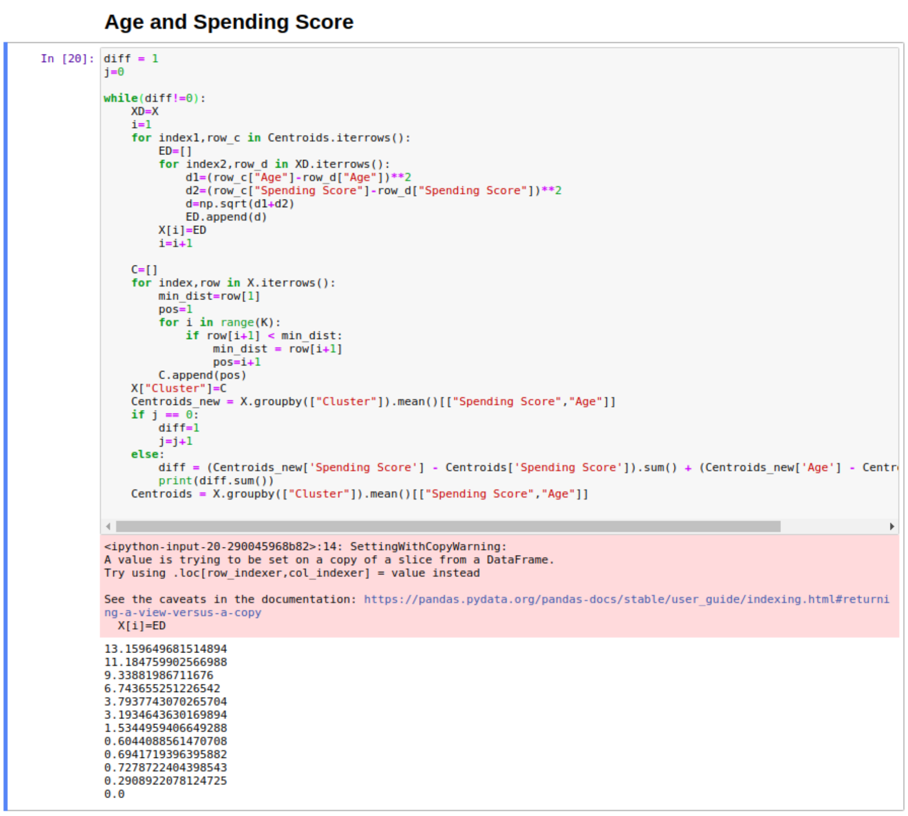
```
22.224236951222668
-12.624268013385915
-8.770138578404183
-6.167290589391833
-1.7747561146560002
-1.5095616267111502
-0.21008073957986895
0.0
```

```
In [33]: color=['blue','green','cyan','yellow','purple','grey']
         for k in range(K):
             data=X[X["Cluster"]==k+1]
             plt.scatter(data["Annual Income"],data["Spending Score"],c=color[k])
         plt.scatter(Centroids["Annual Income"],Centroids["Spending Score"],c='red')
         plt.xlabel('Annual Income (In K Dollars)')
         plt.ylabel('Spending Score (1 - 100)')
         plt.show()
```



Optimal Clusters == 6 The output image is clearly showing the six different clusters with different colors. The clusters are formed between two parameters of the dataset; Annual income of customer and Spending. We can also observe some points from the above pattern. The customers with average salary and average spending so we can categorize these customers as **descent**. The customer has a high income but low spending, so we can categorize them as careful..The low income and also low spending so they can be categorized as **sensible**. The customers with low income with very high spending so they can be categorized as **careless**. The customers with high income and high spending so they can be categorized as target, and these customers can be the **most profitable customers** for the mall owner.

As we mentioned earlier we will try to verify clusters formed by Annual Income and Spending Score with clusters formed by other two combination of attributes (Age and Spending Score) and (Age and Annual Income).
Now we will see the result when clusters are formed between paramters Age and Spending Score of the DataSet. Clearly the purple cluster has the highest monetary value and the grey cluster has the lowest. This is because of the product of spending score and Annual Income. The purple cluster has Annual income greater than 70K dollars and the spending score greater than 60. The grey cluster has Annual income less than 60K dollars and the spending score is less than 40. This implies that the customers in the purple cluster are of great importance to the company's profit and should be given highest priority in terms of decision making and money spent on marketing.

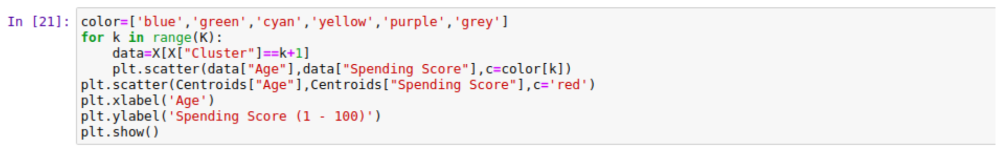


Age and Spending Score

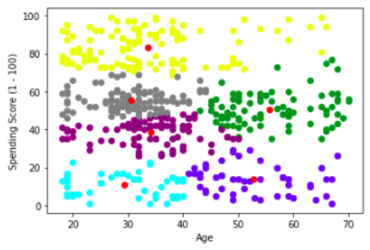
```
In [20]: diff = 1
         j=0

         while(diff!=0):
             XD=X
             i=1
             for index1,row_c in Centroids.iterrows():
                 ED=[]
                 for index2,row_d in XD.iterrows():
                     d1=(row_c["Age"]-row_d["Age"])**2
                     d2=(row_c["Spending Score"]-row_d["Spending Score"])**2
                     d=np.sqrt(d1+d2)
                     ED.append(d)
                 X[i]=ED
                 i=i+1

             C=[]
             for index,row in X.iterrows():
                 min_dist=row[1]
                 pos=1
                 for i in range(K):
                     if row[i+1] < min_dist:
                         min_dist = row[i+1]
                         pos=i+1
                 C.append(pos)
             X["Cluster"]=C
             Centroids_new = X.groupby(["Cluster"]).mean()[["Spending Score","Age"]]
             if j == 0:
                 diff=1
                 j=j+1
             else:
                 diff = (Centroids_new['Spending Score'] - Centroids['Spending Score']).sum() + (Centroids_new['Age'] - Centr
                 print(diff.sum())
             Centroids = X.groupby(["Cluster"]).mean()[["Spending Score","Age"]]
```

```
<ipython-input-20-290045968b82>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy
  X[i]=ED
```

```
13.159649681514894
11.184759902566988
9.33881986711676
6.743655251226542
3.7937743070265704
3.1934643630169894
1.5344959406649288
0.6044088561470708
0.6941719396395882
0.7278722404398543
0.2908922078124725
0.0
```



```
In [21]: color=['blue','green','cyan','yellow','purple','grey']
         for k in range(K):
             data=X[X["Cluster"]==k+1]
             plt.scatter(data["Age"],data["Spending Score"],c=color[k])
         plt.scatter(Centroids["Age"],Centroids["Spending Score"],c='red')
         plt.xlabel('Age')
         plt.ylabel('Spending Score (1 - 100)')
         plt.show()
```



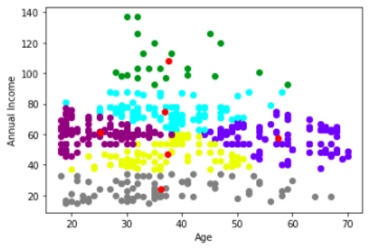
Again , we will see the result when clusters are formed between paramters Age and Annual Income of the DataSet.

```
In [55]: diff = 1
         j=0

         while(diff!=0):
             XD=X
             i=1
             for index1,row_c in Centroids.iterrows():
                 ED=[]
                 for index2,row_d in XD.iterrows():
                     d1=(row_c["Age"]-row_d["Age"])**2
                     d2=(row_c["Annual Income"]-row_d["Annual Income"])**2
                     d=np.sqrt(d1+d2)
                     ED.append(d)
                 X[i]=ED
                 i=i+1

             C=[]
             for index,row in X.iterrows():
                 min_dist=row[1]
                 pos=1
                 for i in range(K):
                     if row[i+1] < min_dist:
                         min_dist = row[i+1]
                         pos=i+1
                 C.append(pos)
             X["Cluster"]=C
             Centroids_new = X.groupby(["Cluster"]).mean()[["Annual Income","Age"]]
             if j == 0:
                 diff=1
                 j=j+1
             else:
                 diff = (Centroids_new['Annual Income'] - Centroids['Annual Income']).sum() + (Centroids_new['Age'] - Centroi
                 print(diff.sum())
             Centroids = X.groupby(["Cluster"]).mean()[["Annual Income","Age"]]
```

```
<ipython-input-55-da219eac357b>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy
  X[i]=ED
```

```
-0.1605772433412973
5.653773970583533
4.052819988074958
4.012371884091003
1.5363864741968918
1.6954051062033955
1.2808338405269026
4.487682122532377
4.359779751343368
1.0794409658064126
-0.04365481563340978
0.0
```

```
In [56]: color=['blue','green','cyan','yellow','purple','grey']
         for k in range(K):
             data=X[X["Cluster"]==k+1]
             plt.scatter(data["Age"],data["Annual Income"],c=color[k])
         plt.scatter(Centroids["Age"],Centroids["Annual Income"],c='red')
         plt.xlabel('Age')
         plt.ylabel('Annual Income')
         plt.show()
```



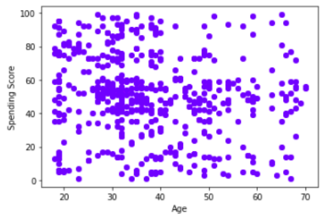
3.4 Cluster Prediction

Given set of points now we will try to predict to which cluster these points belongs.
First we train the model using all data points with the help of scikit learn.

Cluster Prediction

```
In [146]: import pandas as pd
          import numpy as np
          import matplotlib.pyplot as plt
          from sklearn.cluster import KMeans
          %matplotlib inline
```

```
In [147]: X = dframe[['Age','Spending Score']]
          plt.scatter(X['Age'],X['Spending Score'],c='blue')
          plt.xlabel('Age')
          plt.ylabel('Spending Score')
          plt.show()
```

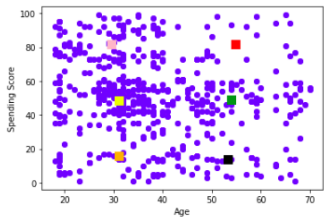


```
In [148]: from sklearn.cluster import KMeans
          Kmean = KMeans(n_clusters=6)
          Kmean.fit(X)

Out[148]: KMeans(n_clusters=6)
```

```
In [149]: Kmean.cluster_centers_

Out[149]: array([[54.13043478, 48.82608696],
                 [29.51376147, 81.74311927],
                 [31.30120482, 47.72289157],
                 [30.43859649, 13.87719298],
                 [52.83333333, 13.97916667],
                 [54.5       , 81.        ]])
```

Now we can see all the points are classified into 6 clusters numbered 0,1,2,3,4,5.

```
In [152]: plt.scatter(X['Age'],X['Spending Score'],c='blue')
          plt.scatter(53.93617021, 48.85106383, s=100, c='green', marker='s')
          plt.scatter(54.77777778, 81.59259259, s=100, c='red', marker='s')
          plt.scatter(31.14556962, 48.51898734, s=100, c='yellow', marker='s')
          plt.scatter(53.30434783, 13.7826087, s=100, c='black', marker='s')
          plt.scatter(29.51376147, 81.74311927, s=100, c='pink', marker='s')
          plt.scatter(31.07575758, 15.89393939, s=100, c='orange', marker='s')
          plt.xlabel('Age')
          plt.ylabel('Spending Score')
          plt.show()
```



```
In [153]: Kmean.labels_

Out[153]: array([2, 1, 3, 1, 2, 1, 3, 1, 4, 1, 4, 1, 4, 1, 3, 1, 2, 1, 4, 1, 2, 1,
                 4, 1, 4, 1, 0, 2, 2, 1, 4, 1, 4, 1, 4, 1, 4, 1, 3, 1, 0, 1, 0, 2,
                 4, 1, 0, 2, 2, 2, 0, 2, 2, 0, 0, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 2,
                 0, 0, 2, 2, 0, 0, 0, 0, 0, 2, 0, 2, 2, 0, 0, 2, 0, 0, 2, 0, 0, 2,
                 2, 0, 0, 2, 0, 2, 2, 2, 0, 2, 0, 2, 2, 0, 0, 2, 0, 2, 0, 0, 0, 0,
                 0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 2, 2, 2, 1, 3, 1, 2, 1, 4, 1, 4, 1,
                 2, 1, 3, 1, 4, 1, 3, 1, 4, 1, 2, 1, 3, 1, 0, 1, 3, 1, 4, 1, 4, 1,
                 4, 1, 3, 1, 3, 1, 0, 1, 3, 1, 4, 1, 4, 1, 3, 2, 3, 1, 3, 1, 4, 1,
                 4, 1, 4, 1, 2, 1, 4, 1, 2, 1, 4, 1, 3, 1, 3, 1, 3, 1, 4, 1, 4, 1,
                 3, 1, 2, 1, 4, 5, 2, 5, 3, 5, 4, 5, 4, 5, 3, 1, 3, 1, 2, 5, 4, 5,
                 0, 1, 3, 1, 3, 1, 2, 0, 2, 5, 3, 5, 3, 1, 3, 1, 3, 5, 3, 1, 2, 5,
                 2, 2, 3, 1, 2, 0, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2,
                 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 2, 0, 2, 2,
                 2, 0, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 2, 2, 2, 0, 0, 0, 5, 3, 1, 2,
                 1, 3, 5, 4, 5, 4, 1, 3, 1, 3, 1, 2, 5, 3, 5, 2, 5, 3, 1, 3, 1, 2,
                 0, 2, 1, 3, 5, 3, 1, 3, 1, 3, 5, 3, 1, 2, 5, 2, 0, 3, 5, 2, 2, 2,
                 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2,
                 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 2, 2, 2, 2, 2,
                 0, 2, 0, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 0, 2,
                 0, 2, 2, 2, 0, 0, 0, 0, 1, 3, 1, 2, 1, 4, 5, 4, 5, 2, 1, 3, 1, 3,
                 1, 4, 1, 4, 1, 0, 1, 3, 1, 2, 1, 4, 1, 3, 1, 4, 1, 3, 1, 3, 5, 4,
                 1, 0, 1, 4, 5, 4, 5, 4, 1, 3, 2, 3, 1, 4, 5, 4, 5, 3, 1, 3, 1, 2,
                 1, 4, 1, 0, 1, 4, 1, 3, 1, 3, 1, 4, 1, 3, 1, 0], dtype=int32)
```

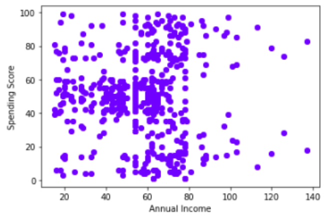
Now I am giving this model my points with Age = 70 and Spending Score = 45 and see to which cluster it belongs.

```
In [154]: sample_test=np.array([70,45])
          second_test=sample_test.reshape(1, -1)
          Kmean.predict(second_test)

Out[154]: array([0], dtype=int32)
```

Now train the model with different attributes this time Annual Income and Spending Score.

```
In [155]: import pandas as pd
          import numpy as np
          import matplotlib.pyplot as plt
          from sklearn.cluster import KMeans
          %matplotlib inline
```

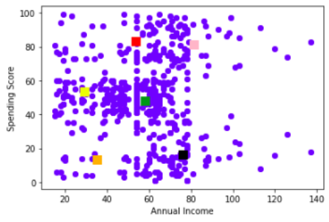
```
In [156]: X = dframe[['Annual Income','Spending Score']]
          plt.scatter(X['Annual Income'],X['Spending Score'],c='blue')
          plt.xlabel('Annual Income')
          plt.ylabel('Spending Score')
          plt.show()
```



```
In [157]: from sklearn.cluster import KMeans
          Kmean = KMeans(n_clusters=6)
          Kmean.fit(X)

Out[157]: KMeans(n_clusters=6)
```

```
In [158]: Kmean.cluster_centers_

Out[158]: array([[58.03389831, 48.11864407],
                 [53.61666667, 83.05      ],
                 [29.2804878 , 53.36585366],
                 [75.94318182, 16.5       ],
                 [81.16666667, 81.34848485],
                 [35.40740741, 13.22222222]])
```

```
In [159]: plt.scatter(X['Annual Income'],X['Spending Score'],c='blue')
          plt.scatter(58.03389831, 48.11864407, s=100, c='green', marker='s')
          plt.scatter(53.61666667, 83.05 , s=100, c='red', marker='s')
          plt.scatter(29.2804878 , 53.36585366, s=100, c='yellow', marker='s')
          plt.scatter(75.94318182, 16.5 , s=100, c='black', marker='s')
          plt.scatter(81.16666667, 81.34848485, s=100, c='pink', marker='s')
          plt.scatter(35.40740741, 13.22222222, s=100, c='orange', marker='s')
          plt.xlabel('Annual Income')
          plt.ylabel('Spending Score')
          plt.show()
```



```
In [160]: Kmean.labels_

Out[160]: array([2, 2, 5, 2, 2, 2, 5, 1, 5, 2, 5, 1, 5, 2, 5, 2, 2, 2, 5, 1, 2, 2,
                 5, 2, 5, 1, 5, 2, 5, 1, 5, 2, 5, 1, 5, 1, 5, 2, 5, 1, 2, 1, 2, 2,
                 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 3, 4, 0, 4, 3, 4, 3, 4,
                 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 0, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4,
                 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4,
                 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4,
                 3, 4, 0, 1, 5, 1, 0, 1, 5, 1, 5, 1, 5, 1, 5, 1, 5, 1, 0, 1, 0, 1,
                 0, 1, 5, 1, 5, 1, 0, 0, 0, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 0, 1,
                 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 1, 0,
                 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 0, 4, 3, 4, 0, 4, 3, 4, 3, 4, 3,
                 0, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 0, 0, 0,
                 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 5, 1, 5, 1, 0, 1, 3, 1, 3,
                 1, 3, 1, 3, 1, 0, 1, 3, 1, 0, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3,
                 1, 0, 1, 3, 1, 3, 1, 3, 1, 3, 0, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
                 4, 3, 4, 0, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 0], dtype=int32)
```

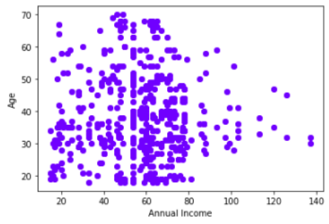
Now I am giving this model my points with Annual Income = 17 and Spending Score = 45 and see to which cluster it belongs.

```
In [162]: sample_test=np.array([17,45])
          second_test=sample_test.reshape(1, -1)
          Kmean.predict(second_test)

Out[162]: array([2], dtype=int32)
```

Now train the model with different attributes this time Annual Income and Age.

```
In [163]: import pandas as pd
          import numpy as np
          import matplotlib.pyplot as plt
          from sklearn.cluster import KMeans
          %matplotlib inline
```

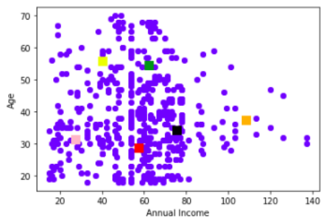
```
In [164]: X = dframe[['Annual Income','Age']]
          plt.scatter(X['Annual Income'],X['Age'],c='blue')
          plt.xlabel('Annual Income')
          plt.ylabel('Age')
          plt.show()
```



```
In [165]: from sklearn.cluster import KMeans
          Kmean = KMeans(n_clusters=6)
          Kmean.fit(X)

Out[165]: KMeans(n_clusters=6)
```

```
In [166]: Kmean.cluster_centers_

Out[166]: array([[ 62.15116279,  54.48837209],
                 [ 57.42553191,  28.74468085],
                 [ 40.28070175,  55.87719298],
                 [ 75.27192982,  34.39473684],
                 [ 27.3375    ,  31.4875    ],
                 [108.18181818,  37.54545455]])
```

```
In [168]: plt.scatter(X['Annual Income'],X['Age'],c='blue')
          plt.scatter(62.15116279,  54.48837209, s=100, c='green', marker='s')
          plt.scatter( 57.42553191,  28.74468085  , s=100, c='red', marker='s')
          plt.scatter(40.28070175,  55.87719298 , s=100, c='yellow', marker='s')
          plt.scatter(75.27192982,  34.39473684, s=100, c='black', marker='s')
          plt.scatter(27.3375    ,  31.4875 , s=100, c='pink', marker='s')
          plt.scatter(108.18181818,  37.54545455, s=100, c='orange', marker='s')
          plt.xlabel('Annual Income')
          plt.ylabel('Age')
          plt.show()
```



```
In [169]: Kmean.labels_

Out[169]: array([4, 4, 4, 4, 4, 4, 4, 4, 2, 4, 2, 4, 2, 4, 4, 4, 4, 4, 2, 4, 4, 4,
                 4, 4, 2, 4, 4, 4, 4, 4, 2, 4, 2, 4, 2, 4, 4, 4, 4, 4, 2, 4, 2, 4,
                 2, 4, 2, 4, 4, 4, 2, 4, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1,
                 2, 2, 1, 1, 2, 2, 2, 2, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
                 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
                 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 3, 3, 3, 3, 3, 3, 3, 0, 3, 0, 3,
                 3, 3, 3, 3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
                 3, 3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
                 0, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
                 5, 5, 1, 1, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 0, 0, 0,
                 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0,
                 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 3, 1, 1, 3, 0, 1, 1,
                 3, 0, 3, 3, 3, 3, 3, 3, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
                 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1,
                 1, 1, 0, 0, 0, 0, 1, 3, 3, 3, 1, 3, 3, 3, 0, 3, 0, 3, 3, 3, 3, 3,
                 3, 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
                 4, 4, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 2, 4, 2, 4, 2, 4,
                 4, 4, 4, 4, 4, 4, 2, 4, 4, 4, 4, 4, 4, 4, 2, 4, 2, 4, 4, 4, 4, 4,
                 2, 4, 2, 4, 2, 4, 4, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1,
                 2, 1, 1, 1, 2, 2, 2, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1,
                 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0,
                 1, 0, 1, 0, 0, 0, 0, 0, 1, 3, 1, 1, 1, 0, 0, 0, 0, 3, 3, 3, 3, 3,
                 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2], dtype=int32)
```

Now I am giving this model my points with Annual Income = 17 and Age = 15 and see to which cluster it belongs.

```
In [171]: sample_test=np.array([17,15])
          second_test=sample_test.reshape(1, -1)
          Kmean.predict(second_test)

Out[171]: array([4], dtype=int32)
```

3.5 Our Next Move

3.5.1 Extension of Customer Segmentation by Recency/Monetary Matrix

What Is Recency, Frequency, Monetary Value (RFM)?

Recency, frequency, monetary value is a marketing analysis tool used to identify a company's or an organization's best customers by using certain measures.

The RFM model is based on three quantitative factors:

- Recency: How recently a customer has made a purchase
- Frequency: How often a customer makes a purchase
- Monetary Value: How much money a customer spends on purchases

3.5.2 Implementaion Of RFM Using Some DataSet

```
In [79]: import pandas as pd
         import warnings
         warnings.filterwarnings('ignore')
```

```
In [82]: #importing data file
         orders = pd.read_csv('/home/vivek/Desktop/Spring Project/rfm analsis/sample-data.csv',encoding= 'unicode_escape')
```

```
In [83]: orders.head()
```

Out[83]:

	order_date	order_id	customer	grand_total
0	9/7/2011	CA-2011-100006	Dennis Kane	378.0
1	7/8/2011	CA-2011-100090	Ed Braxton	699.0
2	3/14/2011	CA-2011-100293	Neil Franz?sisch	91.0
3	1/29/2011	CA-2011-100328	Jasper Cacioppo	4.0
4	4/8/2011	CA-2011-100363	Jim Mitchum	21.0

Making the RFM Table

Set this date to the current day and extract all orders.

```
In [103]: import datetime as dt
          NOW = dt.datetime(2014,12,31)
```

```
In [104]: # Make the date_placed column datetime
          orders['order_date'] = pd.to_datetime(orders['order_date'])
```

Create the RFM Table

```
In [105]: #calculating RFM table using lambda function
          rfmTable = orders.groupby('customer').agg({'order_date': lambda x: (NOW - x.max()).days, # calculation Recency
                                                  'order_id': lambda x: len(x),      # calculation Frequency
                                                  'grand_total': lambda x: x.sum()}) # calculation Monetary Value

          rfmTable['order_date'] = rfmTable['order_date'].astype(int)
          rfmTable.rename(columns={'order_date': 'recency',
                                  'order_id': 'frequency',
                                  'grand_total': 'monetary_value'}, inplace=True)
```

Validating the RFM Table

```
In [89]: rfmTable.head()
```

Out[89]:

customer	recency	frequency	monetary_value
Aaron Bergman	415	3	887.0
Aaron Hawkins	12	7	1744.0
Aaron Smayling	88	7	3050.0
Adam Bellavance	54	8	7756.0
Adam Hart	34	10	3249.0

Customer **Aaron Bergman** has

- frequency:"3"
- monetary value:"$887"
- recency:"415 days"

```
In [90]: aaron = orders[orders['customer']=='Aaron Bergman']
         aaron
```

Out[90]:

	order_date	order_id	customer	grand_total
624	2011-02-19	CA-2011-152905	Aaron Bergman	13.0
665	2011-03-07	CA-2011-156587	Aaron Bergman	310.0
2336	2013-11-11	CA-2013-140935	Aaron Bergman	564.0

Inserting the date of Aaron purchase and comparing it to the recency in the rfmTable we verify our RFM table is correct.

```
In [106]: (NOW - dt.datetime(2013,11,11)).days==415

Out[106]: True
```

Determining RFM Quartiles

In [92]:
```
quantiles = rfmTable.quantile(q=[0.25,0.5,0.75])
```

In [93]:
```
quantiles
```

Out[93]:

	recency	frequency	monetary_value
0.25	30.0	5.0	1145.0
0.50	75.0	6.0	2257.0
0.75	183.0	8.0	3784.0

Send quantiles to a dictionary, easier to use.

In [94]:
```
quantiles = quantiles.to_dict()
```

In [95]:
```
quantiles
```

Out[95]:
```
{'recency': {0.25: 30.0, 0.5: 75.0, 0.75: 183.0},
 'frequency': {0.25: 5.0, 0.5: 6.0, 0.75: 8.0},
 'monetary_value': {0.25: 1145.0, 0.5: 2257.0, 0.75: 3784.0}}
```

Creating the RFM segmentation table

In [96]:
```
rfmSegmentation = rfmTable
```

We create two classes for the RFM segmentation since, being high recency is bad, while high frequency and monetary value is good.

n [107]:
```
# Arguments (x = value, p = recency, monetary_value, frequency, k = quartiles dict)
def RClass(x,p,d):
    if x <= d[p][0.25]:
        return 1
    elif x <= d[p][0.50]:
        return 2
    elif x <= d[p][0.75]:
        return 3
    else:
        return 4

# Arguments (x = value, p = recency, monetary_value, frequency, k = quartiles dict)
def FMClass(x,p,d):
    if x <= d[p][0.25]:
        return 4
    elif x <= d[p][0.50]:
        return 3
    elif x <= d[p][0.75]:
        return 2
    else:
        return 1
```

n [108]:
```
rfmSegmentation['R_Quartile'] = rfmSegmentation['recency'].apply(RClass, args=('recency',quantiles,))
rfmSegmentation['F_Quartile'] = rfmSegmentation['frequency'].apply(FMClass, args=('frequency',quantiles,))
rfmSegmentation['M_Quartile'] = rfmSegmentation['monetary_value'].apply(FMClass, args=('monetary_value',quantiles,))
```

In [109]:
```
rfmSegmentation['RFMClass'] = rfmSegmentation.R_Quartile.map(str) \
                            + rfmSegmentation.F_Quartile.map(str) \
                            + rfmSegmentation.M_Quartile.map(str)
```

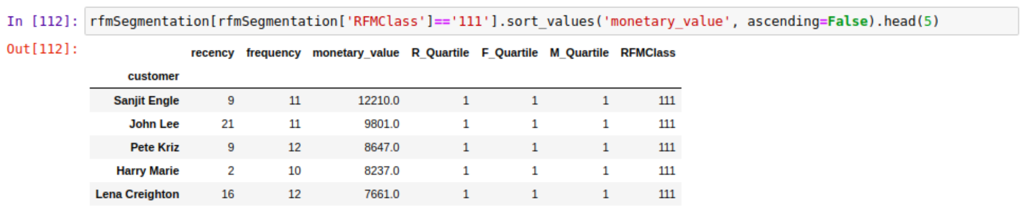
In [110]:
```
rfmSegmentation.head()
```

Out[110]:

	recency	frequency	monetary_value	R_Quartile	F_Quartile	M_Quartile	RFMClass
customer							
Aaron Bergman	415	3	887.0	4	4	4	444
Aaron Hawkins	12	7	1744.0	1	2	3	123
Aaron Smayling	88	7	3050.0	3	2	2	322
Adam Bellavance	54	8	7756.0	2	2	1	221
Adam Hart	34	10	3249.0	2	1	2	212

In [111]:
```
# Uncomment any of the following lines to: copy data to clipboard or save it to a CSV file.
# rfmSegmentation.to_clipboard()
# rfmSegmentation.to_csv('rfm-table.csv', sep=',')
```

Who are the top 5 best customers? by RFM Class (111), high spenders who buy recently and frequently



```
In [112]: rfmSegmentation[rfmSegmentation['RFMClass']=='111'].sort_values('monetary_value', ascending=False).head(5)
Out[112]:
```

customer	recency	frequency	monetary_value	R_Quartile	F_Quartile	M_Quartile	RFMClass
Sanjit Engle	9	11	12210.0	1	1	1	111
John Lee	21	11	9801.0	1	1	1	111
Pete Kriz	9	12	8647.0	1	1	1	111
Harry Marie	2	10	8237.0	1	1	1	111
Lena Creighton	16	12	7661.0	1	1	1	111

3.5.3 Why using RFM is not feasible over K-Means?

More the Quartiles more will be the RFM cells. For ex. if quartiles = 3 Total no. of RFM cells are 27 , if quartiles = 4 Total no. of RFM cells are 64.

But in K-means the value of K(no. of clusters) is usually less than 10. It can be seen in experiments that several RFM cells having different values usually are almost same.

The meaning of the Frequency and Monetary with the same score point might be different, e.g. one customer with a "543" RFM score and the other customers with a "443" RFM score, their Frequency might be quite different though they have the same score of "4".

This implies that more unnecessary computation is required in case of RFM and in terms of Marketing , more strategies and more money have to be spent when compared to K-means.

3.6 Scope of Analysis

In general, the methods used to gather the data for this project can easily be extended into other relevant contexts.The scope of the paper is limited to the following four intertwined goals:

- To cluster customers based on common purchasing behaviors for future operations/marketing projects.
- To incorporate best mathematical, visual, programming, and business practices into a thoughtful analysis that is understood across a variety of contexts and disciplines.
- To investigate how similar data and algorithms could be used in future data mining projects.
- To create an understanding and inspiration of how data science can be used to solve real-world problems.

4 Conclusion and Future Scope

4.1 Conclusion

The benefits of customer segmentation analysis are clear. By having a stronger understanding of their consumer base, retailers can properly allocate resources to collect and mine relevant information to boost profits. However, getting to the point of peforming high-level customer segmentation analysis is more difficult than originally thought for many retailers. Many retailers may have the rights to the necessary data to perform the analysis, but do not have either the ability to access it in a user-friendly manner or have an employee that has the skillset to work with it. The lack of proper personnel or equipment to handle the necessary volume of data is perhaps the biggest hindrance to smaller firms being able to perform such analysis. The popularity of open-source programming software such as R or Python has certainly helped make this type of analysis more accessible, but it still would require retailers having someone on their team who can code in either of those languages. Additionally, some retailers are simply unaware of either the extent of their data collection or are not yet inspired to dig into it. Nevertheless, retatilers that have not fully adopted customer segmentation analysis are likely not doing so simply because they cannot afford to spend the time, money, or labor to perform the analysis. Therefore, it is an aim of this paper to show that this rich analysis can be performed cheaply and efficiently.

4.2 Future Scope

K means clustering is one of the most popular clustering algorithms and usually the first thing practitioners apply when solving clustering tasks to get an idea of the structure of the dataset. The goal of K means is to group data points into distinct non-overlapping subgroups. One of the major applications of K means clustering is segmentation of customers to get a better understanding of them which in turn could be used to increase the revenue of the company.

Segmentation models vary from basic to complex, and the approaches to developing and applying them is a topic for an entire book itself

Example : A hotel chain that caters to families might simply segment its market based on income level and travel frequency. The chain might find that a group of moderate income-frequent travelers exists that is swayed by certain loyalty program rewards. It might find another important group exists that is more swayed by on-site amenities. With this information, the hotel can optimize its offerings and loyalty programs to appeal to each group's unique needs.

References

[1] https://towardsdatascience.com/clustering-algorithms-for-customer-segmentation-af637c68

[2] https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/

[3] https://www.edupristine.com/blog/beyond-k-means

[4] https://www.greenbook.org/marketing-research/project-scope-market-segmentation-hm08