Let’s begin the analysis by importing libraries and the data set into Python.

#Importing Libraries  
%matplotlib inline  
**import** **pandas** **as** **pd**  
**import** **matplotlib.pyplot** **as** **plt**  
**from** **sklearn.preprocessing** **import** scale  
**from** **sklearn.cluster** **import** KMeans  
**import** **seaborn** **as** **sns**  
  
#reading Dataset  
retail = pd.read\_csv("Online Retail.csv", sep = ',',encoding = "ISO-8859-1", header= **0**)  
# parse date  
retail['InvoiceDate'] = pd.to\_datetime(retail['InvoiceDate'], format = "%d-%m-%Y %H:%M")



Let's start with some preliminary data cleaning. Now, as you can notice, the data set is at the granularity of order level. So, it doesn’t make much sense to do missing value imputation because it would be very difficult to predict the individual missing details of individual orders. Hence, you have to remove the entries with the missing values.

#dropping the na cells  
order\_wise = retail.dropna()



But, if you remember, our main objective is to cluster the customers of the online store. So, you need to transform this order-wise data into customer-wise data by finding key attributes that best characterises a customer. This is achieved through RFM analysis.

**RFM analysis**

In RFM analysis, you look at the recency, frequency and the monetary scores of all the customers for segmentation.

* **Recency:** It measures how recently you visited the store or made a purchase
* **Frequency:** It measures the frequency of the transactions the customers made
* **Monetary:** It measures how much the customer spent on purchases he/she made

So, your target is to compute the RFM numbers for each customer, which effectively means that the granularity level of your data set will change from Invoice number to the CustomerID. Thus, you will have one unique row corresponding to each customer.

Let’s start with creating customer-wise data. We begin with the computation of M of the RFM, that is the total monetary value of the purchases made by each customer.

Create a vector named Amount, which creates the total monetary value of each order, and append the column to your data set.

#RFM implementation  
amount = pd.DataFrame(order\_wise.Quantity \* order\_wise.UnitPrice, columns = ["Amount"])  
  
#merging amount in order\_wise  
order\_wise = pd.concat(objs = [order\_wise, amount], axis = **1**, ignore\_index = **False**)



Now, sort the data set in order of CustomerID. Next, create a new vector — monetary — which gives the aggregated purchase amount for each customer.

#Monetary Function  
monetary = order\_wise.groupby("CustomerID").Amount.sum()  
monetary = monetary.reset\_index()



This data frame monetary is the M of the RFM framework.

Next, let’s compute the frequency of purchase for each customer, i.e. the F of the RFM framework. For this, you will count the number of unique Invoice Numbers for each Customer ID. This is the “Frequency” corresponding to each customer.

#Frequency function  
frequency = order\_wise[['CustomerID', 'InvoiceNo']]  
  
k = frequency.groupby("CustomerID").InvoiceNo.count()  
k = pd.DataFrame(k)  
k = k.reset\_index()  
k.columns = ["CustomerID", "Frequency"]



Finally, merge this data frame with the “Frequency” of each customer into your earlier data set containing the “Monetary” value.

#creating master dataset  
master = monetary.merge(k, on = "CustomerID", how = "inner")



Thus, the data frame master contains both the monetary and the frequency attributes corresponding to each customer IDs. Now, you have to turn your attention towards the computation of the recency, i.e. for how long a customer has not visited the online store.

Begin by extracting the Customer ID and Invoice Date from the data. Now, find the latest “Invoice Date” which forms the reference point for the calculation of the “Recency” of each customer. For each order corresponding to each customer, you find the difference from the latest “Invoice Date” and then find the minimum “Recency” value for each customer.

#Generating recency function  
recency = order\_wise[['CustomerID','InvoiceDate']]  
maximum = max(recency.InvoiceDate)  
maximum = maximum + pd.DateOffset(days=**1**)  
recency['diff'] = maximum - recency.InvoiceDate  
  
#Dataframe merging by recency  
df = pd.DataFrame(recency.groupby('CustomerID').diff.min())  
df = df.reset\_index()  
df.columns = ["CustomerID", "Recency"]



Now, the data frame recency contains the recency for each customer. Let’s merge it to the RFM data set and change the format to the required form.

#Combining all recency, frequency and monetary parameters  
RFM = k.merge(monetary, on = "CustomerID")  
RFM = RFM.merge(df, on = "CustomerID")



Thus, you have obtained the RFM data corresponding to each customer. These 3 attributes will form the basis, depending on which the customers will be segregated into different clusters.

However, your data preparation is still not complete. You have already seen previously how the clustering process can be impacted due to the presence of outliers. So, let’s treat the data set for outliers. One way to do it is by eliminating all the data points which fall outside the 1.5 times the IQR of the 1st and the 3rd quartile.

# outlier treatment for Amount  
plt.boxplot(RFM.Amount)  
Q1 = RFM.Amount.quantile(**0.25**)  
Q3 = RFM.Amount.quantile(**0.75**)  
IQR = Q3 - Q1  
RFM = RFM[(RFM.Amount >= Q1 - **1.5**\*IQR) & (RFM.Amount <= Q3 + **1.5**\*IQR)]  
  
# outlier treatment for Frequency  
plt.boxplot(RFM.Frequency)  
Q1 = RFM.Frequency.quantile(**0.25**)  
Q3 = RFM.Frequency.quantile(**0.75**)  
IQR = Q3 - Q1  
RFM = RFM[(RFM.Frequency >= Q1 - **1.5**\*IQR) & (RFM.Frequency <= Q3 + **1.5**\*IQR)]  
  
# outlier treatment for Recency  
plt.boxplot(RFM.Recency)  
Q1 = RFM.Recency.quantile(**0.25**)  
Q3 = RFM.Recency.quantile(**0.75**)  
IQR = Q3 - Q1  
RFM = RFM[(RFM.Recency >= Q1 - **1.5**\*IQR) & (RFM.Recency <= Q3 + **1.5**\*IQR)]

**Please note that results for the cluster may vary since random\_state is not used in the K-Menas code.**

Now, you will actually create clusters on the cleaned data. Let's see how this is done.

classmodel\_clus = KMeans(n\_clusters=**3**, init='random', n\_init=**10**, max\_iter=**50**).fit(RFM\_norm1)



Here, let's try to understand the different parameters from [Scikit Learn Documentation](http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html):

* **n\_clusters:** The number of clusters to form as well as the number of centroids to generate
* **init:** Method for initialization of the cluster centres
* **n\_init:** Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n\_init consecutive runs in terms of inertia (explained below).
* **max\_iter:** Maximum number of iterations of the k-means algorithm for a single run in case it doesn't converge on its own

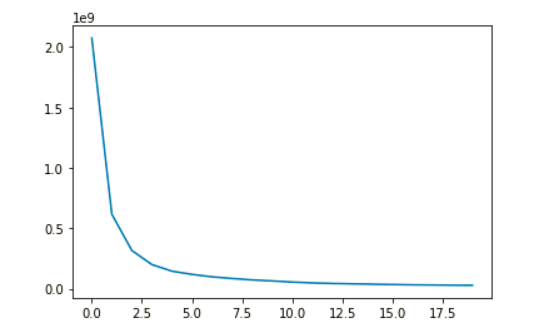
Let's now try to understand the output of the KMeans() function.

* **cluster\_centers:** Coordinates of cluster centres
* **labels:** Labels of each point i.e. the labels to which the points belong
* **inertia:** Sum of squared distances of points to their closest cluster centr

As you saw, business constraints are one of the factors that help decide the value of K. However, there is a mathematical route as well to arrive at the optimal value of K. This is done through the **elbow curve method**.

You have seen earlier that KMeans() stores the sum of the squared distance of the points to their respective clusters centres as **inertia.** In other words, inertia represents, how tightly the different clusters are formed. As we increase the number of clusters, the inertia value is bound to decrease as the individual clusters become more compact. Thus, the plot of inertia against the number of clusters becomes a monotonically decreasing plot.

However, in this plot (Fig 1), you can notice a distinct elbow. Beyond the elbow point, the additional (marginal) decrease in inertia with each increase in the cluster number is not very prominent. Thus, the elbow in the curve gives an estimate of the optimal number K in K Means.

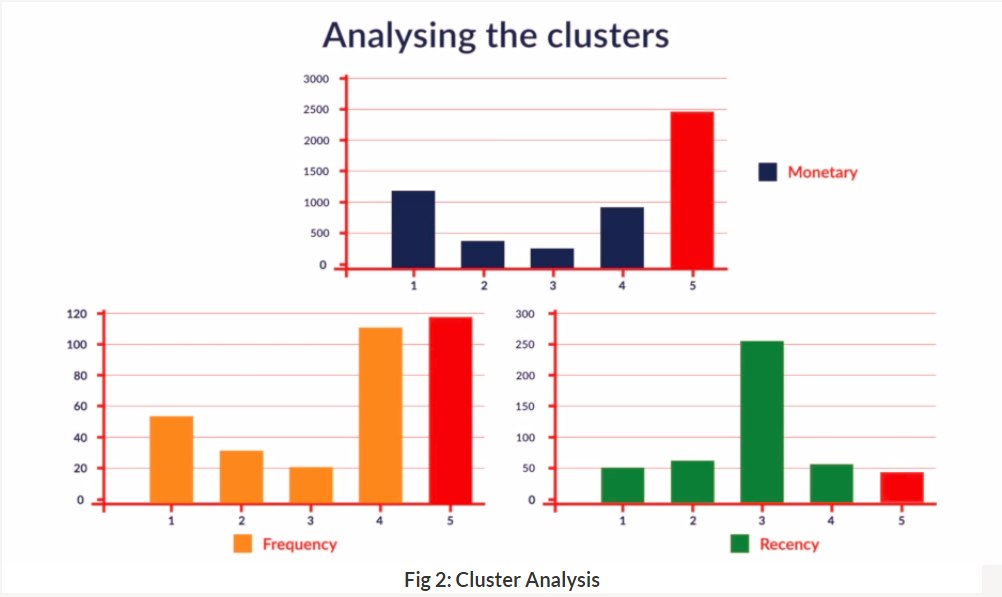


**Fig 1: Elbow Curve**

Now let's implement the elbow curve method to see what would be the optimal number of K in our case. We would use a loop to store the inertia value while changing the value of K from 1 to 21.

We will also be looking at Hopkins statistics in Python to check if our data has some meaningful clusters or not. Together with Hopkins and SSD, we will also test our clusters using Silhouette analysis.

You found that cluster 5 was the best customer segment from the store’s point of view. These customers make a purchase for a higher amount, more frequently, and these customers had visited the site recently. Thus, the store may offer them a reward or loyalty points or some privileged status, to keep them attracted and coming back to the store.



**Clusters**

On the other hand, cluster 3 had the worst customers from the store’s point of view. Thus, the store may decide to focus more on this group. Similarly, in cluster 1, the customers had favourable features in terms of the purchase amount and recency; however, these have low frequency. Thus, if the store can re-design its incentive strategy and entice these customers into making a purchase more frequently, they could turn profitable for the store.