```
#importing required libraries
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt
%matplotlib inline
```

```
#importing the dataset
data = pd.read_excel('Dataset.xlsx')
```

```
# first 5 rows of the data
data.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	U King
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	U King
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	U King
			KNITTED UNION					

```
# shape of the data
data.shape
```

```
(541909, 8)
```

Cleaning the Dataset

```
# Checking missing values

round(100*data.isnull().sum()/len(data))
```

```
InvoiceNo      0.0
StockCode      0.0
```

```
StockCode       0.0
Description     0.0
Quantity        0.0
InvoiceDate     0.0
UnitPrice       0.0
CustomerID     25.0
Country         0.0
dtype: float64
```

Here, CustomerID column having 25% missing values. but, the column is very important for modelling. we can't impute those missing values, we simply dropping all the rows having missing values

```
# dropping missing values using dropna()
data = data.dropna()
data.shape
```

```
(406829, 8)
```

```
# viewing the dataset
data.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerII
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.(
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.(
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.(
			KNITTED UNION				

Data Preparation

- based on anlysis the retail store customers segmented by using 'RFM',

```
* R (Recency): Number of days since last purchase
* F (Frequency): Number of tracsactions
```

```
* M (Monetary): Total amount of transactions (revenue contributed)
```

1. Monetary

```
# finding monetary
data['Amount'] = data['UnitPrice']*data['Quantity']
data.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerII
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.(
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.(
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.(
			KNITTED				

```
# Grouping 'CustomerId' and 'Amount'
grouped_data = data.groupby('CustomerID')['Amount'].sum()
grouped_data = grouped_data.reset_index()
grouped_data.head()
```

	CustomerID	Amount
0	12346.0	0.00
1	12347.0	4310.00
2	12348.0	1797.24
3	12349.0	1757.55
4	12350.0	334.40

2. Frequency

```
# Finding Frequency
frequency = data.groupby('CustomerID')['InvoiceNo'].count()
frequency = frequency.reset_index()
frequency.columns = ['CustomerID', 'frequency']
frequency.head()
```

	CustomerID	frequency
0	12346.0	2
1	12347.0	182
2	12348.0	31
3	12349.0	73
4	12350.0	17

```
# Merge the two dataframes
grouped_data = pd.merge(grouped_data, frequency, on='CustomerID', how='inner')
grouped_data.head()
```

	CustomerID	Amount	frequency
0	12346.0	0.00	2
1	12347.0	4310.00	182
2	12348.0	1797.24	31
3	12349.0	1757.55	73
4	12350.0	334.40	17

3. Recency

```
# Checking datatypes
data.dtypes
```

```
InvoiceNo              object
StockCode              object
Description            object
Quantity                int64
InvoiceDate    datetime64[ns]
UnitPrice             float64
CustomerID            float64
Country                object

Amount                float64
dtype: object
```

```
# compute the max date
max_date = max(data['InvoiceDate'])
max_date
```

```
Timestamp('2011-12-09 12:50:00')
```

```
# compute the Recency
data['Recency'] = max_date - data['InvoiceDate']
data.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	U King
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	U King
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	U King
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	U King

```
# Recency
last_purchase = data.groupby('CustomerID')['Recency'].min()
last_purchase = last_purchase.reset_index()
last_purchase.head()
```

```
# merging the datasets
grouped_data = pd.merge(grouped_data, last_purchase, on='CustomerID', how='inner')
```

```
grouped_data.head()
```

	CustomerID	Amount	frequency	Recency
0	12346.0	0.00	2	325 days 02:33:00
1	12347.0	4310.00	182	1 days 20:58:00
2	12348.0	1797.24	31	74 days 23:37:00
3	12349.0	1757.55	73	18 days 02:59:00
4	12350.0	334.40	17	309 days 20:49:00

```
# getting number of days only
grouped_data['Recency'] = grouped_data['Recency'].dt.days
grouped_data.head()
```

	CustomerID	Amount	frequency	Recency
0	12346.0	0.00	2	325
1	12347.0	4310.00	182	1
2	12348.0	1797.24	31	74
3	12349.0	1757.55	73	18
4	12350.0	334.40	17	309

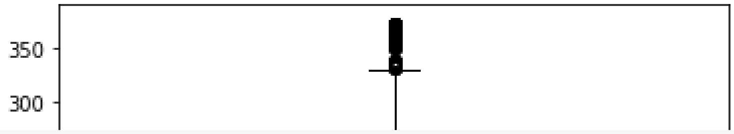
Checking Outliers

- 'RFM' values may have some Extreme points

```
# 1. outlier treatment
plt.boxplot(grouped_data['Recency'])
```

```
{'boxes': [<matplotlib.lines.Line2D at 0x7f65c8e3fc10>],
 'caps': [<matplotlib.lines.Line2D at 0x7f65bbdc8490>,
  <matplotlib.lines.Line2D at 0x7f65bb3672d0>],
```

```
 'fliers': [<matplotlib.lines.Line2D at 0x7f65bcf21150>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f65bb367a10>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f65bbdc8a50>,
  <matplotlib.lines.Line2D at 0x7f65bbdc8090>]}
```




```
# removing (statistical) outliers
Q1 = grouped_data.Amount.quantile(0.05)
Q3 = grouped_data.Amount.quantile(0.95)
IQR = Q3 - Q1
grouped_data = grouped_data[(grouped_data.Amount >= Q1 - 1.5*IQR) & (grouped_data.Amount <= Q

# outlier treatment for Recency
Q1 = grouped_data.Recency.quantile(0.05)
Q3 = grouped_data.Recency.quantile(0.95)
IQR = Q3 - Q1
grouped_data = grouped_data[(grouped_data.Recency >= Q1 - 1.5*IQR) & (grouped_data.Recency <=

# outlier treatment for frequency
Q1 = grouped_data.frequency.quantile(0.05)
Q3 = grouped_data.frequency.quantile(0.95)
IQR = Q3 - Q1
grouped_data = grouped_data[(grouped_data.frequency >= Q1 - 1.5*IQR) & (grouped_data.frequenc
```

Scaling

- Kmeans clustering is distance based algorithm scaling the data is more important

```
# Scaling
from sklearn.preprocessing import StandardScaler
data_scaled= grouped_data[['Amount', 'frequency', 'Recency']]

# instantiate
scaler = StandardScaler()

# fit_transform
df_scaled = scaler.fit_transform(data_scaled)
df_scaled.shape
```

```
(4293, 3)
```

```
df_scaled = pd.DataFrame(df_scaled)
df_scaled.columns = ['Amount', 'Frequency', 'Recency']
df_scaled.head()
```

	Amount	Frequency	Recency
0	-0.723738	-0.752888	2.301611
1	1.731617	1.042467	-0.906466
2	0.300128	-0.463636	-0.183658
3	0.277517	-0.044720	-0.738141
4	-0.533235	-0.603275	2.143188

KMeans Clustering

```
# importing KMeans
from sklearn.cluster import KMeans
```

```
# k-means with k value 2
kmeans = KMeans(n_clusters=2)
kmeans.fit(df_scaled)
```

```
KMeans(n_clusters=2)
```

```
pred=kmeans.predict(df_scaled)
```

```
pred
```

```
array([0, 1, 0, ..., 0, 0, 0], dtype=int32)
```

```
pd.Series(pred).value_counts()
```

```
0    3282
1    1011
dtype: int64
```

```
kmeans.inertia_
```

```
11408.667844165004
```

```
kmeans.score(df_scaled)
```

```
-11408.667844165004
```

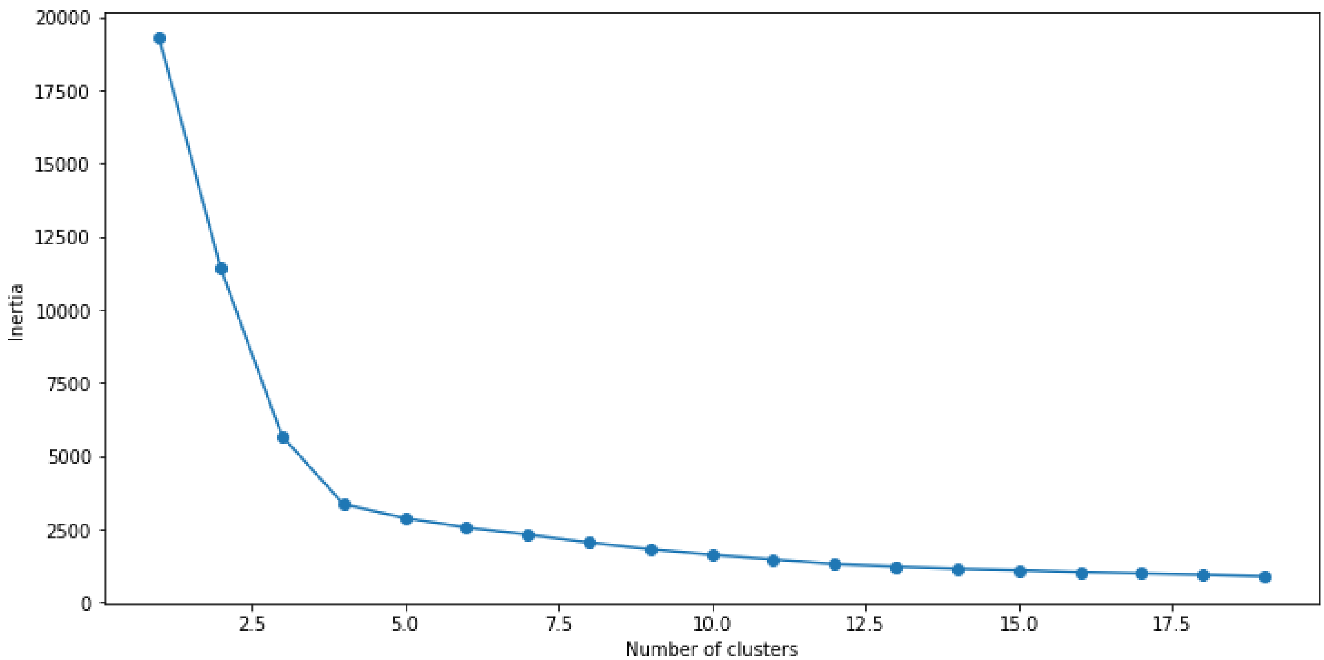
```
SSE = []
for cluster in range(1,20):
```

```
for cluster in range(1,20):
    kmeans = KMeans(n_clusters = cluster)
    kmeans.fit(df_scaled)
    SSE.append(kmeans.inertia_)
```

```
frame = pd.DataFrame({'Cluster':range(1,20), 'SSE':SSE})
```

```
# Elbow curve
plt.figure(figsize=(12,6))
plt.plot(frame['Cluster'], frame['SSE'], marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

```
Text(0, 0.5, 'Inertia')
```




```
# from elbow curve we setting the k value to '4'
kmeans = KMeans(n_clusters = 4)
kmeans.fit(df_scaled)
pred = kmeans.predict(df_scaled)
```

```
pred
```

```
array([0, 2, 1, ..., 0, 1, 1], dtype=int32)
```

```
frame = pd.DataFrame(df_scaled)
```

```
frame['cluster'] = pred
```

```
frame['cluster'].value_counts()
```

```
1    2250
0    1035
2     782
3     226
Name: cluster, dtype: int64
```

Here, we segmented the customers into 4 groups or clusters

✓ 0s completed at 12:12 PM