In [5]:
```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
#We do not want to see warnings
warnings.filterwarnings("ignore")
```

In [6]:
```
#import data
data = pd.read_csv("uber.csv")
```

In [7]:
```
#Create a data copy
df = data.copy()
```

In [8]:
```
#Print data
df.head
```

Out[8]:
```
<bound method NDFrame.head of         Unnamed: 0                            key  f
are_amount  \
0         24238194    2015-05-07 19:52:06.0000003         7.5
1         27835199    2009-07-17 20:04:56.0000002         7.7
2         44984355   2009-08-24 21:45:00.00000061        12.9
3         25894730    2009-06-26 08:22:21.0000001         5.3
4         17610152  2014-08-28 17:47:00.000000188        16.0
...            ...                            ...         ...
199995    42598914   2012-10-28 10:49:00.00000053         3.0
199996    16382965    2014-03-14 01:09:00.0000008         7.5
199997    27804658   2009-06-29 00:42:00.00000078        30.9
199998    20259894    2015-05-20 14:56:25.0000004        14.5
199999    11951496   2010-05-15 04:08:00.00000076        14.1

                pickup_datetime  pickup_longitude  pickup_latitude  \
0       2015-05-07 19:52:06 UTC        -73.999817        40.738354
1       2009-07-17 20:04:56 UTC        -73.994355        40.728225
2       2009-08-24 21:45:00 UTC        -74.005043        40.740770
3       2009-06-26 08:22:21 UTC        -73.976124        40.790844
4       2014-08-28 17:47:00 UTC        -73.925023        40.744085
...                         ...               ...              ...
199995  2012-10-28 10:49:00 UTC        -73.987042        40.739367
199996  2014-03-14 01:09:00 UTC        -73.984722        40.736837
199997  2009-06-29 00:42:00 UTC        -73.986017        40.756487
199998  2015-05-20 14:56:25 UTC        -73.997124        40.725452
199999  2010-05-15 04:08:00 UTC        -73.984395        40.720077

        dropoff_longitude  dropoff_latitude  passenger_count
0              -73.999512         40.723217                1
1              -73.994710         40.750325                1
2              -73.962565         40.772647                1
3              -73.965316         40.803349                3
4              -73.973082         40.761247                5
...                   ...               ...              ...
199995         -73.986525         40.740297                1
199996         -74.006672         40.739620                1
199997         -73.858957         40.692588                2
199998         -73.983215         40.695415                1
199999         -73.985508         40.768793                1

[200000 rows x 9 columns]>
```

In [9]:
```python
#Get Info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   Unnamed: 0         200000 non-null  int64  
 1   key                200000 non-null  object 
 2   fare_amount        200000 non-null  float64
 3   pickup_datetime    200000 non-null  object 
 4   pickup_longitude   200000 non-null  float64
 5   pickup_latitude    200000 non-null  float64
 6   dropoff_longitude  199999 non-null  float64
 7   dropoff_latitude   199999 non-null  float64
 8   passenger_count    200000 non-null  int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

In [10]:
```
#pickup_datetime is not in required data format
df["pickup_datetime"] = pd.to_datetime(df["pickup_datetime"])
```

In [11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype              
---  ------             --------------   -----              
 0   Unnamed: 0         200000 non-null  int64              
 1   key                200000 non-null  object             
 2   fare_amount        200000 non-null  float64            
 3   pickup_datetime    200000 non-null  datetime64[ns, UTC]
 4   pickup_longitude   200000 non-null  float64            
 5   pickup_latitude    200000 non-null  float64            
 6   dropoff_longitude  199999 non-null  float64            
 7   dropoff_latitude   199999 non-null  float64            
 8   passenger_count    200000 non-null  int64              
dtypes: datetime64[ns, UTC](1), float64(5), int64(2), object(1)
memory usage: 13.7+ MB
```

In [12]:
```
#Statistics of data
df.describe()
```

Out[12]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude
count	2.000000e+05	200000.000000	200000.000000	200000.000000	199999.000000
mean	2.771250e+07	11.359955	-72.527638	39.935885	-72.525292
std	1.601382e+07	9.901776	11.437787	7.720539	13.117408
min	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.666300
25%	1.382535e+07	6.000000	-73.992065	40.734796	-73.991407
50%	2.774550e+07	8.500000	-73.981823	40.752592	-73.980093
75%	4.155530e+07	12.500000	-73.967154	40.767158	-73.963658
max	5.542357e+07	499.000000	57.418457	1644.421482	1153.572603

In [13]:

```
#Number of missing values
df.isnull().sum()
```

Out[13]:

```
Unnamed: 0           0
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

In [14]:

```
#Correlation
df.corr()
```

Out[14]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_long
Unnamed: 0	1.000000	0.000589	0.000230	-0.000341	0.0
fare_amount	0.000589	1.000000	0.010457	-0.008481	0.0
pickup_longitude	0.000230	0.010457	1.000000	-0.816461	0.8
pickup_latitude	-0.000341	-0.008481	-0.816461	1.000000	-0.7
dropoff_longitude	0.000270	0.008986	0.833026	-0.774787	1.0
dropoff_latitude	0.000271	-0.011014	-0.846324	0.702367	-0.9
passenger_count	0.002257	0.010150	-0.000414	-0.001560	0.0

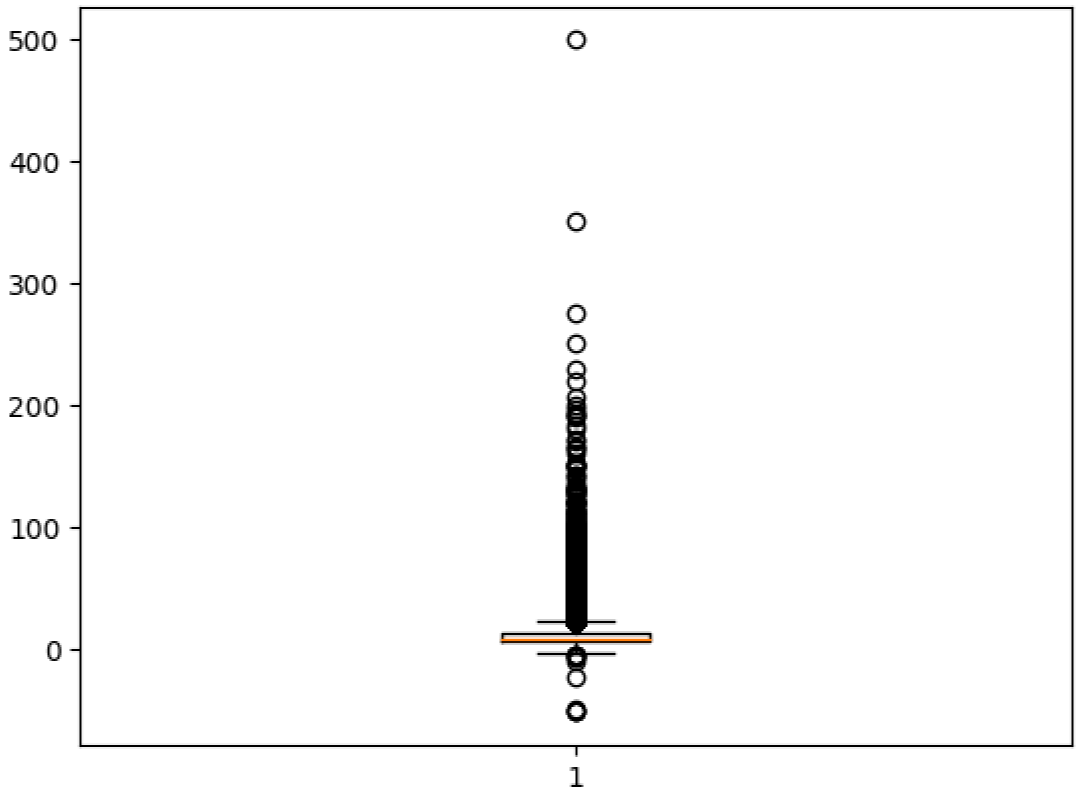
In [15]:

```
#Drop the rows with missing values
df.dropna(inplace=True)
```

In [16]:

```
plt.boxplot(df['fare_amount'])
```

Out[16]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x1be07575f10>,
  <matplotlib.lines.Line2D at 0x1be075931f0>],
 'caps': [<matplotlib.lines.Line2D at 0x1be07593490>,
  <matplotlib.lines.Line2D at 0x1be07593730>],
 'boxes': [<matplotlib.lines.Line2D at 0x1be07575c70>],
 'medians': [<matplotlib.lines.Line2D at 0x1be075939d0>],
 'fliers': [<matplotlib.lines.Line2D at 0x1be07593c70>],
 'means': []}
```



In [17]:
```python
#Remove Outliers
q_low = df["fare_amount"].quantile(0.01)
q_hi  = df["fare_amount"].quantile(0.99)

df = df[(df["fare_amount"] < q_hi) & (df["fare_amount"] > q_low)]
```

In [18]:
```python
#Check the missing values now
df.isnull().sum()
```

Out[18]:
```
Unnamed: 0           0
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

In [19]:
```python
#Time to apply learning models
from sklearn.model_selection import train_test_split
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[19], line 2
      1 #Time to apply learning models
----> 2 from sklearn.model_selection import train_test_split

ModuleNotFoundError: No module named 'sklearn'
```

In [ ]:
```
#Take x as predictor variable
x = df.drop("fare_amount", axis = 1)
#And y as target variable
y = df['fare_amount']
```

In [ ]:
```
#Necessary to apply model
x['pickup_datetime'] = pd.to_numeric(pd.to_datetime(x['pickup_datetime']))
x = x.loc[:, x.columns.str.contains('^Unnamed')]
```

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_s
```

In [ ]:
```
from sklearn.linear_model import LinearRegression
```

In [ ]:
```
lrmodel = LinearRegression()
lrmodel.fit(x_train, y_train)
```

In [ ]:
```
#Prediction
predict = lrmodel.predict(x_test)
```

In [ ]:
```
#Check Error
from sklearn.metrics import mean_squared_error
lrmodelrmse = np.sqrt(mean_squared_error(predict, y_test))
print("RMSE error for the model is ", lrmodelrmse)
```

In [ ]:
```
#Let's Apply Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
rfrmodel = RandomForestRegressor(n_estimators = 100, random_state = 101)
```

In [ ]:
```
#Fit the Forest
rfrmodel.fit(x_train, y_train)
rfrmodel_pred = rfrmodel.predict(x_test)
```

In [ ]:
```
#Errors for the forest
rfrmodel_rmse = np.sqrt(mean_squared_error(rfrmodel_pred, y_test))
print("RMSE value for Random Forest is:",rfrmodel_rmse)
```

In [ ]:

In [ ]:

In [1]:
```python
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
```

In [2]:
```python
df = pd.read_csv("./emails.csv")
```

In [3]:
```python
df.head()
```

Out[3]:

	Email No.	the	to	ect	and	for	of	a	you	hou	...	connevey	jay	valued	lay	infra
0	Email 1	0	0	1	0	0	0	2	0	0	...	0	0	0	0	
1	Email 2	8	13	24	6	6	2	102	1	27	...	0	0	0	0	
2	Email 3	0	0	1	0	0	0	8	0	0	...	0	0	0	0	
3	Email 4	0	5	22	0	5	1	51	2	10	...	0	0	0	0	
4	Email 5	7	6	17	1	5	2	57	0	9	...	0	0	0	0	

5 rows × 3002 columns

In [4]:
```python
df.isnull().sum()
```

Out[4]:
```
Email No.     0
the           0
to            0
ect           0
and           0
             ..
military      0
allowing      0
ff            0
dry           0
Prediction    0
Length: 3002, dtype: int64
```

In [5]:
```python
X = df.iloc[:,1:3001]
X
```

Out[5]:

	the	to	ect	and	for	of	a	you	hou	in	...	enhancements	connevey	jay	v
0	0	0	1	0	0	0	2	0	0	0	...	0	0	0	
1	8	13	24	6	6	2	102	1	27	18	...	0	0	0	
2	0	0	1	0	0	0	8	0	0	4	...	0	0	0	
3	0	5	22	0	5	1	51	2	10	1	...	0	0	0	
4	7	6	17	1	5	2	57	0	9	3	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
5167	2	2	2	3	0	0	32	0	0	5	...	0	0	0	
5168	35	27	11	2	6	5	151	4	3	23	...	0	0	0	
5169	0	0	1	1	0	0	11	0	0	1	...	0	0	0	
5170	2	7	1	0	2	1	28	2	0	8	...	0	0	0	
5171	22	24	5	1	6	5	148	8	2	23	...	0	0	0	

5172 rows × 3000 columns

In [6]:
```
Y = df.iloc[:,-1].values
Y
```

Out[6]:
```
array([0, 0, 0, ..., 1, 1, 0], dtype=int64)
```

In [7]:
```
train_x,test_x,train_y,test_y = train_test_split(X,Y,test_size = 0.25)
```

In [ ]:
```
svc = SVC(C=1.0,kernel='rbf',gamma='auto')
# C here is the regularization parameter. Here, L2 penalty is used(default). It is
# As C increases, model overfits.
# Kernel here is the radial basis function kernel.
# gamma (only used for rbf kernel) : As gamma increases, model overfits.
svc.fit(train_x,train_y)
y_pred2 = svc.predict(test_x)
print("Accuracy Score for SVC : ", accuracy_score(y_pred2,test_y))
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_s
```

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=7)
```

In [ ]:
```
knn.fit(X_train, y_train)
```

In [ ]:
```
print(knn.predict(X_test))
```

In [ ]:
```
print(knn.score(X_test, y_test))
```

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
sns.set()
```

In [2]:
```python
dataset = pd.read_csv('/content/Churn_Modelling.csv', index_col = 'RowNumber')
dataset.head()
```

Out[2]:

RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Bala
1	15634602	Hargrave	619	France	Female	42	2	(
2	15647311	Hill	608	Spain	Female	41	1	8380
3	15619304	Onio	502	France	Female	42	8	15966(
4	15701354	Boni	699	France	Female	39	1	(
5	15737888	Mitchell	850	Spain	Female	43	2	12551(

In [3]:
```python
#Customer ID and Surname would not be relevant as features
X_columns = dataset.columns.tolist()[2:12]
Y_columns = dataset.columns.tolist()[-1:]
print(X_columns)
print(Y_columns)
```

```
['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts',
'HasCrCard', 'IsActiveMember', 'EstimatedSalary']
['Exited']
```

In [4]:
```python
X = dataset[X_columns].values
Y = dataset[Y_columns].values
```

In [5]:
```python
#We need to encode categorical variables such as geography and gender
from sklearn.preprocessing import LabelEncoder
X_column_transformer = LabelEncoder()
X[:, 1] = X_column_transformer.fit_transform(X[:, 1])
```

In [6]:
```python
#Lets Encode gender now
X[:, 2] = X_column_transformer.fit_transform(X[:, 2])
```

We are treating countries with ordinal values(0 < 1 < 2) but they are incomparable. To solve this we can use one hot encoding. We will perform some standardization

In [7]:
```python
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

pipeline = Pipeline(
    [
```

```
        ('Categorizer', ColumnTransformer(
            [
                ("Gender Label Encoder", OneHotEncoder(categories = 'auto', drop = 
                ("Geography Label Encoder", OneHotEncoder(categories = 'auto', drop
            ],
            remainder = 'passthrough', n_jobs = 1)),
        ('Normalizer', StandardScaler())
    ]
)
```

In [8]:
```
#Standardize the features
X = pipeline.fit_transform(X)
```

In [9]:
```
#Spilt the data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_s
```

In [10]:
```
#Let us create the Neural Network
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

In [11]:
```
#Initialize ANN
classifier = Sequential()
```

In [12]:
```
#Add input layer and hidden layer
classifier.add(Dense(6, activation = 'relu', input_shape = (X_train.shape[1], )))
classifier.add(Dropout(rate = 0.1))
```

In [13]:
```
#Add second layer
classifier.add(Dense(6, activation = 'relu'))
classifier.add(Dropout(rate = 0.1))
```

In [14]:
```
#Add output layer
classifier.add(Dense(1, activation = 'sigmoid'))
```

In [15]:
```
#Let us take a look at our network
classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 6)                 72

 dropout (Dropout)           (None, 6)                 0

 dense_1 (Dense)             (None, 6)                 42

 dropout_1 (Dropout)         (None, 6)                 0

 dense_2 (Dense)             (None, 1)                 7

=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
```

In [16]:

```
#Optimize the weights
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['ac
```

In [ ]:

```
#Fitting the Neural Network
history = classifier.fit(X_train, y_train, batch_size = 32, epochs = 200, validatio
```

In [19]:

```
y_pred = classifier.predict(X_test)
print(y_pred[:5])
```

```
63/63 [==============================] - 0s 1ms/step
[[0.21353428]
 [0.3550975 ]
 [0.1884149 ]
 [0.04963601]
 [0.2057534 ]]
```

In [20]:

```
#Let us use confusion matrix with cutoff value as 0.5
y_pred = (y_pred > 0.5).astype(int)
print(y_pred[:5])
```

```
[[0]
 [0]
 [0]
 [0]
 [0]]
```

In [21]:

```
#Making the Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[1569   26]
 [ 293  112]]
```

In [22]:

```
#Accuracy of our NN
print(((cm[0][0] + cm[1][1])* 100) / len(y_test), '% of data was classified correct
```

```
84.05 % of data was classified correctly
```

Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset

In [1]:
```
    import numpy as np
    import pandas as pd
```

In [2]:
```
data = pd.read_csv('./diabetes.csv')
data.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree	Age	Outc
0	6	148	72	35	0	33.6	0.627	50	
1	1	85	66	29	0	26.6	0.351	31	
2	8	183	64	0	0	23.3	0.672	32	
3	1	89	66	23	94	28.1	0.167	21	
4	0	137	40	35	168	43.1	2.288	33	

In [4]:
```
#Check for null or missing values
data.isnull().sum()
```

Out[4]:
```
Pregnancies      0
Glucose          0
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
Pedigree         0
Age              0
Outcome          0
dtype: int64
```

In [6]:
```
#Replace zero values with mean values
for column in data.columns[1:-3]:
    data[column].replace(0, np.NaN, inplace = True)
    data[column].fillna(round(data[column].mean(skipna=True)), inplace = True)
data.head(10)
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree	Age	Outc
0	6	148.0	72.0	35.0	156.0	33.6	0.627	50	
1	1	85.0	66.0	29.0	156.0	26.6	0.351	31	
2	8	183.0	64.0	29.0	156.0	23.3	0.672	32	
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	
5	5	116.0	74.0	29.0	156.0	25.6	0.201	30	
6	3	78.0	50.0	32.0	88.0	31.0	0.248	26	
7	10	115.0	72.0	29.0	156.0	35.3	0.134	29	
8	2	197.0	70.0	45.0	543.0	30.5	0.158	53	
9	8	125.0	96.0	29.0	156.0	32.0	0.232	54	

In [7]:

```
X = data.iloc[:, :8] #Features
Y = data.iloc[:, 8:] #Predictor
```

In [22]:

```
#Perform Spliting
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_sta
```

In [23]:

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn_fit = knn.fit(X_train, Y_train.values.ravel())
knn_pred = knn_fit.predict(X_test)
```

In [24]:

```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_sco
print("Confusion Matrix")
print(confusion_matrix(Y_test, knn_pred))
print("Accuracy Score:", accuracy_score(Y_test, knn_pred))
print("Reacal Score:", recall_score(Y_test, knn_pred))
print("F1 Score:", f1_score(Y_test, knn_pred))
print("Precision Score:",precision_score(Y_test, knn_pred))
```

```
Confusion Matrix
[[88 19]
 [19 28]]
Accuracy Score: 0.7532467532467533
Reacal Score: 0.5957446808510638
F1 Score: 0.5957446808510638
Precision Score: 0.5957446808510638
```

In [ ]:

Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset. Determine thenumber of clusters using the elbow method.

In [4]:
```
import pandas as pd
import numpy as np
```

In [5]:
```
df = pd.read_csv('./sales_data_sample.csv', encoding='unicode_escape')
```

In [6]:
```
df.head
```

```
Out[6]: <bound method NDFrame.head of       ORDERNUMBER  QUANTITYORDERED  PRICEEACH  ORDER
LINENUMBER    SALES  \
0           10107               30      95.70                2  2871.00
1           10121               34      81.35                5  2765.90
2           10134               41      94.74                2  3884.34
3           10145               45      83.26                6  3746.70
4           10159               49     100.00               14  5205.27
...           ...              ...        ...              ...      ...
2818        10350               20     100.00               15  2244.40
2819        10373               29     100.00                1  3978.51
2820        10386               43     100.00                4  5417.57
2821        10397               34      62.24                1  2116.16
2822        10414               47      65.52                9  3079.44

            ORDERDATE    STATUS  QTR_ID  MONTH_ID  YEAR_ID  ...  \
0      2/24/2003 0:00   Shipped       1         2     2003  ...
1       5/7/2003 0:00   Shipped       2         5     2003  ...
2       7/1/2003 0:00   Shipped       3         7     2003  ...
3      8/25/2003 0:00   Shipped       3         8     2003  ...
4     10/10/2003 0:00   Shipped       4        10     2003  ...
...               ...       ...     ...       ...      ...  ...
2818   12/2/2004 0:00   Shipped       4        12     2004  ...
2819   1/31/2005 0:00   Shipped       1         1     2005  ...
2820    3/1/2005 0:00  Resolved       1         3     2005  ...
2821   3/28/2005 0:00   Shipped       1         3     2005  ...
2822    5/6/2005 0:00   On Hold       2         5     2005  ...

                       ADDRESSLINE1  ADDRESSLINE2           CITY STATE  \
0           897 Long Airport Avenue           NaN            NYC    NY
1                59 rue de l'Abbaye           NaN          Reims   NaN
2     27 rue du Colonel Pierre Avia           NaN          Paris   NaN
3               78934 Hillside Dr.           NaN       Pasadena    CA
4                  7734 Strong St.           NaN  San Francisco    CA
...                             ...           ...            ...   ...
2818            C/ Moralzarzal, 86           NaN         Madrid   NaN
2819                  Torikatu 38           NaN           Oulu   NaN
2820            C/ Moralzarzal, 86           NaN         Madrid   NaN
2821          1 rue Alsace-Lorraine           NaN       Toulouse   NaN
2822              8616 Spinnaker Dr.           NaN         Boston    MA

     POSTALCODE  COUNTRY TERRITORY CONTACTLASTNAME CONTACTFIRSTNAME DEALSIZE
0         10022      USA       NaN              Yu             Kwai    Small
1         51100   France      EMEA         Henriot             Paul    Small
2         75508   France      EMEA        Da Cunha           Daniel   Medium
3         90003      USA       NaN           Young            Julie   Medium
4           NaN      USA       NaN           Brown            Julie   Medium
...         ...      ...       ...             ...              ...      ...
2818      28034    Spain      EMEA          Freyre            Diego    Small
2819      90110  Finland      EMEA       Koskitalo           Pirkko   Medium
2820      28034    Spain      EMEA          Freyre            Diego   Medium
2821      31000   France      EMEA          Roulet          Annette    Small
2822      51003      USA       NaN         Yoshido             Juri   Medium

[2823 rows x 25 columns]>
```

In [7]:
```
df.info
```

```
Out[7]: <bound method DataFrame.info of       ORDERNUMBER  QUANTITYORDERED  PRICEEACH  ORD
ERLINENUMBER    SALES  \
0           10107               30      95.70                2  2871.00
1           10121               34      81.35                5  2765.90
2           10134               41      94.74                2  3884.34
3           10145               45      83.26                6  3746.70
4           10159               49     100.00               14  5205.27
...           ...              ...        ...              ...      ...
2818        10350               20     100.00               15  2244.40
2819        10373               29     100.00                1  3978.51
2820        10386               43     100.00                4  5417.57
2821        10397               34      62.24                1  2116.16
2822        10414               47      65.52                9  3079.44

            ORDERDATE    STATUS  QTR_ID  MONTH_ID  YEAR_ID  ...  \
0      2/24/2003 0:00   Shipped       1         2     2003  ...
1       5/7/2003 0:00   Shipped       2         5     2003  ...
2       7/1/2003 0:00   Shipped       3         7     2003  ...
3      8/25/2003 0:00   Shipped       3         8     2003  ...
4     10/10/2003 0:00   Shipped       4        10     2003  ...
...               ...       ...     ...       ...      ...  ...
2818   12/2/2004 0:00   Shipped       4        12     2004  ...
2819   1/31/2005 0:00   Shipped       1         1     2005  ...
2820    3/1/2005 0:00  Resolved       1         3     2005  ...
2821   3/28/2005 0:00   Shipped       1         3     2005  ...
2822    5/6/2005 0:00   On Hold       2         5     2005  ...

                       ADDRESSLINE1  ADDRESSLINE2           CITY STATE  \
0           897 Long Airport Avenue           NaN            NYC    NY
1                59 rue de l'Abbaye           NaN          Reims   NaN
2      27 rue du Colonel Pierre Avia           NaN          Paris   NaN
3               78934 Hillside Dr.           NaN       Pasadena    CA
4                   7734 Strong St.           NaN  San Francisco    CA
...                             ...           ...            ...   ...
2818           C/ Moralzarzal, 86           NaN         Madrid   NaN
2819                  Torikatu 38           NaN           Oulu   NaN
2820           C/ Moralzarzal, 86           NaN         Madrid   NaN
2821         1 rue Alsace-Lorraine           NaN       Toulouse   NaN
2822             8616 Spinnaker Dr.           NaN         Boston    MA

     POSTALCODE  COUNTRY TERRITORY CONTACTLASTNAME CONTACTFIRSTNAME DEALSIZE
0         10022      USA       NaN              Yu             Kwai    Small
1         51100   France      EMEA         Henriot             Paul    Small
2         75508   France      EMEA        Da Cunha           Daniel   Medium
3         90003      USA       NaN           Young            Julie   Medium
4           NaN      USA       NaN           Brown            Julie   Medium
...         ...      ...       ...             ...              ...      ...
2818      28034    Spain      EMEA          Freyre            Diego    Small
2819      90110  Finland      EMEA       Koskitalo           Pirkko   Medium
2820      28034    Spain      EMEA          Freyre            Diego   Medium
2821      31000   France      EMEA          Roulet          Annette    Small
2822      51003      USA       NaN         Yoshido             Juri   Medium

[2823 rows x 25 columns]>
```

In [8]:
```python
#Columns to Remove
to_drop = ['ADDRESSLINE1', 'ADDRESSLINE2', 'STATE', 'POSTALCODE', 'PHONE']
df = df.drop(to_drop, axis=1)
```

In [9]:
```python
#Check for null values
df.isnull().sum()
```

Out[9]:
```
ORDERNUMBER            0
QUANTITYORDERED        0
PRICEEACH              0
ORDERLINENUMBER        0
SALES                  0
ORDERDATE              0
STATUS                 0
QTR_ID                 0
MONTH_ID               0
YEAR_ID                0
PRODUCTLINE            0
MSRP                   0
PRODUCTCODE            0
CUSTOMERNAME           0
CITY                   0
COUNTRY                0
TERRITORY           1074
CONTACTLASTNAME        0
CONTACTFIRSTNAME       0
DEALSIZE               0
dtype: int64
```

In [10]:
```python
#Bhai bhai look at territory
#But territory does not have significant impact on analysis, let it be
```

In [11]:
```python
df.dtypes
```

Out[11]:
```
ORDERNUMBER           int64
QUANTITYORDERED       int64
PRICEEACH           float64
ORDERLINENUMBER       int64
SALES               float64
ORDERDATE            object
STATUS               object
QTR_ID                int64
MONTH_ID              int64
YEAR_ID               int64
PRODUCTLINE          object
MSRP                  int64
PRODUCTCODE          object
CUSTOMERNAME         object
CITY                 object
COUNTRY              object
TERRITORY            object
CONTACTLASTNAME      object
CONTACTFIRSTNAME     object
DEALSIZE             object
dtype: object
```

In [12]:
```python
#ORDERDATE Should be in date time
df['ORDERDATE'] = pd.to_datetime(df['ORDERDATE'])
```

In [13]:
```python
#We need to create some features in order to create cluseters
#Recency: Number of days between customer's latest order and today's date
#Frequency : Number of purchases by the customers
#MonetaryValue : Revenue generated by the customers
import datetime as dt
snapshot_date = df['ORDERDATE'].max() + dt.timedelta(days = 1)
df_RFM = df.groupby(['CUSTOMERNAME']).agg({
    'ORDERDATE' : lambda x : (snapshot_date - x.max()).days,
    'ORDERNUMBER' : 'count',
    'SALES' : 'sum'
})

#Rename the columns
df_RFM.rename(columns = {
    'ORDERDATE' : 'Recency',
    'ORDERNUMBER' : 'Frequency',
    'SALES' : 'MonetaryValue'
}, inplace=True)
```

In [14]:
```python
df_RFM.head()
```

Out[14]:

	Recency	Frequency	MonetaryValue
CUSTOMERNAME			
AV Stores, Co.	196	51	157807.81
Alpha Cognac	65	20	70488.44
Amica Models & Co.	265	26	94117.26
Anna's Decorations, Ltd	84	46	153996.13
Atelier graphique	188	7	24179.96

In [16]:
```python
# Divide into segments
# We create 4 quartile ranges
df_RFM['M'] = pd.qcut(df_RFM['MonetaryValue'], q = 4, labels = range(1,5))
df_RFM['R'] = pd.qcut(df_RFM['Recency'], q = 4, labels = list(range(4,0,-1)))
df_RFM['F'] = pd.qcut(df_RFM['Frequency'], q = 4, labels = range(1,5))

df_RFM.head()
```

Out[16]:

CUSTOMERNAME	Recency	Frequency	MonetaryValue	M	R	F
AV Stores, Co.	196	51	157807.81	4	2	4
Alpha Cognac	65	20	70488.44	2	4	2
Amica Models & Co.	265	26	94117.26	3	1	2
Anna's Decorations, Ltd	84	46	153996.13	4	3	4
Atelier graphique	188	7	24179.96	1	2	1

In [17]:

```
#Create another column for RFM score
df_RFM['RFM_Score'] = df_RFM[['R', 'M', 'F']].sum(axis=1)
df_RFM.head()
```

Out[17]:

CUSTOMERNAME	Recency	Frequency	MonetaryValue	M	R	F	RFM_Score
AV Stores, Co.	196	51	157807.81	4	2	4	10
Alpha Cognac	65	20	70488.44	2	4	2	8
Amica Models & Co.	265	26	94117.26	3	1	2	6
Anna's Decorations, Ltd	84	46	153996.13	4	3	4	11
Atelier graphique	188	7	24179.96	1	2	1	4

We create levels for our Customers

RFM Score > 10 : High Value Customers

RFM Score < 10 and RFM Score >= 6 : Mid Value Customers

RFM Score < 6 : Low Value Customers

In [20]:

```
def rfm_level(df):
    if bool(df['RFM_Score'] >= 10):
        return 'High Value Customer'

    elif bool(df['RFM_Score'] < 10) and bool(df['RFM_Score'] >= 6):
        return 'Mid Value Customer'
    else:
        return 'Low Value Customer'
df_RFM['RFM_Level'] = df_RFM.apply(rfm_level, axis = 1)
df_RFM.head()
```

Out[20]:

CUSTOMERNAME	Recency	Frequency	MonetaryValue	M	R	F	RFM_Score	RFM_Level
AV Stores, Co.	196	51	157807.81	4	2	4	10	High Value Customer
Alpha Cognac	65	20	70488.44	2	4	2	8	Mid Value Customer
Amica Models & Co.	265	26	94117.26	3	1	2	6	Mid Value Customer
Anna's Decorations, Ltd	84	46	153996.13	4	3	4	11	High Value Customer
Atelier graphique	188	7	24179.96	1	2	1	4	Low Value Customer

In [21]:

```
# Time to perform KMeans
data = df_RFM[['Recency', 'Frequency', 'MonetaryValue']]
data.head()
```

Out[21]:

CUSTOMERNAME	Recency	Frequency	MonetaryValue
AV Stores, Co.	196	51	157807.81
Alpha Cognac	65	20	70488.44
Amica Models & Co.	265	26	94117.26
Anna's Decorations, Ltd	84	46	153996.13
Atelier graphique	188	7	24179.96

In [22]:

```
# Our data is skewed we must remove it by performing log transformation
data_log = np.log(data)
data_log.head()
```

Out[22]:

CUSTOMERNAME	Recency	Frequency	MonetaryValue
AV Stores, Co.	5.278115	3.931826	11.969133
Alpha Cognac	4.174387	2.995732	11.163204
Amica Models & Co.	5.579730	3.258097	11.452297
Anna's Decorations, Ltd	4.430817	3.828641	11.944683
Atelier graphique	5.236442	1.945910	10.093279

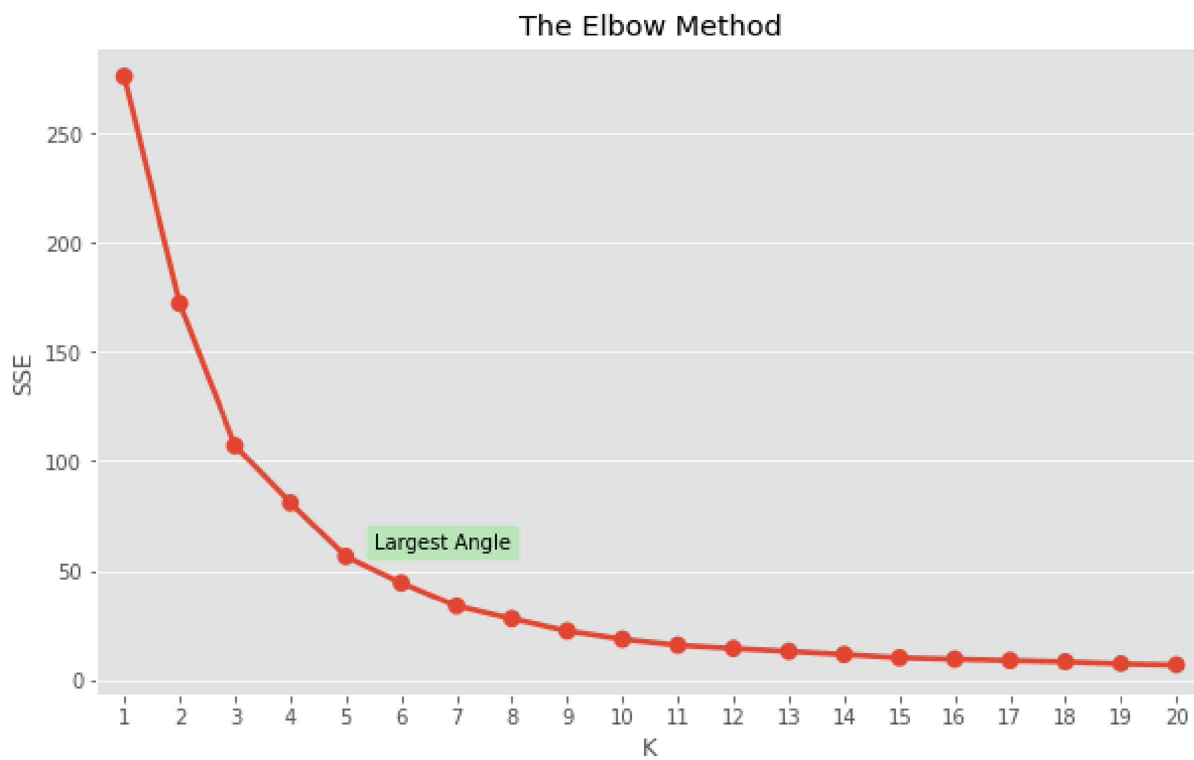
In [25]:
```
#Standardization
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(data_log)
data_normalized = scaler.transform(data_log)
data_normalized = pd.DataFrame(data_normalized, index = data_log.index, columns=dat
data_normalized.describe().round(2)
```

Out[25]:

	Recency	Frequency	MonetaryValue
count	92.00	92.00	92.00
mean	0.00	-0.00	0.00
std	1.01	1.01	1.01
min	-3.51	-3.67	-3.82
25%	-0.24	-0.41	-0.39
50%	0.37	0.06	-0.04
75%	0.53	0.45	0.52
max	1.12	4.03	3.92

In [28]:
```
#Fit KMeans and use elbow method to choose the number of clusters
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans

sse = {}

for k in range(1, 21):
    kmeans = KMeans(n_clusters = k, random_state = 1)
    kmeans.fit(data_normalized)
    sse[k] = kmeans.inertia_
```

In [31]:
```
plt.figure(figsize=(10,6))
plt.title('The Elbow Method')

plt.xlabel('K')
plt.ylabel('SSE')
plt.style.use('ggplot')

sns.pointplot(x=list(sse.keys()), y = list(sse.values()))
plt.text(4.5, 60, "Largest Angle", bbox = dict(facecolor = 'lightgreen', alpha = 0.
plt.show()
```



In [32]:

```
# 5 number of clusters seems good
kmeans = KMeans(n_clusters=5, random_state=1)
kmeans.fit(data_normalized)
cluster_labels = kmeans.labels_

data_rfm = data.assign(Cluster = cluster_labels)
data_rfm.head()
```

Out[32]:

	Recency	Frequency	MonetaryValue	Cluster
CUSTOMERNAME				
AV Stores, Co.	196	51	157807.81	3
Alpha Cognac	65	20	70488.44	0
Amica Models & Co.	265	26	94117.26	0
Anna's Decorations, Ltd	84	46	153996.13	3
Atelier graphique	188	7	24179.96	2

Mini Project

Start here! Predict survival on the Titanic and get familiar with ML basics

In [64]:
```
import numpy as np
import pandas as pd
```

In [65]:
```
train_data=pd.read_csv("Datasets/train.csv")
test=pd.read_csv("Datasets/test.csv")
```

In [66]:
```
train=train_data
train.head()
```

Out[66]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

In [67]:
```
test.head()
```

Out[67]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	

In [68]:

```
train.shape
```

Out[68]: (891, 12)

In [69]:

```
test.shape
```

Out[69]: (418, 11)

In [70]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [71]:
```
## sum of null values

train.isnull().sum()
```

Out[71]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Filling null values in Training dataset

In [72]:
```
train['Age'].fillna(train['Age'].mean(),inplace=True)
```

In [73]:
```
train['Cabin'].fillna(train['Cabin'].mode().values[0],inplace=True)
```

In [74]:
```
train['Embarked'].fillna(train['Embarked'].mode().values[0],inplace=True)
```

In [75]:
```
train.isnull().sum()
```

Out[75]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

Filling null values for Test datasets

In [76]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

In [77]:
```
test.isnull().sum()
```

Out[77]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [78]:
```
test['Age'].fillna(test['Age'].mean(),inplace=True)
```

In [79]:
```
test['Fare'].fillna(test['Fare'].mean(),inplace=True)
```

In [80]:
```
test['Cabin'].fillna(test['Cabin'].mode().values[0],inplace=True)
```

In [81]:
```
test.isnull().sum()
```

Out[81]:
```
PassengerId    0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

In [82]:
```
train.head()
```

Out[82]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.283
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.050

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

Out[83]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	B57 B59 B63 B66	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	B57 B59 B63 B66	
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	B57 B59 B63 B66	
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	B57 B59 B63 B66	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	B57 B59 B63 B66	

In [84]:
```
train=train.drop(columns=['Name'],axis=1)
```

In [85]:
```
train=train.drop(columns=['Ticket'])
```

In [86]:
```
train=train.drop(columns=['Cabin'])
```

In [87]:
```
test=test.drop(columns=['Name'])
```

In [88]:
```
test=test.drop(columns=['Ticket'])
```

In [89]:
```
test=test.drop(columns=['Cabin'])
```

In [90]:
```
train.head()
```

Out[90]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.2500	S
1	2	1	1	female	38.0	1	0	71.2833	C
2	3	1	3	female	26.0	0	0	7.9250	S
3	4	1	1	female	35.0	1	0	53.1000	S
4	5	0	3	male	35.0	0	0	8.0500	S

In [91]:
```
test.head()
```

Out[91]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	892	3	male	34.5	0	0	7.8292	Q
1	893	3	female	47.0	1	0	7.0000	S
2	894	2	male	62.0	0	0	9.6875	Q
3	895	3	male	27.0	0	0	8.6625	S
4	896	3	female	22.0	1	1	12.2875	S

Now LabelEncoding

In [92]:
```
from sklearn.preprocessing import LabelEncoder
```

In [93]:
```
encoder=LabelEncoder()
```

In [94]:
```
train['Sex']=encoder.fit_transform(train['Sex'])
```

In [95]:
```
train['Embarked']=encoder.fit_transform(train['Embarked'])
```

In [96]:
```
train.head()
```

Out[96]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	1	22.0	1	0	7.2500	2
1	2	1	1	0	38.0	1	0	71.2833	0
2	3	1	3	0	26.0	0	0	7.9250	2
3	4	1	1	0	35.0	1	0	53.1000	2
4	5	0	3	1	35.0	0	0	8.0500	2

In [97]:
```
test['Sex']=encoder.fit_transform(test['Sex'])
```

In [98]:
```
test['Embarked']=encoder.fit_transform(test['Embarked'])
```

In [99]:
```
test.head()
```

Out[99]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	892	3	1	34.5	0	0	7.8292	1
1	893	3	0	47.0	1	0	7.0000	2
2	894	2	1	62.0	0	0	9.6875	1
3	895	3	1	27.0	0	0	8.6625	2
4	896	3	0	22.0	1	1	12.2875	2

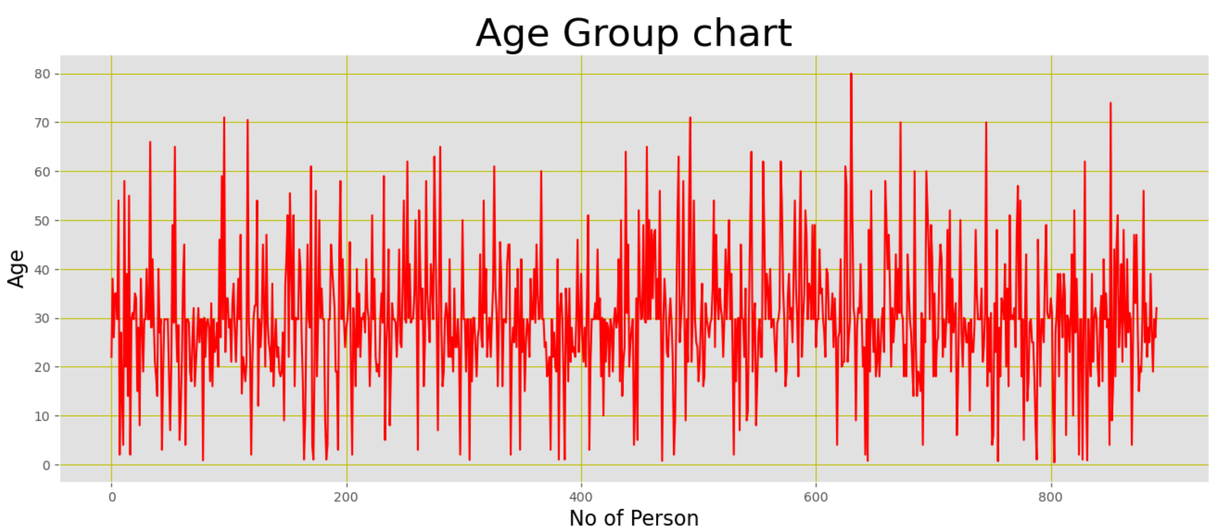
Data Visualization

In [100…

```
import matplotlib.pyplot as plt
from matplotlib import style
```

In [101…

```
plt.figure(figsize=(16,6))
style.use("ggplot")
plt.plot(train['Age'],color='r')
plt.xlabel("No of Person",fontsize=16)
plt.ylabel("Age",fontsize=16)
plt.title("Age Group chart",fontsize=30)
plt.grid(True,color="y")
plt.show()
```



Feature Selection

In [102…

```
train.head()
```

Out[102…

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	1	22.0	1	0	7.2500	2
1	2	1	1	0	38.0	1	0	71.2833	0
2	3	1	3	0	26.0	0	0	7.9250	2
3	4	1	1	0	35.0	1	0	53.1000	2
4	5	0	3	1	35.0	0	0	8.0500	2

In [103…

```
test.head()
```

Out[103…

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	892	3	1	34.5	0	0	7.8292	1
1	893	3	0	47.0	1	0	7.0000	2
2	894	2	1	62.0	0	0	9.6875	1
3	895	3	1	27.0	0	0	8.6625	2
4	896	3	0	22.0	1	1	12.2875	2

Correlation

In [104…

```
train.corr()
```

Out[104…

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch
PassengerId	1.000000	-0.005007	-0.035144	0.042939	0.033207	-0.057527	-0.001652
Survived	-0.005007	1.000000	-0.338481	-0.543351	-0.069809	-0.035322	0.081629
Pclass	-0.035144	-0.338481	1.000000	0.131900	-0.331339	0.083081	0.018443
Sex	0.042939	-0.543351	0.131900	1.000000	0.084153	-0.114631	-0.245489
Age	0.033207	-0.069809	-0.331339	0.084153	1.000000	-0.232625	-0.179191
SibSp	-0.057527	-0.035322	0.083081	-0.114631	-0.232625	1.000000	0.414838
Parch	-0.001652	0.081629	0.018443	-0.245489	-0.179191	0.414838	1.000000
Fare	0.012658	0.257307	-0.549500	-0.182333	0.091566	0.159651	0.216225
Embarked	0.013128	-0.167675	0.162098	0.108262	-0.026749	0.068230	0.039798

In [105…

```
x=['Pclass','Sex','Age','Fare','Embarked']
```

In [106…

```
y=['Survived']
```

In [107…
```
x=train[x].values
```

In [108…
```
y=train[y].values
```

In [109…
```
x.shape
```

Out[109…
```
(891, 5)
```

In [110…
```
y.shape
```

Out[110…
```
(891, 1)
```

In [111…
```
x_test=['Pclass','Sex','Age','Fare','Embarked']
```

In [112…
```
 x_test=test[x_test].values
```

In [113…
```
x_test.shape
```

Out[113…
```
(418, 5)
```

In [ ]:
```

```

In [ ]:
```

```

Deploying on Machine Learning Model

In [114…
```
from sklearn.linear_model import LogisticRegression
```

In [115…
```
logr=LogisticRegression()
```

In [116…
```
logr.fit(x,y)
```

```
C:\Users\Dell\anaconda3\lib\site-packages\sklearn\utils\validation.py:1143: DataConv
ersionWarning: A column-vector y was passed when a 1d array was expected. Please cha
nge the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[116…
```
▾ LogisticRegression
LogisticRegression()
```

In [117…
```
y_pred=logr.predict(x_test)
```

In [118…
```
y_pred
```

Out[118…

```
array([0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0],
      dtype=int64)
```

In [119…

```
sample=pd.read_csv("Datasets/gender_submission.csv")
```

In [120…

```
sample.shape
```

Out[120…

```
(418, 2)
```

In [121…

```
sample.head()
```

Out[121…

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1

In [122…

```
Survived=y_pred
```

In [123…

```
sample=sample.drop(columns=("Survived"))
```

In [124…

```
sample['Survived']=Survived
```

In [125…

```
sample.head()
```

Out[125…

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	1

In [126…

```
sample.to_csv("logr_03.csv",index=False)
```

In [ ]: