ASSIGNMENT NO. : 1

AIM: Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks:
1. Pre-process the dataset.
2. Identify outliers.
3. Check the correlation.
4. Implement linear regression and random forest regression models.
5. Evaluate the models and compare their respective scores like R2, RMSE, etc.

Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset

INTRODUCTION:

❖ DATA PRE-PROCESSING IN MACHINE LEARNING

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves below steps:

1. Getting the dataset
2. Importing libraries
3. Importing datasets
4. Finding Missing Data
5. Encoding Categorical Data
6. Splitting dataset into training and test set
7. Feature scaling

❖ OUTLIERS IN MACHINE LEARNING

In simple terms, an outlier is an extremely high or extremely low data point relative to the nearest data point and the rest of the neighbouring co-existing values in a data graph or dataset you're working with. Outliers are extreme values that stand out greatly from the overall pattern of values in a dataset or graph.

❖ CORRELATION IN MACHINE LEARNING

Data correlation is the way in which one set of data may correspond to another set. In ML, think of how your features correspond with your output. For example, the image below visualizes a dataset of brain size versus body size. Notice that as the body size increases, so does the brain size.

❖ LINEAR REGRESSION

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.

❖ RANDOM FOREST REGRESSION MODELS

Random Forest Regression is a supervised learning algorithm that uses ensemble learning method for regression. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model.

❖ CODE :

```
In [40]: import numpy as py
         import pandas as pd

In [41]: pd = pd.read_csv('uber.csv')

In [42]: pd.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 [43]:
```
pd.head(10)
```

Out[43]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085
5	44470845	2011-02-12 02:27:09.0000006	4.9	2011-02-12 02:27:09 UTC	-73.969019	40.755910
6	48725865	2014-10-12 07:04:00.0000002	24.5	2014-10-12 07:04:00 UTC	-73.961447	40.693965
7	44195482	2012-12-11 13:52:00.00000029	2.5	2012-12-11 13:52:00 UTC	0.000000	0.000000
8	15822268	2012-02-17 09:32:00.00000043	9.7	2012-02-17 09:32:00 UTC	-73.975187	40.745767
9	50611056	2012-03-29 19:06:00.000000273	12.5	2012-03-29 19:06:00 UTC	-74.001065	40.741787

In [44]:
```
pd.isnull().sum()
```

Out[44]:
```
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 [45]:
```
pd['dropoff_longitude'].fillna(pd['dropoff_longitude'].mean(),inplace=True)
```

In [46]:
```
pd['dropoff_latitude'].fillna(pd['dropoff_latitude'].mean(),inplace=True)
```

In [47]:
```
x = pd[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']
y = pd['fare_amount']
```

In [48]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

```
In [49]: x_train , x_test , y_train , y_test = train_test_split(x , y , test_size=0.30)

In [50]: from sklearn.preprocessing import StandardScaler

In [51]: scaler = StandardScaler()

In [52]: x_train = scaler.fit_transform(x_train)
         x_test = scaler.fit_transform(x_test)

In [53]: model = LinearRegression()
         model.fit(x_train , y_train)

Out[53]: LinearRegression()

In [54]: predictions  = model.predict(x_test)

In [55]: from sklearn.metrics import r2_score , mean_squared_error

In [56]: print("R2 score" , r2_score(y_test , predictions))
         print("Root mean squared error : " , mean_squared_error(y_test , predictions))

R2 score 0.00012207990674539815
Root mean squared error :  97.69408703176353
```

CONCLUSION: We have successfully studied about pre-processing the dataset, outliers, correlation, linear regression and random forest regression models.

ASSIGNMENT NO. : 2

AIM: Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance.
Dataset link: The emails.csv dataset on the Kaggle https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv

INTRODUCTION:

❖ BINARY CLASSIFICATION

Binary classification refers to those classification tasks that have two class labels. Examples include:

- Email spam detection (spam or not).
- Churn prediction (churn or not).
- Conversion prediction (buy or not).

Typically, binary classification tasks involve one class that is the normal state and another class that is the abnormal state.

For example "*not spam*" is the normal state and "*spam*" is the abnormal state. Another example is "*cancer not detected*" is the normal state of a task that involves a medical test and "*cancer detected*" is the abnormal state.
The class for the normal state is assigned the class label 0 and the class with the abnormal state is assigned the class label 1.
It is common to model a binary classification task with a model that predicts a Bernoulli probability distribution for each example.
The Bernoulli distribution is a discrete probability distribution that covers a case where an event will have a binary outcome as either a 0 or 1. For classification, this means that the model predicts a probability of an example belonging to class 1, or the abnormal state.

Popular algorithms that can be used for binary classification include:

- Logistic Regression
- k-Nearest Neighbors
- Decision Trees
- Support Vector Machine
- Naive Bayes

Some algorithms are specifically designed for binary classification and do not natively support more than two classes; examples include Logistic Regression and Support Vector Machines.

❖ K-NEAREST NEIGHBORS (KNN)

KNN is a simple, supervised machine learning (ML) algorithm that can be used for classification or regression tasks - and is also frequently used in missing value imputation. It is based on the idea that the observations closest to a given data point are the most "similar" observations in a data set, and we can therefore classify unforeseen points based on the values of the closest existing points. By choosing *K*, the user can select the number of nearby observations to use in the algorithm.

❖ SUPPORT VECTOR MACHINE CLASSIFICATION

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

In [21]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score ,classification_report
from sklearn.svm import SVC
```

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

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

Out[3]:

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

5 rows × 3002 columns

In [4]:
```
df.columns
```

Out[4]:
```
Index(['Email No.', 'the', 'to', 'ect', 'and', 'for', 'of', 'a', 'you', 'hou',
       ...
       'connevey', 'jay', 'valued', 'lay', 'infrastructure', 'military',
       'allowing', 'ff', 'dry', 'Prediction'],
      dtype='object', length=3002)
```

In [5]:
```
df.drop(columns=['Email No.'], inplace=True)
```

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

Out[6]:
```
(5172, 3001)
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
the            0
to             0
ect            0
and            0
```

```
for           0
               ..
military      0
allowing      0
ff            0
dry           0
Prediction    0
Length: 3001, dtype: int64
```

In [8]:

```
df.describe()
```

Out[8]:

	the	to	ect	and	for	of	a
count	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000
mean	6.640565	6.188128	5.143852	3.075599	3.124710	2.627030	55.517401
std	11.745009	9.534576	14.101142	6.045970	4.680522	6.229845	87.574172
min	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	1.000000	1.000000	0.000000	1.000000	0.000000	12.000000
50%	3.000000	3.000000	1.000000	1.000000	2.000000	1.000000	28.000000
75%	8.000000	7.000000	4.000000	3.000000	4.000000	2.000000	62.250000
max	210.000000	132.000000	344.000000	89.000000	47.000000	77.000000	1898.000000

8 rows × 3001 columns

In [12]:

```
X=df.iloc[:, :3000]
y=df.iloc[:, -1]
```

In [13]:

```
X.shape , y.shape
```

Out[13]:

```
((5172, 3000), (5172,))
```

In [14]:

```
X.head()
```

Out[14]:

	the	to	ect	and	for	of	a	you	hou	in	...	enhancements	connevey	jay	valued	lay
0	0	0	1	0	0	0	2	0	0	0	...	0	0	0	0	0
1	8	13	24	6	6	2	102	1	27	18	...	0	0	0	0	0
2	0	0	1	0	0	0	8	0	0	4	...	0	0	0	0	0
3	0	5	22	0	5	1	51	2	10	1	...	0	0	0	0	0
4	7	6	17	1	5	2	57	0	9	3	...	0	0	0	0	0

5 rows × 3000 columns

In [16]:

```
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.15, random_state
```

```
In [17]: knn = KNeighborsClassifier(n_neighbors=2)
         knn.fit(X_train, y_train)
         y_pred= knn.predict(X_test)

In [18]: accuracy_score(y_test, y_pred)

Out[18]: 0.8878865979381443

In [20]: print(classification_report(y_pred, y_test))

              precision    recall  f1-score   support

           0       0.95      0.90      0.92       579
           1       0.74      0.85      0.79       197

    accuracy                           0.89       776
   macro avg       0.85      0.88      0.86       776
weighted avg       0.90      0.89      0.89       776

In [22]: clf = SVC(kernel='linear')
         clf.fit(X_train, y_train)
         y_pred2 = clf.predict(X_test)

In [24]: accuracy_score(y_test, y_pred2)

Out[24]: 0.9690721649484536

In [25]: print(classification_report(y_pred2, y_test))

              precision    recall  f1-score   support

           0       0.98      0.97      0.98       554
           1       0.94      0.95      0.95       222

    accuracy                           0.97       776
   macro avg       0.96      0.96      0.96       776
weighted avg       0.97      0.97      0.97       776
```

CONCLUSION: We have successfully studied about binary classification model and its two states- Normal State and Abnormal State.

ASSIGNMENT NO. : 3

AIM: Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.
Dataset Description: The case study is from an open-source dataset from Kaggle. The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc.
Link to the Kaggle project: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling
Perform following steps:
1. Read the dataset.
2. Distinguish the feature and target set and divide the data set into training and test sets.
3. Normalize the train and test data.
4. Initialize and build the model. Identify the points of improvement and implement the same.
5. Print the accuracy score and confusion matrix (5 points).

INTRODUCTION:

❖ NORMALIZATION IN MACHINE LEARNING:

Normalization is one of the most frequently used data preparation techniques, which helps us to change the values of numeric columns in the dataset to use a common scale. Normalization is a scaling technique in Machine Learning applied during data preparation to change the values of numeric columns in the dataset to use a common scale. It is not necessary for all datasets in a model. It is required only when features of machine learning models have different ranges. Mathematically, we can calculate normalization with the below

Xn = (X - Xminimum) / (Xmaximum - Xminimum)

Xn = Value of Normalization Xmaximum = Maximum value of a feature Xminimum = Minimum value of a feature

❖ ACCURACY SCORE

Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives

❖ CONFUSION MATRIX IN MACHINE LEARNING

The confusion matrix is a matrix used to determine the performance of the classification models for a givenset of test data. It can only be determined if the true values for test data are known. The matrix itself can be easily understood, but the related terminologies may be confusing. Since it shows the errors in the model performance in the form of a matrix, hence also known as an error matrix. Some features of Confusion matrix are given below:

- For the 2 prediction classes of classifiers, the matrix is of 2*2 table, for 3 classes, it is 3*3 table, andso on
- The matrix is divided into two dimensions, that are predicted values and actual values along with thetotal number of predictions.

Predicted values are those values, which are predicted by the model, and actual values are the true values for the given observations:

n = total predictions	Actual: No	Actual: Yes
Predicted: No	True Negative	False Positive
Predicted: Yes	False Negative	True Positive

In [46]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt #Importing the libraries
```

In [47]:
```python
df = pd.read_csv("Churn_Modelling.csv")
```

Preprocessing.

In [48]:
```python
df.head()
```

Out[48]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

In [49]:
```python
df.shape
```

Out[49]:
```
(10000, 14)
```

In [50]:
```python
df.describe()
```

Out[50]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

In [51]:
```python
df.isnull()
```

Out[51]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	False	False	False	False	False	False	False	False	False	False	False	False	False	False
9996	False	False	False	False	False	False	False	False	False	False	False	False	False	False
9997	False	False	False	False	False	False	False	False	False	False	False	False	False	False
9998	False	False	False	False	False	False	False	False	False	False	False	False	False	False
9999	False	False	False	False	False	False	False	False	False	False	False	False	False	False

10000 rows × 14 columns

In [52]:
```python
df.isnull().sum()
```

Out[52]:
```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

In [53]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          10000 non-null  object
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

In [54]:
```
df.dtypes
```

Out[54]:
```
RowNumber            int64
CustomerId           int64
Surname             object
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

In [55]:
```
df.columns
```

Out[55]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [56]:
```
df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1) #Dropping the unnecessary columns
```

In [57]:
```
df.head()
```

Out[57]:

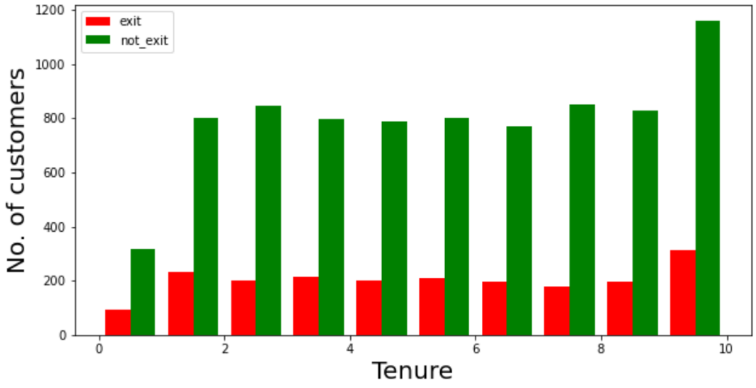
	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

Visualization

In [101…
```
def visualization(x, y, xlabel):
    plt.figure(figsize=(10,5))
    plt.hist([x, y], color=['red', 'green'], label = ['exit', 'not_exit'])
    plt.xlabel(xlabel,fontsize=20)
    plt.ylabel("No. of customers", fontsize=20)
    plt.legend()
```

In [102…
```
df_churn_exited = df[df['Exited']==1]['Tenure']
df_churn_not_exited = df[df['Exited']==0]['Tenure']
```

In [103…
```
visualization(df_churn_exited, df_churn_not_exited, "Tenure")
```

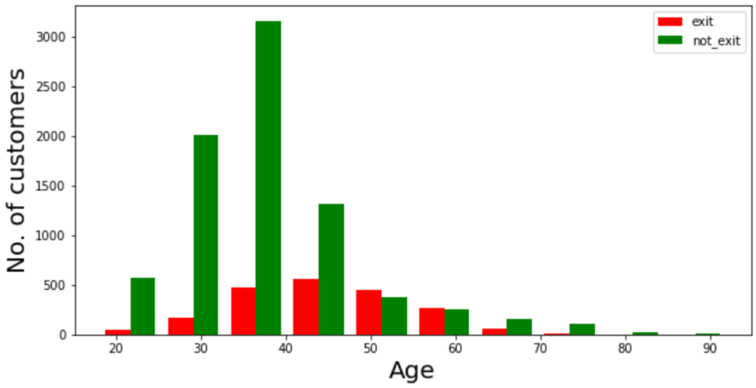


In [105…

```
df_churn_exited2 = df[df['Exited']==1]['Age']
df_churn_not_exited2 = df[df['Exited']==0]['Age']
```

In [106…

```
visualization(df_churn_exited2, df_churn_not_exited2, "Age")
```



Converting the Categorical Variables

In [59]:

```
X = df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary']]
states = pd.get_dummies(df['Geography'],drop_first = True)
gender = pd.get_dummies(df['Gender'],drop_first = True)
```

In [61]:

```
df = pd.concat([df,gender,states], axis = 1)
```

Splitting the training and testing Dataset

In [62]:

```
df.head()
```

Out[62]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Male	Germany	Spain
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1	0	0	0
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0	0	0	1
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1	0	0	0
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0	0	0	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0	0	0	1

In [63]:

```
X = df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','Male','Germany','Spain']]
```

In [64]:

```
y = df['Exited']
```

In [65]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.30)
```

Normalizing the values with mean as 0 and Standard Deviation as 1

In [66]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [67]:

```
X_train  = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [68]:

```
X_train
```

Out[68]:
```
array([[ 4.56838557e-01, -9.45594735e-01,  1.58341939e-03, ...,
         9.13181783e-01, -5.81969145e-01, -5.73611200e-01],
       [-2.07591864e-02, -2.77416637e-01,  3.47956411e-01, ...,
        -1.09507222e+00, -5.81969145e-01,  1.74334114e+00],
       [-1.66115021e-01,  1.82257167e+00, -1.38390855e+00, ...,
        -1.09507222e+00, -5.81969145e-01, -5.73611200e-01],
       ...,
       [-3.63383654e-01, -4.68324665e-01,  1.73344838e+00, ...,
         9.13181783e-01, -5.81969145e-01, -5.73611200e-01],
       [ 4.67221117e-01, -1.42286480e+00,  1.38707539e+00, ...,
         9.13181783e-01, -5.81969145e-01,  1.74334114e+00],
       [-8.82511636e-01,  2.95307447e-01, -6.91162564e-01, ...,
         9.13181783e-01, -5.81969145e-01, -5.73611200e-01]])
```

In [69]:
```python
X_test
```

Out[69]:
```
array([[ 3.63395520e-01,  1.99853433e-01,  1.58341939e-03, ...,
         9.13181783e-01, -5.81969145e-01, -5.73611200e-01],
       [-4.15243057e-02,  4.86215475e-01,  1.58341939e-03, ...,
        -1.09507222e+00, -5.81969145e-01,  1.74334114e+00],
       [-1.87923736e+00, -3.72870651e-01, -1.38390855e+00, ...,
         9.13181783e-01, -5.81969145e-01, -5.73611200e-01],
       ...,
       [-6.02182526e-01, -5.63778679e-01, -1.73028154e+00, ...,
        -1.09507222e+00, -5.81969145e-01, -5.73611200e-01],
       [ 1.51585964e+00, -6.59232693e-01,  1.73344838e+00, ...,
         9.13181783e-01, -5.81969145e-01, -5.73611200e-01],
       [-5.19122049e-01,  1.04399419e-01,  1.73344838e+00, ...,
         9.13181783e-01, -5.81969145e-01, -5.73611200e-01]])
```

Building the Classifier Model using Keras

In [70]:
```python
import keras #Keras is the wrapper on the top of tenserflow
#Can use Tenserflow as well but won't be able to understand the errors initially.
```

In [71]:
```python
from keras.models import Sequential #To create sequential neural network
from keras.layers import Dense #To create hidden layers
```

In [72]:
```python
classifier = Sequential()
```

In [74]:
```python
#To add the layers
#Dense helps to contruct the neurons
#Input Dimension means we have 11 features
# Units is to create the hidden layers
#Uniform helps to distribute the weight uniformly
classifier.add(Dense(activation = "relu",input_dim = 11,units = 6,kernel_initializer = "uniform"))
```

In [75]:
```python
classifier.add(Dense(activation = "relu",units = 6,kernel_initializer = "uniform"))   #Adding second hidden layers
```

In [76]:
```python
classifier.add(Dense(activation = "sigmoid",units = 1,kernel_initializer = "uniform")) #Final neuron will be having siigmoid function
```

In [77]:
```python
classifier.compile(optimizer="adam",loss = 'binary_crossentropy',metrics = ['accuracy']) #To compile the Artificial Neural Network. Ussed B
```

In [79]:
```python
classifier.summary() #3 layers created. 6 neurons in 1st,6neurons in 2nd layer and 1 neuron in last
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 6)                 72
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 42
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 7
=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
```

In [89]:
```python
classifier.fit(X_train,y_train,batch_size=10,epochs=50) #Fitting the ANN to training dataset
```

```
Epoch 1/50
700/700 [==============================] - 0s 674us/step - loss: 0.4293 - accuracy: 0.7947
Epoch 2/50
700/700 [==============================] - 0s 647us/step - loss: 0.4239 - accuracy: 0.7947
Epoch 3/50
700/700 [==============================] - 0s 657us/step - loss: 0.4203 - accuracy: 0.8067
Epoch 4/50
700/700 [==============================] - 0s 664us/step - loss: 0.4167 - accuracy: 0.8260
Epoch 5/50
700/700 [==============================] - 0s 674us/step - loss: 0.4153 - accuracy: 0.8287
Epoch 6/50
700/700 [==============================] - 0s 653us/step - loss: 0.4137 - accuracy: 0.8310
Epoch 7/50
700/700 [==============================] - 0s 658us/step - loss: 0.4125 - accuracy: 0.8317
Epoch 8/50
700/700 [==============================] - 1s 842us/step - loss: 0.4116 - accuracy: 0.8306
Epoch 9/50
700/700 [==============================] - 0s 671us/step - loss: 0.4103 - accuracy: 0.8331
Epoch 10/50
700/700 [==============================] - 0s 682us/step - loss: 0.4100 - accuracy: 0.8326
```

```
Epoch 11/50
700/700 [==============================] - 0s 690us/step - loss: 0.4093 - accuracy: 0.8337
Epoch 12/50
700/700 [==============================] - 0s 688us/step - loss: 0.4087 - accuracy: 0.8339
Epoch 13/50
700/700 [==============================] - 0s 675us/step - loss: 0.4081 - accuracy: 0.8341
Epoch 14/50
700/700 [==============================] - 1s 722us/step - loss: 0.4071 - accuracy: 0.8331
Epoch 15/50
700/700 [==============================] - 1s 811us/step - loss: 0.4065 - accuracy: 0.8341
Epoch 16/50
700/700 [==============================] - 0s 711us/step - loss: 0.4056 - accuracy: 0.8356
Epoch 17/50
700/700 [==============================] - 0s 702us/step - loss: 0.4046 - accuracy: 0.8366
Epoch 18/50
700/700 [==============================] - 0s 688us/step - loss: 0.4035 - accuracy: 0.8343
Epoch 19/50
700/700 [==============================] - 1s 715us/step - loss: 0.4024 - accuracy: 0.8363
Epoch 20/50
700/700 [==============================] - 0s 714us/step - loss: 0.4020 - accuracy: 0.8337
Epoch 21/50
700/700 [==============================] - 0s 705us/step - loss: 0.4010 - accuracy: 0.8374
Epoch 22/50
700/700 [==============================] - 1s 720us/step - loss: 0.4003 - accuracy: 0.8370
Epoch 23/50
700/700 [==============================] - 0s 692us/step - loss: 0.3993 - accuracy: 0.8374
Epoch 24/50
700/700 [==============================] - 0s 709us/step - loss: 0.3990 - accuracy: 0.8356
Epoch 25/50
700/700 [==============================] - 1s 871us/step - loss: 0.3984 - accuracy: 0.8366
Epoch 26/50
700/700 [==============================] - 1s 719us/step - loss: 0.3984 - accuracy: 0.8367
Epoch 27/50
700/700 [==============================] - 1s 719us/step - loss: 0.3980 - accuracy: 0.8366
Epoch 28/50
700/700 [==============================] - 0s 695us/step - loss: 0.3981 - accuracy: 0.8366
Epoch 29/50
700/700 [==============================] - 0s 667us/step - loss: 0.3976 - accuracy: 0.8374
Epoch 30/50
700/700 [==============================] - 0s 669us/step - loss: 0.3972 - accuracy: 0.8373
Epoch 31/50
700/700 [==============================] - 0s 670us/step - loss: 0.3970 - accuracy: 0.8370
Epoch 32/50
700/700 [==============================] - 1s 720us/step - loss: 0.3972 - accuracy: 0.8376
Epoch 33/50
700/700 [==============================] - 0s 675us/step - loss: 0.3965 - accuracy: 0.8367
Epoch 34/50
700/700 [==============================] - 0s 680us/step - loss: 0.3961 - accuracy: 0.8364
Epoch 35/50
700/700 [==============================] - 0s 685us/step - loss: 0.3962 - accuracy: 0.8379
Epoch 36/50
700/700 [==============================] - 1s 771us/step - loss: 0.3960 - accuracy: 0.8370
Epoch 37/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3963 - accuracy: 0.8366
Epoch 38/50
700/700 [==============================] - 1s 764us/step - loss: 0.3962 - accuracy: 0.8373
Epoch 39/50
700/700 [==============================] - 1s 823us/step - loss: 0.3950 - accuracy: 0.8384
Epoch 40/50
700/700 [==============================] - 1s 759us/step - loss: 0.3956 - accuracy: 0.8361
Epoch 41/50
700/700 [==============================] - 1s 773us/step - loss: 0.3949 - accuracy: 0.8366
Epoch 42/50
700/700 [==============================] - 0s 695us/step - loss: 0.3953 - accuracy: 0.8369
Epoch 43/50
700/700 [==============================] - 0s 701us/step - loss: 0.3952 - accuracy: 0.8369
Epoch 44/50
700/700 [==============================] - 0s 707us/step - loss: 0.3952 - accuracy: 0.8366
Epoch 45/50
700/700 [==============================] - 0s 680us/step - loss: 0.3955 - accuracy: 0.8376
Epoch 46/50
700/700 [==============================] - 0s 665us/step - loss: 0.3947 - accuracy: 0.8373
Epoch 47/50
700/700 [==============================] - 0s 708us/step - loss: 0.3947 - accuracy: 0.8371
Epoch 48/50
700/700 [==============================] - 0s 681us/step - loss: 0.3944 - accuracy: 0.8371
Epoch 49/50
700/700 [==============================] - 0s 678us/step - loss: 0.3947 - accuracy: 0.8383
Epoch 50/50
700/700 [==============================] - 1s 869us/step - loss: 0.3944 - accuracy: 0.8370
Out[89]: <tensorflow.python.keras.callbacks.History at 0x1fb1eb93df0>
```

```
In [90]: y_pred =classifier.predict(X_test)
         y_pred = (y_pred > 0.5) #Predicting the result
```

```
In [97]: from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
```

```
In [92]: cm = confusion_matrix(y_test,y_pred)
```

```
In [93]: cm
```

```
Out[93]: array([[2328,   72],
                [ 425,  175]], dtype=int64)
```

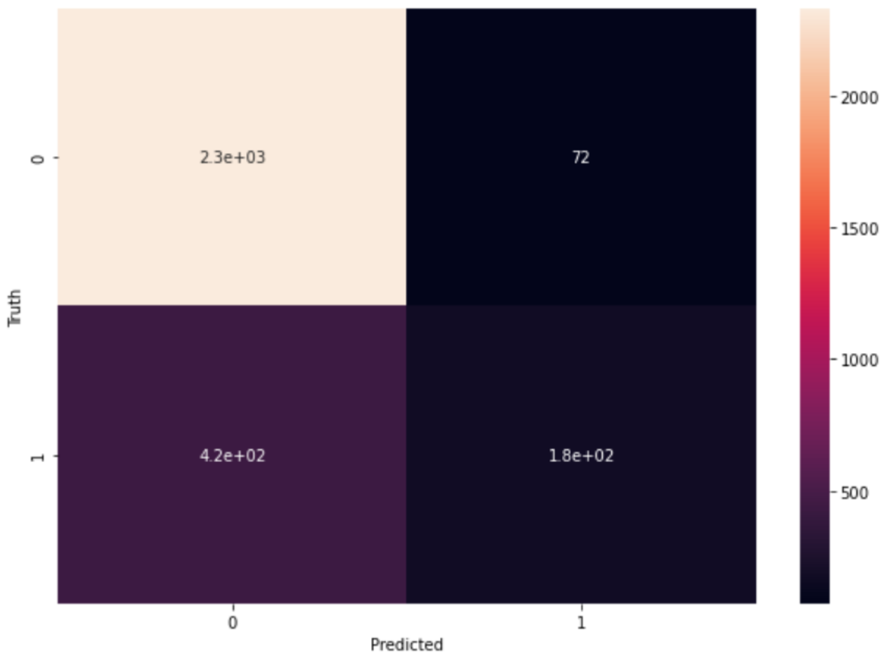
```
In [94]: accuracy = accuracy_score(y_test,y_pred)
```

```
In [95]: accuracy
```

```
Out[95]: 0.8343333333333334
```

```
In [98]: plt.figure(figsize = (10,7))
         sns.heatmap(cm,annot = True)
         plt.xlabel('Predicted')
         plt.ylabel('Truth')
```

```
Out[98]: Text(69.0, 0.5, 'Truth')
```




```
In [100…  print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.97      0.90      2400
           1       0.71      0.29      0.41       600

    accuracy                           0.83      3000
   macro avg       0.78      0.63      0.66      3000
weighted avg       0.82      0.83      0.81      3000
```

CONCLUSION: We have successfully studied normalization of data, accuracy score and confusion matrix. Also implemented and calculated the accuracy score.

ASSIGNMENT NO. : 5

AIM: Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.
Dataset link: https://www.kaggle.com/datasets/abdallamahgoub/diabetes

INTRODUCTION:

❖ K-NEAREST NEIGHBORS

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

In [ ]:
```
import pandas as pd
```

In [ ]:
```
df1 = pd.read_csv('/content/diabetes.csv')
df1.head()
```

Out[ ]:

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

In [ ]:
```
df1.shape
```

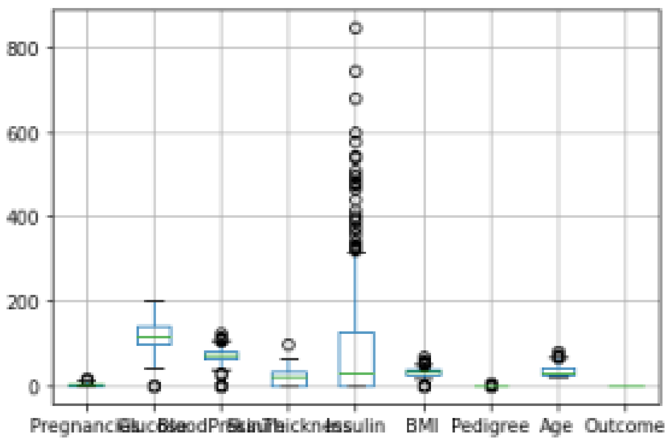
Out[ ]:
```
(768, 9)
```

In [ ]:
```
df1.isnull().sum()
```

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

In [ ]:
```
df1.boxplot()
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcea52b15d0>
```

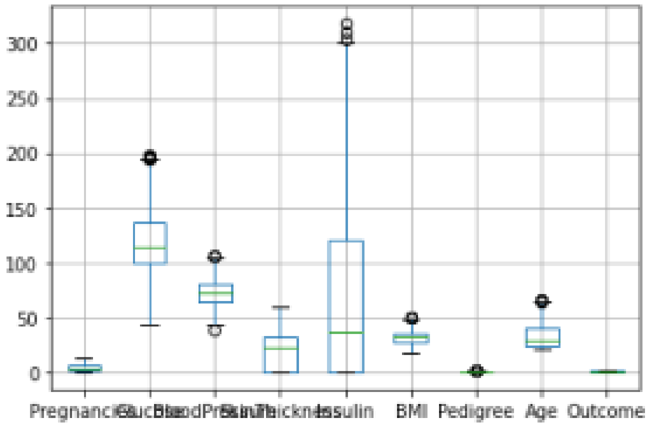


In [ ]:
```
Q1 = df1.quantile(0.25)
Q3 = df1.quantile(0.75)
IQR = Q3 - Q1
```

```
df1 = df1[~((df1 < (Q1 - 1.5 * IQR)) |(df1 > (Q3 + 1.5 * IQR))).any(axis=1)]
```

In [ ]:
```
df1.boxplot()
```

Out[ ]: `<matplotlib.axes._subplots.AxesSubplot at 0x7fcea534bcd0>`



In [ ]:
```
df1.shape
```

Out[ ]: `(639, 9)`

In [ ]:
```
X = df1.iloc[:, :-1].values
y = df1.iloc[:, 8].values
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(X_train, y_train)
```

Out[ ]: `KNeighborsClassifier()`

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

In [ ]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
```

```
print (result1)
# accuracy = zero_one_score(y_test, y_pred)
# error_rate = 1 - accuracy
# print(error_rate)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2*100)
error_rate = 1 - result2
print('Error Rate:',error_rate)
```

```
Confusion Matrix:
[[100   9]
 [ 33  18]]
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.92      0.83       109
           1       0.67      0.35      0.46        51

    accuracy                           0.74       160
   macro avg       0.71      0.64      0.64       160
weighted avg       0.72      0.74      0.71       160

Accuracy: 73.75
Error Rate: 0.26249999999999996
```

```
In [ ]:
```

CONCLUSION: We have successfully studied about K-Nearest Neighbours algorithm.

ASSIGNMENT NO. : 6

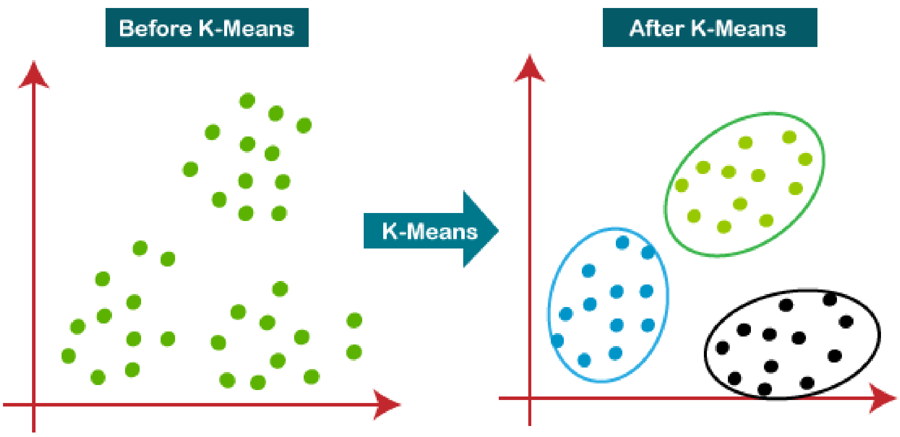
AIM: Implement K-Means clustering / hierarchical clustering on sales_data_sample.csv dataset. Determine the number of clusters using the elbow method.
Dataset link: https://www.kaggle.com/datasets/kyanyoga/sample-sales-data

INTRODUCTION:

❖ K-MEANS CLUSTERING

- K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science.
- It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.
- It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.
- It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.
- The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.
- It performs two tasks:

  1. Determines the best value for K center points or centroids by an iterative process.
  2. Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.



❖ HIERARCHICAL CLUSTERING

- Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as hierarchical cluster analysis or HCA.
- In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the dendrogram.
- Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm.
- The hierarchical clustering technique has two approaches:
  1. Agglomerative: Agglomerative is a bottom-up approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.
  2. Divisive: Divisive algorithm is the reverse of the agglomerative algorithm as it is a top-down approach.

In [ ]:
```
import pandas as pd
```

In [ ]:
```
df1 = pd.read_csv('/content/sales_data_sample.csv', encoding='latin1')
df1.head()
```

Out[ ]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	ST
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Sh
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	Sh
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	Sh
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Sh
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Sh

5 rows × 25 columns

In [ ]:
```
df1.shape
```

Out[ ]:
```
(2823, 25)
```

In [ ]:
```
df1.isnull().sum()
```

```
Out[ ]: 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
        PHONE                  0
        ADDRESSLINE1           0
        ADDRESSLINE2        2521
        CITY                   0
        STATE               1486
        POSTALCODE            76
        COUNTRY                0
        TERRITORY           1074
        CONTACTLASTNAME        0
        CONTACTFIRSTNAME       0
```

```
        DEALSIZE               0
        dtype: int64
```

```
In [ ]: df2 = df1.drop(['ADDRESSLINE1','ADDRESSLINE2','STATUS','POSTALCODE','CITY','TERRITOR
        df2.columns
```

```
Out[ ]: Index(['QUANTITYORDERED', 'PRICEEACH', 'ORDERLINENUMBER', 'SALES', 'ORDERDATE',
               'QTR_ID', 'MONTH_ID', 'YEAR_ID', 'PRODUCTLINE', 'MSRP', 'PRODUCTCODE',
               'COUNTRY', 'DEALSIZE'],
              dtype='object')
```

```
In [ ]: df2.isnull().sum()
```

```
Out[ ]: QUANTITYORDERED    0
        PRICEEACH          0
        ORDERLINENUMBER    0
        SALES              0
        ORDERDATE          0
        QTR_ID             0
        MONTH_ID           0
        YEAR_ID            0
        PRODUCTLINE        0
        MSRP               0
        PRODUCTCODE        0
        COUNTRY            0
        DEALSIZE           0
        dtype: int64
```

```
In [ ]: productline = pd.get_dummies(df2['PRODUCTLINE'])
        dealsize = pd.get_dummies(df2['DEALSIZE'])
```

```
In [ ]: df3 =pd.concat([df2,productline,dealsize],axis=1)
```

```
In [ ]: df4 = df3.drop(['DEALSIZE','PRODUCTLINE','COUNTRY'], axis =1)
```

In [ ]:
```
df4['PRODUCTCODE'] = pd.Categorical(df4['PRODUCTCODE']).codes
```

In [ ]:
```
df5 = df4.drop(['ORDERDATE'], axis =1)
```

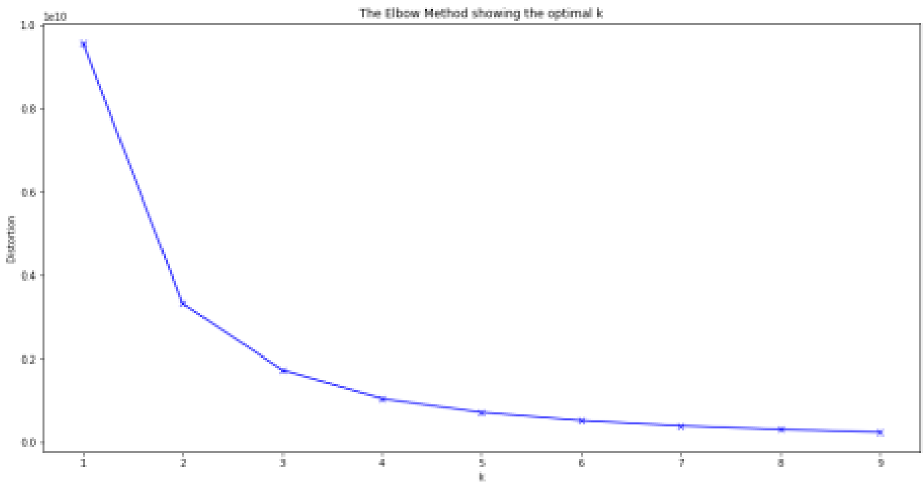
In [ ]:
```
df5.dtypes
```

Out[ ]:
```
QUANTITYORDERED       int64
PRICEEACH           float64
ORDERLINENUMBER       int64
SALES               float64
QTR_ID                int64
MONTH_ID              int64
YEAR_ID               int64
MSRP                  int64
PRODUCTCODE            int8
Classic Cars          uint8
Motorcycles           uint8
Planes                uint8
Ships                 uint8
Trains                uint8
```

```
Trucks and Buses      uint8
Vintage Cars          uint8
Large                 uint8
Medium                uint8
Small                 uint8
dtype: object
```

In [ ]:
```
from sklearn.cluster import KMeans
distortions = []
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(df5)
    distortions.append(kmeanModel.inertia_)
```

In [ ]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.figure(figsize=(16,8))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```



CONCLUSION: We have successfully studied about K-Means clustering and hierarchical clustering.