Group B

Assignment No:1

Title of the Assignment: Predict the price of the Uber ride from a given pickup point to theagreed 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 Description:

The project is about on world's largest taxi company Uber inc. In this project, we're looking to predict the fare for their future transactional cases. Uber deliversservice to lakhs of customers daily. Now it becomes really important to manage their dataproperly to come up with new business ideas to get best results. Eventually, it becomesreally important to estimate the fare prices accurately.

Link for Dataset:https://www.kaggle.com/datasets/yasserh/uber-fares-datasetObjective of the Assignment:

Students should be able to pre-process dataset and identify outliers, to check correlation and

implement linear regression and random forest regression models. Evaluate them with respective scores like R2, RMSE etc.

Prerequisite:

1. Basic knowledge of Python
2. Concept of pre-processing data
3. Basic knowledge of Data Science and Big Data Analytics.

Contents of the Theory:

1. Data Pre-processing
2. Linear regression
3. Random Forest regression models
4. Box Plot
5. Outliers
6. Haversine
7. Mathplotlib
8. Mean Squared Error

Data Preprocessing:

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is therst and crucial step while creating a machine learning model.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data preprocessing task.

Why do we need Data Preprocessing?

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and eciency of a machine learning model.

It involves below steps:

- Getting the dataset
- Importing libraries
- Importing datasets
- Finding Missing Data
- Encoding Categorical Data
- Splitting dataset into training and test set
- Feature scaling

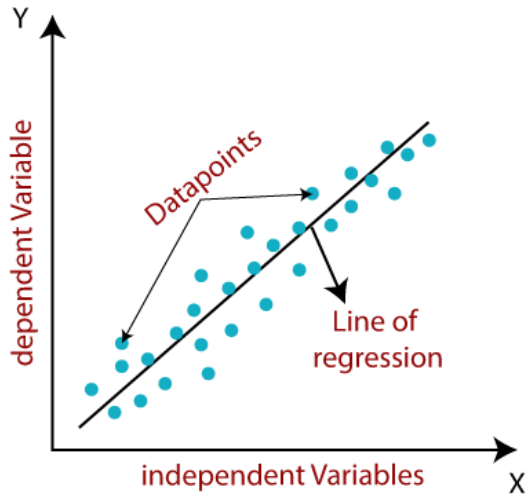
Linear Regression:

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price,** etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means itnds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:

Random Forest



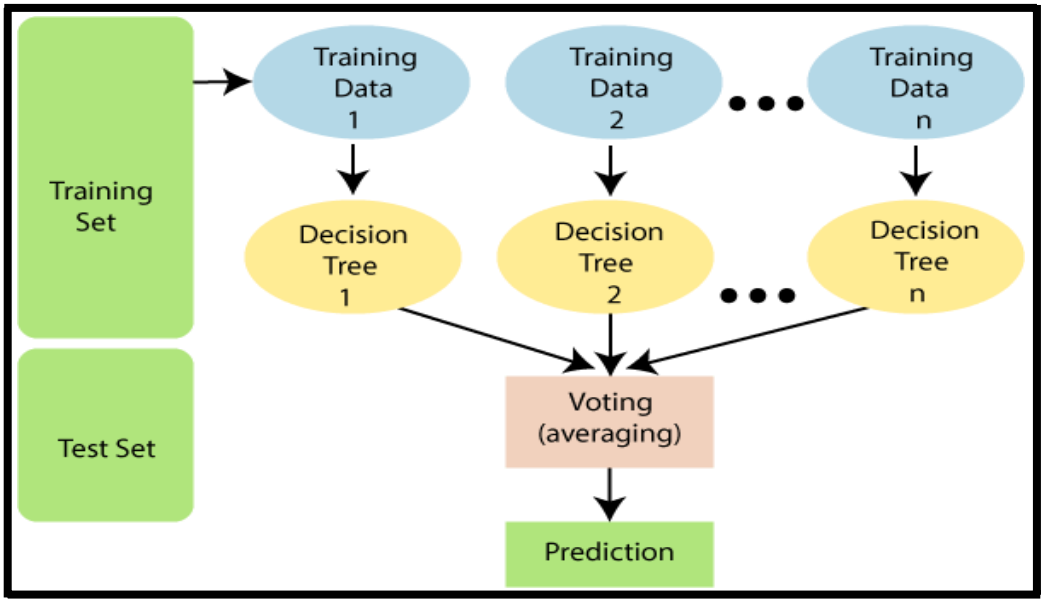
1.	x	It is a vector or a formula.
2.	data	It is the data frame.
3.	notch	It is a logical value set as true to draw a notch.
4.	Var width	It is also a logical value set as true to draw the width of the box same as the sample size.
5.	names	It is the group of labels that will be printed under each boxplot.
6.	main	It is used to give a title to the graph.

Random Forest Regression Models:

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of combining multiple classier to solve a complex problem and to improve the performance of the model.

As the name suggests,***"Random Forest is a classier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."***Instead of relying on one decision tree, the random

prediction from each tree and based on the majority votes of predictions, and it predicts the Final output.



The greater number of trees in the forest leads to higher accuracy and prevents the problem of overtting.

Boxplot:

Boxplots are a measure of how well data is distributed across a data set. This divides the data set into three quartiles. This graph represents the minimum, maximum, average,rst quartile, and the third quartile in the data set. Boxplot is also useful in comparing the distribution of data in a data set by drawing a boxplot for each of them.

R provides a boxplot() function to create a boxplot. There is the following syntax of boxplot() function:

boxplot(x, data, notch, varwidth, names, main)

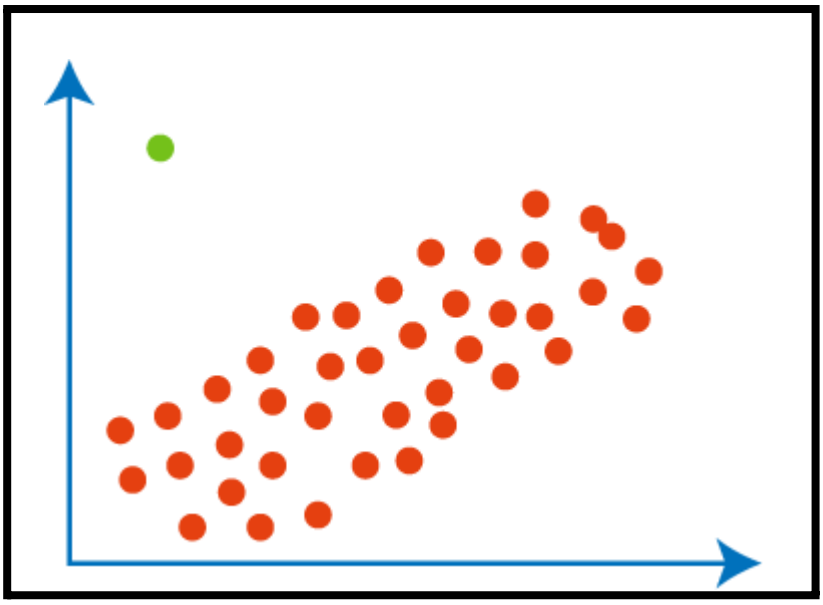
boxplot(x, data, notch, vaíwidth, names, main)

Outliers:

As the name suggests, "outliers" refer to the data points that exist outside of what is to be expected. The major thing about the outliers is what you do with them. If you are going to analyze any task to analyse data sets, you will always have some assumptions based on how this data is generated. If yound some data points that are likely to contain some form of error, then these are denitely outliers, and depending on the context, you want to overcome those errors. The data mining process involves the analysis and prediction of data that the data holds. In 1969, Grubbs introduced therst dentition of outliers.

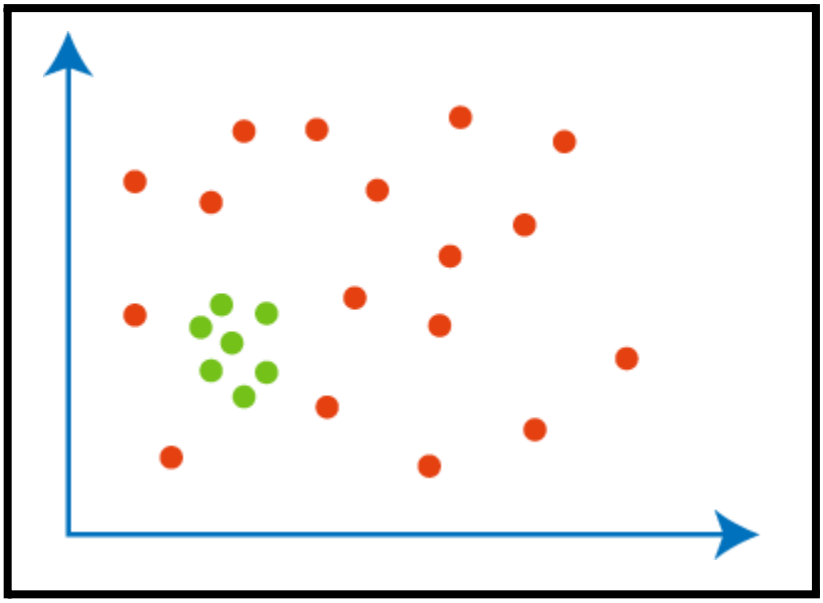
Global Outliers

Global outliers are also called point outliers. Global outliers are taken as the simplest form of outliers. When data points deviate from all the rest of the data points in a given data set, it is known as the global outlier. In most cases, all the outlier detection procedures are targeted to determine the global outliers. The green data point is the global outlier. known as the global



Collective Outliers

In a given set of data, when a group of data points deviates from the rest of the data set is called collective outliers. Here, the particular set of data objects may not be outliers, but when you consider the data objects as a whole, they may behave as outliers. To identify the types of different outliers, you need to go through background information about the relationship between the behavior of outliers shown by different data objects. For example, in an Intrusion Detection System, the DOS package from one system to another is taken as normal behavior. Therefore, if this happens with the various computer simultaneously, it is considered abnormal behavior, and as a whole, they are called collective outliers. The green data points as a whole represent the collective outlier.

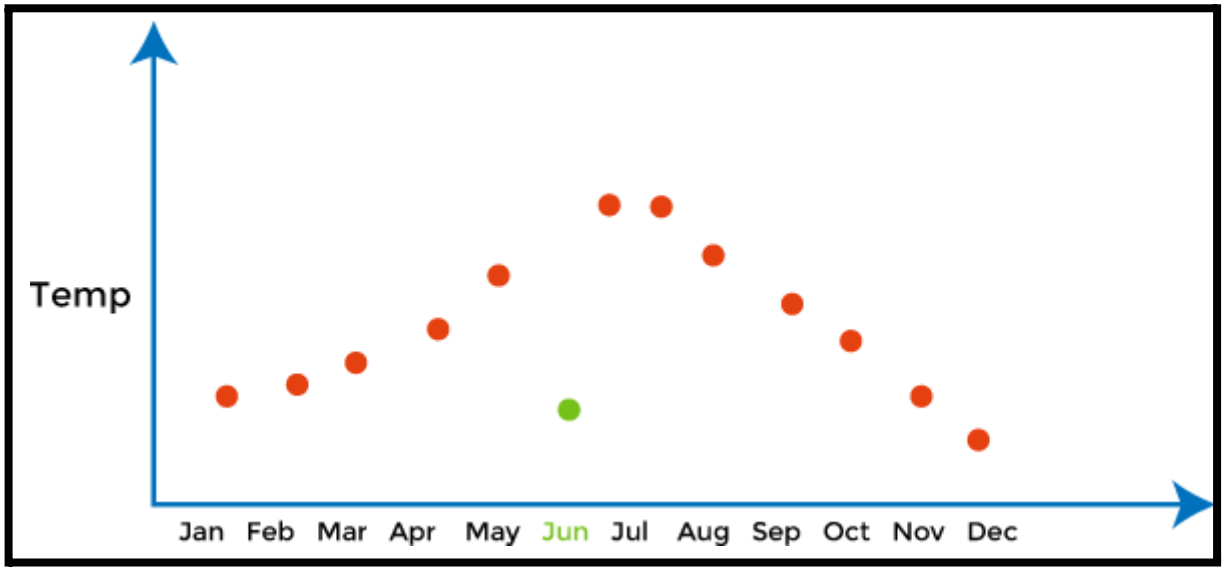


Collective Outliers

In a given set of data, when a group of data points deviates from the rest of the data set is called collective outliers. Here, the particular set of data objects may not be outliers, but when you consider the data objects as a whole, they may behave as outliers. To identify the types of different outliers, you need to go through background information about the relationship between the behavior of outliers shown by different data objects. For example, in an Intrusion Detection System, the DOS package from one system to another is taken as normal behaviour. Therefore, if this happens with the various computer simultaneously, it is considered abnormal behaviour, and as a whole, they are called collective outliers. The green data points as a whole represents the collective outlier.

Contextual Outliers

As the name suggests, "Contextual" means this outlier introduced within a context. For example, in the speech recognition technique, the single background noise. Contextual outliers are also known as Conditional outliers. These types of outliers happen if a data object deviates from the other data points because of any specic condition in a given data set. As we know, there are two types of attributes of objects of data: contextual attributes and behavioural attributes. Contextual outlier analysis enables the users to examine outliers in different contexts and conditions, which can be useful in various applications. For example, A temperature reading of 45 degrees Celsius may behave as an outlier in a rainy season. Still, it will behave like a normal data point in the context of a summer season. In the given diagram, a green dot representing the low-temperature value in June is a contextual outlier since the same value in December is not an outlier.



Haversine:

The Haversine formula calculates the shortest distance between two points on a sphere using their latitudes and longitudes measured along the surface. It is important for use in navigation.

Matplotlib:

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Mean Squared Error;

The**Mean Squared Error (MSE)**or**Mean Squared Deviation (MSD)**of an estimator measures the average of error squares i.e. the average squared difference between the estimated values and true value. It is a risk function, corresponding to the expected value of the squared error loss. It is always non – negative and values close to zero are better. The MSE is the second moment of the error (about the origin) and thus incorporates both the variance of the estimator and its bias.

Conclusion:

In this way we have explored Concept correlation and implement linear regression and random forest regression models.

Assignment Questions:

1. What is data preprocessing?

2. Dene Outliers?

3. What is Linear Regression?

4. What is Random Forest Algorithm?

5. Explain: pandas, numpy?

Program

#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 (https://www.kaggle.com/datasets/yasserh/uber-fares-dataset)

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

In [ ]:
```
#importing the dataset
df  = pd.read_csv("uber.csv")
```

1. Pre-process the dataset.

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

Out[3]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pa
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	

In [4]:
```
df.info() #To get the required information of the dataset
```

```
<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 [5]:
```
df.columns #TO get number of columns in the dataset
```

Out[5]:
```
Index(['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count'],
      dtype='object')
```

In [6]:
```
df = df.drop(['Unnamed: 0', 'key'], axis= 1) #To drop unnamed column as it isn't required
```

In [7]:
```
df.head()
```

Out[7]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1
2	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1
3	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3
4	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5

In [8]:
```
df.shape #To get the total (Rows,Columns)
```

Out[8]:
```
(200000, 7)
```

In [9]:
```
df.dtypes #To get the type of each column
```

Out[9]:
```
fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```


In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 7 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   fare_amount        200000 non-null  float64
 1   pickup_datetime    200000 non-null  object 
 2   pickup_longitude   200000 non-null  float64
 3   pickup_latitude    200000 non-null  float64
 4   dropoff_longitude  199999 non-null  float64
 5   dropoff_latitude   199999 non-null  float64
 6   passenger_count    200000 non-null  int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 10.7+ MB
```

In [11]:
```
df.describe() #To get statistics of each columns
```

Out[11]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	200000.000000
mean	11.359955	-72.527638	39.935885	-72.525292	39.923890	1.684535
std	9.901776	11.437787	7.720539	13.117408	6.794829	1.385997
min	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	0.000000
25%	6.000000	-73.992065	40.734796	-73.991407	40.733823	1.000000
50%	8.500000	-73.981823	40.752592	-73.980093	40.753042	1.000000
75%	12.500000	-73.967154	40.767158	-73.963658	40.768001	2.000000
max	499.000000	57.418457	1644.421482	1153.572603	872.697628	208.000000

Filling Missing values

In [12]:
```
df.isnull().sum()
```

Out[12]:
```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

In [13]:
```
df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inplace = True)
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inplace = True)
```

In [14]:
```
df.isnull().sum()
```

Out[14]:
```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

In [15]:
```
df.dtypes
```

Out[15]:
```
fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

Column pickup_datetime is in wrong format (Object). Convert it to DateTime Format

In [16]:
```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime, errors='coerce')
```


In [17]:
```
df.dtypes
```

Out[17]:
```
fare_amount                      float64
pickup_datetime      datetime64[ns, UTC]
pickup_longitude                 float64
pickup_latitude                  float64
dropoff_longitude                float64
dropoff_latitude                 float64
passenger_count                    int64
dtype: object
```

To segregate each time of date and time

In [18]:
```
df= df.assign(hour = df.pickup_datetime.dt.hour,
              day= df.pickup_datetime.dt.day,
              month = df.pickup_datetime.dt.month,
              year = df.pickup_datetime.dt.year,
              dayofweek = df.pickup_datetime.dt.dayofweek)
```

In [19]:
```
df.head()
```

Out[19]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day	mon
0	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.738354	-73.999512	40.723217	1	19	7	
1	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.728225	-73.994710	40.750325	1	20	17	
2	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.740770	-73.962565	40.772647	1	21	24	
3	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.790844	-73.965316	40.803349	3	8	26	
4	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.744085	-73.973082	40.761247	5	17	28	

In [20]:
```
# drop the column 'pickup_daetime' using drop()
# 'axis = 1' drops the specified column

df = df.drop('pickup_datetime',axis=1)
```

In [21]:
```
df.head()
```

Out[21]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day	month	year	dayofwe
0	7.5	-73.999817	40.738354	-73.999512	40.723217	1	19	7	5	2015	
1	7.7	-73.994355	40.728225	-73.994710	40.750325	1	20	17	7	2009	
2	12.9	-74.005043	40.740770	-73.962565	40.772647	1	21	24	8	2009	
3	5.3	-73.976124	40.790844	-73.965316	40.803349	3	8	26	6	2009	
4	16.0	-73.925023	40.744085	-73.973082	40.761247	5	17	28	8	2014	

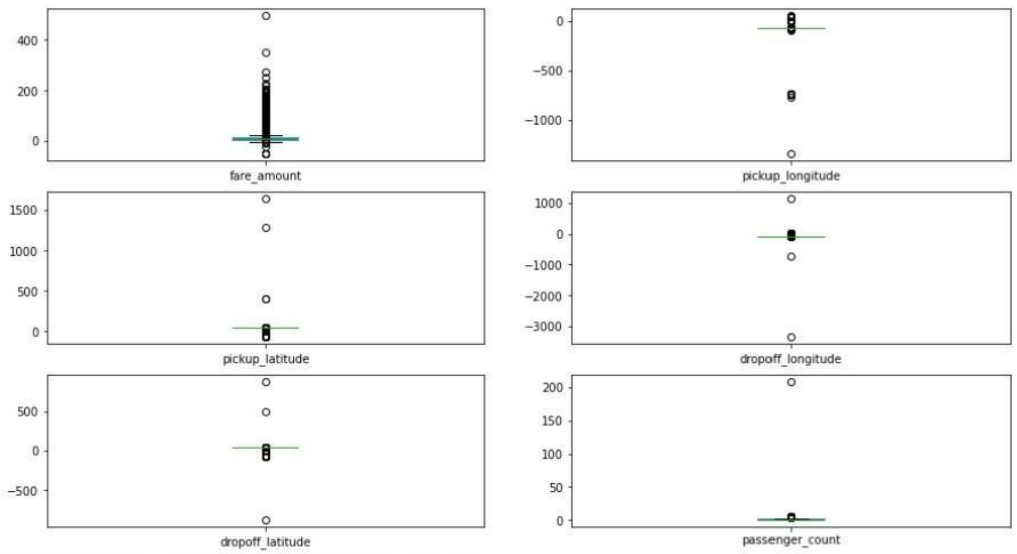
In [22]:
```
df.dtypes
```

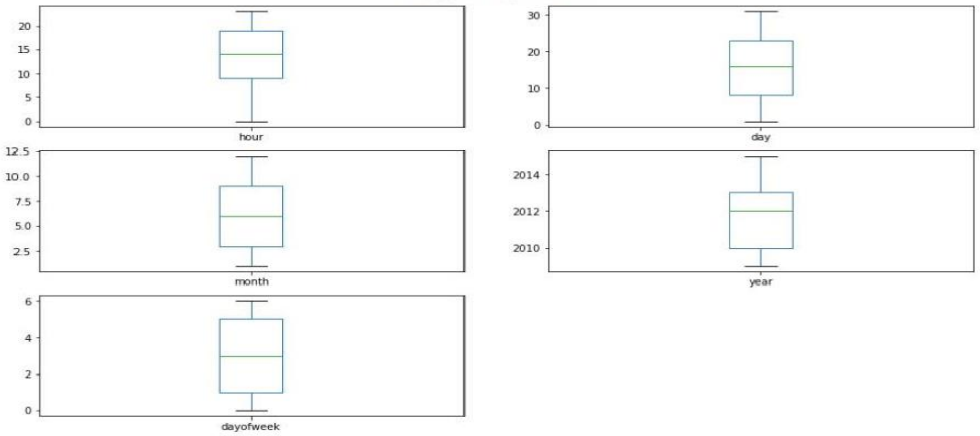
Out[22]:
```
fare_amount          float64
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
hour                   int64
day                    int64
month                  int64
year                   int64
dayofweek              int64
dtype: object
```

Checking outliers and filling them

In [23]:
```
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxplot to check the outliers
```

Out[23]:
```
fare_amount            AxesSubplot(0.125,0.787927;0.352273x0.0920732)
pickup_longitude     AxesSubplot(0.547727,0.787927;0.352273x0.0920732)
pickup_latitude        AxesSubplot(0.125,0.677439;0.352273x0.0920732)
dropoff_longitude    AxesSubplot(0.547727,0.677439;0.352273x0.0920732)
dropoff_latitude       AxesSubplot(0.125,0.566951;0.352273x0.0920732)
passenger_count      AxesSubplot(0.547727,0.566951;0.352273x0.0920732)
hour                   AxesSubplot(0.125,0.456463;0.352273x0.0920732)
day                  AxesSubplot(0.547727,0.456463;0.352273x0.0920732)
month                  AxesSubplot(0.125,0.345976;0.352273x0.0920732)
year                 AxesSubplot(0.547727,0.345976;0.352273x0.0920732)
dayofweek              AxesSubplot(0.125,0.235488;0.352273x0.0920732)
dtype: object
```



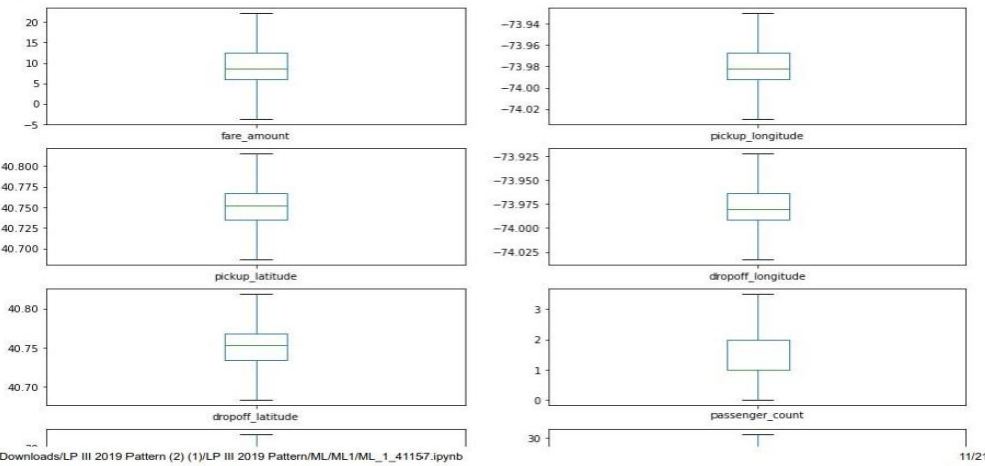


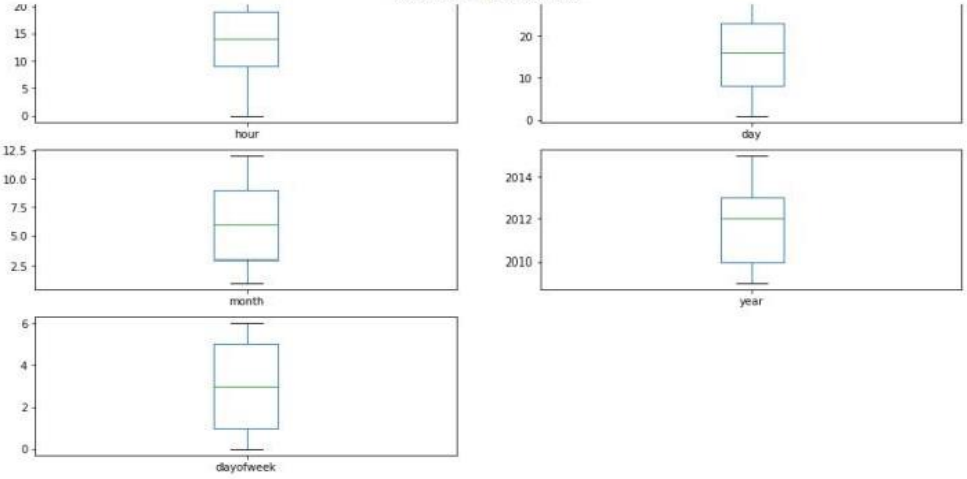
In [24]:
```
#Using the InterQuartile Range to fill the values
def remove_outlier(df1 , col):
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_whisker = Q1-1.5*IQR
    upper_whisker = Q3+1.5*IQR
    df[col] = np.clip(df1[col] , lower_whisker , upper_whisker)
    return df1

def treat_outliers_all(df1 , col_list):
    for c in col_list:
        df1 = remove_outlier(df , c)
    return df1
```


In [25]:
```
df = treat_outliers_all(df , df.iloc[: , 0::])
```


In [26]:
```
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxplot shows that dataset is free from ou
```

Out[26]:
```
fare_amount          AxesSubplot(0.125,0.787927;0.352273x0.0920732)
pickup_longitude     AxesSubplot(0.547727,0.787927;0.352273x0.0920732)
pickup_latitude      AxesSubplot(0.125,0.677439;0.352273x0.0920732)
dropoff_longitude    AxesSubplot(0.547727,0.677439;0.352273x0.0920732)
dropoff_latitude     AxesSubplot(0.125,0.566951;0.352273x0.0920732)
passenger_count      AxesSubplot(0.547727,0.566951;0.352273x0.0920732)
hour                 AxesSubplot(0.125,0.456463;0.352273x0.0920732)
day                  AxesSubplot(0.547727,0.456463;0.352273x0.0920732)
month                AxesSubplot(0.125,0.345976;0.352273x0.0920732)
year                 AxesSubplot(0.547727,0.345976;0.352273x0.0920732)
dayofweek            AxesSubplot(0.125,0.235488;0.352273x0.0920732)
dtype: object
```





In [27]:
```
#pip install haversine
import haversine as hs  #Calculate the distance using Haversine to calculate the distance between to points. Can
travel_dist = []
for pos in range(len(df['pickup_longitude'])):
        long1,lati1,long2,lati2 = [df['pickup_longitude'][pos],df['pickup_latitude'][pos],df['dropoff_longitude'
        loc1=(lati1,long1)
        loc2=(lati2,long2)
        c = hs.haversine(loc1,loc2)
        travel_dist.append(c)

print(travel_dist)
df['dist_travel_km'] = travel_dist
df.head()
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

Out[27]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day	month	year	dayofwe
0	7.5	-73.999817	40.738354	-73.999512	40.723217	1.0	19	7	5	2015	
1	7.7	-73.994355	40.728225	-73.994710	40.750325	1.0	20	17	7	2009	
2	12.9	-74.005043	40.740770	-73.962565	40.772647	1.0	21	24	8	2009	
3	5.3	-73.976124	40.790844	-73.965316	40.803349	3.0	8	26	6	2009	
4	16.0	-73.929786	40.744085	-73.973082	40.761247	3.5	17	28	8	2014	

In [28]:
```
#Uber doesn't travel over 130 kms so minimize the distance
df= df.loc[(df.dist_travel_km >= 1) | (df.dist_travel_km <= 130)]
print("Remaining observastions in the dataset:", df.shape)
```

```
Remaining observastions in the dataset: (200000, 12)
```

In [29]:
```
#Finding inccorect latitude (Less than or greater than 90) and longitude (greater than or less than 180)
incorrect_coordinates = df.loc[(df.pickup_latitude > 90) |(df.pickup_latitude < -90) |
                               (df.dropoff_latitude > 90) |(df.dropoff_latitude < -90) |
                               (df.pickup_longitude > 180) |(df.pickup_longitude < -180) |
                               (df.dropoff_longitude > 90) |(df.dropoff_longitude < -90)
                                ]
```

In [30]:
```
df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')
```

In [31]:
```
df.head()
```

Out[31]:

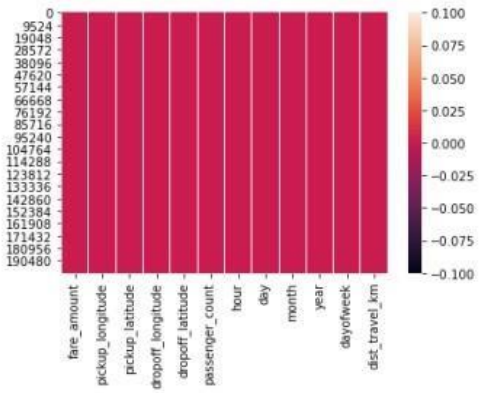
	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day	month	year	dayofwe
0	7.5	-73.999817	40.738354	-73.999512	40.723217	1.0	19	7	5	2015	
1	7.7	-73.994355	40.728225	-73.994710	40.750325	1.0	20	17	7	2009	
2	12.9	-74.005043	40.740770	-73.962565	40.772647	1.0	21	24	8	2009	
3	5.3	-73.976124	40.790844	-73.965316	40.803349	3.0	8	26	6	2009	
4	16.0	-73.929786	40.744085	-73.973082	40.761247	3.5	17	28	8	2014	

In [32]:
```
df.isnull().sum()
```

Out[32]:
```
fare_amount          0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
hour                 0
day                  0
month                0
year                 0
dayofweek            0
dist_travel_km       0
dtype: int64
```


In [33]:
```
sns.heatmap(df.isnull()) #Free for null values
```

Out[33]: <AxesSubplot:>



In [34]:
```
corr = df.corr() #Function to find the correlation
```

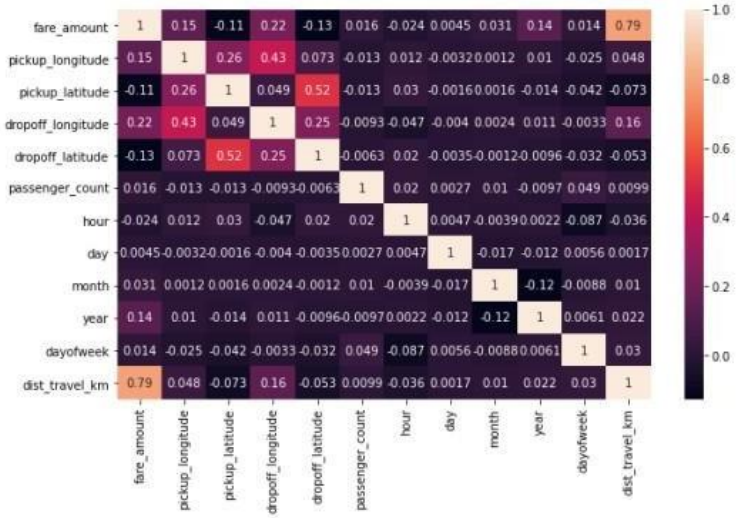
In [35]:
```
corr
```

Out[35]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	d
fare_amount	1.000000	0.154069	-0.110842	0.218675	-0.125898	0.015778	-0.023623	0.0045
pickup_longitude	0.154069	1.000000	0.259497	0.425619	0.073290	-0.013213	0.011579	-0.0032
pickup_latitude	-0.110842	0.259497	1.000000	0.048889	0.515714	-0.012889	0.029681	-0.0015
dropoff_longitude	0.218675	0.425619	0.048889	1.000000	0.245667	-0.009303	-0.046558	-0.0040
dropoff_latitude	-0.125898	0.073290	0.515714	0.245667	1.000000	-0.006308	0.019783	-0.0034
passenger_count	0.015778	-0.013213	-0.012889	-0.009303	-0.006308	1.000000	0.020274	0.0027
hour	-0.023623	0.011579	0.029681	-0.046558	0.019783	0.020274	1.000000	0.0046
day	0.004534	-0.003204	-0.001553	-0.004007	-0.003479	0.002712	0.004677	1.0000
month	0.030817	0.001169	0.001562	0.002391	-0.001193	0.010351	-0.003926	-0.0173
year	0.141277	0.010198	-0.014243	0.011346	-0.009603	-0.009749	0.002156	-0.0121
dayofweek	0.013652	-0.024652	-0.042310	-0.003336	-0.031919	0.048550	-0.086947	0.0056
dist_travel_km	0.786385	0.048446	-0.073362	0.155191	-0.052701	0.009884	-0.035708	0.0017

In [36]:
```
fig,axis = plt.subplots(figsize = (10,6))
sns.heatmap(df.corr(),annot = True) #Correlation Heatmap (Light values means highly correlated)
```

Out[36]: <AxesSubplot:>



Dividing the dataset into feature and target values

In [182]:
```
x = df[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude','passenger_count','hour','da
```

In [183]:
```
y = df['fare_amount']
```

Dividing the dataset into training and testing dataset

In [184]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size = 0.33)
```

Linear Regression

In [185]:
```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
```

In [186]:
```
regression.fit(X_train,y_train)
```

Out[186]:
```
LinearRegression()
```

In [80]:
```
regression.intercept_ #To find the linear intercept
```

Out[80]:
```
2640.1356169149753
```

In [187]:
```
regression.coef_ #To find the linear coeeficient
```

Out[187]:
```
array([ 2.54805415e+01, -7.18365435e+00,  1.96232986e+01, -1.79401980e+01,
        5.48472723e-02,  5.32910041e-03,  4.05930990e-03,  5.74261856e-02,
        3.66574831e-01, -3.03753790e-02,  1.84233728e+00])
```


In [188]:
```
prediction = regression.predict(X_test) #To predict the target values
```

In [189]:
```
print(prediction)
```

```
[ 5.47848314 10.11016249 12.19490542 ...  7.11952609 20.2482979
  8.82791961]
```

In [190]:
```
y_test
```

Out[190]:
```
155740     4.90
47070     10.00
116192    14.50
164589     6.50
154309    11.30
          ...
76552      7.70
27926     10.90
38972      6.50
120341    22.25
178449     8.10
Name: fare_amount, Length: 66000, dtype: float64
```

Metrics Evaluation using R2, Mean Squared Error, Root Mean Sqared Error

In [191]:
```
from sklearn.metrics import r2_score
```

In [192]:
```
r2_score(y_test,prediction)
```

Out[192]:
```
0.6651880468683617
```

In [193]:
```
from sklearn.metrics import mean_squared_error
```

In [194]:
```
MSE = mean_squared_error(y_test,prediction)
```

In [195]:
```
MSE
```

Out[195]:
```
9.961516917717704
```

In [196]:
```
RMSE = np.sqrt(MSE)
```

In [197]:
```
RMSE
```

Out[197]:
```
3.156187085348032
```

Random Forest Regression

In [198]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [199]:
```
rf = RandomForestRegressor(n_estimators=100) #Here n_estimators means number of trees you want to build before m
```

In [200]:
```
rf.fit(X_train,y_train)
```

Out[200]:
```
RandomForestRegressor()
```

In [201]:
```
y_pred = rf.predict(X_test)
```

In [202]:
```
y_pred
```

Out[202]:
```
array([ 5.714 , 10.285 , 12.68  , ...,  6.338 , 19.4685,  7.712 ])
```

Metrics evaluatin for Random Forest

In [210]:
```
R2_Random = r2_score(y_test,y_pred)
```

In [211]:
```
R2_Random
```

Out[211]:
```
0.7948374920410631
```

In [205]:
```
MSE_Random = mean_squared_error(y_test,y_pred)
```


In [206]:
```
MSE_Random
```

Out[206]:
```
6.104112397417331
```

In [207]:
```
RMSE_Random = np.sqrt(MSE_Random)
```

In [208]:
```
RMSE_Random
```

Out[208]:
```
2.4706501972997574
```

Group B

Assignment No:2

Title of Assignment: Classify the email using the binary classification method. Email Spam detection has two states:
a) Normal State – Not Spam,
b) Abnormal State – Spam. 1. Basic knowledge of Python

Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance.

Dataset Description: The csv file contains 5172 rows, each row for each email. There are 3002 columns. The first column indicates Email name. The name has been set with numbers and not recipients' name to protect privacy. The last column has the labels for prediction : 1 for spam, 0 for not spam. The remaining 3000 columns are the 3000 most common words in all the emails, after excluding the non-alphabetical characters/words. For each row, the count of each word(column) in that email(row) is stored in the respective cells. Thus, information regarding all 5172 emails are stored in a compact dataframe rather than as separate text files.

Link:https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv

Objective of the Assignment:
Students should be able to classify email using the binary Classification and implement email spam detection technique by using K-Nearest Neighbors and Support Vector Machine algorithm.

Prerequisite:

1. Data Preprocessing:

2. Concept of K-Nearest Neighbors and Support Vector Machine for classification.

Contents of the Theory:

1. Data Preprocessing
2. Binary Classification
3. K-Nearest Neighbours
4. Support Vector Machine
5. Train, Test and Split Procedure

Data Preprocessing:

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is therst and crucial step while creating a machine learning model.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data preprocessing task.

Why do we need Data Preprocessing?

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and eciency of a machine learning model.

It involves below steps:

- Getting the dataset
- Importing libraries
- Importing datasets
- Finding Missing Data
- Encoding Categorical Data
- Splitting dataset into training and test set
- Feature scaling

Assignment 2

2. 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 (https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv)

In [19]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import metrics
```

In [20]:
```
df=pd.read_csv('emails.csv')
```


In [21]:
```
df.head()
```

Out[21]:

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

5 rows × 3002 columns

In [22]:
```
df.columns
```

Out[22]:
```
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 [23]:
```
df.isnull().sum()
```

Out[23]:
```
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 [24]:
```
df.dropna(inplace = True)
```

In [25]:
```
df.drop(['Email No.'],axis=1,inplace=True)
X = df.drop(['Prediction'],axis = 1)
y = df['Prediction']
```

In [26]:
```
from sklearn.preprocessing import scale
X = scale(X)
# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
```

##KNN classifier

In [35]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

In [36]:
```
print("Prediction",y_pred)
```

```
Prediction [0 0 1 ... 1 1 1]
```


In [37]:
```
print("KNN accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

```
KNN accuracy =  0.8009020618556701
```

In [39]:
```
print("Confusion matrix",metrics.confusion_matrix(y_test,y_pred))
```

```
Confusion matrix [[804 293]
 [ 16 439]]
```

SVM classifier

In [27]:
```
# cost C = 1
model = SVC(C = 1)

# fit
model.fit(X_train, y_train)

# predict
y_pred = model.predict(X_test)
```

In [28]:
```
metrics.confusion_matrix(y_true=y_test, y_pred=y_pred)
```

Out[28]:
```
array([[1091,    6],
       [  90,  365]])
```

In [29]:
```
print("SVM accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

```
SVM accuracy =  0.9381443298969072
```

Group B

Assignment No:3

Title of the Assignment: 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 for Dataset:https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling

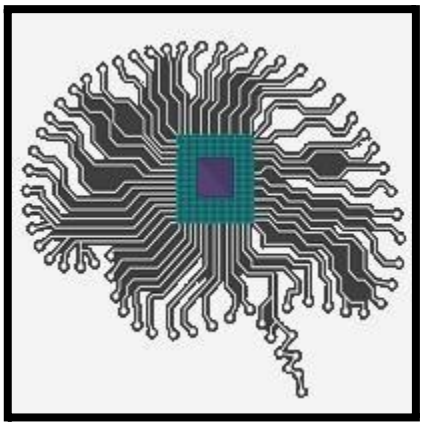
Perform the following steps:
1. Read the dataset.
2. Distinguish the feature and target set and divide the data set into training and test sets.
3. Normalize the train and test data.
4. Initialize and build the model. Identify the points of improvement and implement the same.
5. Print the accuracy score and confusion matrix (5 points).

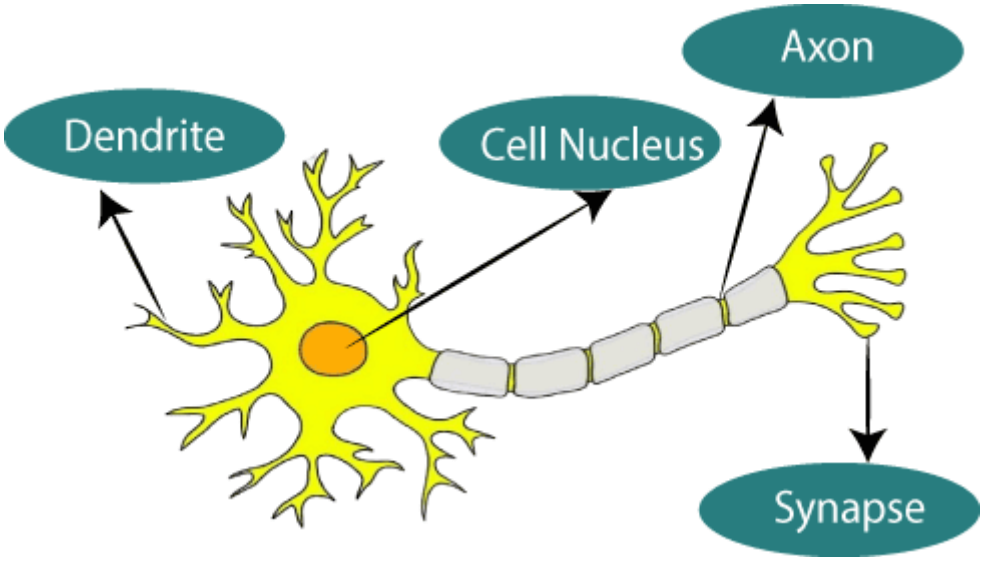
Objective of the Assignment:
Students should be able to distinguish the feature and target set and divide the data set into training and test sets and normalize them and students should build the model on the basis of that.

Prerequisite:
1. Basic knowledge of Python
2. Concept of Confusion Matrix

Contents of the Theory:
1. Artificial Neural Network
2. Keras
3. tensorflow
4. Normalization
5. Confusion Matrix

Artificial Neural Network:
The term "Artificial Neural Network" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.





The given figure illustrates the typical diagram of Biological Neural Network.

The typical Artificial Neural Network looks something like the given figure.

Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.
Relationship between Biological neural network and artificial neural network:

An Artifi**cial Neural Network** in held of Artifi**cial intelligence** where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.

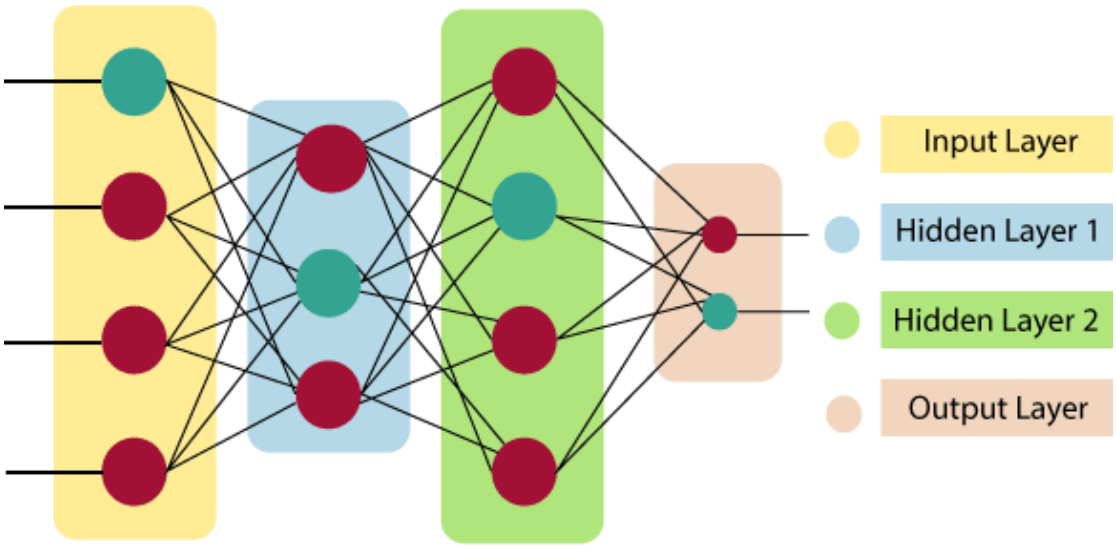
There are around 1000 billion neurons in the human brain. Each neuron has an association point somewhere in the range of 1,000 and 100,000. In the human brain, data is stored in such a manner as to be distributed, and we can extract more than one piece of this data when necessary, from our memory parallelly. We can say that the human brain is made up of incredibly amazing parallel processors.

We can understand the artificial neural network with an example, consider an example of a digital logic gate that takes an input and gives an output. "OR" gate, which takes two inputs. If one or both the inputs are "On," then we get "On" in output. If both the inputs are "Off," then we get "Off" in output. Here the output depends upon input. Our brain does not perform the same task. The outputs to inputs relationship keep changing because of the neurons in our brain, which are "learning."

The architecture of an artificial neural network:

To understand the concept of the architecture of an artificial neural network, we have to understand what a neural network consists of. In order to dene a neural network that consists of a large number of artificial neurons, which are termed units arranged in a sequence of layers. Lets us look at various types of layers available in an artificial neural network.

Artificial Neural Network primarily consists of three layers:



Input Layer:

As the name suggests, it accepts inputs in several different formats provided by the programmer.

Hidden Layer: The hidden layer presents in-between input and output layers. It performs all the calculations tond hidden features and patterns.

Output Layer:

The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.

The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

$$\sum_{i=1}^{n} Wi * Xi + b$$

It determines weighted total is passed as an input to an activation function to produce the output. Activation functions choose whether a node should re or not. Only those who are
red make it to the output layer. There are distinctive activation functions available that can be applied upon the sort of task we are performing.

Keras:

Keras is an open-source high-level Neural Network library, which is written in Python is capable enough to run on Theano, TensorFlow, or CNTK. It was developed by one of the Google engineers, Francois Chollet. It is made user-friendly, extensible, and modular for facilitating faster experimentation with deep neural networks. It not only supports Convolutional Networks and Recurrent Networks individually but also their combination.

It cannot handle low-level computations, so it makes use of the **Backend** library to resolve it. The backend library act as a high-level API wrapper for the low-level API, which lets it run on TensorFlow, CNTK, or Theano.

Initially, it had over 4800 contributors during its launch, which now has gone up to 250,000 developers. It has a 2X growth ever since every year it has grown. Big companies like Microsoft, Google, NVIDIA, and Amazon have actively contributed to the development of Keras. It has an amazing industry interaction, and it is used in the development of popular likes Netix, Uber, Google, Expedia, etc.

Tensorow:

TensorFlow is a Google product, which is one of the most famous deep learning tools widely used in the research area of machine learning and deep neural network. It came into the market on 9th November 2015 under the Apache License 2.0. It is built in such a way that it can easily run on multiple CPUs and GPUs as well as on mobile operating systems. It consists of various wrappers in distinct languages such as Java, C++, or Python.

Normalization:

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 formula:

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

Where,

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

Example: Let's assume we have a model dataset having maximum and minimum values of feature as mentioned above. To normalize the machine learning model, values are shifted and rescaled so their range can vary between 0 and 1. This technique is also known as Min-Max scaling. In this scaling technique, we will change the feature values as follows:

Case1-If the value of X is minimum, the value of Numerator will be 0; hence Normalization will also be 0.
Xn = (X - Xminimum) / (Xmaximum - Xminimum) ---------- formula
Put X =Xminimum in above formula, we get;
Xn = Xminimum- Xminimum/ (Xmaximum - Xminimum) Xn = 0
Case2-If the value of X is maximum, then the value of the numerator is equal to the denominator; hence Normalization will be 1.
Xn = (X - Xminimum) / (Xmaximum - Xminimum) Put X =Xmaximum in above formula, we get;
Xn = Xmaximum - Xminimum/ (Xmaximum - Xminimum) Xn = 1
Case3-On the other hand, if the value of X is neither maximum nor minimum, then values of normalization will also be between 0 and 1.
Hence, Normalization can be dened as a scaling method where values are shifted and rescaled to maintain their ranges between 0 and 1, or in other words; it can be referred to as Min-Max scaling technique.
Normalization techniques in Machine Learning
Although there are so many feature normalization techniques in Machine Learning, few of them are most frequently used. These are as follows:

● **Min-Max Scaling:**This technique is also referred to as scaling. As we have already discussed above, the Min-Max scaling method helps the dataset to shift and rescale the values of their attributes, so they end up ranging between 0 and 1.

● **Standardization scaling:**
Standardization scaling is also known as**Z-score**normalization, in which values are centered around the mean with a unit standard deviation, which means the attribute becomes zero and the resultant distribution has a unit standard deviation. Mathematically, we can calculate the standardization by subtracting the feature value from the mean and dividing it by standard deviation.
Hence, standardization can be expressed as follows:

$$X' = \frac{X - \mu}{\sigma}$$

Here, **μ** represents the mean of feature value, and **σ** represents the standard deviation of feature values.
However, unlike Min-Max scaling technique, feature values are not restricted to a specific range in the standardization technique.

This technique is helpful for various machine learning algorithms that use distance measures such as **KNN, K-means clustering, and Principal component analysis**, etc. Further, it is also important that the model is built on assumptions and data is normally distributed.

When to use Normalization or Standardization?
Which is suitable for our machine learning model, Normalization or Standardization? This is probably a big confusion among all data scientists as well as machine learning engineers. Although both terms have the almost same meaning choice of using normalization or standardization will depend on your problem and the algorithm you are using in models.
1. Normalization is a transformation technique that helps to improve the performance as well as the accuracy of your model better. Normalization of a machine learning model is useful when you don't know feature distribution exactly. In other words, the feature distribution of data does not follow a **Gaussian**(bell curve) distribution. Normalization must have an abounding range, so if you have outliers in data, they will be affected by Normalization.
Further, it is also useful for data having variable scaling techniques such as **KNN, artificial neural network**s. Hence, you can't use assumptions for the distribution of data.
2. Standardization in the machine learning model is useful when you are exactly aware of the feature distribution of data or, in other words, your data follows a Gaussian distribution. However, this does not have to be necessarily true. Unlike Normalization, Standardization does not necessarily have a bounding range, so if you have outliers in your data, they will not be affected by Standardization.

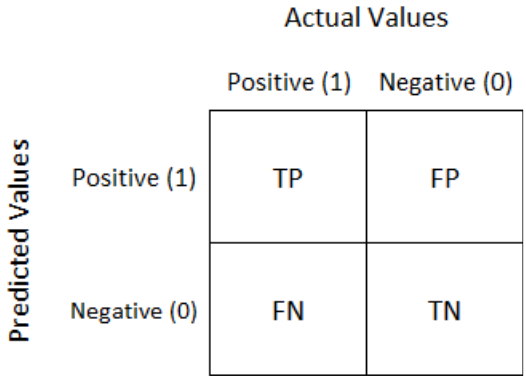
Further, it is also useful when data has variable dimensions and techniques such as **linear regression, logistic regression, and linear discriminant analysis**.
Example: Let's understand an experiment where we have a dataset having two attributes, i.e., age and salary. Where the age ranges from 0 to 80 years old, and the income varies from 0 to 75,000 dollars or more. Income is assumed to be 1,000 times that of age. As a result, the ranges of these two attributes are much different from one another.
Because of its bigger value, the attributed income will organically inuence the conclusion more when we undertake further analysis, such as multivariate linear regression. However, this does not necessarily imply that it is a better predictor. As a result, we normalize the data so that all of the variables are in the same range.
Further, it is also helpful for the prediction of credit risk scores where normalization is applied to all numeric data except the class column. It uses the **tanh transformation** technique, which converts all numeric features into values of range between 0 to 1.

Confusion Matrix:
The confusion matrix is a matrix used to determine the performance of the classification models for a given set 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, and so on.

● The matrix is divided into two dimensions, that are **predicted values** and **actual values** along with the total 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.

● It looks like the below table:



The above table has the following cases:
● **True Negative:** Model has given prediction No, and the real or actual value was also No.

● **True Positive:** The model has predicted yes, and the actual value was also true.

● **False Negative:** The model has predicted no, but the actual value was Yes, it is also called as **Type-II error**.

● **False Positive:** The model has predicted Yes, but the actual value was No. It is also called a **Type-I error.**

Need for Confusion Matrix in Machine learning
● It evaluates the performance of the classification models, when they make predictions on test data, and tells how good our classification model is.

● It not only tells the error made by the classifiers but also the type of errors such as it is either type-I or type-II error.

● With the help of the confusion matrix, we can calculate the different parameters for the model, such as accuracy, precision, etc.

Example: We can understand the confusion matrix using an example.
Suppose we are trying to create a model that can predict the result for the disease that is either a person has that disease or not. So, the confusion matrix for this is given as:

n = 100	Actual: No	Actual: Yes	
Predicted: No	TN: 65	FP: 3	68
Predicted: Yes	FN: 8	TP: 24	32
	73	27	

From the above example, we can conclude that

The table is given for the two-class classier, which has two predictions "Yes" and "NO." Here, Yes denes that patient has the disease, and No denes that patient does not has that disease.

● The classier has made a total of**100 predictions**. Out of 100 predictions,**89 are true predictions**, and**11 are incorrect predictions**.

● The model has given prediction "yes" for 32 times, and "No" for 68 times. Whereas the actual "Yes" was 27, and actual "No" was 73 times.

Calculations using Confusion Matrix:
We can perform various calculations for the model, such as the model's accuracy, using this matrix. These calculations are given below:

● Classification **Accuracy:**It is one of the important parameters to determine the accuracy of the classification problems. It denes how often the model predicts the correct output. It can be calculated as the ratio of the number of correct predictions made by the classier to all number of predictions made by the classifiers . The formula is given below:

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

● Misclassification **rate:**It is also termed as Error rate, and it denes how often the model gives the wrong predictions. The value of error rate can be calculated as the number of incorrect predictions to all number of the predictions made by the classier. The formula is given below:

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

● Precision:It can be dened as the number of correct outputs provided by the model

or out of all positive classes that have predicted correctly by the model, how many of them were actually true. It can be calculated using the below formula

$$\text{Precision}=\frac{TP}{TP+FP}$$

Recall: It is defined as the out of total positive classes, how our model predicted correctly. The recall must be as high as possible.

$$\text{Recall} = \frac{TP}{TP+FN}$$

F-measure: If two models have low precision and high recall or vice versa, it is dicult to compare these models. So, for this purpose, we can use F-score. This score helps us to evaluate the recall and precision at the same time. The F-score is
maximum if the recall is equal to the precision. It can be calculated using the below formula:

$$\text{F-measure} = \frac{2*Recall*Precision}{Recall+Precision}$$

Other important terms used in Confusion Matrix:

- **Null Error rate:**It denes how often our model would be incorrect if it always predicted the majority class. As per the accuracy paradox, it is said that "*the best classier has a higher error rate than the null error rate.*"

$$\text{F-measure} = \frac{2*Recall*Precision}{Recall+Precision}$$

- **ROC Curve:** The ROC is a graph displaying a classifier's performance for all possible thresholds. The graph is plotted between the true positive rate (on the Y-axis) and the false Positive rate (on the x-axis).

Conclusion:
In this way we build a a neural network-based classier that can determine whether they will leave or not in the next 6 months

Assignment Questions:
1) What is Normalization?

2) What is Standardization?

3) Explain Confusion Matrix ?

4) Dene the following: Classification Accuracy, Misclassification Rate, Precision.

5) One Example of Confusion Matrix?

Program

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

Out[48]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMe
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	

11/8/22, 11:00 AM ML_3_41157 - Jupyter Notebook

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 (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.

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

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

Preprocessing.

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

Out[51]:

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

10000 rows × 14 columns

In [52]:
```
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]:

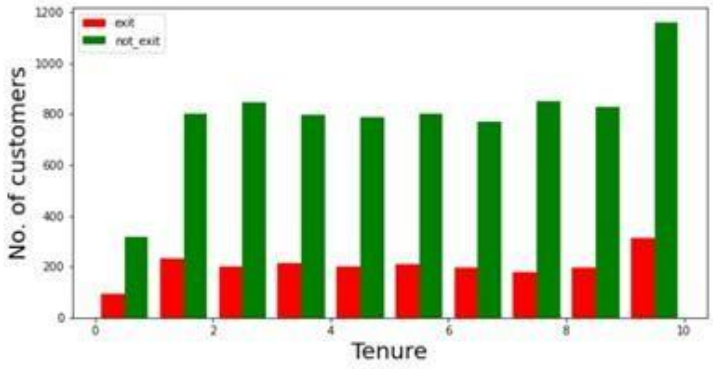
	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63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

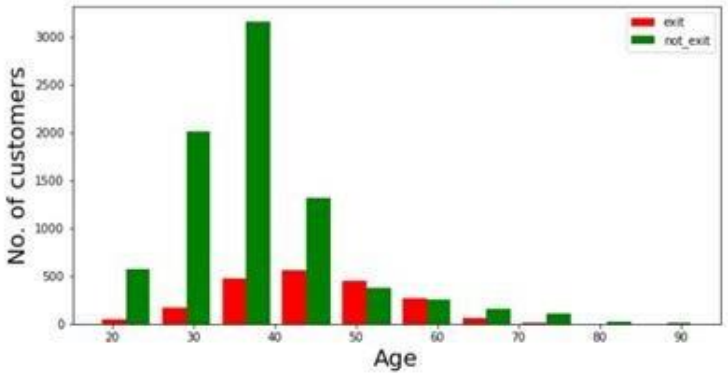
In [102]:
```
df_churn_exited = df[df['Exited']==1]['Tenure']
df_churn_not_exited = df[df['Exited']==0]['Tenure']
```

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','EstimatedS
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
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1	0
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1	0
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0	0

In [63]:
```
X = df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','
```

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

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

In [74]:
```
#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]:
```
classifier.add(Dense(activation = "relu",units = 6,kernel_initializer = "uniform"))  #Adding second hidden laye
```

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

In [77]:
```
classifier.compile(optimizer="adam",loss = 'binary_crossentropy',metrics = ['accuracy']) #To compile the Artific
```

In [79]:
```
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]:
```
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)
```

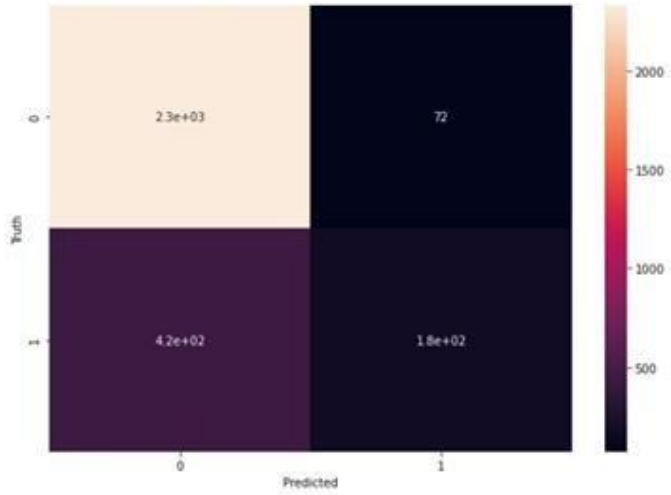
In [94]:
```
accuracy = accuracy_score(y_test,y_pred)
```

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
```

In [ ]:

Group B
Assignment No:4

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

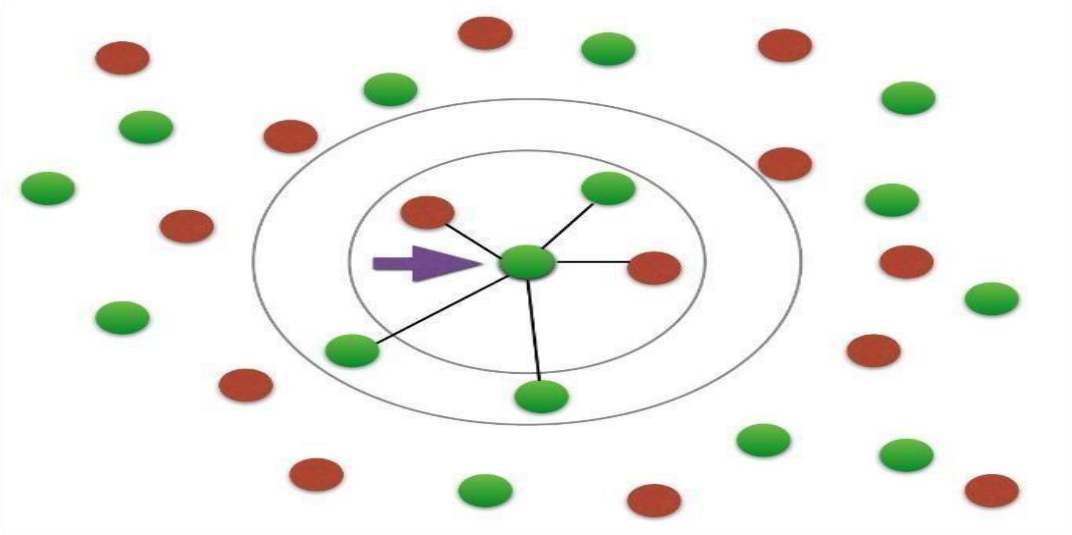
Dataset Description: We will try to build a machine learning model to accurately predict whether or not the patients in the dataset have diabetes or not?
The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

Link for Dataset: Diabetes predication system with KNN algorithm | Kaggle

Objective of the Assignment:
Students should be able to pre-process dataset and identify outliers, to check correlation and implement KNN algorithm and random forest classification models. Evaluate them with respective scores like confusion matrix, accuracy score, mean_squared_error, r2_score, roc_auc_score, roc_curve etc.

Prerequisite:

1. Basic knowledge of Python
2. Concept of Confusion Matrix
3. Concept of roc_auc curve.
4. Concept of Random Forest and KNN algorithms



k-Nearest-Neighbors (k-NN) is a supervised machine learning model. Supervised learning is when a model learns from data that is already labeled. A supervised learning model takes in a set of input objects and output values. The model then trains on that data to learn how to map the inputs to the desired output so it can learn to make predictions on unseen data.

k-NN models work by taking a data point and looking at the 'k' closest labeled data points. The data point is then assigned the label of the majority of the 'k' closest points.

For example, if k = 5, and 3 of points are 'green' and 2 are 'red', then the data point in question would be labeled 'green', since 'green' is the majority (as shown in the above graph).

Scikit-learn is a machine learning library for Python. In this tutorial, we will build a k-NN model using Scikit- learn to predict whether or not a patient has diabetes.

Reading in the training data

For our k-NN model, the first step is to read in the data we will use as input. For this example, we are using the diabetes dataset. To start, we will use Pandas to read in the data. I will not go into detail on Pandas, but it is a library you should become familiar with if you're looking to dive further into data science and machine learning.

```
Import  pandas as pd            #read in the data using pandas

df = pd.read_csv('data/diabetes_data.csv')          #check data has been
read in properlydf.head()
```

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age	diabetes
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

Next, let's see how much data we have. We will call the 'shape' function on our dataframe to see how many rows and columns there are in our data. The rows indicate the number of patients and the
columns indicate the number of features (age, weight, etc.) in the dataset for each patient.

#check number of rows and columns in dataset df.shape
Op → (768,9)

We can see that we have 768 rows of data (potential diabetes patients) and 9 columns (8 input features and 1 target output).

Split up the dataset into inputs and targets
Now let's split up our dataset into inputs (X) and our target (y). Our input will be every column except 'diabetes' because 'diabetes' is what we will be attempting to predict. Therefore, 'diabetes' will be our target.
We will use pandas 'drop' function to drop the column 'diabetes' from our dataframe and store it in the variable 'X'. This will be our input.

#create a dataframe with all training data except the target column
X = df.drop(columns=['diabetes'])*#check that the target variable has been removed*
X.head()

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age
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

We will insert the 'diabetes' column of our dataset into our target variable (y).
#Separate target values
y = df['diabetes'].values
#view target values y[0:5]

```
array([1, 0, 1, 0, 1])
```

Split the dataset into train and test data

Now we will split the dataset into into training data and testing data. The training data is the data that the model will learn from. The testing data is the data we will use to see how well the model performs on unseen data.

Scikit-learn has a function we can use called 'train_test_split' that makes it easy for us to split our dataset into training and testing data.

from sklearn.model_selection import train_test_split#split dataset into train and test data X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1,

train_test_split' takes in 5 parameters. The first two parameters are the input and target data we split up earlier. Next, we will set 'test_size' to 0.2. This means that 20% of all the data will be used for testing, which leaves 80% of the data as training data for the model to learn from. Setting 'random_state' to 1 ensures that we get the same split each time so we can reproduce our results.

Setting 'stratify' to y makes our training split represent the proportion of each value in the y variable. For example, in our dataset, if 25% of patients have diabetes and 75% don't have diabetes, setting 'stratify' to y will ensure that the random split has 25% of patients with diabetes and 75% of patients without diabetes.

Building and training the model Next, we have to build the model. Here is the code:

Next, we have to build the model. Here is the code:

from sklearn.neighbors import K Neighbors Classifier# Create KNN classifier knn = K Neighbors Classifier(n_neighbors = 3)# Fit the classifier to the data knn.fit(X_train,y_train)

First, we will create a new k-NN classifier and set 'n_neighbors' to 3. To recap, this means that if at least 2 out of the 3 nearest points to an new data point are patients without diabetes, then the new data point will be labeled as 'no diabetes', and vice versa. In other words, a new data point is labeled with by majority from the 3 nearest points.

We have set 'n_neighbors' to 3 as a starting point. We will go into more detail below on how to better

select a value for 'n_neighbors' so that the model can improve its performance.

Next, we need to train the model. In order to train our new model, we will use the 'fit' function and pass in our training data as parameters to fit our model to the training data.

Testing the model

Once the model is trained, we can use the 'predict' function on our model to make predictions on our test data. As seen when inspecting 'y' earlier, 0 indicates that the patient does not have diabetes and 1 indicates that the patient does have diabetes. To save space, we will only show print the first 5 predictions of our test set.

#show first 5 model predictions on the test data knn.predict(X_test)[0:5]

```
array([0, 0, 0, 0, 1])
```

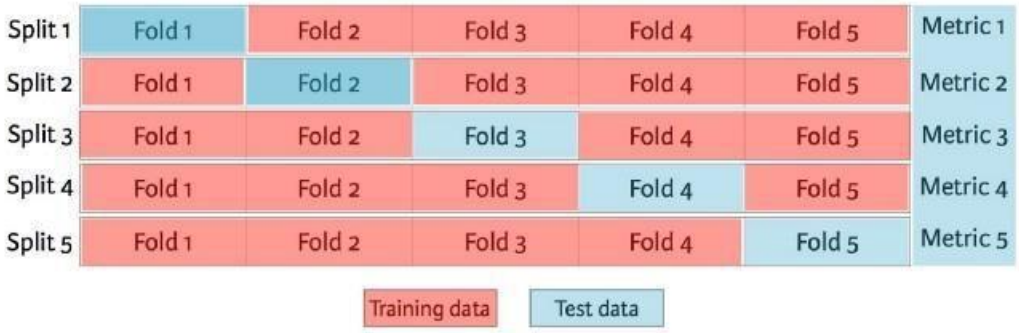
We can see that the model predicted 'no diabetes' for the first 4 patients in the test set and 'has diabetes' for the 5th patient.
Now let's see how our accurate our model is on the full test set. To do this, we will use the 'score' function and pass in our test input and target data to see how well our model predictions match up to the actual results

#check accuracy of our model on the test data knn.score(X_test, y_test)

```
0.6688311688311688
```

Our model has an accuracy of approximately 66.88%. It's a good start, but we will see how we can increase model performance below.
Congrats! You have now built an amazing k-NN model!
k-Fold Cross-Validation
Cross-validation is when the dataset is randomly split up into 'k' groups. One of the groups is used as the test set and the rest are used as the training set. The model is trained on the training set and scored on the test set. Then the process is repeated until each unique group as been used as the test set.

For example, for 5-fold cross validation, the dataset would be split into 5 groups, and the model would be trained and tested 5 separate times so each group would get a chance to be the test set. This can be seen in the graph below.



4- fold cross validation (image credit)

The train-test-split method we used in earlier is called 'holdout'. Cross-validation is better than using the holdout method because the holdout method score is dependent on how the data is split into train and test sets. Cross-validation gives the model an opportunity to test on multiple splits so we can get a better idea on how the model will perform on unseen data.

In order to train and test our model using cross-validation, we will use the 'cross_val_score' function with a cross-validation value of 5. 'cross_val_score' takes in our k-NN model and our data as parameters. Then it splits our data into 5 groups and fits and scores our data 5 seperate times, recording the accuracy score in an array each time. We will save the accuracy scores in the 'cv_scores' variable.

To find the average of the 5 scores, we will use numpy's mean function, passing in 'cv_score'. Numpy is a useful math library in Python

```
from sklearn.model_selection import cross_val_score import numpy as np#create a new KNN model
knn_cv = KNeighborsClassifier(n_neighbors=3)#train model with cv of 5
cv_scores = cross_val_score(knn_cv, X, y, cv=5)#print each cv score (accuracy) and average them
print(cv_scores)
print('cv_scores mean:{}'.format(np.mean(cv_scores)))
```

```
[ 0.68181818  0.69480519  0.75324675  0.75163399  0.68627451]
cv_scores mean:0.7135557253204311
```

Using cross-validation, our mean score is about 71.36%. This is a more accurate representation of how our model will perform on unseen data than our earlier testing using the holdout method.

Hypertuning model parameters using GridSearchCV
When built our initial k-NN model, we set the parameter 'n_neighbors' to 3 as a starting point with no real logic behind that choice.
Hypertuning parameters is when you go through a process to find the optimal parameters for your model to improve accuracy. In our case, we will use GridSearchCV to find the optimal value for 'n_neighbors'.
GridSearchCV works by training our model multiple times on a range of parameters that we specify. That way, we can test our model with each parameter and figure out the optimal values to get the best accuracy results.

For our model, we will specify a range of values for 'n_neighbors' in order to see which value works best for our model. To do this, we will create a dictionary, setting 'n_neighbors' as the key and using numpy to create an array of values from 1 to 24.
Our new model using grid search will take in a new k-NN classifier, our param_grid and a cross-validation value of 5 in order to find the optimal value for 'n_neighbors'

```
from sklearn.model_selection import GridSearchCV#create new a knn model
knn2 = KNeighborsClassifier()#create a dictionary of all values we want to test for n_neighbors
param_grid = {'n_neighbors': np.arange(1, 25)}#use gridsearch to test all values for n_neighbors
knn_gscv = GridSearchCV(knn2, param_grid, cv=5)#fit model to data
knn_gscv.fit(X, y)
```

After training, we can check which of our values for 'n_neighbors' that we tested performed the best. To do this, we will call 'best_params_' on our model.

#check top performing n_neighbors value knn_gscv.best_params

```
{'n_neighbors': 14}
```

We can see that 14 is the optimal value for ‘n_neighbors’. We can use the ‘best_score_’ function to check the accuracy of our model when ‘n_neighbors’ is 14. ‘best_score_’ outputs the mean accuracy of the scores obtained through cross-validation

#check mean score for the top performing value of n_neighbors knn_gscv.best_score_

```
0.7578125
```

By using grid search to find the optimal parameter for our model, we have improved our model accuracy by over 4%!

Code :- https://www.kaggle.com/code/shrutimechlearn/step-by-step-diabetes-classification-knn-detailed

Conclusion:

In this way we build a a neural network-based classifier that can determine whether they will leave or not in the next 6 months

KNN algorithm on diabetes dataset

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import metrics
```

In [2]:
```
df=pd.read_csv('diabetes.csv')
```

In [3]:
```
df.columns
```

Out[3]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'Pedigree', 'Age', 'Outcome'],
      dtype='object')
```

Check for null values. If present remove null values from the dataset

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

Outcome is the label/target, other columns are features

In [7]:
```
X = df.drop('Outcome',axis = 1)
y = df['Outcome']
```

In [8]:
```
from sklearn.preprocessing import scale
X = scale(X)
# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
```

In [9]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

In [17]:
```
print("Confusion matrix: ")
cs = metrics.confusion_matrix(y_test,y_pred)
print(cs)
```

```
Confusion matrix:
[[123  28]
 [ 37  43]]
```

In [12]:
```
print("Acccuracy ",metrics.accuracy_score(y_test,y_pred))
```

```
Acccuracy  0.7186147186147186
```

Classification error rate: proportion of instances misclassified over the whole set of instances. Error rate is calculated as the total number of two incorrect predictions (FN + FP) divided by the total number of a dataset (examples in the dataset.

Also error_rate = 1- accuracy

11/8/22, 11:01 AM ML_Assignmet_5_ - Jupyter Notebook

In [29]:
```
total_misclassified = cs[0,1] + cs[1,0]
print(total_misclassified)
total_examples = cs[0,0]+cs[0,1]+cs[1,0]+cs[1,1]
print(total_examples)
print("Error rate",total_misclassified/total_examples)
print("Error rate ",1-metrics.accuracy_score(y_test,y_pred))
```

```
65
231
Error rate 0.2813852813852814
Error rate  0.2813852813852814
```

In [13]:
```
print("Precision score",metrics.precision_score(y_test,y_pred))
```

```
Precision score 0.6056338028169014
```

In [14]:
```
print("Recall score ",metrics.recall_score(y_test,y_pred))
```

```
Recall score  0.5375
```

In [15]:
```
print("Classification report ",metrics.classification_report(y_test,y_pred))
```

```
Classification report               precision    recall  f1-score   support

           0       0.77      0.81      0.79       151
           1       0.61      0.54      0.57        80

    accuracy                           0.72       231
   macro avg       0.69      0.68      0.68       231
weighted avg       0.71      0.72      0.71       231
```

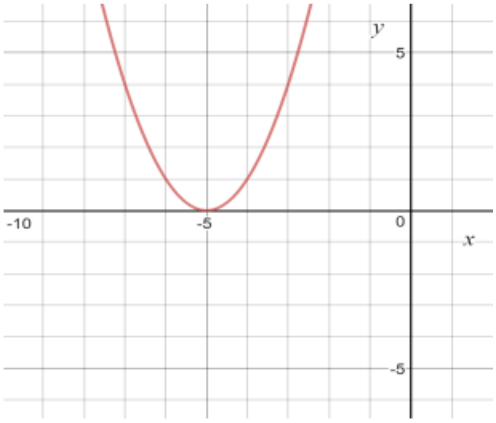
Group B

Assignment No:5

Title of Assignment: Implement Gradient Descent Algorithm.

Problem Statement: Develop a program in Python to create a Gradient Descent.
Example by hand :

Question : Find the local minima of the function $y=(x+5)^2$ starting from the point x=3



Solution : We know the answer just by looking at the graph. $y = (x+5)^2$ reaches it's minimum value when x = -5 (i.e when x=-5, y=0). Hence x=-5 is the local and global minima of the function.

Now, let's see how to obtain the same numerically using gradient descent.
Step 1 : Initialize x =3. Then, find the gradient of the function, dy/dx = 2*(x+5).
Step 2 : Move in the direction of the negative of the gradient (Why?). But wait, how much to move? For that, we require a learning rate. Let us assume the **learning rate → 0.01**
Step 3 : Let's perform 2 iterations of gradient descent
Step 4: We can observe that the X value is slowly decreasing and should converge to -5 (the local minima). However, how many iterations should we perform?
Let us set a precision variable in our algorithm which calculates the difference between two consecutive "x" values. If the difference between x values from 2 consecutive iterations is lesser than the precision we set, stop the algorithm!
Step 5: We can observe that the X value is slowly decreasing and should converge to -5 (the local minima).
However, how many iterations should we perfor?
Let us set a precision variable in our algorithm which calculates the difference between two consecutive "x" values. If the difference between x values from 2 consecutive iterations is lesser than the precision we set, stop the algorithm!

Gradient descent in Python :

Step 1 : Initialize parameters

cur_x = 3 # The algorithm starts at x=3 rate = 0.01 # Learning rate

precision = 0.000001 #This tells us when to stop the algorithm previous_step_size = 1 #

max_iters = 10000 # maximum number of iterations iters = 0 #iteration counter

df = lambda x: 2*(x+5) #Gradient of our function

Step 2 : Run a loop to perform gradient descent :

i. Stop loop when difference between x values from 2 consecutive iterations is less than 0.000001 or when number of iterations exceeds 10,000

while previous_step_size > precision and iters < max_iters: prev_x = cur_x #Store current x value in prev_x

cur_x = cur_x - rate * df(prev_x) #Grad descent previous_step_size = abs(cur_x - prev_x) #Change in x iters = iters+1 #iteration count

print("Iteration",iters,"\nX value is",cur_x) #Print

iterations print("The local minimum occurs at", cur_x)

Facilities:

Google Colab , Jupiter notebook

Input:

To find the local minima of the given function from its starting point.

Output:

Finds the optimal solution by taking a step in the direction of the maximum rate of decrease of the function.

Conclusion:

Hence, we have successfully studied implementation of Gradient Descent Algorithm successfully.