EXPERIMENT -01

Aim : To perform preprocessing for the given dataset**.**

Description :

Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.

Need of Data Preprocessing

- For achieving better results from the applied model in Machine Learning projects the format of the data has to be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values have to be managed from the original raw data set.
- Another aspect is that the data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithm are executed in one data set, and best out of them is chosen.

Dataset Description :

The Carseats data set tracks sales information for car seats. It has 400 observations (each at a different store) and 11 variables:

- CompPrice: price charged by competitor at each location
- Sales : unit sales in thousands
- Income: community income level in 1000s of dollars
- Advertising: local ad budget at each location in 1000s of dollars
- Population: regional pop in thousands
- Price: price for car seats at each site
- ShelveLoc: Bad, Good or Medium indicates quality of shelving location
- Age: age level of the population
- Education: ed level at location
- Urban: Yes/No
- US: Yes/No

Code :

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
```

Importing all the required packages using import statement.

numpy -> numerical python used mainly for working with arrays,linear algebra and matrices.

pandas -> an open source library which can be used to analyze data easily in Python. Pandas provide an easy way to create, manipulate, and wrangle the data.

scikit-learn -> sklearn package for selection of efficient tools like classification, regression, clustering and dimensionality reduction etc.
matplotlib ->for data visualization and graphical plotting.

```
dt= pd.read_csv('Carseats.csv')
x=dt.iloc[:, 1:11]
y=dt.iloc[:, 0]
```

The iloc() function in python is defined in the Pandas module, which helps us select a specific row or column from the data set. Here in x we are selecting all rows and columns from 1 to 11 and in y we are selecting all rows and first column i.e., 0.

```
print(x)

     CompPrice  Income  Advertising  Population  Price ShelveLoc  Age  \
0          138      73           11         276    120       Bad   42
1          111      48           16         260     83      Good   65
2          113      35           10         269     80    Medium   59
3          117     100            4         466     97    Medium   55
4          141      64            3         340    128       Bad   38
..         ...     ...          ...         ...    ...       ...  ...
395        138     108           17         203    128      Good   33
396        139      23            3          37    120    Medium   55
397        162      26           12         368    159    Medium   40
398        100      79            7         284     95       Bad   50
399        134      37            0          27    120      Good   49

     Education Urban   US
0           17   Yes  Yes
1           10   Yes  Yes
2           12   Yes  Yes
3           14   Yes  Yes
```

```
     4            13   Yes   No
     ..          ...   ...  ...
     395          14   Yes  Yes
     396          11    No  Yes
     397          18   Yes  Yes
     398          12   Yes  Yes
     399          16   Yes  Yes

     [400 rows x 10 columns]
```

```
print(x.head())
```

```
        CompPrice  Income  Advertising  Population  Price ShelveLoc  Age  \
     0        138      73           11         276    120       Bad   42
     1        111      48           16         260     83      Good   65
     2        113      35           10         269     80    Medium   59
     3        117     100            4         466     97    Medium   55
     4        141      64            3         340    128       Bad   38

        Education Urban   US
     1         17   Yes  Yes
     2         10   Yes  Yes
     3         12   Yes  Yes
     4         14   Yes  Yes
     5         13   Yes   No
```

The head() function displays the first five rows of the dataframe by default.

```
print(y)
```

```
     0       9.50
     1      11.22
     2      10.06
     3       7.40
     4       4.15
            ...
     395    12.57
     396     6.14
     397     7.41
     398     5.94
     399     9.71
     Name: Sales, Length: 400, dtype: float64
```

```
print(y.head())
```

```
     0     9.50
     1    11.22
     2    10.06
     3     7.40
     4     4.15
     Name: Sales, dtype: float64
```

```
le=LabelEncoder()
x.iloc[:,8]=le.fit_transform(x.iloc[:,8])
x.iloc[:,9]=le.fit_transform(x.iloc[:,9])
x.head()
```

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban
0	138	73	11	276	120	Bad	42	17	1
1	111	48	16	260	83	Good	65	10	1
2	113	35	10	269	80	Medium	59	12	1
3	117	100	4	466	97	Medium	55	14	1
4	141	64	3	340	128	Bad	38	13	1

For coverting the labels which have categorical values into numerical values, we use Label Encoder.

```
ohe=OneHotEncoder()
x1=pd.DataFrame(ohe.fit_transform(x[['ShelveLoc']]).toarray())
x1.head()
```

	0	1	2
0	1.0	0.0	0.0
1	0.0	1.0	0.0
2	0.0	0.0	1.0
3	0.0	0.0	1.0
4	1.0	0.0	0.0

One-hot encoding converts the categorical data into numeric data by splitting the column into multiple columns. The numbers are replaced by 1s and 0s, depending on which column has what value.

```
x=x.join(x1)
x.head()
```

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban
0	138	73	11	276	120	Bad	42	17	1
1	111	48	16	260	83	Good	65	10	1

We join or concatenate the converted column to the entire dataset using join().

```
x.drop('ShelveLoc',axis=1,inplace=True)
x.head()
```

	CompPrice	Income	Advertising	Population	Price	Age	Education	Urban	US	0	
0	138	73	11	276	120	42	17	1	1	1.0	0.
1	111	48	16	260	83	65	10	1	1	0.0	1.
2	113	35	10	269	80	59	12	1	1	0.0	0.
3	117	100	4	466	97	55	14	1	1	0.0	0.
4	141	64	3	340	128	38	13	1	0	1.0	0

We can drop the redundant columns or the columns which are not of use.

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

Dividing the dataset into test dataset and trainning data by using train_test_split(), here 20% of the data is taken for testing and remaining 80% for training.

```
sc=StandardScaler()
x_train.iloc[:,0:7]=sc.fit_transform(x_train.iloc[:,0:7])
x_test.iloc[:,0:7]=sc.fit_transform(x_test.iloc[:,0:7])
```

```
x_train.head()
```

	CompPrice	Income	Advertising	Population	Price	Age	Education	Urba
336	0.885481	-1.189418	-0.089216	-1.418971	1.155793	-1.530974	1.574353	
64	-1.614507	-0.031327	0.807613	-0.570074	-0.507142	-1.283981	0.799287	
55	1.214427	0.475337	-0.238687	-1.418971	1.624826	0.506723	1.574353	
106	-1.482929	-1.261799	-0.986044	-0.344158	0.985236	1.062458	1.574353	
300	-0.561881	0.366766	-0.836573	-0.748069	-0.720339	-0.481252	-1.138378	

.

EXPERIMENT -02

Aim : To convert continuous values to categorical values for the given dataset**.**

Description :

Numerical data such as continuous, highly skewed data is frequently seen in data analysis. Sometimes analysis becomes effortless on conversion from continuous to discrete data. There are many ways in which conversion can be done, one such way is by using Pandas' integrated cut-function. Pandas' cut function is a distinguished way of converting numerical continuous data into categorical data. It has 3 major necessary parts:

1. First and foremost is the 1-D array/DataFrame required for input.
2. The other main part is bins. Bins that represent boundaries of separate bins for continuous data. The first number denotes the start point of the bin and the following number denotes the endpoint of the bin. Cut function permits more explicitness of the bins
3. The final main part is labels. The number of labels without exception will be one lower than the number of bins.

Dataset Description :

The Carseats data set tracks sales information for car seats. It has 400 observations (each at a different store) and 11 variables:

- CompPrice: price charged by competitor at each location
- Sales : unit sales in thousands
- Income: community income level in 1000s of dollars
- Advertising: local ad budget at each location in 1000s of dollars
- Population: regional pop in thousands
- Price: price for car seats at each site
- ShelveLoc: Bad, Good or Medium indicates quality of shelving location
- Age: age level of the population
- Education: ed level at location
- Urban: Yes/No
- US: Yes/No

Code :

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
```

Importing all the required packages using import statement.

numpy -> numerical python used mainly for working with arrays,linear algebra and matrices.
pandas -> an open source library which can be used to analyze data easily in Python. Pandas provide an easy way to create, manipulate, and wrangle the data.
scikit-learn -> sklearn package for selection of efficient tools like classification, regression, clustering and dimensionality reduction etc.
matplotlib ->for data visualization and graphical plotting.

```
dt = pd.read_csv('Carseats.csv')
X = dt.iloc[:, 1:11].values
y = dt.iloc[:, 0].values
```

The iloc() function in python is defined in the Pandas module, which helps us select a specific row or column from the data set. Here in x we are selecting all rows and columns from 1 to 11 and in y we are selecting all rows and first column i.e., 0.

```
X

     array([[138, 73, 11, ..., 17, 'Yes', 'Yes'],
            [111, 48, 16, ..., 10, 'Yes', 'Yes'],
            [113, 35, 10, ..., 12, 'Yes', 'Yes'],
            ...,
            [162, 26, 12, ..., 18, 'Yes', 'Yes'],
            [100, 79, 7, ..., 12, 'Yes', 'Yes'],
            [134, 37, 0, ..., 16, 'Yes', 'Yes']], dtype=object)
```

Values of x dataset

```
y

     array([ 9.5 , 11.22, 10.06,  7.4 ,  4.15, 10.81,  6.63, 11.85,  6.54,
             4.69,  9.01, 11.96,  3.98, 10.96, 11.17,  8.71,  7.58, 12.29,
            13.91,  8.73,  6.41, 12.13,  5.08,  5.87, 10.14, 14.9 ,  8.33,
        5.27,  2.99,  7.81, 13.55,  8.25,  6.2 ,  8.77,  2.67, 11.07,
        8.89,  4.95,  6.59,  3.24,  2.07,  7.96, 10.43,  4.12,  4.16,
        4.56, 12.44,  4.38,  3.91, 10.61,  1.42,  4.42,  7.91,  6.92,
        4.9 ,  6.85, 11.91,  0.91,  5.42,  5.21,  8.32,  7.32,  1.82,
        8.47,  7.8 ,  4.9 ,  8.85,  9.01, 13.39,  7.99,  9.46,  6.5 ,
        5.52, 12.61,  6.2 ,  8.55, 10.64,  7.7 ,  4.43,  9.14,  8.01,
        7.52, 11.62,  4.42,  2.23,  8.47,  8.7 , 11.7 ,  6.56,  7.95,
        5.33,  4.81,  4.53,  8.86,  8.39,  5.58,  9.48,  7.45, 12.49,
        4.88,  4.11,  6.2 ,  5.3 ,  5.07,  4.62,  5.55,  0.16,  8.55,
        3.47,  8.98,  9.  ,  6.62,  6.67,  6.01,  9.31,  8.54,  5.08,
        8.8 ,  7.57,  7.37,  6.87, 11.67,  6.88,  8.19,  8.87,  9.34,
       11.27,  6.52,  4.96,  4.47,  8.41,  6.5 ,  9.54,  7.62,  3.67,
        6.44,  5.17,  6.52, 10.27, 12.3 ,  6.03,  6.53,  7.44,  0.53,
        9.09,  8.77,  3.9 , 10.51,  7.56, 11.48, 10.49, 10.77,  7.64,
        5.93,  6.89,  7.71,  7.49, 10.21, 12.53,  9.32,  4.67,  2.93,
        3.63,  5.68,  8.22,  0.37,  6.71,  6.71,  7.3 , 11.48,  8.01,
       12.49,  9.03,  6.38,  0.  ,  7.54,  5.61, 10.48, 10.66,  7.78,
        4.94,  7.43,  4.74,  5.32,  9.95, 10.07,  8.68,  6.03,  8.07,
       12.11,  8.79,  6.67,  7.56, 13.28,  7.23,  4.19,  4.1 ,  2.52,
        3.62,  6.42,  5.56,  5.94,  4.1 ,  2.05,  8.74,  5.68,  4.97,
        8.19,  7.78,  3.02,  4.36,  9.39, 12.04,  8.23,  4.83,  2.34,
        5.73,  4.34,  9.7 , 10.62, 10.59,  6.43,  7.49,  3.45,  4.1 ,
        6.68,  7.8 ,  8.69,  5.4 , 11.19,  5.16,  8.09, 13.14,  8.65,
        9.43,  5.53,  9.32,  9.62,  7.36,  3.89, 10.31, 12.01,  4.68,
        7.82,  8.78, 10.  ,  6.9 ,  5.04,  5.36,  5.05,  9.16,  3.72,
        8.31,  5.64,  9.58,  7.71,  4.2 ,  8.67,  3.47,  5.12,  7.67,
        5.71,  6.37,  7.77,  6.95,  5.31,  9.1 ,  5.83,  6.53,  5.01,
       11.99,  4.55, 12.98, 10.04,  7.22,  6.67,  6.93,  7.8 ,  7.22,
        3.42,  2.86, 11.19,  7.74,  5.36,  6.97,  7.6 ,  7.53,  6.88,
        6.98,  8.75,  9.49,  6.64, 11.82, 11.28, 12.66,  4.21,  8.21,
        3.07, 10.98,  9.4 ,  8.57,  7.41,  5.28, 10.01, 11.93,  8.03,
        4.78,  5.9 ,  9.24, 11.18,  9.53,  6.15,  6.8 ,  9.33,  7.72,
        6.39, 15.63,  6.41, 10.08,  6.97,  5.86,  7.52,  9.16, 10.36,
        2.66, 11.7 ,  4.69,  6.23,  3.15, 11.27,  4.99, 10.1 ,  5.74,
        5.87,  7.63,  6.18,  5.17,  8.61,  5.97, 11.54,  7.5 ,  7.38,
        7.81,  5.99,  8.43,  4.81,  8.97,  6.88, 12.57,  9.32,  8.64,
       10.44, 13.44,  9.45,  5.3 ,  7.02,  3.58, 13.36,  4.17,  3.13,
        8.77,  8.68,  5.25, 10.26, 10.5 ,  6.53,  5.98, 14.37, 10.71,
       10.26,  7.68,  9.08,  7.8 ,  5.58,  9.44,  7.9 , 16.27,  6.81,
        6.11,  5.81,  9.64,  3.9 ,  4.95,  9.35, 12.85,  5.87,  5.32,
        8.67,  8.14,  8.44,  5.47,  6.1 ,  4.53,  5.57,  5.35, 12.57,
        6.14,  7.41,  5.94,  9.71])
```

Values of y dataset

```
avg=y.mean()
print(avg)
```

```
    7.496325000000001
```

Calculating the mean or average of the values of y dataset which is a required parameter for the conversion of continous to categorical values.

```
maxi=y.max()
print(maxi)
```

```
    16.27
```

Calculating the maximum value of the values of y dataset which is a required parameter for the conversion of continous to categorical values.

```
y=pd.cut(y, bins=[0,avg,maxi], right=True, labels=['Bad','Good'], retbins=False, precision=3,
print(y)
```

```
    ['Good', 'Good', 'Good', 'Bad', 'Bad', ..., 'Good', 'Bad', 'Bad', 'Bad', 'Good']
    Length: 400
    Categories (2, object): ['Bad' < 'Good']
```

Pandas cut function is used for converting numerical continuous data into categorical data. Here the values of 'y' dataset are in numerical form, we have converted these values into two categorical values 'Good','Bad' based upon the calculae average and maximum values of y.

- If the value of 'y' lies in the range between 0 and average value, then it is categorized as 'Bad'.
- Else if the value of 'y' lies in the range between average value and maximum value, then it is categorized as 'Good'.
- The bins parameter denotes the bin boundaries for segmentation.
- The right parameter denotes whether rightmost edge of bins should be included or not.It is of Boolean type and default value is True.
- The labels parameter defines labels for returned segmented bins.

```
print(y)
```

```
    ['Good', 'Good', 'Good', 'Bad', 'Bad', ..., 'Good', 'Bad', 'Bad', 'Bad', 'Good']
    Length: 400
    Categories (2, object): ['Bad' < 'Good']
```

EXPERIMENT -03

Aim : To perform linear regression on the given dataset**.**

Description :

The term regression is used when you try to find the relationship between variables. In Machine Learning, and in statistical modeling, that relationship is used to predict the outcome of future events. Linear regression uses the relationship between the data-points to draw a straight line through all them. It is a statistical method for modeling relationships between a dependent variable with a given set of independent variables. It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).

Dataset Description :

The Carseats data set tracks sales information for car seats. It has 400 observations (each at a different store) and 11 variables:

- CompPrice: price charged by competitor at each location
- Sales : unit sales in thousands
- Income: community income level in 1000s of dollars
- Advertising: local ad budget at each location in 1000s of dollars
- Population: regional pop in thousands
- Price: price for car seats at each site
- ShelveLoc: Bad, Good or Medium indicates quality of shelving location
- Age: age level of the population
- Education: ed level at location
- Urban: Yes/No
- US: Yes/No

Code :

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, RobustScaler
import matplotlib.pyplot as plt
import seaborn as sb
```

Importing all the required packages using import statement.

numpy -> numerical python used mainly for working with arrays,linear algebra and matrices.

pandas -> an open source library which can be used to analyze data easily in Python. Pandas provide an easy way to create, manipulate, and wrangle the data.

scikit-learn -> sklearn package for selection of efficient tools like classification, regression, clustering and dimensionality reduction etc.
matplotlib ->for data visualization and graphical plotting.

```
df=pd.read_csv('Carseats.csv')
x=df.iloc[:,1:11].values
y=df.iloc[:,0].values
```

The iloc() function in python is defined in the Pandas module, which helps us select a specific row or column from the data set. Here in x we are selecting all rows and columns from 1 to 11 and in y we are selecting all rows and first column i.e., 0.

```
x
```

```
     array([[138, 73, 11, ..., 17, 'Yes', 'Yes'],
            [111, 48, 16, ..., 10, 'Yes', 'Yes'],
            [113, 35, 10, ..., 12, 'Yes', 'Yes'],
            ...,
            [162, 26, 12, ..., 18, 'Yes', 'Yes'],
            [100, 79, 7, ..., 12, 'Yes', 'Yes'],
            [134, 37, 0, ..., 16, 'Yes', 'Yes']], dtype=object)
```

Values of x dataset

```
df.corr()
```

	Sales	CompPrice	Income	Advertising	Population	Price	Age
Sales	1.000000	0.064079	0.151951	0.269507	0.050471	-0.444951	-0.231815
CompPrice	0.064079	1.000000	-0.080653	-0.024199	-0.094707	0.584848	-0.100239
Income	0.151951	-0.080653	1.000000	0.058995	-0.007877	-0.056698	-0.004670
Advertising	0.269507	-0.024199	0.058995	1.000000	0.265652	0.044537	-0.004557
Population	0.050471	-0.094707	-0.007877	0.265652	1.000000	-0.012144	-0.042663
Price	-0.444951	0.584848	-0.056698	0.044537	-0.012144	1.000000	-0.102177
Age	-0.231815	-0.100239	-0.004670	-0.004557	-0.042663	-0.102177	1.000000
Education	-0.051955	0.025197	-0.056855	-0.033594	-0.106378	0.011747	0.006488

corr() is used to find the pairwise correlation of all columns in the Pandas Dataframe in Python.

```
k=df[['Advertising']]
l=df[['Sales']]
```

```
print(k)
```

```
     Advertising
0             11
1             16
2             10
3              4
4              3
..           ...
395           17
396            3
397           12
398            7
399            0

[400 rows x 1 columns]
```

```
print(l)
```

```
     Sales
0     9.50
1    11.22
2    10.06
3     7.40
4     4.15
..     ...
395  12.57
396   6.14
397   7.41
398   5.94
399   9.71

[400 rows x 1 columns]
```

```
X_train,X_test,y_train,y_test=train_test_split(k,l,test_size=0.2,random_state=0)
```

Dividing the dataset into test dataset and trainning data by using train_test_split(), here 20% of the data is taken for testing and remaining 80% for training.

```
from sklearn.linear_model import LinearRegression
```

Linear Regression is a type of Regression algorithms that models the relationship between a dependent variable and a single independent variable.
It is an approach for predicting a response using a single feature. It is assumed that the two

variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).

```
r=LinearRegression()
r.fit(X_train,y_train)
```

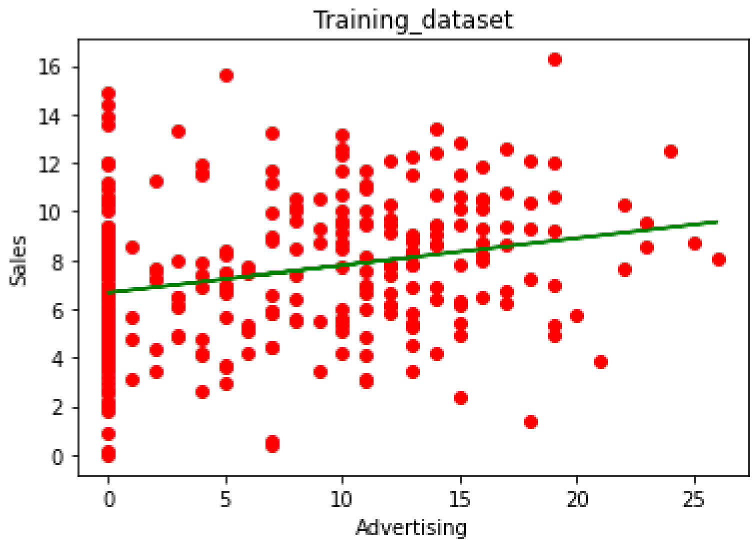
```
LinearRegression()
```

Linear regression uses the relationship between the data-points to draw a straight line through all them. This line can be used to predict future values. Here x and y are k and l respectively.

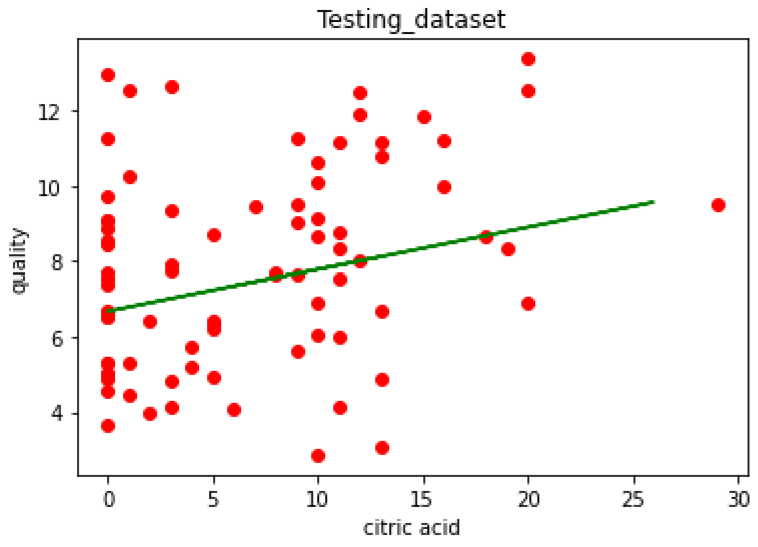
```
y_predict=r.predict(X_test)
y_predict
```

```
array([[7.67649322],
       [8.12173552],
       [6.67469806],
       [7.3425615 ],
       [8.90090953],
       [8.78959896],
       [6.67469806],
       [7.11994035],
       [6.67469806],
       [7.00862978],
       [7.23125093],
       [6.67469806],
       [7.89911437],
       [8.90090953],
       [6.89731921],
       [7.11994035],
       [6.67469806],
       [7.7878038 ],
       [6.67469806],
       [7.23125093],
       [7.67649322],
       [6.67469806],
       [6.67469806],
       [6.67469806],
       [6.67469806],
       [7.00862978],
       [6.78600863],
       [7.23125093],
       [7.89911437],
       [6.67469806],
       [8.01042494],
       [8.45566724],
       [7.67649322],
       [6.67469806],
       [6.67469806],
       [7.7878038 ],
       [6.67469806],
       [7.23125093],
       [7.00862978],
       [6.67469806],
       [7.89911437],
       [7.7878038 ],
       [6.67469806],
       [8.45566724],
       [8.12173552],
       [8.12173552],
       [7.45387208],
       [7.56518265],
       [6.78600863],
       [8.67828839],
       [6.67469806],
       [6.67469806],
       [7.00862978],
       [8.01042494],
       [7.7878038 ],
       [7.7878038 ],
       [7.00862978],
```

```
plt.scatter(X_train, y_train, color = "red")
plt.plot(X_train, r.predict(X_train), color = "green")
plt.title("Training_dataset")
plt.xlabel("Advertising")
plt.ylabel("Sales")
plt.show()
```



```
plt.scatter(X_test, y_test, color = "red")
plt.plot(X_train, r.predict(X_train), color = "green")
plt.title("Testing_dataset")
plt.xlabel("citric acid")
plt.ylabel("quality")
plt.show()
```



Error in linear regression :

There are three error metrics that are commonly used for evaluating and reporting the performanceof a regression model. They are:

- Mean Squared Error (MSE).
- Root Mean Squared Error
- (RMSE)Mean Absolute Error
  (MAE)

```
from sklearn import metrics
print(metrics.mean_absolute_error(y_test,y_predict

     ))2.0516368159520124
```

Calculation of Mean Absolute Error (MAE).

```
print(metrics.mean_squared_error(y_test,y_predict

     ))6.3927399636374895
```

Calculation of Mean Squared Error (MSE).

```
print(np.sqrt(metrics.mean_squared_error(y_test,y_predict)

     ))2.5283868303005947
```

Calculation of Root Mean Squared Error (MSE).

EXPERIMENT -04

Aim : To perform Multiple Linear Regression for the given dataset**.**

Description :

Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression is to model the linear relationship between the explanatory (independent) variables and response (dependent) variables. In essence, multiple regression is the extension of ordinary least-squares (OLS) regression because it involves more than one explanatory variable.

Dataset Description :

This particular dataset holds data from 50 startups in New York, California, and Florida. The features in this dataset are R&D spending, Administration Spending, Marketing Spending, and location features, while the target variable is: Profit.

1. R&D spending: The amount which startups are spending on Research and development.
2. Administration spending: The amount which startups are spending on the Admin panel.
3. Marketing spending: The amount which startups are spending on marketing strategies.
4. State: To which state that particular startup belongs.
5. Profit: How much profit that particular startup is making

Code :

```
import numpy as np
import pandas as pd
from numpy import math
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

```
dt=pd.read_csv('50_Startups.csv')
```

Reading the dataset using read_csv function

```
len(dt)
```

```
     50
```

The len() function returns the number of items in an object.

```
dt.head()
```

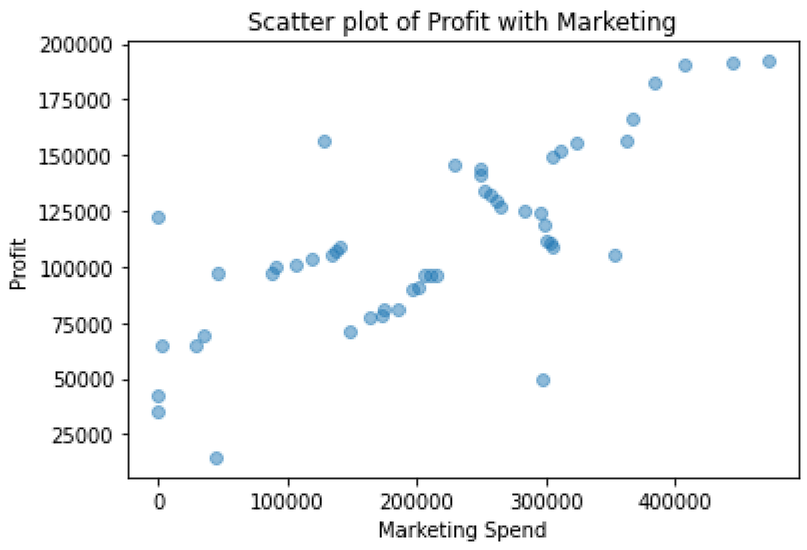
	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

```
dt.shape
```

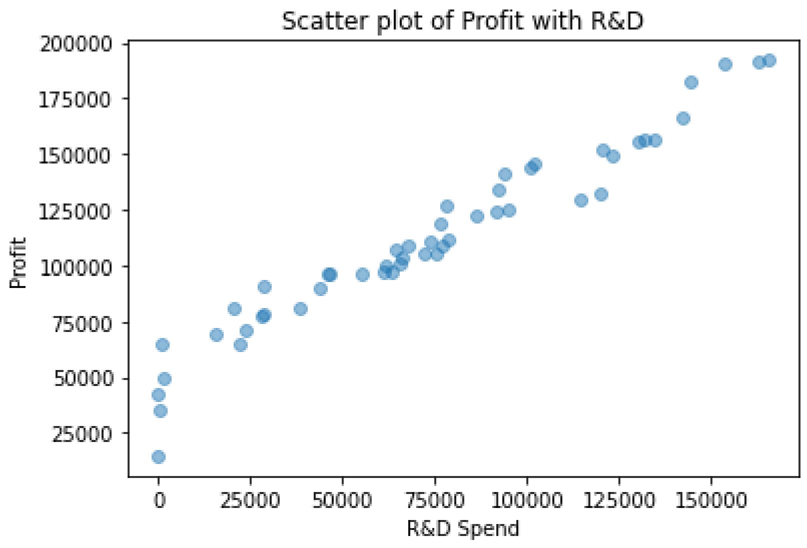
```
(50, 5)
```

the shape() method is used to fetch the dimensions of Pandas and NumPy type objects in python.

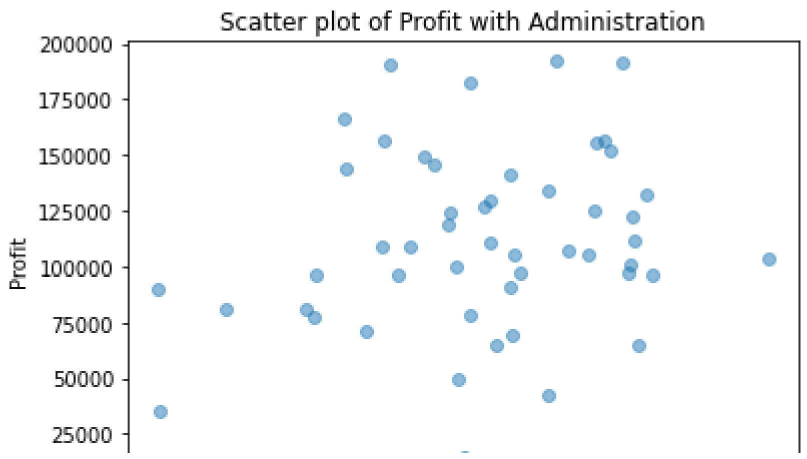
```
plt.scatter(dt['Marketing Spend'],dt ['Profit'],·alpha=0.5)
plt.title('Scatter·plot·of·Profit·with·Marketing')
plt.xlabel('Marketing·Spend')
plt.ylabel("Profit")
plt.show()
```



```
plt.scatter(dt['R&D Spend'],dt ['Profit'], alpha=0.5)
plt.title('Scatter plot of Profit with R&D')
plt.xlabel('R&D Spend')
plt.ylabel("Profit")
plt.show()
```

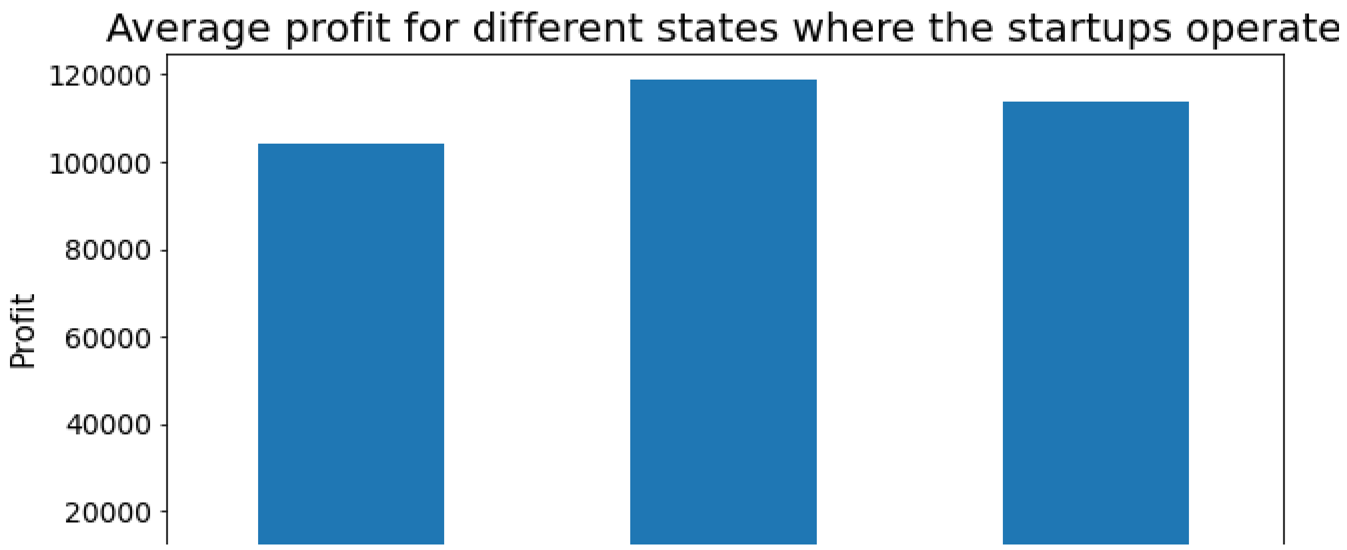


```
plt.scatter(dt['Administration'],dt ['Profit'], alpha=0.5)
plt.title('Scatter plot of Profit with Administration')
plt.xlabel('Administration')
plt.ylabel("Profit")
plt.show()
```



```
ax = dt.groupby(['State'])['Profit'].mean().plot.bar(
figsize = (10,5), fontsize = 14)
ax.set_title("Average profit for different states where the startups operate", fontsize=20)
ax.set_xlabel("State", fontsize = 15)
ax.set_ylabel("Profit", fontsize = 15)
```

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



```
dt.State.value_counts()
```

```
    New York      17
    California    17
    Florida       16
    Name: State, dtype: int64
```

```
dt['NewYork_State'] = np.where(dt['State']== 'New York', 1, 0)
dt['California_State'] = np.where(dt['State']=='California', 1, 0)
dt['Florida_State'] = np.where(dt['State']== 'Florida', 1, 0)
dt.drop(columns=['State'],axis=1, inplace=True)
```

```
dv='Profit'
iv=dt.columns.tolist()
```

```
iv.remove(dv
)iv

     ['R&D Spend',
      'Administration',
      'Marketing
      Spend',
      'NewYork_State',
      'California_State'
      ,'Florida State']


X=dt[iv].value
s
y=dt[dv].value
s


x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)


sc=MinMaxScaler()
x_train=sc.fit_transform(x_train
)x_test=sc.transform(x_test)


r=LinearRegression()
r.fit(x_train,y_train)

     LinearRegression(

     )


y_pred=r.predict(x_test)


math.sqrt(mean_squared_error(y_test,y_pred)

     )9137.990152794944


r2_score(y_test,y_pred

     )

     0.934706847328242

     5
```

EXPERIMENT - 05

Aim : To perform decision tree classification for the given dataset**.**

Description :

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves.

- Decision tree algorithm falls under the category of supervised learning. They can be used to solve both regression and classification problems.
- Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.
- We can represent any boolean function on discrete attributes using the decision tree.

Dataset Description :

The data set pizza. csv contains measurements that capture the kind of things that make a pizza tasty. Can you determine which pizza brand works best for you and explain why. The variables in the data set are:
brand -- Pizza brand (class label)
id -- Sample analysed
mois -- Amount of water per 100 grams in the sample
prot -- Amount of protein per 100 grams in the sample
fat -- Amount of fat per 100 grams in the sample
ash -- Amount of ash per 100 grams in the sample
sodium -- Amount of sodium per 100 grams in the sample
carb -- Amount of carbohydrates per 100 grams in the sample
cal -- Amount of calories per 100 grams in the sample

Code :

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, RobustScaler
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
import seaborn as sb
```

Importing all the required packages using import statement.

numpy -> numerical python used mainly for working with arrays,linear algebra and matrices.
pandas -> an open source library which can be used to analyze data easily in Python. Pandas provide an easy way to create, manipulate, and wrangle the data.
scikit-learn -> sklearn package for selection of efficient tools like classification, regression, clustering and dimensionality reduction etc.
matplotlib ->for data visualization and graphical plotting.

```
df=pd.read_csv('Pizza.csv')
x=df.iloc[:,1:8]
y=df.iloc[:,0]
```

The iloc() function in python is defined in the Pandas module, which helps us select a specific row or column from the data set. Here in x we are selecting all rows and columns from 1 to 11 and in y we are selecting all rows and first column i.e., 0.

```
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

Dividing the dataset into test dataset and trainning data by using train_test_split(), here 20% of the data is taken for testing and remaining 80% for training

```
sc=StandardScaler()
X_train.iloc[:,:]=sc.fit_transform(X_train.iloc[:,:])
X_test.iloc[:,:]=sc.fit_transform(X_test.iloc[:,:])
```

```
print(X_train)
```

```
                id      mois      prot       fat       ash    sodium      carb
     134  0.412836 -0.369959 -0.850236 -0.760297 -0.953703 -0.594200  0.945070
     145 -0.990029 -0.350380 -0.792500 -0.821021 -0.787417 -0.484227  0.931629
```

```
63  -0.994676  0.802727  1.911735 -0.107235  0.669565 -0.319267 -1.117583
293  0.411006  0.531710 -0.527226 -0.284822 -0.248967 -0.209293  0.056835
285  1.814011  0.675978 -0.491336 -0.447515 -0.106436  0.010654  0.034994
..        ...       ...       ...       ...       ...       ...       ...
251 -1.003407  1.439563 -0.383666 -0.882890 -0.343988 -0.429240 -0.189025
192 -0.995803 -1.432384 -0.753489 -0.576982 -0.890356 -0.566707  1.393668
117  0.413400  0.579112  1.142440  0.559576  1.516832  0.093134 -1.105822
47   0.404528  0.975847 -0.037249  0.982348  0.867525  1.165374 -1.058778
172  1.813589 -1.257202 -0.886126 -0.406269 -0.914111 -0.539213  1.264297

[240 rows x 7 columns]
```

```
from sklearn.tree import DecisionTreeClassifier
d=DecisionTreeClassifier(criterion='entropy',random_state=0)
```

```
d.fit(X_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
y_predict=d.predict(X_test)
y_predict
```

```
array(['E', 'G', 'A', 'H', 'I', 'E', 'E', 'H', 'B', 'D', 'J', 'A', 'G',
       'I', 'A', 'C', 'J', 'D', 'E', 'F', 'H', 'D', 'E', 'E', 'E', 'D',
       'F', 'J', 'H', 'J', 'D', 'C', 'A', 'E', 'H', 'F', 'G', 'F', 'J',
       'A', 'H', 'D', 'C', 'B', 'B', 'E', 'E', 'C', 'A', 'B', 'C', 'H',
       'F', 'D', 'B', 'I', 'F', 'A', 'J', 'F'], dtype=object)
```

```
from sklearn import metrics
```

```
m=metrics.accuracy_score(y_test,y_predict)
print(m*100)
```

```
85.0
```

Calculating the accuracy score metrics

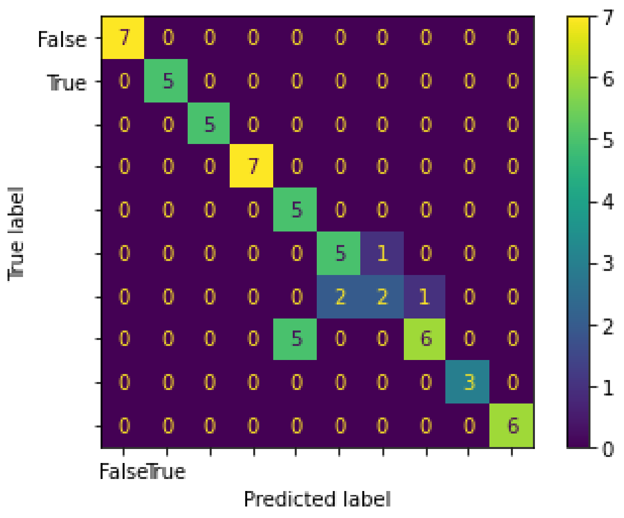
```
cm = metrics.confusion_matrix(y_test, y_predict)
```

Calculating the confusion matrix

```
print(cm)
```

```
[[7 0 0 0 0 0 0 0 0 0]
 [0 5 0 0 0 0 0 0 0 0]
 [0 0 5 0 0 0 0 0 0 0]
 [0 0 0 7 0 0 0 0 0 0]
 [0 0 0 0 5 0 0 0 0 0]
 [0 0 0 0 0 5 1 0 0 0]
 [0 0 0 0 0 2 2 1 0 0]
 [0 0 0 0 5 0 0 6 0 0]
 [0 0 0 0 0 0 0 0 3 0]
 [0 0 0 0 0 0 0 0 0 6]]
```

```
import matplotlib.pyplot as plt



cm_display=metrics.ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[False,True])
cm_display.plot()
plt.show()
```



EXPERIMENT - 05

Aim : To perform decision tree regression for the given dataset**.**

Description :

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves.

- Decision tree algorithm falls under the category of supervised learning. They can be used to solve both regression and classification problems.
- Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.
- We can represent any boolean function on discrete attributes using the decision tree.

Dataset Description :

The data set `'Height_Age_Dataset.csv'` contains the heights an ages of different people. The variables in the data set are:
Height – height of the person in cm
Age – age of the person

Code :

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, RobustScaler
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
import seaborn as sb


# Import the Height Weight Dataset
data = pd.read_csv('Height_Age_Dataset.csv')
data.head()
```

	Age	Height
0	10	138
1	11	138
2	12	138
3	13	139

```
    4   14     139
```

```
#Store the data in the form of dependent and independent variables separately
X = data.iloc[:, 0:1].values
y = data.iloc[:, 1].values
```

```
#Split the Dataset into Training and Test Dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

```
#Import the Decision Tree Regressor
from sklearn.tree import DecisionTreeRegressor

#Create a decision tree regressor object  from DecisionTreeRegressor class
DtReg = DecisionTreeRegressor(random_state = 0)

#Fit the decision tree regressor with training data represented by X_train and y_train
DtReg.fit(X_train, y_train)
```

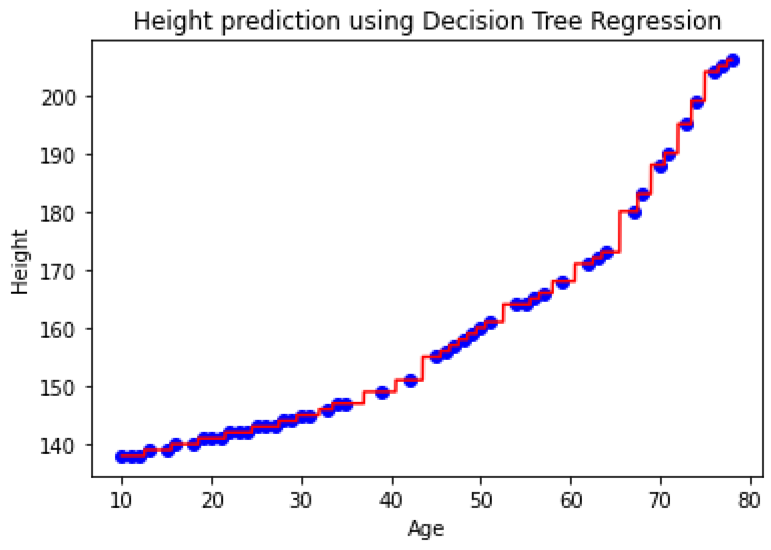
```
     DecisionTreeRegressor(random_state=0)
```

```
#Predicted Height from test dataset w.r.t Decision Tree Regression
y_predict_dtr = DtReg.predict((X_test))
```

```
#Model Evaluation using R-Square for Decision Tree Regression
from sklearn import metrics
 r_square = metrics.r2_score(y_test, y_predict_dtr)
 print('R-Square Error associated with Decision Tree Regression is:', r_square)
```

```
      R-Square Error associated with Decision Tree Regression is: 0.9941828370498541
```

```
 ''' Visualise the Decision Tree Regression by creating range of values from min value of X_tr
 having a difference of 0.01 between two consecutive values'''
 X_val = np.arange(min(X_train), max(X_train), 0.01)

 #Reshape the data into a len(X_val)*1 array in order to make a column out of the X_val values
 X_val = X_val.reshape((len(X_val), 1))

 #Define a scatter plot for training data
 plt.scatter(X_train, y_train, color = 'blue')

 #Plot the predicted data
 plt.plot(X_val, DtReg.predict(X_val), color = 'red')

 #Define the title
 plt.title('Height prediction using Decision Tree Regression')

 #Define X axis label
 plt.xlabel('Age')

 #Define Y axis label
 plt.ylabel('Height')

 #Set the size of the plot for better clarity
 plt.figure(figsize=(1,1))

 #Draw the plot
 plt.show()
```



```
#Import export_graphviz package
from sklearn.tree import export_graphviz

#Store the decision tree in a tree.dot file in order to visualize the plot.
#Visualize it on http://www.webgraphviz.com/ by copying and pasting related data from dtregre
export_graphviz(DtReg, out_file ='dtregression.dot',
               feature_names =['Age'])
```

```
# Predicting Height based on Age using Decision Tree Regression
height_pred = DtReg.predict([[41]])
print("Predicted Height: % d"% height_pred)
```

```
Predicted Height:  151
```

\

\

EXPERIMENT -06

Aim : To perform Linear Discriminant Analysis for the given dataset**.**

Description :

Linear Discriminant Analysis (LDA) is one of the commonly used sdimensionality reduction techniques in machine learning to solve more than two-class classification problems. It is also known as Normal Discriminant Analysis (NDA) or Discriminant Function Analysis (DFA).This can be used to project the features of higher dimensional space into lower-dimensional space in order to reduce resources and dimensional costs. In this topic, "Linear Discriminant Analysis (LDA) in machine learning", we will discuss the LDA algorithm for classification predictive modeling problems, limitation of logistic regression, representation of linear Discriminant analysis model, how to make a prediction using LDA, how to prepare data for LDA, extensions to LDA and much more.
Although the logistic regression algorithm is limited to only two-class, linear Discriminant analysis is applicable for more than two classes of classification problems.
Linear Discriminant analysis is one of the most popular dimensionality reduction techniques used for supervised classification problems in machine learning. It is also considered a pre-processing step for modeling differences in ML and applications of pattern classification.

Dataset Description :

The Iris dataset was used in R.A. Fisher's classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems, and can also be found on the UCI Machine Learning Repository.
It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.
The columns in this dataset are:

- Id
- SepalLengthCm
- SepalWidthCm
- PetalLengthCm
- PetalWidthCm
- Species

Code :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

Importing all the required packages using import statement.
numpy -> numerical python used mainly for working with arrays.
pandas -> package which can be used to analyze data easily.
scikit-learn -> sklearn package for selection of efficient tools like classification, regression, clustering and dimensionality reduction etc.

\ \

matplotlib ->for data visualization and graphical plotting.

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
cls = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
dataset = pd.read_csv(url, names=cls)
```

Specifying the column or attribute names and Reading the dataset from URL.

```
dataset.head()
```

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

head() method prints the first 5 rows in the dataset

```
X = dataset.iloc[:, 0:4].values
y = dataset.iloc[:, 4].values
```

\ \

iloc() function is used to retrieve the rows and columns and we divide the dataset into class and target variable.

```
print(X,y)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6  1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.1 1.5 0.1]
 [4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [5.  3.5 1.3 0.3]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
```

\
\

```
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
```

```
sc = StandardScaler()
X = sc.fit_transform(X)
le = LabelEncoder()
y = le.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

We preprocess the dataset and divide into train and test sets by using the train_test_split method by performing testing with 20% data. The remaining 80% will be for training.
Here we preprocess the data as there are no missing values and noise in the data so we directly perform StandardScaler, It is used to resize the distribution of values so that the mean of the observed values is 0 and the standard deviation is 1. The labelEncoder is used to encode the independent call variable i.e, the Iris feature.
fit_transform() method is used to calculate the mean and standard deviation of the feature at a time it will transform the data points of the feature.

```
print(X,y)
```

```
[-9.00681170e-01 -1.28197243e+00 -4.31421141e-01 -1.29719997e-01]
[-1.73673948e-01 -5.87763531e-01  1.94101603e-01  1.33225943e-01]
[ 5.53333275e-01 5.69251294e-01 1.27454998e+00 1.71090158e+00][-
5.25060772e-02 -8.19166497e-01  7.62758643e-01  9.22063763e-01][
1.52267624e+00 -1.24957601e-01  1.21768427e+00  1.18500970e+00][
5.53333275e-01 -3.56360566e-01  1.04708716e+00  7.90590793e-01][
7.95669016e-01 -1.24957601e-01  1.16081857e+00  1.31648267e+00][
2.12851559e+00  -1.24957601e-01 1.61574420e+00 1.18500970e+00] [-
1.14301691e+00 -1.28197243e+00  4.21564419e-01  6.59117823e-01][
1.76501198e+00 -3.56360566e-01  1.44514709e+00  7.90590793e-01][
1.03800476e+00 -1.28197243e+00  1.16081857e+00  7.90590793e-01][
1.64384411e+00  1.26346019e+00  1.33141568e+00  1.71090158e+00][
7.95669016e-01  3.37848329e-01  7.62758643e-01  1.05353673e+00][
6.74501145e-01 -8.19166497e-01  8.76490051e-01  9.22063763e-01][
1.15917263e+00  -1.24957601e-01 9.90221459e-01 1.18500970e+00] [-
1.73673948e-01  -1.28197243e+00 7.05892939e-01 1.05353673e+00] [-
5.25060772e-02 -5.87763531e-01  7.62758643e-01  1.57942861e+00][
6.74501145e-01  3.37848329e-01  8.76490051e-01  1.44795564e+00][
7.95669016e-01 -1.24957601e-01  9.90221459e-01  7.90590793e-01][
2.24968346e+00  1.72626612e+00  1.67260991e+00  1.31648267e+00][
2.24968346e+00 -1.05056946e+00  1.78634131e+00  1.44795564e+00][
1.89829664e-01 -1.97618132e+00  7.05892939e-01  3.96171883e-01][
1.28034050e+00  3.37848329e-01  1.10395287e+00  1.44795564e+00]
```

\

\

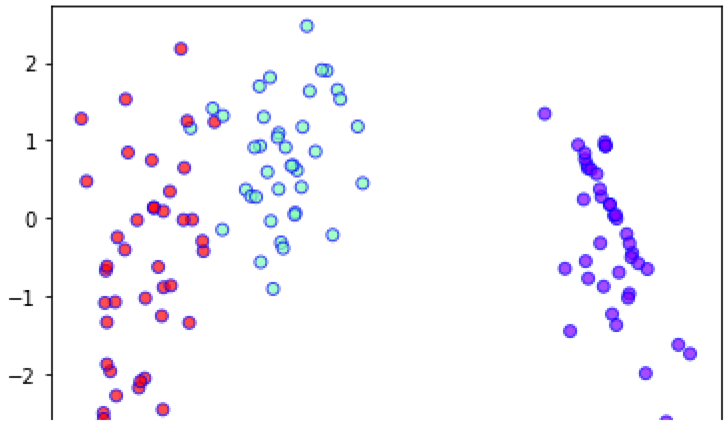
```
[-2.94841818e-01 -5.87763531e-01  6.49027235e-01  1.05353673e+00]
[ 2.24968346e+00 -5.87763531e-01  1.67260991e+00  1.05353673e+00]
[ 5.53333275e-01 -8.19166497e-01  6.49027235e-01  7.90590793e-01]
[ 1.03800476e+00  5.69251294e-01  1.10395287e+00  1.18500970e+00]
[ 1.64384411e+00  3.37848329e-01  1.27454998e+00  7.90590793e-01]
[ 4.32165405e-01 -5.87763531e-01  5.92161531e-01  7.90590793e-01]
[ 3.10997534e-01 -1.24957601e-01  6.49027235e-01  7.90590793e-01]
[ 6.74501145e-01 -5.87763531e-01  1.04708716e+00  1.18500970e+00]
[ 1.64384411e+00 -1.24957601e-01  1.16081857e+00  5.27644853e-01]
[ 1.88617985e+00 -5.87763531e-01  1.33141568e+00  9.22063763e-01]
[ 2.49201920e+00  1.72626612e+00  1.50201279e+00  1.05353673e+00]
[ 6.74501145e-01 -5.87763531e-01  1.04708716e+00  1.31648267e+00]
[ 5.53333275e-01 -5.87763531e-01  7.62758643e-01  3.96171883e-01]
[ 3.10997534e-01 -1.05056946e+00  1.04708716e+00  2.64698913e-01]
[ 2.24968346e+00 -1.24957601e-01  1.33141568e+00  1.44795564e+00]
[ 5.53333275e-01  8.00654259e-01  1.04708716e+00  1.57942861e+00]
[ 6.74501145e-01  1.06445364e-01  9.90221459e-01  7.90590793e-01]
[ 1.89829664e-01 -1.24957601e-01  5.92161531e-01  7.90590793e-01]
[ 1.28034050e+00  1.06445364e-01  9.33355755e-01  1.18500970e+00]
[ 1.03800476e+00  1.06445364e-01  1.04708716e+00  1.57942861e+00]
[ 1.28034050e+00  1.06445364e-01  7.62758643e-01  1.44795564e+00]
[-5.25060772e-02 -8.19166497e-01  7.62758643e-01  9.22063763e-01]
[ 1.15917263e+00  3.37848329e-01  1.21768427e+00  1.44795564e+00]
[ 1.03800476e+00  5.69251294e-01  1.10395287e+00  1.71090158e+00]
[ 1.03800476e+00 -1.24957601e-01  8.19624347e-01  1.44795564e+00]
[ 5.53333275e-01 -1.28197243e+00  7.05892939e-01  9.22063763e-01]
[ 7.95669016e-01 -1.24957601e-01  8.19624347e-01  1.05353673e+00]
[ 4.32165405e-01  8.00654259e-01  9.33355755e-01  1.44795564e+00]
[ 6.86617933e-02 -1.24957601e-01  7.62758643e-01  7.90590793e-01]] [0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0  0 0 0 0 1 1 1 1 1  1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 2  2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2  2 2 2 2 2 2 2 2 2  2 2 2 2 2 2 2 2 2  2 2 2 2 2 2 2 2 2
2  2]
```

```
lda = LinearDiscriminantAnalysis(n_components=2)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
```

Now we apply Linear Discriminant Analysis(lda)
Here in SKLearn we already have a built in method of LinearDiscriminantAnalysis(). So we give x train and y train to the model And then test with x test to the model.

```
plt.scatter(
    X_train[:,0],X_train[:,1],c=y_train,cmap='rainbow',
  alpha=0.7,edgecolors='b'
)
```

```
<matplotlib.collections.PathCollection at 0x7fad177da290>
```



plot the scatterplot

Now we plot the graph using matplotlib.pyplot.scatter and we plot the graph as the x train data on 0th coloumn and x train on 1st column.

classify using random forest classifier

```
classifier = RandomForestClassifier(max_depth=2, random_state=0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)


print('Accuracy : ' + str(accuracy_score(y_test, y_pred)))
conf_m = confusion_matrix(y_test, y_pred)
print(conf_m)

     Accuracy : 0.9333333333333333
     [[ 9  0  0]
      [ 0 10  1]
      [ 0  1  9]]
```

print the accuracy and confusion matrix

By using the testing predicted output we calculate the accuracy and confusion matrix for the model to see how accurate the model is.

EXPERIMENT - 07

Aim : To perform support vector machine analysis for the given dataset**.**

Description :

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.
The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.
SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

Dataset Description :

- Digits Dataset is a part of sklearn library. Sklearn comes loaded with datasets to practice machine learning techniques and digits is one of them.
- Digits has 64 numerical features(8×8 pixels) and a 10 class target variable(0-9). Digits dataset can be used for classification as well as clustering. Digits dataset is first step to image recognition.
- Each datapoint is a 8x8 image of a digit.

Classes	10
Samples per class	~180
Samples total	1797
Dimensionality	64
Features	integers 0-16

Code :

```
from sklearn.metrics import classification_report
from sklearn import datasets
from skimage import exposure
import numpy as np
import cv2
import imutils
from sklearn.model_selection import train_test_split


mnist=datasets.load_digits()


traindata,testdata,trainlabels,testlabels=train_test_split(np.array(mnist.data),mnist.target,


traindata,valdata,trainlabels,vallabels=train_test_split(traindata,trainlabels,test_size=0.1,
print("training data points : {}".format(len(trainlabels)))
```

```
    training data points : 1212
```

```
print("validation data points : {}".format(len(vallabels)))
```

```
    validation data points : 135
```

```
print("testing data points : {}".format(len(testlabels)))
```

```
    testing data points : 450
```

```
from sklearn.svm import SVC
model= SVC(C=0.5, kernel= 'linear')
model.fit(traindata, trainlabels)
score= model.score(valdata, vallabels)
print (score*100)
```

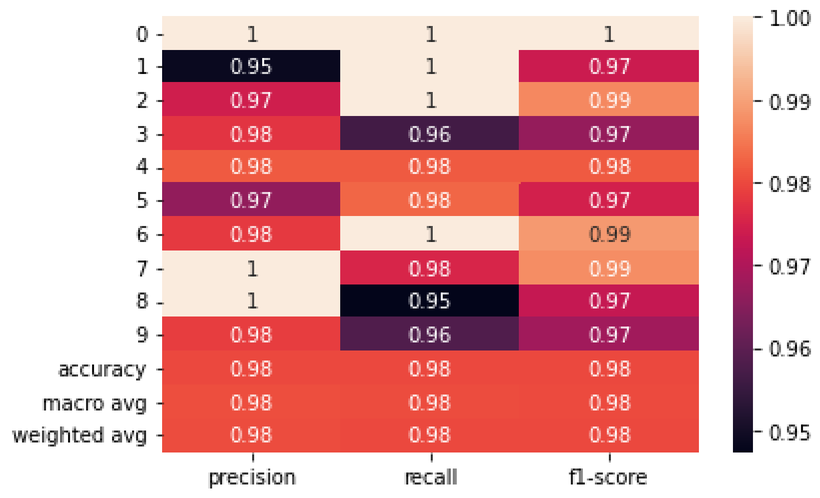
```
    98.51851851851852
```

```
predictions = model.predict(testdata)
print("EVALUATION ON TESTING DATA")
report = (classification_report(testlabels, predictions, output_dict=True))
print(report)
```

```
    EVALUATION ON TESTING DATA
    {'0': {'precision': 1.0, 'recall': 1.0, 'f1-score': 1.0, 'support': 43}, '1': {'precisi
```

```
import seaborn as sb
import pandas as pd
sb.heatmap(pd.DataFrame(report).iloc[:-1,:].T,annot=True)
```

```
    <matplotlib.axes._subplots.AxesSubplot at 0x7f127e1ca090>
```



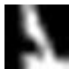
```
from google.colab.patches import cv2_imshow
for i in list(map(int, np.random.randint(0, high=len(testlabels), size=(5,)))):
  image= testdata[i]
  prediction= model.predict(image.reshape(1, -1))[0]
  image =image.reshape((8, 8)).astype("uint8")
  image =exposure.rescale_intensity(image, out_range=(0, 255))
  image= imutils.resize(image, width=32, inter=cv2.INTER_CUBIC)
  print("I think that digit is: {}".format(prediction))
  cv2_imshow(image)
```

```
    I think that digit is: 6I
```

```
    think that digit is: 5
```

```
    I think that digit is: 8I
```

```
    think  that  digit  is:  1 I
```



```
    think that digit is: 7
```

EXPERIMENT - 08

Aim : To perform Bayesian Classification for the given dataset**.**

Description :

Bayesian classification is a probabilistic approach to learning and inference based on a different view of what it means to learn from data, in which probability is used to represent uncertainty about the relationship being learnt. In numerous applications, the connection between the attribute set and the class variable is non- deterministic. In other words, we can say the class label of a test record can't be assumed with certainty even though its attribute set is the same as some of the training examples. These circumstances may emerge due to the noisy data or the presence of certain confusing factors that influence classification, but it is not included in the analysis. Bayesian classification uses Bayes theorem to predict the occurrence of any event. Bayesian classifiers are the statistical classifiers with the Bayesian probability understandings.

Dataset Description :

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. Attribute Information:

a) radius (mean of distances from center to points on the perimeter)
b) texture (standard deviation of gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2 / area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave points (number of concave portions of the contour)
i) symmetry
j) fractal dimension ("coastline approximation" - 1)

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

Code :

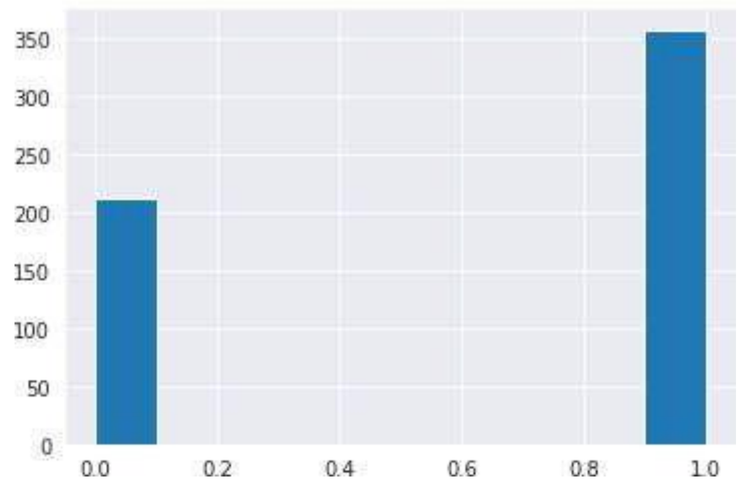
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")


data= pd.read_csv("cancer_data.csv")
data.head(10)
```

```
     mean_radius  mean_texture  mean_perimeter mean_area  mean_smoothness diagnosis
```

0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0
5	12.45	15.70	82.57	477.1	0.12780	0
6	18.25	19.98	119.60	1040.0	0.09463	0
7	13.71	20.83	90.20	577.9	0.11890	0
8	13.00	21.82	87.50	519.8	0.12730	0
9	12.46	24.04	83.97	475.9	0.11860	0

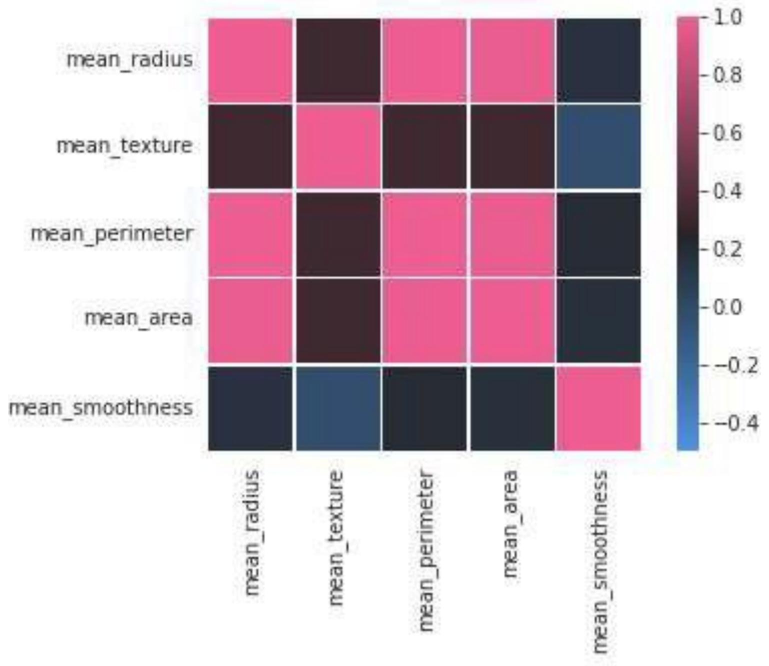
```
data["diagnosis"].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb834219c50>
```



```
corr= data.iloc[:,:-1].corr(method="pearson")
cmap = sns.diverging_palette(250,354,80,60,center='dark',as_cmap=True)
sns.heatmap(corr, vmax=1, vmin=-.5, cmap=cmap, square=True, linewidths=.2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb834159150>
```



```
data= data[["mean_radius", "mean-texture", "mean-smoothness", "diagnosis"]]
data.head(10)
```

	mean_radius	mean_texture	mean_smoothness	diagnosis
0	17.99	10.38	0.11840	0
1	20.57	17.77	0.08474	0
2	19.69	21.25	0.10960	0
3	11.42	20.38	0.14250	0
4	20.29	14.34	0.10030	0
5	12.45	15.70	0.12780	0
6	18.25	19.98	0.09463	0
7	13.71	20.83	0.11890	0
8	13.00	21.82	0.12730	0
9	12.46	24.04	0.11860	0

```
fig, axes= plt.subplots(1, 3, figsize=(18, 6), sharey=True)
sns.histplot(data, ax=axes[0], x="mean_radius'", kde=True, color='r')
sns.histplot(data, ax=axes[1], x="mean_smoothness'', kde=True, color='b')
sns.histplot(data, ax=axes[2], x="mean_texture", kde=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb831359250>
```

```
def calculate_prior(df, Y):
    classes= sorted(list(df[Y].unique())) prior=[]
    for i in classes:
        prior.append(len(df[df[Y]==i])/len(df))
    return prior


data["cat_mean_radius"] = pd.cut(data["mean_radius"].values, bins= 3, labels= [0,1,2])
data["cat_mean_texture"] = pd.cut(data["mean_texture"].values, bins = 3, labels = [0,1,2])
data["cat_mean_smoothness"] = pd.cut(data["mean_smoothness"].values, bins= 3, labels= [0,1,

data= data.drop(columns=["mean_radius", "mean_texture", "mean_smoothness"])
data= data[["cat_mean_radius", "cat_mean_texture", "cat_mean_smoothness", "diagnosis"]]
data.head(10)
```

	cat mean radius	cat mean texture	cat mean smoothness	diagnosis
0	1	0	1	0
1	1	0	0	0
2	1	1	1	0
3	0	1	2	0
4	1	0	1	0
5	0	0	2	0
6	1	1	1	0
7	0	1	1	0
8	0	1	2	0
9	0	1	1	0

```
def calculate_likelihood_categorical(df, feat_name, feat_val, Y, label):
    feat= list(df.columns)
    df = df[df[Y]==label]
    p_x_given_y = len(df[df[feat_name]==feat_val]) / len(df)
    return p_x_given_y


def naive_bayes_categorical(df, X, Y):
    # get feature names
    features=  list(df.columns)[:-1]

    # calculate prior
    prior= calculate_prior(df, Y)

    Y_pred = []
    # loop over every data sample
    for x in X:
        # calculate likelihood
        labels= sorted(list(df[Y].unique()))
        likelihood= [1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):
                likelihood[j] *= calculate_likelihood_categorical(df, features[i], x[i], Y, 1

        # calculate posterior probability (numerator only)
        post_prob = [1]*len(labels)
        for j in range(len(labels)):
            post_prob[j] = likelihood[j] * prior[j]

        Y_pred.append(np.argmax(post_prob))

    return np.array(Y_pred)


from sklearn.model_selection import train_test_split
train, test= train_test_split(data, test_size=.2, random_state=41)

X_test  test.iloc[:,:-1].values
Y test  test.iloc[:,-1].values
Y_pred  = naive_bayes_categorical(train, X=X_test, Y="diagnosis")

from sklearn.metrics import confusion_matrix, f1 score
print(confusion_matrix(Y_test, Y_pred))
print(f1_score(Y_test, Y_pred))
```

```
    [[38  2]
     [ 5 69]]
     0.9517241379310345
```

EXPERIMENT -09

Aim : To perform perceptron learning of the neural network for the given dataset**.**

Description :

It is a machine learning algorithm that uses supervised learning of binary classifiers. In Perceptron, the weight coefficient is automatically learned. Initially, weights are multiplied with input features, and then the decision is made whether the neuron is fired or not. A neural network is formed when a collection of nodes or neurons are interlinked through synaptic connections. There are three layers in every artificial neural network – input layer, hidden layer, and output layer. The input layer that is formed from a collection of several nodes or neurons receives inputs. Every neuron in the network has a function, and every connection has a weight value associated with it. Inputs then move from the input layer to layer made from a separate set of neurons – the hidden layer. The output layer gives the final outputs.

The concept of perceptron has a critical role in machine learning. It is used as an algorithm or a linear classifier to facilitate supervised learning of binary classifiers. Supervised learning is amongst the most researched of learning problems. A supervised learning sample always consists of an input and a correct/explicit output. The objective of this learning problem is to use data with correct labels for making predictions on future data, for training a model. Some of the common problems of supervised learning include classification to predict class labels.

Dataset Description :

The seeds dataset involves the prediction of species given measurements seeds from different varieties of wheat.

There are 201 records and 7 numerical input variables. It is a classification problem with 3 output classes. The scale for each numeric input value vary, so some data normalization may be required for use with algorithms that weight inputs like the backpropagation algorithm. Using the Zero Rule algorithm that predicts the most common class value, the baseline accuracy for the problem is 28.095%.

Code :

```
from·math·import·exp
from·random·import·seed
from·random·import·random
```

A function named initialize_network() that creates a new neural network ready for training. It accepts three parameters, the number of inputs, the number of neurons to have in the hidden layer and the number of outputs.

```
# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in range(n_hidd
    network.append(hidden_layer)
    output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range(n_outp
    network.append(output_layer)
    return network
seed(1)
network = initialize_network(2, 1, 2)
for layer in network:
    print(layer)
```

```
[{'weights': [0.13436424411240122, 0.8474337369372327, 0.763774618976614]}]
[{'weights': [0.2550690257394217, 0.49543508709194095]}, {'weights': [0.449491064788738
```

In this function named activate(). You can see that the function assumes that the bias is the last weight in the list of weights. Neuron activation is calculated as the weighted sum of the inputs.

```
#·Calculate·neuron·activation·for·an·input
def·activate(weights,·inputs):
→ activation·=·weights[-1]
→ for·i·in·range(len(weights)-1):
→ → activation·+=·weights[i]·*·inputs[i]
→ return·activation
```

A function named transfer() that implements the sigmoid equation. The sigmoid activation function looks like an S shape, it's also called the logistic function. It can take any input value and produce a number between 0 and 1 on an S-curve.

```
# Transfer neuron activation
def transfer(activation):
  return 1.0 / (1.0 + exp(-activation))
```

A function named forward_propagate() that implements the forward propagation for a row of data from our dataset with our neural network.The function returns the outputs from the last layer also called the output layer.

```
# Forward propagate input to a network output
def forward_propagate(network, row):
  inputs = row
  for layer in network:
    new_inputs = []
    for neuron in layer:
      activation = activate(neuron['weights'], inputs)
      neuron['output'] = transfer(activation)
      new_inputs.append(neuron['output'])
    inputs = new_inputs
  return inputs


# Calculate the derivative of an neuron output
def transfer_derivative(output):
    return output * (1.0 - output)
```

Testing forward propagation

```
network = [[{'weights': [0.13436424411240122, 0.8474337369372327, 0.763774618976614]}],
        [{'weights': [0.2550690257394217, 0.49543508709194095]}, {'weights': [0.4494910647887
row = [1, 0, None]
output = forward_propagate(network, row)
print(output)

     [0.6629970129852887, 0.7253160725279748]
```

EXPERIMENT -10

Aim : To perform backpropogation of the neural network for the given dataset**.**

Description :

The Backpropagation algorithm is a supervised learning method for multilayer feed-forward networks from the field of Artificial Neural Networks.
Feed-forward neural networks are inspired by the information processing of one or more neural cells, called a neuron. A neuron accepts input signals via its dendrites, which pass the electrical signal down to the cell body. The axon carries the signal out to synapses, which are the connections of a cell's axon to other cell's dendrites.
The principle of the backpropagation approach is to model a given function by modifying internal weightings of input signals to produce an expected output signal. The system is trained using a supervised learning method, where the error between the system's output and a known expected output is presented to the system and used to modify its internal state.
Technically, the backpropagation algorithm is a method for training the weights in a multilayer feed-forward neural network. As such, it requires a network structure to be defined of one or more layers where one layer is fully connected to the next layer. A standard network structure is one input layer, one hidden layer, and one output layer.
Backpropagation can be used for both classification and regression problems.

Dataset Description :

The seeds dataset involves the prediction of species given measurements seeds from different varieties of wheat.

There are 201 records and 7 numerical input variables. It is a classification problem with 3 output classes. The scale for each numeric input value vary, so some data normalization may be required for use with algorithms that weight inputs like the backpropagation algorithm. Using the Zero Rule algorithm that predicts the most common class value, the baseline accuracy for the problem is 28.095%.

Code :

```
from math import exp
from random import seed
from random import random
```

Initialize a network

```
def initialize_network(n_inputs, n_hidden, n_outputs):
  network = list()
  hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in range(n_hidden
  network.append(hidden_layer)
  output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range(n_output
  network.append(output_layer)
  return network
```

Calculate neuron activation for an input

```
def activate(weights, inputs):
  activation = weights[-1]
  for i in range(len(weights)-1):
    activation += weights[i] * inputs[i]
  return activation
```

Transfer neuron activation

```
def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))
```

Forward propagate input to a network output

```
def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = transfer(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs
```

Calculate the derivative of an neuron output

```
def transfer_derivative(output):
    return output * (1.0 - output)
```

Backpropagate error and store in neurons

```
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] * neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(neuron['output'] - expected[j])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])
```

Update network weights with error

```
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] -= l_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] -= l_rate * neuron['delta']
```

Train a network for a fixed number of epochs

```
def train_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            outputs = forward_propagate(network, row)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]] = 1
            sum_error += sum([(expected[i]-outputs[i])**2 for i in range(len(expected))])
            backward_propagate_error(network, expected)
            update_weights(network, row, l_rate)
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))


# Test training backprop algorithm
seed(1)
dataset = [[2.7810836,2.550537003,0],
    [1.465489372,2.362125076,0],
    [3.396561688,4.400293529,0],
    [1.38807019,1.850220317,0],
    [3.06407232,3.005305973,0],
    [7.627531214,2.759262235,1],
    [5.332441248,2.088626775,1],
    [6.922596716,1.77106367,1],
    [8.675418651,-0.242068655,1],
    [7.673756466,3.508563011,1]]
n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))
network = initialize_network(n_inputs, 2, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
    print(layer)
```

```
>epoch=0, lrate=0.500, error=6.350
>epoch=1, lrate=0.500, error=5.531
>epoch=2, lrate=0.500, error=5.221
>epoch=3, lrate=0.500, error=4.951
>epoch=4, lrate=0.500, error=4.519
>epoch=5, lrate=0.500, error=4.173
>epoch=6, lrate=0.500, error=3.835
>epoch=7, lrate=0.500, error=3.506
>epoch=8, lrate=0.500, error=3.192
>epoch=9, lrate=0.500, error=2.898
>epoch=10, lrate=0.500, error=2.626
>epoch=11, lrate=0.500, error=2.377
>epoch=12, lrate=0.500, error=2.153
>epoch=13, lrate=0.500, error=1.953
>epoch=14, lrate=0.500, error=1.774
>epoch=15, lrate=0.500, error=1.614
>epoch=16, lrate=0.500, error=1.472
>epoch=17, lrate=0.500, error=1.346
>epoch=18, lrate=0.500, error=1.233
```

CASE STUDY :
RANDOM FORESTS

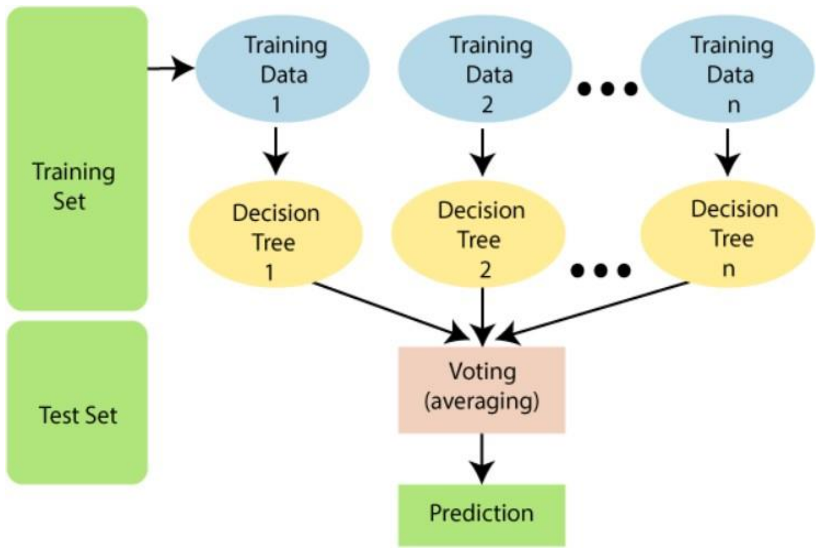
AIM:

The main objective is to build a model which predicts whether a person is having diabetes or not based on some parameters like glucose levels, insulin levels, etc using "Random Forest"

CONTENTS:

Random Forest

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique.It 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 forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.



Dataset Description:

The dataset is gathered from Kaggle, which is named the Pima Indian Diabetes Dataset(PIDD). The dataset has many attributes of 768 patients.

The 9th attribute is the class variable of each data point. This class variable shows the outcome 0 and 1 for diabetics which indicates positive or negative for diabetics.

Distribution of Diabetic patient records- We made a model to predict diabetes however the dataset was slightly imbalanced having around 500 classes labeled as 0 means negative means no diabetes and 268 labeled as 1 means positive means diabetic.

PROCEDURE:

Flowchart

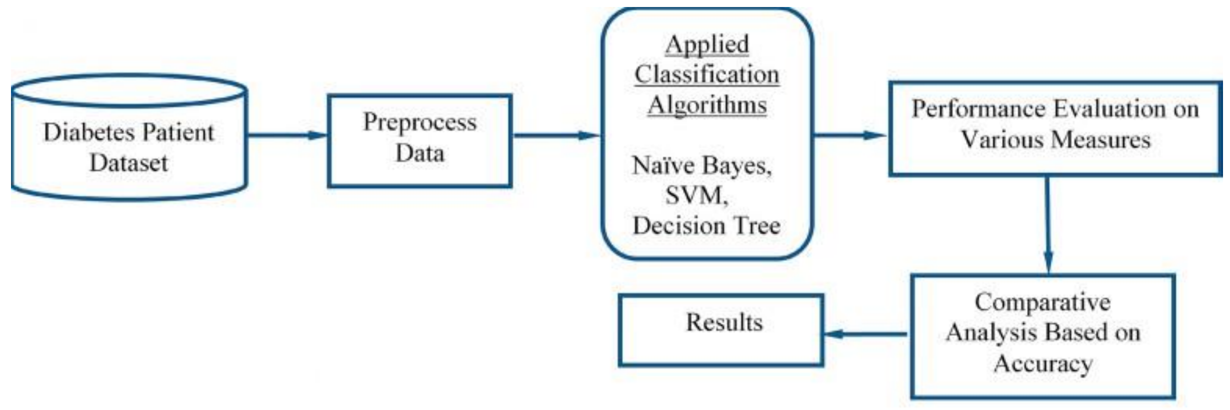
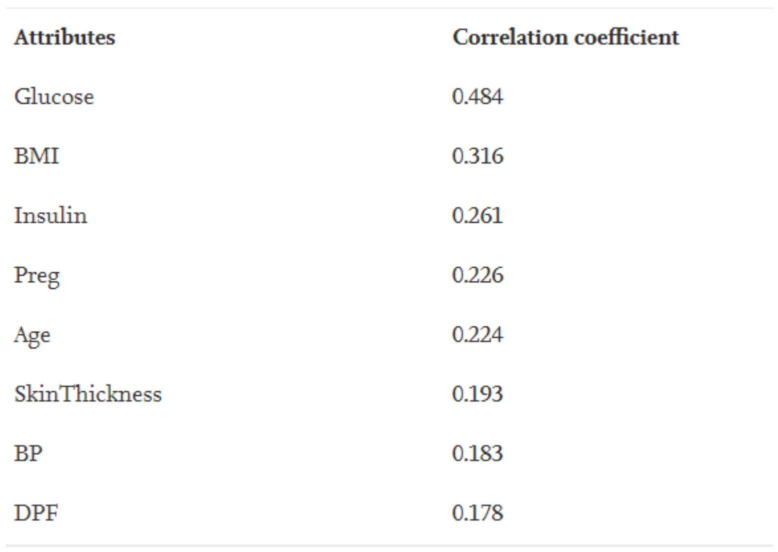

Diabetes Patient Dataset
Preprocess Data
Applied Classification Algorithms
Naïve Bayes, SVM, Decision Tree
Performance Evaluation on Various Measures
Comparative Analysis Based on Accuracy
Results

Fig. 1. Proposed Model Diagram

Feature selection

Pearson's correlation method is a popular method to find the most relevant attributes/features. The correlation coefficient is calculated in this method, which correlates with the output and input attributes. The coefficient value remains in the range between −1 and 1. The value above 0.5 and below −0.5 indicates a notable correlation, and the zero value means no correlation. In Weka, the correlation filter is used to find the correlation coefficient, and the results are shown in Table 3. We used 0.2 as a cut-off for relevant attributes. Hence SkinThickness, BP, and DPF features are removed. Glucose, BMI, Insulin, Preg, and Age are our most relevant five input attributes.

Table 3. The correlation between input and output attributes.



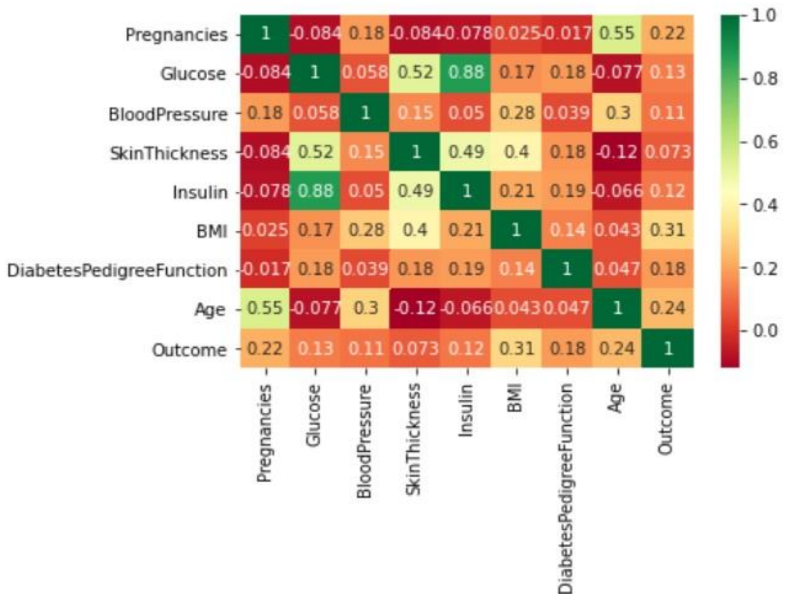
Attributes	Correlation coefficient
Glucose	0.484
BMI	0.316
Insulin	0.261
Preg	0.226
Age	0.224
SkinThickness	0.193
BP	0.183
DPF	0.178

Pearson Correlation is the default method of the function “corr”.

We can also create a heatmap using Seaborn to visualize the correlation between the different columns of our data:

```
sns.heatmap(diabetes_dataset.corr(),annot=True,cmap="RdYlGn")
```

```
<AxesSubplot:>
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1	-0.084	0.18	-0.084	-0.078	0.025	-0.017	0.55	0.22
Glucose	-0.084	1	0.058	0.52	0.88	0.17	0.18	-0.077	0.13
BloodPressure	0.18	0.058	1	0.15	0.05	0.28	0.039	0.3	0.11
SkinThickness	-0.084	0.52	0.15	1	0.49	0.4	0.18	-0.12	0.073
Insulin	-0.078	0.88	0.05	0.49	1	0.21	0.19	-0.066	0.12
BMI	-0.025	0.17	0.28	0.4	0.21	1	0.14	0.043	0.31
DiabetesPedigreeFunction	-0.017	0.18	0.039	0.18	0.19	0.14	1	0.047	0.18
Age	0.55	-0.077	0.3	-0.12	-0.066	0.043	0.047	1	0.24
Outcome	0.22	0.13	0.11	0.073	0.12	0.31	0.18	0.24	1

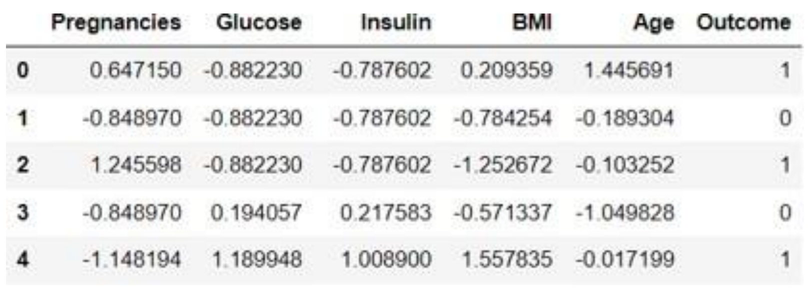
Normalization

Data Normalization or feature scaling is a process to standardize the range of features of the data as the range may vary a lot. So we can preprocess the data using ML algorithms. So for this, we will use StandardScaler for the numerical values, which uses the formula as x-mean/std deviation.

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

```
StandardScaler()
```

```
diabetes_dataset["Insulin"] = stdScale.fit_transform(diabetes_dataset[["Insulin"]])
diabetes_dataset["Glucose"] = stdScale.fit_transform(diabetes_dataset[["Glucose"]])
diabetes_dataset["Pregnancies"] = stdScale.fit_transform(diabetes_dataset[["Pregnancies"]])
diabetes_dataset["BMI"] = stdScale.fit_transform(diabetes_dataset[["BMI"]])
diabetes_dataset["Age"] = stdScale.fit_transform(diabetes_dataset[["Age"]])
```



	Pregnancies	Glucose	Insulin	BMI	Age	Outcome
0	0.647150	-0.882230	-0.787602	0.209359	1.445691	1
1	-0.848970	-0.882230	-0.787602	-0.784254	-0.189304	0
2	1.245598	-0.882230	-0.787602	-1.252672	-0.103252	1
3	-0.848970	0.194057	0.217583	-0.571337	-1.049828	0
4	-1.148194	1.189948	1.008900	1.557835	-0.017199	1

CODE SNIPPETS:

Deployment of Model (deployment.py):

```
# Importing essential libraries
import numpy as np
import pandas as pd
import pickle

# Loading the dataset
df = pd.read_csv('diabetes.csv')

# Renaming DiabetesPedigreeFunction as DPF
df = df.rename(columns={'DiabetesPedigreeFunction':'DPF'})

# Replacing the 0 values from ['Glucose','BloodPressure','SkinThickness','Insulin','BMI'] by NaN
df_copy = df.copy(deep=True)
df_copy[['Pregnancies','Glucose','Insulin','BMI','Age']] =
df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)
# Replacing NaN value by mean, median depending upon distribution
df_copy['Pregnancies'].fillna(df_copy['Pregnancies'].mean(), inplace=True)
df_copy['Glucose'].fillna(df_copy['Glucose'].mean(), inplace=True)
df_copy['Insulin'].fillna(df_copy['Insulin'].median(), inplace=True)
df_copy['BMI'].fillna(df_copy['BMI'].median(), inplace=True)
df_copy['Age'].fillna(df_copy['Age'].median(), inplace=True)

# Model Building
from sklearn.model_selection import train_test_split
X = df.drop(columns='Outcome')
y = df['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)

# Creating Random Forest Model
```

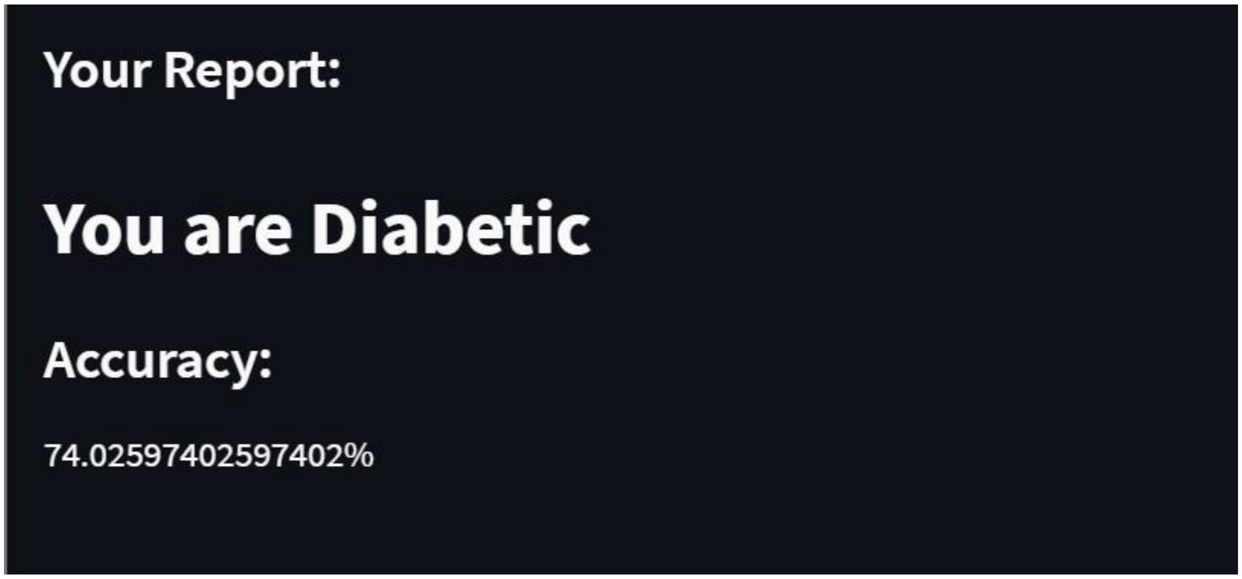
```
from   sklearn.ensemble   import   RandomForestClassifier
classifier    =    RandomForestClassifier(n_estimators=20)
classifier.fit(X_train, y_train)

# Creating a pickle file for the classifier filename =
'diabetes-prediction-rfc-model.pkl'
pickle.dump(classifier, open(filename, 'wb'))
```

III. User Interface (app.py):

```
# IMPORT STATEMENTS
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifierfrom
sklearn.model_selection import train_test_split import
seaborn as sns
import streamlit as st
import pickle

df = pd.read_csv(r'diabetes.csv')
print(df.shape)

# HEADINGS
st.title('Diabetes Checkup')
st.sidebar.header('Patient Data')
st.subheader('Training Data Statistics')
st.write(df.describe())

# X AND Y DATA
x = df.drop(['Outcome'], axis = 1)y =
df.iloc[:, -1]
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2, random_state = 0)

# FUNCTION
def user_report():
  pregnancies = st.sidebar.slider('Pregnancies', 0,17, 3 )
  glucose = st.sidebar.slider('Glucose', 0,200, 120 )
  insulin = st.sidebar.slider('Insulin', 0,846, 79 )
  bmi = st.sidebar.slider('BMI', 0,67, 20 )
  age = st.sidebar.slider('Age', 21,88, 33 )

  user_report_data = {
      'pregnancies':pregnancies,
      'glucose':glucose,
      'insulin':insulin,
      'bmi':bmi,
      'age':age
  }
  data = np.array([[glucose, bp, insulin, bmi]])
  report_data = pd.DataFrame(user_report_data, index=[0])
  temp=[data, report_data]
  return temp


# PATIENT DATA
samp = user_report()
forpkl=samp[0]
user_data=samp[1]
st.subheader('Patient Data')
st.write(user_data)

# MODEL
filename = 'diabetes-prediction-rfc-model.pkl'
rf=pickle.load(open(filename, 'rb'))
user_res = rf.predict(forpkl)
# OUTPUT
st.subheader('Your Report: ')
output=' '

if user_res[0]==0:
  output = 'You are not Diabetic'
else:
  output = 'You are Diabetic'

st.title(output)
st.subheader('Accuracy: ')
st.write(str(accuracy_score(y_test, rf.predict(x_test))*100)+'%')
```

CONCLUSION:

The main aim of this project was to design and implement Diabetes Prediction Using Machine Learning Methods and Performance Analysis of those methods and it has been achieved successfully. The proposed approach uses various classification and ensemble learning methods in which SVM, Knn, Random Forest, Decision Tree, Logistic Regression and Gradient Boosting classifiers are used. And 74% classification accuracy has been achieved. The Experimental results can assist health care to take early prediction and make early decision to cure diabetes and save humans life.



Your Report:

You are Diabetic

Accuracy:

74.02597402597402%