LAKIREDDY BALI REDDYCOLLEGE OF ENGINEERING (AUTONOMOUS)

Department of
COMPUTER SCIENCE AND ENGINEERING

20AD54 – MACHINE LEARNING LAB

Name of the Student: **BATHULA MALYADRI**

Registered Number: **21761A0573**

Branch & Section : **CSE & B-Sec**

Academic Year: **2023-2024**

Index

Module 1

Aim:

Basic statistical functions for data exploration **No**

of Attributes:

1. Alcohol: the amount of alcohol in wine
2. Volatile acidity: are high acetic acid in wine which leads to an unpleasant vinegar taste
3. Sulphates: a wine additive that contributes to SO2 levels and acts as an antimicrobial and antioxidant
4. Citric Acid: acts as a preservative to increase acidity (small quantities add freshness and flavour to wines)
5. Total Sulphur Dioxide: is the amount of free + bound forms of SO2
6. Density: sweeter wines have a higher density
7. Chlorides: the amount of salt in the wine
8. Fixed acidity: are non-volatile acids that do not evaporate readily
9. pH: the level of acidity
10. Free Sulphur Dioxide: it prevents microbial growth and the oxidation of wine
11. Residual sugar: is the amount of sugar remaining after fermentation stops. The key is to have a perfect balance between — sweetness and sourness (wines > 45g/ltrs are sweet)

Program:

#load the dataset:

```
import pandas as pd
df=pd.read_csv('wr.csv') print(df)
```

output:

```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.076
1               7.8             0.880         0.00             2.6      0.098
2               7.8             0.760         0.04             2.3      0.092
3              11.2             0.280         0.56             1.9      0.075
4               7.4             0.700         0.00             1.9      0.076
...             ...               ...          ...             ...        ...
1594            6.2             0.600         0.08             2.0      0.090
1595            5.9             0.550         0.10             2.2      0.062
1596            6.3             0.510         0.13             2.3      0.076
1597            5.9             0.645         0.12             2.0      0.075
1598            6.0             0.310         0.47             3.6      0.067

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
...                   ...                   ...      ...   ...        ...
1594                 32.0                  44.0  0.99490  3.45       0.58
1595                 39.0                  51.0  0.99512  3.52       0.76
1596                 29.0                  40.0  0.99574  3.42       0.75
1597                 32.0                  44.0  0.99547  3.57       0.71
1598                 18.0                  42.0  0.99549  3.39       0.66

      alcohol  quality
0         9.4        5
1         9.8        5
2         9.8        5
3         9.8        6
4         9.4        5
...       ...      ...
1594     10.5        5
1595     11.2        6
1596     11.0        6
1597     10.2        5
1598     11.0        6
```

Info and description of dataset:

```
df.describe()
```

Output:

	fixed acidity;"volatile acidity";"citric acid";"residual sugar";"chlorides";"free sulfur dioxide";"total sulfur dioxide";"density";"pH";"sulphates";"alcohol";"quality"
count	1599
unique	1359
top	7.2;0.36;0.46;2.1;0.074;24;44;0.99534;3.4;0.85...
freq	4

Retrieving the data types of attributes:

```
df.dtypes
```

Output:

```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

Maximum and Minimum values of attributes:

```
df.min()
```

Output:

```
fixed acidity           4.60000
volatile acidity        0.12000
citric acid             0.00000
residual sugar          0.90000
chlorides               0.01200
free sulfur dioxide     1.00000
total sulfur dioxide    6.00000
density                 0.99007
pH                      2.74000
sulphates               0.33000
alcohol                 8.40000
quality                 3.00000
dtype: float64
```

```
df.max()
```

Output:

```
fixed acidity           15.90000
volatile acidity         1.58000
citric acid              1.00000
residual sugar          15.50000
chlorides                0.61100
free sulfur dioxide     72.00000
total sulfur dioxide   289.00000
density                  1.00369
pH                       4.01000
sulphates                2.00000
alcohol                 14.90000
quality                  8.00000
dtype: float64
```

Sum and mean,std,var of a particular attribute:

Standard deviation:

```
df['chlorides'].std()
```

Output:

```
0.0470653020100901
```

Mean:

```
df['chlorides'].mean()
```

Output:

```
0.08746654158849279
```

```
df['chlorides'].var()
```

Output:

```
0.0022151426533009912
```

TO check the values are null or not:

```
df.isnull()
```

Output:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	False	False	False	False	False	False	False	False	False	False	False	False
1595	False	False	False	False	False	False	False	False	False	False	False	False
1596	False	False	False	False	False	False	False	False	False	False	False	False
1597	False	False	False	False	False	False	False	False	False	False	False	False
1598	False	False	False	False	False	False	False	False	False	False	False	False

1599 rows × 12 columns

Converting dataset values into an array:

```
df.to_numpy()
```

Output:

```
array([[ 7.4  ,  0.7  ,  0.   , ...,  0.56 ,  9.4  ,  5.   ],
       [ 7.8  ,  0.88 ,  0.   , ...,  0.68 ,  9.8  ,  5.   ],
       [ 7.8  ,  0.76 ,  0.04 , ...,  0.65 ,  9.8  ,  5.   ],
       ...,
       [ 6.3  ,  0.51 ,  0.13 , ...,  0.75 , 11.   ,  6.   ],
       [ 5.9  ,  0.645,  0.12 , ...,  0.71 , 10.2  ,  5.   ],
       [ 6.   ,  0.31 ,  0.47 , ...,  0.66 , 11.   ,  6.   ]])
```

Filtering the data:

```
df[df.chlorides>1]
```

OUTPUT:

fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality

Module 2

Aim:

Data Visualization: Box plot, scatter plot, histogram **Description**:

A Box Plot is also known as Whisker plot is created to display the summary of the set of data values having properties like minimum, first quartile, median, third quartile and maximum. In the box plot, a box is created from the first quartile to the third quartile, a vertical line is also there which goes through the box at the median

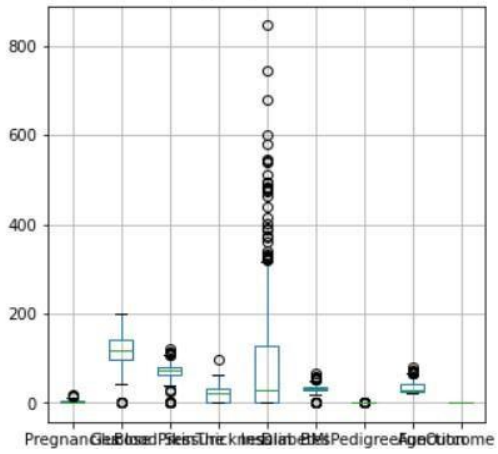
Program:

#Boxplot import pandas as pd
import numpy as np import
matplotlib.pyplot as plt import
seaborn as sns
df=pd.read_csv("/content/drive/MyDrive/diabetes.csv")
#creation of boxplot with dataset
df.boxplot(figsize = (5,5))

Output:

```
#creation of boxplot with dataset
df.boxplot(figsize = (5,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7c33da56d0>
```



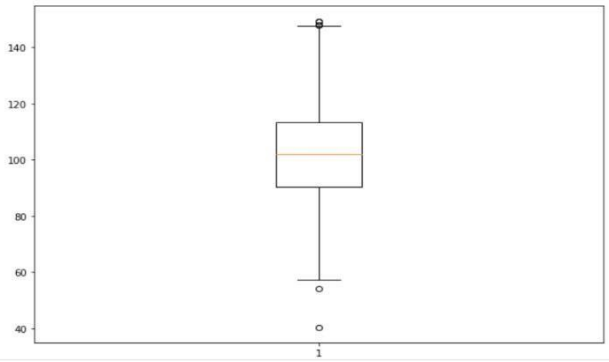
#boxplot without dataset:

np.random.seed(10)

data = np.random.normal(100, 20, 200)
fig = plt.figure(figsize =(10, 7))
plt.boxplot(data) plt.show()

output:

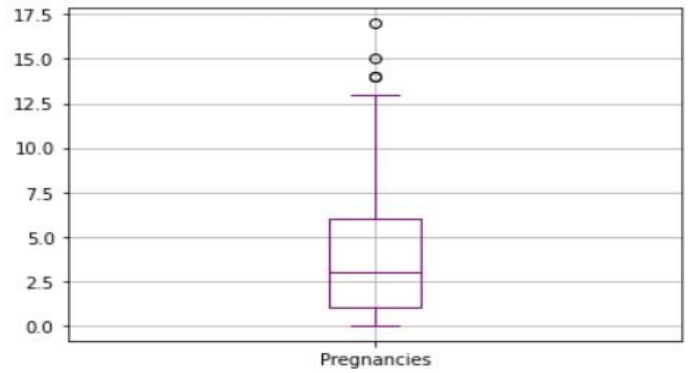
```
#boxplot without dataset:
np.random.seed(10)
data = np.random.normal(100, 20, 200)
fig = plt.figure(figsize =(10, 7))
plt.boxplot(data)
plt.show()
```



Boxplot with particular attribute:

```
#boxplot for particular attribute
df.boxplot(column="Pregnancies",color="purple")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7c3412ca50>
```



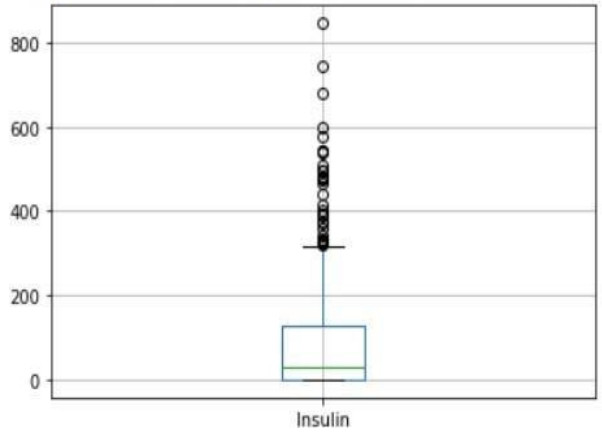
#creating boxplot with Quantiles

Q1=df['Insulin'].quantile(0.25)
Q2=df['Insulin'].quantile(0.50)
Q3=df['Insulin'].quantile(0.75)
IQR=Q3-Q1
LowestQuartile=Q1-(1.5*IQR)
HighestQuartile=Q3+(1.5*IQR) print("first Quantile is :",Q1) print("second Quantile is :",Q2) print("third Quantile is :",Q3)

```
print("IQR is:",IQR) print("LowestQuartile
is:",LowestQuartile) print("HighestQuartile
is:",HighestQuartile)
df.boxplot(column="Insulin")
```

output:

```
first Quantile is : 0.0
second Quantile is : 30.5
third Quantile is : 127.25
IQR is: 127.25
LowestQuartile is: -190.875
HighestQuartile is: 318.125
<matplotlib.axes._subplots.AxesSubplot at 0x7f7c3434c2d0>
```



Model performance:

```
TP=int(input("enter True Positive Value"))
TN=int(input("enter True Negative Value"))
FP=int(input("enter False Positive Value"))
FN=int(input("enter False Positive Value"))
acc=(TP+TN)/(TP+TN+FP+FN)
err=(FP+FN)/(TP+TN+FP+FN)
sen=(TP)/(TP+FN) spes=(TN)/(TN+FP)
prec=(TP)/(TP+FP)
f1=(2*prec*sen)/(prec+sen)
print("Accuracy:",acc)
print("Errorrate:",err)
print("Sensitivity:",sen)
print("Specificity:",spes)
print("Precision",prec) print("f1-
measure:",f1)
```

Output:

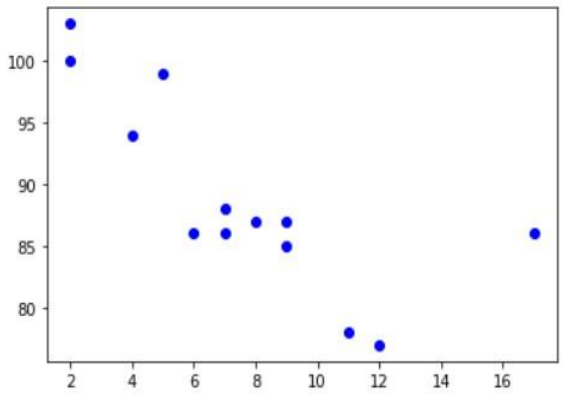
```
enter True Positive Value85
enter True Negative Value9
enter False Positive Value4
enter False Positive Value2
Accuracy: 0.94
Errorrate: 0.06
Sensitivity: 0.9770114942528736
Specificity: 0.6923076923076923
Precision 0.9550561797752809
f1-measure: 0.9659090909090908
```

Program:

#scatterplot without dataset:

```
import matplotlib.pyplot as plt x
=[5, 7, 8, 7, 2, 17, 2, 9,
   4, 11, 12, 9, 6] y =[99,
86, 87, 88, 100, 86,
   103, 87, 94, 78, 77, 85, 86] plt.scatter(x,
y, c ="blue")
plt.show()
```

Output:

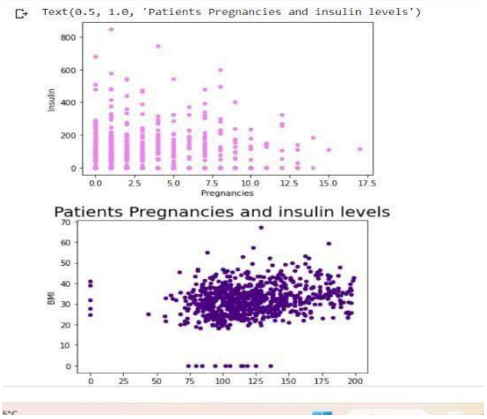


Program:

#Scatter plot with datset:

```
import pandas as pd import
numpy as np import
matplotlib.pyplot as plt import
seaborn as sns
df=pd.read_csv("/content/drive/MyDrive/diabetes.csv")
df.plot.scatter(x="Pregnancies",y="Insulin",s=20,color="violet")
df.plot.scatter(x="Glucose",y="BMI",s=20,color="indigo")
plt.title('Patients Pregnancies and insulin levels', fontsize = 20)
```

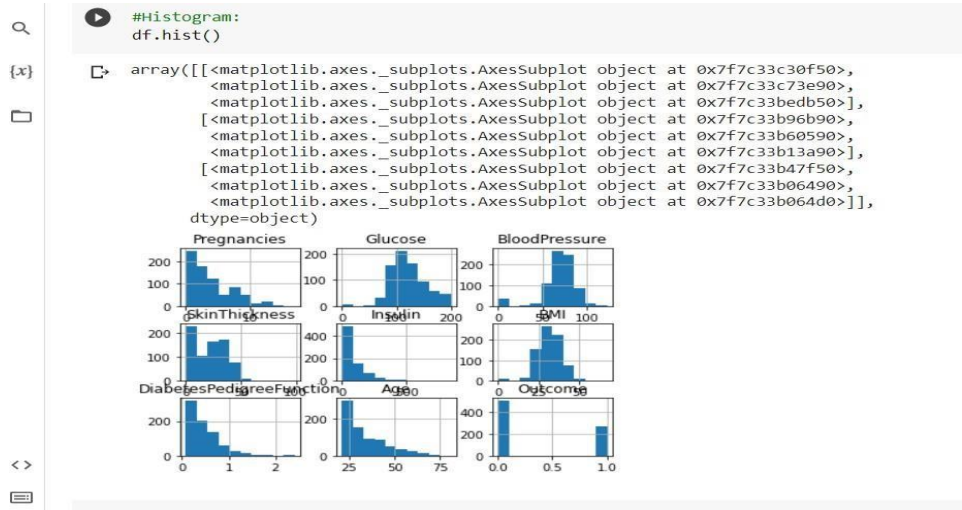
Output:



Program & output:

Histogram:

import pandas as pd import
numpy as np import
matplotlib.pyplot as plt import
seaborn as sns
df=pd.read_csv("/content/drive/MyDrive/diabetes.csv") df.hist()



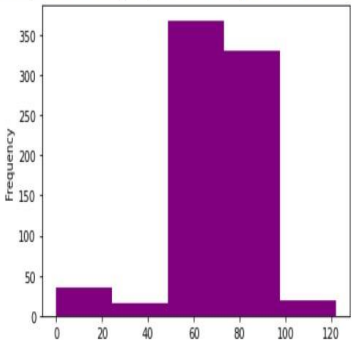
Histogram with particular attribute:

df['BloodPressure'].plot(kind='hist',color="purple", bins=5) df.hist("BMI",color="green")

output:

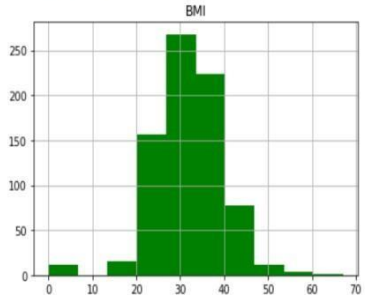
```
df['BloodPressure'].plot(kind='hist',color="purple",bins=5)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7c30e23dd0>
```




```
df.hist("BMI",color="green")
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f7c335f6b90>]],
       dtype=object)
```



Module 3

Aim :

Data Preprocessing: Handling missing values, outliers, normalization, Scaling

Description:

Data preprocessing is essential before its actual use. Data preprocessing is the concept of changing the raw data into a clean data set. The dataset is preprocessed in order to check missing values, noisy data, and other inconsistencies before executing it to the algorithm. Data must be in a format appropriate for ML. For example, if the algorithm processes only numeric data then if a class is labelled with "*malignant*" or "*benign*" then it must be replaced by "0" or "1." Data transformation and feature extraction are used to expand the performance of classifiers and hence a classification algorithm will be able to create a meaningful diagnosis. Only relevant features are selected and extracted for the particular disease. For example, a cancer patient may have diabetes, so it is essential to separate related features of cancer from diabetes. An unsupervised learning algorithm such as PCA is a familiar algorithm for feature extraction. Supervised learning is appropriate for classification and predictive modeling.

Program :

```
import pandas as pd import
numpy as np import
matplotlib.pyplot as plt import
seaborn as sns
df=pd.read_csv("/content/drive/MyDrive/diabetes.csv")
```

Finding null values and deleting the columns with missing data

```python
#to find null values
print(df.isnull().sum())
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```python
[5] #deleting the columns with missing data
updated_df = df.dropna(axis=1)
updated_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
```

Deleting the row with missing data & filling the missing values

```
#deleting the row with missing data
updated_df = df.dropna(axis=0)
updated_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 60.0 KB
```

```
[7] # Filling the Missing Values - Imputation
updated_df = df
updated_df['BMI']=updated_df['BMI'].fillna(updated_df['BMI'].mean())
updated_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
```

#NORMALISATION

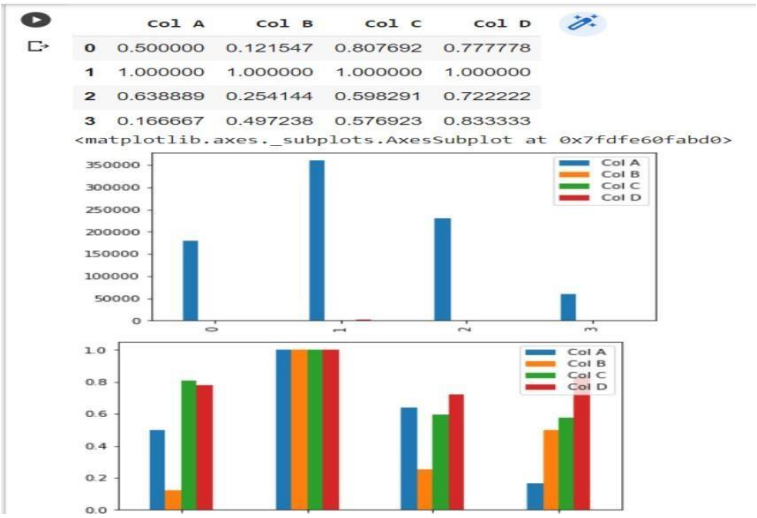
```
# importing packages import
pandas as pd import
matplotlib.pyplot as plt
# create data df =
pd.DataFrame([
            [180000, 110, 18.9, 1400],
            [360000, 905, 23.4, 1800],
            [230000, 230, 14.0, 1300],
            [60000, 450, 13.5, 1500]],

            columns=['Col A', 'Col B',
                  'Col C', 'Col D'])
 # view data display(df)
```

Output :

	Col A	Col B	Col C	Col D
0	180000	110	18.9	1400
1	360000	905	23.4	1800
2	230000	230	14.0	1300
3	60000	450	13.5	1500

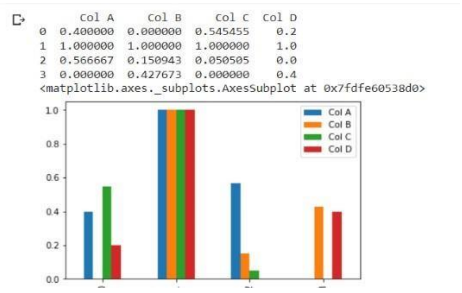
df.plot(kind = 'bar') #
copy the data
df_max_scaled = df.copy() # apply normalization techniques for column in
df_max_scaled.columns: df_max_scaled[column] = df_max_scaled[column] /
df_max_scaled[column].abs().max()
# view normalized data display(df_max_scaled)
df_max_scaled.plot(kind = 'bar')

Output :



	Col A	Col B	Col C	Col D
0	0.500000	0.121547	0.807692	0.777778
1	1.000000	1.000000	1.000000	1.000000
2	0.638889	0.254144	0.598291	0.722222
3	0.166667	0.497238	0.576923	0.833333

copy the data df_min_max_scaled = df.copy() # apply
normalization techniques for column in
df_min_max_scaled.columns:
df_min_max_scaled[column]=(df_min_max_scaled[column]-
df_min_max_scaled[column].min())/(df_min_max_scaled[column].max()- df_min_max_scaled[column].min())
view normalized data
print(df_min_max_scaled) import
matplotlib.pyplot as plt
df_min_max_scaled.plot(kind = 'bar')
Output:



	Col A	Col B	Col C	Col D
0	0.400000	0.000000	0.545455	0.2
1	1.000000	1.000000	1.000000	1.0
2	0.566667	0.150943	0.050505	0.0
3	0.000000	0.427673	0.000000	0.4

Scaling :

```
#scaling
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
X = df[['Glucose', 'Insulin']]
scaledX = scale.fit_transform(X)
print(scaledX)
```

```
[[ 0.84832379 -0.69289057]
 [-1.12339636 -0.69289057]
 [ 1.94372388 -0.69289057]
 ...
 [ 0.00330087  0.27959377]
 [ 0.1597866  -0.69289057]
 [-0.8730192  -0.69289057]]
```

Module 4

Aim:

Principal Component Analysis (PCA) **Description:**

A social network dataset is a dataset containing the structural information of a social network. In the general case, a social network dataset consists of persons connected by edges. Social network datasets can represent friendship relationships or may be extracted from a social networking Web site

Attributes are:
User ID
Gender
Age
Estimated Salary
Purchased

Principal Component Analysis (or PCA) uses linear algebra to transform the dataset into a compressed form.

Generally this is called a data reduction technique. A property of PCA is that you can choose the number of dimensions or principal component in the transformed result.

Program:

#Importing of the dataset and slicing it into independent and dependent variables

```
import numpy as np import matplotlib.pyplot as plt
import pandas as pd import
sklearn
dataset = pd.read_csv('/content/drive/MyDrive/Social_Network_Ads.csv') dataset
```

Output:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

Program:

Read and explore data:

```
X = dataset.iloc[:, [2, 3]].values y
= dataset.iloc[:, 4].values
```

#Encoding of the data using label encoder from

```
sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:,0] = le.fit_transform(X[:,0])
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

#feature Slicing to the training and test set of independent variables for reducing the size to smaller values

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_train
```

Output:

```
array([[ 0.58164944, -0.88670699],
       [-0.60673761,  1.46173768],
       [-0.01254409, -0.5677824 ],
       [-0.60673761,  1.89663484],
       [ 1.37390747, -1.40858358],
       [ 1.47293972,  0.99784738],
       [ 0.08648817, -0.79972756],
       [-0.01254409, -0.24885782],
       [-0.21060859, -0.5677824 ],
       [-0.21060859, -0.19087153],
       [-0.30964085, -1.29261101],
       [-0.30964085, -0.5677824 ],
       [ 0.38358493,  0.09905991],
       [ 0.8787462 , -0.59677555],
       [ 2.06713324, -1.17663843],
       [ 1.07681071, -0.13288524],
       [ 0.68068169,  1.78066227],
       [-0.70576986,  0.56295021],
       [ 0.77971394,  0.35999821],
       [ 0.8787462 , -0.53878926],
       [-1.20093113, -1.58254245],
       [ 2.1661655 ,  0.93986109],
       [-0.01254409,  1.22979253],
       [ 0.18552042,  1.08482681],
```

```
- Text
       [ 0.08648817,  0.1570462 ],
       [-1.89415691, -1.29261101],
       [-0.11157634,  0.30201192],
       [-0.21060859, -0.27785096],
       [ 0.28455268, -0.50979612],
       [-0.21060859,  1.6067034 ],
       [ 0.97777845, -1.17663843],
       [-0.21060859,  1.63569655],
       [ 1.27487521,  1.8676417 ],
       [-1.10189888, -0.3648304 ],
       [-0.01254409,  0.04107362],
       [ 0.08648817, -0.24885782],
       [-1.59706014, -1.23462472],
       [-0.50770535, -0.27785096],
       [ 0.97777845,  0.12805305],
       [ 1.96810099, -1.3505973 ],
       [ 1.47293972,  0.07006676],
       [-0.60673761,  1.37475825],
       [ 1.57197197,  0.01208048],
       [-0.80480212,  0.30201192],
       [ 1.96810099,  0.73690908],
       [-1.20093113, -0.50979612],
       [ 0.68068169,  0.27301877],
       [-1.39899564, -0.42281668],
       [ 0.18552042,  0.1570462 ],
       [-0.50770535, -1.20563157],
       [ 0.58164944,  2.01260742],
       [-1.59706014, -1.49556302],
       [-0.50770535, -0.53878926],
       [ 0.48261718,  1.83864855],
       [-1.39899564, -1.089659  ],
       [ 0.77971394, -1.37959044],
       [-0.30964085, -0.42281668],
       [ 1.57197197,  0.99784738],
```

Program:

#prediction of y

y_pred = classifier.predict(X_test)
y_pred

Output:

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

#evaluation of model using confusion matrix and accuracy

from sklearn.metrics import confusion_matrix,accuracy_score cm
= confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test,y_pred) cm

```
[ ] #evaluation of model using confusion matrix and accuracy
    from sklearn.metrics import confusion_matrix,accuracy_score
    cm = confusion_matrix(y_test, y_pred)
    ac = accuracy_score(y_test,y_pred)
    cm

    array([[64,  4],
           [ 3, 29]])
```

Program:

Implementation of PCA

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score,precision_score,recall_score,precision_recall_curve,plot_precision_recall_curve,f1_score from sklearn import metrics from sklearn.decomposition import PCA pca = PCA()
X_train = pca.fit_transform(X_train)

X_test = pca.transform(X_test) from

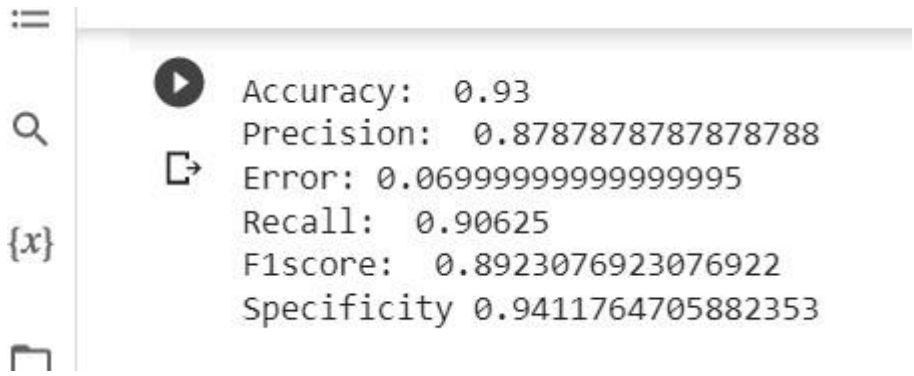
sklearn.decomposition import PCA pca

= PCA(n_components = 2)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

```
explained_variance = pca.explained_variance_ratio_ from
sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
knn_classifier.fit(X_train, y_train) y_pred_knn = knn_classifier.predict(X_test)
accuracy=accuracy_score(y_test, y_pred_knn) precision = precision_score(y_test,
y_pred_knn) recall = recall_score(y_test, y_pred_knn)
specificity = metrics.recall_score(y_test, y_pred_knn, pos_label=0)
f=f1_score(y_test,y_pred_knn) e=(1-accuracy) print('Accuracy:
',accuracy) print('Precision: ',precision) print('Error:',e)
print('Recall: ',recall) print('F1score: ',f)
print('Specificity',specificity)
```

Output:



Module 5

Aim :

Singular Value Decomposition (SVD)

Description:

The Singular Value Decomposition (SVD) of a matrix is a factorization of that matrix into three matrices. It has some interesting algebraic properties and conveys important geometrical and theoretical insights about linear transformations. It also has some important applications in data science.

Mathematics behind SVD

The SVD of mxn matrix A is given by the formula :

$A=UWV^T$

- U: *mxn* matrix of the orthonormal eigenvectors of .
- V^T: transpose of a *nxn* matrix containing the orthonormal eigenvectors of A^{T}A.

- W: a *nxn* diagonal matrix of the singular values which are the square roots of the eigenvalues of $A^T A$.

Program :

```
from numpy import array from
scipy.linalg import svd A =
array([[-4,-7], [1, 4]])
print(A)
U, s, V = svd(A) print("value
of U:")
print('_____')
print(U) print("value of
sigma(s):")
print(' ..................
        ') print(s)
print("value of v:")
print(' ..................          ')
print(V)
```

Output :

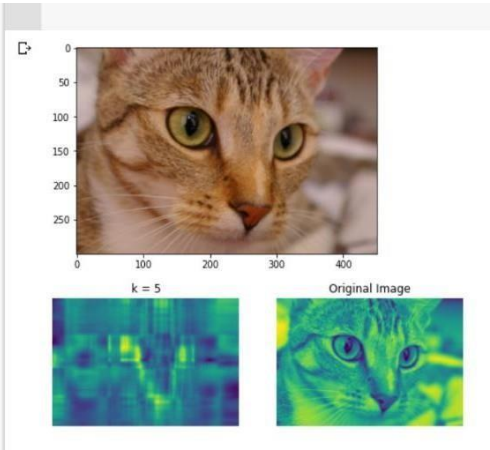
```
[[-4 -7]
 [ 1  4]]
value of U:
----------------
[[-0.89442719  0.4472136 ]
 [ 0.4472136   0.89442719]]
value of sigma(s):
-----------------
[9. 1.]
value of v:
-----------------
[[ 0.4472136   0.89442719]
 [-0.89442719  0.4472136 ]]
```

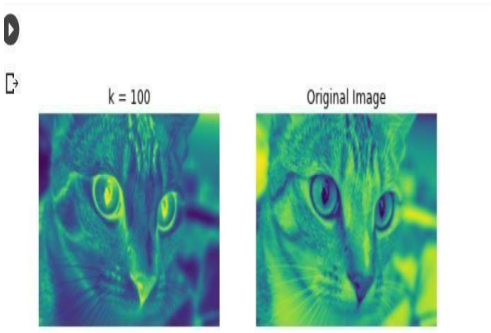
Program:
Singular Value Decomposition on Image:

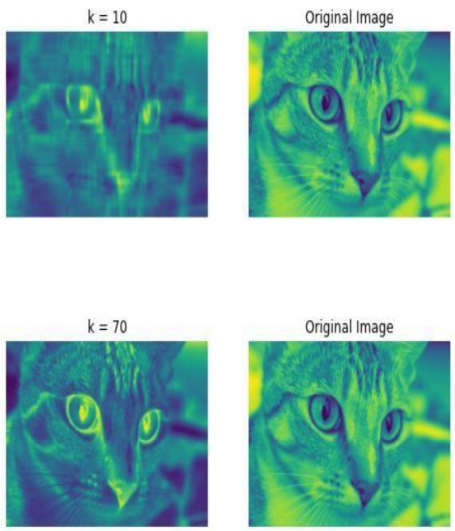
```
import numpy as np import
matplotlib.pyplot as plt from
skimage import data from
skimage.color import rgb2gray from
scipy.linalg import svd

cat = data.chelsea()
plt.imshow(cat) #
convert to grayscale
gray_cat = rgb2gray(cat)

# calculate the SVD and plot the image
```

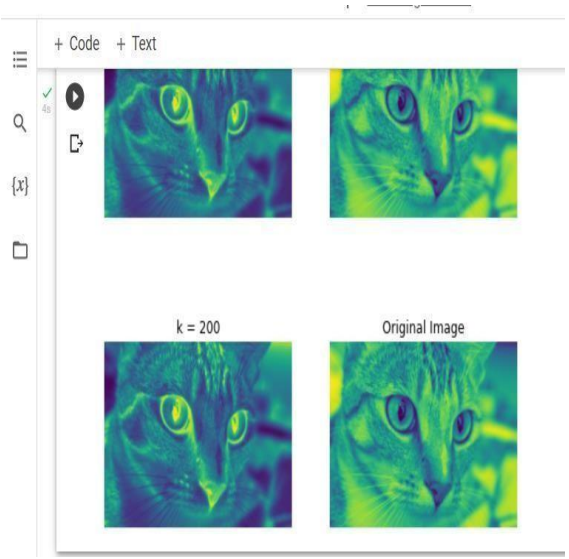
```
U,S,V_T = svd(gray_cat, full_matrices=False)
S = np.diag(S) fig, ax = plt.subplots(5, 2,
figsize=(8, 20))

curr_fig=0 for r in [5, 10, 70, 100, 200]:
cat_approx =U[:, :r] @ S[0:r, :r] @ V_T[:r, :]
ax[curr_fig][0].imshow(256-cat_approx)
ax[curr_fig][0].set_title("k = "+str(r))
ax[curr_fig,0].axis('off')
ax[curr_fig][1].set_title("Original Image")
ax[curr_fig][1].imshow(gray_cat)
ax[curr_fig,1].axis('off') curr_fig +=1
plt.show()
```

Output:






k = 10
Original Image
k = 70
Original Image


+ Code
+ Text
k = 200
Original Image

Module 6

Aim :

Linear Discriminant Analysis (LDA)

Description:

Linear Discriminant Analysis or Normal Discriminant Analysis or Discriminant Function Analysis is a dimensionality reduction technique that is commonly used for supervised classification problems. It is used for modelling differences in groups i.e. separating two or more classes. It is used to project the features in higher dimension space into a lower dimension space.

Program:

```
import numpy as np import
matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('/content/drive/MyDrive/Wine.csv') dataset
```

Output:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Phenols	Proanthocyanins	Color_Intensity	Hue	OD280	Proline	Customer_Segment
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065	1
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050	1
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185	1
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480	1
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740	3
174	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750	3
175	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835	3
176	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840	3
177	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560	3

178 rows × 14 columns

```
X = dataset.iloc[:, 0:13].values y
= dataset.iloc[:, 13].values
```

```
[ ] x = dataset.iloc[:, 0:13].values
    y = dataset.iloc[:, 13].values
    x

array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,
        1.065e+03],
       [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
        1.050e+03],
       [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
        1.185e+03],
       ...,
       [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,
        8.350e+02],
       [1.317e+01, 2.590e+00, 2.370e+00, ..., 6.000e-01, 1.620e+00,
        8.400e+02],
       [1.413e+01, 4.100e+00, 2.740e+00, ..., 6.100e-01, 1.600e+00,
        5.600e+02]])

[ ] y

array([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, 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, 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, 3, 3,
```

Splitting of data:

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0) from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

```
X_test

array([[915691, 'M', 13.4, ..., 0.2116, 0.07325, 0.3906],
       [9112367, 'B', 13.21, ..., 0.1619, 0.05584, 0.2084],
       [909410, 'B', 14.02, ..., 0.1589, 0.05586, 0.2142],
       ...,
       [908194, 'M', 20.18, ..., 0.1724, 0.06053, 0.4331],
       [916799, 'M', 18.31, ..., 0.186, 0.05941, 0.5449],
       [914862, 'B', 15.04, ..., 0.1668, 0.06869, 0.372]], dtype=object)

[ ] X_train

       [-4.14433134,  1.37391708],
       [ 2.45009727, -2.49336285],
       [-1.20844631, -2.30679956],
       [ 2.55631466, -0.98550214],
       [-1.6091476 ,  0.55066705],
       [-5.52462148,  2.19178828],
       [-2.44685583, -2.28937848],
       [-1.95474568, -2.02963924],
       [ 5.54394234,  1.5236766 ],
       [ 5.74409562,  1.85156779],
       [ 1.13553056, -3.93865462],
       [-1.2483554 , -3.08106324],
       [-0.00961488, -3.62708415],
       [ 5.21418108,  2.66981962],
       [ 4.2290474 ,  0.3886969 ],
       [-3.94237521,  0.76214343],
       [ 5.30822458,  2.18894363],
       [-0.20862902, -3.05785486],
       [ 0.47295413, -2.560251  ],
```

Applying LDA #Apply
LDA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA lda
= LDA(n_components = 2)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
X_train

```
X_test

array([[915691, 'M', 13.4, ..., 0.2116, 0.07325, 0.3906],
       [9112367, 'B', 13.21, ..., 0.1619, 0.05584, 0.2084],
       [909410, 'B', 14.02, ..., 0.1589, 0.05586, 0.2142],
       ...,
       [908194, 'M', 20.18, ..., 0.1724, 0.06053, 0.4331],
       [916799, 'M', 18.31, ..., 0.186, 0.05941, 0.5449],
       [914862, 'B', 15.04, ..., 0.1668, 0.06869, 0.372]], dtype=object)

X_train

       [-4.14433134,  1.37391708],
       [ 2.45009727, -2.49336285],
       [-1.20844631, -2.30679956],
       [ 2.55631466, -0.98550214],
       [-1.6091476 ,  0.55066705],
       [-5.52462148,  2.19178828],
       [-2.44685583, -2.28937848],
       [-1.95474568, -2.02963924],
       [ 5.54394234,  1.5236766 ],
       [ 5.74409562,  1.85156779],
       [ 1.13553056, -3.93865462],
       [-1.2483554 , -3.08106324],
       [-0.00961488, -3.62708415],
       [ 5.21418108,  2.66981962],
       [ 4.2290474 ,  0.3886969 ],
       [-3.94237521,  0.76214343],
       [ 5.30822458,  2.18894363],
       [-0.20862902, -3.05785486],
       [ 0.47295413, -2.560251  ],
```

#fitting logistic regression

from sklearn.linear_model import LogisticRegression classifier = LogisticRegression(random_state = 0) classifier.fit(X_train, y_train)

#predict the test results

y_pred = classifier.predict(X_test)
y_pred

```
#fitting logistic regression
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
#predict the test results
y_pred = classifier.predict(X_test)
y_pred

array([1, 3, 2, 1, 2, 2, 1, 3, 2, 2, 3, 3, 1, 2, 3, 2, 1, 1, 2, 1, 2, 1,
       1, 2, 2, 2, 2, 2, 2, 3, 1, 1, 2, 1, 1, 1])

#accuracy by confusion matrix
```

#accuracy by confusion matrix

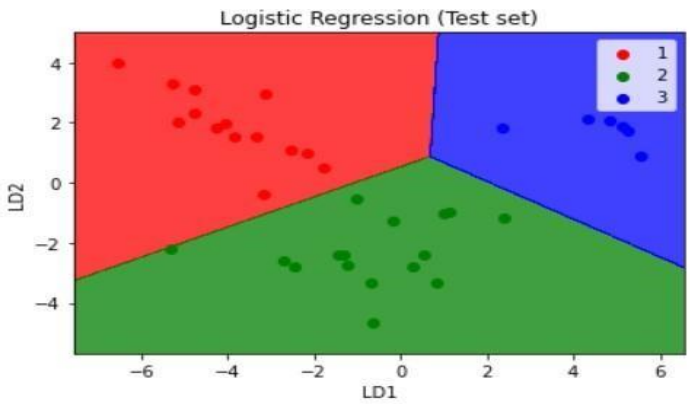
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred) print(cm)
accuracy_score(y_test,y_pred)

```
[ ] #accuracy by confusion matrix
    from sklearn.metrics import confusion_matrix, accuracy_score
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    accuracy_score(y_test,y_pred)
```

```
[[14  0  0]
 [ 0 16  0]
 [ 0  0  6]]
1.0
```

#visualization of test results:

```
from matplotlib.colors import ListedColormap X_set,
y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01), np.a
range(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1,X2,classifier.predict(np.array([X1.ravel(),X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, c
map = ListedColormap(('red', 'green', 'blue')))
plt.xlim(X1.min(),              X1.max())
plt.ylim(X2.min(), X2.max()) for i, j in
enumerate(np.unique(y_set)):
   plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
            c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('LD1') plt.ylabel('LD2')
plt.legend() plt.show()
```



Module 7

Aim :

Regression Analysis: Linear regression, Logistic regression, Polynomial regression

Description:

Regression is a technique for investigating the relationship between independent variables or features and a dependent variable or outcome. It's used as a method for predictive modelling in machine learning, in which an algorithm is used to predict continuous outcomes.

Program:

#LINEAR REGESSION:

```
import pandas as pd import numpy as np from
sklearn.model_selection import train_test_split as tts from
sklearn.linear_model import LinearRegression from
sklearn.metrics import r2_score
from sklearn.preprocessing import PolynomialFeatures

data = pd.read_csv('bottle.csv',nrows=1000) data['Salnty']=data['Salnty'].fillna(value=data['Salnty'].mean())
data['T_degC']=data['T_degC'].fillna(value=data['T_degC'].mean())

x=data[['Salnty']] y=data['T_degC']
pf1=PolynomialFeatures(degree=4)
x1=pf1.fit_transform(x)

regr=LinearRegression() regr.fit(x1,y)
y_pred=regr.predict(x1)

R_square = r2_score(y,y_pred) print('Coefficient

of Determination:', R_square)


ch='y' while(ch=='y' or ch=='Y'):
sal=float(input("Enter Salinity to Predict :"))
sal1=pf1.fit_transform([[sal]])
p=regr.predict(sal1) print("\nTemperature is
",p)
    ch=input("Enter y to calculate more : ")
```

Output:

```
Coefficient of Determination: 0.7838361038646351
Enter Salinity to Predict :32.45

Temperature is [6.07079706]
Enter y to calculate more : n
```

#LOGISTIC REGRESSION

```
import pandas as pd import numpy as np import
matplotlib.pyplot as plt from
sklearn.model_selection import train_test_split from
sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,mean_squared_error
data=pd.read_csv("bottle.csv",nrows=100)
```

```
data['Salnty'] = data['Salnty'].fillna(value=data['Salnty'].mean()) data['T_degC']
= data['T_degC'].fillna(value=data['T_degC'].mean()) x=data['Salnty']
y=data['T_degC']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=3/10,random_state=0)

#Converting int 2-D arrays x_train=x_train.to_numpy().reshape(-1,
1) x_test=x_test.to_numpy().reshape(-1, 1)
y_train=y_train.to_numpy().reshape(-1, 1)
y_test=y_test.to_numpy().reshape(-1, 1)

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

# M and C values
print("Intercept (C) : ",reg.intercept_) print("Slope
        (M) : ",reg.coef_)

#Predection of testing sets y_pred=reg.predict(x_test)
x_pred=reg.predict(x_train) print('Mean Absolute Error :
',mean_absolute_error(y_test,y_pred)) print('Mean Squared Error :
',mean_squared_error(y_test,y_pred))
print('Root MeanSquared Error : ',np.sqrt(mean_squared_error(y_test,y_pred)))
```

Output:

```
Intercept (C) : [131.27879866]
Slope       (M) : [[-3.67906099]]
Mean Absolute Error : 1.0035873375117492
Mean Squared Error :  1.4170202202564186
Root MeanSquared Error : 1.1903865843735044
```

POLYNOMIAL REGRESSION

```
import pandas as pd import numpy as np from
sklearn.model_selection import train_test_split as tts from
sklearn.linear_model import LinearRegression from
sklearn.metrics import r2_score
from sklearn.preprocessing import PolynomialFeatures

data = pd.read_csv('bottle.csv',nrows=1000) data['Salnty']=data['Salnty'].fillna(value=data['Salnty'].mean())
data['T_degC']=data['T_degC'].fillna(value=data['T_degC'].mean())

x=data[['Salnty']] y=data['T_degC']
pf1=PolynomialFeatures(degree=4)
x1=pf1.fit_transform(x)

regr=LinearRegression()
regr.fit(x1,y) y_pred=regr.predict(x1)
```

```
R_square = r2_score(y,y_pred) print('Coefficient

of Determination:', R_square)

ch='y' while(ch=='y' or ch=='Y'):
sal=float(input("Enter Salinity to Predict :"))
sal1=pf1.fit_transform([[sal]])
p=regr.predict(sal1) print("\nTemperature is
",p) ch=input("Enter y to calculate more : ")
```

Output:

```
Coefficient of Determination: 0.7838361038562431
Enter Salinity to Predict :33.44

Temperature is [10.80800579]
Enter y to calculate more : n
```

Module 8 AIM:

Regularized Regression **Program:**

```
import pandas as pd import numpy as np import matplotlib.pyplot as
plt from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split, cross_val_score
from statistics import mean
data = pd.read_csv('/content/drive/MyDrive/auto-mpg.csv') data
```

Output:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10

398 rows × 9 columns

```
data = pd.read_csv('/content/drive/MyDrive/auto-mpg.csv')
# Dropping the numerically non-sensical variables
dropColumns = ['horsepower']
data = data.drop(dropColumns, axis = 1)
# Separating the dependent and independent variables
y = data['mpg']
X = data[['weight', 'acceleration','displacement']]
# Dividing the data into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
```

Evaluating the model:

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

# Evaluating the Linear Regression model
print(linearModel.score(X_test, y_test))
```

```
0.7061885609118318
```

```
ridgeModelChosen = Ridge(alpha = 2)
ridgeModelChosen.fit(X_train, y_train)

# Evaluating the Ridge Regression model
print(ridgeModelChosen.score(X_test, y_test))
```

```
0.7061981540914275
```


```
ridgeModelChosen = Ridge(alpha = 2)
ridgeModelChosen.fit(X_train, y_train)

# Evaluating the Ridge Regression model
print(ridgeModelChosen.score(X_test, y_test))
```

```
0.7061981540914275
```


```
# Building and fitting the Lasso Regression Model
lassoModelChosen = Lasso(alpha = 2, tol = 0.0925)
lassoModelChosen.fit(X_train, y_train)

# Evaluating the Lasso Regression model
print(lassoModelChosen.score(X_test, y_test))
```

```
0.7098594282035298
```

Module 9

AIM:

K-Nearest Neighbour (kNN) Classifier

DESCRIPTION:

The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. It's easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows.

KNN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

PROGRAM:

```
import pandas as pd import numpy as np from
sklearn.model_selection import train_test_split from
sklearn.neighbors import KNeighborsClassifier from
sklearn.metrics import confusion_matrix, accuracy_sc ore
,recall_score,precision_score,f1_score df =
pd.read_csv(&quot;/data.csv&quot;) X= df.iloc[:,
[0,3]].values y= df.iloc[:, 4].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
 = 0.3, random_state = 0)
classifier = KNeighborsClassifier(n_neighbors=5,metric=&quot;e uclidean&quot;)
classifier.fit(X_train, y_train) y_pred
= classifier.predict(X_test)
#confusion matrix confusion_matrix =
confusion_matrix(y_test, y_pred)
print(confusion_matrix) accuracy =
accuracy_score(y_test, y_pred)
recall=recall_score(y_test,y_pred) precision =
precision_score(y_test, y_pred)
f1score=f1_score(y_test,y_pred)
print(&#39;Accuracy of the model:&#39;,accuracy)
print(&#39;precision of the model:&#39;,precision)
print(&#39;Recall of the model:&#39;,recall)
print(&#39;f1_score of the model:&#39;,f1score)
tp=confusion_matrix[0,0] fp=confusion_matrix[0,1]
fn=confusion_matrix[1,0] tn=confusion_matrix[1,1]
senstivity=tp/(tp+fn)
```

print('Sensitivity:',senstivity*100) specificity=tn/(fp+tn)

print('Specificity:',specificity*100)

OUTPUT

[[67 12]

[20 21]]

Accuracy of the model: 0.7333333333333333

precision of the model: 0.6363636363636364

Recall of the model: 0.5121951219512195

f1_score of the model: 0.5675675675675675

Sensitivity: 77.01149425287356

Specificity: 63.63636363636363

KNN with Label Encoding and Scaling

Dataset: Social Networks whether purchased or not

Features: User Id, Gender, Age ,Estimated ,Purchased

Selected X features: userId,Gender,Age

Target Y : Published

PROGRAM:

import pandas as pd import numpy as np from

sklearn.model_selection import train_test_split from

sklearn.preprocessing import LabelEncoder from

sklearn.preprocessing import StandardScaler from

sklearn.neighbors import KNeighborsClassifier

```
from sklearn.metrics import confusion_matrix, accuracy_sc
ore ,recall_score,precision_score,f1_score df =
pd.read_csv(&quot;/data.csv&quot;) X= df.iloc[:,
[0,2]].values y= df.iloc[:, 4].values #label encoding le =
LabelEncoder()
y = le.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
 = 0.3, random_state = 0)
#Scaling sc=
StandardScaler()
X_train= sc.fit_transform(X_train) X_test= sc.transform(X_test)
classifier = KNeighborsClassifier(n_neighbors=5,metric=&quot;e
uclidean&quot;) classifier.fit(X_train, y_train) y_pred =
classifier.predict(X_test)
#confusion matrix confusion_matrix =
confusion_matrix(y_test, y_pred)
print(confusion_matrix) accuracy =
accuracy_score(y_test, y_pred)
recall=recall_score(y_test,y_pred) precision =
precision_score(y_test, y_pred)
f1score=f1_score(y_test,y_pred)
print(&#39;Accuracy of the model:&#39;,accuracy)
print(&#39;precision of the model:&#39;,precision)
print(&#39;Recall of the model:&#39;,recall)
print(&#39;f1_score of the model:&#39;,f1score)
tp=confusion_matrix[0,0] fp=confusion_matrix[0,1]
fn=confusion_matrix[1,0] tn=confusion_matrix[1,1]
senstivity=tp/(tp+fn)
print(&#39;Sensitivity:&#39;,senstivity*100)
specificity=tn/(fp+tn)
print(&#39;Specificity:&#39;,specificity*100)
```

OUTPUT

```
[[67 12]
```

[13 28]]

Accuracy of the model: 0.7916666666666666 precision

of the model: 0.7

Recall of the model: 0.6829268292682927 f1_score

of the model: 0.6913580246913581

Sensitivity: 83.75

Specificity: 70.0

KNN with Principal Component Analysis

PROGRAM

```
import pandas as pd from sklearn.preprocessing import
StandardScaler,LabelEnc oder from sklearn.decomposition
import PCA from sklearn.neighbors import
KNeighborsClassifier from sklearn.metrics import
confusion_matrix,accuracy_sco
re,recall_score,precision_score,f1_score
df=pd.read_csv(&quot;/data.csv&quot;) X= df.iloc[:,
[0,2]].values y= df.iloc[:, 4].values
#Principal Component Analysis pca=PCA(n_components=2)
principalComponents=pca.fit_transform(X)
principalDf=pd.DataFrame(data=principalComponents,columns
=[&quot;pc1&quot;,&quot;pc2&quot;])
finalDf=pd.concat([principalDf,df[[&#39;Purchased&#39;]]],axis=1)
finalDf=pd.DataFrame(finalDf)
#print(finalDf)
X=finalDf[[&#39;pc1&#39;,&#39;pc2&#39;]].values
y=finalDf[&#39;Purchased&#39;].values
#Label Encoding and Scaling
le = LabelEncoder() y =
le.fit_transform(y)
X_train,X_test,y_train,y_test=train_test_split(X,y,test_s
ize=0.3,random_state=0)
sc= StandardScaler()
X_train= sc.fit_transform(X_train)
```

```
X_test= sc.transform(X_test) #kNN classifier classifier =
KNeighborsClassifier(n_neighbors=10,metric=&#39;
euclidean&#39;) classifier.fit(X_train,y_train)
y_pred=classifier.predict(X_test) #Confusion Matrix
confusion_matrix=confusion_matrix(y_test,y_pred)
print(confusion_matrix)
accuracy=accuracy_score(y_test,y_pred)*100
precision=precision_score(y_test,y_pred)*100
recall=recall_score(y_test,y_pred)*100
f1_measure=f1_score(y_test,y_pred)*100 print(&#39;Accuracy
of the model:&#39;,accuracy) print(&#39;Precision of the
model:&#39;,precision) print(&#39;Recall of the
model:&#39;,recall) print(&#39;F1 Measure of the
model:&#39;,f1_measure) tp=confusion_matrix[0,0]
fp=confusion_matrix[0,1] fn=confusion_matrix[1,0]
tn=confusion_matrix[1,1] senstivity=tp/(tp+fn)
print(&#39;Sensitivity:&#39;,senstivity*100)
specificity=tn/(fp+tn)
print(&#39;Specificity:&#39;,specificity*100)
```

OUTPUT

```
[[75 4]
[13 28]]
Accuracy of the model: 85.83333333333333
Precision of the model: 87.5
Recall of the model: 68.29268292682927
F1 Measure of the model: 76.7123287671233
Sensitivity: 85.22727272727273
Specificity: 87.5
```

AIM:

Module 10

Support Vector Machines (SVMs)

DESCRIPTION:

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split as tts
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix,accuracy_score

data=pd.read_csv('Social_Network_Ads.csv')
x=data.iloc[:,[2,3]].values y=data.iloc[:,4].values

x_train,x_test,y_train,y_test=tts(x,y,test_size=0.25,random_state=0)

sc=StandardScaler() x_train=sc.fit_transform(x_train)
x_test=sc.fit_transform(x_test)

model=SVC(kernel='rbf',random_state=0)
# Why 'rbf', because it is nonlinear and gives better results as compared to linear
model.fit(x_train,y_train) y_pred=model.predict(x_test)

cm=confusion_matrix(y_test,y_pred) print("Confusion
Matrix : \n",cm)
print("\nAccuracy Score :",accuracy_score(y_test,y_pred)*100)
```

OUTPUT:

```
Confusion Matrix :
[[64 4]
[ 3 29]]

Accuracy Score : 93.0
```

AIM:

Module 11

Random Forest model

DESCRIPTION:

The random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split as tts
from sklearn.preprocessing import StandardScaler from
sklearn.metrics import confusion_matrix,accuracy_score from
sklearn.ensemble import RandomForestClassifier

data=pd.read_csv("Social_Network_Ads.csv")
x=data.iloc[:,[2,3]].values y=data.iloc[:,4].values

x_train,x_test,y_train,y_test=tts(x,y,test_size=0.3,random_state=0)

sc=StandardScaler() x_train=sc.fit_transform(x_train)

x_test=sc.fit_transform(x_test)

forest=RandomForestClassifier(criterion='gini',n_estimators=10)

forest.fit(x_train, y_train)

y_pred = forest.predict(x_test)
cm=confusion_matrix(y_test,y_pred) print("Confusion
Matrix : \n",cm)
print("\nAccuracy Score :",accuracy_score(y_test,y_pred)*100)
```

AIM:

OUTPUT:

Confusion Matrix :
[[72 7] [
5 36]]

Accuracy Score : 90.0

Module 12

AdaBoost Classifier and XGBoost

DESCRIPTION:

The AUC results show that AdaBoost and XGBoost model have similar value 0.94 and 0.95. To obtain the AdaBoost model we need to run model for 60 minutes, while the XGBoost model only need ~60 seconds. We can say that XGBoost works better than AdaBoost for speed.

PROGRAM FOR ADABOOST:

```
import pandas as pd from sklearn.model_selection
import train_test_split from sklearn.ensemble
import AdaBoostClassifier
df= pd.read_csv('diabetes.csv')
x=df[['Age','Glucose']] y=df['Outcome']

x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
abc = AdaBoostClassifier(n_estimators=50, learning_rate=1) model = abc.fit(x_train,
y_train) y_pred = model.predict(x_test)
from sklearn.metrics import mean_absolute_error, mean_squared_error,r2_score,accuracy_score
Accuracy =accuracy_score(y_test,y_pred)
print("Accuracy:",Accuracy*100)
```

OUTPUT:

Accuracy: 72.91666666666666

AIM:

PROGRAM FOR XGBOOST:

```
import pandas as pd
from sklearn.model_selection import train_test_split as tts
from sklearn.metrics import accuracy_score from xgboost
import XGBClassifier

df= pd.read_csv('diabetes.csv')
x=df[['Age','Glucose']] y=df['Outcome']
x_train, x_test, y_train, y_test= tts(x, y, test_size= 0.25, random_state=0)
model = XGBClassifier() model.fit(x_train, y_train) y_pred =
model.predict(x_test)

Accuracy =accuracy_score(y_test,y_pred) print("Accuracy:",Accuracy*100)
```

OUTPUT:

Accuracy: 75.0