B.Tech. (IV Sem.)	2AM51 –INTRODUCTION TO ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LAB	L	T	P	Cr.
		0	0	3	1.5

Pre-requisite : Python Programming

Course Educational Objective: The main objective of this course is that a student will be familiar with principles behind the Object-Oriented Design and able to apply those principles in a project setting. Students will analyze applications and know how to take a pragmatic approach to software design and development.

Course Outcomes (CO): *At the end of this course, the student will be able to:*

CO1: Apply the basic principles of AI in problem solving using LISP/PROLOG. **(Apply – L3)**

CO2: Implement different algorithms using LISP/PROLOG.**(Apply – L3)**

CO3: Develop an Expert System using JESS/PROLOG**(Apply – L3)**

CO 4: Improve individual / teamwork skills, communication & report writing skills with ethical values.

List of Experiments (Artificial Intelligence)

1. Implementation of DFS for water jug problem using LISP/PROLOG.
2. Implementation of BFS for tic-tac-toe problem using LISP/PROLOG/Java.
3. Implementation of TSP using heuristic approach using Java/LISP/Prolog
4. Implementation of Simulated Annealing Algorithm using LISP/PROLOG
5. Implementation of Hill-climbing to solve 8- Puzzle Problem
6. Implementation of Monkey Banana Problem using LISP/PROLOG

List of Experiments (Machine Learning)
Python Libraries required: Sklearn
Note: Standard datasets can be downloaded from UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.php**)**

1. Implement and demonstrate FIND-S algorithm for finding the most specific hypothesis based on agiven set of training data samples. Read the training data from a .csv file.
2. For a given set of training data examples stored in a .csv file, implement and demonstrate thecandidate elimination algorithm to output a description of the set of all hypotheses consistent withthe training examples.
3. Write a program to demonstrate the working of the decision tree classifier. Use appropriate datasetfor building the decision tree and apply this knowledge to classify a new sample.
4. Write a program to demonstrate the working of Decision tree regressor. Use appropriate dataset for decision tree regressor.
5. Write a program to demonstrate the working of Random Forest classifier. Use appropriate dataset for Random Forest Classifier.
6. Write a program to demonstrate the working of Logistic Regression classifier. Use appropriate dataset for Logistic Regression.

Experiments (Machine Learning)

AIM :**Develop a program to make Linear Regression Model ?**

Description :-

Linear Regression :-

Linear regression is a statistical method that models the relationship between a dependent variable and one independent variable by fitting a straight line to the data.

$y = a_0 + a_1 x + \varepsilon$
Y= Dependent Variable (Target Variable)
X= Independent Variable (predictor Variable)
a0= intercept of the line (Gives an additional degree of freedom)
a1 = Linear regression coefficient (scale factor to each input value).
ε = random error

Data Set : **Attendence.csv**

S.NO	Attendenc	Marks
1	60	63
2	66	64
3	67	65
4	68	66
5	69	67
6	70	68
7	71	69
8	72	70
9	73	71
10	74	72
11	75	73
12	76	74
13	77	75
14	78	76
15	79	77
16	80	78
17	81	79
18	82	80
19	83	81
20	84	82

Program :

```
# read the data
import pandas as pd
df= pd.read_csv("Attendence.csv")
print(df)
#making independent and dependent variables
#linear
X = df[['Attendence']]
y = df['Marks']
print(X)
print(X.shape)
print(y)
print(y.shape)
# dividing the data into testing and training data set
# training = 80 % testing = 20%
#import the module
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) # 0.2 = 20%
print(X_train)
print(X_train.shape)
print(X_test)
print(X_test.shape)
print(y_train)
print(y_test)
#making the linear regression model
#fit is used to train the machine
# First import the linear regression
```

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(X_train,y_train)

# predicting value by the machine by our train data

result = model.predict([[89]]) # input

print(result)

# predictint the values of all test data

prediction = model.predict(X_test) # input # pridicting the output by model with training data

print(prediction)

# checking the machine how accurately the machine giving the output

# by r2_score

# by comparing the predicted values and y_test # output

#first import

from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score

predit_r2_score = r2_score(y_test,prediction)

print(predit_r2_score) # 1.0 = 100%

# predicting the actual and predicted value is same.. 0.0 = same

print("Mean Squared Error : " , mean_squared_error(y_test,prediction))

data = pd.DataFrame({'Actual ML_marks :': y_test , 'predicted_Values by AI :': prediction})

print(data)
```

Output :

```
S.NO  Attendence  Marks
0       1           60      63
1       2           66      64
2       3           67      65
3       4           68      66
4       5           69      67
5       6           70      68
6       7           71      69
```

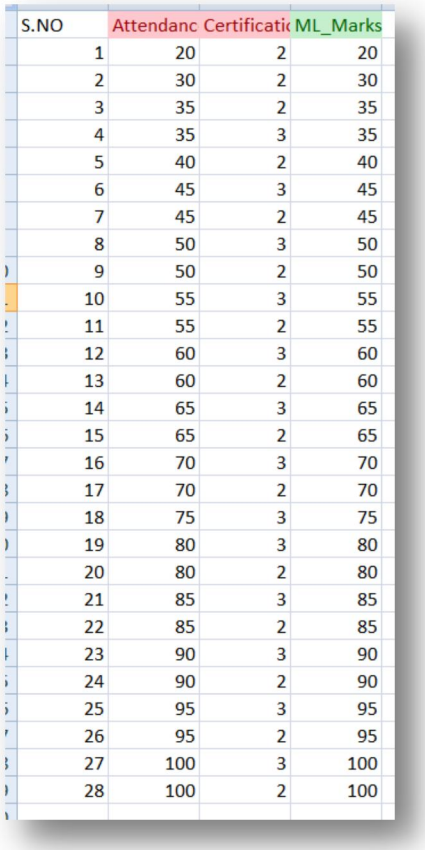
```
7      8          72     70
8      9          73     71
9     10          74     72
10    11          75     73
11    12          76     74
12    13          77     75
13    14          78     76
14    15          79     77
15    16          80     78
16    17          81     79
17    18          82     80
18    19          83     81
19    20          84     82
    Attendence
0           60
1           66
2           67
3           68
4           69
5           70
6           71
7           72
8           73
9           74
10          75
11          76
12          77
13          78
14          79
15          80
16          81
17          82
18          83
19          84
(20, 1)
0     63
1     64
2     65
3     66
4     67
5     68
6     69
7     70
8     71
```

```
9     72
10    73
11    74
12    75
13    76
14    77
15    78
16    79
17    80
18    81
19    82
Name: Marks, dtype: int64
(20,)
    Attendence
0           60
1           66
15          80
13          78
2           67
17          82
7           72
3           68
10          75
6           71
9           74
4           69
19          84
14          79
8           73
12          77
(16, 1)
    Attendence
11          76
18          83
16          81
5           70
(4, 1)
0     63
1     64
15    78
13    76
2     65
17    80
7     70
```

```
3     66
10    73
6     69
9     72
4     67
19    82
14    77
8     71
12    75
Name: Marks, dtype: int64
11    74
18    81
16    79
5     68
Name: Marks, dtype: int64
[85.65264411]
[74.0391904  80.29258855 78.50590336 68.67913483]
0.9880463123148753
Mean Squared Error :  0.30183061404939876
    Actual ML_marks :  predicted_Values by AI :
11                 74                 74.039190
18                 81                 80.292589
16                 79                 78.505903
5                  68                 68.679135
```

AIM :**Develop a program to make Multi Linear Regression Model ?**

Data Set : **ml_data.csv**



S.NO	Attendanc	Certificati	ML_Marks
1	20	2	20
2	30	2	30
3	35	2	35
4	35	3	35
5	40	2	40
6	45	3	45
7	45	2	45
8	50	3	50
9	50	2	50
10	55	3	55
11	55	2	55
12	60	3	60
13	60	2	60
14	65	3	65
15	65	2	65
16	70	3	70
17	70	2	70
18	75	3	75
19	80	3	80
20	80	2	80
21	85	3	85
22	85	2	85
23	90	3	90
24	90	2	90
25	95	3	95
26	95	2	95
27	100	3	100
28	100	2	100

< Program >

```
import pandas as pd

df = pd.read_csv('ml_data.csv')

print(df)
```

 # dividing the dependent and indepedent variables

#here, the multi linear have two inputs

```
X = df[['Attendance of Ml students ', 'Certifications based on ML']] # input

y = df['ML_Marks'] #output

print(X)
```

```
print(X.shape)
print(y)
print(y.shape)
#spliting the data into training and testing = training = 80% , testing  = 20%
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)
print(X_train)
print(X_train.shape)
print(X_test)
print(X_test.shape)
print(y_train)
print(y_train.shape)
print(y_test)
print(y_test.shape)
#make the multilinear model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)
#predict the output [marks] from inputs of attendence and certifications
result = model.predict([[45,4]])
print(result)
predicted_result = model.predict(X_test)
print(predicted_result)
from sklearn.metrics import r2_score,mean_squared_error
predicted_r2_score = r2_score(y_test,predictes_result)
print(predicted_r2_score)
print("Mean Squared Error : " , mean_squared_error(y_test,predicted_result))
```

```
data = pd.DataFrame({'Actual ML_marks :': y_test , 'predicted_Values by AI :': predicted_result})

print(data)
```

< output >

```
S.NO  Attendance of Ml students   Certifications based on ML  ML_Marks
0      1                          20                          2         20
1      2                          30                          2         30
2      3                          35                          2         35
3      4                          35                          3         35
4      5                          40                          2         40
5      6                          45                          3         45
6      7                          45                          2         45
7      8                          50                          3         50
8      9                          50                          2         50
9     10                          55                          3         55
10    11                          55                          2         55
11    12                          60                          3         60
12    13                          60                          2         60
13    14                          65                          3         65
14    15                          65                          2         65
15    16                          70                          3         70
16    17                          70                          2         70
17    18                          75                          3         75
18    19                          80                          3         80
19    20                          80                          2         80
20    21                          85                          3         85
21    22                          85                          2         85
22    23                          90                          3         90
23    24                          90                          2         90
24    25                          95                          3         95
25    26                          95                          2         95
26    27                         100                          3        100
27    28                         100                          2        100
    Attendance of Ml students   Certifications based on ML
0                           20                           2
1                           30                           2
2                           35                           2
3                           35                           3
4                           40                           2
5                           45                           3
6                           45                           2
7                           50                           3
8                           50                           2
9                           55                           3
10                          55                           2
11                          60                           3
12                          60                           2
13                          65                           3
14                          65                           2
15                          70                           3
16                          70                           2
17                          75                           3
18                          80                           3
19                          80                           2
20                          85                           3
21                          85                           2
22                          90                           3
23                          90                           2
24                          95                           3
25                          95                           2
26                         100                           3
27                         100                           2
(28, 2)
0      20
1      30
2      35
3      35
```

```
4      40
5      45
6      45
7      50
8      50
9      55
10     55
11     60
12     60
13     65
14     65
15     70
16     70
17     75
18     80
19     80
20     85
21     85
22     90
23     90
24     95
25     95
26    100
27    100
Name: ML_Marks, dtype: int64
(28,)
    Attendance of Ml students   Certifications based on ML
13                          65                           3
9                           55                           3
3                           35                           3
27                         100                           2
18                          80                           3
22                          90                           3
15                          70                           3
8                           50                           2
25                          95                           2
24                          95                           3
21                          85                           2
14                          65                           2
2                           35                           2
19                          80                           2
10                          55                           2
11                          60                           3
26                         100                           3
5                           45                           3
1                           30                           2
16                          70                           2
12                          60                           2
4                           40                           2
(22, 2)
    Attendance of Ml students   Certifications based on ML
23                          90                           2
17                          75                           3
0                           20                           2
6                           45                           2
7                           50                           3
20                          85                           3
(6, 2)
13     65
9      55
3      35
27    100
18     80
22     90
15     70
8      50
25     95
24     95
21     85
14     65
2      35
```

```
19     80
10     55
11     60
26    100
5      45
1      30
16     70
12     60
4      40
Name: ML_Marks, dtype: int64
(22,)
23    90
17    75
0     20
6     45
7     50
20    85
Name: ML_Marks, dtype: int64
(6,)
[45.]
[90. 75. 20. 45. 50. 85.]
1.0
Mean Squared Error :  1.682903264471492e-28
    Actual ML_marks :  predicted_Values by AI :
23                 90                        90.0
17                 75                        75.0
0                  20                        20.0
6                  45                        45.0
7                  50                        50.0
20                 85                        85.0
```

AIM :**Develop a program to make Polynomial Regression Model ?**

Description :-

Polynomial regression is a type of linear regression in which the relationship between the dependent variable and one or more independent variables is modeled as an nth-degree polynomial. The technique allows for a more flexible model that can capture nonlinear relationships between the variables. The degree of the polynomial can be chosen based on the complexity of the relationship between the variables, but higher degrees can lead to overfitting. The parameters of the polynomial regression model can be estimated using methods such as the least squares method or maximum likelihood estimation. Once the parameters are estimated, they can be used to make predictions about the dependent variable based on the independent variables.

$$f(x) = c_0 + c_1 x + c_2 x^2 \cdots c_n x^n$$

Data Set : **Marks.csv**

S.NO	Attendanc	ML_Marks
1	20	20
2	30	30
3	35	35
4	35	35
5	40	40
6	45	45
7	45	45
8	50	50
9	50	50
10	55	55
11	55	55
12	60	60
13	60	60
14	65	65
15	65	65
16	70	70
17	70	70
18	75	75
19	80	80
20	80	80
21	85	85
22	85	85
23	90	90
24	90	90
25	95	95
26	95	95
27	100	100
28	100	100

< program >

```
import pandas as pd

df = pd.read_csv('Marks.csv')

print(df)

#making the independent and dependent varables  = input and output varibles

# the polynomial regression is same as linear regression

X = df[['Attendance of Ml students ']]

y = df['ML_Marks']

 print(X)

print(X.shape)

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)

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear_model import LinearRegression


# create polynomial features

poly = PolynomialFeatures(degree=2)

#trasform is used to transform the the train data into all polynomial combinations
  upto specified degree = 2 and fit

X_train_poly = poly.fit_transform(X_train)

#trasform is used to transform the the test data into all polynomial combinations
  upto specified degree = 2

X_test_poly = poly.transform(X_test)


# fit polynomial regression model
```

```
model = LinearRegression()

model.fit(X_train_poly, y_train)


# make predictions on testing data

y_pred = model.predict(X_test_poly)

print(y_pred)

print(poly) # polynomialfeatures

print(X_train_poly) # trained polynomials

print(X_test_poly)

print("m=", model.coef_)

print("c=",model.intercept_)


from sklearn.metrics import r2_score, mean_squared_error

# make predictions on testing data

y_pred = model.predict(X_test_poly)

print(y_pred)

#r2_score value of y_test and predit

print("R-squared score on testing data: ", r2_score(y_test, y_pred))

#mean squared error

print("Mean squared error on testing data: ", mean_squared_error(y_test,y_pred))

# make prediction for new instance

new_instance = [[32]]  # assuming there is only one feature

new_instance_poly = poly.transform(new_instance)

prediction = model.predict(new_instance_poly) # we only predict the 3 inputs bacause the degree=2
```

print("Prediction for new instance: ", prediction)

#y_test(output)and x_test(input) predicted values of polynomial is same or not (always prediction = x_test == x_test_poly(degree=2))

data = pd.DataFrame({'Actual ML_marks :': y_test , 'predicted_Values by AI :': y_pred})

print(data)

< output >

```
S.NO  Attendance of Ml students   ML_Marks
0      1                          20        20
1      2                          30        30
2      3                          35        35
3      4                          35        35
4      5                          40        40
5      6                          45        45
6      7                          45        45
7      8                          50        50
8      9                          50        50
9     10                          55        55
10    11                          55        55
11    12                          60        60
12    13                          60        60
13    14                          65        65
14    15                          65        65
15    16                          70        70
16    17                          70        70
17    18                          75        75
18    19                          80        80
19    20                          80        80
20    21                          85        85
21    22                          85        85
22    23                          90        90
23    24                          90        90
24    25                          95        95
25    26                          95        95
26    27                         100       100
27    28                         100       100
    Attendance of Ml students
0                           20
1                           30
2                           35
3                           35
4                           40
5                           45
6                           45
7                           50
8                           50
9                           55
10                          55
11                          60
12                          60
13                          65
14                          65
15                          70
16                          70
17                          75
18                          80
```

```
19                          80
20                          85
21                          85
22                          90
23                          90
24                          95
25                          95
26                         100
27                         100
(28, 1)
[35. 50. 60. 70. 80. 65.]
PolynomialFeatures()
[[1.000e+00 5.500e+01 3.025e+03]
 [1.000e+00 1.000e+02 1.000e+04]
 [1.000e+00 4.000e+01 1.600e+03]
 [1.000e+00 7.500e+01 5.625e+03]
 [1.000e+00 5.000e+01 2.500e+03]
 [1.000e+00 8.000e+01 6.400e+03]
 [1.000e+00 9.000e+01 8.100e+03]
 [1.000e+00 6.000e+01 3.600e+03]
 [1.000e+00 9.500e+01 9.025e+03]
 [1.000e+00 3.500e+01 1.225e+03]
 [1.000e+00 4.500e+01 2.025e+03]
 [1.000e+00 4.500e+01 2.025e+03]
 [1.000e+00 9.000e+01 8.100e+03]
 [1.000e+00 3.000e+01 9.000e+02]
 [1.000e+00 8.500e+01 7.225e+03]
 [1.000e+00 7.000e+01 4.900e+03]
 [1.000e+00 2.000e+01 4.000e+02]
 [1.000e+00 8.500e+01 7.225e+03]
 [1.000e+00 1.000e+02 1.000e+04]
 [1.000e+00 6.500e+01 4.225e+03]
 [1.000e+00 5.500e+01 3.025e+03]
 [1.000e+00 9.500e+01 9.025e+03]]
[[1.000e+00 3.500e+01 1.225e+03]
 [1.000e+00 5.000e+01 2.500e+03]
 [1.000e+00 6.000e+01 3.600e+03]
 [1.000e+00 7.000e+01 4.900e+03]
 [1.000e+00 8.000e+01 6.400e+03]
 [1.000e+00 6.500e+01 4.225e+03]]
m= [ 0.00000000e+00  1.00000000e+00 -6.11661943e-17]
c= -5.684341886080802e-14
[35. 50. 60. 70. 80. 65.]
R-squared score on testing data:  1.0
Mean squared error on testing data:  2.2298468254247266e-27
Prediction for new instance:  [32.]
    Actual ML_marks :  predicted_Values by AI :
2                  35                       35.0
8                  50                       50.0
12                 60                       60.0
15                 70                       70.0
19                 80                       80.0
14                 65                       65.0
```

AIM :**Develop a program to make logistic Regression Model ?**

Description :-

Logistic regression is a statistical technique used to model the relationship between a binary dependent variable and one or more independent variables. The technique estimates the probability of the dependent variable taking a particular value (e.g., 0 or 1) based on the values of the independent variables. The model uses a logistic function to transform the linear regression equation into a probability score between 0 and 1

Data Set : **diabetes.csv**

Pregnancie	Glucose	BloodPressure	SkinThickn	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0
7	196	90	0	0	39.8	0.451	41	1
9	119	80	35	0	29	0.263	29	1
11	143	94	33	146	36.6	0.254	51	1
10	125	70	26	115	31.1	0.205	41	1
7	147	76	0	0	39.4	0.257	43	1
1	97	66	15	140	23.2	0.487	22	0
13	145	82	19	110	22.2	0.245	57	0
5	117	92	0	0	34.1	0.337	38	0

This code performs the following steps:

1. Loads the diabetes dataset and checks for missing values
2. Separates the independent and dependent variables
3. Splits the dataset into training and test sets
4. Fits a logistic regression model on the training set
5. Predicts on the test set and creates a confusion matrix heatmap
6. Calculates evaluation metrics (accuracy, precision, recall, and F1 score)
7. Prompts the user to input new data to make a prediction on
8. Makes a prediction on the new data using the trained model and prints the result

< program >

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
import seaborn as sns
import matplotlib.pyplot as plt

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

# Check for missing values
print(df.isnull().sum())
```

```
# Separate the independent and dependent variables
X = df[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']]
y = df['Outcome']

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=0.25, random_state=0)

# Fit a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create the heatmap using seaborn
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d")

# Set the axis labels and title
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
```

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

# Calculate the evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)

# Take input from the user to predict on new data
pregnancies = int(input("Enter the number of pregnancies: "))
glucose = int(input("Enter the glucose level: "))
blood_pressure = int(input("Enter the blood pressure: "))
skin_thickness = int(input("Enter the skin thickness: "))
insulin = int(input("Enter the insulin level: "))
bmi = float(input("Enter the BMI: "))
dpf = float(input("Enter the diabetes pedigree function: "))
age = int(input("Enter the age: "))
```

```
# Create a new row with the user's input
new_data = pd.DataFrame({
    'Pregnancies': [pregnancies],
    'Glucose': [glucose],
    'BloodPressure': [blood_pressure],
    'SkinThickness': [skin_thickness],
    'Insulin': [insulin],
    'BMI': [bmi],
    'DiabetesPedigreeFunction': [dpf],
    'Age': [age]
})

# Make a prediction on the new data using the trained model
prediction = model.predict(new_data)

# Print the prediction
if prediction[0] == 0:
    print("Prediction: Not Diabetic")
else:
    print("Prediction: Diabetic")
```

< Output >

```
Pregnancies                 0
```

```
Glucose                       0
BloodPressure                 0
SkinThickness                 0
Insulin                       0
BMI                           0
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
C:\Users\saichand.digumarthi\anaconda3\lib\site-packages\sklearn\linear_model\_logist
  ic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(

Accuracy: 0.7916666666666666
Precision: 0.7115384615384616
Recall: 0.5967741935483871
F1 Score: 0.6491228070175439
Enter the number of pregnancies: 2
Enter the glucose level: 34
Enter the blood pressure: 120
Enter the skin thickness: 67
Enter the insulin level: 55
Enter the BMI: 78
Enter the diabetes pedigree function: 45
Enter the age: 34
Prediction: Diabetic
```

Write a program to demonstrate the working of the decision tree classifier. Use appropriate datasetfor building the decision tree and apply this knowledge to classify a new sample.

AIM : **Develop a program to make Decision tree classifier Model ?**

Data Set : **diabetes2.csv**

Pregnancie	Glucose	BloodPressure	SkinThickn	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0
7	196	90	0	0	39.8	0.451	41	1
9	119	80	35	0	29	0.263	29	1
11	143	94	33	146	36.6	0.254	51	1
10	125	70	26	115	31.1	0.205	41	1
7	147	76	0	0	39.4	0.257	43	1
1	97	66	15	140	23.2	0.487	22	0
13	145	82	19	110	22.2	0.245	57	0
5	117	92	0	0	34.1	0.337	38	0

< program >

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

 # load the dataset

```
df=pd.read_csv("diabetes2.csv")
```

 # display the dataset

```
df
```

 # generate descriptive statistics of the dataset

```
df.describe()

# check for missing values in the dataset
print(df.isnull().sum())

# compute the correlation matrix of the dataset
df.corr()

# separate the input features (X) and the target variable (y)
X=df.drop("Outcome",axis=1)
y=df[['Outcome']]

# split the dataset into training and testing sets
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)

# print the shapes of the datasets
print(X.shape)
print(y.shape)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

# perform feature scaling on the training and testing sets
from sklearn.preprocessing import StandardScaler
st=StandardScaler()
X_train=st.fit_transform(X_train)
X_test=st.fit_transform(X_test)

# train a decision tree classifier on the training set
from sklearn.tree import DecisionTreeClassifier
classifier=DecisionTreeClassifier(criterion='entropy',random_state=0)
classifier.fit(X_train,y_train)

# make predictions on the testing set
y_pred=classifier.predict(X_test)

# compute the confusion matrix
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm

# compute the classification report and accuracy score
```

```
from sklearn.metrics import classification_report,accuracy_score
res=classification_report(y_test,y_pred)
print("Classification Report:")
print(res)
result=accuracy_score(y_test,y_pred)
print("Accuracy:",result)
```

 # compute the accuracy, specificity, sensitivity, and F1 score

```
from sklearn.metrics import accuracy_score,recall_score,f1_score
Accuracy=accuracy_score(y_test,y_pred)
Specificity = cm[0,0]/(cm[0,0]+cm[0,1])
Sensitivity_recall = recall_score(y_test, y_pred)
F1_score = f1_score(y_test,y_pred)

print("Accuracy:",Accuracy)
print("Specificity:",Specificity)
print("Sensitivity/Recall:",Sensitivity_recall)
print("F1 Score:",F1_score)
```

 # compute the mean squared error

```
from sklearn.metrics import mean_squared_error
res1=mean_squared_error(y_test,y_pred)
res1
```

 # compute the confusion matrix, classification report, and accuracy score

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred))
```

 # compute the R2 score

```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
predit_r2_score = r2_score(y_test,y_pred)
print(predit_r2_score) # 1.0 = 100%
```

< Output >

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
(768, 8)
(768, 1)
(576, 8)
(192, 8)
(576, 1)
(192, 1)
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.79      0.82       130
           1       0.61      0.69      0.65        62

    accuracy                           0.76       192
   macro avg       0.73      0.74      0.73       192
weighted avg       0.77      0.76      0.76       192

Accuracy: 0.7604166666666666
Accuracy: 0.7604166666666666
Specificity: 0.7923076923076923
Sensitivity/Recall: 0.6935483870967742
F1 Score: 0.6515151515151515
Confusion Matrix:
[[103  27]
 [ 19  43]]

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.79      0.82       130
           1       0.61      0.69      0.65        62

    accuracy                           0.76       192
   macro avg       0.73      0.74      0.73       192
weighted avg       0.77      0.76      0.76       192


Accuracy Score:
0.7604166666666666
-0.09578163771712145
```

AIM : **Develop a program to make Decision tree regressor Model ?**

Data Set : **Attendence.csv**

S.NO	Attendenc	Marks
1	60	63
2	66	64
3	67	65
4	68	66
5	69	67
6	70	68
7	71	69
8	72	70
9	73	71
10	74	72
11	75	73
12	76	74
13	77	75
14	78	76
15	79	77
16	80	78
17	81	79
18	82	80
19	83	81
20	84	82

< program >

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Load the dataset
```

```
df = pd.read_csv('Attendence.csv')

print(df)

# Define input and output variables

X = df[['Attendence']] # Use 'Attendance' as input

y = df['Marks']

# Print shapes of input and output data

print(X.shape)

print(y.shape)

# Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

# Print shapes of training and testing data

print(X_train.shape)

print(X_test.shape)

print(y_train.shape)

print(y_test.shape)

# Create a decision tree regressor with max_depth=3

tree = DecisionTreeRegressor(max_depth=3)

# Fit the model using the training data

tree.fit(X_train, y_train)
```

 # Predict on the test data

```
y_pred = tree.predict(X_test)

print(y_pred)
```

 # Evaluate the model using mean absolute error

```
mae = mean_absolute_error(y_test, y_pred)

print('Mean Absolute Error:', mae)
```

 # Evaluate the performance of the model using root mean squared error

```
mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

print('Root Mean Squared Error:', rmse)
```

< Output >

S.NO	Attendence	Marks	
0	1	60	63
1	2	66	64
2	3	67	65
3	4	68	66
4	5	69	67
5	6	70	68
6	7	71	69
7	8	72	70
8	9	73	71
9	10	74	72
10	11	75	73
11	12	76	74
12	13	77	75
13	14	78	76
14	15	79	77
15	16	80	78

```
16    17          81     79
17    18          82     80
18    19          83     81
19    20          84     82
(20, 1)
(20,)
(15, 1)
(5, 1)
(15,)
(5,)
[79.5 65.5 79.5 69.5 72. ]
Mean Absolute Error: 1.6
Root Mean Squared Error: 1.6733200530681511
```

AIM : **Write a program to demonstrate the working of Random Forest classifier. Use appropriate dataset for Random Forest Classifier**.

Data Set : **diabetes2.csv**

Pregnancie	Glucose	BloodPressure	SkinThickn	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0
7	196	90	0	0	39.8	0.451	41	1
9	119	80	35	0	29	0.263	29	1
11	143	94	33	146	36.6	0.254	51	1
10	125	70	26	115	31.1	0.205	41	1
7	147	76	0	0	39.4	0.257	43	1
1	97	66	15	140	23.2	0.487	22	0
13	145	82	19	110	22.2	0.245	57	0
5	117	92	0	0	34.1	0.337	38	0

< program >

```
import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix


# Load the dataset
```

```
df = pd.read_csv("diabetes2.csv")

# Split the dataset into features and target
X = df.drop("Outcome", axis=1)
y = df["Outcome"]

# Split the dataset into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

# Create the random forest classifier and fit the model using the training data
classifier = RandomForestClassifier(n_estimators=100, random_state=0)
classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = classifier.predict(X_test)

# Print the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print("{:<10}{:<10}{}".format("", "Actual", ""))
print("{:<10}{:<10}{:<10}".format("", "Negative", "Positive"))
print("{:<10}{:<10}{:<10}".format("Predicted", "----------", "----------"))
print("{:<10}{:<10}{:<10}".format("Negative", cm[0][0], cm[0][1]))
```

```
print("{:<10}{:<10}{:<10}".format("Positive", cm[1][0], cm[1][1]))
```

```
# Print the classification report
from sklearn.metrics import classification_report
res = classification_report(y_test, y_pred)
print("\nClassification Report:\n", res)
```

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create the heatmap using seaborn
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d")
#In this code snippet, we set the fmt parameter of seaborn's heatmap function to "d",
#which specifies that the numbers should be displayed as integers (i.e., no decimal places).
#This will display the confusion matrix heatmap with full numbers instead of in scientific notation.

# Set the axis labels and title
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

```
plt.title("Confusion Matrix")
```

 # Show the plot

```
plt.show()
```

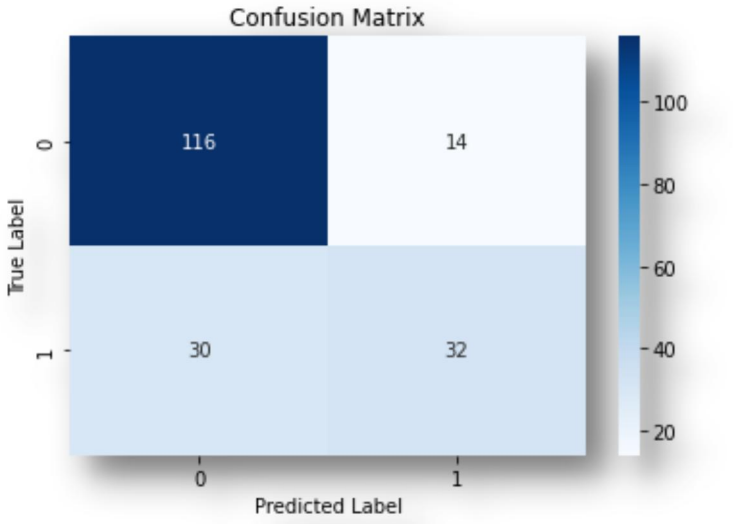
< Output >

```
Confusion Matrix:
          Actual
          Negative  Positive
Predicted --------------------
Negative  116       14
Positive  30        32

Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.89      0.84       130
           1       0.70      0.52      0.59        62

    accuracy                           0.77       192
   macro avg       0.75      0.70      0.72       192
weighted avg       0.76      0.77      0.76       192
```



AIM : **Implement and demonstrate FIND-S algorithm for finding the most specific hypothesis based on agiven set of training data samples. Read the training data from a .csv file.**

Data Set : **my.csv**

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoyReport
sunny	Warm	Normal	Strong	Warm	Same	Yes
sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	change	No
Sunny	Warm	High	Strong	Cool	change	Yes

< program >

```
import pandas as pd

import numpy as np


# Load the data

data = pd.read_csv("my.csv")


# Convert the data to a numpy array

d = np.asarray(data)

print(d)


# Initialize the hypothesis with 'NULL' values

h = ['NULL', 'NULL', 'NULL', 'NULL', 'NULL', 'NULL']


# Create a list to store the matching instances
```

```
m = []

# Find the instances that match the positive target concept 'EnjoyReport = Yes'
for i in range(len(d)):
    if d[i][-1] == 'Yes':
        m.append(d[i])

# Iterate through the matching instances to find the most specific hypothesis
for i in range(len(m)):
    # For each attribute, check if the hypothesis matches the instance attribute
    # If they match, continue to the next attribute
    # If the hypothesis has 'NULL' for that attribute, update it to match the instance
    # If the hypothesis already has a value for that attribute and it doesn't match the instance, set it to '?'
    for j in range(6):
        if h[j] == m[i][j]:
            pass
        elif h[j] == 'NULL':
            h[j] = m[i][j]
        else:
            h[j] = '?'

    # Print the updated hypothesis for each instance
    print(h)
```

< Output >

```
[['sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same' 'Yes']
 ['sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same' 'Yes']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'change' 'No']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'change' 'Yes']]
['sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
['sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
['?', 'Warm', '?', 'Strong', '?', '?']
```

In [ ]:

Experiments (Machine Learning)

AIM :**Develop a program to make Linear Regression Model ?**

Data Set : **Attendence.csv**

S.NO	Attendenc	Marks
1	60	63
2	66	64
3	67	65
4	68	66
5	69	67
6	70	68
7	71	69
8	72	70
9	73	71
10	74	72
11	75	73
12	76	74
13	77	75
14	78	76
15	79	77
16	80	78
17	81	79
18	82	80
19	83	81
20	84	82

Program :

```
# read the data

import pandas as pd

df= pd.read_csv("Attendence.csv")

print(df)

#making independent and dependent variables

#linear

X = df[['Attendence']]
```

```
y = df['Marks']

print(X)

print(X.shape)

print(y)

print(y.shape)

# dividing the data into testing and training data set

# training = 80 % testing = 20%

#import the module

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) # 0.2 = 20%

print(X_train)

print(X_train.shape)

print(X_test)

print(X_test.shape)

print(y_train)

print(y_test)

#making the linear regression model

#fit is used to train the machine

# First import the linear regression

from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(X_train,y_train)

# predicting value by the machine by our train data

result = model.predict([[89]]) # input

print(result)
```

 # predictint the values of all test data

```
prediction = model.predict(X_test) # input # pridicting the output by model with training data

print(prediction)
```

 # checking the machine how accurately the machine giving the output

 # by r2_score

 # by comparing the predicted values and y_test # output

#first import

```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score

predit_r2_score = r2_score(y_test,prediction)

print(predit_r2_score) # 1.0 = 100%
```

 # predicting the actual and predicted value is same.. 0.0 = same

```
print("Mean Squared Error : " , mean_squared_error(y_test,prediction))

data = pd.DataFrame({'Actual ML_marks :': y_test , 'predicted_Values by AI :': prediction})

print(data)
```

Output :

S.NO	Attendence	Marks	
0	1	60	63
1	2	66	64
2	3	67	65
3	4	68	66
4	5	69	67
5	6	70	68
6	7	71	69
7	8	72	70
8	9	73	71
9	10	74	72
10	11	75	73
11	12	76	74
12	13	77	75
13	14	78	76
14	15	79	77
15	16	80	78
16	17	81	79

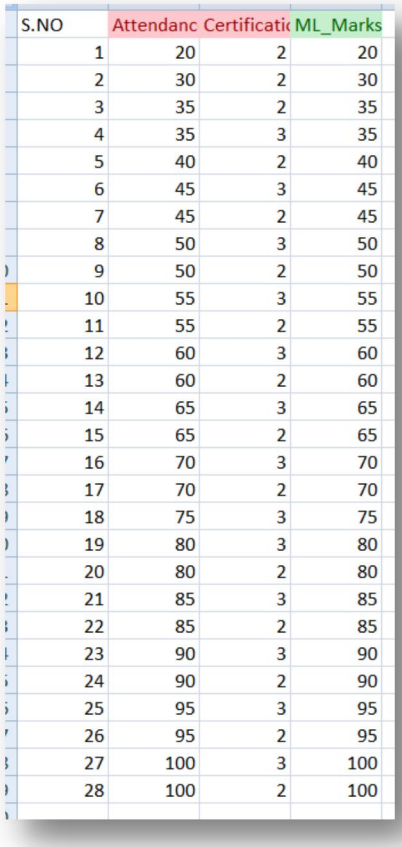
```
17    18          82     80
18    19          83     81
19    20          84     82
    Attendence
0           60
1           66
2           67
3           68
4           69
5           70
6           71
7           72
8           73
9           74
10          75
11          76
12          77
13          78
14          79
15          80
16          81
17          82
18          83
19          84
(20, 1)
0     63
1     64
2     65
3     66
4     67
5     68
6     69
7     70
8     71
9     72
10    73
11    74
12    75
13    76
14    77
15    78
16    79
17    80
18    81
```

```
19    82
Name: Marks, dtype: int64
(20,)
    Attendence
0           60
1           66
15          80
13          78
2           67
17          82
7           72
3           68
10          75
6           71
9           74
4           69
19          84
14          79
8           73
12          77
(16, 1)
    Attendence
11          76
18          83
16          81
5           70
(4, 1)
0     63
1     64
15    78
13    76
2     65
17    80
7     70
3     66
10    73
6     69
9     72
4     67
19    82
14    77
8     71
12    75
Name: Marks, dtype: int64
```

```
11    74
18    81
16    79
5     68
Name: Marks, dtype: int64
[85.65264411]
[74.0391904  80.29258855 78.50590336 68.67913483]
0.9880463123148753
Mean Squared Error :  0.30183061404939876
    Actual ML_marks :  predicted_Values by AI :
11                  74                  74.039190
18                  81                  80.292589
16                  79                  78.505903
5                   68                  68.679135
```

AIM :**Develop a program to make Multi Linear Regression Model ?**

Data Set : **ml_data.csv**



S.NO	Attendanc	Certificati	ML_Marks
1	20	2	20
2	30	2	30
3	35	2	35
4	35	3	35
5	40	2	40
6	45	3	45
7	45	2	45
8	50	3	50
9	50	2	50
10	55	3	55
11	55	2	55
12	60	3	60
13	60	2	60
14	65	3	65
15	65	2	65
16	70	3	70
17	70	2	70
18	75	3	75
19	80	3	80
20	80	2	80
21	85	3	85
22	85	2	85
23	90	3	90
24	90	2	90
25	95	3	95
26	95	2	95
27	100	3	100
28	100	2	100

< Program >

```
import pandas as pd

df = pd.read_csv('ml_data.csv')

print(df)

# dividing the dependent and indepedent variables

#here, the multi linear have two inputs

X = df[['Attendance of Ml students ', 'Certifications based on ML']] # input

y = df['ML_Marks'] #output

print(X)

print(X.shape)

print(y)

print(y.shape)

#spliting the data into training and testing = training = 80% , testing  = 20%

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)

print(X_train)

print(X_train.shape)

print(X_test)

print(X_test.shape)

print(y_train)

print(y_train.shape)

print(y_test)

print(y_test.shape)

#make the multilinear model

from sklearn.linear_model import LinearRegression

model = LinearRegression()
```

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

#predict the output [marks] from inputs of attendence and certifications

```
result = model.predict([[45,4]])

print(result)

predicted_result = model.predict(X_test)

print(predicted_result)

from sklearn.metrics import r2_score,mean_squared_error

predicted_r2_score = r2_score(y_test,predictes_result)

print(predicted_r2_score)

print("Mean Squared Error : " , mean_squared_error(y_test,predicted_result))

data = pd.DataFrame({'Actual ML_marks :': y_test , 'predicted_Values by AI :': predicted_result})

print(data)
```

< output >

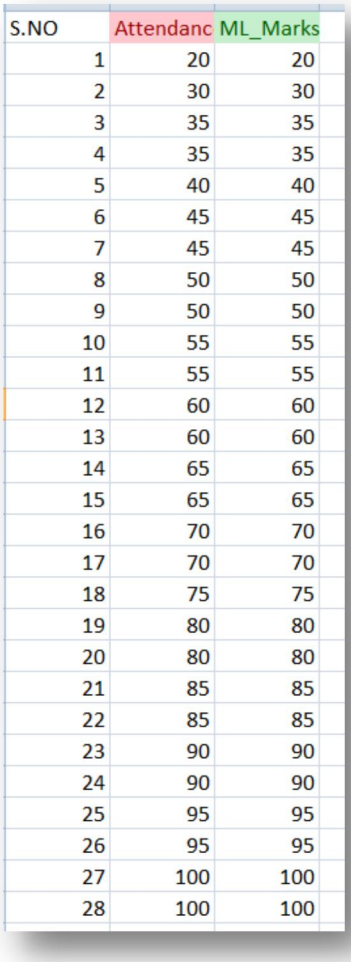
```
S.NO  Attendance of Ml students   Certifications based on ML  ML_Marks
0      1                          20                          2        20
1      2                          30                          2        30
2      3                          35                          2        35
3      4                          35                          3        35
4      5                          40                          2        40
5      6                          45                          3        45
6      7                          45                          2        45
7      8                          50                          3        50
8      9                          50                          2        50
9     10                          55                          3        55
10    11                          55                          2        55
11    12                          60                          3        60
12    13                          60                          2        60
13    14                          65                          3        65
14    15                          65                          2        65
15    16                          70                          3        70
16    17                          70                          2        70
17    18                          75                          3        75
18    19                          80                          3        80
19    20                          80                          2        80
20    21                          85                          3        85
21    22                          85                          2        85
22    23                          90                          3        90
23    24                          90                          2        90
24    25                          95                          3        95
25    26                          95                          2        95
26    27                         100                          3       100
27    28                         100                          2       100
    Attendance of Ml students   Certifications based on ML
0                           20                           2
1                           30                           2
2                           35                           2
3                           35                           3
4                           40                           2
5                           45                           3
```

```
6                            45                            2
7                            50                            3
8                            50                            2
9                            55                            3
10                           55                            2
11                           60                            3
12                           60                            2
13                           65                            3
14                           65                            2
15                           70                            3
16                           70                            2
17                           75                            3
18                           80                            3
19                           80                            2
20                           85                            3
21                           85                            2
22                           90                            3
23                           90                            2
24                           95                            3
25                           95                            2
26                          100                            3
27                          100                            2
(28, 2)
0      20
1      30
2      35
3      35
4      40
5      45
6      45
7      50
8      50
9      55
10     55
11     60
12     60
13     65
14     65
15     70
16     70
17     75
18     80
19     80
20     85
21     85
22     90
23     90
24     95
25     95
26    100
27    100
Name: ML_Marks, dtype: int64
(28,)
    Attendance of Ml students   Certifications based on ML
13                           65                            3
9                            55                            3
3                            35                            3
27                          100                            2
18                           80                            3
22                           90                            3
15                           70                            3
8                            50                            2
25                           95                            2
24                           95                            3
21                           85                            2
14                           65                            2
2                            35                            2
19                           80                            2
10                           55                            2
11                           60                            3
26                          100                            3
```

```
5                           45                              3
1                           30                              2
16                          70                              2
12                          60                              2
4                           40                              2
(22, 2)
    Attendance of Ml students   Certifications based on ML
23                          90                              2
17                          75                              3
0                           20                              2
6                           45                              2
7                           50                              3
20                          85                              3
(6, 2)
13     65
9      55
3      35
27    100
18     80
22     90
15     70
8      50
25     95
24     95
21     85
14     65
2      35
19     80
10     55
11     60
26    100
5      45
1      30
16     70
12     60
4      40
Name: ML_Marks, dtype: int64
(22,)
23    90
17    75
0     20
6     45
7     50
20    85
Name: ML_Marks, dtype: int64
(6,)
[45.]
[90. 75. 20. 45. 50. 85.]
1.0
Mean Squared Error :  1.682903264471492e-28
    Actual ML_marks :  predicted_Values by AI :
23                 90                       90.0
17                 75                       75.0
0                  20                       20.0
6                  45                       45.0
7                  50                       50.0
20                 85                       85.0
```

AIM :**Develop a program to make Polynomial Regression Model ?**

Data Set : **Marks.csv**



S.NO	Attendanc	ML_Marks
1	20	20
2	30	30
3	35	35
4	35	35
5	40	40
6	45	45
7	45	45
8	50	50
9	50	50
10	55	55
11	55	55
12	60	60
13	60	60
14	65	65
15	65	65
16	70	70
17	70	70
18	75	75
19	80	80
20	80	80
21	85	85
22	85	85
23	90	90
24	90	90
25	95	95
26	95	95
27	100	100
28	100	100

< program >

```
import pandas as pd

df = pd.read_csv('Marks.csv')

print(df)
```

#making the independent and dependent varables = input and output varibles

 # the polynomial regression is same as linear regression

```
X = df[['Attendance of MI students ']]

y = df['ML_Marks']

 print(X)

print(X.shape)

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)

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear_model import LinearRegression


# create polynomial features

poly = PolynomialFeatures(degree=2)

#trasform is used to transform the the train data into all polynomial combinations
  upto specified degree = 2 and fit

X_train_poly = poly.fit_transform(X_train)

#trasform is used to transform the the test data into all polynomial combinations
  upto specified degree = 2

X_test_poly = poly.transform(X_test)


# fit polynomial regression model

model = LinearRegression()

model.fit(X_train_poly, y_train)


# make predictions on testing data

y_pred = model.predict(X_test_poly)

print(y_pred)

print(poly) # polynomialfeatures
```

```
print(X_train_poly) # trained polynomials

print(X_test_poly)

print("m=", model.coef_)

print("c=",model.intercept_)


from sklearn.metrics import r2_score, mean_squared_error

# make predictions on testing data

y_pred = model.predict(X_test_poly)

print(y_pred)

#r2_score value of y_test and predit

print("R-squared score on testing data: ", r2_score(y_test, y_pred))

#mean squared error

print("Mean squared error on testing data: ", mean_squared_error(y_test,y_pred))

# make prediction for new instance

new_instance = [[32]]  # assuming there is only one feature

new_instance_poly = poly.transform(new_instance)

prediction = model.predict(new_instance_poly) # we only predict the 3 inputs bacause the degree=2


print("Prediction for new instance: ", prediction)

#y_test(output)and x_test(input) predicted values of polynomial  is same or not
  ( always prediction = x_test == x_test_poly(degree=2))

data = pd.DataFrame({'Actual ML_marks :': y_test , 'predicted_Values by AI :': y_pred})

print(data)
```

< output >

```
S.NO  Attendance of Ml students   ML_Marks
0      1                          20        20
1      2                          30        30
2      3                          35        35
3      4                          35        35
4      5                          40        40
5      6                          45        45
6      7                          45        45
7      8                          50        50
8      9                          50        50
9     10                          55        55
10    11                          55        55
11    12                          60        60
12    13                          60        60
13    14                          65        65
14    15                          65        65
15    16                          70        70
16    17                          70        70
17    18                          75        75
18    19                          80        80
19    20                          80        80
20    21                          85        85
21    22                          85        85
22    23                          90        90
23    24                          90        90
24    25                          95        95
25    26                          95        95
26    27                         100       100
27    28                         100       100
    Attendance of Ml students
0                           20
1                           30
2                           35
3                           35
4                           40
5                           45
6                           45
7                           50
8                           50
9                           55
10                          55
11                          60
12                          60
13                          65
14                          65
15                          70
16                          70
17                          75
18                          80
19                          80
20                          85
21                          85
22                          90
23                          90
24                          95
25                          95
26                         100
27                         100
(28, 1)
[35. 50. 60. 70. 80. 65.]
PolynomialFeatures()
[[1.000e+00 5.500e+01 3.025e+03]
 [1.000e+00 1.000e+02 1.000e+04]
 [1.000e+00 4.000e+01 1.600e+03]
 [1.000e+00 7.500e+01 5.625e+03]
 [1.000e+00 5.000e+01 2.500e+03]
 [1.000e+00 8.000e+01 6.400e+03]
 [1.000e+00 9.000e+01 8.100e+03]
```

```
 [1.000e+00 6.000e+01 3.600e+03]
 [1.000e+00 9.500e+01 9.025e+03]
 [1.000e+00 3.500e+01 1.225e+03]
 [1.000e+00 4.500e+01 2.025e+03]
 [1.000e+00 4.500e+01 2.025e+03]
 [1.000e+00 9.000e+01 8.100e+03]
 [1.000e+00 3.000e+01 9.000e+02]
 [1.000e+00 8.500e+01 7.225e+03]
 [1.000e+00 7.000e+01 4.900e+03]
 [1.000e+00 2.000e+01 4.000e+02]
 [1.000e+00 8.500e+01 7.225e+03]
 [1.000e+00 1.000e+02 1.000e+04]
 [1.000e+00 6.500e+01 4.225e+03]
 [1.000e+00 5.500e+01 3.025e+03]
 [1.000e+00 9.500e+01 9.025e+03]]
[[1.000e+00 3.500e+01 1.225e+03]
 [1.000e+00 5.000e+01 2.500e+03]
 [1.000e+00 6.000e+01 3.600e+03]
 [1.000e+00 7.000e+01 4.900e+03]
 [1.000e+00 8.000e+01 6.400e+03]
 [1.000e+00 6.500e+01 4.225e+03]]
m= [ 0.00000000e+00  1.00000000e+00 -6.11661943e-17]
c= -5.684341886080802e-14
[35. 50. 60. 70. 80. 65.]
R-squared score on testing data:  1.0
Mean squared error on testing data:  2.2298468254247266e-27
Prediction for new instance:  [32.]
    Actual ML_marks :  predicted_Values by AI :
2                  35                       35.0
8                  50                       50.0
12                 60                       60.0
15                 70                       70.0
19                 80                       80.0
14                 65                       65.0
```

AIM :**Develop a program to make logistic Regression Model ?**

Data Set : **diabetes.csv**

Pregnancie	Glucose	BloodPressure	SkinThickn	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0
7	196	90	0	0	39.8	0.451	41	1
9	119	80	35	0	29	0.263	29	1
11	143	94	33	146	36.6	0.254	51	1
10	125	70	26	115	31.1	0.205	41	1
7	147	76	0	0	39.4	0.257	43	1
1	97	66	15	140	23.2	0.487	22	0
13	145	82	19	110	22.2	0.245	57	0
5	117	92	0	0	34.1	0.337	38	0

This code performs the following steps:

1. Loads the diabetes dataset and checks for missing values
2. Separates the independent and dependent variables
3. Splits the dataset into training and test sets
4. Fits a logistic regression model on the training set
5. Predicts on the test set and creates a confusion matrix heatmap
6. Calculates evaluation metrics (accuracy, precision, recall, and F1 score)
7. Prompts the user to input new data to make a prediction on
8. Makes a prediction on the new data using the trained model and prints the result

< program >

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
import seaborn as sns
import matplotlib.pyplot as plt

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

# Check for missing values
print(df.isnull().sum())
```

```
# Separate the independent and dependent variables
X = df[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']]
y = df['Outcome']

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=0.25, random_state=0)

# Fit a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create the heatmap using seaborn
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d")

# Set the axis labels and title
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
```

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

# Calculate the evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)

# Take input from the user to predict on new data
pregnancies = int(input("Enter the number of pregnancies: "))
glucose = int(input("Enter the glucose level: "))
blood_pressure = int(input("Enter the blood pressure: "))
skin_thickness = int(input("Enter the skin thickness: "))
insulin = int(input("Enter the insulin level: "))
bmi = float(input("Enter the BMI: "))
dpf = float(input("Enter the diabetes pedigree function: "))
age = int(input("Enter the age: "))
```

```
# Create a new row with the user's input
new_data = pd.DataFrame({
    'Pregnancies': [pregnancies],
    'Glucose': [glucose],
    'BloodPressure': [blood_pressure],
    'SkinThickness': [skin_thickness],
    'Insulin': [insulin],
    'BMI': [bmi],
    'DiabetesPedigreeFunction': [dpf],
    'Age': [age]
})

# Make a prediction on the new data using the trained model
prediction = model.predict(new_data)

# Print the prediction
if prediction[0] == 0:
    print("Prediction: Not Diabetic")
else:
    print("Prediction: Diabetic")
```

< Output >

```
Pregnancies                 0
```

```
Glucose                       0
BloodPressure                 0
SkinThickness                 0
Insulin                       0
BMI                           0
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
C:\Users\saichand.digumarthi\anaconda3\lib\site-packages\sklearn\linear_model\_logist
  ic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(

Accuracy: 0.7916666666666666
Precision: 0.7115384615384616
Recall: 0.5967741935483871
F1 Score: 0.6491228070175439
Enter the number of pregnancies: 2
Enter the glucose level: 34
Enter the blood pressure: 120
Enter the skin thickness: 67
Enter the insulin level: 55
Enter the BMI: 78
Enter the diabetes pedigree function: 45
Enter the age: 34
Prediction: Diabetic
```

Write a program to demonstrate the working of the decision tree classifier. Use appropriate datasetfor building the decision tree and apply this knowledge to classify a new sample.

AIM : **Develop a program to make Decision tree classifier Model ?**

Data Set : **diabetes2.csv**

Pregnancie	Glucose	BloodPressure	SkinThickn	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0
7	196	90	0	0	39.8	0.451	41	1
9	119	80	35	0	29	0.263	29	1
11	143	94	33	146	36.6	0.254	51	1
10	125	70	26	115	31.1	0.205	41	1
7	147	76	0	0	39.4	0.257	43	1
1	97	66	15	140	23.2	0.487	22	0
13	145	82	19	110	22.2	0.245	57	0
5	117	92	0	0	34.1	0.337	38	0

< program >

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

 # load the dataset

```
df=pd.read_csv("diabetes2.csv")
```

 # display the dataset

```
df
```

 # generate descriptive statistics of the dataset

```
df.describe()
```

```
# check for missing values in the dataset
print(df.isnull().sum())

# compute the correlation matrix of the dataset
df.corr()

# separate the input features (X) and the target variable (y)
X=df.drop("Outcome",axis=1)
y=df[['Outcome']]

# split the dataset into training and testing sets
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)

# print the shapes of the datasets
print(X.shape)
print(y.shape)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

# perform feature scaling on the training and testing sets
from sklearn.preprocessing import StandardScaler
st=StandardScaler()
X_train=st.fit_transform(X_train)
X_test=st.fit_transform(X_test)

# train a decision tree classifier on the training set
from sklearn.tree import DecisionTreeClassifier
classifier=DecisionTreeClassifier(criterion='entropy',random_state=0)
classifier.fit(X_train,y_train)

# make predictions on the testing set
y_pred=classifier.predict(X_test)

# compute the confusion matrix
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm

# compute the classification report and accuracy score
from sklearn.metrics import classification_report,accuracy_score
```

```
res=classification_report(y_test,y_pred)
print("Classification Report:")
print(res)
result=accuracy_score(y_test,y_pred)
print("Accuracy:",result)
```

 # compute the accuracy, specificity, sensitivity, and F1 score

```
from sklearn.metrics import accuracy_score,recall_score,f1_score
Accuracy=accuracy_score(y_test,y_pred)
Specificity = cm[0,0]/(cm[0,0]+cm[0,1])
Sensitivity_recall = recall_score(y_test, y_pred)
F1_score = f1_score(y_test,y_pred)

print("Accuracy:",Accuracy)
print("Specificity:",Specificity)
print("Sensitivity/Recall:",Sensitivity_recall)
print("F1 Score:",F1_score)
```

 # compute the mean squared error

```
from sklearn.metrics import mean_squared_error
res1=mean_squared_error(y_test,y_pred)
res1
```

 # compute the confusion matrix, classification report, and accuracy score

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred))
```

 # compute the R2 score

```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
predit_r2_score = r2_score(y_test,y_pred)
print(predit_r2_score) # 1.0 = 100%
```

< Output >

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
(768, 8)
(768, 1)
(576, 8)
(192, 8)
(576, 1)
(192, 1)
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.79      0.82       130
           1       0.61      0.69      0.65        62

    accuracy                           0.76       192
   macro avg       0.73      0.74      0.73       192
weighted avg       0.77      0.76      0.76       192

Accuracy: 0.7604166666666666
Accuracy: 0.7604166666666666
Specificity: 0.7923076923076923
Sensitivity/Recall: 0.6935483870967742
F1 Score: 0.6515151515151515
Confusion Matrix:
[[103  27]
 [ 19  43]]

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.79      0.82       130
           1       0.61      0.69      0.65        62

    accuracy                           0.76       192
   macro avg       0.73      0.74      0.73       192
weighted avg       0.77      0.76      0.76       192


Accuracy Score:
0.7604166666666666
-0.09578163771712145
```

AIM : **Develop a program to make Decision tree regressor Model ?**

Data Set : **Attendence.csv**

S.NO	Attendenc	Marks
1	60	63
2	66	64
3	67	65
4	68	66
5	69	67
6	70	68
7	71	69
8	72	70
9	73	71
10	74	72
11	75	73
12	76	74
13	77	75
14	78	76
15	79	77
16	80	78
17	81	79
18	82	80
19	83	81
20	84	82

< program >

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Load the dataset
df = pd.read_csv('Attendence.csv')
```

```
print(df)

# Define input and output variables
X = df[['Attendence']] # Use 'Attendance' as input
y = df['Marks']

# Print shapes of input and output data
print(X.shape)
print(y.shape)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

# Print shapes of training and testing data
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

# Create a decision tree regressor with max_depth=3
tree = DecisionTreeRegressor(max_depth=3)

# Fit the model using the training data
tree.fit(X_train, y_train)
```

 # Predict on the test data

```
y_pred = tree.predict(X_test)

print(y_pred)
```

 # Evaluate the model using mean absolute error

```
mae = mean_absolute_error(y_test, y_pred)

print('Mean Absolute Error:', mae)
```

 # Evaluate the performance of the model using root mean squared error

```
mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

print('Root Mean Squared Error:', rmse)
```

< Output >

S.NO	Attendence	Marks	
0	1	60	63
1	2	66	64
2	3	67	65
3	4	68	66
4	5	69	67
5	6	70	68
6	7	71	69
7	8	72	70
8	9	73	71
9	10	74	72
10	11	75	73
11	12	76	74
12	13	77	75
13	14	78	76
14	15	79	77
15	16	80	78
16	17	81	79

```
17    18          82     80
18    19          83     81
19    20          84     82
(20, 1)
(20,)
(15, 1)
(5, 1)
(15,)
(5,)
[79.5 65.5 79.5 69.5 72. ]
Mean Absolute Error: 1.6
Root Mean Squared Error: 1.6733200530681511
```

AIM : **Write a program to demonstrate the working of Random Forest classifier. Use**

appropriate dataset for Random Forest Classifier.

Data Set : **diabetes2.csv**

Pregnancie	Glucose	BloodPressure	SkinThickn	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0
7	196	90	0	0	39.8	0.451	41	1
9	119	80	35	0	29	0.263	29	1
11	143	94	33	146	36.6	0.254	51	1
10	125	70	26	115	31.1	0.205	41	1
7	147	76	0	0	39.4	0.257	43	1
1	97	66	15	140	23.2	0.487	22	0
13	145	82	19	110	22.2	0.245	57	0
5	117	92	0	0	34.1	0.337	38	0

< program >

```
import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix


# Load the dataset

df = pd.read_csv("diabetes2.csv")
```

```
# Split the dataset into features and target
X = df.drop("Outcome", axis=1)
y = df["Outcome"]

# Split the dataset into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

# Create the random forest classifier and fit the model using the training data
classifier = RandomForestClassifier(n_estimators=100, random_state=0)
classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = classifier.predict(X_test)

# Print the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print("{:<10}{:<10}{}".format("", "Actual", ""))
print("{:<10}{:<10}{:<10}".format("", "Negative", "Positive"))
print("{:<10}{:<10}{:<10}".format("Predicted", "----------", "----------"))
print("{:<10}{:<10}{:<10}".format("Negative", cm[0][0], cm[0][1]))
print("{:<10}{:<10}{:<10}".format("Positive", cm[1][0], cm[1][1]))
```

```
# Print the classification report
from sklearn.metrics import classification_report
res = classification_report(y_test, y_pred)
print("\nClassification Report:\n", res)



import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create the heatmap using seaborn
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d")
#In this code snippet, we set the fmt parameter of seaborn's heatmap function to "d",
#which specifies that the numbers should be displayed as integers (i.e., no decimal places).
#This will display the confusion matrix heatmap with full numbers instead of in scientific
  notation.
# Set the axis labels and title
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
```

 # Show the plot

```
plt.show()
```

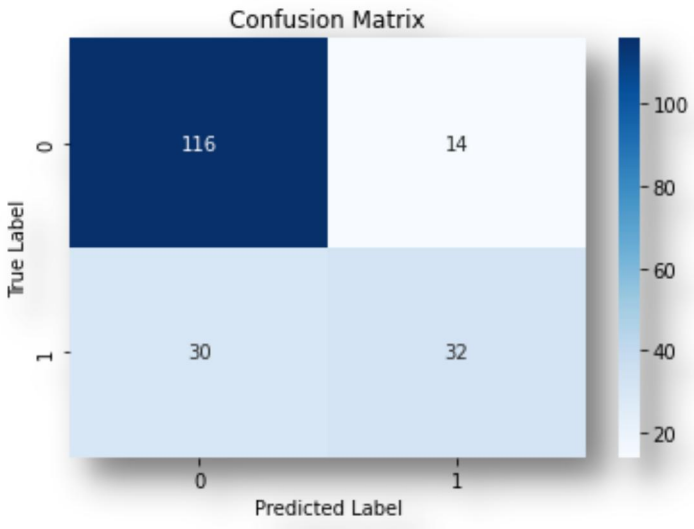
< Output >

```
Confusion Matrix:
          Actual
          Negative  Positive
Predicted --------------------
Negative  116       14
Positive  30        32

Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.89      0.84       130
           1       0.70      0.52      0.59        62

    accuracy                           0.77       192
   macro avg       0.75      0.70      0.72       192
weighted avg       0.76      0.77      0.76       192
```



AIM : **Implement and demonstrate FIND-S algorithm for finding the most specific hypothesis**

based on agiven set of training data samples. Read the training data from a .csv file.

Data Set : **my.csv**

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoyReport
sunny	Warm	Normal	Strong	Warm	Same	Yes
sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	change	No
Sunny	Warm	High	Strong	Cool	change	Yes

< program >

```
import pandas as pd

import numpy as np


# Load the data

data = pd.read_csv("my.csv")


# Convert the data to a numpy array

d = np.asarray(data)

print(d)


# Initialize the hypothesis with 'NULL' values

h = ['NULL', 'NULL', 'NULL', 'NULL', 'NULL', 'NULL']


# Create a list to store the matching instances

m = []
```

```
# Find the instances that match the positive target concept 'EnjoyReport = Yes'
for i in range(len(d)):
    if d[i][-1] == 'Yes':
        m.append(d[i])

# Iterate through the matching instances to find the most specific hypothesis
for i in range(len(m)):
    # For each attribute, check if the hypothesis matches the instance attribute
    # If they match, continue to the next attribute
    # If the hypothesis has 'NULL' for that attribute, update it to match the instance
    # If the hypothesis already has a value for that attribute and it doesn't match the instance, set it to '?'
    for j in range(6):
        if h[j] == m[i][j]:
            pass
        elif h[j] == 'NULL':
            h[j] = m[i][j]
        else:
            h[j] = '?'

    # Print the updated hypothesis for each instance
    print(h)
```

< Output >

```
[['sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same' 'Yes']
 ['sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same' 'Yes']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'change' 'No']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'change' 'Yes']]
['sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
['sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
['?', 'Warm', '?', 'Strong', '?', '?']
```

In [ ]: