LAKIREDDY BALI REDDYCOLLEGE OF ENGINEERING

(AUTONOMOUS)

Department of Computer Science& Engineering
(Artificial Intelligence and Machine Learning)

Introduction to Artificial Intelligence and Machine Learning Lab
(20AM51)

Name of the Student: _____

Registered Number: _____

Branch&Section: _____& ___ ../Sec

AcademicYear: 2022 -23

LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING

(AUTONOMOUS)

CERTIFICATE

Certificate that this is a bonafied record of the practical work done in _____Laboratory by _____ with Regd. No. _____ of _____B.Tech Course_____ Semester in _____Branch during the Academic Year2022-23

No.of Experiments held:_____

No.of Experiments Done: _____

Date: ..... /____/2023 **Signature of the Faculty**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

Introduction To Artificial Intelligence and Machine Learning Observation

AIM : Implement Linear and multi Linear Regression.

Introduction:

Linear Regression: Linear regression is a fundamental and widely used statistical technique in the field of artificial intelligence (AI) and machine learning. It is a supervised learning algorithm that aims to establish a linear relationship between a dependent variable and one or more independent variables. The goal of linear regression is to predict the value of the dependent variable based on the values of the independent variables.

In linear regression, the relationship between the variables is assumed to be linear, which means it can be represented by a straight line. The algorithm estimates the coefficients of the line that best fits the given data points, minimizing the overall difference between the predicted values and the actual values.

The basic form of a linear regression model can be expressed as:

$$Y = mx + c$$

where:

- Y is the dependent variable or the variable to be predicted.
- C is the intercept or the y-axis intercept of the line.
- M is the coefficient or the weight assigned to the independent variables x1, x2, ..., xn.
- x1, x2, ..., xn are the independent variables or the features used to predict the value of Y.

DATASET:

Dataset contains the features of Attendence and marks.

Where,

Attendence is considered as input attribute.

Marks is considered as target attribute.

CODE:

```
#Import Libraries.
import pandas as pd
#Load the Dataset.
df=pd.read_csv("D:/attendence.csv")
print(df)
```

OUTPUT:

	ATTENDENCE	MARKS
0	21	71
1	22	72
2	23	73
3	24	74
4	25	75
5	26	76
6	27	77
7	28	78
8	29	79
9	30	80
10	31	81
11	32	82
12	33	83
13	34	84
14	35	85
15	36	86
16	37	87
17	38	88
18	39	89
19	40	90

#Choose the dependent and independent attributes.

```
x=df[["ATTENDENCE"]]
y=df["MARKS"]
print(x)
print(x.shape)
print(y)
print(y.shape)
```

OUTPUT:

	ATTENDENCE
0	21
1	22
2	23
3	24

4 25
5 26
6 27
7 28
8 29
9 30
10 31
11 32
12 33
13 34
14 35
15 36
16 37
17 38
18 39
19 40
(20, 1)

0 71
1 72
2 73
3 74
4 75
5 76
6 77
7 78
8 79
9 80
10 81
11 82
12 83
13 84
14 85
15 86
16 87
17 88
18 89
19 90
Name: MARKS, dtype: int64

(20,)

#Split the dataset into training and testing

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

OUTPUT:

```
(14, 1)
(14,)
(6, 1)
(6,)
```

#Fit the linear regression model.

```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
result=model.predict(x_test)
print(result)
```

OUTPUT:

```
[86. 85. 89. 83. 75. 87.]
```

#Find the accuracy.

```
from sklearn.metrics import r2_score
r=r2_score(y_test,result)
print(r)
```

OUTPUT:

```
1.0
```

#Find the error rate.

```
from sklearn.metrics import mean_absolute_error
r1=mean_absolute_error(y_test,result)
print(r1)
```

OUTPUT:

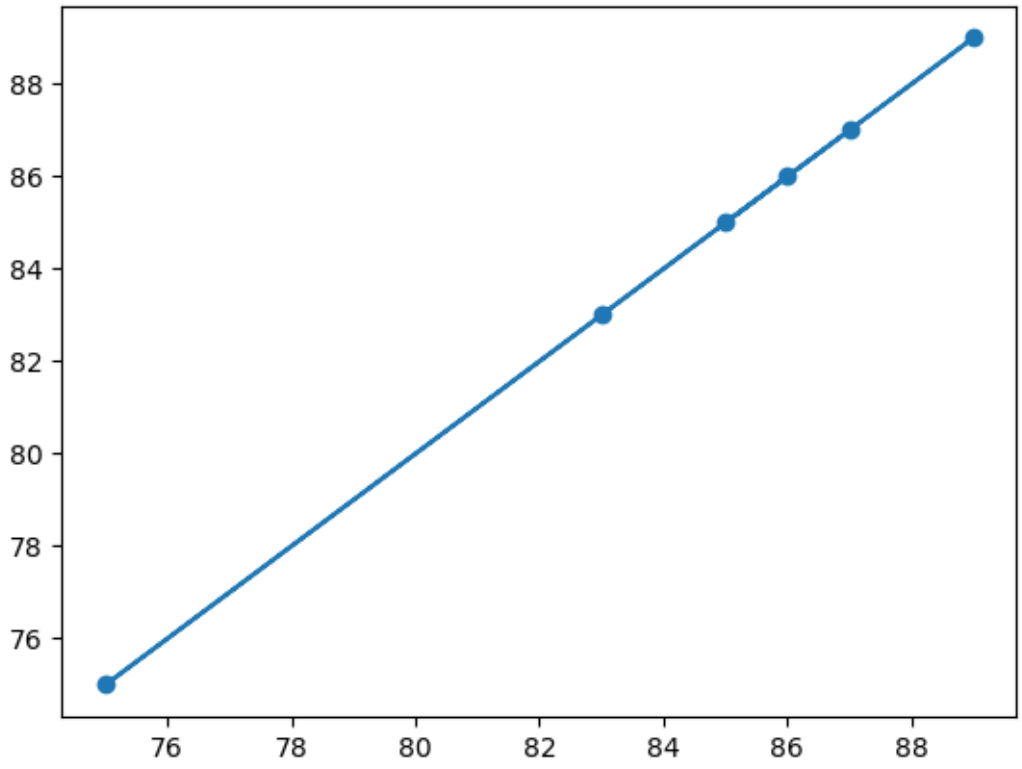
0.0

#Plotting

```
import matplotlib.pyplot as plt
plt.scatter(y_test,result)
plt.plot(y_test,result)
plt.show
```

OUTPUT:

<function matplotlib.pyplot.show(close=None, block=None)>



AIM : Implement Multi Linear Regression.

Introduction:

Multi linear regression:

multiple linear regression, is an extension of simple linear regression that allows for the prediction of a dependent variable based on multiple independent variables. In multivariate linear regression, the relationship between the dependent variable and the independent variables is assumed to be a linear combination.

The multivariate linear regression model can be expressed as:

y = C + m1*x1 + m2*x2 + ... + mn*xn

where:

- y is the dependent variable or the variable to be predicted.
- C is the intercept or the y-axis intercept of the line.
- m1, m2, ..., mn are the coefficients or the weights assigned to the independent variables x1, x2, ..., xn.
- x1, x2, ..., xn are the independent variables or the features used to predict the value of y.

The coefficients b0, b1, b2, ..., bn are estimated using the ordinary least squares (OLS) method, which minimizes the sum of the squared differences between the predicted values and the actual values.

In multivariate linear regression, each independent variable contributes to the prediction of the dependent variable, and the coefficients represent the change in the dependent variable associated with a unit change in the corresponding independent variable, holding all other variables constant.

The assumptions for multivariate linear regression are similar to those for simple linear regression:

1. Linearity: The relationship between the dependent variable and the independent variables is assumed to be linear.
2. Independence: The observations are assumed to be independent of each other.
3. Homoscedasticity: The variance of the errors is constant across all levels of the independent variables.
4. Normality: The errors are assumed to be normally distributed.

DATASET:

The dataset consists features like Attendence,No of certifictions,marks etc.
Attendence and No of certifications considered as input attribute.
Marks are considered as target attribute.

CODE:

#Import the required libraries.

```
import pandas as pd
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
#Load the dataset
df=pd.read_csv("Attendence.csv")
df
```

OUTPUT:

	ATTENDENCE	NO OF CERTIFICATIONS	MARKS
0	21	4	71
1	22	5	72
2	23	3	73
3	24	2	74
4	25	6	75
5	26	4	76
6	27	6	77
7	28	5	78
8	29	2	79
9	30	3	80
10	31	1	81
11	32	0	82
12	33	4	83
13	34	6	84
14	35	4	85
15	36	3	86
16	37	5	87
17	38	6	88
18	39	8	89
19	40	2	90

#Choose the dependent and independent attributes.

```
x=df[["ATTENDENCE","NO OF CERTIFICATIONS"]]
y=df["MARKS"]
print(x.shape)
print(y.shape)
```

OUTPUT:

```
(20, 2)
Name: MARKS, dtype: int64
(20,)
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

OUTPUT:

```
(14, 2)
(14,)
(6, 2)
(6,)
```

#Fit the linear regression model.

```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
result=model.predict(x_test)
print(result)
```

OUTPUT:

```
[86. 85. 89. 83. 75. 87.]
```

#Find the accuracy.

```
from sklearn.metrics import r2_score
r=r2_score(y_test,result)
print(r)
```

OUTPUT:

```
1.0
```

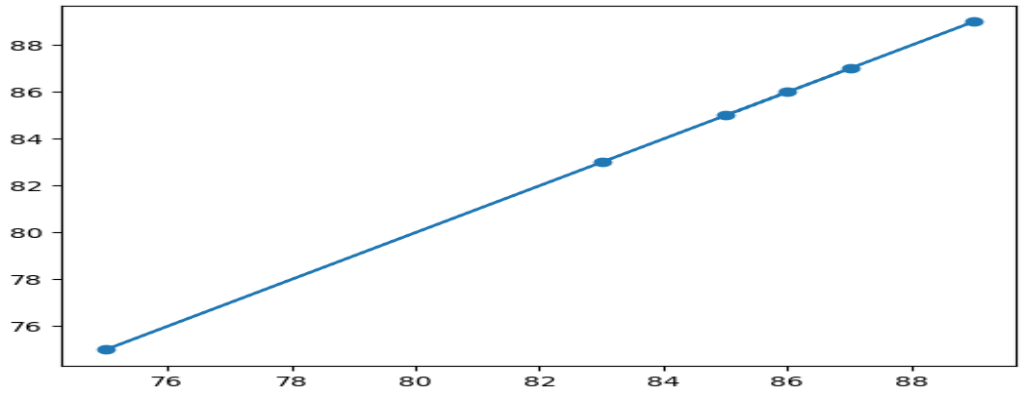
#Find the error rate.

```
from sklearn.metrics import mean_absolute_error
r1=mean_absolute_error(y_test,result)
print(r1)
```

OUTPUT:
0.0

#Plotting
import matplotlib.pyplot as plt
plt.scatter(y_test,result)
plt.plot(y_test,result)
plt.show

OUTPUT:
<function matplotlib.pyplot.show(close=None, block=None)>



AIM : Implement PolynomialRegression.

Introduction:

POLYNOMIAL REGRESSION:
Polynomial regression is a form of regression analysis used to model relationships between independent variables and a dependent variable. It is an extension of linear regression, where the relationship between variables is assumed to be linear. In polynomial regression, the relationship is modeled as a polynomial equation of a specified degree.
The polynomial equation is defined as:
$y = \beta_0 + \beta_1 x + \beta_2 x^2 + ... + \beta_n x^n$
where:

- y is the dependent variable,
- x is the independent variable,
- $\beta_0, \beta_1, \beta_2, ..., \beta_n$ are the coefficients to be estimated, and

- n is the degree of the polynomial.

Polynomial regression allows for more flexible modeling, as it can capture nonlinear relationships between variables. By including higher-order terms (x^2, x^3, etc.) in the equation, polynomial regression can accommodate curved patterns in the data.
Polynomial regression can be used in various fields, such as economics, finance, social sciences, and engineering, where the relationships between variables are not linear. It provides a flexible and interpretable approach to capturing and analyzing nonlinear relationships in the data.

DATASET: WeatherHistory datset.

CODE:

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

data=pd.read_csv("C:/Users/BHARGAVI/Downloads/archive/weatherHistory.csv")
data
```

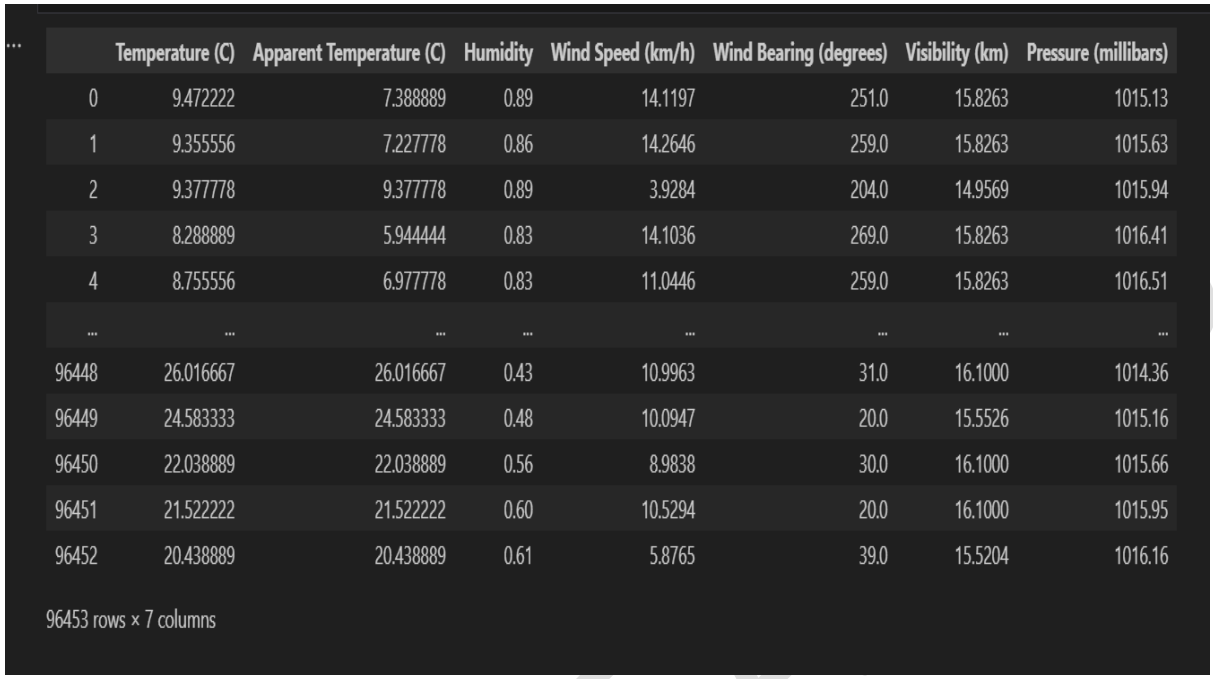
OUTPUT:

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Daily Summary
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	Partly cloudy throughout the day.
1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63	Partly cloudy throughout the day.
2	2006-04-01 02:00:00.000 +0200	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94	Partly cloudy throughout the day.
3	2006-04-01 03:00:00.000 +0200	Partly Cloudy	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41	Partly cloudy throughout the day.
4	2006-04-01 04:00:00.000 +0200	Mostly Cloudy	rain	8.755556	6.977778	0.83	11.0446	259.0	15.8263	0.0	1016.51	Partly cloudy throughout the day.
...	...	...	...	...	...	...	...	...	...	...	...	...
96448	2016-09-09 19:00:00.000 +0200	Partly Cloudy	rain	26.016667	26.016667	0.43	10.9963	31.0	16.1000	0.0	1014.36	Partly cloudy starting in the morning.
96449	2016-09-09 20:00:00.000 +0200	Partly Cloudy	rain	24.583333	24.583333	0.48	10.0947	20.0	15.5526	0.0	1015.16	Partly cloudy starting in the morning.
96450	2016-09-09 21:00:00.000 +0200	Partly Cloudy	rain	22.038889	22.038889	0.56	8.9838	30.0	16.1000	0.0	1015.66	Partly cloudy starting in the morning.
96451	2016-09-09 22:00:00.000 +0200	Partly Cloudy	rain	21.522222	21.522222	0.60	10.5294	20.0	16.1000	0.0	1015.95	Partly cloudy starting in the morning.
96452	2016-09-09 23:00:00.000 +0200	Partly Cloudy	rain	20.438889	20.438889	0.61	5.8765	39.0	15.5204	0.0	1016.16	Partly cloudy starting in the morning.

96453 rows × 12 columns

```
df=data.drop(['Loud Cover','Precip Type','Daily Summary','Summary','Formatted
Date'],axis=1)
df
```

OUTPUT:



	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Pressure (millibars)
0	9.472222	7.388889	0.89	14.1197	251.0	15.8263	1015.13
1	9.355556	7.227778	0.86	14.2646	259.0	15.8263	1015.63
2	9.377778	9.377778	0.89	3.9284	204.0	14.9569	1015.94
3	8.288889	5.944444	0.83	14.1036	269.0	15.8263	1016.41
4	8.755556	6.977778	0.83	11.0446	259.0	15.8263	1016.51
...	...	...	...	...	...	...	...
96448	26.016667	26.016667	0.43	10.9963	31.0	16.1000	1014.36
96449	24.583333	24.583333	0.48	10.0947	20.0	15.5526	1015.16
96450	22.038889	22.038889	0.56	8.9838	30.0	16.1000	1015.66
96451	21.522222	21.522222	0.60	10.5294	20.0	16.1000	1015.95
96452	20.438889	20.438889	0.61	5.8765	39.0	15.5204	1016.16

96453 rows × 7 columns

```
df.isnull().sum()
```

OUTPUT:

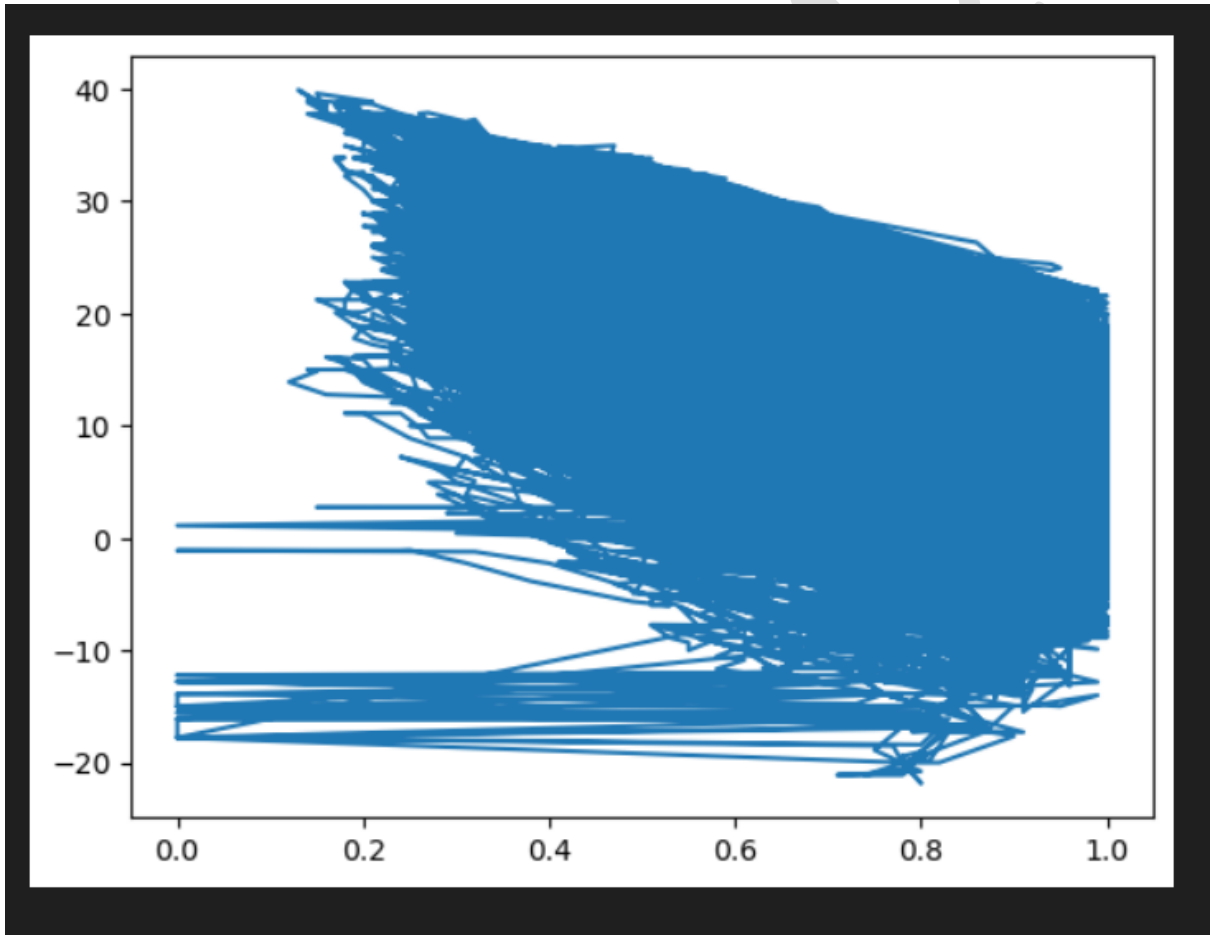
```
Temperature (C)           0
Apparent Temperature (C)  0
Humidity                  0
Wind Speed (km/h)         0
Wind Bearing (degrees)    0
Visibility (km)           0
Pressure (millibars)      0
dtype: int64
#Correlation between the attributes.
df.corr()
```

OUTPUT:

	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Pressure (millibars)
Temperature (C)	1.000000	0.992629	-0.632255	0.008957	0.029988	0.392847	-0.005447
Apparent Temperature (C)	0.992629	1.000000	-0.602571	-0.056650	0.029031	0.381718	-0.000219
Humidity	-0.632255	-0.602571	1.000000	-0.224951	0.000735	-0.369173	0.005454
Wind Speed (km/h)	0.008957	-0.056650	-0.224951	1.000000	0.103822	0.100749	-0.049263
Wind Bearing (degrees)	0.029988	0.029031	0.000735	0.103822	1.000000	0.047594	-0.011651
Visibility (km)	0.392847	0.381718	-0.369173	0.100749	0.047594	1.000000	0.059818
Pressure (millibars)	-0.005447	-0.000219	0.005454	-0.049263	-0.011651	0.059818	1.000000

```
x=data['Humidity']
y=data['Temperature (C)']
plt.plot(x,y)
```

OUTPUT:



```
x=df[["Humidity"]]
x.shape
```

OUTPUT:

```
(96453, 1)
```

```
y=df["Temperature (C)"]
y.shape
```

OUTPUT:

```
(96453,)
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.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

OUTPUT:

```
(77162, 1)
(19291, 1)
(77162,)
(19291,)
```

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
p=PolynomialFeatures(degree=3)
x_train_poly=p.fit_transform(x_train)
x_test_poly=p.fit_transform(x_test)
model=LinearRegression()
model.fit(x_train_poly,y_train)
```

OUTPUT:

```
LinearRegression()
result=model.predict(x_test_poly)
result
```

OUTPUT:

```
array([ 6.14273716, 20.57593483, 15.64422201, ..., 19.84655099,
        6.06683058,  5.96697777])
```

```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
acc=r2_score(y_test,result)
acc
```

OUTPUT:

```
0.932857446082616
import numpy as np
#pred = 0.86
predarray = np.array([[0.56]])
model.predict(p.fit_transform(predarray))
```

OUTPUT:

```
array([17.57593229])
```


AIM : Write a program to demonstrate the working of Logistic

Regression. Use appropriate dataset for Logistic Regression

Introduction:

Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class. It is used for classification algorithms its name is logistic regression. it's referred to as regression because it takes the output of the linear regression function as input and uses a sigmoid function to estimate the probability for the given class. The difference between linear regression and logistic regression is that linear regression output is the continuous value that can be anything while logistic regression predicts the probability that an instance belongs to a given class or not.

Terminologies involved in Logistic Regression:

Here are some common terms involved in logistic regression:

- **Independent variables:** The input characteristics or predictor factors applied to the dependent variable's predictions.

- **Dependent variable:** The target variable in a logistic regression model, which we are trying to predict.
- **Logistic function:** The formula used to represent how the independent and dependent variables relate to one another. The logistic function transforms the input variables into a probability value between 0 and 1, which represents the likelihood of the dependent variable being 1 or 0.
- **Odds:** It is the ratio of something occurring to something not occurring. it is different from probability as probability is the ratio of something occurring to everything that could possibly occur.
- **Log-odds:** The log-odds, also known as the logit function, is the natural logarithm of the odds. In logistic regression, the log odds of the dependent variable are modeled as a linear combination of the independent variables and the intercept.
- **Coefficient:** The logistic regression model's estimated parameters, show how the independent and dependent variables relate to one another.
- **Intercept:** A constant term in the logistic regression model, which represents the log odds when all independent variables are equal to zero.
- **Maximum likelihood estimation:** The method used to estimate the coefficients of the logistic regression model, which maximizes the likelihood of observing the data given the model.

Assumptions for Logistic regression

The assumptions for Logistic regression are as follows:

- **Independent observations:** Each observation is independent of the other. meaning there is no correlation between any input variables.
- **Binary dependent variables:** It takes the assumption that the dependent variable must be binary or dichotomous, meaning it can take only two values. For more than two categories softmax functions are used.
- **Linearity relationship between independent variables and log odds:** The relationship between the independent variables and the log odds of the dependent variable should be linear.
- **No outliers:** There should be no outliers in the dataset.
- **Large sample size:** The sample size is sufficiently large

Applying steps in logistic regression modeling:

The following are the steps involved in logistic regression modeling:

- **Define the problem:** Identify the dependent variable and independent variables and determine if the problem is a binary classification problem.
- **Data preparation:** Clean and preprocess the data, and make sure the data is suitable for logistic regression modeling.
- **Exploratory Data Analysis (EDA):** Visualize the relationships between the dependent and independent variables, and identify any outliers or anomalies in the data.
- **Feature selection:** Choose the independent variables that have a significant relationship with the dependent variable, and remove any redundant or irrelevant features.
- **Model building:** Train the logistic regression model on the selected independent variables and estimate the coefficients of the model.
- **Model evaluation:** Evaluate the performance of the logistic regression model using appropriate metrics such as accuracy, precision, recall, F1-score, or AUC-ROC.
- **Model improvement:** Based on the results of the evaluation, fine-tune the model by adjusting the independent variables, adding new features, or using regularization techniques to reduce overfitting.
- **Model deployment:** Deploy the logistic regression model in a real-world scenario and make predictions on new data.

Data Set : diabetes2.csv

#Import Required Libraries

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

#Load the Data Set and Display

```
data=pd.read_csv(r"C:\Users\Surendra
Babu\Downloads\diabetes.csv")
data
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

#Correlation

data.corr()

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

#Check if there are any Null Values

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

Split the dataset into features and target

```
x=data.drop("Outcome",axis=1)
y=data["Outcome"]
```

 # Split the dataset into training and testing 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)
x_train.shape
```

```
y_train.shape
```

O/P:

```
(614,)
```

Create the Logistic Regression Model and fit the model using the training data

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(x_train, y_train)
```

O/P:

```
LogisticRegression()
```

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

O/P:

```
array([1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0,
0, 0, 1, 0],
      dtype=int64)
```

```
# Print the confusion matrix
```

```
from sklearn.metrics import confusion_matrix
c=confusion_matrix(y_test,y_pred)
c
```

O/P:

```
array([[87, 10],
       [33, 24]], dtype=int64)
```

Print the classification report

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

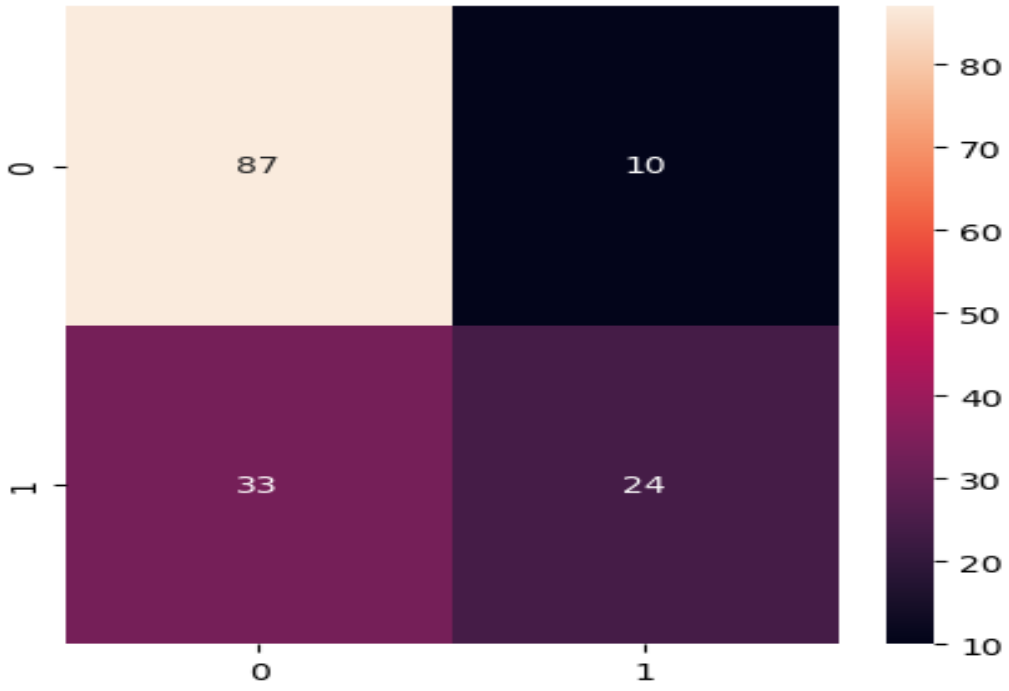
O/P:

	precision	recall	f1-score	support
0	0.72	0.90	0.80	97
1	0.71	0.42	0.53	57
accuracy			0.72	154
macro avg	0.72	0.66	0.66	154
weighted avg	0.72	0.72	0.70	154

In [13]:

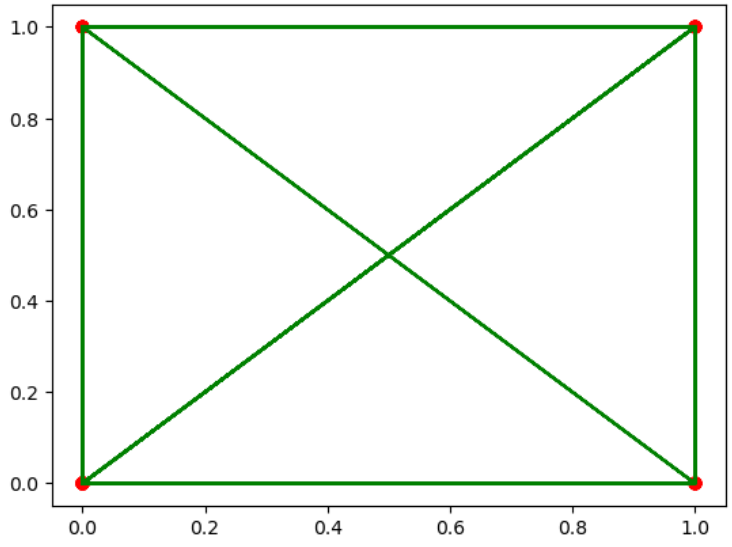
Create the heatmap using seaborn

```
import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix

sns.heatmap(c,annot=True,cmap="YlGnBu")

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



#Graph Plot

```
plt.scatter(y_test,y_pred,color='red')
plt.plot(y_test,y_pred,color='green')
```



#Evaluation Parameters

```
from sklearn.metrics import
precision_score,accuracy_score,f1_score,recall_score

print("precision_score : ",precision_score(y_test,y_pred))
```

O/P:

```
precision_score :  0.7058823529411765

print("accuracy_score : ",accuracy_score(y_test,y_pred))

O/P:
accuracy_score :  0.7207792207792207

print("f1_score",f1_score(y_test,y_pred))

O/P:
f1_score 0.5274725274725274

print("recall_score",recall_score(y_test,y_pred))

O/P:
recall_score 0.42105263157894735
```


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

Introduction

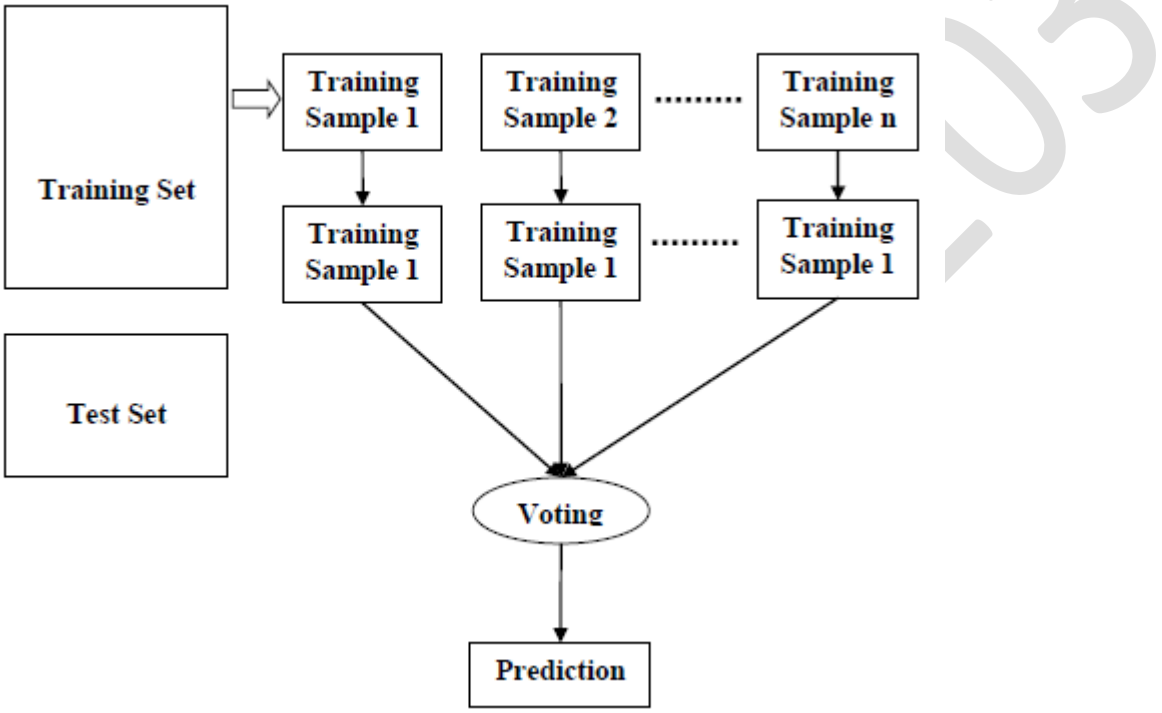
Random forest is a supervised learning algorithm which is used for both classification as well as regression. But however, it is mainly used for classification problems. As we know that a forest is made up of trees and more trees means more robust forest. Similarly, random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

Working of Random Forest Algorithm

We can understand the working of Random Forest algorithm with the help of following steps –

- **Step 1** − First, start with the selection of random samples from a given dataset.
- **Step 2** − Next, this algorithm will construct a decision tree for every sample. Then it will get the prediction result from every decision tree.
- **Step 3** − In this step, voting will be performed for every predicted result.
- **Step 4** − At last, select the most voted prediction result as the final prediction result.

The following diagram will illustrate its working −



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

< 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"]
```

#Correlation

```
data.corr()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

 # 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)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

O/P;

```
(576, 8)
(192, 8)
(576,)
(192,)
```

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)
```

O/P:

```
RandomForestClassifier()
```

Make predictions on the test data

```
  y_pred = classifier.predict(X_test)
print(y_pred)
```

```
[1 1 0 0 0 1 0 1 1 1 0 1 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 0 0 0
1 0 0 1 1 1 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0
0 0 0 0 0 0 1
 0 0 0 1 0 0 0 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0
1 0 0 0 1 0 1
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 1 1
0 0 0 0 1 0 0
 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 0 0
0 0 0 0 1 0 1
 1 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0
0 1 0 0 0 1 0
 0 0 0 1 0 0 0 1 0]
```

```
  # Print the confusion matrix cm =

  confusion_matrix(y_test, y_pred)

  print("Confusion Matrix:")


  cm
```

O/P:

Confusion Matrix:

Out[22]:

```
array([[115,  12],
       [ 31,  34]], dtype=int64)
```

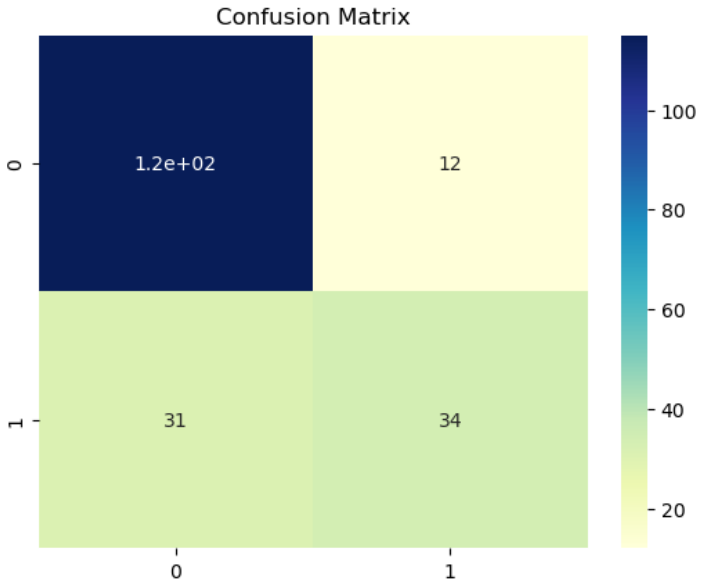
```
  # Print the classification report

  from sklearn.metrics import

  classification_report res =

  classification_report(y_test, y_pred)

  print("\nClassification Report:\n", res)
```

O/P:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.91      0.84       127
           1       0.74      0.52      0.61        65

    accuracy                           0.78       192
   macro avg       0.76      0.71      0.73       192
weighted avg       0.77      0.78      0.76       192
```

 # Generate the confusion matrix
 # Create the heatmap using seaborn

```
import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix

sns.heatmap(cm,annot=True,cmap="YlGnBu")

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



#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.

#Evaluation Parameters

from sklearn.metrics import precision_score,recall_score, f1_score, accuracy_score,confusion_matrix

```
print("Accuracy:", accuracy_score(y_test,y_pred))
```

```
Accuracy: 0.7760416666666666
```

```
print("Precision:",
precision_score(y_test,y_pred,average="weighted"))
```

```
Precision: 0.7712381501886044
```

```
print('Recall:',
recall_score(y_test,y_pred,average="weighted"))
```

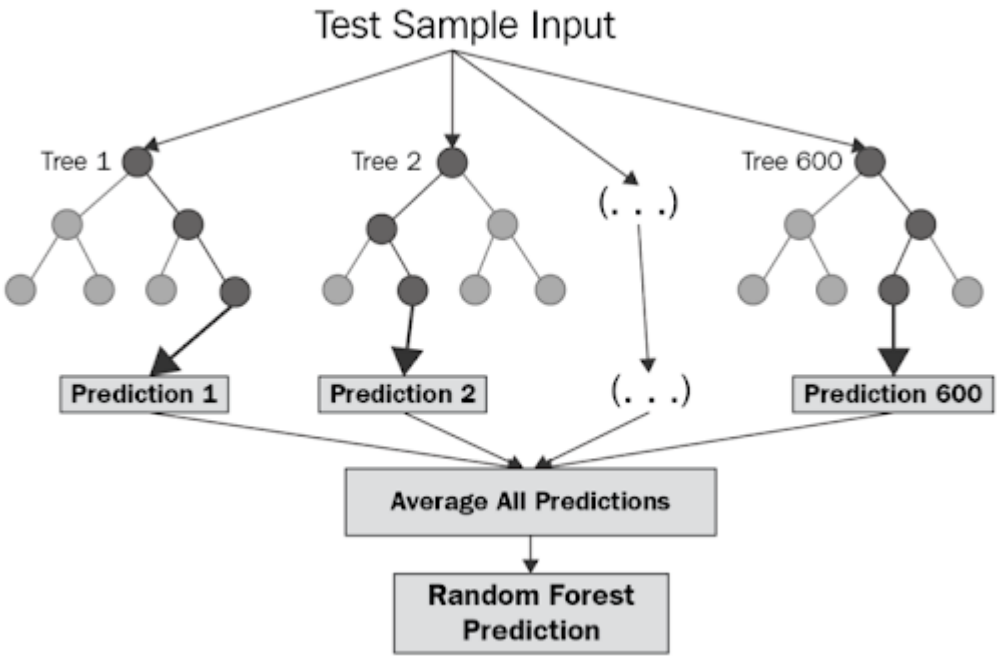
```
Recall: 0.77604
```

Aim: Write a program to demonstrate the working of Random Forest Regressor. Use appropriate dataset for Random Forest Regressor.

Random Forest Regressor:

Random forest regression is a supervised learning algorithm that uses an ensemble learning method for regression.

Random forest is a bagging technique and not a boosting technique. The trees in random forests run in parallel, meaning there is no interaction between these trees while building the trees.



Algorithm:

- Design a specific question or data and get the source to determine the required data.
- Make sure the data is in an accessible format else convert it to the required format.
- Specify all noticeable anomalies and missing data points that may be required to achieve the required data.
- Create a machine-learning model.
- Set the baseline model that you want to achieve

- Train the data machine learning model.
- Provide an insight into the model with test data
- Now compare the performance metrics of both the test data and the predicted data from the model.
- If it doesn't satisfy your expectations, you can try improving your model accordingly or dating your data, or using another data modeling technique.
- At this stage, you interpret the data you have gained and report accordingly.

Important Hyperparameters in Random Forest

Hyperparameters are used in random forests to either enhance the performance and predictive power of models or to make the model faster.

Hyperparameters to Increase the Predictive Power

n_estimators: Number of trees the algorithm builds before averaging the predictions.

max_features: Maximum number of features random forest considers splitting a node.

mini_sample_leaf: Determines the minimum number of leaves required to split an internal node.

criterion: How to split the node in each tree? (Entropy/Gini impurity/Log Loss)

max_leaf_nodes: Maximum leaf nodes in each tree

Hyperparameters to Increase the Speed

n_jobs: it tells the engine how many processors it is allowed to use. If the value is 1, it can use only one processor, but if the value is -1, there is no limit.

random_state: controls randomness of the sample. The model will always produce the same results if it has a definite value of random state and has been given the same hyperparameters and training data.

oob_score: *OOB* means out of the bag. It is a random forest cross-validation method. In this, one-third of the sample is not used to train the data; instead used to evaluate its performance. These samples are called out-of-bag samples.

Python code:

Step 1: Load Pandas library and the dataset using Pandas

Step 1: Load Pandas library and the dataset using Pandas

In [1]:
```
import pandas as pd
dataset = pd.read_csv('Cancer_data.csv')
dataset
dataset.head()
```

Out[1]:

	Radius_mean	Texture_mean	Perimeter_mean	Area_mean	Diagnosis
0	17.99	10.38	122.80	1001.0	1
1	20.57	17.77	132.90	1326.0	1
2	19.69	21.25	130.00	1203.0	1
3	11.42	20.38	77.58	386.1	1
4	20.29	14.34	135.10	1297.0	1

Step 2: Define the features and the target

In [2]:
```
X = pd.DataFrame(dataset.iloc[:,:-1])
y = pd.DataFrame(dataset.iloc[:,-1])
```

In [3]:
```
X
```


Step 2: Define the features and the target

In [2]:
```
X = pd.DataFrame(dataset.iloc[:,:-1])
y = pd.DataFrame(dataset.iloc[:,-1])
```

In [3]:
```
X
```

Out[3]:

	Radius_mean	Texture_mean	Perimeter_mean	Area_mean
0	17.990	10.38	122.80	1001.0
1	20.570	17.77	132.90	1326.0
2	19.690	21.25	130.00	1203.0
3	11.420	20.38	77.58	386.1
4	20.290	14.34	135.10	1297.0
5	12.450	15.70	82.57	477.1
6	18.250	19.98	119.60	1040.0
7	13.710	20.83	90.20	577.9
8	13.000	21.82	87.50	519.8
9	12.460	24.04	83.97	475.9
10	16.020	23.24	102.70	797.8
11	15.780	17.89	103.60	781.0
12	14.610	15.69	92.68	664.9

In [4]:
```
y
```

Out[4]:

	Diagnosis
0	1
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1
12	0

Step 3: Split the dataset into train and test sklearn

In [5]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

Step 4: Import the random forest classifier function from sklearn ensemble module. Build the random forest classifier model with the help of the random forest classifier function

```
In [25]: from sklearn.ensemble import RandomForestClassifier
         classifier = RandomForestClassifier(n_estimators=20, criterion='gini', random_state=1, max_depth=3)
         classifier.fit(X_train, y_train)
```

Step 5: Predict values using the random forest classifier model

```
In [19]: y_pred = classifier.predict(X_test)
```

Step 6: Evaluate the random forest classifier model

```
In [20]: from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
         print(confusion_matrix(y_test, y_pred))
         print(classification_report(y_test, y_pred))
         print(accuracy_score(y_test, y_pred))
```

```
[[10  2]
 [ 1  7]]
              precision    recall  f1-score   support

           0       0.91      0.83      0.87        12
           1       0.78      0.88      0.82         8

   micro avg       0.85      0.85      0.85        20
   macro avg       0.84      0.85      0.85        20
weighted avg       0.86      0.85      0.85        20

0.85
```

Feature Selection in Random Forest Algorithm Model

With the help of Scikit-Learn, we can select important features to build the random forest algorithm model in order to avoid the overfitting issue. There are two ways to do this:

- Visualize which feature is not adding any value to the model
- Take help of the built-in function **SelectFromModel**, which allows us to add a threshold value to neglect features below that threshold value.

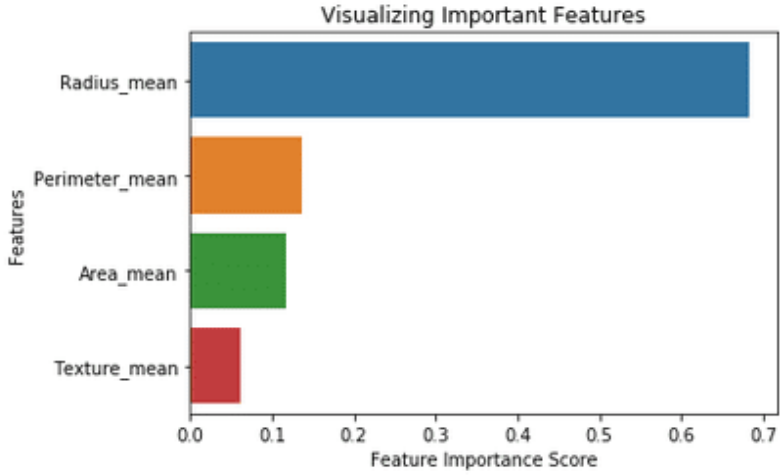
Let us see if selecting features make any difference in the accuracy score of the model.

Step 7: Let us find out important features and visualize them using Seaborn

```
In [18]: import pandas as pd
         feature_imp = pd.Series(classifier.feature_importances_,index=X.columns).sort_values(ascending=False)
         feature_imp
```

```
Out[18]: Radius_mean        0.658762
         Perimeter_mean     0.166428
         Area_mean          0.090985
         Texture_mean       0.083825
         dtype: float64
```

```
In [17]: import matplotlib.pyplot as plt
         import seaborn as sns
         %matplotlib inline
         #Creating a bar plot
         sns.barplot(x=feature_imp, y=feature_imp.index)
         plt.xlabel('Feature Importance Score')
         plt.ylabel('Features')
         plt.title("Visualizing Important Features")
         plt.show()
```



Now let us see how the 'SelectFromModel' function helps in building a random forest classifier model with important features.

Step 8: Import the SelectFromModel function. We will pass the classifier object we've created above. Also, we will add a threshold value of 0.1

```
In [43]: from sklearn.feature_selection import SelectFromModel
         feat_sel = SelectFromModel(classifier,threshold=0.1)
         feat_sel.fit(X_train, y_train)
```

```
Out[43]: SelectFromModel(estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=7, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=20, n_jobs=None,
                     oob_score=False, random_state=1, verbose=0, warm_start=False),
                 max_features=None, norm_order=1, prefit=False, threshold=0.1)
```

Step 9: With the help of the 'transform' method, we will pick the important features and store them in new train and test objects

```
In [39]: X_imp_train = feat_sel.transform(X_train)
         X_imp_test = feat_sel.transform(X_test)
```

Step 10: Let us now build a new random forest classifier model (so that we can compare the results of this model with the old one)

```
In [40]: clf_imp = RandomForestClassifier(n_estimators=20, criterion='gini', random_state=1, max_depth=7)
         clf_imp.fit(X_imp_train, y_train)

Out[40]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=7, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=20, n_jobs=None,
                     oob_score=False, random_state=1, verbose=0, warm_start=False)
```

Step 11: Let us see the accuracy result of the old model

```
In [41]: y_pred = classifier.predict(X_test)
         accuracy_score(y_test, y_pred)

Out[41]: 0.9
```

Step 12: Let us see the accuracy result of the new model after feature selection

```
In [42]: y_imp_pred = clf_imp.predict(X_imp_test)
         accuracy_score(y_test, y_imp_pred)

Out[42]: 0.85
```

Note: After the feature selection process, the accuracy score is decreased. But, we have successfully picked out the important features at a small cost of accuracy.

Also, automatic feature selection reduces the complexity of the model but does not necessarily increase the accuracy. In order to get the desired accuracy, we have to perform the feature selection process manually.

Step 13: To find Confusion Matrix:

```
y=confusion_matrix(y_test,y_pred)
y
```

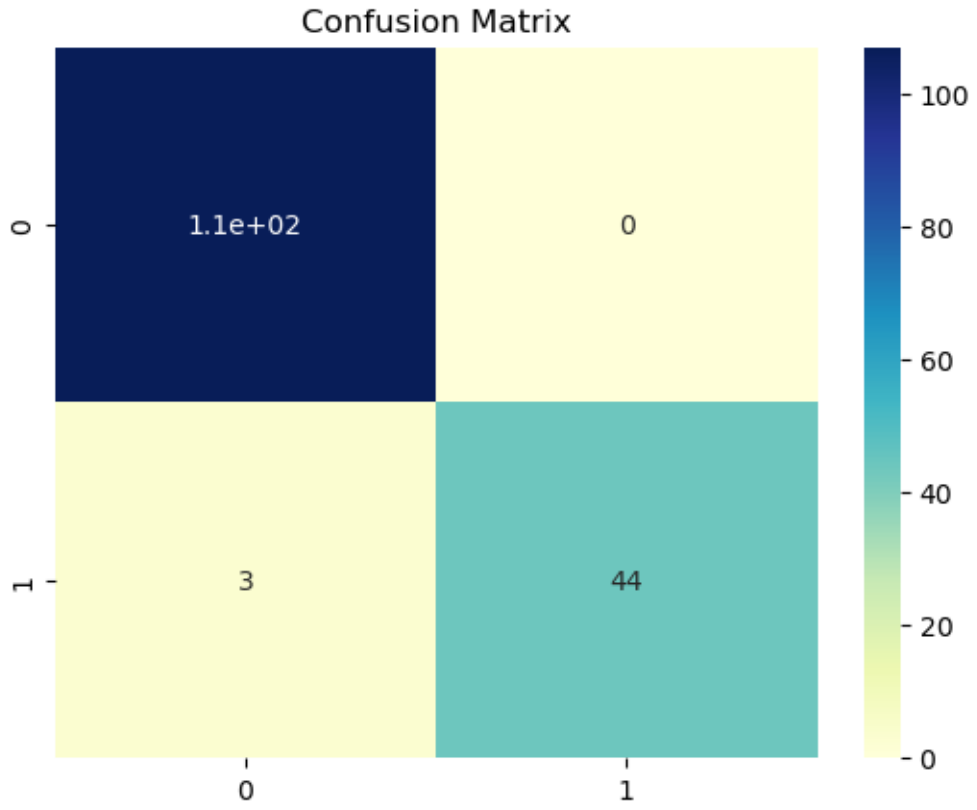
Out[71]:

```
array([[107,   0],
       [  3,  44]], dtype=int64)
```

```
import seaborn as sns
sns.heatmap(y,annot=True,cmap="YlGnBu")
plt.title("Confusion Matrix")
```

Out[72]:

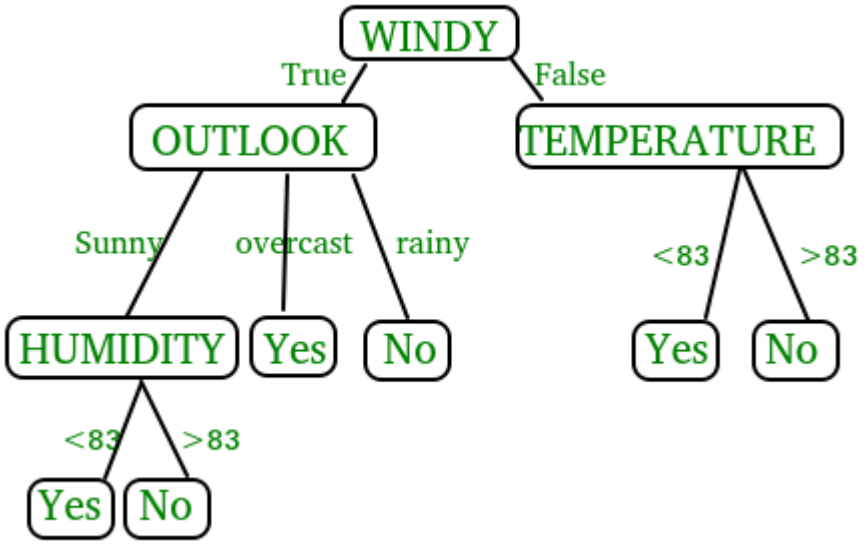
Text(0.5, 1.0, 'Confusion Matrix')



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

Description:-

Decision Tree is one of the most powerful and popular algorithm. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.



Pseudocode :

1. Find the best attribute and place it on the root node of the tree.
2. Now, split the training set of the dataset into subsets. While making the subset make sure that each subset of training dataset should have the same value for an attribute.
3. Find leaf nodes in all branches by repeating 1 and 2 on each subset.

While implementing the decision tree we will go through the following two phases:

1. Building Phase
    - Preprocess the dataset.
    - Split the dataset from train and test using Python sklearn package.

- Train the classifier.

2. Operational Phase
    - Make predictions.
    - Calculate the accuracy.

Syntax:-

class sklearn.tree.**DecisionTreeClassifier**(**, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0)*

Parameters:-

1. **criterion*{"gini", "entropy", "log_loss"}, default="gini"***
   The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "log_loss" and "entropy" both for the Shannon information gain, see Mathematical formulation.

2. **splitter*{"best", "random"}, default="best"***
   The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

3. **max_depth*int, default=None***
   The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

4. **min_samples_split*int or float, default=2***
   The minimum number of samples required to split an internal node:
    - If int, then consider min_samples_split as the minimum number.
    - If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

   Changed in version 0.18: Added float values for fractions.
5. **min_samples_leaf*int or float, default=1***
   The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

- If int, then consider min_samples_leaf as the minimum number.
- If float, then min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

Changed in version 0.18: Added float values for fractions.

6. **min_weight_fraction_leaf*float, default=0.0***
   The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

7. **max_features*int, float or {"auto", "sqrt", "log2"}, default=None***
   The number of features to consider when looking for the best split:

   - If int, then consider max_features features at each split.
   - If float, then max_features is a fraction and max(1, int(max_features * n_features_in_)) features are considered at each split.
   - If "auto", then max_features=sqrt(n_features).
   - If "sqrt", then max_features=sqrt(n_features).
   - If "log2", then max_features=log2(n_features).
   - If None, then max_features=n_features.

   Deprecated since version 1.1: The "auto" option was deprecated in 1.1 and will be removed in 1.3.

   Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max_features features.

8. **random_state*int, RandomState instance or None, default=None***
   Controls the randomness of the estimator. The features are always randomly permuted at each split, even if splitter is set to "best".
   When max_features < n_features, the algorithm will select max_features at random at each split before finding the best split among them. But the best found split may vary across different runs, even if max_features=n_features. That is the case, if the improvement of the criterion is identical for several splits and one split has to be selected at random. To obtain a deterministic behaviour during fitting, random_state has to be fixed to an integer. See Glossary for details.

9. **max_leaf_nodes*int, default=None***
   Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as

relative reduction in impurity. If None then unlimited number of leaf nodes.

10. **min_impurity_decrease*float, default=0.0***
    A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

    The weighted impurity decrease equation is the following:

    ```
    N_t / N * (impurity - N_t_R / N_t * right_impurity
                        - N_t_L / N_t * left_impurity)
    ```

    where N is the total number of samples, N_t is the number of samples at the current node, N_t_L is the number of samples in the left child, and N_t_R is the number of samples in the right child.

    N, N_t, N_t_R and N_t_L all refer to the weighted sum, if sample_weight is passed.

    New in version 0.19.
11. **class_weight*dict, list of dict or "balanced", default=None***
    Weights associated with classes in the form {class_label: weight}. If None, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.

    Note that for multioutput (including multilabel) weights should be defined for each class of every column in its own dict. For example, for four-class multilabel classification weights should be [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] instead of [{1:1}, {2:5}, {3:1}, {4:1}].

    The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data
    as n_samples / (n_classes * np.bincount(y))

    For multi-output, the weights of each column of y will be multiplied.

    Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified.

12. **ccp_alpha*non-negative float, default=0.0***
    Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed.

Info. About criterion:-

1. **Gini index:**

$$\text{Gini Index} = 1 - \sum_j j^2$$

- Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified.
- It means an attribute with lower gini index should be preferred.
- Sklearn supports “gini” criteria for Gini Index and by default, it takes “gini” value.

2. **Entropy:**

if a random variable x can take N different value , the i'value x_i with probability $p(x_{ii})$,we can associate the following entropy with x :

$$H(x) = -\sum_{i=1}^{N} p(x_i) \log_2 p(x_i)$$

Entropy is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples. The higher the entropy the more the information content.

3. **Information Gain:-**

Definition : SupposeSisasetofinstances,Aisanattribute,S_v is the sub-set of s with A = v and Values(A) is the set of all possible of A,then

- The entropy typically changes when we use a node in a decision tree to partition the training instances into smaller subsets. Information gain is a measure of this change in entropy.
- Sklearn supports “entropy” criteria for Information Gain and if we want to use Information Gain method in sklearn then we have to mention it explicitly.

Accuracy score

- Accuracy score is used to calculate the accuracy of the trained classifier.

Confusion Matrix

- Confusion Matrix is used to understand the trained classifier behavior over the test dataset or validate dataset.

Dataset:-Diabetes dataset is used in the Decision Tree Classifier.

	A	B	C	D	E	F	G	H	I	J
1	Pregnancie	Glucose	BloodPres	SkinThickn	Insulin	BMI	DiabetesP	Age	Outcome	
2	6	148	72	35	0	33.6	0.627	50	1	
3	1	85	66	29	0	26.6	0.351	31	0	
4	8	183	64	0	0	23.3	0.672	32	1	
5	1	89	66	23	94	28.1	0.167	21	0	
6	0	137	40	35	168	43.1	2.288	33	1	
7	5	116	74	0	0	25.6	0.201	30	0	
8	3	78	50	32	88	31	0.248	26	1	
9	10	115	0	0	0	35.3	0.134	29	0	
10	2	197	70	45	543	30.5	0.158	53	1	
11	8	125	96	0	0	0	0.232	54	1	
12	4	110	92	0	0	37.6	0.191	30	0	
13	10	168	74	0	0	38	0.537	34	1	
14	10	139	80	0	0	27.1	1.441	57	0	
15	1	189	60	23	846	30.1	0.398	59	1	
16	5	166	72	19	175	25.8	0.587	51	1	
17	7	100	0	0	0	30	0.484	32	1	
18	0	118	84	47	230	45.8	0.551	31	1	
19	7	107	74	0	0	29.6	0.254	31	1	
20	1	103	30	38	83	43.3	0.183	33	0	
21	1	115	70	30	96	34.6	0.529	32	1	
22	3	126	88	41	235	39.3	0.704	27	0	
23	8	99	84	0	0	35.4	0.388	50	0	
24	7	196	90	0	0	39.8	0.451	41	1	

Dataset Description :-

Context -

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian Heritage .

Context-

The datasets consist of several medical predictor (independent) variables and one target (dependent) variable, **Outcome**. Independent variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

Attributes in the dataset:-

There are 9 columns in the dataset -

7- Integer

2-Decimal

#Pregnancies- Number of Times Pregnant.

Glucose- Plasma glucose concentration a 2 hours in an oral glucose tolerance test.

#BloodPressure- Diastolic blood pressure (mm Hg).
#SkinThickness- Triceps skin fold thickness (mm).
#Insulin- 2-Hour serum insulin (mu U/ml).
#BMI- Body mass index (weight in kg/(height in m)^2).

DiabetesPedigreeFunction- DiabetesPedigreeFunction.
#Age- Age(Years).
#Outcome- Class variable (0 or 1) 268 of 768 are 1, the others are 0.
0 – No Diabetes and 1 – Diabetes.

Python Code:-

```
import pandas as pd
data=pd.read_csv("C:/Users/ML Lab/Downloads/diabetes.csv")
data
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

```
x=data.drop(["Outcome"],axis=1)
y=data['Outcome']
x
y
```

```
0      1
1      0
2      1
3      0
4      1
```

```
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest= train_test_split(x, y, test_size= 0.25, random_state=0)
print(xtrain.shape)
print(xtest.shape)
print(ytrain.shape)
print(ytest.shape)
```

```
(576, 8)
(192, 8)
(576,)
(192,)
```

```
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
xtrain= st_x.fit_transform(xtrain)
xtest= st_x.transform(xtest)
from sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier()
classifier.fit(xtrain, ytrain)
```

```
DecisionTreeClassifier()
```

```
ypred=classifier.predict(xtest)
ypred
```

```
array([1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0,
1, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1,
1, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0,
0, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0,
```

```
       1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
0, 1, 0, 1,
       0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
0, 1, 0, 1,
       0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0],
dtype=int64)
```

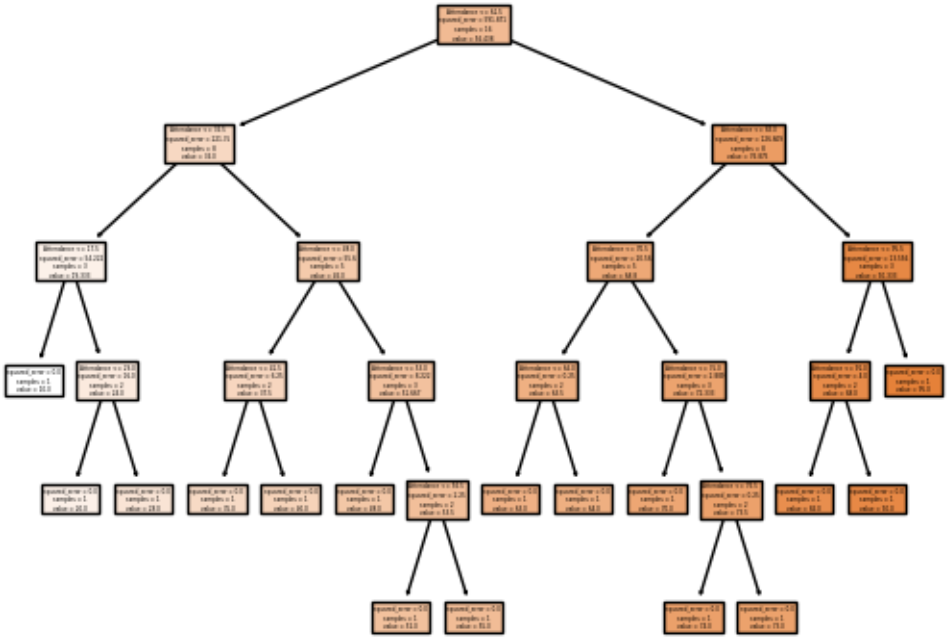
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(ytest,ypred)
cm

```
array([[100,  30],
       [ 25,  37]], dtype=int64)
```

from sklearn.metrics import accuracy_score
acc=accuracy_score(ytest,ypred)
acc

```
0.7135416666666666
```

#from sklearn.tree import export_graphviz
import matplotlib.pyplot as plt
from sklearn import tree
#fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(model,feature_names =['BMI'],class_names=['Outcome'],filled=True)
plt.show('imagename1.png')



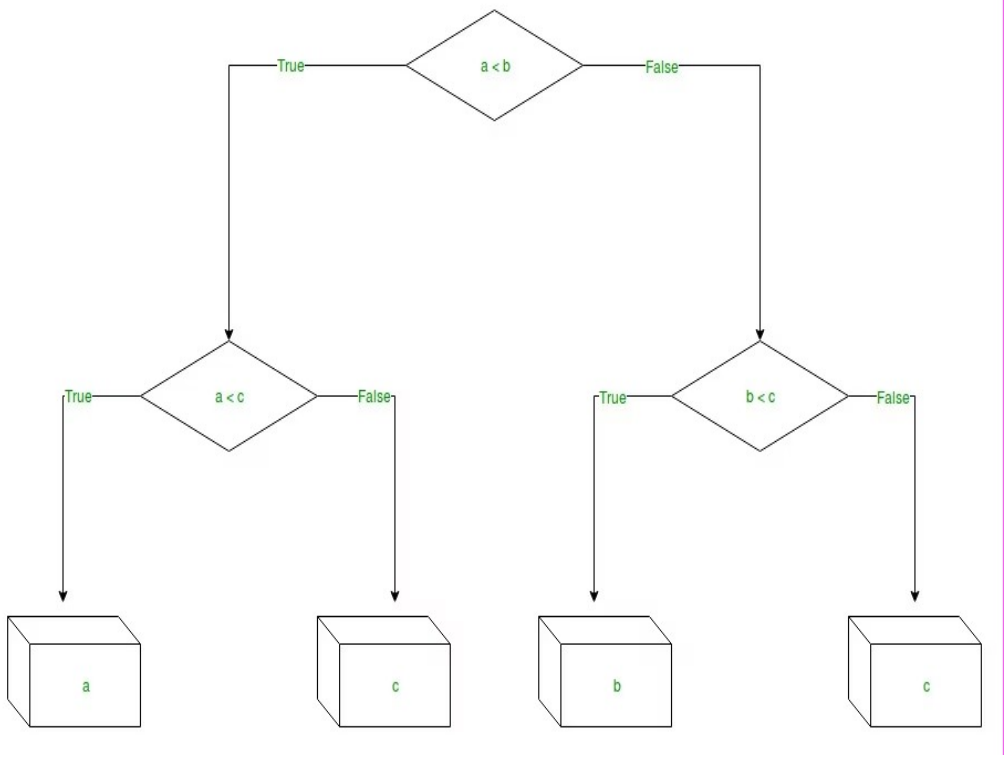
Aim: Write a program to demonstrate the working of the decision tree Regressor. Use appropriate dataset for building the decision tree and apply this knowledge to classify a new sample.

Decision Tree is a decision-making tool that uses a flowchart-like tree structure or is a model of decisions and all of their possible results, including outcomes, input costs, and utility.
Decision-tree algorithm falls und
the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.

The branches/edges represent the result of the node and the nodes have either:

Conditions [Decision Nodes]
Result [End Nodes]
The branches/edges represent the truth/falsity of the statement and take makes a decision based on that in the

example below which shows a decision tree that evaluates the smallest of three numbers:



Decision Tree Regression:

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.

Discrete output example: A weather prediction model that predicts whether or not there'll be rain on a particular day.
Continuous output example: A profit prediction model that states the probable profit that can be generated from the sale of a product.
Here, continuous values are predicted with the help of a decision tree regression model.

Syntax:

sklearn.tree.DecisionTreeRegressor(*,
criterion='squared_error', splitter='best', max_depth=None,
min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_features=None,
random_state=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, ccp_alpha=0.0)

Parameters:

criterion{“squared_error”, “friedman_mse”, “absolute_error”, “poisson”}, default=”squared_error”
The function to measure the quality of a split. Supported criteria are “squared_error” for the mean squared error, which is equal to variance reduction as feature selection criterion and minimizes the L2 loss using the mean of each terminal node, “friedman_mse”, which uses mean squared error with Friedman’s improvement score for potential splits, “absolute_error” for the mean absolute error, which minimizes the L1 loss using the median of each terminal node, and “poisson” which uses reduction in Poisson deviance to find splits.

New in version 0.18: Mean Absolute Error (MAE) criterion.

New in version 0.24: Poisson deviance criterion.

splitter{“best”, “random”}, default=”best”
The strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split.

max_depthint, default=None
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_splitint or float, default=2
The minimum number of samples required to split an internal node:

If int, then consider min_samples_split as the minimum number.

If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

Changed in version 0.18: Added float values for fractions.

min_samples_leafint or float, default=1
The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

If int, then consider min_samples_leaf as the minimum number.

If float, then min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

min_weight_fraction_leaffloat, default=0.0
The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

max_featuresint, float or {“auto”, “sqrt”, “log2”}, default=None
The number of features to consider when looking for the best split:

If int, then consider max_features features at each split.

If float, then max_features is a fraction and max(1, int(max_features * n_features_in_)) features are considered at each split.

If “auto”, then max_features=n_features.

If “sqrt”, then max_features=sqrt(n_features).

If “log2”, then max_features=log2(n_features).

If None, then max_features=n_features.

random_stateint, RandomState instance or None, default=None
Controls the randomness of the estimator. The features are always randomly permuted at each split, even if splitter is set to "best". When max_features < n_features, the algorithm will select max_features at random at each split before finding the best split among them. But the best found split may vary across different runs, even if max_features=n_features. That is the case, if the improvement of the criterion is identical for several splits and one split has to be selected at random. To obtain a deterministic behaviour during fitting, random_state has to be fixed to an integer. See Glossary for details.

max_leaf_nodesint, default=None
Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

min_impurity_decreasefloat, default=0.0
A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

The weighted impurity decrease equation is the following:

```
N_t / N * (impurity - N_t_R / N_t * right_impurity
                    - N_t_L / N_t * left_impurity)
```

where N is the total number of samples, N_t is the number of samples at the current node, N_t_L is the number of samples in the left child, and N_t_R is the number of samples in the right child.

N, N_t, N_t_R and N_t_L all refer to the weighted sum, if sample_weight is passed.
ccp_alphanon-negative float, default=0.0
Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed. See Minimal Cost-Complexity Pruning for details.

Data Set: Marks.csv

	Attendance	Marks
0	90	87
1	45	40
2	55	52
3	35	28
4	49	45
5	65	63
6	38	35
7	76	74
8	83	80
9	58	55
10	89	86
11	23	20
12	93	90
13	98	95
14	12	10
15	67	64
16	77	73
17	51	48
18	47	43
19	74	70

Python code:

```
import pandas as pd
data=pd.read_csv("D:/21761A4232/marks.csv")
data
```

```
   Attendance  Marks
0          90     87
1          45     40
2          55     52
3          35     28
4          49     45
5          65     63
6          38     35
7          76     74
8          83     80
```

```
9     58    55
10    89    86
11    23    20
12    93    90
13    98    95
14    12    10
15    67    64
16    77    73
17    51    48
18    47    43
19    74    70
```

```
x=data[['Attendance']]
y=data['Marks']
```

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2)
print(xtrain.shape)
print(xtest.shape)
print(ytrain.shape)
print(ytest.shape)
```

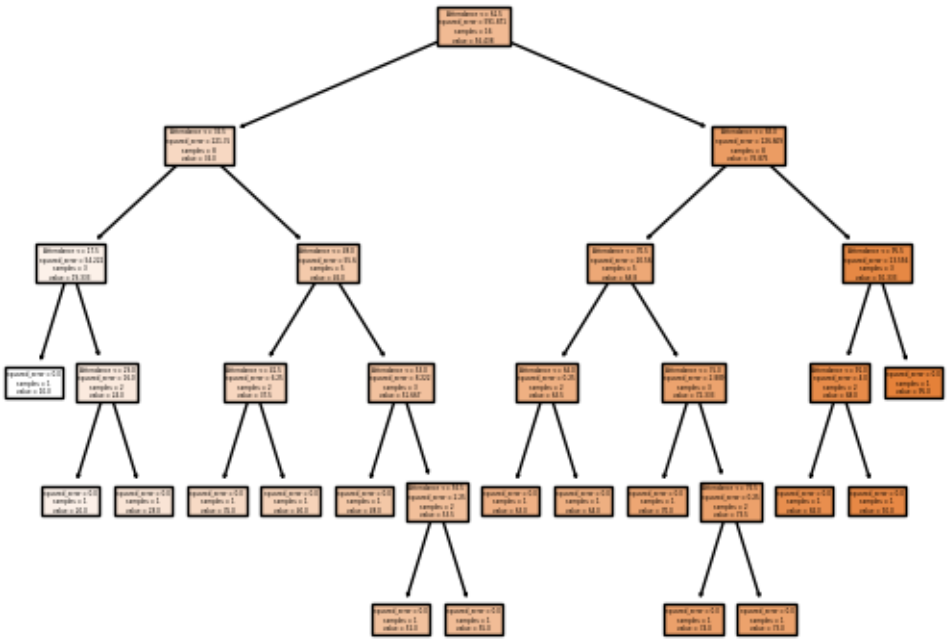
```
(16, 1)
(4, 1)
(16,)
(4,)
```

```
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor()
model.fit(xtrain,ytrain)
pred=model.predict(xtest)
pred
```

```
array([48., 40., 86., 73.])
```

```
#from sklearn.tree import export_graphviz
import matplotlib.pyplot as plt
from sklearn import tree
```

```
#fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(model,feature_names =['Attendance'],class_names=['Marks'],filled=True)
plt.show('imagename.png')
```



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

Introduction

Random forest is a supervised learning algorithm which is used for both classification as well as regression. But however, it is mainly used for classification problems. As we know that a forest is made up of trees and more trees means more robust forest. Similarly, random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

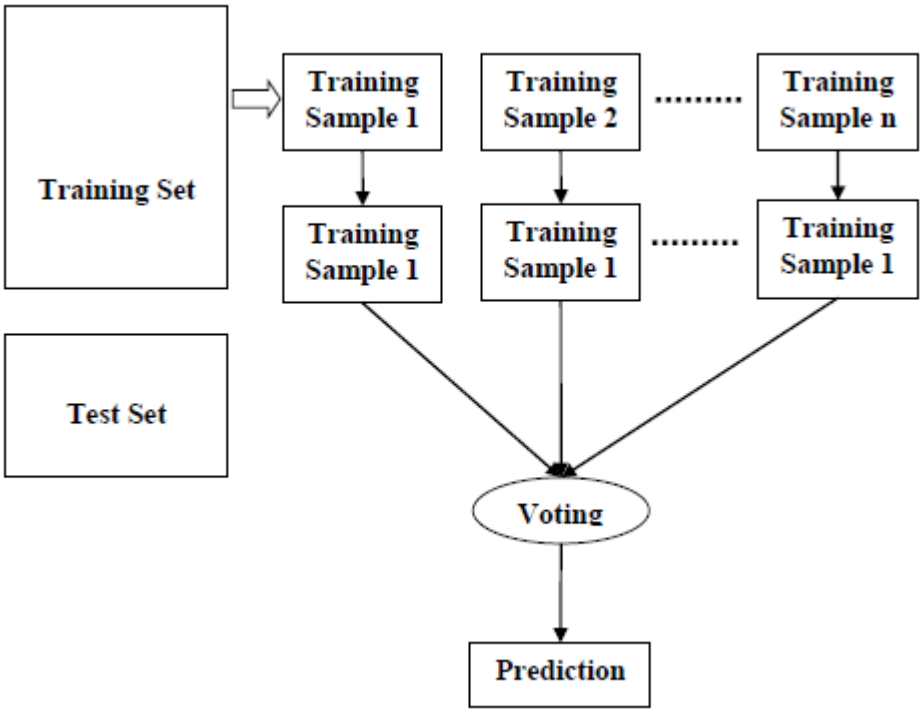
Working of Random Forest Algorithm

We can understand the working of Random Forest algorithm with the help of following steps –

- **Step 1** − First, start with the selection of random samples from a given dataset.

- **Step 2** − Next, this algorithm will construct a decision tree for every sample. Then it will get the prediction result from every decision tree.
- **Step 3** − In this step, voting will be performed for every predicted result.
- **Step 4** − At last, select the most voted prediction result as the final prediction result.

The following diagram will illustrate its working −



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"]

#Correlation

data.corr()

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

 # 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)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

O/P;

```
(576, 8)
(192, 8)
(576,)
(192,)
```

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)

O/P:

```
RandomForestClassifier()
```

Make predictions on the test data

y_pred = classifier.predict(X_test)
print(y_pred)

```
[1 1 0 0 0 1 0 1 1 1 0 1 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 0 0 0
1 0 0 1 1 1 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0
0 0 0 0 0 0 1
 0 0 0 1 0 0 0 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0
1 0 0 0 1 0 1
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 1 1
0 0 0 0 1 0 0
 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 0 0
0 0 0 0 1 0 1
 1 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0
0 1 0 0 0 1 0
 0 0 0 1 0 0 0 1 0]
```

Print the confusion matrix cm =

confusion_matrix(y_test, y_pred)

print("Confusion Matrix:")

cm

O/P:

Confusion Matrix:

Out[22]:

```
array([[115,  12],
       [ 31,  34]], dtype=int64)
```

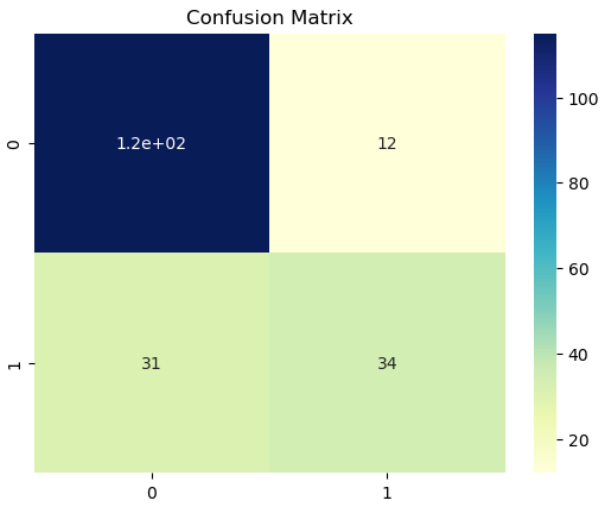
```
# Print the classification report

from sklearn.metrics import

classification_report res =

classification_report(y_test, y_pred)

print("\nClassification Report:\n", res)
```

O/P:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.91      0.84       127
           1       0.74      0.52      0.61        65

    accuracy                           0.78       192
   macro avg       0.76      0.71      0.73       192
weighted avg       0.77      0.78      0.76       192
```

 # Generate the confusion matrix
 # Create the heatmap using seaborn

```
import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix

sns.heatmap(cm,annot=True,cmap="YlGnBu")

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



#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.

#Evaluation Parameters

from sklearn.metrics import precision_score,recall_score, f1_score, accuracy_score,confusion_matrix

```
print("Accuracy:", accuracy_score(y_test,y_pred))

Accuracy: 0.7760416666666666

print("Precision:",
precision_score(y_test,y_pred,average="weighted"))

Precision: 0.7712381501886044


print('Recall:',
recall_score(y_test,y_pred,average="weighted"))
```

Recall: 0.776041666666666

AIM: Implementation of DFS for water jug problem:

Explanation:

Water Jug Problem is one of the most important problems to solve in Java. The water jug problem is a problem where we have two jugs, "**i**" liter jug and **"j"** liter jug **(0 < i < j)**. Both jugs will initially be empty, and they don't have marking to measure small quantities. Now, we need to measure d liters of water by using these two jugs where d < j. We use the following three operations to measure small quantities by using the two jars:

1. Empty a Jug.
2. Fill a Jug
3. We pour the water of one jug into another one until one of them is either full or empty.

In Java, we implement the logic for getting the minimum number of operations required to measure the d liter quantity of water.

There are various ways to solve water jug problems in Java, including GCD, BFS, and DP. In this section, we implement the logic for solving the Water Jug problem by using GCD.

Example: Water Jug Problem
Consider the following problem:
A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug?
State Representation and Initial State – we will represent a state of the problem as a tuple (x, y) where x represents the amount of water in the 4-gallon jug and y represents the amount of water in the 3-gallon jug. Note $0 \leq x \leq 4$, and $0 \leq y \leq 3$.
Our initial state: (0,0)
Goal Predicate – state = (2,y) where $0 \leq y \leq 3$.

<u>Production rules for solving the water jug problem</u>

Here, let ***x*** denote the 4-gallon jug and ***y*** denote the 3-gallon jug.

S.No.	Initial State	Condition	Final state	Description of action taken
1.	(x,y)	If x<4	(4,y)	Fill the 4 gallon jug completely
2.	(x,y)	if y<3	(x,3)	Fill the 3 gallon jug completely
3.	(x,y)	If x>0	(x-d,y)	Pour some part from the 4 gallon jug
4.	(x,y)	If y>0	(x,y-d)	Pour some part from the 3 gallon jug
5.	(x,y)	If x>0	(0,y)	Empty the 4 gallon jug
6.	(x,y)	If y>0	(x,0)	Empty the 3 gallon jug
7.	(x,y)	If (x+y)<7	(4, y-[4-x])	Pour some water from the 3 gallon jug to fill the four gallon jug
8.	(x,y)	If (x+y)<7	(x-[3-y],y)	Pour some water from the 4 gallon jug to fill the 3 gallon jug.
9.	(x,y)	If (x+y)<4	(x+y,0)	Pour all water from 3 gallon jug to the 4 gallon jug

10.	(x,y)	if (x+y)<3	(0, x+y)	Pour all water from the 4 gallon jug to the 3 gallon ju

The listed production rules contain all the actions that could be performed by the agent in transferring the contents of jugs. But, to solve the water jug problem in a minimum number of moves, following set of rules in the given sequence should be performed:

Solution of water jug problem according to the production rules

.	**4 gallon jug contents**	**3 gallon jug contents**	**Rule followed**
	0 gallon	0 gallon	Initial state
	0 gallon	3 gallons	Rule no.2
	3 gallons	0 gallon	Rule no. 9
	3 gallons	3 gallons	Rule no. 2
	4 gallons	2 gallons	Rule no. 7
	0 gallon	2 gallons	Rule no. 5
	2 gallons	0 gallon	Rule no. 9

On reaching the 7th attempt, we reach a state which is our goal state. Therefore, at this state,

our problem is solved

Implementation of water jug problem Using Python:

```
print("Rule 1:Fill x\n Rule 2:Fill y\n Rule 3:Empty x\n Rule 4:Empty y\n Rule 5:From y to x\n Rule 6:From x to y\n Rule 7:From y to x complete\n Rule 8:From x to y complete\n")
cap_x = int(input("Enter the jug 1 capacity: "))
cap_y = int(input("Enter the jug 2 capacity: "))
req_lis = list(map(str,input("Enter the required amount of water and in the jug you needed with space seperate: ").split()))
req_amount = int(req_lis[0])
req_jug = req_lis[1]
x=y=0
while(True):
   rule = int(input("Enter the rule: "))
   if rule==1:
      if x<cap_x:
         x = cap_x
   if rule==2:
      if y<cap_y:
         y = cap_y
   if rule==3:
      if x>0:
         x = 0
   if rule==4:
      if y>0:
         y = 0
   if rule==5:
      if 0<x+y>=cap_x and y>0:
         x,y = cap_x,y-(cap_x-x)
   if rule==6:
      if 0<x+y>=cap_y and x>0:
         x,y = x-(cap_y-y),cap_y
   if rule==7:
      if 0<x+y<=cap_x and y>=0:
         x = x+y
         y = 0
   if rule==8:
      if 0<x+y<=cap_y and x>=0:
```

```
            y = x+y
            x = 0
    print("x :",x)
    print("y :",y)
    if req_jug=='x':
        if req_amount==x:
            print("Goal reached")
            break
    elif req_jug=='y':
        if req_amount==y:
            print("Goal reached")
            break
```

output:

Rule 1:Fill x
Rule 2:Fill y
Rule 3:Empty x
Rule 4:Empty y
Rule 5:From y to x
Rule 6:From x to y
Rule 7:From y to x complete
Rule 8:From x to y complete

Enter the jug 1 capacity: 4
Enter the jug 2 capacity: 3
Enter the required amount of water and in the jug you needed with space seperate: 2 x
Enter the rule: 1
x : 4
y : 0
Enter the rule: 6
x : 1
y : 3
Enter the rule: 4
x : 1
y : 0
Enter the rule: 8
x : 0
y : 1
Enter the rule: 1
x : 4

y : 1
Enter the rule: 6
x : 2
y : 3
Goal reached

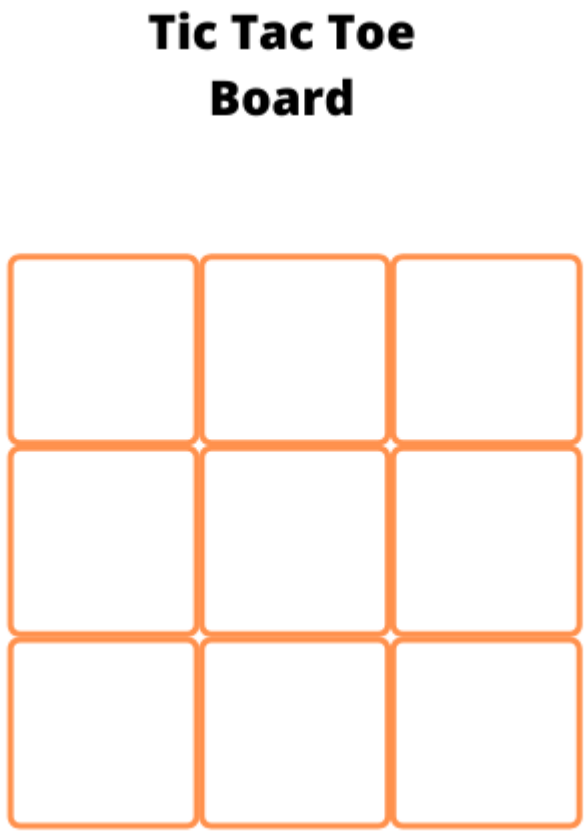
Implementation of water jug problem Using Java:

```
import java.util.Scanner;
public class Main
{
public static void main(String[] args) {
System.out.println("WATER JUG PROBLEM");
Scanner res=new Scanner(System.in);
  System.out.println("ENTER CAPACITY OF JUG-1 :");
  int x=res.nextInt();
  System.out.println("ENTER CAPACITY OF JUG-2 :");
  int y=res.nextInt();
  System.out.println("ENTER THE GOAL STATE :");
  int a=res.nextInt();
  do{
    System.out.println("ENTER rule num :");
    int rule=res.nextInt();
    if (rule==1){
       if (x<4)
            x=4;
    }
     else if (rule==2)
       {
          if (y<3)
           y=3;
       }
     else if (rule==3)
       {
           if (x>0)
            x=0;
       }
     else if (rule==4)
```

```
      {
         if (y>0)
         y=0;
      }
    else if (rule==5)
      {
         if (x+y>=4 && y>0)
         y=y-(4-x);x=4;
      }



else if (rule==6)
     {
        if (x+y>=3 && x>0)
         x=x-(3-y);y=3;
     }
    else if (rule==7)
      {
          if (x+y<=4 && y>=0)
         x=x+y;y=0;
      }
   else if (rule==8)
      {
          if (x+y<=4 && y>=0)
            y=x+y;x=0;
      }
   //if (x==a || y==a)
      //System.out.println("goal reached");
      //break;
   System.out.println("x= "+x);
   System.out.println("y= "+y);

  }while(x!=a && y!=a);
  System.out.println("GOAL REACHED");
}
}
```

Aim: Implement and demonstrate the Tic-Tac-Toe problem in python code.

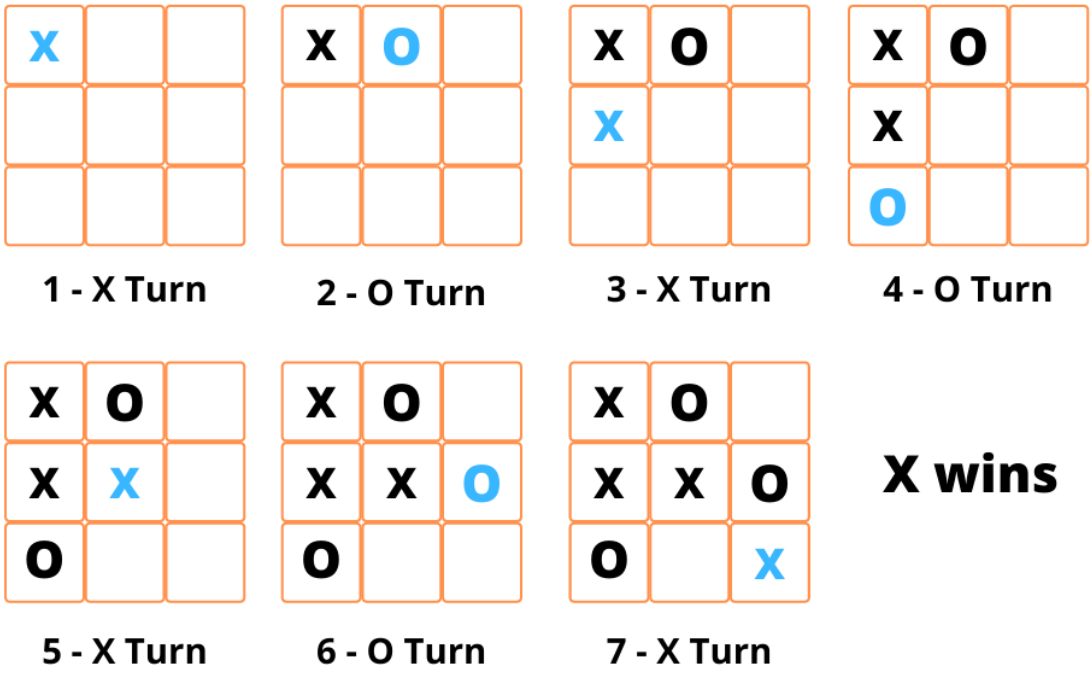
Explanation:
There will be two players in a game. Two signs represent each player. The general signs used in the game are **X** and **O**. Finally, there will be a board with **9** boxes.



The gameplay will be as follows.

- First, one user will place their sign in one of the available empty boxes.
- Next, the second user will place their sign in one of the available empty boxes.
- The goal of the players is to place their respective signs completely row-wise or column-wise, or diagonally.
- The game goes on until a player wins the game or it ended up in a draw by filling all boxes without a winning match.

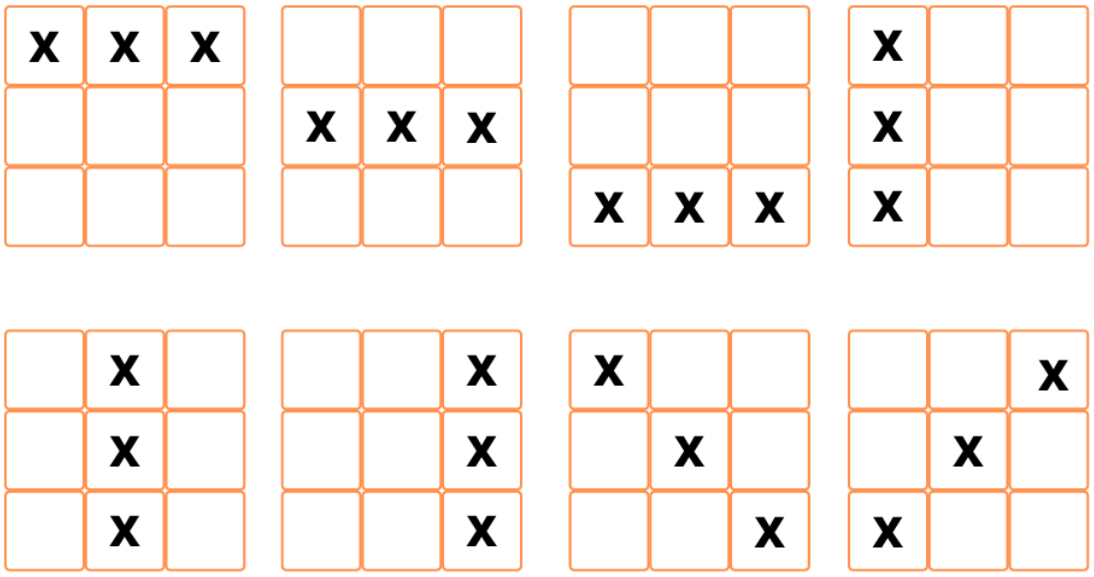
Tic Tac Toe - Gameplay



The player **X** wins the game in the above gameplay. All boxes diagonally fill with **X** signs. So, the respective player wins the game. There are a total of **8** ways to arrange the same sign and win the game. Let's see all the 8 arrangements that can win the game.

.

Tic Tac Toe - Winning Arrangements



Algorithm

- Create a board using a 2-dimensional array and initialize each element as empty.
    - You can represent empty using any symbol you like. Here, we are going to use a hyphen. '-'.
- Write a function to check whether the board is filled or not.
    - Iterate over the board and return false if the board contains an empty sign or else return true.
- Write a function to check whether a player has won or not.
    - We have to check all the possibilities that we discussed in the previous section.
    - Check for all the rows, columns, and two diagonals.
- Write a function to show the board as we will show the board multiple times to the users while they are playing.
- Write a function to start the game.
    - Select the first turn of the player randomly.
    - Write an infinite loop that breaks when the game is over (either win or draw).
        - Show the board to the user to select the spot for the next move.
        - Ask the user to enter the row and column number.
        - Update the spot with the respective player sign.
        - Check whether the current player won the game or not.

- If the current player won the game, then print a winning message and break the infinite loop.
- Next, check whether the board is filled or not.
- If the board is filled, then print the draw message and break the infinite loop.

- Finally, show the user the final view of the board.

Implementation of Tic-tac-Toe using python code:

```
def print_board(board):
    print('-------------')
    for i in range(3):
        row = '| '
        for j in range(3):
            row += board[i][j] + ' | '
        print(row)
        print('-------------')

def check_win(board, player):
    for i in range(3):
        if board[i][0] == player and board[i][1] == player and board[i][2] == player:
            return True
        if board[0][i] == player and board[1][i] == player and board[2][i] == player:
            return True
    if board[0][0] == player and board[1][1] == player and board[2][2] == player:
        return True
    if board[0][2] == player and board[1][1] == player and board[2][0] == player:
        return True
    return False

def tic_tac_toe():
    board = [[' ' for _ in range(3)] for _ in range(3)]
    players = ['X', 'O']
    current_player = players[0]
    print_board(board)
    while True:
        print(f"It's {current_player}'s turn.")
        row = int(input('Enter row (0-2): '))
        col = int(input('Enter column (0-2): '))
```

```
        if board[row][col] != ' ':
            print('That cell is already taken. Try again.')
            continue
        board[row][col] = current_player
        print_board(board)
        if check_win(board, current_player):
            print(f'{current_player} wins!')
            return

        if all([cell != ' ' for row in board for cell in row]):

            print("It's a tie!")

            return

        current_player = players[(players.index(current_player) + 1) % 2]
```

tic_tac_toe()

Output:

```
-------------
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
It's X's turn.
Enter row (0-2): 1
Enter column (0-2): 2
-------------
|   |   |   |
-------------
|   |   | X |
-------------
|   |   |   |
-------------
It's O's turn.
Enter row (0-2): 1
```

```
Enter column (0-2): 1
-------------
|   |   |   |
-------------
|   | O | X |
-------------
|   |   |   |
-------------
It's X's turn.
Enter row (0-2): 0
Enter column (0-2): 2
-------------
|   |   | X |
-------------
|   | O | X |
-------------
|   |   |   |
-------------
```

```
It's O's turn.
Enter row (0-2): 2
Enter column (0-2): 2
-------------
|   |   | X |
-------------
|   | O | X |
-------------
|   |   | O |
-------------
It's X's turn.
Enter row (0-2): 1
Enter column (0-2): 0
-------------
|   |   | X |
-------------
| X | O | X |
```

```
-------------
|   |   | O |
-------------
It's O's turn.
Enter row (0-2): 0
Enter column (0-2): 0
-------------
| O |   | X |
-------------
| X | O | X |
-------------
|   |   | O |
-------------
O wins!
```

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.

Explanation:

The find-S algorithm is a basic concept learning algorithm in machine learning. The find-S algorithm finds the most specific hypothesis that fits all the positive examples. We have to note here that the algorithm considers only those positive training example. The find-S algorithm starts with the most specific hypothesis and generalizes this hypothesis each time it fails to classify an observed positive training data. Hence, the Find-S algorithm moves from the most specific hypothesis to the most general hypothesis.

Important Representation :

1. **?** indicates that any value is acceptable for the attribute.
2. specify a single required value (e.g., Cold) for the attribute.
3. **ϕ**indicates that no value is acceptable.
4. The most **general hypothesis** is represented by: **{?, ?, ?, ?, ?, ?}**
5. The most **specific hypothesis** is represented by: **{ϕ, ϕ, ϕ, ϕ, ϕ, ϕ}**

Steps Involved In Find-S :

1. Start with the most specific hypothesis.
   h = {ϕ, ϕ, ϕ, ϕ, ϕ, ϕ}
2. Take the next example and if it is negative, then no changes occur to the hypothesis.
3. If the example is positive and we find that our initial hypothesis is too specific then we update our current hypothesis to a general condition.

4. Keep repeating the above steps till all the training examples are complete.
5. After we have completed all the training examples we will have the final hypothesis when can use to classify the new examples.

Implementation of Find-S Algorithm

In [100]:
```
import pandas as pd
data = pd.read_csv(r"C:\Users\ML Lab\Downloads\find-s.csv")
print(data)
```

```
     sky airtemp humidity    wind water forecast enjoysport
0  sunny    warm   normal  strong  warm     same        yes
1  sunny    warm     high  strong  warm     same        yes
2  rainy    cold     high  strong  warm   change         no
3  sunny    warm     high  strong  cool   change        yes
```

In [105]:
```
lis1 = X.values.tolist()
lis2 = y.values.tolist()
h = ['*']*len(lis1[0])
h = lis1[0]
res = []
for i in range(len(lis2)):
    if lis2[i]=="yes":
        res.append(lis1[i])
for i in range(len(res)):
    for j in range(len(res[i])):
        if h[j]==res[i][j]:
            pass
        else:
            h[j]="?"
    print(h)
```

```
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
```

Output:

```
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
['sunny', 'warm', '?', 'strong', 'warm', 'same']
['sunny', 'warm', '?', 'strong', '?', '?']
```

AIM: For a given set of training data examples stored in a .csv file, implement and demonstrate the Candidate Elimination algorithm to output a description of the set of all hypothesis consistent withthe training examples.

Explanation:

The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example.

- You can consider this as an extended form of the Find-S algorithm.
- Consider both positive and negative examples.
- Actually, positive examples are used here as the Find-S algorithm (Basically they are generalizing from the specification).
- While the negative example is specified in the generalizing form.

Terms Used:

- **Concept learning:** Concept learning is basically the learning task of the machine (Learn by Train data)
- **General Hypothesis:** Not Specifying features to learn the machine.
- **G = {'?', '?','?','?'…}:** Number of attributes
- **Specific Hypothesis:** Specifying features to learn machine (Specific feature)
- **S= {'pi','pi','pi'…}: The number** of pi depends on a number of attributes.
- **Version Space:** It is an intermediate of general hypothesis and Specific hypothesis. It not only just writes one hypothesis but a set of all possible hypotheses based on training data-set.

Algorithm:

Step1: Load Data set

Step2: Initialize General Hypothesis and Specific Hypothesis.
Step3: For each training example
Step4: If example is positive example

```
        if attribute_value == hypothesis_value:
           Do nothing
        else:
           replace attribute value with '?' (Basically
generalizing it)
```

Step5: If example is Negative example

```
        Make generalize hypothesis more specific.
```

Example:
Consider the dataset given below:

Sky	Temperature	Humid	Wind	Water	Forest	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

Algorithmic steps:

Initially : G = [[?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?],
[?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?]]
S = [Null, Null, Null, Null, Null, Null]

For instance 1 : <'sunny','warm','normal','strong','warm','same'> and positive output.
G1 = G
S1 = ['sunny','warm','normal','strong','warm','same']

For instance 2 : <'sunny','warm','high','strong','warm','same'> and positive output.
G2 = G

```
            S2 = ['sunny','warm',?,'strong','warm ','same']
```

For instance 3 : <'rainy','cold','high','strong','warm ','change'> and negative output.

```
            G3 = [['sunny', ?, ?, ?, ?, ?], [?, 'warm', ?, ?, ?, ?], [?, ?, ?, ?, ?, ?],
                  [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, 'same']]
            S3 = S2
```

For instance 4 :

<'sunny','warm','high','strong','cool','change'> and positive output.

```
            G4 = G3
            S4 = ['sunny','warm',?,'strong', ?, ?]
```

At last, by synchronizing the G4 and S4 algorithm produce the output.

Output :

```
G = [['sunny', ?, ?, ?, ?, ?], [?, 'warm', ?, ?, ?, ?]]

S = ['sunny','warm',?,'strong', ?, ?]
```

The Candidate Elimination Algorithm (CEA) is an improvement over the Find-S algorithm for classification tasks. While CEA shares some similarities with Find-S, it also has some essential differences that offer advantages and disadvantages. Here are some advantages and disadvantages of CEA in comparison with Find-S:

Advantages of CEA over Find-S:

1. Improved accuracy: CEA considers both positive and negative examples to generate the hypothesis, which can result in higher accuracy when dealing with noisy or incomplete data.
2. Flexibility: CEA can handle more complex classification tasks, such as those with multiple classes or non-linear decision boundaries.
3. More efficient: CEA reduces the number of hypotheses by generating a set of general hypotheses and then eliminating them one by one. This can result in faster processing and improved efficiency.
4. Better handling of continuous attributes: CEA can handle continuous attributes by creating boundaries for each attribute, which makes it more suitable for a wider range of datasets.

Disadvantages of CEA in comparison with Find-S:

1. More complex: CEA is a more complex algorithm than Find-S, which may make it more difficult for beginners or those without a strong background in machine learning to use and understand.
2. Higher memory requirements: CEA requires more memory to store the set of hypotheses and boundaries, which may make it less suitable for memory-constrained environments.
3. Slower processing for large datasets: CEA may become slower for larger datasets due to the increased number of hypotheses generated.
4. Higher potential for overfitting: The increased complexity of CEA may make it more prone to overfitting on the training data, especially if the dataset is small or has a high degree of noise.

Implementation of Candidate algorithm:

In [203]:
```
import pandas as pd
data=pd.read_excel(r"C:\Users\B V N DURGA VINAY\Downloads\candidate.csv")
data
```

Out[203]:

	sky	temperature	humid	wind	water	forest	output
0	sunny	warm	normal	strong	warm	same	yes
1	sunny	warm	high	strong	warm	same	yes
2	rainy	cold	high	strong	warm	change	no
3	sunny	warm	high	strong	cool	change	yes

In [204]:
```
data.shape
```

Out[204]: (4, 7)

In [205]:
```
S=[]
for i in range(0,data.shape[1]-1):
    S.append('0')
print(S)
```

```
['0', '0', '0', '0', '0', '0']
```

In [206]:
```
G = [['?' for i in range(data.shape[1]-1)] for i in range(data.shape[1]-1)]
print(G)
```

In [206]:
```
G = [['?' for i in range(data.shape[1]-1)] for i in range(data.shape[1]-1)]
print(G)
```

```
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

In [207]:
```
x = data.iloc[:,:-1]
y = data.iloc[:,-1]
a = X.values.tolist()
b = Y.values.tolist()
print(a)
print(b)
```

```
[['sunny', 'warm', 'normal', 'strong', 'warm', 'same'], ['sunny', 'warm', 'high', 'strong', 'warm', 'same'], ['rainy', 'col
d', 'high', 'strong', 'warm', 'change'], ['sunny', 'warm', 'high', 'strong', 'cool', 'change']]
['yes', 'yes', 'no', 'yes']
```

In [208]:
```
l=a[0]
m=0
for i in range(0,data.shape[0]):
        n=b[i]
        k=list(a[i])
        print(n)
        print(k)
        if(n=='yes'):
            for j in range(0,len(l)):
                    if k[j]!=S[j] and S[j]!='0':
                        S[j]='?'
                    elif k[j]!=S[j] and S[j]=='0':
                        S=k
        else:
            for j in range(0,len(l)):
                if k[j]!=S[j]:
                    G[m][j]=S[j]
                    m=m+1
                else:
                    G[m][j]='?'
                    m=m+1
print("G:",G)
```

```
yes
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
yes
['sunny', 'warm', 'high', 'strong', 'warm', 'same']
no
['rainy', 'cold', 'high', 'strong', 'warm', 'change']
yes
['sunny', 'warm', 'high', 'strong', 'cool', 'change']
G: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
```

In [209]:
```
G1 = list(G)
r = [G[j].count('?') for j in range(len(G))]
for i in r:
    if i>data.shape[1]-2:
        G.pop(r.index(i))
```

```
    if i>data.shape[1]-2:
        G.pop(r.index(i))
p = []
for i in range(len(a)):
    if b[i]=='yes':
        p.append(a[i])
c = 0
for i in range(len(G1[0])):
    if p[c][i]!=G1[c][i]:
        G1.pop(c)
    else:
        c=c+1
print("G is",G1)
print("S is",S)
```

Output:

```
G is [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
S is ['sunny', 'warm', '?', 'strong', '?', '?']
```