\* MACHINE LEARNING ALGORITHMS \*

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

Description :-

The Find-S algorithm is a basic concept-learning algorithm used in machine learning. It starts with the most specific hypothesis, which is an empty hypothesis that contains no attribute or value. It then considers each training example and generalizes the hypothesis by adding the attribute values that are consistent with the example. The algorithm stops when all the training examples are consistent with the hypothesis, or when there are no more examples to consider. The resulting hypothesis is the most specific hypothesis that is consistent with the training data.

Dataset:-

|  | **sky** | **airtemp** | **humidity** | **wind** | **water** | **forecar** | **enjoysport** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | sunny | warm | normal | strong | warm | same | yes |
| **1** | sunny | warm | high | strong | warm | same | yes |
| **2** | rain | cold | high | strong | warm | change | no |
| **3** | sunny | warm | high | strong | cool | change | yes |

Python Code:-

import pandas as pd

data=pd.read\_csv("C:/Users/ramya/OneDrive/Documents/ml\_csv/finds.csv")

print(data)

h0=[]

d=0

for i in enumerate(data):

d=d+1

for i in range(d-1):

h0.append(0)

print("hypothesis initially:",h0)

count=0

output:-

sky airtemp humidity wind water forecar enjoysport

sunny warm normal strong warm same yes

sunny warm high strong warm same yes

rain cold high strong warm change no

sunny warm high strong cool change yes

hypothesis initially: [0, 0, 0, 0, 0, 0]

**for** i **in** range(len(data)):

**if**(data.iloc[i,**-**1]**==**'yes'):

count**=**count**+**1

*#print(count)*

**for** j **in** range(len(h0)):

a**=**data.iloc[i,j]

**if**(h0[j]**==**a):

**continue**

**elif**(count**==**1):

h0[j]**=**a

*#print(h0[j])*

**else**:

h0[j]**=**'?'

print(h0)

output:-

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

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

['sunny', 'warm', '?', 'strong', '?', '?']

2. For a given set of training data examples stored in a .csv file, implement and demonstrate thecandidate elimination algorithm to output a description of the set of all hypotheses consistent withthe training examples.

Description :-

The Candidate Elimination algorithm is a concept-learning algorithm used in machine learning. It starts with the most specific and most general hypotheses, which are the hypotheses that contain the smallest and largest possible sets of attribute values, respectively. It then considers each training example and updates the hypotheses by eliminating inconsistent hypotheses and generalizing or specializing the remaining hypotheses as necessary. The algorithm stops when it finds a single hypothesis that fits all the training examples or when it cannot further generalize or specialize the hypotheses. The resulting set of hypotheses is a minimal set of hypotheses that are consistent with the training data.

Dataset :-

| **sky** | **airtemp** | **humidity** | **wind** | **water** | **forecar** | **enjoysport** |
| --- | --- | --- | --- | --- | --- | --- |
| sunny | warm | normal | strong | warm | same | yes |
| sunny | warm | high | strong | warm | same | yes |
| rain | cold | high | strong | warm | change | no |
| sunny | warm | high | strong | cool | change | yes |

Python Code :-

import pandas as pd

data=pd.read\_csv("C:/Users/ramya/OneDrive/Documents/ml\_csv/finds.csv")

d=0

count=0

g=[]

s=[]

for i in enumerate(data):

d=d+1

g= [["?" for i in range(d-1)] for i in range(d-1)]

for i in range(d-1):

s.append(data.iloc[1,i])

for i in range(len(data)):

if(data.iloc[i,-1]=='yes'):

for j in range(d-1):

if(s[j]==data.iloc[i,j]):

continue

else:

s[j]="?"

print(“Specific : ”,s)

else:

for j in range(d-1):

if(s[j]!=data.iloc[i,j]):

g[j][j]=s[j]

else:

continue

print(“Generalized : ”,g)

for i in range(d-1):

if(g[i][i]!=s[i]):

g[i][i]='?'

indices = [i for i, val in enumerate(g) if val == ['?', '?', '?', '?', '?', '?']]

for i in indices:

g.remove(['?', '?', '?', '?', '?', '?'])

print(“Synchronization : ”,g)

Output :-

Specific : ['sunny', 'warm', '?', 'strong', 'warm', 'same']

Specific : ['sunny', 'warm', '?', 'strong', 'warm', 'same']

Generalized : [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

Specific : ['sunny', 'warm', '?', 'strong', '?', '?']

Synchronization : [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

3.Linear and Multilinear Regression on two different .csv Files.

Description :-

Linear Regression :-

Description:-

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

Dataset :-

| **attendence** | **marks** |
| --- | --- |
| **0** | 67 | 69 |
| **1** | 68 | 70 |
| **2** | 69 | 71 |
| **3** | 70 | 72 |
| **4** | 71 | 73 |
| **5** | 72 | 74 |
| **6** | 73 | 75 |
| **7** | 74 | 76 |
| **8** | 75 | 77 |
| **9** | 76 | 78 |
| **10** | 77 | 79 |
| **11** | 78 | 80 |
| **12** | 79 | 81 |
| **13** | 80 | 82 |
| **14** | 81 | 83 |
| **15** | 82 | 84 |
| **16** | 83 | 85 |
| **17** | 84 | 86 |
| **18** | 85 | 87 |
| **19** | 86 | 88 |

Python Code :-

import pandas as pd

from sklearn.metrics import r2\_score

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as mp

data=pd.read\_csv("C:/Users/ramya/OneDrive/Documents/ml\_csv/linear.csv")

print(data)

x=data.iloc[:,0:-1]

y=data.iloc[:,-1]

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3)

model=LinearRegression()

model.fit(x\_train,y\_train)

result=model.predict(x\_test)

result

print(r2\_score(y\_test,result))

Output :-

Data :-

Attendance marks

0 67 69

1 68 70

2 69 71

3 70 72

4 71 73

5 72 74

6 73 75

7 74 76

8 75 77

9 76 78

10 77 79

11 78 80

12 79 81

13 80 82

14 81 83

15 82 84

16 83 85

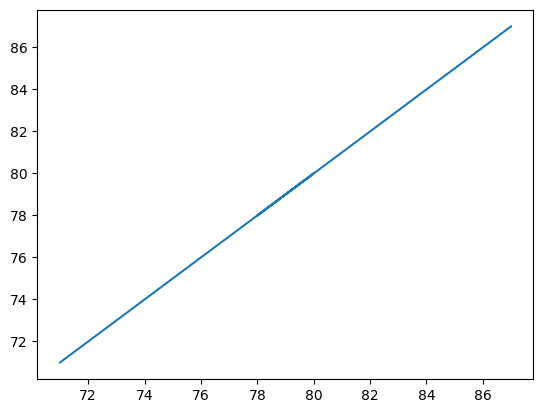
17 84 86

18 85 87

19 86 88

Accuracy :1.0

mp. plot(y\_test,result)



Multilinear Regression:-

Description :-

Multilinear regression is an extension of linear regression that models the relationship between a dependent variable and multiple independent variables by fitting a hyperplane to the data.

Dataset : -

| **attendence** | **certification** | **marks** |
| --- | --- | --- |
| **0** | 65 | 2 | 68 |
| **1** | 66 | 3 | 69 |
| **2** | 67 | 3 | 70 |
| **3** | 68 | 2 | 71 |
| **4** | 69 | 3 | 72 |
| **5** | 70 | 2 | 73 |
| **6** | 71 | 4 | 74 |
| **7** | 72 | 2 | 75 |
| **8** | 73 | 3 | 76 |
| **9** | 74 | 3 | 77 |
| **10** | 75 | 2 | 78 |
| **11** | 76 | 1 | 79 |
| **12** | 77 | 2 | 80 |
| **13** | 78 | 2 | 81 |
| **14** | 79 | 2 | 82 |
| **15** | 80 | 2 | 83 |
| **16** | 81 | 2 | 84 |
| **17** | 82 | 2 | 85 |
| **18** | 83 | 1 | 86 |
| **19** | 84 | 2 | 87 |

Python Code:-

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import r2\_score

import matplotlib.pyplot as plt

data=pd.read\_csv("C:/Users/ramya/OneDrive/Documents/ml\_csv/multi.csv")

print(data)

x=data.iloc[:,0:-1]

y=data.iloc[:,-1]

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3)

model=LinearRegression()

model.fit(x\_train,y\_train)

result=model.predict(x\_test)

result

print(r2\_score(y\_test,result))

attendence certification marks

0 65 2 68

1 66 3 69

2 67 3 70

3 68 2 71

4 69 3 72

5 70 2 73

6 71 4 74

7 72 2 75

8 73 3 76

9 74 3 77

10 75 2 78

11 76 1 79

12 77 2 80

13 78 2 81

14 79 2 82

15 80 2 83

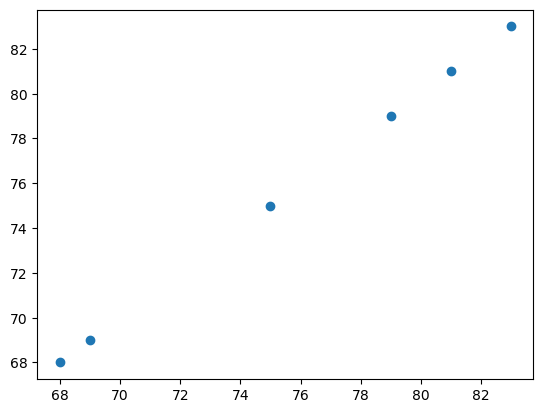
16 81 2 84

17 82 2 85

18 83 1 86

19 84 2 87

Accuracy :1.0



4.Polynomial Regression on student marks data

Description :-

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

Dataset :-

| **attendence** | **certification** | **marks** |
| --- | --- | --- |
| **0** | 65 | 2 | 68 |
| **1** | 66 | 3 | 69 |
| **2** | 67 | 3 | 70 |
| **3** | 68 | 2 | 71 |
| **4** | 69 | 3 | 72 |
| **5** | 70 | 2 | 73 |
| **6** | 71 | 4 | 74 |
| **7** | 72 | 2 | 75 |
| **8** | 73 | 3 | 76 |
| **9** | 74 | 3 | 77 |
| **10** | 75 | 2 | 78 |
| **11** | 76 | 1 | 79 |
| **12** | 77 | 2 | 80 |
| **13** | 78 | 2 | 81 |
| **14** | 79 | 2 | 82 |
| **15** | 80 | 2 | 83 |
| **16** | 81 | 2 | 84 |
| **17** | 82 | 2 | 85 |
| **18** | 83 | 1 | 86 |
| **19** | 84 | 2 | 87 |

Python Code :-

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

data=pd.read\_csv("C:/Users/ramya/OneDrive/Documents/ml\_csv/multi.csv")

x=data.drop("marks",axis=1)

y=data[["marks"]]

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3)

p=PolynomialFeatures(degree=3)

x\_train\_poly=p.fit\_transform(x\_train)

x\_test\_poly=p.fit\_transform(x\_test)

print(x\_train)

print(x\_train\_poly)

print(x\_test)

print(x\_test\_poly)

Output : #Train

attendence certification

6 71 4

15 80 2

5 70 2

7 72 2

10 75 2

14 79 2

12 77 2

1 66 3

18 83 1

4 69 3

19 84 2

13 78 2

16 81 2

11 76 1

#after polynomial degree applied Train

[[1.00000e+00 7.10000e+01 4.00000e+00 5.04100e+03 2.84000e+02 1.60000e+01

3.57911e+05 2.01640e+04 1.13600e+03 6.40000e+01

#Test

attendence certification

17 82 2

0 65 2

8 73 3

2 67 3

9 74 3

3 68 2

#After polynomial degree Aplied test

[[1.00000e+00 8.20000e+01 2.00000e+00 6.72400e+03 1.64000e+02 4.00000e+00

5.51368e+05 1.34480e+04 3.28000e+02 8.00000e+00]

. . .

model**=**LinearRegression()

model.fit(x\_train\_poly,y\_train)

y\_predict**=**model.predict(x\_test\_poly)

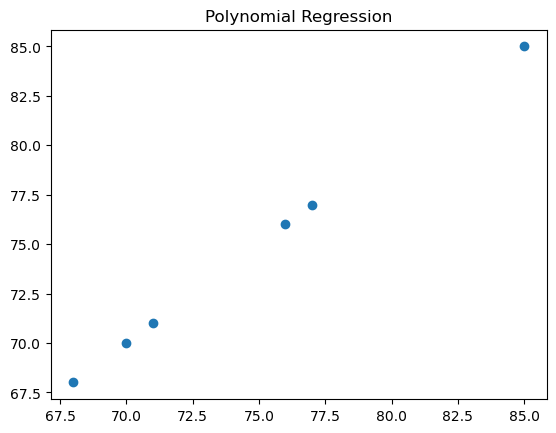
acc**=**r2\_score(y\_test,y\_predict)

print(acc)

plt.title("Polynomial Regression")

plt.scatter(y\_test,y\_predict)

accuracy: 0.9999999997846436



5. Write a program to demonstrate the working of Logistic Regression classifier. Use appropriate dataset for Logistic Regression.

Evalution Parameters : f1\_score,accuracy,recall

Description :-

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

Dataset :-

<https://github.com/plotly/datasets/blob/master/diabetes.csv>

Python code:-

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy\_score, f1\_score, precision\_score, recall\_score

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

data=pd.read\_csv("C:/Users/ramya/OneDrive/Documents/ml\_csv/diabetes/diabetes.csv")

df=data.isnull().sum()

#print(df)

x=data.drop("Outcome",axis=1)

y=data[["Outcome"]]

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3)

print(x)

print(y)

Pregnancies 0

Glucose 0

BloodPressure 0

SkinThickness 0

Insulin 0

BMI 0

DiabetesPedigreeFunction 0

Age 0

Outcome 0

st=StandardScaler()

x\_train=st.fit\_transform(x\_train)

x\_test=st.fit\_transform(x\_test)

model=LogisticRegression()

model.fit(x\_train,y\_train)

y\_predict=model.predict(x\_test)

print("predicted:",y\_predict)

acc=accuracy\_score(y\_test,y\_predict)

print("accuracy : ",acc)

print(y\_test)

from sklearn import metrics

confusion\_matrix = metrics.confusion\_matrix(y\_test,y\_predict)

print(confusion\_matrix)

#Tp,Fp,Tn,Fn output:-

[[138 26]

[ 25 42]]

print("f1score: ",f1\_score(y\_test,y\_predict))

print("recall\_score",recall\_score(y\_test,y\_predict))

print("precission",precision\_score(y\_test,y\_predict))

output :-

f1score: 0.6222222222222222

recall\_score 0.6268656716417911

precission 0.6176470588235294

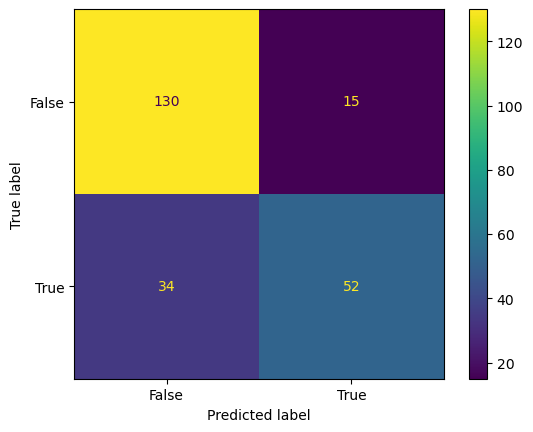
cm\_display = metrics.ConfusionMatrixDisplay(confusion\_matrix = confusion\_matrix, display\_labels = [False, True])

cm\_display.plot()

plt.show()

Output:-

Visualization with Confusion matrix



**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**

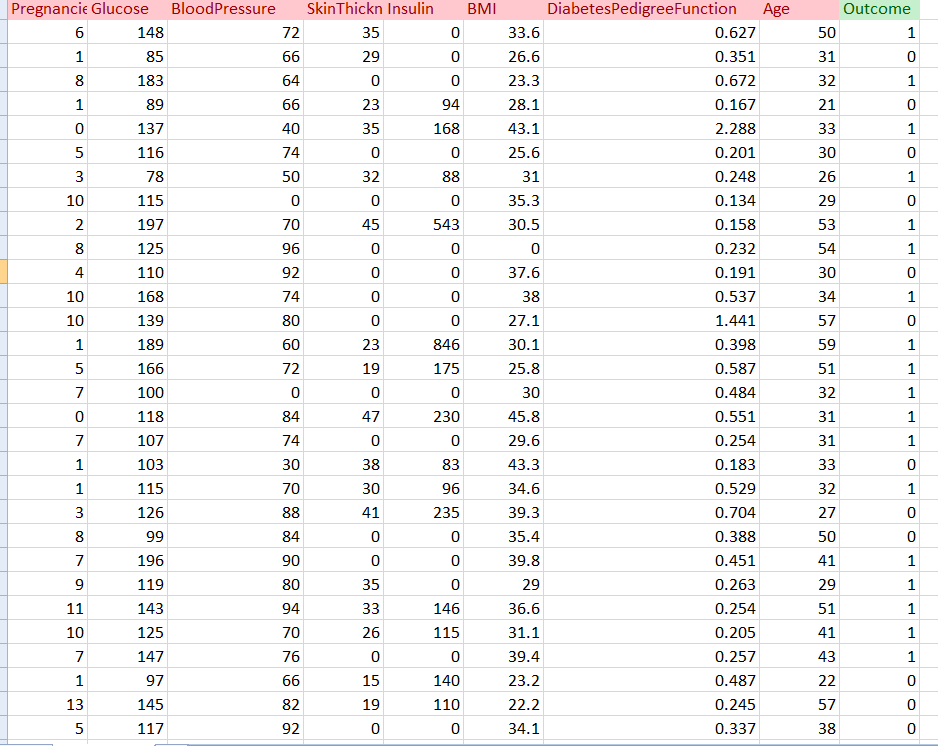
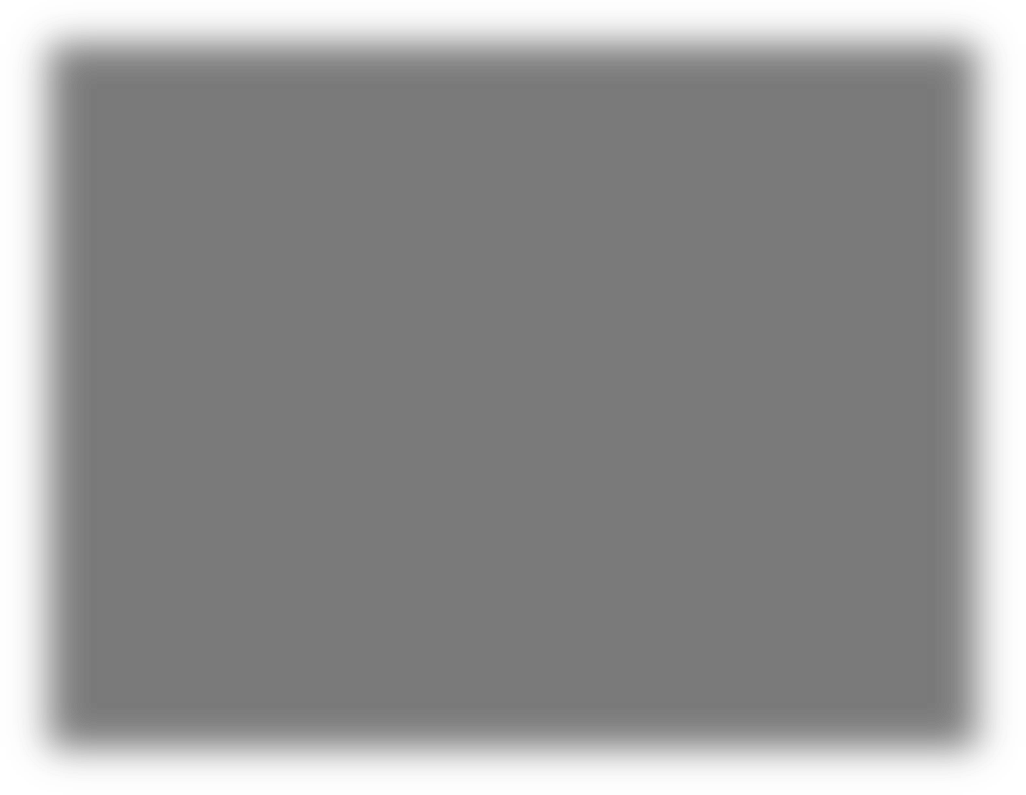
We can understand the working of Random Forest algorithm with the help of following steps −

* **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



# < 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()



**# 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")

# 

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"))

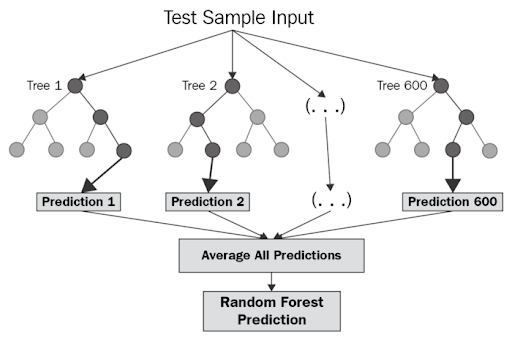
Recall: 0.776041666666666

**Aim: 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](https://builtin.com/data-science/regression-machine-learning).

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.

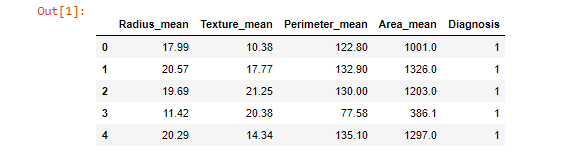
**random\_state:**controls randomness of the sample. The model will always produce the same results if it has a definite value of random state and has been given the same hyperparameters and training data.

**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**

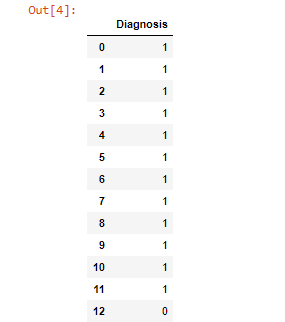
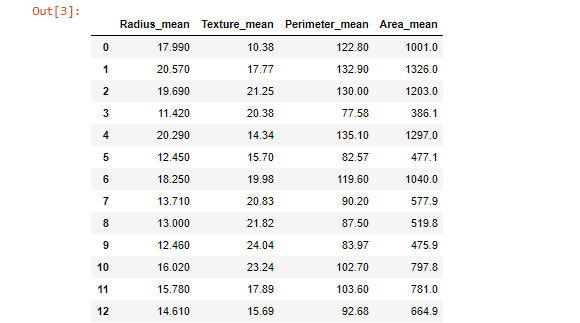
**Step 1: Load Pandas library and the dataset using Pandas**



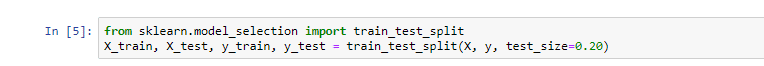
**Step 2: Define the features and the target**



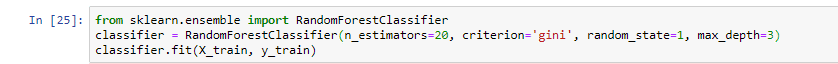
**Step 2: Define the features and the target**



**Step 3: Split the dataset into train and test sklearn**



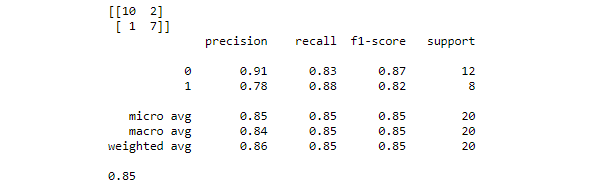
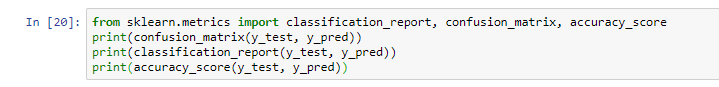
**Step 4:** **Import the random forest classifier function from sklearn ensemble module.** **Build the random forest classifier model with the help of the random forest classifier function**



**Step 5: Predict values using the random forest classifier model**



**Step 6: Evaluate the random forest classifier model**



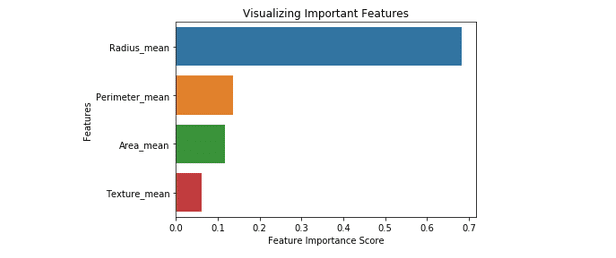
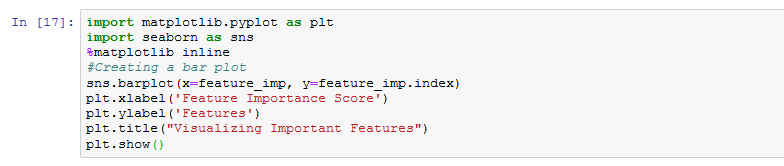
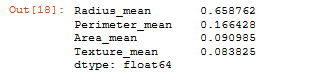
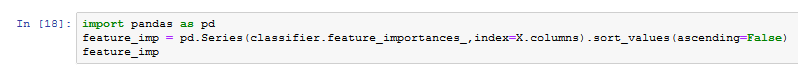
## **Feature Selection in Random Forest Algorithm Model**

With the help of [Scikit-Learn](https://intellipaat.com/blog/tutorial/python-tutorial/scikit-learn-tutorial/), 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.

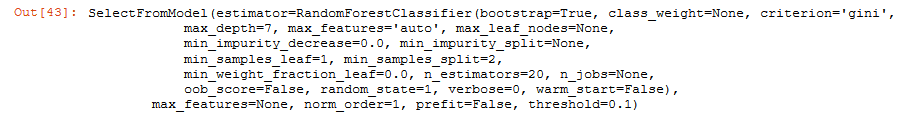
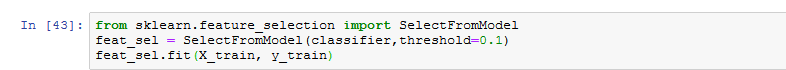
Let us see if selecting features make any difference in the accuracy score of the model.

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

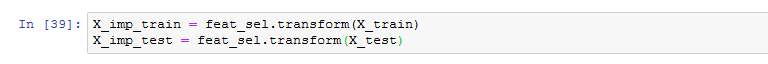


Now let us see how the ‘SelectFromModel’ function helps in building a random forest classifier model with important features.

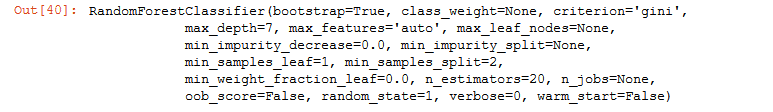
**Step 8: Import the SelectFromModel function. We will pass the classifier object we’ve created above. Also, we will add a threshold value of 0.1**



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



**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)**



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



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



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)

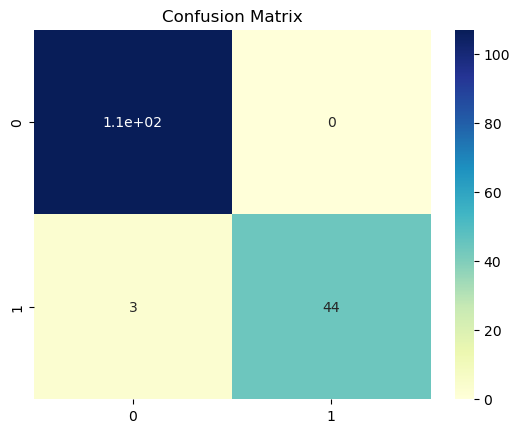
**import** seaborn **as** sns

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

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

Out[72]:

Text(0.5, 1.0, 'Confusion Matrix')



**AIM: 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 ≤ x ≤ 4, and 0 ≤ y ≤ 3.

**Our initial state: (0,0)**

**Goal Predicate – state = (2,y) where 0 ≤ y ≤ 3.**

**Production rules for solving the water jug problem**

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 jug |

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**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **4 gallon jug contents** | **3 gallon jug contents** | **Rule followed** |
| 1. | 0 gallon | 0 gallon | Initial state |
| 2. | 0 gallon | 3 gallons | Rule no.2 |
| 3. | 3 gallons | 0 gallon | Rule no. 9 |
| 4. | 3 gallons | 3 gallons | Rule no. 2 |
| 5. | 4 gallons | 2 gallons | Rule no. 7 |
| 6. | 0 gallon | 2 gallons | Rule no. 5 |
| 7. | 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**

from tkinter import \*

import random

def next\_turn(row, column):

    global player

    if buttons[row][column]['text'] == "" and check\_winner() is False:

        if player == players[0]:

            buttons[row][column]['text'] = player

            if check\_winner() is False:

                player = players[1]

                label.config(text=(players[1]+" turn"))

            elif check\_winner() is True:

                label.config(text=(players[0]+" wins"))

            elif check\_winner() == "Tie":

                label.config(text="Tie!")

        else:

            buttons[row][column]['text'] = player

            if check\_winner() is False:

                player = players[0]

                label.config(text=(players[0]+" turn"))

            elif check\_winner() is True:

                label.config(text=(players[1]+" wins"))

            elif check\_winner() == "Tie":

                label.config(text="Tie!")

def check\_winner():

    for row in range(5):

        if buttons[row][0]['text'] == buttons[row][1]['text'] == buttons[row][2]['text'] == buttons[row][3]['text'] == buttons[row][4]['text'] != "":

            buttons[row][0].config(bg="green")

            buttons[row][1].config(bg="green")

            buttons[row][2].config(bg="green")

            buttons[row][3].config(bg="green")

            buttons[row][4].config(bg="green")

            return True

    for column in range(5):

        if buttons[0][column]['text'] == buttons[1][column]['text'] == buttons[2][column]['text'] == buttons[3][column]['text'] == buttons[4][column]['text']!= "":

            buttons[0][column].config(bg="green")

            buttons[1][column].config(bg="green")

            buttons[2][column].config(bg="green")

            buttons[3][column].config(bg="green")

            buttons[4][column].config(bg="green")

            return True

    if buttons[0][0]['text'] == buttons[1][1]['text'] == buttons[2][2]['text'] == buttons[3][3]['text'] == buttons[4][4]['text']!= "":

        buttons[0][0].config(bg="green")

        buttons[1][1].config(bg="green")

        buttons[2][2].config(bg="green")

        buttons[3][3].config(bg="green")

        buttons[4][4].config(bg="green")

        return True

    elif buttons[0][4]['text'] == buttons[1][3]['text'] == buttons[2][2]['text'] == buttons[3][1]['text'] == buttons[4][0]['text']!= "":

        buttons[0][4].config(bg="green")

        buttons[1][3].config(bg="green")

        buttons[2][2].config(bg="green")

        buttons[3][1].config(bg="green")

        buttons[4][0].config(bg="green")

        return True

    elif empty\_spaces() is False:

        for row in range(5):

            for column in range(5):

                buttons[row][column].config(bg="yellow")

        return "Tie"

    else:

        return False

def empty\_spaces():

    spaces = 25

    for row in range(5):

        for column in range(5):

            if buttons[row][column]['text'] != "":

                spaces -= 1

    if spaces == 0:

        return False

    else:

        return True

def new\_game():

    global player

    player = random.choice(players)

    label.config(text=player+" turn")

    for row in range(5):

        for column in range(5):

            buttons[row][column].config(text="",bg="#F0F0F0")

window = Tk()

window.title("Tic-Tac-Toe")

players = ["x","o"]

player = random.choice(players)

buttons = [[0,0,0,0,0],

           [0,0,0,0,0],

           [0,0,0,0,0],

           [0,0,0,0,0],

           [0,0,0,0,0]]

label = Label(text=player + " turn", font=('consolas',40))

label.pack(side="top")

reset\_button = Button(text="restart", font=('consolas',20), command=new\_game)

reset\_button.pack(side="bottom")

frame = Frame(window)

frame.pack()

for row in range(5):

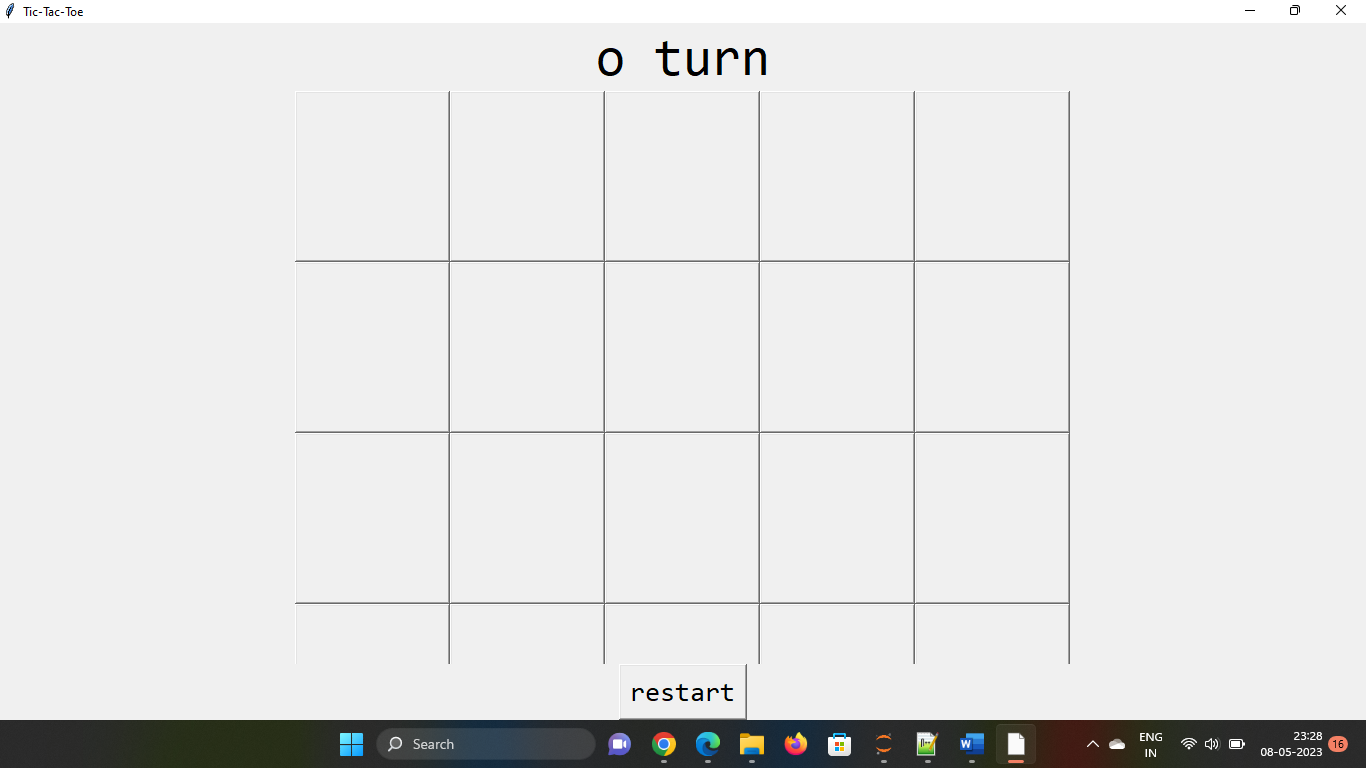
    for column in range(5):

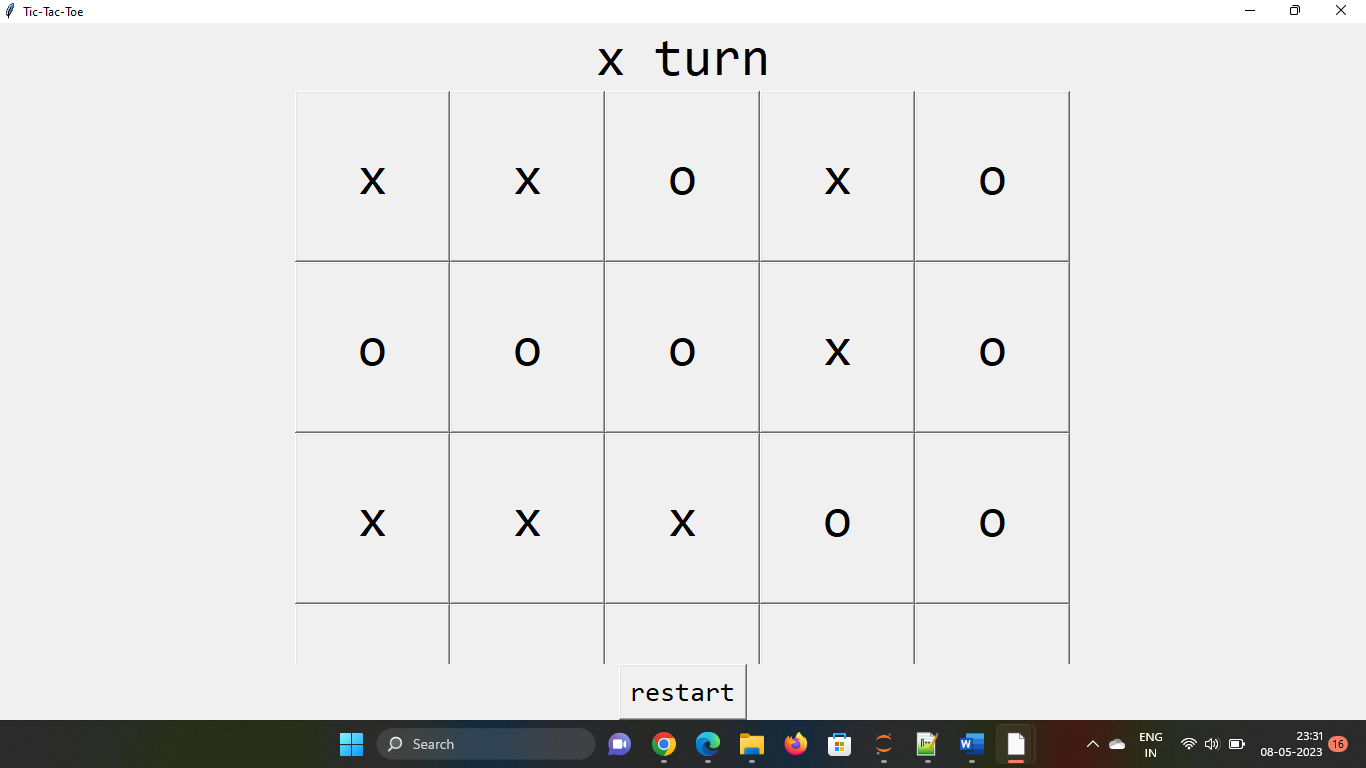
        buttons[row][column] = Button(frame, text="",font=('consolas',40), width=5, height=2,

                                      command= lambda row=row, column=column: next\_turn(row,column))

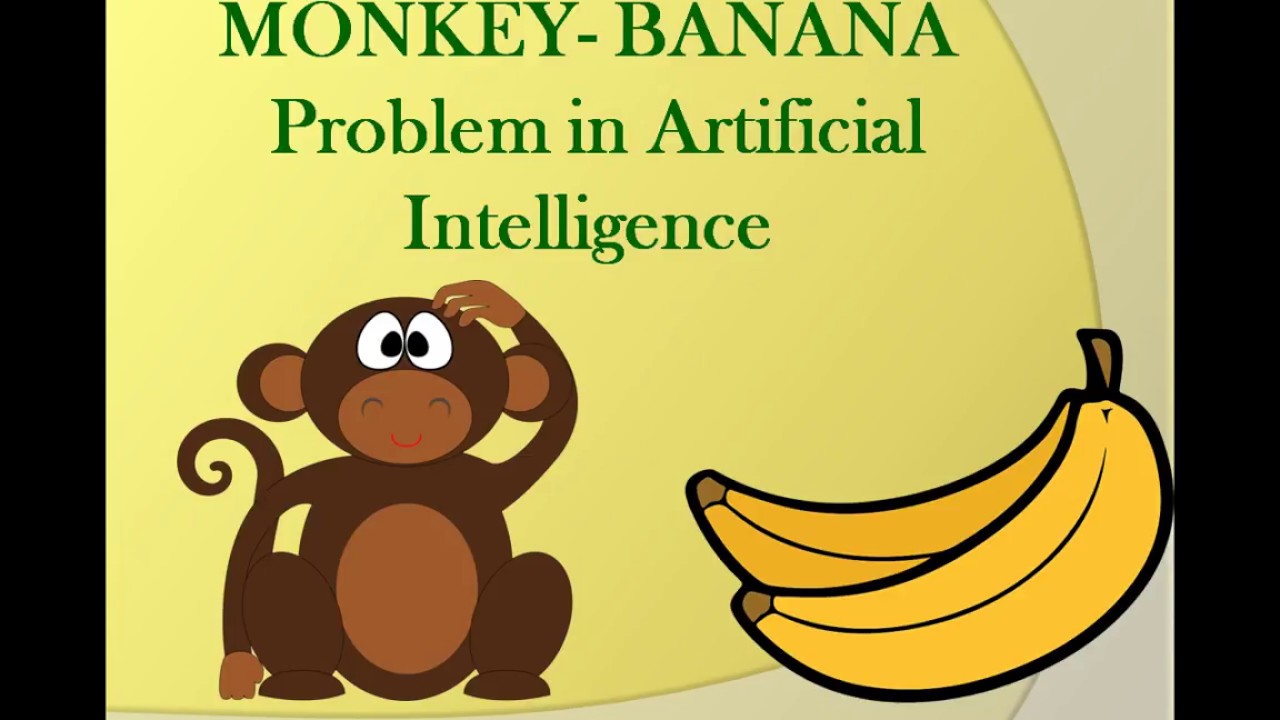
        buttons[row][column].grid(row=row,column=column)

window.mainloop()





**Aim:-Implementation of Monkey Banana Problem using LISP/PROLOG**

on(floor,monkey).

on(floor,box).

in(room,monkey).

in(room,box).

at(ceiling,banana).

strong(monkey).

grasp(monkey).

climb(monkey,box).

push(monkey,box):-

strong(monkey).

under(banana,box):-

push(monkey,box).

canreach(banana,monkey):-

at(floor,banana);

at(ceiling,banana),

under(banana,box),

climb(monkey,box).

canget(banana,monkey):-

canreach(banana,monkey),

grasp(monkey).

o/p:-

?-[‘E:/monkey.pl].

True

?- canget(banana,monkey):-

True.