| Experiment no.: 1a | Dataset creation |
| --- | --- |
| Date: |

## 

## 

## Aim:

The Objective of this experiment is to create a dataset using numpy, pandas package in python and export it as CSV file.

## Algorithm:

1. Import the NumPy and Pandas libraries.
2. Create a NumPy array of cricket player data.
3. Convert the NumPy array to a Pandas DataFrame and assign column names.
4. Calculate the total score for each player based on the number of 6s, 4s, 2s, and 1s they scored.
5. Add the calculated total score as a new column to the DataFrame.
6. Export the DataFrame to a CSV file.

## Program:

import numpy as np

import pandas as pd

dataArray=np.array([["Dhoni",10,6,5,10],["kohli",15,3,8,5],["Rohit",15,3,8,5],

["Suresh",8,18,5],["Sandy",0,3,8,5],["gokul",15,3,8,5],["Rahul",1,1,8,5],

["Sam",1,3,8,5],["Ram",4,3,8,5],["Pravin",3,3,8,5]])

df=pd.DataFrame(dataArray,columns=['Name','6s','4s','2s','1s'])

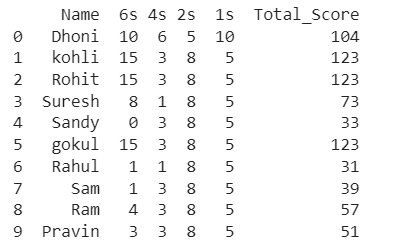
df['Total\_Score']=6\*(df['6s'].astype(int))+4\*(df['4s'].astype(int))+

2\*(df['2s'].astype(int))+1\*(df['1s'].astype(int))

print(df)

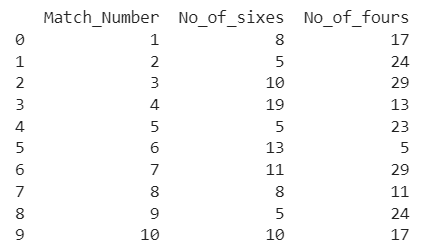
df.to\_csv('cricket.csv')

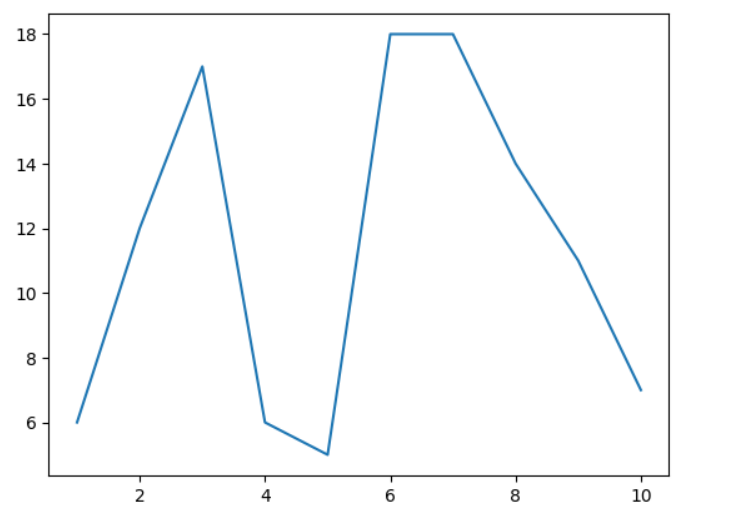
## Output:

****

## 

## Output:

****

****

| **Experiment no.: 1b** | Random data set creation and visualization |
| --- | --- |
| **Date:** |

## 

## Aim:

The Objective of this experiment is to generate random dataset and to visualize it in various forms of data representation.

## Algorithm:

1. Import the Pandas and NumPy libraries and the Matplotlib pyplot module
2. Create a Pandas DataFrame with columns for match number, number of sixes, and number of fours
3. Populate the number of sixes and number of fours columns with random integers using the NumPy randint() function
4. Print the DataFrame
5. Export the DataFrame to a CSV file using the to\_csv() method.

## Program:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

df= pd.DataFrame({"Match\_Number" : range(1,11),

"No\_of\_sixes" : np.random.randint(5, 20, size=10),

"No\_of\_fours" : np.random.randint(5, 30, size=10)

})

print(df)

df.to\_csv('sport.csv')

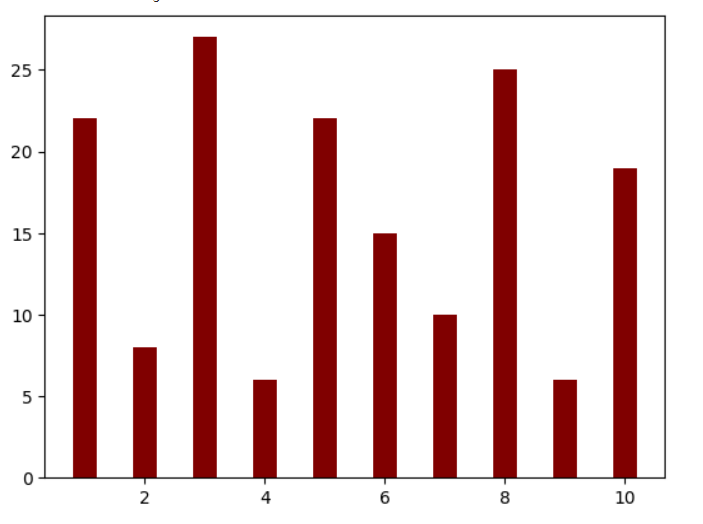
plt.plot(df['Match\_Number'],df[ "No\_of\_sixes"])

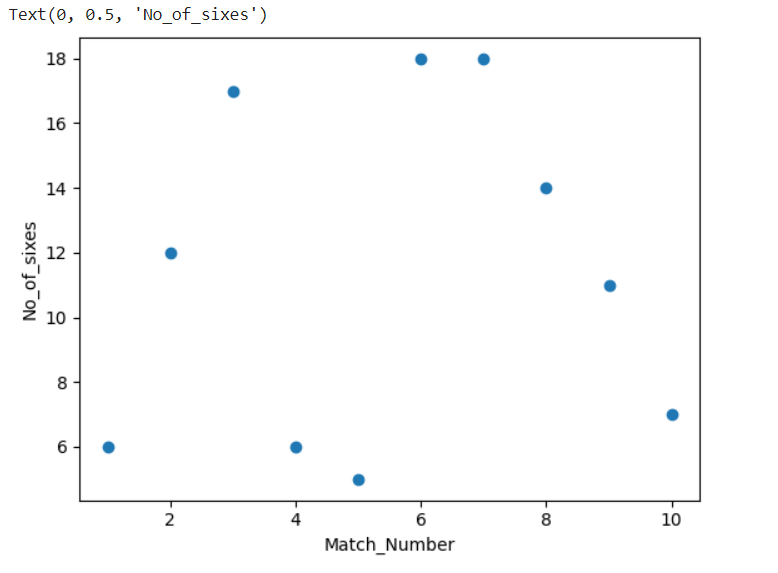
plt.bar(df['Match\_Number'],df[ "No\_of\_fours"], color ='maroon',width = 0.4)

plt.scatter(df['Match\_Number'],df[ "No\_of\_sixes"])

plt.xlabel('Match\_Number')

plt.ylabel('No\_of\_sixes')

****

****

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

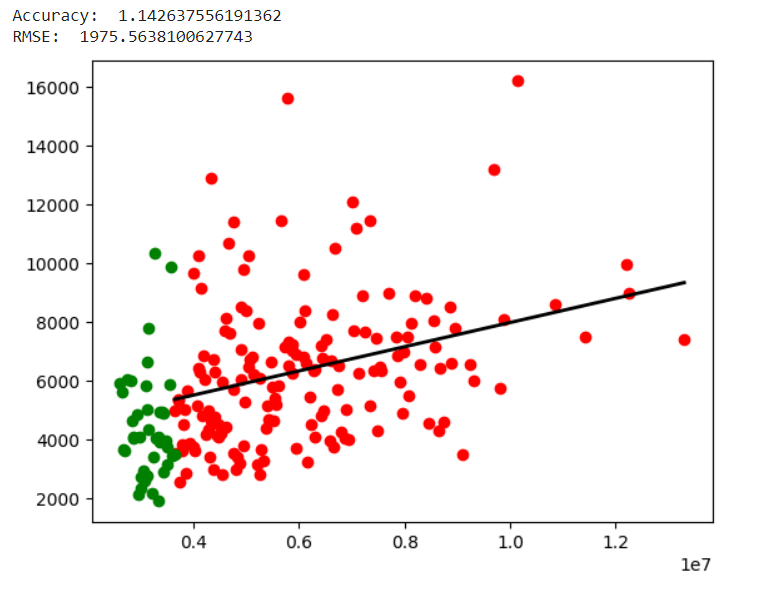
## 

## Result:

Thus the python program to create dataset is executed and verified successfully.

## 

## Output:

****

| **Experiment no.: 2a** | Implementation of Simple linear regression |
| --- | --- |
| **Date:** |

## Aim:

The Objective of this experiment is to implement simple linear regression for sports dataset.

## Algorithm:

1. Import the Matplotlib pyplot module and the Pandas library.
2. Read data from a CSV file named "stadium.csv" using the read\_csv() function and assign the data to a Pandas DataFrame.
3. Extract columns from the DataFrame and convert them to lists.
4. Calculate the regression line coefficients using the extracted data.
5. Use the regression line coefficients to predict y values for the given x values.
6. Calculate the accuracy and RMSE of the regression line using the predicted values and test data.
7. Plot the data points, predicted values, and regression line using Matplotlib's scatter() and plot() functions.
8. Show the plot using Matplotlib's show() function.

## Program:

import matplotlib.pyplot as plt

import math

import pandas as p

df = p.read\_csv("stadium.csv")

x = df["price"].tolist()[:160]

y = df["area"].tolist()[:160]

sum\_x = sum(x)

sum\_y = sum(y)

xy = []

def square\_list(x):

return x\*x

for a in range(len(x)):

pro = x[a]\*y[a]

xy.append(pro)

sum\_xy = sum(xy)

sum\_x\_squared = sum(list(map(square\_list,x)))

a = (len(x) \* sum\_xy - sum\_x \* sum\_y) / (len(x) \* sum\_x\_squared - sum\_x \*\* 2)

b = (sum\_y - a \* sum\_x) / len(x)

y\_pred = [a\*x[i] + b for i in range(len(x))]

x\_test = df["price"].tolist()[160:200]

y\_test = df["area"].tolist()[160:200]

y\_test\_pred = [a \* x\_test[i] + b for i in range(len(x\_test))]

numerator=sum((y\_test[i] - y\_test\_pred[i])\*\*2 for i in range(len(x\_test)))

denominator=sum((y\_test[i] - sum(y\_test) / len(x\_test))\*\*2 for i in range(len(x\_test)))

accuracy = numerator / denominator

rmse = math.sqrt(sum((y\_test[i] - y\_test\_pred[i]) \*\* 2 for i in range(len(x\_test))) / len(x\_test))

print("Accuracy: ", accuracy)

print("RMSE: ",rmse)

plt.scatter(x, y, color='red')

plt.scatter(x\_test, y\_test, color='green')

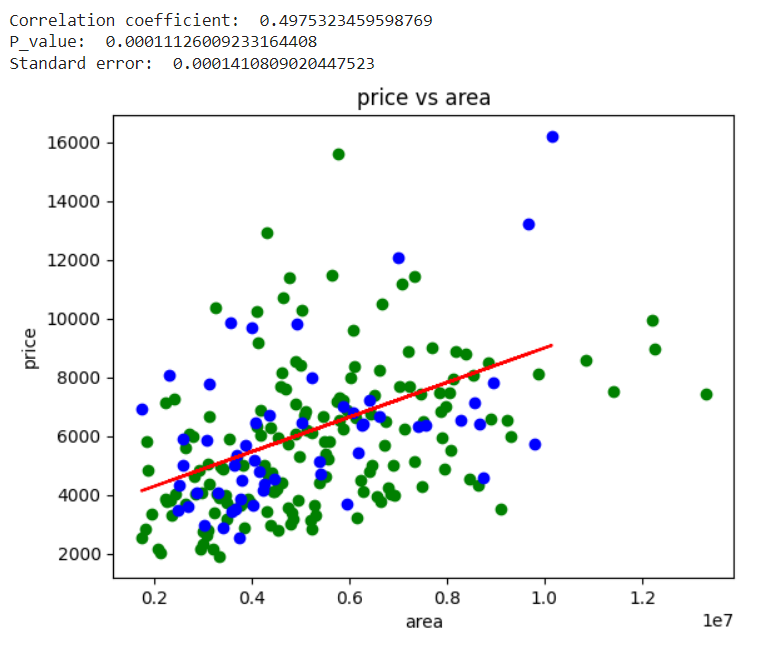
plt.plot(x, y\_pred, color='black', linewidth=2)

plt.show()

## 

## 

## Output:

****

| **Experiment no.: 2b** | Simple Linear Regression using inbuilt libraries |
| --- | --- |
| **Date:** |

## Aim:

The objective of the experiment is to implement simple linear regression using inbuild functions in python.

## Algorithm:

1. Import necessary libraries: numpy, sklearn.linear\_model, pandas, sklearn.model\_selection, matplotlib.pyplot, and scipy.stats.
2. Load the stadium.csv dataset using pd.read\_csv.
3. Separate the dataset into features (x) and target (y) variables.
4. Split the dataset into training and testing sets using train\_test\_split.
5. Calculate the linear regression parameters (slope, intercept, r-value, p-value, and standard error) using stats.linregress.
6. Define a function that takes x as input and returns the predicted y using the calculated parameters.
7. Initialize a LinearRegression model and fit it to the training data using fit.
8. Predict the target variable for the test and training data using predict.
9. Plot the training and test data points along with the predicted values using scatter and plot from matplotlib.pyplot.
10. Display the correlation coefficient, p-value, and standard error using print.
11. Display the plot using show.

## 

## Program:

import numpy as np

from sklearn.linear\_model import LinearRegression

import pandas as pd

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

import matplotlib.pyplot as mtp

from scipy import stats

df = pd.read\_csv("stadium.csv")

x = np.array(df["price"])

y = np.array(df["area"])

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)

slope,intercept,r,p,std\_err=stats.linregress(x\_test,y\_test)

def myfunc(x):

return slope\*x+intercept

model=LinearRegression()

model.fit(x.reshape(-1,1),y)

mymodel=list(map(myfunc,x\_test))

y\_pred= model.predict(x\_test.reshape(-1,1))

x\_pred= model.predict(x\_train.reshape(-1,1))

mtp.scatter(x\_train, y\_train, color="green")

mtp.scatter(x\_test,y\_test,color='blue')

mtp.plot(x\_test, mymodel, color="red")

mtp.title("price vs area")

mtp.xlabel("area")

mtp.ylabel("price")

print("Correlation coefficient: ",r)

print("P\_value: ",p)

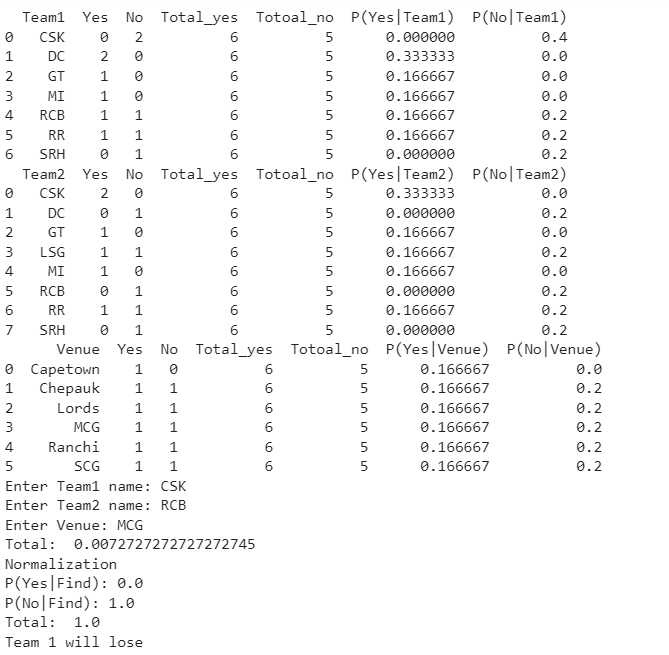
print("Standard error: ",std\_err)

mtp.show()

## Result:

Thus, Simple linear regression is implemented using inbuilt libraries and verified successfully.

## Output:

****

| **Experiment no.: 3a** | Implement classification without using inbuilt libraries |
| --- | --- |
| **Date:** |

## Aim:

The objective of this experiment is to implement classification without using inbuilt libraries in python.

## Algorithm:

1. Import necessary libraries: math, pandas.
2. Read data from the "Average.csv" file using pandas.
3. Group data based on Team1, Team2, and Venue using pandas' groupby function.
4. Calculate probabilities of yes and no for each group.
5. Take input from the user for Team1, Team2, and Venue.
6. Filter the data for Team1, Team2, and Venue respectively.
7. Calculate conditional probabilities based on the filtered data.
8. Calculate the probability of Team1 winning the match using Bayes' theorem.
9. Normalize the probabilities obtained in the previous step.
10. If the probability of Team1 winning is greater than or equal to the probability of losing, print "Team 1 can win" else print "Team 1 will lose".

## Program:

import math

import pandas as pd

data=pd.read\_csv('Average.csv')

counts=data.groupby(['Team1','Team1\_Status']).size().unstack(fill\_value=0)

counts1=data.groupby(['Team2','Team1\_Status']).size().unstack(fill\_value=0)

counts2=data.groupby(['Venue','Team1\_Status']).size().unstack(fill\_value=0)

status\_count=counts.sum(axis=0)

p\_yes=counts['Yes']/status\_count['Yes']

p\_no=counts['No']/status\_count['No']

p\_yes1=counts1['Yes']/status\_count['Yes']

p\_no1=counts1['No']/status\_count['No']

p\_yes2=counts2['Yes']/status\_count['Yes']

p\_no2=counts2['No']/status\_count['No']

p\_yestotal=status\_count['Yes']/(status\_count['Yes']+status\_count['No'])

p\_nototal=status\_count['No']/(status\_count['Yes']+status\_count['No'])

result=pd.DataFrame({'Team1':counts.index,'Yes':counts['Yes'],'No':counts['No'],

'Total\_yes':status\_count['Yes'],'Totoal\_no':status\_count['No'],

'P(Yes|Team1)':p\_yes,'P(No|Team1)':p\_no})

result=result.reset\_index(drop=True)

result1=pd.DataFrame({'Team2':counts1.index,'Yes':counts1['Yes'],'No':counts1['No'],

'Total\_yes':status\_count['Yes'],'Totoal\_no':status\_count['No'],

'P(Yes|Team2)':p\_yes1,'P(No|Team2)':p\_no1})

result1=result1.reset\_index(drop=True)

result2=pd.DataFrame({'Venue':counts2.index,'Yes':counts2['Yes'],'No':counts2['No'],

'Total\_yes':status\_count['Yes'],'Totoal\_no':status\_count['No'],

'P(Yes|Venue)':p\_yes2,'P(No|Venue)':p\_no2})

result2=result2.reset\_index(drop=True)

print(result)

print(result1)

print(result2)

Team1=input('Enter Team1 name: ')

Team2=input('Enter Team2 name: ')

Venue=input('Enter Venue: ')

res\_filtered=result[(result['Team1']==Team1)]

res1\_filtered=result1[(result1['Team2']==Team2)]

res2\_filtered=result2[(result2['Venue']==Venue)]

p\_yesTeam1=res\_filtered['P(Yes|Team1)'].values[0]

p\_yesTeam2=res1\_filtered['P(Yes|Team2)'].values[0]

p\_yesVenue=res2\_filtered['P(Yes|Venue)'].values[0]

p\_noTeam1=res\_filtered['P(No|Team1)'].values[0]

p\_noTeam2=res1\_filtered['P(No|Team2)'].values[0]

p\_noVenue=res2\_filtered['P(No|Venue)'].values[0]

p\_yesfind=p\_yesTeam1\*p\_yesTeam2\*p\_yesVenue\*p\_yestotal

p\_nofind=p\_noTeam1\*p\_noTeam2\*p\_noVenue\*p\_nototal

total = p\_yesfind+p\_nofind

print("Total: ",total)

while(math.isclose(total,1.0,abs\_tol=0.001)==False):

print("Normalization")

p\_yesfind=p\_yesfind/total

print("P(Yes|Find):",p\_yesfind)

p\_nofind=p\_nofind/total

print("P(No|Find):",p\_nofind)

total=p\_yesfind+p\_nofind

print('Total: ',total)

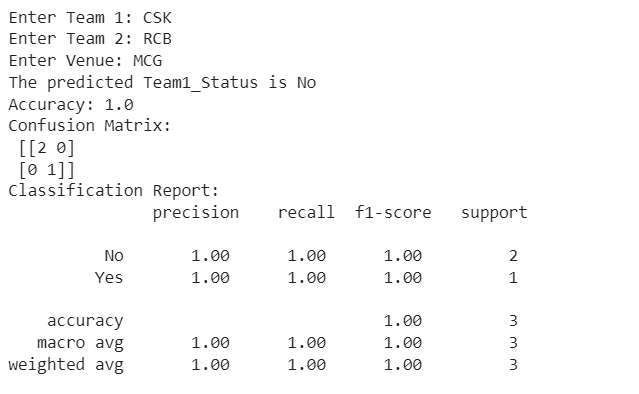
if(p\_yesfind>=p\_nofind):

print("Team 1 can win")

else:

print("Team 1 will lose")

## Output:

****

| **Experiment no.: 3b** | Implement classification with using inbuilt libraries |
| --- | --- |
| **Date:** |

## Aim:

The objective is to implement simple classification using inbuilt libraries in python.

## Algorithm:

1. Import the necessary libraries: pandas, sklearn.model\_selection.train\_test\_split, sklearn.metrics.accuracy\_score, sklearn.metrics.confusion\_matrix, sklearn.metrics.classification\_report, and sklearn.naive\_bayes.GaussianNB.
2. Read the data from a CSV file named 'Average.csv' into a pandas DataFrame called 'data'.
3. Preprocess the data by using one-hot encoding to convert categorical variables ('Team1', 'Team2', and 'Venue') into binary columns.
4. Split the data into training and testing sets using train\_test\_split.
5. Create an instance of the GaussianNB classifier and fit it to the training data.
6. Prompt the user to input values for 'Team1', 'Team2', and 'Venue'.
7. Create a new DataFrame with the user's input and one-hot encode the categorical variables using the same format as before.
8. Use the fitted GaussianNB model to predict the outcome of the match using the user's input.
9. Calculate the accuracy, confusion matrix, and classification report for the model's predictions on the testing data.
10. Print the predicted outcome of the match and the model's performance metrics.

## Program:

import pandas as pd

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

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.naive\_bayes import GaussianNB

data = pd.read\_csv('Average.csv')

data = pd.get\_dummies(data, columns=['Team1', 'Team2', 'Venue'])

df = pd.read\_csv('Average.csv')

X = pd.get\_dummies(df.drop('Team1\_Status', axis=1))

y = df['Team1\_Status']X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.drop('Team1\_Status', axis=1), data['Team1\_Status'], test\_size=0.2, random\_state=42)

model = GaussianNB()

model.fit(X, y)

team1 = input("Enter Team 1: ")

team2 = input("Enter Team 2: ")

venue = input("Enter Venue: ")

X\_new = pd.DataFrame({'Team1\_' + team1: [1], 'Team2\_' + team2: [1],

'Venue\_' + venue: [1]})

X\_new = X\_new.reindex(columns=X.columns, fill\_value=0)

y\_new = model.predict(X\_new)

y\_pred = model.predict(X\_test)

print(f"The predicted Team1\_Status is {y\_new[0]}")

accuracy = accuracy\_score(y\_test, y\_pred)

confusion\_mat = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

print('Accuracy:', accuracy)

print('Confusion Matrix:\n', confusion\_mat)

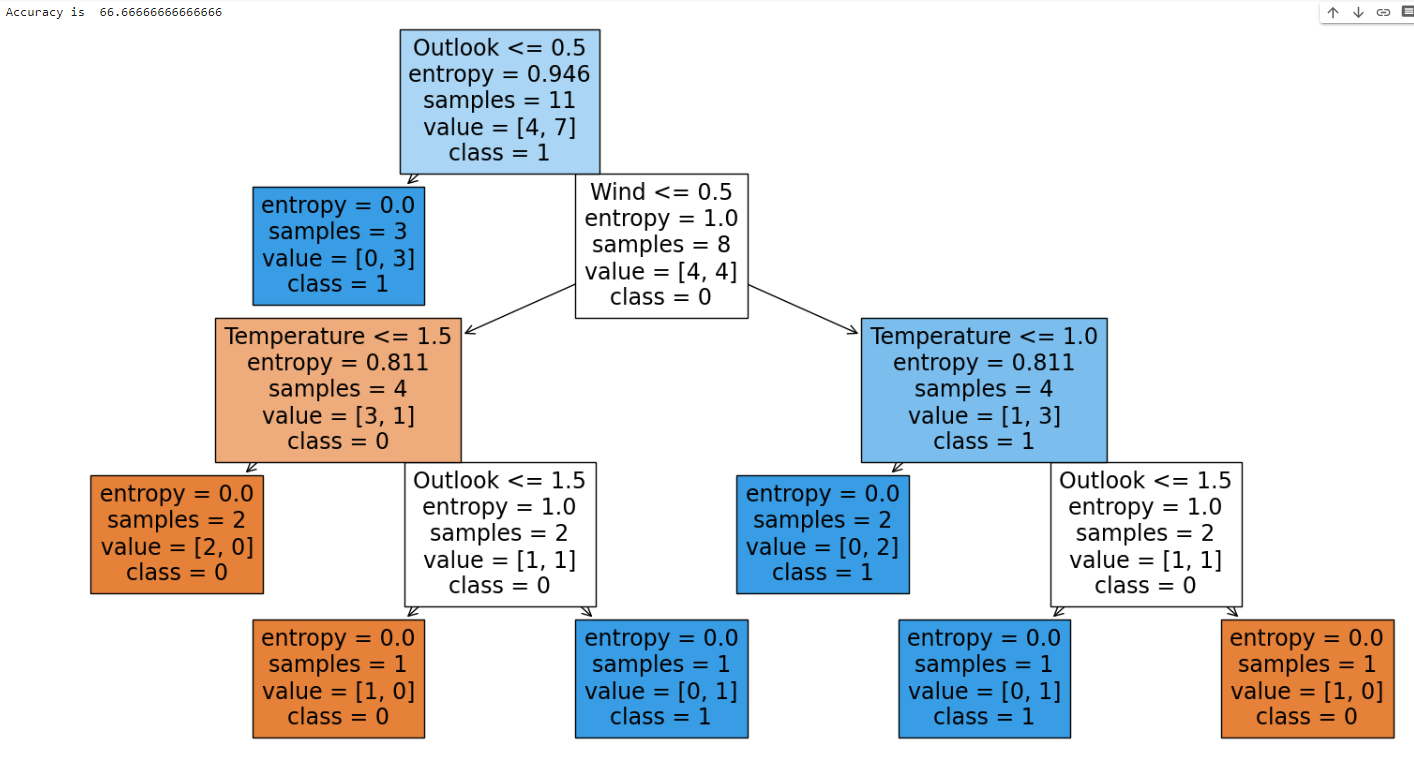
print('Classification Report:\n', class\_report)

## Result:

Thus Classification is implemented with and without using inbuilt libraries and it is verified successfully.

## Output:

Accuracy is 66.666

****

| **Experiment no.: 4a** | Implementation of Decision Tree using ID3 algorithm |
| --- | --- |
| **Date:** |

## Aim:

The Objective of this experiment is to implement decision tree using ID3 algorithm with inbuilt libraries in python.

## Algorithm:

1. Import necessary libraries: pandas, matplotlib, sklearn.tree, sklearn.preprocessing, sklearn.model\_selection, and sklearn.metrics.
2. Read in the CSV file ('PlayTennis.csv') using pandas and assign the data to a DataFrame variable called 'df'.
3. Separate the predictor features from the target variable by dropping the 'Play' column and assigning the remaining columns to a new variable 'X', and assigning the 'Play' column to a new variable 'y'.
4. Encode the categorical predictor features in 'X' using LabelEncoder.
5. Split the preprocessed data into training and testing sets using train\_test\_split, with a test size of 0.2 and a random state of 42.
6. Initialize a decision tree classifier with entropy as the criterion.
7. Train the decision tree classifier on the training set using fit method.
8. Use the trained classifier to make predictions on the test set using predict method and assign the predictions to 'y\_pred'.
9. Evaluate the accuracy of the classifier on the test set using accuracy\_score method and print the result.
10. Visualize the trained decision tree using plot\_tree method with feature names from 'X.columns' and class names ['0', '1'], and show the plot using plt.show().

## Program:

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.tree import plot\_tree

from sklearn.tree import DecisionTreeClassifier

from sklearn.preprocessing import LabelEncoder

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

from sklearn.metrics import accuracy\_score

df = pd.read\_csv('PlayTennis.csv')

X = df.drop('Play', axis=1)

y = df['Play']

le = LabelEncoder()

X = X.apply(le.fit\_transform)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

clf = DecisionTreeClassifier(criterion='entropy')

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

print("Accuracy is ", accuracy\_score(y\_test,y\_pred)\*100)

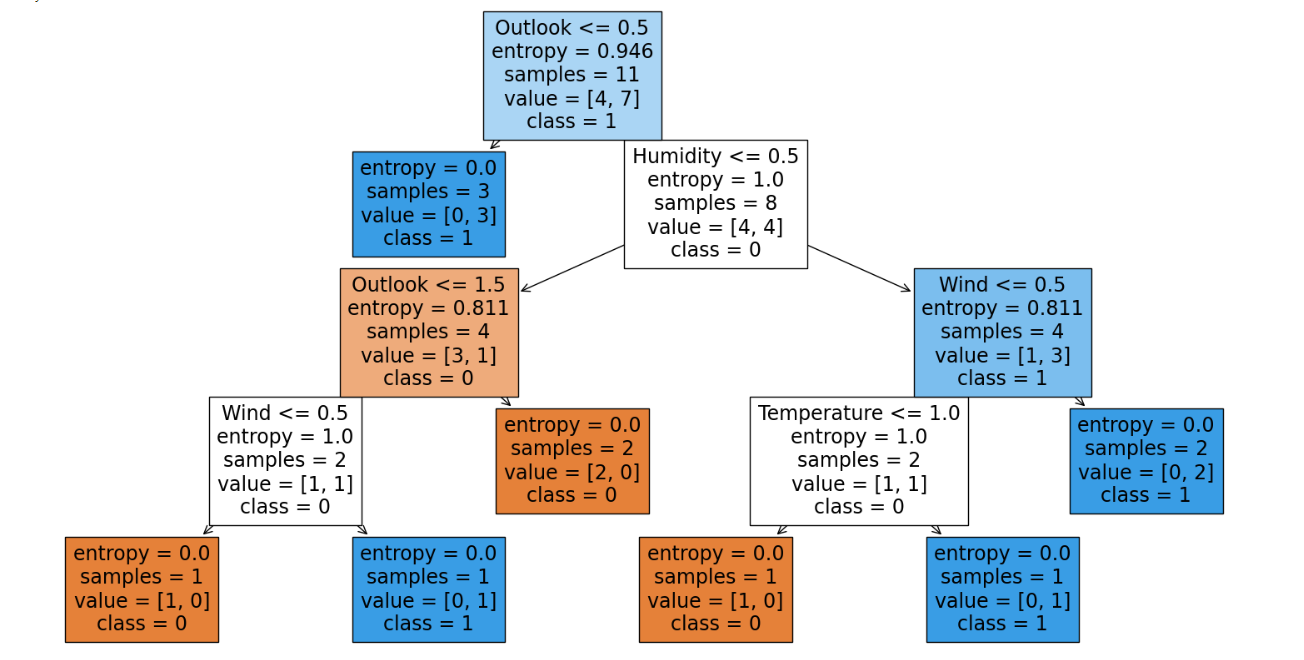
plt.figure(figsize=(20,10))

plot\_tree(clf, filled=True, feature\_names=X.columns, class\_names=['0', '1'])

plt.show()

## Output:

Accuracy is 72.333

****

| **Experiment no.: 4b** | Implementation of Decision Tree using C4.5 algorithm |
| --- | --- |
| **Date:** |

## Aim:

The Objective of this experiment is to implement decision tree using C4.5 algorithm with inbuilt library function in python.

## Algorithm:

1. Import necessary libraries: pandas, matplotlib, sklearn.tree, sklearn.preprocessing, sklearn.model\_selection, and sklearn.metrics.
2. Read in the CSV file ('PlayTennis.csv') using pandas and assign the data to a DataFrame variable called 'df'.
3. Separate the predictor features from the target variable by dropping the 'Play' column and assigning the remaining columns to a new variable 'x', and assigning the 'Play' column to a new variable 'y'.
4. Encode the categorical predictor features in 'x' using LabelEncoder.
5. Split the preprocessed data into training and testing sets using train\_test\_split, with a test size of 0.2 and a random state of 42.
6. Initialize a decision tree classifier with entropy as the criterion and 'best' as the splitter.
7. Train the decision tree classifier on the training set using fit method.
8. Use the trained classifier to make predictions on the test set using predict method and assign the predictions to 'y\_pred'.
9. Evaluate the accuracy of the classifier on the test set using accuracy\_score method and print the result.
10. Visualize the trained decision tree using plot\_tree method with feature names from 'x.columns' and class names ['0', '1'], and show the plot using plt.show().

## Program:

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.tree import plot\_tree

from sklearn.tree import DecisionTreeClassifier

from sklearn.preprocessing import LabelEncoder

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

from sklearn.metrics import accuracy\_score

df = pd.read\_csv('PlayTennis.csv')

x = df.drop('Play', axis=1)

y = df['Play']

le = LabelEncoder()

x = x.apply(le.fit\_transform)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

clf = DecisionTreeClassifier(criterion='entropy', splitter='best')

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

y\_pred = clf.predict(x\_test)

print("Accuracy is ", accuracy\_score(y\_test,y\_pred)\*100)

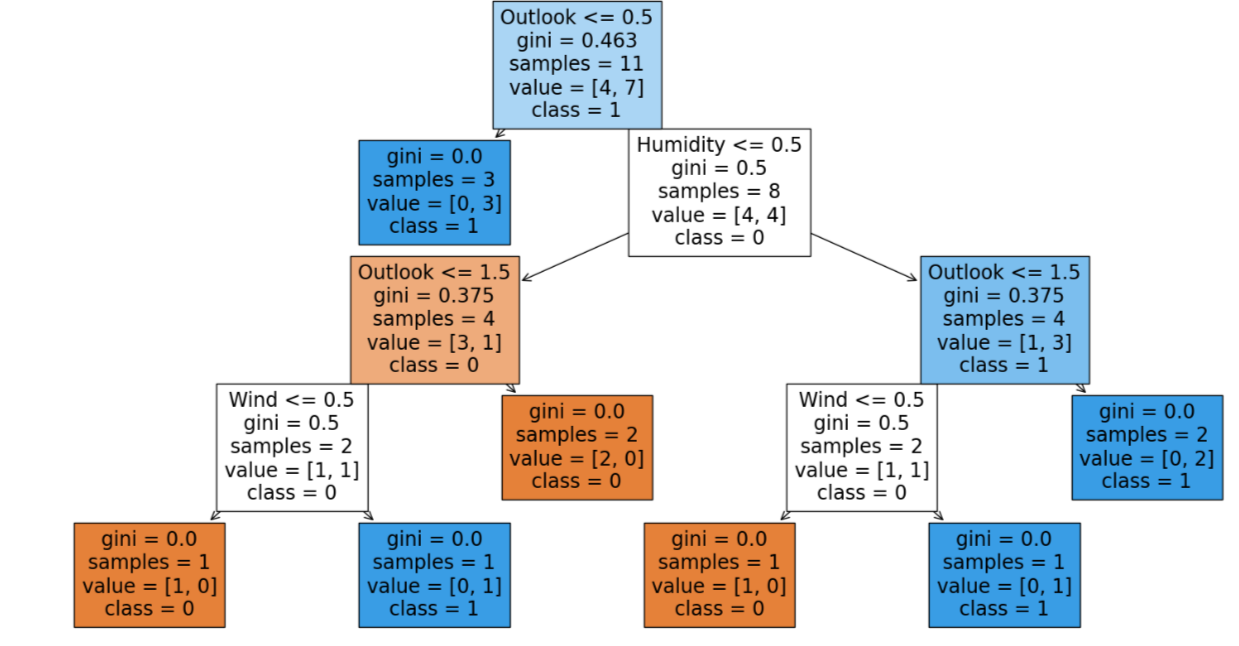
plt.figure(figsize=(20,10))

plot\_tree(clf, filled=True, feature\_names=x.columns, class\_names=['0', '1'])

plt.show()

## Output:

Accuracy is 87.222

****

| **Experiment no.: 4c** | Implementation of Decision Tree using CART algorithm |
| --- | --- |
| **Date:** |

## Aim:

The Objective of this experiment is to implement decision tree using CART algorithm with inbuilt libraries in python.

## Algorithm:

1. Import necessary libraries: matplotlib.pyplot, sklearn.tree, pandas, sklearn.preprocessing, sklearn.model\_selection, sklearn.metrics.
2. Load the PlayTennis.csv dataset using pd.read\_csv.
3. Separate the dataset into features (x) and target (y) variables.
4. Encode the categorical features using LabelEncoder.
5. Split the dataset into training and testing sets using train\_test\_split.
6. Initialize the DecisionTreeClassifier with criterion='gini' and max\_depth=None.
7. Fit the classifier to the training data using fit.
8. Predict the target variable for the test data using predict.
9. Calculate the accuracy of the model using accuracy\_score.
10. Visualize the decision tree using plot\_tree and matplotlib.pyplot.

## Program:

import matplotlib.pyplot as plt

from sklearn.tree import plot\_tree

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.preprocessing import LabelEncoder

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

from sklearn.metrics import accuracy\_score

df = pd.read\_csv('PlayTennis.csv')

x = df.drop('Play', axis=1)

y = df['Play']

le = LabelEncoder()

x = x.apply(le.fit\_transform)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

clf = DecisionTreeClassifier(criterion='gini', max\_depth=None)

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

y\_pred = clf.predict(x\_test)

print("Accuracy is ", accuracy\_score(y\_test,y\_pred)\*100)

plt.figure(figsize=(20,10))

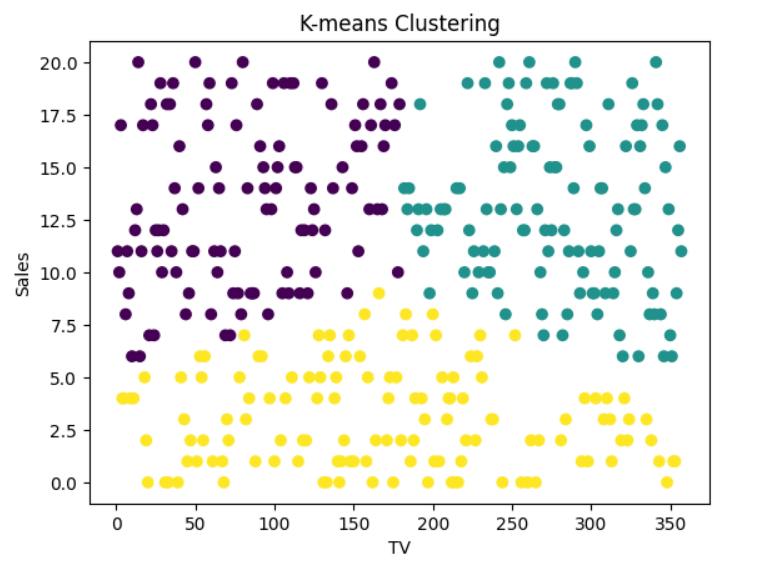
plot\_tree(clf, filled=True, feature\_names=x.columns, class\_names=['0', '1'])

plt.show()

## Result:

Thus, Decision tree is implemented using ID3, C4.5 and CART algorithm and verified successfully.

## Output:

****

| **Experiment no.: 5a** | K – Means Clustering |
| --- | --- |
| **Date:** |

## Aim:

The Objective of this experiment is to implement K means clustering algorithm in sports dataset with inbuilt libraries in python.

## Algorithm:

1. Import the required libraries: pandas, sklearn.cluster.KMeans, sklearn.preprocessing.MinMaxScaler, matplotlib.pyplot.
2. Load the dataset from "Overrun.csv".
3. Scale the data using MinMaxScaler.
4. Perform K-means clustering with 3 clusters.
5. Assign cluster labels to the data.
6. Visualize the results using a scatter plot.

## Program:

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt

data = pd.read\_csv("Overrun.csv")

scaler = MinMaxScaler()

data\_scaled = scaler.fit\_transform(data)

kmeans = KMeans(n\_clusters=3, random\_state=0).fit(data\_scaled)

data["cluster"] = kmeans.labels\_

plt.scatter(data.iloc[:, 0], data.iloc[:, 1], c=data["cluster"])

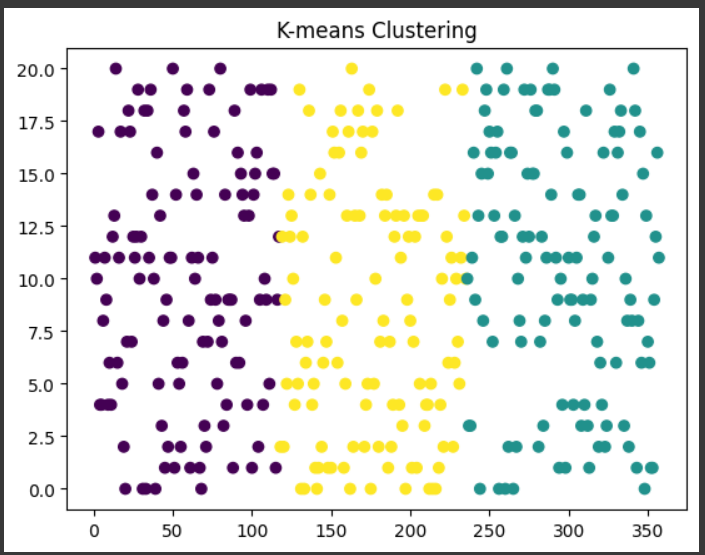
plt.xlabel("TV")

plt.ylabel("Sales")

plt.title("K-means Clustering")

plt.show()

## Output:



| **Experiment no.: 5b** | K – Means Clustering |
| --- | --- |
| **Date:** |

## Aim:

The Objective of this experiment is to implement K-means clustering without using inbuilt libraries in python.

## Algorithm:

1. Import the required libraries: pandas, numpy, random, matplotlib.pyplot.
2. Load the dataset from "Overrun.csv" using pandas.
3. Convert the data to a numpy array.
4. Set the number of clusters (K) and maximum iterations (max\_iters).
5. Initialize random centroids from the data.
6. Iterate for the maximum number of iterations:
   * Compute the distances between data points and centroids.
   * Assign data points to the nearest centroid.
   * Update the centroids by computing the mean of the data points in each cluster.
7. Visualize the results using a scatter plot.

## Program:

import pandas as pd

import numpy as np

import random

import matplotlib.pyplot as plt

data = pd.read\_csv("Overrun.csv")

X = np.array(data)

K = 3

max\_iters = 100

centroids = X[random.sample(range(len(X)), K)]

for i in range(max\_iters):

distances = np.sqrt(((X - centroids[:, np.newaxis])\*\*2).sum(axis=2))

clusters = np.argmin(distances, axis=0)

for k in range(K):

centroids[k] = X[clusters == k].mean(axis=0)

plt.scatter(X[:, 0], X[:, 1], c=clusters)

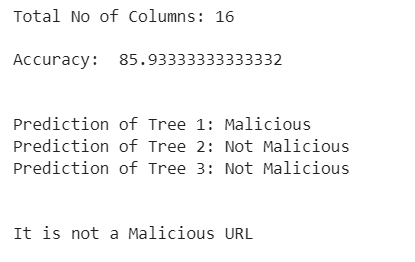
plt.title("K-means Clustering")

plt.show()

## Result:

Thus, K-means clustering is implemented with and without libraries and verified successfully.

## Output:

****

| **Experiment no.: 6a** | Ensemble Learning:Bagging - Random Forest algorithm |
| --- | --- |
| **Date:** |

## Aim:

The Objective of this experiment is to implement Random forest algorithm using inbuilt libraries in python.

## Algorithm:

1. Import libraries: pandas, sklearn.ensemble, sklearn.model\_selection, and sklearn.metrics.
2. Read data from CSV into df.
3. Split data into input features (X) and target variable (y).
4. Split data into train and test sets.
5. Create a RandomForestClassifier (rf) with desired parameters.
6. Fit rf to the training data.
7. Predict target variable for test data.
8. Calculate and print accuracy.
9. Define example test data (test\_data).
10. Predict class for test\_data using each tree in rf.
11. Store predictions in a list.
12. Determine final prediction by majority voting.
13. Print if URL is malicious or not.

## Program:

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

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

from sklearn import metrics

df = pd.read\_csv("url.csv")

X = df.iloc[:, 1:-1].values

y = df.iloc[:, -1].values

print("Total No of Columns:",X.shape[1])

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

rf = RandomForestClassifier(n\_estimators=3, random\_state=42)

rf.fit(X\_train, Y\_train)

Y\_pred = rf.predict(X\_test)

accuracy\_value = metrics.accuracy\_score(Y\_test, Y\_pred)

print("\nAccuracy: ", accuracy\_value\*100)

test\_data = [[0,0,1,5,0,0,0,0,0,1,1,1,0,0,1,0]] # Example test data

predictions = []

print('\n')

for tree in rf.estimators\_:

prediction = tree.predict(test\_data)

predictions.append(prediction[0])

print(f"Prediction of Tree {rf.estimators\_.index(tree) + 1}: {'Not Malicious' if prediction[0]==0 else 'Malicious'}")

final\_prediction = int(max(set(predictions), key=predictions.count))

print('\n')

if final\_prediction == 0:

print('It is not a Malicious URL')

else:

print('It is a Malicious URL')

## Output:

The accuracy of the model on validation set is 80.32

| **Experiment no.: 6b** | Ensemble Learning:Boosting - Adaptive Boosting |
| --- | --- |
| **Date:** |

## Aim:

The Objective of this experiment is to implement Adaptive boosting using inbuilt libraries in python.

## Algorithm:

1. Import the necessary libraries, including pandas for data manipulation and scikit-learn for model building and evaluation.
2. Read the CSV file containing the data into a pandas DataFrame.
3. Separate the input features (X) from the target variable (y).
4. Split the data into training and testing sets using the train\_test\_split function from scikit-learn.
5. Create an instance of the AdaBoostClassifier with a specified learning rate (0.3 in this case).
6. Fit the AdaBoost model to the training data using the fit method.
7. Evaluate the accuracy of the model on the validation set using the score method.
8. Print the accuracy score.

## Program:

import pandas as pd

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

from sklearn.ensemble import AdaBoostClassifier

data = pd.read\_csv("url.csv")

x = data.drop(columns=['Domain','Label'],axis=1)

y = data['Label']

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(x,y,test\_size=0.25)

adb = AdaBoostClassifier(learning\_rate=0.3)

adb\_model = adb.fit(X\_train,Y\_train)

print("The accuracy of the model on validation set is", adb\_model.score(X\_test,Y\_test))

## Output:

Accuracy of Gradient Boosted model: 86.133

| **Experiment no.: 6c** | Boosting – Gradient Boosting |
| --- | --- |
| **Date:** |

## Aim:

The objective of the experiment is to implement Gradient boosting using inbuilt libraries in python.

## Algorithm:

1. Import the required libraries.
2. Read the data into a DataFrame.
3. Split the data into training and testing sets.
4. Build a Gradient Boosting Classifier model.
5. Train the model using the training data.
6. Predict labels for the test data.
7. Evaluate the model's accuracy.
8. Print the accuracy.

## Program:

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import GradientBoostingClassifier

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

from sklearn import metrics

data = pd.read\_csv('url.csv')

X = data.drop(columns=['Domain','Label'],axis = 1)

Y = data['Label']

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.3)

GradBoost = GradientBoostingClassifier(init=tree\_model, learning\_rate=0.1, n\_estimators=200)

GradBoost.fit(X\_train, Y\_train)

Y\_pred\_boost = GradBoost.predict(X\_test)

accuracy\_boost = metrics.accuracy\_score(Y\_test, Y\_pred\_boost)

print("Accuracy of Gradient Boosted model: ", accuracy\_boost\*100)

## Output:

The accuracy of the model on validation set is 85.92

| **Experiment no.: 6d** | Boosting - LG Boosting |
| --- | --- |
| **Date:** |

## Aim:

The objective of the experiment is to implement LG boosting using inbuilt libraries in python.

## Algorithm:

1. Import the required libraries.
2. Read the data into a DataFrame.
3. Split the data into training and testing sets.
4. Build a LightGBM Classifier model.
5. Train the model using the training data.
6. Evaluate the model's accuracy on the validation set.
7. Print the accuracy.

## Program:

import pandas as pd

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

from lightgbm import LGBMClassifier

data = pd.read\_csv("url.csv")

x = data.drop(columns=['Domain','Label'],axis=1)

y = data['Label']

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(x,y,test\_size=0.25)

lgb = LGBMClassifier(learning\_rate=0.3)

lgb\_model = lgb.fit(X\_train,Y\_train)

print("The accuracy of the model on validation set is", lgb\_model.score(X\_test,Y\_test)\*100)

## Output:

The accuracy of the model on validation set is 86.0

| **Experiment no.: 6e** | Boosting - Categorical Boosting |
| --- | --- |
| **Date:** |

## Aim:

The Objective of this experiment is to implement Categorical boosting using inbuilt libraries in python.

## Algorithm:

1. Import libraries: pandas, train\_test\_split, catboost.
2. Read data from CSV file.
3. Split data into input features (x) and target variable (y).
4. Split data into train and test sets.
5. Initialize CatBoost classifier.
6. Fit the model on the training data.
7. Print the accuracy of the model on the test set.

## Program:

import pandas as pd

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

import catboost as ctb

data = pd.read\_csv("url.csv")

x = data.drop(columns=['Domain','Label'],axis=1)

y = data['Label']

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(x,y,test\_size=0.25)

catb = ctb.CatBoostClassifier()

catb\_model = catb.fit(X\_train,Y\_train)

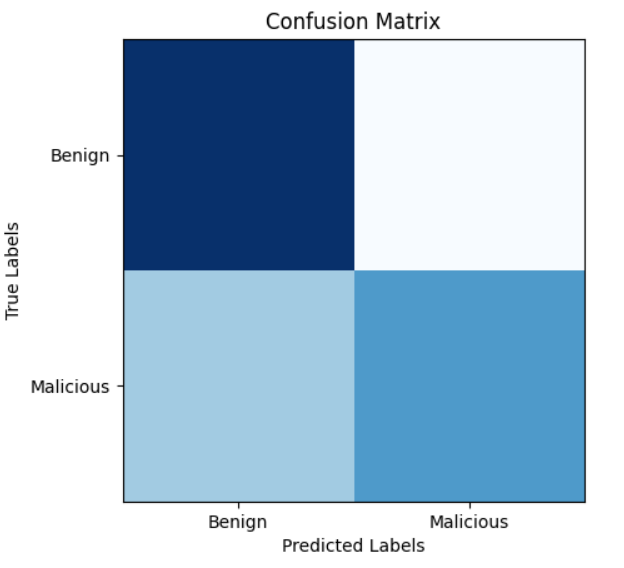
print("The accuracy of the model on validation set is", catb\_model.score(X\_test,Y\_test)\*100)

## Result:

Thus bagging and boosting methods of Ensemble learning is implemented.

## Output:

Accuracy:80.1

****

| **Experiment no.: 7** | Support Vector Machine |
| --- | --- |
| **Date:** |

## Aim:

The Objective of this experiment is to implement Support vector machine using inbuilt libraries in python.

## Algorithm:

1. Import the necessary libraries.
2. Load the dataset from a CSV file.
3. Separate the features and the target variable.
4. Split the data into training and testing sets.
5. Scale the features using StandardScaler.
6. Train an SVM model with a linear kernel and regularization parameter C=1.0.
7. Make predictions on the testing set.
8. Evaluate the model's accuracy using accuracy\_score.
9. Plot the confusion matrix using confusion\_matrix and matplotlib.pyplot.

## Program:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.preprocessing import StandardScaler

# Load the dataset

url\_data = pd.read\_csv('url.csv')

# Separate the features and target variable

X = url\_data.drop(columns=['Domain','Label'],axis=1)

y = url\_data['Label']

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Scale the features using StandardScaler

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Train the SVM model

svm = SVC(kernel='linear', C=1.0, random\_state=42)

svm.fit(X\_train, y\_train)

# Make predictions on the testing set

y\_pred = svm.predict(X\_test)

# Evaluate the model's accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print('Accuracy:', accuracy)

# Plot the confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

plt.imshow(cm, cmap=plt.cm.Blues)

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.xticks([0, 1], ['Benign', 'Malicious'])

plt.yticks([0, 1], ['Benign', 'Malicious'])

plt.title('Confusion Matrix')

plt.show()

## Result:

Thus Support vector machine is implemented and verified successfully.

## Output:

Episode 1: Total reward = -506

Episode 2: Total reward = -560

Episode 3: Total reward = -569

.

.

.

Episode 998: Total reward = -2

Episode 999: Total reward = -2

Episode 1000: Total reward = 10

| **Experiment no.: 8** | Reinforcement Learning:Q-learning |
| --- | --- |
| **Date:** |

## Aim:

The Objective of this experiment is to implement Q-learning method of reinforcement learning using inbuilt libraries in python.

## Algorithm:

1. Initialize the Q-table with zeros.
2. Repeat for a fixed number of episodes:

a. Reset the environment to start a new episode

. b. Initialize the total reward for this episode.

c. Repeat until the episode ends:

i. Choose an action using an epsilon-greedy policy.

ii. Take the chosen action and observe the next state and reward.

iii. Update the Q-table using the Q-learning update rule.

iv. Update the total reward.

v. Update the state for the next iteration.

1. Return the learned Q-table.

## Program:

import numpy as np

import gym

def q\_learning(env, num\_episodes, alpha, gamma, epsilon):

Q = np.zeros((env.observation\_space.n, env.action\_space.n))

for episode in range(num\_episodes):

state = env.reset()

total\_reward = 0

done = False

while not done:

if np.random.random() < epsilon:

action = env.action\_space.sample()

else:

action = np.argmax(Q[state])

next\_state, reward, done, \_ = env.step(action)

Q[state, action] += alpha \* (reward + gamma \* np.max(Q[next\_state]) - Q[state, action])

total\_reward += reward

state = next\_state

print(f"Episode {episode + 1}: Total reward = {total\_reward}")

return Q

env = gym.make('Taxi-v3')

num\_episodes = 1000

alpha = 0.1

gamma = 0.99

epsilon = 0.1

Q = q\_learning(env, num\_episodes, alpha, gamma, epsilon)

env.close()

## Result:

Thus, Q-learning method of reinforcement learning is executed and verified successfully.

## Output:

Epoch: 9999, Cost: 0.0021

[[0.]

[1.]

[1.]

[0.]]

| **Experiment no.: 9** | Multi Layer Perceptron |
| --- | --- |
| **Date:** |

## Aim:

The Objective of this experiment is to implement Multilayer perceptron using inbuilt libraries in python.

## Algorithm:

1. Initialize the MLP neural network with random weights and biases.
2. Perform forward propagation to calculate the output of the network.
3. Calculate the cost using the mean squared error between the predicted output and the target output.
4. Perform backward propagation to update the weights and biases based on the calculated gradients.
5. Repeat steps 2-4 for a specified number of epochs.
6. Use the trained network to predict the outputs for test input.
7. Print the predicted outputs.

## Program:

import numpy as np

import matplotlib.pyplot as plt

class MLP:

def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):

self.input\_size = input\_size

self.hidden\_size = hidden\_size

self.output\_size = output\_size

self.weights1 = np.random.randn(input\_size, hidden\_size)

self.bias1 = np.random.randn(hidden\_size)

self.weights2 = np.random.randn(hidden\_size, output\_size)

self.bias2 = np.random.randn(output\_size)

def sigmoid(self, x):

return 1 / (1 + np.exp(-x))

def forward(self, X):

self.hidden\_output = self.sigmoid(np.dot(X, self.weights1) + self.bias1)

self.output = self.sigmoid(np.dot(self.hidden\_output, self.weights2) + self.bias2)

return self.output

def backward(self, X, y, learning\_rate):

d\_output = (self.output - y) \* self.output \* (1 - self.output)

d\_hidden\_output = np.dot(d\_output, self.weights2.T) \* self.hidden\_output \* (1 - self.hidden\_output)

self.weights2 -= learning\_rate \* np.dot(self.hidden\_output.T, d\_output)

self.bias2 -= learning\_rate \* np.sum(d\_output, axis=0)

self.weights1 -= learning\_rate \* np.dot(X.T, d\_hidden\_output)

self.bias1 -= learning\_rate \* np.sum(d\_hidden\_output, axis=0)

def train(self, X, y, epochs, learning\_rate):

for i in range(epochs):

output = self.forward(X)

cost = np.mean((y - output) \*\* 2)

self.backward(X, y, learning\_rate)

# if i % 100 == 0:

print(f"Epoch: {i}, Cost: {cost:.4f}")

def predict(self, X):

output = self.forward(X)

return np.round(output)

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

y = np.array([[0], [1], [1], [0]])

mlp = MLP(2, 4, 1)

mlp.train(X, y, 10000, 0.1)

test\_input = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

test\_output = mlp.predict(test\_input)

print(test\_output)

## Result:

Thus, Multi Layer perceptron is executed and verified successfully.