DEPARTMENT OF COMPUTER APPLICATION

TKM COLLEGE OF ENGINEERING

KOLLAM – 691005

22MCA241 -DATA SCIENCE LAB

PRACTICAL RECORD BOOK Third

Semester MCA

2022-2024

Submitted by:

NAME: ADARSH CHANDRAN VA V

ROLL NO: TKM22MCA-2002

DEPARTMENT OF COMPUTER APPLICATION
TKM COLLEGE OF ENGINEERING
KOLLAM – 691005

Certificate

This is a bonafide record of the work done by **ADARSH CHANDRAN V A V (TKM22MCA-2002)** in the Third Semester in **DATA SCIENCE LAB** Course(22MCA241) towards the partial fulfillment of the degree of Master of Computer Applications during the academic year 2023- 2024.

Staff Member in-charge

Examiner

……..

……..

CONTENTS

9	Program to implement k-means clustering technique using any standard dataset available in the public domain	28
	COURSE OUTCOME 4	
10	Programs on feedforward network to classify any standard dataset available in the public domain.	32
11	Programs on convolutional neural network to classify images from any standard dataset in the public domain.	36
	COURSE OUTCOME 5	
12	Natural Language Processing Problems may be designed for the following topics so that students can get hands on experience in using python for natural language processing: • Part of Speech tagging • N-gram	38

COURSE OUTCOME 1

PROGRAM NO:1

AIM: Review of python programming-Programs review the fundamentals of python

1.Write a Python program to reverse a number and also find the sum of digits of the number. Prompt the user for input.

ALGORITHM:

Step 1: Prompt the user for a number input.

Step 2: Read and store the input as a string in user_input.

Step 3: Check if user_input consists only of digits using isdigit().

Step 4: If valid, convert user_input to an integer and store it in num.

Step 5: Reverse the integer num and store it in reversed_num.

Step 6: Calculate the sum of the digits of num and store it in sum_of_digits.

Step 7: Display the reversed number and the sum of digits.

Step 8: If the input is not valid, print an error message.

Step 9: Call the function reverse_and_sum_digits() to execute the code.

PROGRAM CODE:

```
def reverse_and_sum_digits():

    # Prompt user for input

    user_input = input("Enter a number: ")



    # Check if the input is a valid integer

    if user_input.isdigit():

        num = int(user_input)
```

```
        # Reverse the number
        reversed_num = int(str(num)[::-1])

        # Calculate the sum of digits
        sum_of_digits = sum(int(digit) for digit in str(num))

        # Display the results
        print(f"Reversed number: {reversed_num}")
        print(f"Sum of digits: {sum_of_digits}")
    else:
        print("Invalid input. Please enter a valid number.")
# Call the function
reverse_and_sum_digits()
```

OUTPUT:

Enter a number: 376

Reversed number: 673

Sum of digits: 16

RESULT: The output has been obtained successfully.

2.Write a Python program to find the number of uppercase and lowercase letters in a given text.

ALGORITHM:

Step 1: Prompt the user for text input and store it in the variable text.

Step 2: Initialize counters upper and lower to zero.

Step 3: Iterate through each character char in the input text.

Step 4: If char is an uppercase letter, increment the upper counter.

Step 5: If char is a lowercase letter, increment the lower counter.

Step 6: Print the count of uppercase letters using upper.

Step 7: Print the count of lowercase letters using lower.

PROGRAM CODE:

```
text=input("Enter a text:")
upper=0
lower=0
for char in text:
  if char.isupper():
    upper+=1
  elif char.islower():
    lower+=1
print(f"Number of uppercases :{upper}")
print(f"Number of lowercases :{lower}")
```

OUTPUT:

```
Enter a text:Data Science
Number of uppercases :2
Number of lowercases :9
```

RESULT: The output has been obtained successfully.

PROGRAM NO:2

AIM: Programs using matplotlib/plotly/bokeh/seaborn for data visualization

1.Write a program that uses Matplotlib to create a simple line plot of a set of datapoints.

ALGORITHM:

Step 1: Import the Matplotlib library's pyplot module as plt.

Step 2: Define two lists x and y containing data points for the x and y axes, respectively.

Step 3: Use plt.plot(x, y) to create a simple line plot.

Step 4: Add labels to the x-axis and y-axis using plt.xlabel('X-axis') and plt.ylabel('Y-axis'), respectively.

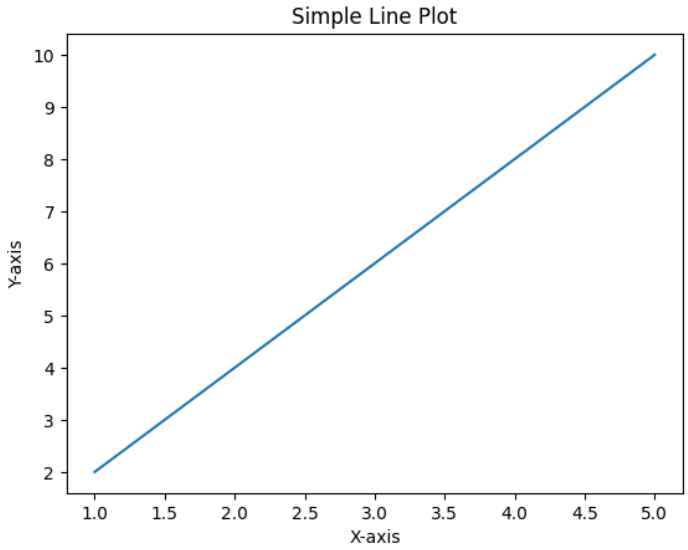
Step 5: Add a title to the plot using plt.title('Simple Line Plot').

Step 6: Display the plot using plt.show().

PROGRAM CODE:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.plot(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Plot')
plt.show()
```

OUTPUT:



RESULT: The output has been obtained successfully.

2.Create a Scatterplot with labeled datapoints, adjusting the colors and markers.

ALGORITHM:

Step 1: Import the Matplotlib library's pyplot module as plt.

Step 2: Define lists x and y containing data points for the x and y axes, respectively.

Step 3: Create a list labels containing labels for each data point.

Step 4: Create lists colors and markers specifying colors and markers for each data point.

Step 5: Use a loop to iterate over the data points, using plt.scatter() to create a scatter plot with labeled points, adjusting colors and markers.

Step 6: Add labels to the x-axis and y-axis using plt.xlabel('X-axis') and plt.ylabel('Y-axis'), respectively.

Step 7: Add a title to the plot using plt.title('Scatter Plot with Labeled Data Points').

Step 8: Display the legend using plt.legend().

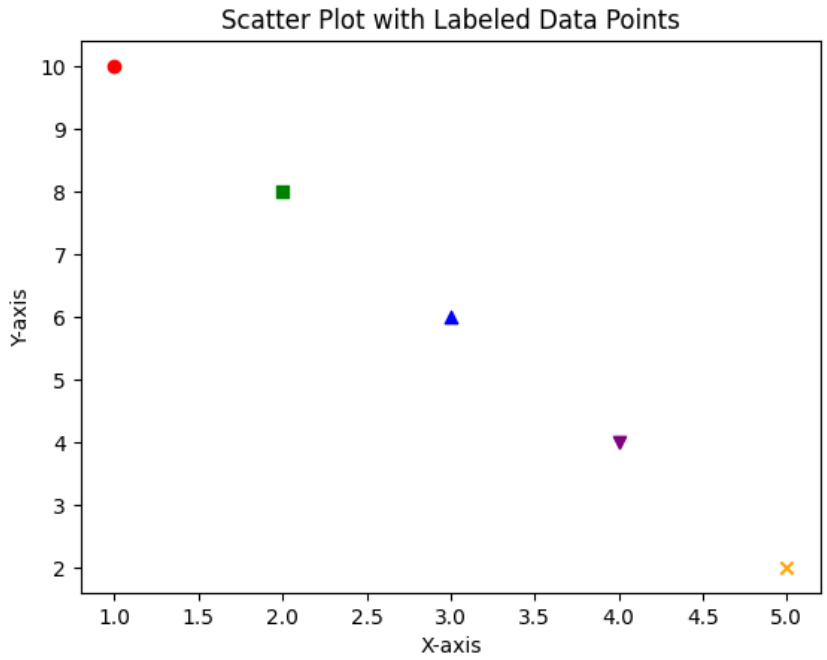
Step 9: Display the plot using plt.show().

PROGRAM CODE:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [10, 8, 6, 4, 2]
labels = ['A', 'B', 'C', 'D', 'E']
colors = ['red', 'green', 'blue', 'purple', 'orange']
markers = ['o', 's', '^', 'v', 'x']
for i in range(len(x)):
    plt.scatter(x[i], y[i], c=colors[i], marker=markers[i],
label=labels[i])
```

```
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot with Labeled Data Points')
```

OUTPUT:



RESULT: The output has been obtained successfully.

PROGRAM NO:3

AIM: Programs to handle data using pandas.

1.Remove Outliers from any standard dataset available in the public domain using standard deviation.

ALGORITHM:

Step 1: Import the Pandas library as pd.

Step 2: Define the file path of the CSV file containing the data (file_path).

Step 3: Read the CSV file into a Pandas DataFrame (df).

Step 4: Define a function fnctn that takes a DataFrame and a column name as input, calculates the mean, standard deviation, lower limit, and upper limit, and filters the data within the 3-sigma range.

Step 5: Specify the column(s) for which to perform the operation (columns).

Step 6: Use a loop to iterate over the specified columns, applying the function to filter the data within the 3-sigma range.

Step 7: Define the output file path for the filtered data (output).

Step 8: Write the filtered DataFrame to a new CSV file.

PROGRAM CODE:

```
import pandas as pd
from google.colab import drive
# Mount Google Drive to access files
drive.mount('/content/drive')
# Update the file path to point to the dataset in Google Drive
```

```
file_path = '/content/drive/My Drive/python_ds/weightdata.csv'
# Read the dataset into a pandas DataFrame
df = pd.read_csv(file_path)
def filter_outliers(data, col):
    mean = data[col].quantile(0.50)
    sd = data[col].std()
    lower_limit = mean - 3 * sd
    upper_limit = mean + 3 * sd
    return data[(data[col] >= lower_limit) & (data[col] <= upper_limit)]
columns = ['Weight(Pounds)']
for column in columns:
    df = filter_outliers(df, column)
# Update the output file path in Google Drive
output = '/content/drive/My Drive/python_ds/Output_sd-Sheet1.csv'
df.to_csv(output, index=False)
# Print the outliers
outliers = pd.concat([df, pd.concat([df,
df]).drop_duplicates(keep=False)]).drop_duplicates(keep=False)
print("Outliers:")
print(outliers)
```

OUTPUT:

```
Outliers:
       Index  Weight(Pounds)
0          1        112.9925
1          2        136.4873
2          3        153.0269
3          4        142.3354
4          5        144.2971
...      ...              ...
24995  24996        118.0312
24996  24997        120.1932
24997  24998        118.2655
24998  24999        132.2682
24999  25000        124.8742
[24933 rows x 2 columns]
```

RESULT: The output has been obtained successfully.

2. Remove Outliers from any standard dataset available in the public domain using quartile.

ALGORITHM:

Step 1: Import the Pandas library as pd and the NumPy library as np.

Step 2: Define the file path of the CSV file containing the data (file_path).

Step 3: Read the CSV file into a Pandas DataFrame (df).

Step 4: Define a function fnctn that takes a DataFrame, a column name, and an optional parameter a (default value is 1.5), calculates the first quartile (Q1), third quartile (Q3), interquartile range (IQR), lower limit, and upper limit, and filters the data within the specified range.

Step 5: Specify the column(s) for which to perform the operation (columns).

Step 6: Use a loop to iterate over the specified columns, applying the function to filter the data within the specified range.

Step 7: Define the output file path for the filtered data (output).

Step 8: Write the filtered DataFrame to a new CSV file.

PROGRAM CODE:

```
import pandas as pd
import numpy as np
from google.colab import drive
# Mount Google Drive to access files
drive.mount('/content/drive')
# Update the file path to point to the dataset in Google Drive
file_path = '/content/drive/My Drive/python_ds/weightdata.csv'
df = pd.read_csv(file_path)

def fnctn(data, col, a=1.5):
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
```

```
    IQR = Q3 - Q1
    lower_limit = Q1 - a * IQR
    upper_limit = Q3 + a * IQR
    return data[(data[col] >= lower_limit) & (data[col] <= upper_limit)]
columns = ['Index', 'Weight(Pounds)']
for column in columns:
    df = fnctn(df, column)
output = '/content/drive/My Drive/python_ds/Output_sd-Sheet2.csv'
df.to_csv(output, index=False)
# Print the outliers
outliers = pd.concat([df, pd.concat([df,
df]).drop_duplicates(keep=False)]).drop_duplicates(keep=False)
print("Outliers:")
print(outliers)
```

OUTPUT:

Outliers:

	Index	Weight(Pounds)
0	1	112.9925
1	2	136.4873
2	3	153.0269
3	4	142.3354
4	5	144.2971
...	...	...
24995	24996	118.0312
24996	24997	120.1932
24997	24998	118.2655
24998	24999	132.2682
24999	25000	124.8742

[24809 rows x 2 columns]

RESULT: The output has been obtained successfully.

COURSE OUTCOME 2

PROGRAM NO:4

AIM: Program to implement k-NN classification using any standard dataset available in the public domain and find the accuracy of the algorithm

ALGORITHM:

Step 1: Initialize the necessary libraries and load the dataset using pandas

Step 2: Prepare the data by separating features (X) and target variable (y) and split the dataset into training and testing sets.

Step 3: Train the k-NN classifier and train the classifier using the training data.

Step 4: Evaluate the model by calculating the accuracy of the model on the testing set and display the confusion matrix and visualize the results.

Step 5: User input the age, smoking status, area quality, and alcohol consumption.

Step 6: Make predictions for user input by using the trained k-NN model to predict the likelihood of lung cancer.

Step 7: Plot the accuracy on the training and testing sets for different k values.

PROGRAM CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from google.colab import drive

drive.mount('/content/drive')

file_path = '/content/drive/My Drive/Colab Notebooks/lungcancer.csv'
lung_cancer_data = pd.read_csv(file_path)

x = lung_cancer_data.drop(['Result', 'Name', 'Surname'], axis=1)
```

```
y = lung_cancer_data['Result']
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,
random_state=42)

k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(x_train, y_train)

accuracy = knn.score(x_test, y_test)
print("Accuracy of k-NN classifier with k={}: ".format(k), accuracy)

y_predict = knn.predict(x_test)
print(f"The predicted values are: {y_predict}")

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predict)

import seaborn as sns
f, ax = plt.subplots(figsize=(5, 5))

sns.heatmap(cm, annot=True, linewidths=0.5, linecolor="red", fmt="d",
ax=ax)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()

user_age = float(input("Enter Age: "))
user_smokes = int(input("Enter Smokes: "))
user_areaq = float(input("Enter AreaQ: "))
user_alkhol = float(input("Enter Alkhol: "))

user_input_data = pd.DataFrame({'Age': [user_age], 'Smokes':
[user_smokes], 'AreaQ': [user_areaq], 'Alkhol': [user_alkhol]})

user_predictions = knn.predict(user_input_data)

if user_predictions[0] == 1:
    print("The model predicts that the user may have lung cancer.")
else:
    print("The model predicts that the user may not have lung cancer.")
```

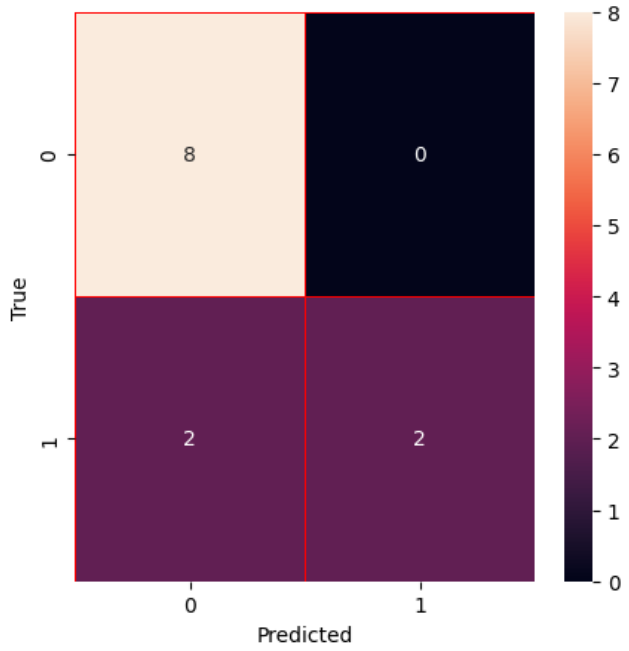
```
neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k )
    knn.fit(x_train, y_train)

    train_accuracy[i] = knn.score(x_train, y_train)
    test_accuracy[i] = knn.score(x_test, y_test)

plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')

plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```

OUTPUT:

Mounted at /content/drive

Accuracy of k-NN classifier with k=5: 0.8333333333333334

The predicted values are: [0 0 0 0 0 0 1 0 0 0 1 0]



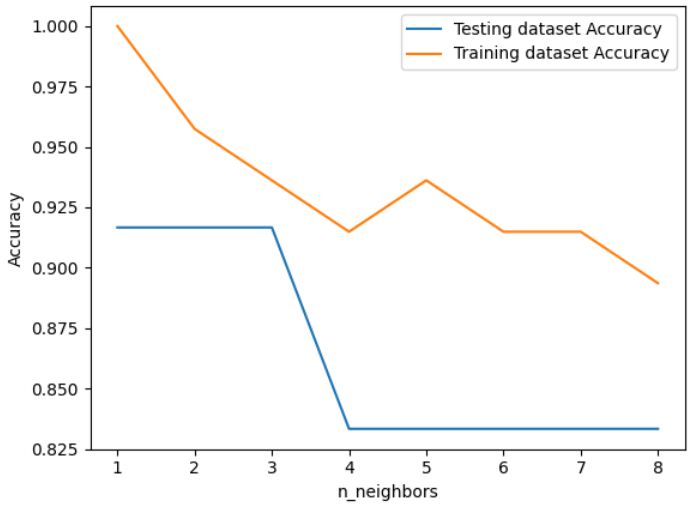
Enter Age: 50

Enter Smokes: 3

Enter AreaQ: 4

Enter Alkhol: 2

The model predicts that the user may not have lung cancer.



RESULT: The output has been obtained successfully.

PROGRAM NO:5

AIM: Program to implement Naive Bayes Algorithm using any standard dataset available in the public domain and find the accuracy of the algorithm.

ALGORITHM:

Step1: Import necessary libraries like pandas as pd ,GaussianNB from sklearn.naive_bayes,accuracy_score, confusion_matrix from sklearn.metrics etc

Step 2: Mount Google Drive

Step 3: Load breastcancer dataset from a CSV file using pd.read_csv

Step 4: Prepare the dataset

Step 5: Normalize the features using Min-Max scaling

Step 6: Split the data into training and testing sets using train_test_split

Step 7: Create a Gaussian Naive Bayes classifier

Step 8: Train the classifier on the training data using model name.fit()

Step 9: Make predictions on the test set using model name.predict()

Step 10: Calculate and print the accuracy of the model

Step 11: Generate and print the confusion matrix

Step 12: Visualize the confusion matrix using a heatmap

PROGRAM CODE:

```
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
from google.colab import drive
# Mount Google Drive
drive.mount('/content/drive')
```

```
# Load data from CSV file
data = pd.read_csv("/content/drive/My Drive/python_ds/breast_cancer.csv")
#Prepare Dataset
X=data.drop('diagnosis',axis=1)
y=data.diagnosis
from sklearn.preprocessing import MinMaxScaler

# Assuming X is your feature matrix
# Create a MinMaxScaler object
scaler = MinMaxScaler()

# Fit the scaler on the training data and transform both the training and
test data
X_normalized = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y,
test_size=0.3, random_state=200)


clf=GaussianNB()
clf.fit(X_train,y_train)
y_pred=clf.predict(X_test)
# Calculate and print the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)


# confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
sns.heatmap(conf_matrix, annot=True, fmt='d',
            xticklabels=['Predicted False', 'Predicted True'],
            yticklabels=['Actual False', 'Actual True'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

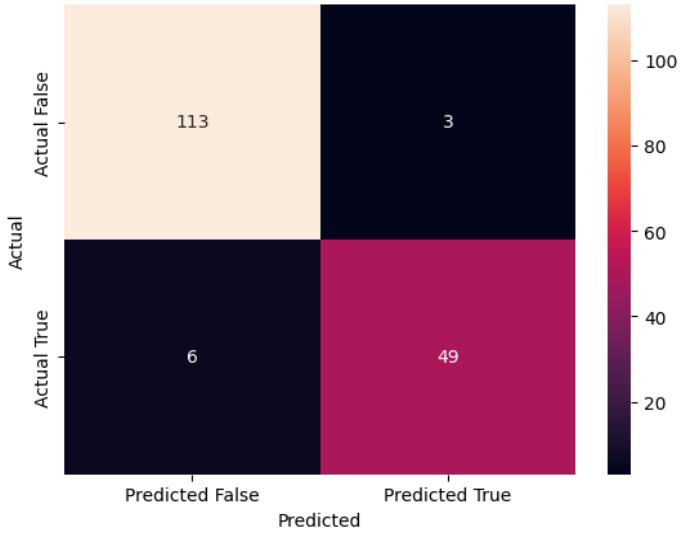
OUTPUT:

Accuracy: 0.9473684210526315

Confusion Matrix:

[[113 3]

[6 49]]



RESULT: The output has been obtained successfully.

PROGRAM NO:6

AIM: Program to implement linear and multiple regression using any standard dataset available in the public domain and find the accuracy of the algorithm

ALGORITHM:

Step 1: Import necessary libraries like pandas ,numpy,sklearn from linear_model etc
Step 2: Mount Google Drive
Step 3: Load house price dataset from a CSV file using pd.read_csv
Step 4: Handle missing values in the 'bedrooms' column by filling with the median using fillna
Step 5: Perform one-hot encoding on categorical variables
Step 6: Split the dataset into features (X) and target (y)
Step 7: Split the data into training and testing sets using train_test_split
Step 8: Create a Linear Regression model
Step 9: Train the model on the training data using model.fit()
Step 10: Make predictions on the test set using model.predict()
Step 11: Evaluate the model's performance using mean_squared_error, r2_score
Step 12: Display the coefficients of the linear regression model using model.coef_

PROGRAM CODE:

```
import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from google.colab import drive
import statsmodels.api as sm
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/MyDrive/python_ds/houseprice.csv')
df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
```

```
ds_encoded=pd.get_dummies(df,columns=['mainroad', 'guestroom', 'basement',
'hotwaterheating', 'airconditioning', 'prefarea', 'furnishingstatus'],
drop_first=True)
#split into features and targets
X=ds_encoded.drop('price',axis=1)
y=ds_encoded['price']
#Split the dataset into training and testing dataset
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,
random_state=42)
#fit the model
model=LinearRegression()
model.fit(X_train,y_train)
#Make predictions
y_pred=model.predict(X_test)
from sklearn.metrics import mean_squared_error, r2_score
print(f'Mean Squared Error: {mean_squared_error(y_test, y_pred)}')
print(f'R-squared: {r2_score(y_test, y_pred)}')
model.coef_
```

OUTPUT:

```
Mean Squared Error: 1754318687330.6638
R-squared: 0.6529242642153184
array([ 2.35968805e+02,  7.67787016e+04,  1.09444479e+06,  4.07476595e+05,
        2.24841913e+05,  3.67919948e+05,  2.31610037e+05,  3.90251176e+05,
        6.84649885e+05,  7.91426736e+05,  6.29890565e+05, -1.26881818e+05,
       -4.13645062e+05])
```

RESULT: The output has been obtained successfully.

COURSE OUTCOME 3

PROGRAM NO:7

AIM: Program to implement text classification using support vector machine algorithm:

ALGORITHM:

Step 1: Import necessary libraries like pandas,sklearn

Step 2: Load the dataset and read using pandas

Step 3: Create a binary spam column

Step 4: Split the dataset into training and testing set

Step 5: Create a pipeline with Countervectorizer and SVM

Step 6: Train the classifier using training data

Step 7: Predict whether the emails are spam or not

Step 8: Evaluate the model on test set

PROGRAM CODE:

```
import pandas as pd
from google.colab import drive
df = pd.read_csv('/content/drive/MyDrive/python_ds/spam.csv')
df['spam']=df['Category'].apply(lambda x: 1 if x=='spam' else 0)
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(df.Message,df.spam,test_siz
e=.20,random_state=43)
from sklearn.feature_extraction.text import CountVectorizer
v= CountVectorizer()
X_train_count = v.fit_transform(X_train.values)
X_train_count.toarray()[2]
from sklearn import svm
model=svm.SVC()
model.fit(X_train_count,y_train)

emails = [
    'Hey mohan, can we get together to watch footbal game tomorrow?',
```

```
    'Upto 90% discount on parking, exclusive offer just for you. Dont miss
this reward!',
    'Hello, Good Morning,how are you',
    'Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005'
]
emails_count = v.transform(emails)
model.predict(emails_count)

X_test_count=v.transform(X_test)
model.score(X_test_count,y_test)
from sklearn.pipeline import Pipeline
clf= Pipeline([ ('vectorizer', CountVectorizer()),
    ('rb',svm.SVC())
])
clf.fit(X_train, y_train)
clf.predict(emails)
```

OUTPUT:

0.97847533632287

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

RESULT: The output has been obtained successfully.

PROGRAM NO:8

AIM: Program to implement decision tree using any standard dataset available in the public domain and find the accuracy of the algorithm

ALGORITHM:

Step 1: Import Libraries like pandas, from sklearn.preprocessing import LabelEncoder, from sklearn.tree import DecisionTreeClassifier, plot_tree.
Step 2: Read Dataset using pd.read_csv
Step 3: Set Feature Column by dropping the target column
Step 4: Set Target Column
Step 5: Encode String Values by using LabelEncoder to encode string values in the feature columns:
Step 6: Drop Original String Columns
Step 7: Create a Decision Tree Model
Step 8: Split Data for Testing and Training using train_test_split to split the data into training and testing sets:
Step 9: Train the Decision Tree model using the training data: model.fit()
Step 10: Evaluate the accuracy of the model on the testing data: model.score()
Step 11: Make a prediction for a specific scenario using model.predict()
Step 12: Plot the Decision Tree

PROGRAM CODE:

```
import pandas as pd
df = pd.read_csv("salaries.csv")
df.head()
#setting feature columns
inputs = df.drop('salary_more_then_100k',axis='columns')
inputs
#setting target columns
```

```
target = df['salary_more_then_100k']
target
#Encoding string values
from sklearn.preprocessing import LabelEncoder
le_company = LabelEncoder()
le_job = LabelEncoder()
le_degree = LabelEncoder()
inputs['company_n'] = le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_job.fit_transform(inputs['job'])
inputs['degree_n'] = le_degree.fit_transform(inputs['degree'])
inputs
inputs_n = inputs.drop(['company','job','degree'],axis='columns')
inputs_n
from sklearn import tree
model = DecisionTreeClassifier()
#splitting data for testing and training
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
            inputs_n, target, test_size = 0.2, random_state=50)
model.fit(X_train,y_train)
#accuracy
model.score(X_test, y_test)
#Is salary of Google, Computer Engineer, Bachelors degree > 100 k ?
model.predict([[2,1,1]])
# Plot the decision tree
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
plt.figure(figsize=(12, 8))
plot_tree(model, feature_names=inputs_n.columns, class_names=['<=100k',
'>100k'], filled=True, rounded=True)
plt.show()
```

OUTPUT:

	company	job	degree	salary_more_then_100k
0	google	sales executive	bachelors	0
1	google	sales executive	masters	0
2	google	business manager	bachelors	1
3	google	business manager	masters	1
4	google	computer programmer	bachelors	0

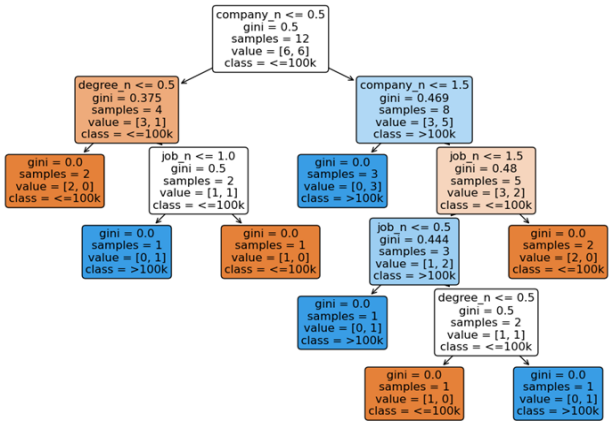
	company	job	degree
0	google	sales executive	bachelors
1	google	sales executive	masters
2	google	business manager	bachelors
3	google	business manager	masters
4	google	computer programmer	bachelors
5	google	computer programmer	masters
6	abc pharma	sales executive	masters
7	abc pharma	computer programmer	bachelors
8	abc pharma	business manager	bachelors
9	abc pharma	business manager	masters
10	facebook	sales executive	bachelors
11	facebook	sales executive	masters
12	facebook	business manager	bachelors
13	facebook	business manager	masters
14	facebook	computer programmer	bachelors
15	facebook	computer programmer	masters

```
0     0
1     0
2     1
3     1
4     0
5     1
6     0
7     0
8     0
9     1
10    1
11    1
12    1
13    1
14    1
15    1
Name: salary_more_then_100k, dtype: int64
```

	company	job	degree	company_n	job_n	degree_n
0	google	sales executive	bachelors	2	2	0
1	google	sales executive	masters	2	2	1
2	google	business manager	bachelors	2	0	0
3	google	business manager	masters	2	0	1
4	google	computer programmer	bachelors	2	1	0
5	google	computer programmer	masters	2	1	1
6	abc pharma	sales executive	masters	0	2	1
7	abc pharma	computer programmer	bachelors	0	1	0
8	abc pharma	business manager	bachelors	0	0	0
9	abc pharma	business manager	masters	0	0	1
10	facebook	sales executive	bachelors	1	2	0
11	facebook	sales executive	masters	1	2	1
12	facebook	business manager	bachelors	1	0	0
13	facebook	business manager	masters	1	0	1
14	facebook	computer programmer	bachelors	1	1	0
15	facebook	computer programmer	masters	1	1	1

	company_n	job_n	degree_n
0	2	2	0
1	2	2	1
2	2	0	0
3	2	0	1
4	2	1	0
5	2	1	1
6	0	2	1
7	0	1	0
8	0	0	0
9	0	0	1
10	1	2	0
11	1	2	1
12	1	0	0
13	1	0	1
14	1	1	0
15	1	1	1

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```



RESULT: The output has been obtained successfully.

PROGRAM NO:9

AIM: Program to implement k-means clustering technique using any standard dataset available in the public domain

ALGORITHM:

Step 1: Import necessary libraries and Mount Google Drive

Step 2: Read Data using pandas

Step 3: Visualize Data using plt.scatter

Step 4: Normalize Data using MinMaxScaler from sklearn.preprocessing

Step 5: Apply KMeans Clustering

Step 6: Assign Cluster Labels

Step 7: Visualize Clusters using DataFrames

Step 8: Determine Optimal K (Number of Clusters)

Step 9: Display Plot using plt.plot() to visualize the Elbow Method by plotting K against SSE.

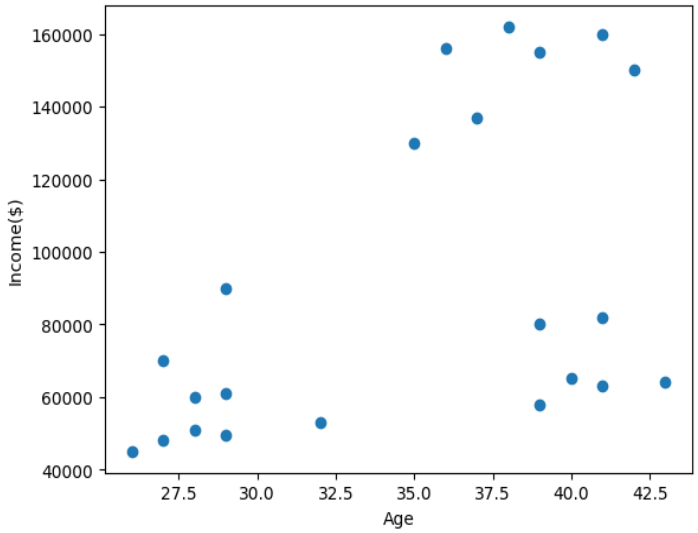
PROGRAM CODE:

```
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
%matplotlib inline
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

df = pd.read_csv("/content/drive/My Drive/python_ds/incomekm.csv")
plt.scatter(df.Age,df['Income($)'])
plt.xlabel('Age')
plt.ylabel('Income($)')
scaler = MinMaxScaler()
```

```
scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age','Income($)']])
y_predicted
df['cluster']=y_predicted
Km.cluster_centers_
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purpl
e',marker='*',label='centroid')
plt.legend()
sse = []
k_rng = range(1,10)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df[['Age','Income($)']])
    sse.append(km.inertia_)

plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
```

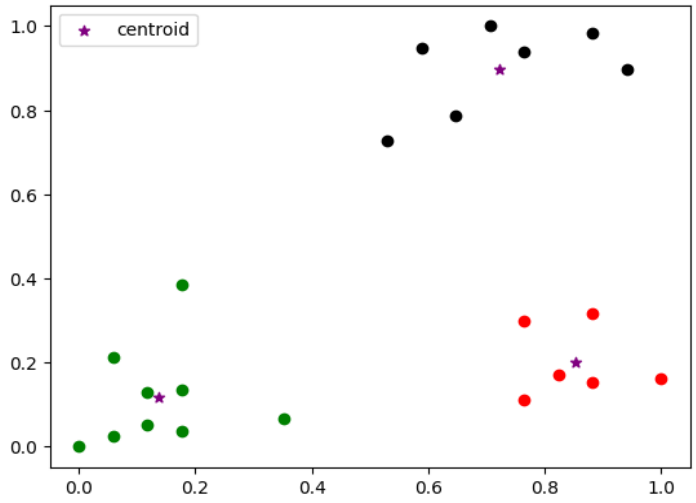
OUTPUT:

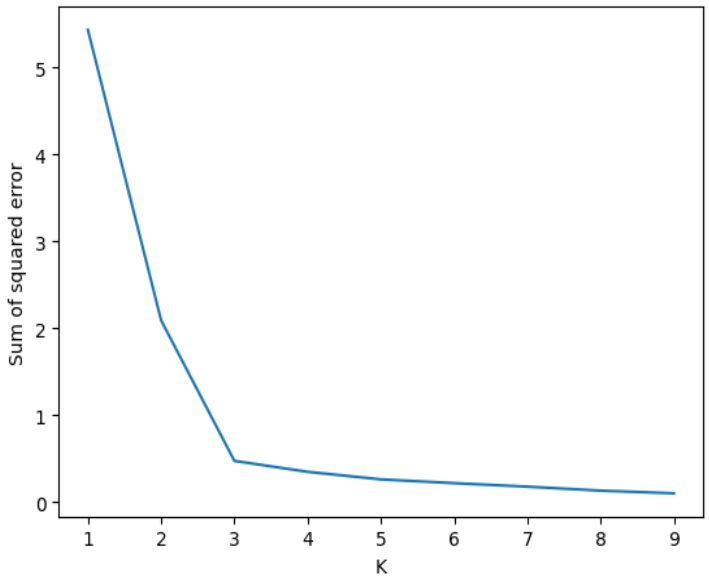
Text(0, 0.5, 'Income($)')




```
array([0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
      dtype=int32)
```

array([[0.1372549 , 0.11633428],
[0.85294118, 0.2022792],
[0.72268908, 0.8974359]])





RESULT: The output has been obtained successfully.

COURSE OUTCOME 4

PROGRAM NO:10

AIM: Programs on feedforward network to classify any standard dataset available in the public domain.

ALGORITHM:

Step 1: Import necessary libraries such as numpy, pandas, matplotlib.pyplot, and tensorflow and set random seeds for reproducibility using tf.random.set_seed(42) and np.random.seed(42).

Step 2: Read the dataset into a DataFrame using Pandas.

Step 3: Separate features (X) and the target variable (y) from the dataset.

Step 4: Check for missing values in the dataset using df.isnull().

Step 5: Split the dataset into training and testing sets using train_test_split.

Step 6: Apply feature scaling using StandardScaler to normalize the features.

Step 7: Build DNN Model and define the architecture of the neural network

Step 8: Compile and train the model.

Step 9: Plot the training and validation R2-scores across epochs using Matplotlib.

Step 10: Evaluate on Test Data and display the test loss and accuracy.

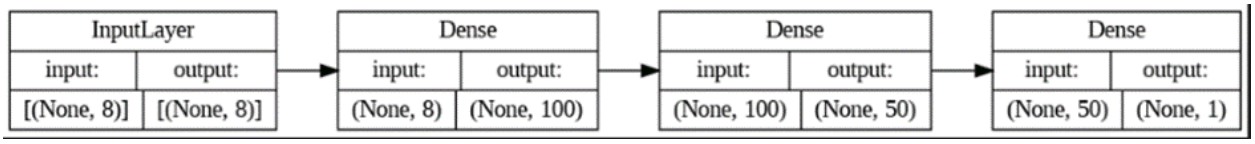
PROGRAM CODE:

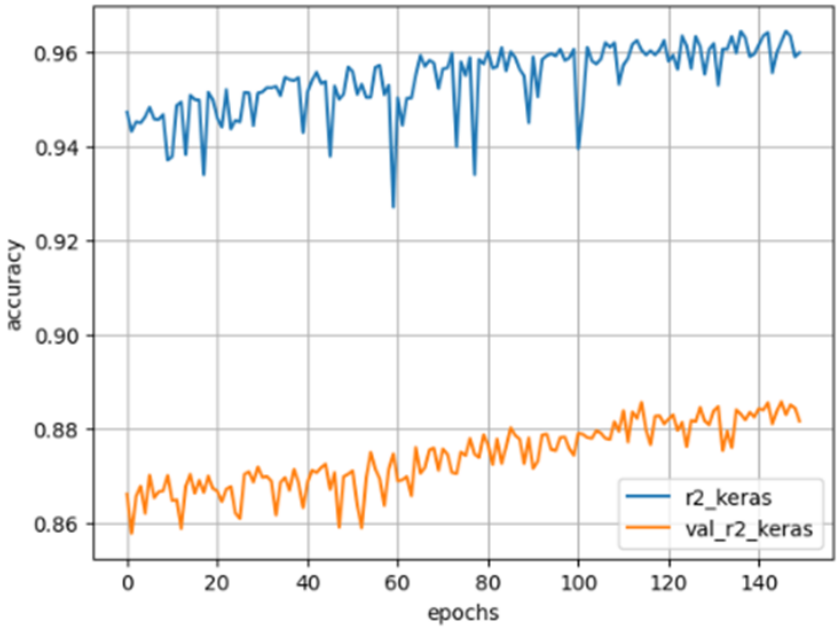
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import tensorflow as tf
from tensorflow import keras

tf.random.set_seed(42)
np.random.seed(42)

df = pd.read_csv('/content/Concrete_Data.csv')

# prints the first five entries by default.
df.head()
X=df.drop('strength',axis='columns')
y=df.strength
df.isnull().sum()
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,
y,test_size=0.2,random_state=42)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

sc.fit(X_train)

X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

y_train = y_train.values.reshape(-1,1)
y_test = y_test.values.reshape(-1,1)

input_ = keras.layers.Input(shape = X_train_std.shape[1:])

x = keras.layers.Dense(units=100, activation='relu')(input_)
x = keras.layers.Dense(units=50, activation='relu')(x)

output_ = keras.layers.Dense(units=1, activation='linear')(x)
model_DNN = keras.Model(inputs=[input_], outputs=[output_])

tf.keras.utils.plot_model(
    model_DNN,
    to_file = 'model_DNN.png',
    show_shapes = True,
```

```
    show_dtype = False,
    show_layer_names=False,
    rankdir='LR',
    expand_nested=False,
    dpi=96,
    layer_range=None,
    show_layer_activations=False,
    show_trainable=False
)

# custom R2-score metrics for keras backend
from keras import backend as K

def r2_keras(y_true, y_pred):
    SS_res =  K.sum(K.square(y_true - y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return ( 1 - SS_res/(SS_tot + K.epsilon()) )
#--> Compile with approriate settings for the model
model_DNN.compile(loss='mse',
                  optimizer='adam',
                  metrics=[r2_keras])
history_DNN = model_DNN.fit(x = X_train_std,
                            y = y_train,
                            epochs = 150,
                            validation_split = 0.1,
                            batch_size = 32)
keys = ['r2_keras', 'val_r2_keras']
progress = {k:v for k,v in history_DNN.history.items() if k in keys}

import pandas as pd
pd.DataFrame(progress).plot()

plt.xlabel("epochs")
plt.ylabel("accuracy")

plt.grid(True)
plt.show()
test_loss, test_accuracy = model_DNN.evaluate(X_test_std, y_test)

print("Test-loss: %f, Test-accuracy: %f" % (test_loss, test_accuracy))
```

OUTPUT:





Test-loss: 38.957378

Test-accuracy: 0.843751

RESULT: The output has been obtained successfully

PROGRAM NO:11

AIM: Program on convolutional neural networks to classify images from any standard dataset available in the public domain.

ALGORITHM:

Step 1: Initialize necessary libraries and load data

Step 2: Set random seed

Step 3: Read the dataset

Step 4: Preprocess the dataset and separate feature and target data

Step 5: Check for missing data.

Step 6: Split the dataset into training and testing data and apply feature Scaling

Step 7: Create compile and train the DNN model

Step 8: Plot training and validation scores

Step 9: Evaluate the model on test data

PROGRAM CODE:

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Load and preprocess the MNIST dataset
mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()

# Reshape and normalize the images
train_images = train_images.reshape(-1, 28, 28, 1).astype("float32") /
255.0
test_images = test_images.reshape(-1, 28, 28, 1).astype("float32") / 255.0

# Pad the images to size 32x32
train_images = tf.pad(train_images, [[0, 0], [2, 2], [2, 2], [0, 0]],
mode='constant')
```

```
test_images = tf.pad(test_images, [[0, 0], [2, 2], [2, 2], [0, 0]],
mode='constant')

# Define the LeNet architecture for 32x32 input size with average pooling
and Tanh activation
model = models.Sequential([
    layers.Conv2D(6, (5, 5), activation='tanh', input_shape=(32, 32, 1)),
    layers.AveragePooling2D((2, 2)),
    layers.Conv2D(16, (5, 5), activation='tanh'),
    layers.AveragePooling2D((2, 2)),
    layers.Conv2D(120, (5, 5), activation='tanh'),
    layers.Flatten(),
    layers.Dense(84, activation='tanh'),
    layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(train_images, train_labels, epochs=5, batch_size=64,
validation_data=(test_images, test_labels))

# Evaluate and print accuracy
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_accuracy}')
```

OUTPUT:

Test accuracy: 0.984000027179718

RESULT: The output has been obtained successfully.

COURSE OUTCOME 5

PROGRAM NO:12

AIM: Natural Language Processing

Problems may designed for the following topics so that students can get hands on experience in using python for natural language processing

1.Part of Speech tagging.

ALGORITHM:

Step 1: Import the NLTK library and download necessary resources like punkt, averaged_perceptron_tagger, tagsets.

Step 2: Tokenize the input text using the NLTK word_tokenize function.

Step 3: Perform part-of-speech tagging on the tokenized words using nltk.pos_tag.

Step 4: Define a simple grammar for noun phrase (NP) chunking.

Step 5: Create a RegexpParser using the defined grammar with nltk.RegexpParser.

Step 6: Apply the chunking to the tagged words using the created parser.

Step 7: Display the resulting chunked tree using NLTK's Tree module and visualize it.

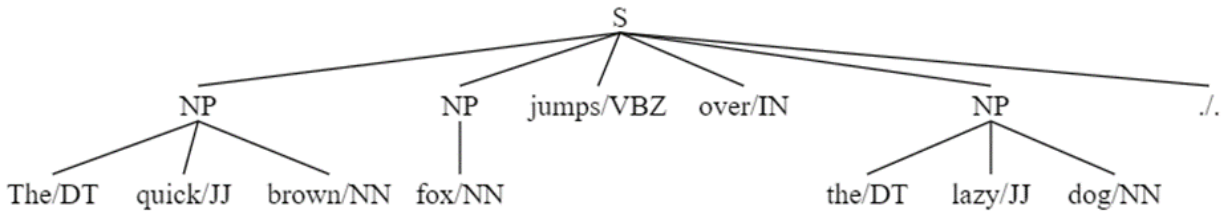
PROGRAM CODE:

```
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('tagsets')
from nltk.tokenize import word_tokenize
```

```
text="The quick brown fox jumps over the lazy dog."
word_tokens=word_tokenize(text)
print(word_tokens)
tagged_words=nltk.pos_tag(word_tokens)
print(tagged_words)
from nltk import RegexpParser
grammar="NP: {<DT>?<JJ>*<NN>}"
parser=RegexpParser(grammar)
chunk=parser.parse(tagged_words)
print(chunk)
from nltk.tree import Tree
tree=Tree.fromstring(str(chunk))
#Displaying Tree
from IPython.display import display
display(tree)
```

OUTPUT:

[('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN'), ('.', '.')]



RESULT: The output has been obtained successfully

2. N-gram

ALGORITHM:

Step 1: Import the necessary libraries like Counter, to count occurrences.

Step 2: Define a corpus containing a list of sentences.

Step 3: Implement a function calculate_ngram_prob to calculate the probability of an n-gram based on its counts and previous n-gram counts.

Step 4: Implement a function calculate_probabilities that takes a corpus and calculates unigram, bigram, and trigram counts as well as previous bigram counts.

Step 5: Implement a function print_probabilities to display the calculated probabilities for unigrams, bigrams, and trigrams.

Step 6: Call calculate_probabilities on the provided corpus to get counts.

Step 7: Call print_probabilities with the obtained counts to display the calculated probabilities.

PROGRAM CODE:

```
from collections import Counter

corpus = [
    "I like natural language processing",
    "Natural language processing is fun",
    "Processing language is interesting"
]

def calculate_ngram_prob(ngram, ngram_counts, prev_ngram_counts):
    if len(ngram) == 1:
        return ngram_counts[ngram] / sum(ngram_counts.values())
    elif len(ngram) == 2:
        prev_ngram = (ngram[0],)
```

```
        count = prev_ngram_counts[prev_ngram] if prev_ngram in
prev_ngram_counts else 0
    else:
        prev_ngram = (ngram[0], ngram[1])
        count = prev_ngram_counts[prev_ngram] if prev_ngram in
prev_ngram_counts else 0

    return ngram_counts[ngram] / count if count > 0 else 0.0

def calculate_probabilities(corpus):
    unigram_counts = Counter()
    bigram_counts = Counter()
    trigram_counts = Counter()
    prev_bigram_counts = Counter()

    for sentence in corpus:
        words = sentence.split()
        unigram_counts.update(words)
        bigrams = [(words[i], words[i+1]) for i in range(len(words)-1)]
        trigrams = [(words[i], words[i+1], words[i+2]) for i in
range(len(words)-2)]
        bigram_counts.update(bigrams)
        trigram_counts.update(trigrams)

        for i in range(len(words) - 1):
            prev_bigram_counts[(words[i],)] += 1

    return unigram_counts, bigram_counts, trigram_counts,
prev_bigram_counts

def print_probabilities(unigram_counts, bigram_counts, trigram_counts,
prev_bigram_counts):
    print("Unigram Probabilities:")
    for unigram in unigram_counts:
        prob = calculate_ngram_prob((unigram,), unigram_counts, None)
        print(f"{unigram}: {prob:.4f}")

    print("\nBigram Probabilities:")
    for bigram in bigram_counts:
```

```
        prob = calculate_ngram_prob(bigram, bigram_counts,
prev_bigram_counts)
        print(f"{bigram}: {prob:.4f}")

    print("\nTrigram Probabilities:")
    for trigram in trigram_counts:
        prob = calculate_ngram_prob(trigram, trigram_counts,
bigram_counts)
        print(f"{trigram}: {prob:.4f}")

unigram_counts, bigram_counts, trigram_counts, prev_bigram_counts =
calculate_probabilities(corpus)
print_probabilities(unigram_counts, bigram_counts, trigram_counts,
prev_bigram_counts)
```

OUTPUT:

Unigram Probabilities:

I: 0.0000

like: 0.0000

natural: 0.0000

language: 0.0000

processing: 0.0000

Natural: 0.0000

is: 0.0000

fun: 0.0000

Processing: 0.0000

interesting: 0.0000

Bigram Probabilities:

('I', 'like'): 1.0000

('like', 'natural'): 1.0000

('natural', 'language'): 1.0000

('language', 'processing'): 0.6667

('Natural', 'language'): 1.0000

('processing', 'is'): 1.0000

('is', 'fun'): 0.5000

('Processing', 'language'): 1.0000

('language', 'is'): 0.3333

('is', 'interesting'): 0.5000

Trigram Probabilities:

('I', 'like', 'natural'): 1.0000

('like', 'natural', 'language'): 1.0000

('natural', 'language', 'processing'): 1.0000

('Natural', 'language', 'processing'): 1.0000

('language', 'processing', 'is'): 0.5000

('processing', 'is', 'fun'): 1.0000

('Processing', 'language', 'is'): 1.0000

('language', 'is', 'interesting'): 1.0000

RESULT: The output has been obtained successfully.