R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, Puduvoyal – 601 206

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(CYBER SECURITY)

22CY904

AI IN CYBER SECURITY

(LAB INTEGRATED)

LAB MANUAL

R2022

ACADEMIC YEAR: 2024-25

ODD SEMESTER

B.E. COMPUTER SCIENCE AND ENGINEERING (CYBER SECURITY)

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, Puduvoyal – 601 206

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (CYBER SECUIRTY)

22CY904

AI IN CYBER SECURITY

(LAB INTEGRATED)

LAB MANUAL

R2022

2024-25 EVEN SEMESTER

B.E. COMPUTER SCIENCE AND ENGINEERING (CYBER SECURITY)

Prepared by	Checked by	Approved By
Dr.Dharini N	**Dr. S. M. Udhaya Sankar**	**Dr.N.Suresh Kumar,**
Associate Professor /CSE(CS)	Professor & Head / CSE (CS)	Principal

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, Puduvoyal – 601 206

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (CYBER SECUIRTY)

Vision

To excel and take the lead in Cyber Security education, profession and research globally with a commitment to effectively address societal needs

Mission

✤ To collaborate with innovators to provide real-world, standards - based cybersecurity capabilities that address business needs.

✤ To prepare the professionals in both academic and industrial settings capable of solving real world cybersecurity threats.

✤ To inculcate the students in designing and developing various projects in different areas of cybersecurity, by providing a distinguished and high-quality education.

Program Educational Objectives

Graduates of Computer Science and Engineering Program (Cyber Security) will be able to:

PEO 1 : Acquire the knowledge, skills and the attitude necessary for theanalysis of Cyber security.

PEO 2 : Apply the cutting-edge latest technology within a professional, legaland ethical frame work and will operate effectively in a multidisciplinary stream.

PEO 3 : Practice continued, self-learning to keep their knowledge and skillsup to date and remain abreast of the latest developments in cyber security.

Program Specific Outcomes

Graduates of Computer Science and Engineering (Cyber Security) Program will be able :

PSO 1 : To understand, analyze, design, and develop computing solutions by applying algorithms, web design, database management, networking & cyber security.

PSO 2 : To develop cyber security skills including network defense, ethical hacking, penetration testing, application security and cryptography to provide real time solutions.

PSO 3 : To apply standard tools, practices and strategies in cyber security for successful career and entrepreneurship

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, Puduvoyal – 601 206

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (CYBER SECURITY)

OBJECTIVES:

- Learn the Concepts of AI in Cybersecurity.
- Understand the methods of Cybersecurity Arsenal
- Learn the techniques for security threats with AI
- Protect the sensitive information and assets.
- Explore the Tools and Applications to learn about the AI in Cybersecurity.

LIST OF EXERCISES:

1. Explore Jupyter notebooks.
2. To study and explore Preprocessing of text (Tokenization, Filtration, Script, Validation, Stop word Removal, Stemming)
3. Explore Social media analysis
4. Implement a program for network access control to illustrate malware attacks.
5. Implement Naïve Bayes Scratch Implementation using Python.
6. Design a Spam detection algorithm.
7. Explore keystroke recognition using NLP.
8. Implement Real-time Facial Recognition Software powered by AI.
9. Explore Image Based Recognition using Software powered AI.
10. Design Fraud detection algorithm.
11. Explore Bagging and boosting Ensemble Classifiers for Classifications.
12. Create a fake news tracker program to collect, detect and help visualize fake news data.

OUTCOMES:

Upon completion of the course, the students will be able to:

CO1: Apply AI to enhance cybersecurity using Jupyter Notebooks for algorithm development and optimization.

CO2: Implement AI for cyber security arsenal.

CO3: Implement Spam Detection using ML and NLP Models

CO4: Protect and handle sensitive information and assets.

CO5: Implement AI-driven applications specifically tailored for fraud detection purposes.

CO6: Evaluate the effectiveness of AI-based solutions in enhancing cybersecurity measures.

Exp No: 1 **EXPLORE JUPYTER NOTEBOOKS** **Date:**

Aim: To explore Jupyter Notebooks

About Jupyter Notebooks

Jupyter notebooks basically provides an interactive computational environment for developing Python based Data Science applications. They are formerly known as ipython notebooks. The following are some of the features of Jupyter notebooks that makes it one of the best components of Python ML ecosystem –

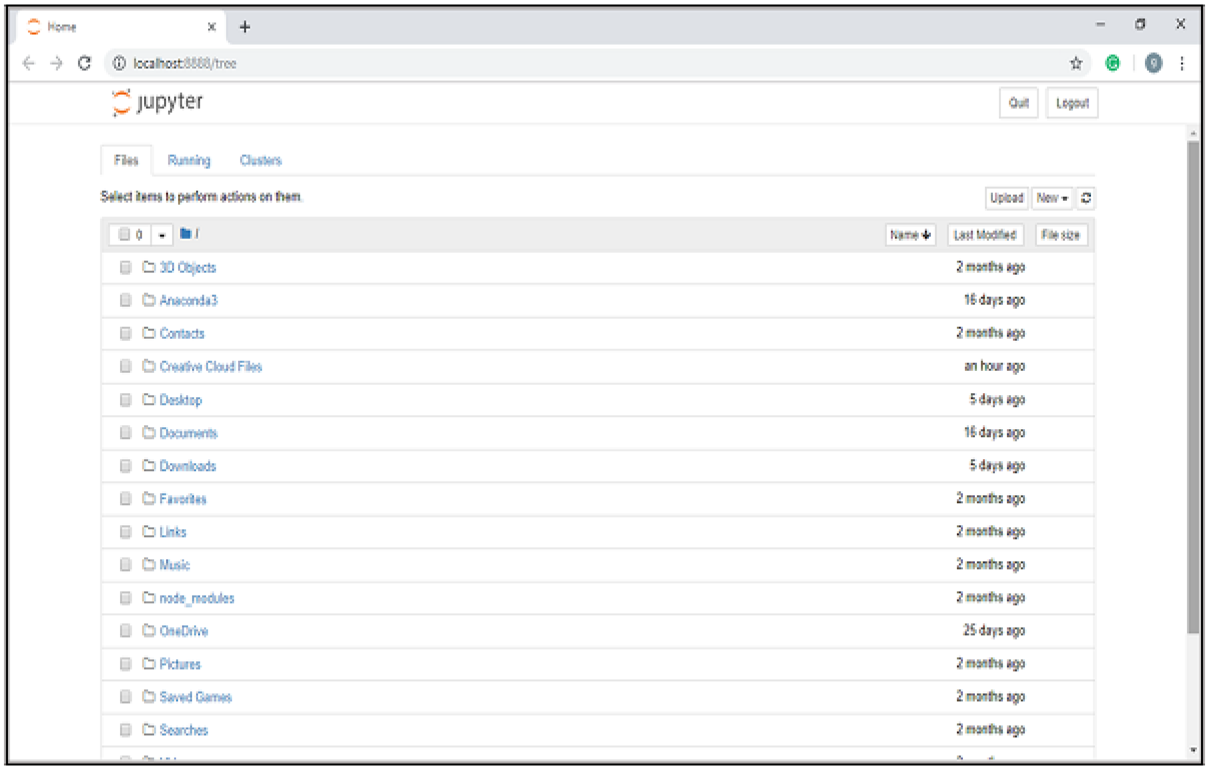
- Jupyter notebooks can illustrate the analysis process step by step by arranging the stuff like code, images, text, output etc. in a step by step manner.
- It helps a data scientist to document the thought process while developing the analysis process.
- One can also capture the result as the part of the notebook.
- With the help of jupyter notebooks, we can share our work with a peer also.

Installation and Execution

If you are using Anaconda distribution, then you need not install jupyter notebook separately as it is already installed with it. You just need to go to Anaconda Prompt and type the following command –

```
C:\>jupyter notebook
```

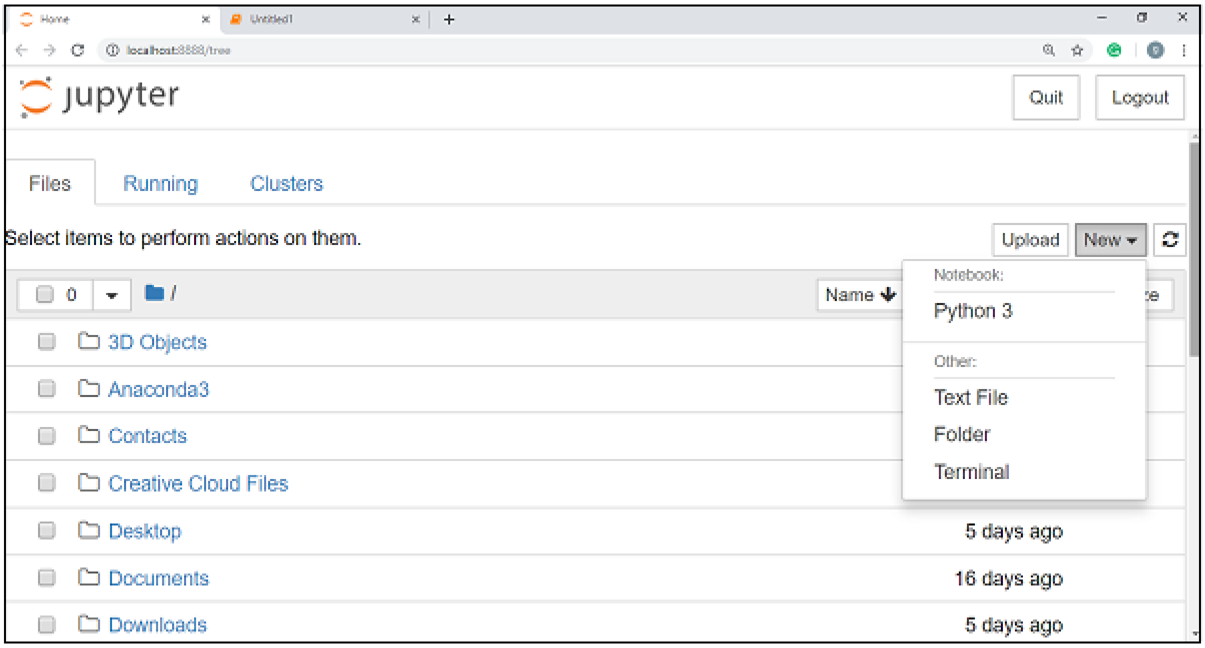
After pressing enter, it will start a notebook server at *localhost:8888* of your computer. It is shown in the following screen shot –



Select items to perform actions on them.

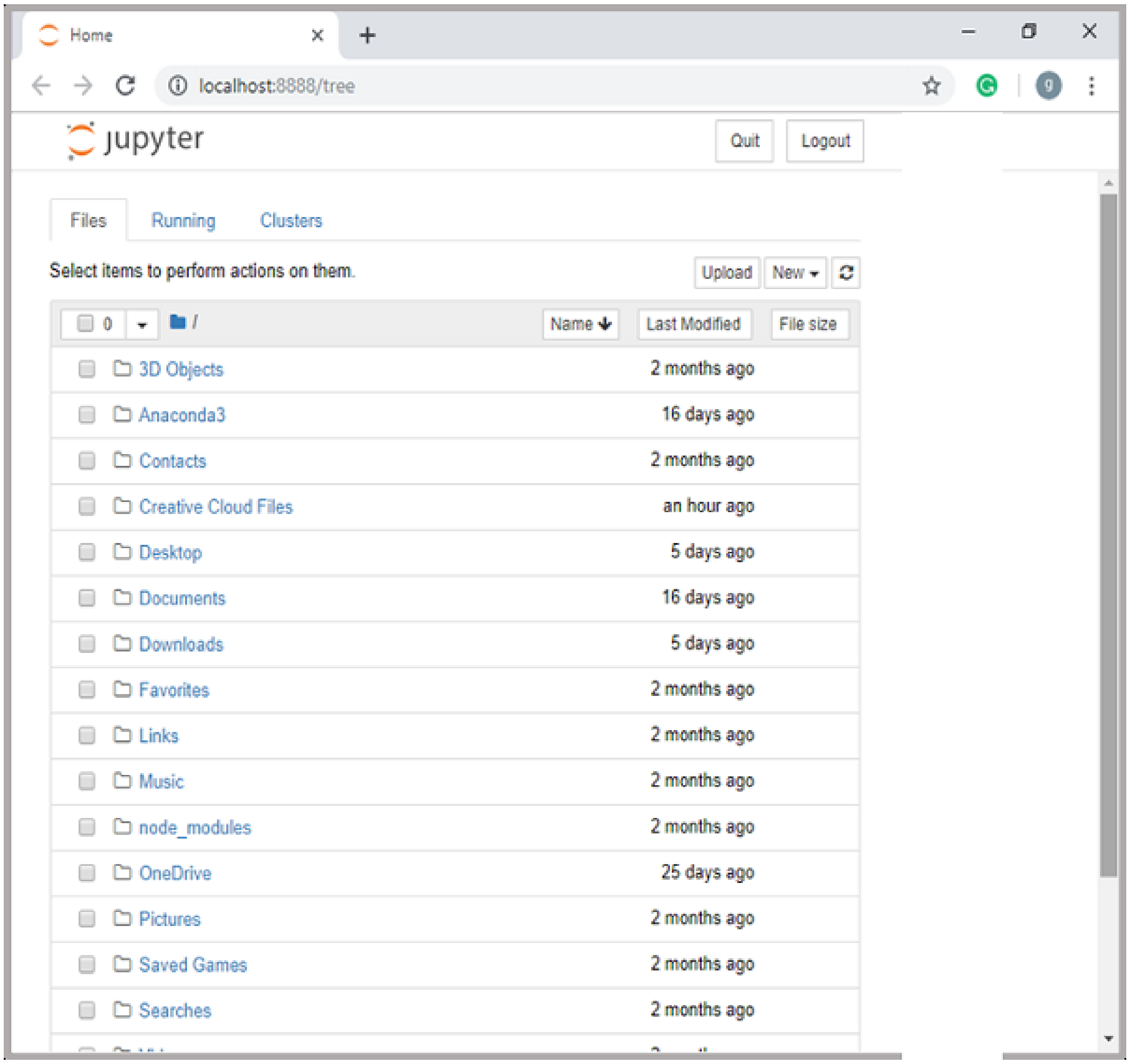
Name	Last Modified	File size
3D Objects	2 months ago	
Anaconda3	16 days ago	
Contacts	2 months ago	
Creative Cloud Files	an hour ago	
Desktop	5 days ago	
Documents	16 days ago	
Downloads	5 days ago	
Favorites	2 months ago	
Links	2 months ago	
Music	2 months ago	
node_modules	2 months ago	
OneDrive	25 days ago	
Pictures	2 months ago	
Saved Games	2 months ago	
Searches	2 months ago	

Now, after clicking the New tab, you will get a list of options. Select Python 3 and it will take you to the new notebook for start working in it. You will get a glimpse of it in the following screenshots –



Select items to perform actions on them.

3D Objects	
Anaconda3	
Contacts	
Creative Cloud Files	
Desktop	5 days ago
Documents	16 days ago
Downloads	5 days ago



Select items to perform actions on them.

Name	Last Modified	File size
3D Objects	2 months ago	
Anaconda3	16 days ago	
Contacts	2 months ago	
Creative Cloud Files	an hour ago	
Desktop	5 days ago	
Documents	16 days ago	
Downloads	5 days ago	
Favorites	2 months ago	
Links	2 months ago	
Music	2 months ago	
node_modules	2 months ago	
OneDrive	25 days ago	
Pictures	2 months ago	
Saved Games	2 months ago	
Searches	2 months ago	

On the other hand, if you are using standard Python distribution then jupyter notebook can be installed using popular python package installer, *pip*.

pip install jupyter

Types of Cells in Jupyter Notebook

The following are the three types of cells in a jupyter notebook −

- Code cells − As the name suggests, we can use these cells to write code. After writing the code/content, it will send it to the kernel that is associated with the notebook.
- Markdown cells − We can use these cells for notating the computation process. They can contain the stuff like text, images, Latex equations, HTML tags etc.
- Raw cells − The text written in them is displayed as it is. These cells are basically used to add the text that we do not wish to be converted by the automatic conversion mechanism of jupyter notebook.

For more detailed study of jupyter notebook, you can go to the link www.tutorialspoint.com/jupyter/index.htm.

NumPy

It is another useful component that makes Python as one of the favorite languages for Data Science. It basically stands for Numerical Python and consists of multidimensional array objects. By using NumPy, we can perform the following important operations −

- Mathematical and logical operations on arrays.
- Fourier transformation
- Operations associated with linear algebra.

We can also see NumPy as the replacement of MatLab because NumPy is mostly used along with Scipy (Scientific Python) and Mat-plotlib (plotting library).

Installation and Execution

If you are using Anaconda distribution, then no need to install NumPy separately as it is already installed with it. You just need to import the package into your Python script with the help of following −

import numpy as np

On the other hand, if you are using standard Python distribution then NumPy can be installed using popular python package installer, pip.

pip install NumPy

After installing NumPy, you can import it into your Python script as you did above.

For more detailed study of NumPy, you can go to the link www.tutorialspoint.com/numpy/index.htm.

Pandas

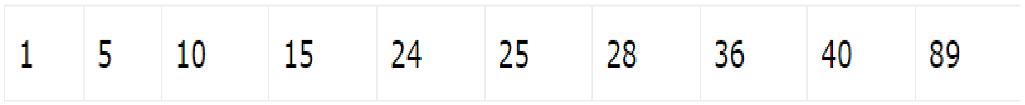
It is another useful Python library that makes Python one of the favorite languages for Data Science. Pandas is basically used for data manipulation, wrangling and analysis. It was developed by Wes McKinney in 2008. With the help of Pandas, in data processing we can accomplish the following five steps −

- Load
- Prepare
- Manipulate
- Model
- Analyze

Data representation in Pandas

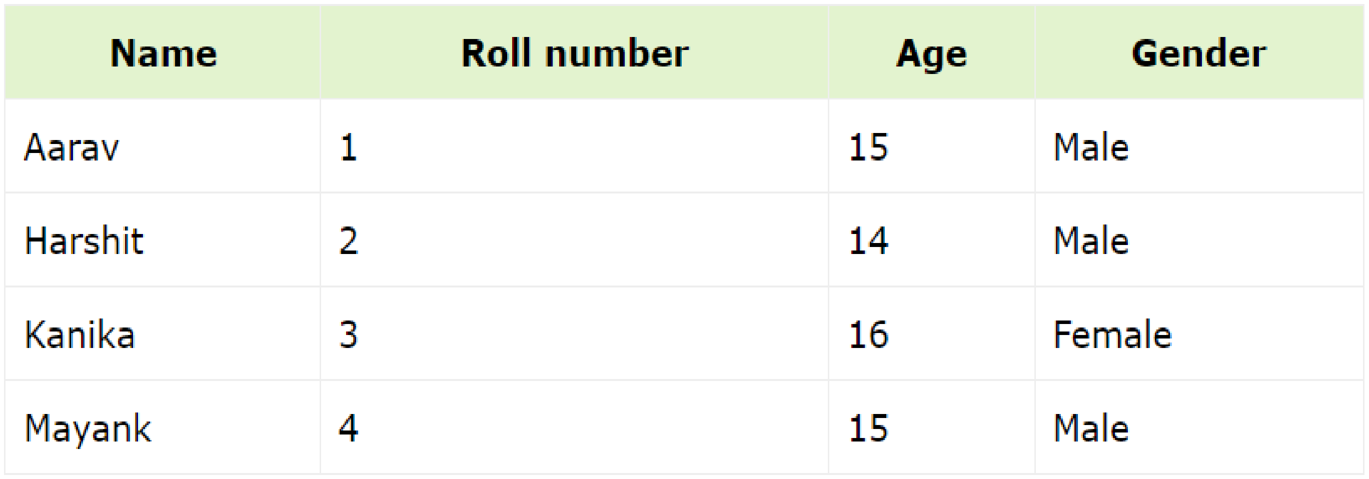
The entire representation of data in Pandas is done with the help of following three data structures −

Series − It is basically a one-dimensional ndarray with an axis label which means it is like a simple array with homogeneous data. For example, the following series is a collection of integers 1,5,10,15,24,25...



1	5	10	15	24	25	28	36	40	89

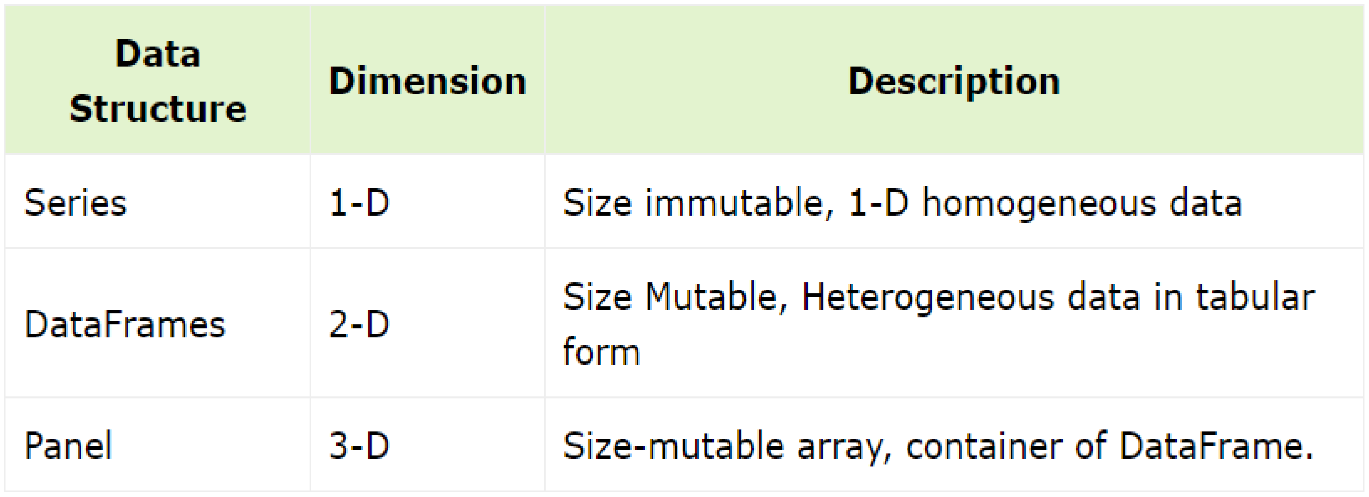
Data frame − It is the most useful data structure and used for almost all kind of data representation and manipulation in pandas. It is basically a two-dimensional data structure which can contain heterogeneous data. Generally, tabular data is represented by using data frames. For example, the following table shows the data of students having their names and roll numbers, age and gender.



Name	Roll number	Age	Gender
Aarav	1	15	Male
Harshit	2	14	Male
Kanika	3	16	Female
Mayank	4	15	Male

Panel − It is a 3-dimensional data structure containing heterogeneous data. It is very difficult to represent the panel in graphical representation, but it can be illustrated as a container of DataFrame.

The following table gives us the dimension and description about above mentioned data structures used in Pandas



Data Structure	Dimension	Description
Series	1-D	Size immutable, 1-D homogeneous data
DataFrames	2-D	Size Mutable, Heterogeneous data in tabular form
Panel	3-D	Size-mutable array, container of DataFrame.

Installation and Execution

If you are using Anaconda distribution, then no need to install *Pandas* separately as it is already installed with it. You just need to import the package into your Python script with the help of following −

```
import pandas as pd
```

On the other hand, if you are using standard Python distribution then Pandas can be installed using popular python package installer, *pip*.

pip install Pandas

After installing *Pandas*, you can import it into your Python script as did above.

Scikit-learn

Another useful and most important python library for Data Science and machine learning in Python is *Scikit-learn*. The following are some features of *Scikit-learn* that makes it so useful −

- It is built on NumPy, SciPy, and Matplotlib.
- It is an open source and can be reused under BSD license.
- It is accessible to everybody and can be reused in various contexts.
- Wide range of machine learning algorithms covering major areas of ML like classification, clustering, regression, dimensionality reduction, model selection etc. can be implemented with the help of it.

Installation and Execution

If you are using Anaconda distribution, then no need to install Scikit-learn separately as it is already installed with it. You just need to use the package into your Python script. For example, with following line of script we are importing dataset of breast cancer patients from **Scikit-learn** −

from sklearn.datasets import load_breast_cancer

On the other hand, if you are using standard Python distribution and having NumPy and SciPy then Scikit-learn can be installed using popular python package installer, pip.

pip install -U scikit-learn

After installing Scikit-learn, you can use it into your Python script as you have done above.

TensorFlow

TensorFlow is an open-source library for machine learning developed by Google. It provides support for building and training deep learning models, along with tools for distributed computing and deployment. TensorFlow is a powerful tool for building complex machine learning models, particularly in the areas of computer vision and natural language processing. Below is the command to install TensorFlow −

pip install tensorflow

PyTorch

PyTorch is another popular deep learning library in Python. Developed by Facebook, it provides a range of tools for building and training neural networks, along with support for dynamic computation graphs and GPU acceleration.

PyTorch is particularly useful for researchers and developers who need a flexible and powerful deep learning framework. Below is the command to install PyTorch −

pip install torch

Keras

Keras is a high-level neural network library that runs on top of TensorFlow and other lower-level frameworks. It provides a simple and intuitive API for building and training deep learning models, making it an excellent choice for beginners and researchers who need to quickly prototype and experiment with different models. Below is the command to install Keras −

pip install keras

OpenCV

OpenCV is a computer vision library that provides tools for image and video processing, along with support for machine learning algorithms. It is widely used in the computer vision community for tasks such as object detection, image segmentation, and facial recognition. Below is the command to install OpenCV −

```
pip install opencv-python
```

In addition to these libraries, there are many other tools and frameworks in the Python ecosystem for machine learning, including **XGBoost, LightGBM, spaCy,** and **NLTK**.

Matplotlib

Matplotlib is basically a data plotting tool inspired by MATLAB, and is similar to the ggplot tool used in R.

Matplotlib is a popular data visualization library in Python. It's often used for creating static, interactive, and animated visualizations in Python. Matplotlib allows you to generate plots, histograms, bar charts, scatter plots, etc., with just a few lines of code.

Program

1. **Arithmetic Operations**

```
# Addition
a = 5
b = 3
sum_result = a + b
print("Sum:", sum_result)

# Subtraction
sub_result = a - b
print("Difference:", sub_result)

# Multiplication
mul_result = a * b
print("Product:", mul_result)

# Division
div_result = a / b
print("Quotient:", div_result)
```

Output

```
Sum: 8
Difference: 2
Product: 15
Quotient: 1.6666666666666667
```

2. **Lists**

```
# Creating a list
numbers = [1, 2, 3, 4, 5]

# Accessing elements
first_element = numbers[0]
print("First Element:", first_element)

# Appending an element
numbers.append(6)
print("List after appending:", numbers)

# Slicing the list
sub_list = numbers[1:4]
print("Sliced List:", sub_list)
```

Output

```
First Element: 1
List after appending: [1, 2, 3, 4, 5, 6]
Sliced List: [2, 3, 4]
```

3. Using Loops

```
# Using a for loop to print numbers
for i in range(5):
    print("Number:", i)
```

Output

```
Number: 0
Number: 1
Number: 2
Number: 3
Number: 4
```

4. Defining Functions

```
# Defining a function to calculate the square of a number
def square(num):
    return num * num

# Using the function
result = square(4)
print("Square of 4:", result)
```

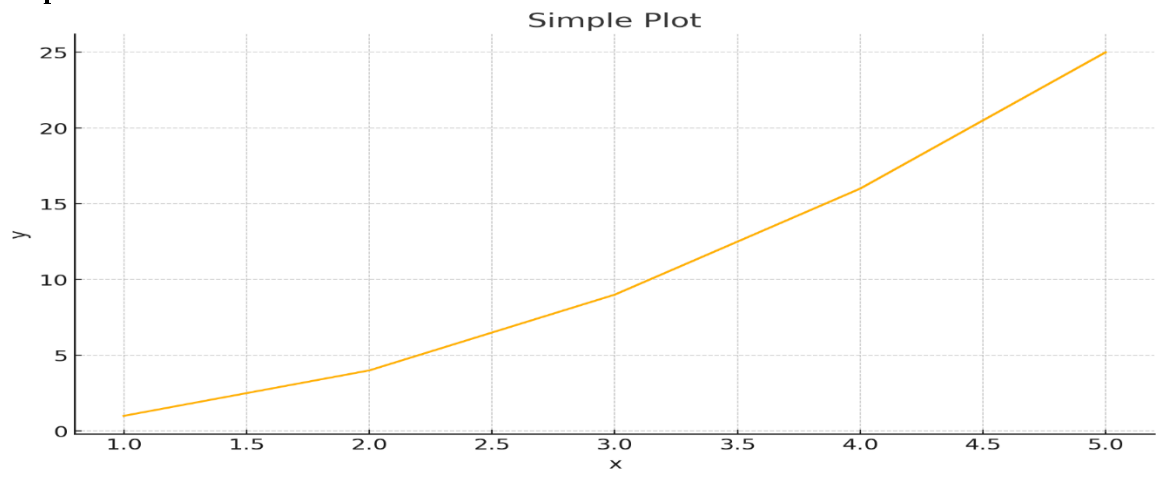
Output

```
Square of 4: 16
```

5. Matplotlib

```
import matplotlib.pyplot as plt
# Data for plotting
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
# Creating a plot
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Simple Plot')
plt.show()
```

Output:



6. <u>Basic String Operations</u>

```
# String concatenation
greeting = "Hello"
name = "Alice"
message = greeting + ", " + name + "!"
print(message)

# String length
length = len(message)
print("Length of message:", length)

# String slicing
slice_message = message[0:5]
print("Sliced message:", slice_message)
```

Output

```
Hello, Alice!
Length of message: 13
Sliced message: Hello
```

7. <u>Using Numpy</u>

```
import numpy as np
# Creating a NumPy array
array = np.array([1, 2, 3, 4, 5])

# Array operations
array_sum = np.sum(array)
array_mean = np.mean(array)
array_squared = np.square(array)

print("Sum of array:", array_sum)
print("Mean of array:", array_mean)
print("Squared array:", array_squared)
```

Output

```
Sum of array: 15
Mean of array: 3.0
Squared array: [ 1  4  9 16 25]
```

8. <u>Using Pandas</u>

```
import pandas as pd

# Sample data for demonstration
data = {'Name': ['John', 'Anna', 'Peter', 'Linda', 'James'],
        'Age': [28, 22, 35, 32, 29],
        'City': ['New York', 'Paris', 'Berlin', 'London', 'Toronto']}

# Creating DataFrame
df = pd.DataFrame(data)

# View first few rows of the DataFrame
print("DataFrame:\n", df)
```

```
# Get summary statistics
print("\nSummary Statistics:\n", df.describe()
# Filter rows where Age is greater than 30
print("\nRows where Age > 30:\n", df[df['Age'] > 30])
```

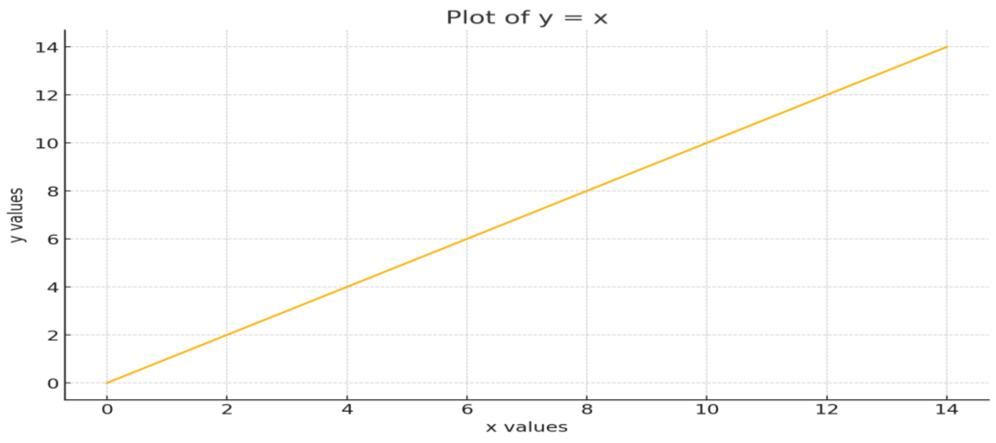
Output

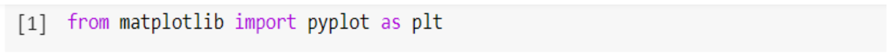
```
DataFrame:
    Name  Age      City
0   John   28  New York
1   Anna   22     Paris
2  Peter   35    Berlin
3  Linda   32    London
4  James   29   Toronto

Summary Statistics:
            Age
count   5.000000
mean   29.200000
std     4.604346
min    22.000000
25%    28.000000
50%    29.000000
75%    32.000000
max    35.000000
Rows where Age > 30:
    Name  Age    City
2  Peter   35  Berlin
3  Linda   32  London
```

9. <u>Matplotlib</u>

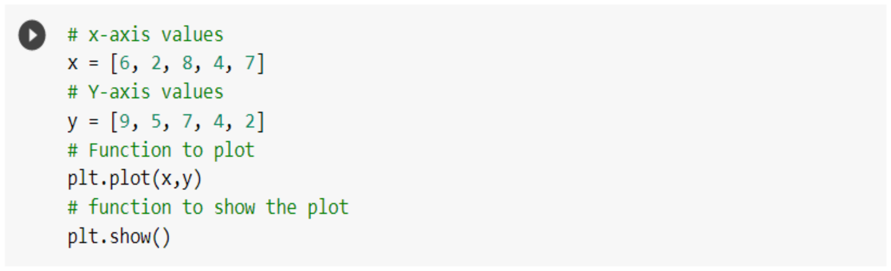
```
import numpy as np
import matplotlib.pyplot as plt
//This line imports the NumPy library (for numerical operations) and the Matplotlib library (for plotting).
plt.plot(np.arange(15), np.arange(15))
//np.arange(15) generates an array of integers from 0 to 14 (inclusive).
plt.plot(x, y) creates a line plot where x and y are arrays of values. Here, both the x-values and y-values are np.arange(15).
plt.show()
```

Output

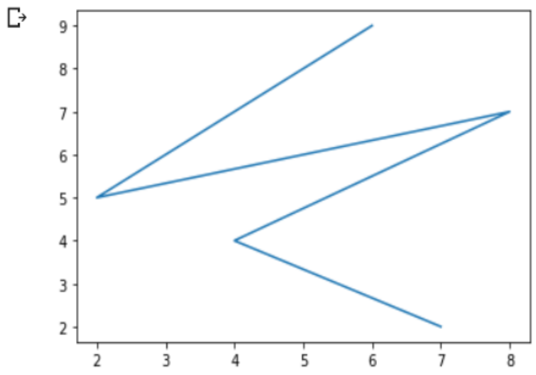


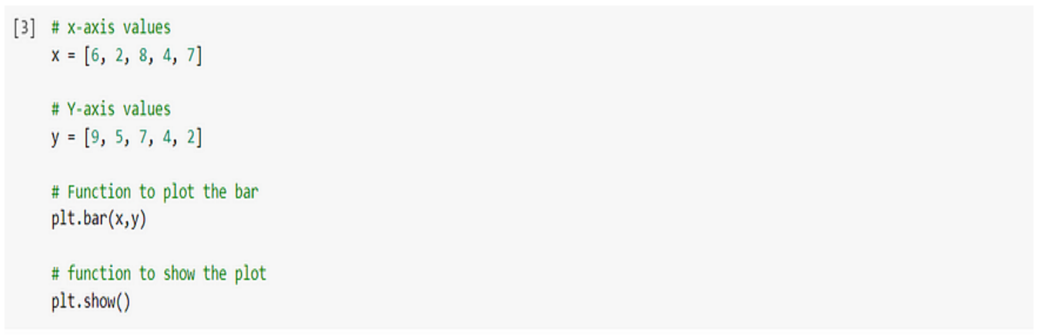


```
[1] from matplotlib import pyplot as plt
```

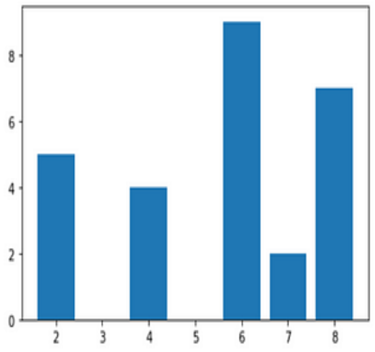


```
# x-axis values
x = [6, 2, 8, 4, 7]
# Y-axis values
y = [9, 5, 7, 4, 2]
# Function to plot
plt.plot(x,y)
# function to show the plot
plt.show()
```





```
[3] # x-axis values
    x = [6, 2, 8, 4, 7]

    # Y-axis values
    y = [9, 5, 7, 4, 2]

    # Function to plot the bar
    plt.bar(x,y)

    # function to show the plot
    plt.show()
```



Result:

Exp No: 2 **Date:**

TO STUDY AND EXPLORE PREPROCESSING OF TEXT (TOKENIZATION, FILTRATION, SCRIPT, VALIDATION, STOP WORD REMOVAL, STEMMING)

Aim:
To study and explore Preprocessing of text (Tokenization, Filtration, Script, Validation, Stop word Removal, Stemming)

Algorithm:
Algorithm: NLP Tokenization, Stopword Removal, and Stemming
Step 1: Import Required Libraries:

- Import the necessary modules from the Natural Language Toolkit (NLTK):
- nltk.corpus.stopwords
- nltk.stem.PorterStemmer
- nltk.tokenize.word_tokenize

Step 2: Download NLTK Resources (if required):

- Use the nltk.download() function to download the required resources:
- 'punkt' for tokenization.
- 'stopwords' for stop word removal.

Step 3: Input Text:

- Define the input text, which is a string containing sentences in natural language.

Step 4: Tokenization:

- Use the word_tokenize() function to split the input text into individual tokens (words and punctuation marks).

Step 5: Filtration (Remove Non-Alphanumeric Tokens):

- Loop through the list of tokens and filter out any token that is not alphanumeric using the .isalnum() method. Store the filtered tokens in a new list.

Step 6: Script Handling (Convert Tokens to Lowercase):

- Convert all tokens to lowercase by iterating over the filtered list and applying the .lower() function to each token. Store the result in another list.

Step 7: Validation (Remove Short Words):

- Remove tokens that are too short (e.g., length ≤ 1). Keep only valid words by checking the length of each token. Store the valid tokens in a new list.

Step 8: Stop Word Removal:

- Define a set of stop words using stopwords.words('english').
- Remove tokens that are stop words by iterating over the list of valid tokens and checking if the token exists in the stop word set. Store the result in a new list.

Step 9: Stemming:

- Create an instance of the PorterStemmer class.
- Apply stemming to each token in the list of tokens without stop words. Use the stem() method to convert each word to its stem (root) form. Store the stemmed tokens in a new list.

Step 10: Display Results:

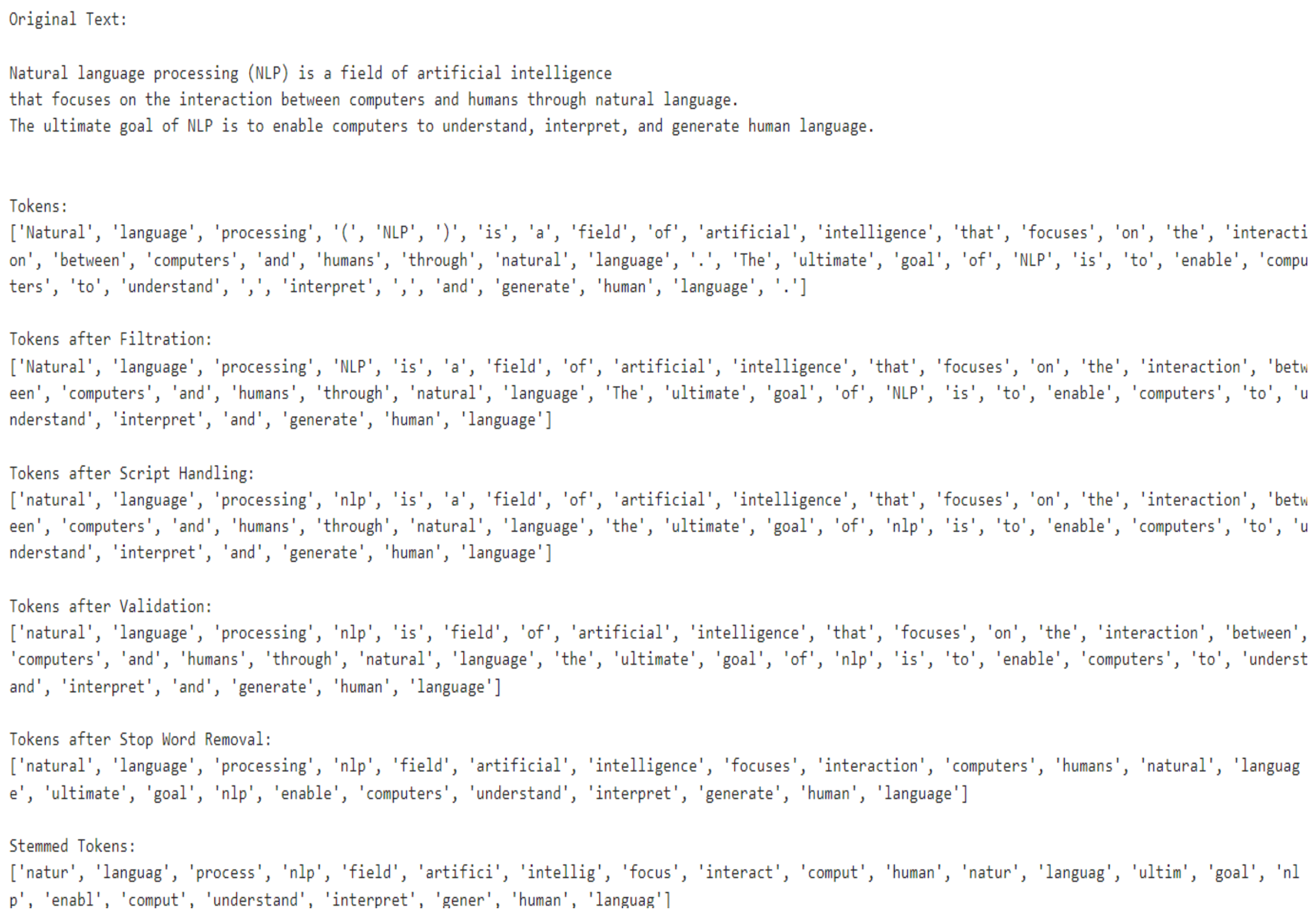
- Print the original text.

- Print the tokens at each stage of the process (after tokenization, filtration, script handling, validation, stop word removal, and stemming).

Program

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

# Download NLTK resources if not already installed
nltk.download('punkt')
nltk.download('stopwords')

# Sample text
text = """
Natural language processing (NLP) is a field of artificial intelligence
that focuses on the interaction between computers and humans through natural language.
The ultimate goal of NLP is to enable computers to understand, interpret, and generate human language.
"""

# Tokenization
tokens = word_tokenize(text)

# Filtration: Remove non-alphanumeric tokens (punctuation, etc.)
tokens_filtr = [word for word in tokens if word.isalnum()]

# Script Handling: Convert to lowercase
tokens_script = [word.lower() for word in tokens_filtr]

# Validation: Check if tokens are valid words (e.g., length greater than 1)
tokens_valid = [word for word in tokens_script if len(word) > 1]

# Stop Word Removal
stop_words = set(stopwords.words('english'))
tokens_no_stopwords = [word for word in tokens_valid if word not in stop_words]

# Stemming
stemmer = PorterStemmer()
stemmed_tokens = [stemmer.stem(word) for word in tokens_no_stopwords]

# Display results
print("Original Text:")
print(text)
print("\nTokens:")
print(tokens)
print("\nTokens after Filtration:")
print(tokens_filtr)
print("\nTokens after Script Handling:")
print(tokens_script)
print("\nTokens after Validation:")
print(tokens_valid)

print("\nTokens after Stop Word Removal:")
print(tokens_no_stopwords)
print("\nStemmed Tokens:")
print(stemmed_tokens)
```

Output:



```
Original Text:

Natural language processing (NLP) is a field of artificial intelligence
that focuses on the interaction between computers and humans through natural language.
The ultimate goal of NLP is to enable computers to understand, interpret, and generate human language.


Tokens:
['Natural', 'language', 'processing', '(', 'NLP', ')', 'is', 'a', 'field', 'of', 'artificial', 'intelligence', 'that', 'focuses', 'on', 'the', 'interacti
on', 'between', 'computers', 'and', 'humans', 'through', 'natural', 'language', '.', 'The', 'ultimate', 'goal', 'of', 'NLP', 'is', 'to', 'enable', 'compu
ters', 'to', 'understand', ',', 'interpret', ',', 'and', 'generate', 'human', 'language', '.']

Tokens after Filtration:
['Natural', 'language', 'processing', 'NLP', 'is', 'a', 'field', 'of', 'artificial', 'intelligence', 'that', 'focuses', 'on', 'the', 'interaction', 'betw
een', 'computers', 'and', 'humans', 'through', 'natural', 'language', 'The', 'ultimate', 'goal', 'of', 'NLP', 'is', 'to', 'enable', 'computers', 'to', 'u
nderstand', 'interpret', 'and', 'generate', 'human', 'language']

Tokens after Script Handling:
['natural', 'language', 'processing', 'nlp', 'is', 'a', 'field', 'of', 'artificial', 'intelligence', 'that', 'focuses', 'on', 'the', 'interaction', 'betw
een', 'computers', 'and', 'humans', 'through', 'natural', 'language', 'the', 'ultimate', 'goal', 'of', 'nlp', 'is', 'to', 'enable', 'computers', 'to', 'u
nderstand', 'interpret', 'and', 'generate', 'human', 'language']

Tokens after Validation:
['natural', 'language', 'processing', 'nlp', 'is', 'field', 'of', 'artificial', 'intelligence', 'that', 'focuses', 'on', 'the', 'interaction', 'between',
'computers', 'and', 'humans', 'through', 'natural', 'language', 'the', 'ultimate', 'goal', 'of', 'nlp', 'is', 'to', 'enable', 'computers', 'to', 'underst
and', 'interpret', 'and', 'generate', 'human', 'language']

Tokens after Stop Word Removal:
['natural', 'language', 'processing', 'nlp', 'field', 'artificial', 'intelligence', 'focuses', 'interaction', 'computers', 'humans', 'natural', 'languag
e', 'ultimate', 'goal', 'nlp', 'enable', 'computers', 'understand', 'interpret', 'generate', 'human', 'language']

Stemmed Tokens:
['natur', 'languag', 'process', 'nlp', 'field', 'artifici', 'intellig', 'focus', 'interact', 'comput', 'human', 'natur', 'languag', 'ultim', 'goal', 'nl
p', 'enabl', 'comput', 'understand', 'interpret', 'gener', 'human', 'languag']
```

Result:

Exp No: 3 **EXPLORE SOCIAL MEDIA ANALYSIS** **Date:**

Aim: To explore social media analysis using Python

Algorithm:

1. Setup
- Import libraries: googleapiclient, nltk, pandas, emoji.
- Download VADER lexicon from NLTK.
2. Initialize API
- Set the YouTube API key and create a YouTube service object.
- Initialize the Sentiment Intensity Analyzer.
3. Fetch Comments
- Define get_comments(video_id):
  - Call YouTube API to retrieve comments.
  - Loop through pages of comments until all are fetched.
  - Return the list of comments.
4. Analyze Sentiment
- Define analyze_sentiments(comments):
  - For each comment, preprocess and compute sentiment score using VADER.
  - Classify as "Positive", "Negative", or "Neutral".
  - Return a DataFrame of comments with their sentiments and scores.
5. Execution
- Define video_id and fetch comments.
- Analyze sentiments of comments and print a random sample of results.
- Save results to CSV.
6. Evaluate Performance
- Define test_data with labeled comments.
- Predict sentiments for test data and calculate accuracy and classification report.

Program:

```
from googleapiclient.discovery import build
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import pandas as pd
import nltk
import emoji

# Setup
nltk.download('vader_lexicon')
api_key = "AIzaSyCqeXS04S2KFXKZM9g0GFMiIp8YB4nET5E"  # Replace with your YouTube API key
youtube = build('youtube', 'v3', developerKey=api_key)
sid = SentimentIntensityAnalyzer()

# Fetch comments from a YouTube video
def get_comments(video_id):
    comments, response = [], youtube.commentThreads().list(part="snippet", videoId=video_id, textFormat="plainText",
maxResults=100).execute()
    while response:
        comments.extend([item['snippet']['topLevelComment']['snippet']['textDisplay'] for item in response['items']])
        if 'nextPageToken' in response:
            response = youtube.commentThreads().list(part="snippet", videoId=video_id,

pageToken=response['nextPageToken'], textFormat="plainText", maxResults=100).execute()
        else:
            break
    return comments

# Preprocess text and analyze sentiment
def analyze_sentiments(comments):
    results = []
    for comment in comments:
        processed = emoji.demojize(comment)
        score = sid.polarity_scores(processed)['compound']
        sentiment = 'Positive' if score > 0.05 else 'Negative' if score < -0.05 else 'Neutral'
        results.append({'Comment': comment, 'Sentiment': sentiment, 'Score': score})
    return pd.DataFrame(results)

# Example Usage
video_id = "FN5_IVIaBx8"  # Replace with actual video IDs
comments = get_comments(video_id)
sentiment_results = analyze_sentiments(comments)

# Display some comments and their sentiments
print("Sample Comments and Sentiments:")
print(sentiment_results.sample(5))  # Display a random sample of 5 comments

# Save results to CSV
sentiment_results.to_csv("all_comments_sentiment.csv", index=False)

# Performance Evaluation
from sklearn.metrics import accuracy_score, classification_report

# Example labeled test data for evaluation
test_data = [
    {"Comment": "Loved it!", "True_Sentiment": "Positive"},
    {"Comment": "Horrible!", "True_Sentiment": "Negative"},
    {"Comment": "It was okay.", "True_Sentiment": "Neutral"},
]

# Extract true sentiments and analyze predictions
true_labels = [item["True_Sentiment"] for item in test_data]
pred_labels = analyze_sentiments([item["Comment"] for item in test_data])['Sentiment']

# Evaluation
print(f"Accuracy: {accuracy_score(true_labels, pred_labels):.2%}")
print("\nClassification Report:\n", classification_report(true_labels, pred_labels))
```

Output:

```
Sample Comments and Sentiments:
                                             Comment Sentiment   Score
18  Amity ranking was above iit kanpur nashe ma ha...  Positive  0.3400
24  Make happy education minister by supporting th...  Positive  0.9008
37             congratulations .. pride of our nation  Positive  0.7430
13  Please allow 2nd dropper also in jee advanced ...  Positive  0.7579
8   Everything is good in the video except that Fk...  Positive  0.4404
Accuracy: 66.67%

Classification Report:
              precision    recall  f1-score   support

    Negative       1.00      1.00      1.00         1
     Neutral       0.00      0.00      0.00         1
    Positive       0.50      1.00      0.67         1

    accuracy                           0.67         3
   macro avg       0.50      0.67      0.56         3
weighted avg       0.50      0.67      0.56         3
```

Result:

Exp No: 4 **Date:**

IMPLEMENT A PROGRAM FOR NETWORK ACCESS CONTROL TO ILLUSTRATE MALWARE ATTACKS

Aim: To Implement a program for network access control to illustrate malware attacks

Algorithm:

1.Initialize VirusTotal API:
- Set your VirusTotal API key and the base URL for IP address reports.

2.Define check_malicious_ip function:
- Input: IP address
- Action: Sends an API request to VirusTotal to check the IP.
- Output: If the IP is found, check the number of detected malicious URLs associated with it.
- If positive detections > 0, print an alert.
- Else, print that the IP is clean or not found in the database.

3. Define packet_handler function:
- Input: Network packet
- Action:
- Check if the packet has an IP layer.
- Extract source and destination IPs.
- Call check_malicious_ip for both IPs to verify if they're flagged as malicious.

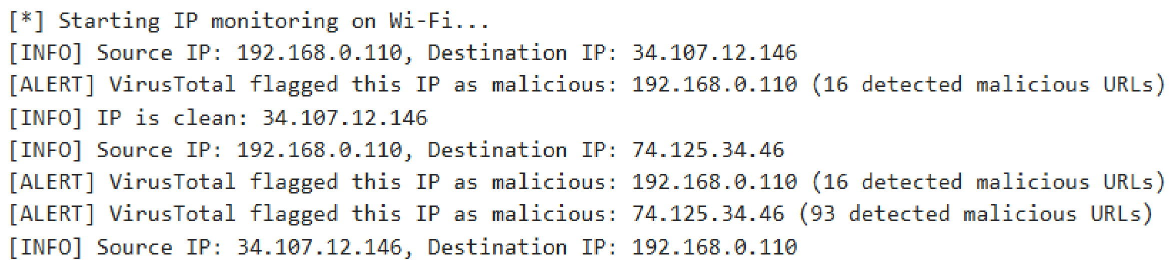
4. Define start_sniffing function:
- Input: Network interface (default: Wi-Fi)
- Action: Use scapy to sniff network packets on the specified interface, sending each packet to packet_handler.

5. Main Execution:
- Call start_sniffing to begin real-time monitoring on the specified network interface.

Program

```
from scapy.all import sniff, IP
import requests

# Your VirusTotal API key
API_KEY = '2399d639231bd6ff55ffd4282ee47b57f5e55d53e89342faa66e51504f480574'

# VirusTotal API base URL for IP address reports
VT_URL = "https://www.virustotal.com/vtapi/v2/ip-address/report"

# Function to check if an IP address is malicious using VirusTotal
def check_malicious_ip(ip):
    params = {'apikey': API_KEY, 'ip': ip}
    try:
        # Send request to VirusTotal API
        response = requests.get(VT_URL, params=params)
        result = response.json()
        if result.get('response_code') == 1:  # IP found in VirusTotal
            positives = len(result.get('detected_urls', []))  # List of detected malicious URLs for this IP
            if positives > 0:

                print(f"[ALERT] VirusTotal flagged this IP as malicious: {ip} ({positives} detected malicious URLs)")
            else:
                print(f"[INFO] IP is clean: {ip}")
        else:
            print(f"[INFO] IP not found in VirusTotal database: {ip}")
    except Exception as e:
        print(f"Error checking IP with VirusTotal: {e}")

# Function to handle each packet captured
def packet_handler(packet):
    if packet.haslayer(IP):  # Check if the packet has an IP layer
        src_ip = packet[IP].src
        dst_ip = packet[IP].dst
        print(f"[INFO] Source IP: {src_ip}, Destination IP: {dst_ip}")

        # Check if the source and destination IPs are malicious
        check_malicious_ip(src_ip)
        check_malicious_ip(dst_ip)

# Start sniffing packets
def start_sniffing(interface="Wi-Fi"):
    print(f"[*] Starting IP monitoring on {interface}...")
    sniff(iface=interface, prn=packet_handler, store=False)

if __name__ == "__main__":
    start_sniffing(interface="Wi-Fi")
```

Output



```
[*] Starting IP monitoring on Wi-Fi...
[INFO] Source IP: 192.168.0.110, Destination IP: 34.107.12.146
[ALERT] VirusTotal flagged this IP as malicious: 192.168.0.110 (16 detected malicious URLs)
[INFO] IP is clean: 34.107.12.146
[INFO] Source IP: 192.168.0.110, Destination IP: 74.125.34.46
[ALERT] VirusTotal flagged this IP as malicious: 192.168.0.110 (16 detected malicious URLs)
[ALERT] VirusTotal flagged this IP as malicious: 74.125.34.46 (93 detected malicious URLs)
[INFO] Source IP: 34.107.12.146, Destination IP: 192.168.0.110
```

Result:

Exp No: 5 **Date:**

IMPLEMENT NAÏVE BAYES SCRATCH IMPLEMENTATION USING PYTHON

Aim: To implement Naïve Bayes Scratch implementation using Python

Algorithm

1. Initialize the Model:
   - Create a class NaiveBayes with methods to fit the model on training data and make predictions on test data.
2. Fit the Model (Training Phase):
   - Input: Training features X and labels y.
   - Step 1: Identify all unique classes in y (e.g., class 0 and class 1).
   - Step 2: For each class:
     - Separate the samples in X that belong to this class.
     - Calculate:
       - The mean of each feature (column) within this class.
       - The variance of each feature within this class.
       - The prior probability of this class (proportion of samples in this class relative to the total number of samples).
     - Store the mean, variance, and prior probability for each class.
3. Define Gaussian Probability Density Function:
   - Purpose: Calculate the probability of a feature value for a given class, assuming a Gaussian (normal) distribution.
   - **Formula:** For a feature value x, mean μ, and variance σ^2:

$$P(x|\text{class}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

4. Make Predictions (Testing Phase):
   - Input: Test samples X_test.
   - Output: Predicted labels for each test sample in X_test.
   - For each test sample:
     - Step 1: For each class, calculate the posterior probability:
       - Prior Probability: Start with the prior probability (log-transformed to avoid underflow).
       - Class Conditional Probability: For each feature, calculate the Gaussian probability density based on the feature's mean and variance for this class. Take the log of each probability and sum them up.
       - Add the prior and the sum of log-likelihoods to get the total posterior for the class.
     - Step 2: Choose the class with the highest posterior probability as the predicted label for the sample.
5. Output Predictions:
   - Return the predicted class labels for all samples in X_test.

Program

```
import numpy as np

class NaiveBayes:
    def fit(self, X, y):
        # Separate data by class and calculate mean, variance, and prior for each class
        self.classes = np.unique(y)
        self.stats = {
            cls: {
                "mean": X[y == cls].mean(axis=0),
                "var": X[y == cls].var(axis=0),
                "prior": len(X[y == cls]) / len(X)
            }
            for cls in self.classes
        }

    def _gaussian_density(self, mean, var, x):
        # Gaussian probability density function
        return (1 / np.sqrt(2 * np.pi * var)) * np.exp(-((x - mean) ** 2) / (2 * var))

    def predict(self, X):
        # Predict class for each sample in X
        y_pred = []
        for x in X:
            posteriors = []
            for cls, params in self.stats.items():
                prior = np.log(params["prior"])
                likelihood = np.sum(np.log(self._gaussian_density(params["mean"], params["var"], x)))
                posteriors.append(prior + likelihood)
            y_pred.append(self.classes[np.argmax(posteriors)])
        return np.array(y_pred)

# Example usage
X = np.array([[1, 2], [2, 3], [3, 4], [6, 5], [7, 8], [8, 9]])
y = np.array([0, 0, 0, 1, 1, 1])  # Labels: 0 or 1

model = NaiveBayes()
model.fit(X, y)

X_test = np.array([[2, 3], [7, 6]])
print("Predictions:", model.predict(X_test))
```

Output: Predictions: [0 1]

Result:

Exp No: 6 **Date:**

DESIGN A SPAM DETECTION ALGORITHM

Aim: To design a spam detection algorithm

Algorithm

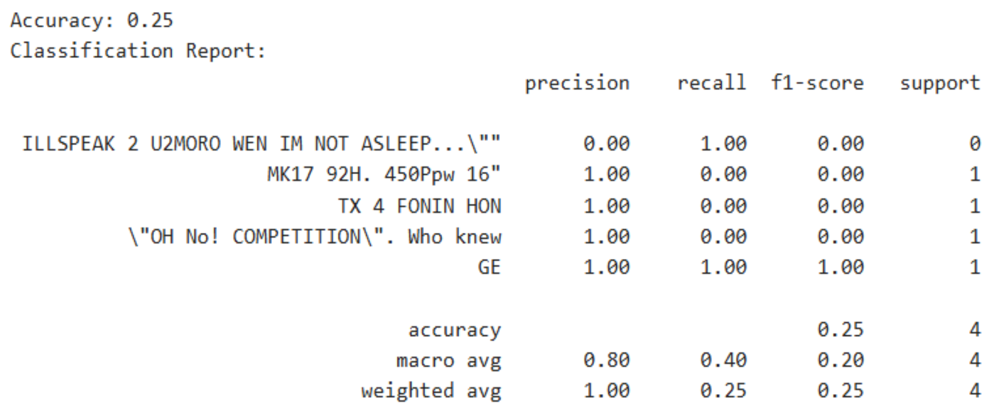
1. Import libraries: pandas, nltk for tokenization and lemmatization, sklearn for model training and evaluation.
2. Read the CSV file containing the spam messages into a DataFrame.
3. Data Cleaning
   - Replace NaN values in the type column with empty strings.
   - Ensure the text column is of string type.
4. Filter Out Empty Labels
   - Remove rows where the type column is empty.
5. Define Text Preprocessing Functions
   - Create a function to tokenize the text.
   - Create a function to lemmatize tokens.
6. Apply Text Preprocessing
   - Tokenize the text column and store tokens in a new column.
   - Lemmatize the tokens and store them in another new column.
7. Vectorize the Text Data
   - Use CountVectorizer to convert the text data into a numerical format (bag-of-words model).
   - Separate the features (X) and labels (y).
8. Split the Data into Training and Testing Sets
   - Divide the dataset into training (70%) and testing (30%) sets using train_test_split.
9. Train the Naive Bayes Classifier
   - Initialize the MultinomialNB classifier.
   - Fit the classifier to the training data.
10. Make Predictions on the Test Set
    - Use the trained classifier to predict labels for the test set.
11. Evaluate the Model
    - Calculate accuracy using accuracy_score.
    - Generate a classification report using classification_report.
12. Output the Results
    - Print the accuracy and classification report to evaluate model performance.

Program

```
import pandas as pd
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
```

```
# Load the dataset
sms = pd.read_csv('C:/Users/dhari/OneDrive/Desktop/AI IN CS/spam-naive bayes and NLP.csv', sep=',',
names=["type", "text"], encoding='ISO-8859-1')

# Ensure all entries in 'type' and 'text' are filled and clean
sms['type'] = sms['type'].fillna('')  # Fill any NaN values in type with empty strings
sms['text'] = sms['text'].fillna('').astype(str)  # Fill NaN in text and ensure it is string

# Filter rows where 'type' is empty
sms = sms[sms['type'] != '']

# Preprocessing functions
def get_tokens(text):
    tokens = word_tokenize(text)
    return tokens

def get_lemmas(tokens):
    lemmatizer = WordNetLemmatizer()
    lemmas = [lemmatizer.lemmatize(token) for token in tokens]
    return lemmas

# Apply tokenization and lemmatization
sms['tokens'] = sms['text'].apply(get_tokens)
sms['lemmas'] = sms['tokens'].apply(get_lemmas)

# Vectorize the text data using CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(sms['text'])  # Use original text data for vectorization
y = sms['type']  # Labels (spam or ham)

# Split the data 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)

# Train a Naive Bayes classifier
classifier = MultinomialNB()
classifier.fit(X_train, y_train)

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

# Evaluate the model with zero_division set to 1
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, zero_division=1)  # Adds zero_division=1 to handle undefined
metrics
# Output the results
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)
```

Output



```
Accuracy: 0.25
Classification Report:
                                             precision    recall  f1-score   support

 ILLSPEAK 2 U2MORO WEN IM NOT ASLEEP...\""       0.00      1.00      0.00         0
                     MK17 92H. 450Ppw 16"        1.00      0.00      0.00         1
                           TX 4 FONIN HON        1.00      0.00      0.00         1
         \"OH No! COMPETITION\". Who knew        1.00      0.00      0.00         1
                                       GE        1.00      1.00      1.00         1

                                 accuracy                            0.25         4
                                macro avg        0.80      0.40      0.20         4
                             weighted avg        1.00      0.25      0.25         4
```

Result:

Exp No: 7 **Date:**

EXPLORE KEYSTROKE RECOGNITION USING NLP

Aim: To explore Keystroke recognition using NLP

Algorithm:

1. Import Libraries: Import numpy, pandas, matplotlib, and sklearn libraries for data handling, plotting, and modeling.
2. Load Dataset: Load keystroke dataset from the URL into a DataFrame.
3. Calculate Latency Averages: Extract latency columns (starting with "DD"), calculate average latency per subject, and plot the first six subjects.
4. Split Data: Split data into 80% training and 20% testing sets.
5. Define Features and Labels: Set feature (X_train, X_test) and label (y_train, y_test) sets for training and testing.
6. Train Classifiers: Train three classifiers (KNN, SVC, MLP) on the training set, predict on X_test, and calculate accuracy for each.
7. Display Accuracies: Print accuracy scores for each classifier.
8. Confusion Matrix: Create and plot a confusion matrix for the predictions of the last classifier.

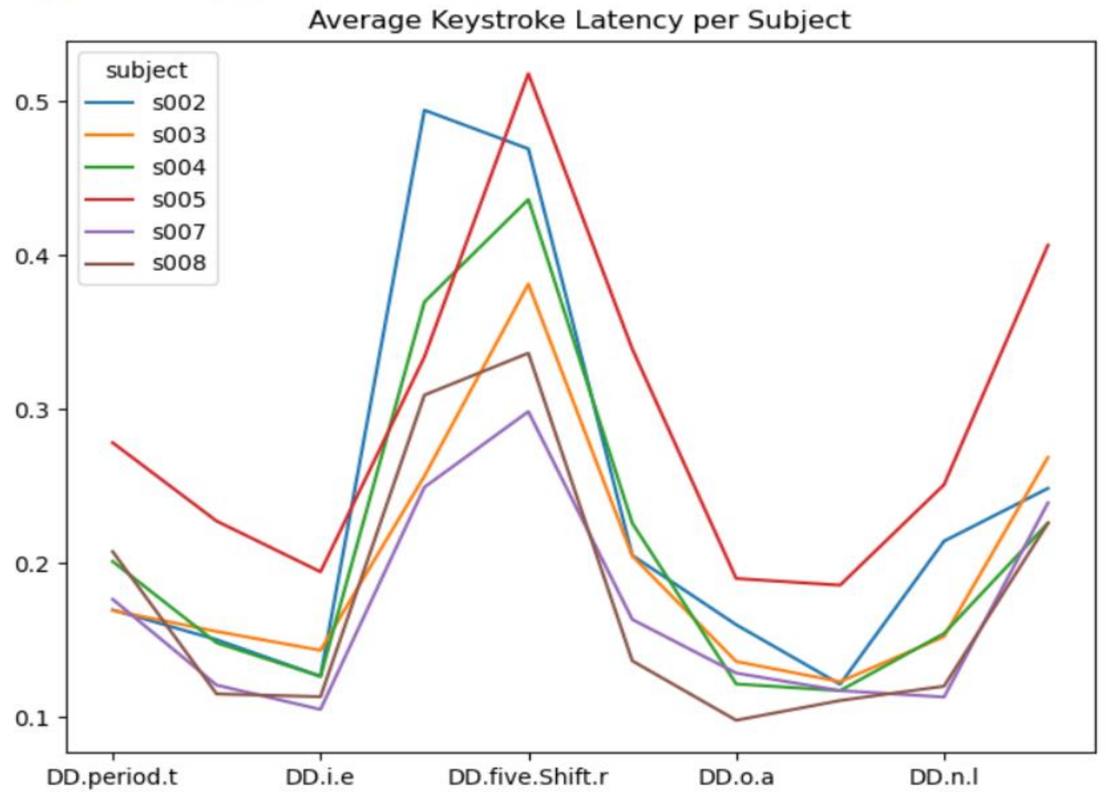
Program

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn.neural_network import MLPClassifier
pwd_data = pd.read_csv("C:/Users/dhari/OneDrive/Desktop/AI IN CS/DSL-StrongPasswordData.csv", header
= 0)
# Average Keystroke Latency per Subject
DD = [dd for dd in pwd_data.columns if dd.startswith('DD')]
plot = pwd_data[DD]
plot['subject'] = pwd_data['subject'].values
plot = plot.groupby('subject').mean()
plot.iloc[:6].T.plot(figsize=(8, 6), title='Average Keystroke Latency per Subject')
data_train, data_test = train_test_split(pwd_data, test_size = 0.2,
random_state=0)
X_train = data_train[pwd_data.columns[2:]]
y_train = data_train['subject']
X_test = data_test[pwd_data.columns[2:]]
y_test = data_test['subject']
# K-Nearest Neighbor Classifier
```

```
knc = KNeighborsClassifier()

knc.fit(X_train, y_train)
y_pred = knc.predict(X_test)
knc_accuracy = metrics.accuracy_score(y_test, y_pred)
print('K-Nearest Neighbor Classifier Accuracy:', knc_accuracy)
svc = svm.SVC(kernel='linear')
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)
svc_accuracy = metrics. accuracy_score(y_test, y_pred)
print('Support Vector Linear Classifier Accuracy:', svc_accuracy)
mlpc = MLPClassifier()
mlpc.fit(X_train,y_train)
y_pred = mlpc.predict(X_test)
mlpc_accuracy = metrics.accuracy_score(y_test, y_pred)
print('Multi Layer Perceptron Classifier Accuracy:', mlpc_accuracy)
```

Output

```
K-Nearest Neighbor Classifier Accuracy: 0.3730392156862745
Support Vector Linear Classifier Accuracy: 0.7629901960784313
```



```
Multi Layer Perceptron Classifier Accuracy: 0.907843137254902
```

Result:

Exp No: 8 **Date:**

IMPLEMENT REAL-TIME FACIAL RECOGNITION SOFTWARE POWERED BY AI

Aim: To implement Real-time Facial Recognition Software powered by AI.

Algorithm:

1. Import Libraries:
   - Import necessary libraries: NumPy, Pandas, Matplotlib, and scikit-learn.
2. Load Dataset:
   - Load the LFW dataset using fetch_lfw_people().
3. Preprocess Data:
   - Reshape images to a 2D array (flatten).
   - Split data into training and testing sets using train_test_split().
4. Apply PCA:
   - Initialize PCA with a specified number of components (e.g., 150).
   - Fit PCA on the training data and transform both training and testing sets.
5. Train Classifier:
   - Initialize a classifier (e.g., RandomForestClassifier).
   - Fit the classifier on the PCA-transformed training data.
6. Predict Labels:
   - Predict labels for the PCA-transformed test set using the trained classifier.
7. Evaluate Model:
   - Generate a classification report and confusion matrix to assess model performance.
8. Visualize Results:
   - Plot the confusion matrix and display sample test images with their true and predicted labels.

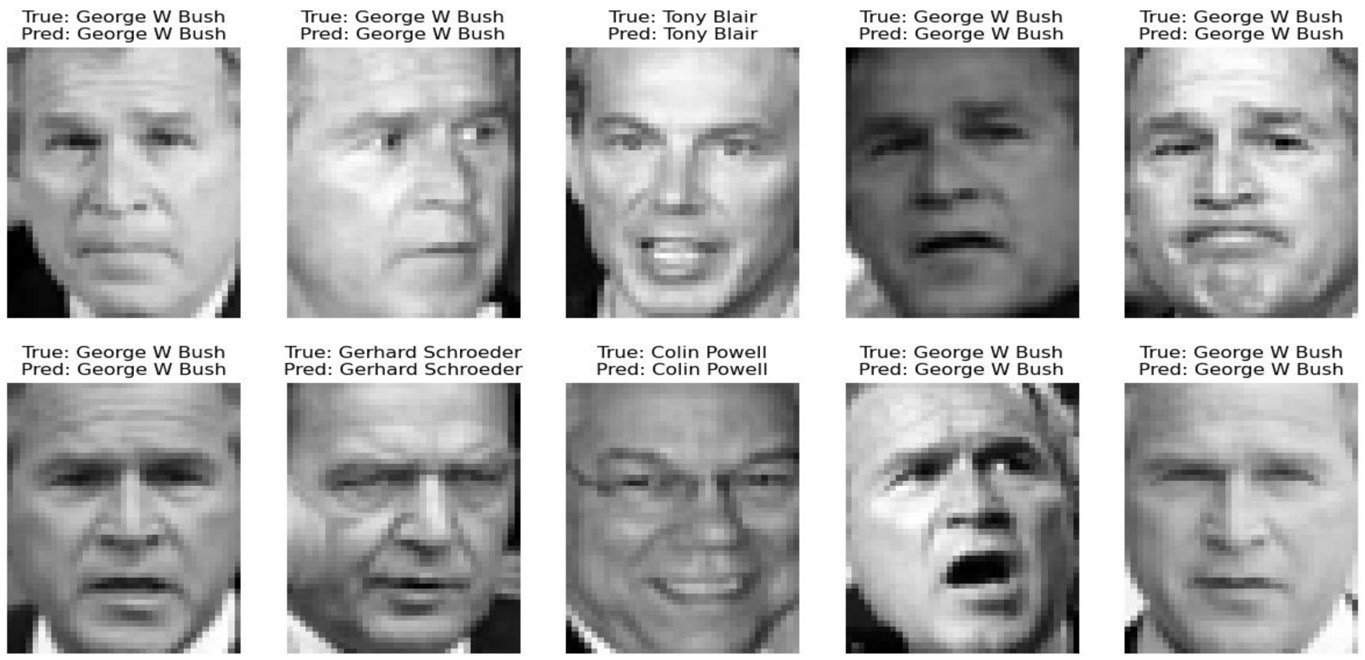
Program

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix

# Fetch the LFW dataset
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)

# Load the images and target labels
X = lfw_people.data
y = lfw_people.target
target_names = lfw_people.target_names
```

```
n_classes = target_names.shape[0]
print(f"Number of classes: {n_classes}")
print(f"Images shape: {lfw_people.images.shape}")
print(f"Number of samples: {X.shape[0]}")

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Apply PCA (Principal Component Analysis) for dimensionality reduction
n_components = 150
print("Extracting the top %d eigenfaces from %d faces" % (n_components, X_train.shape[0]))

pca = PCA(n_components=n_components, whiten=True).fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

# Train a Support Vector Machine (SVM) classifier
svc = SVC(kernel='rbf', class_weight='balanced', C=1.0, gamma='auto')
svc.fit(X_train_pca, y_train)

# Make predictions on the test set
y_pred = svc.predict(X_test_pca)

# Display classification report and confusion matrix
print("Classification report:")
print(classification_report(y_test, y_pred, target_names=target_names))

print("Confusion matrix:")
print(confusion_matrix(y_test, y_pred))

# Plot some results
fig, ax = plt.subplots(2, 5, figsize=(15, 8))
for i in range(10):
    ax[i // 5, i % 5].imshow(X_test[i].reshape(50, 37), cmap='gray')
    ax[i // 5, i % 5].set_title(f'True: {target_names[y_test[i]]}\nPred: {target_names[y_pred[i]]}')
    ax[i // 5, i % 5].axis('off')

plt.show()
```

Output:

```
Number of classes: 7
Images shape: (1288, 50, 37)
Number of samples: 1288
Extracting the top 150 eigenfaces from 966 faces
Classification report:
                   precision    recall  f1-score   support

     Ariel Sharon       1.00      0.54      0.70        13
     Colin Powell       0.67      0.97      0.79        60
  Donald Rumsfeld       0.89      0.63      0.74        27
    George W Bush       0.91      0.92      0.91       146
Gerhard Schroeder       0.87      0.80      0.83        25
      Hugo Chavez       1.00      0.53      0.70        15
       Tony Blair       0.94      0.81      0.87        36

         accuracy                           0.85       322
        macro avg       0.90      0.74      0.79       322
     weighted avg       0.87      0.85      0.85       322

Confusion matrix:
[[  7   3   0   3   0   0   0]
 [  0  58   1   1   0   0   0]
 [  0   5  17   5   0   0   0]
 [  0  11   0 134   0   0   1]
 [  0   2   0   2  20   0   1]
 [  0   5   0   0   2   8   0]
 [  0   3   1   2   1   0  29]]
```



Result:

Exp No: 9 **Date:**

EXPLORE IMAGE BASED RECOGNITION USING SOFTWARE POWERED AI

Aim: To explore Image Based Recognition using Software powered AI.

Algorithm:

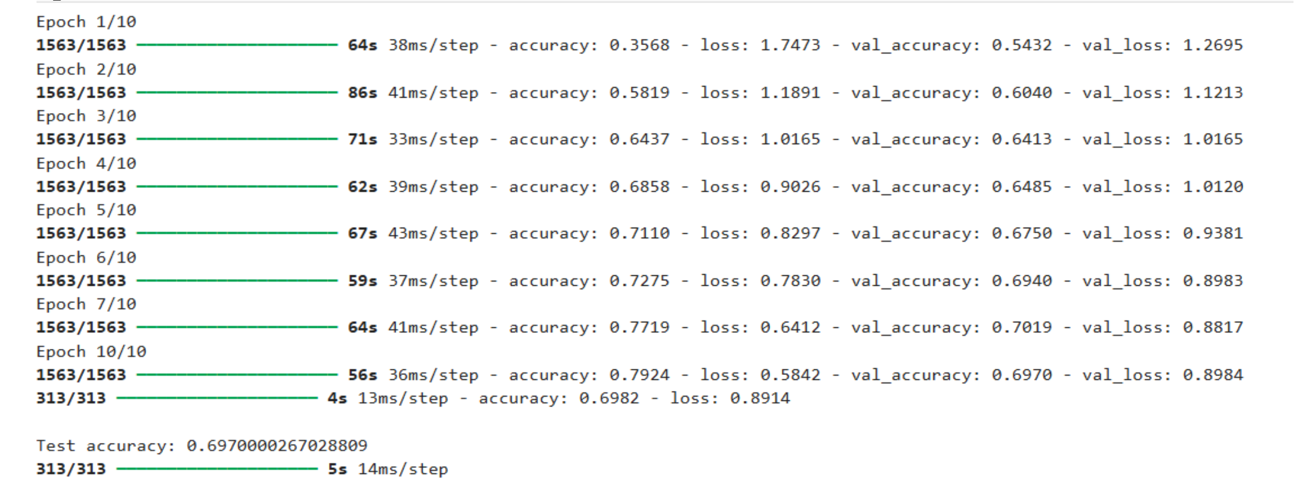
1. Import Libraries: Imports necessary libraries including TensorFlow and Matplotlib.
2. Load the Dataset: Loads the CIFAR-10 dataset, which is included in TensorFlow.
3. Preprocess the Data: Normalizes the image data to the range [0, 1].
4. Build the CNN Model: Defines a simple CNN architecture with convolutional, pooling, flattening, and dense layers.
5. Compile the Model: Compiles the model with the Adam optimizer and Sparse Categorical Crossentropy loss function.
6. Train the Model: Fits the model to the training data over 10 epochs, using a validation set from the test data.
7. Evaluate the Model: Evaluates the model's accuracy on the test data.
8. Make Predictions: Uses the trained model to make predictions on the test data.
9. Visualize Predictions: Plots the first 15 images from the test set along with their predicted and true labels.

Program

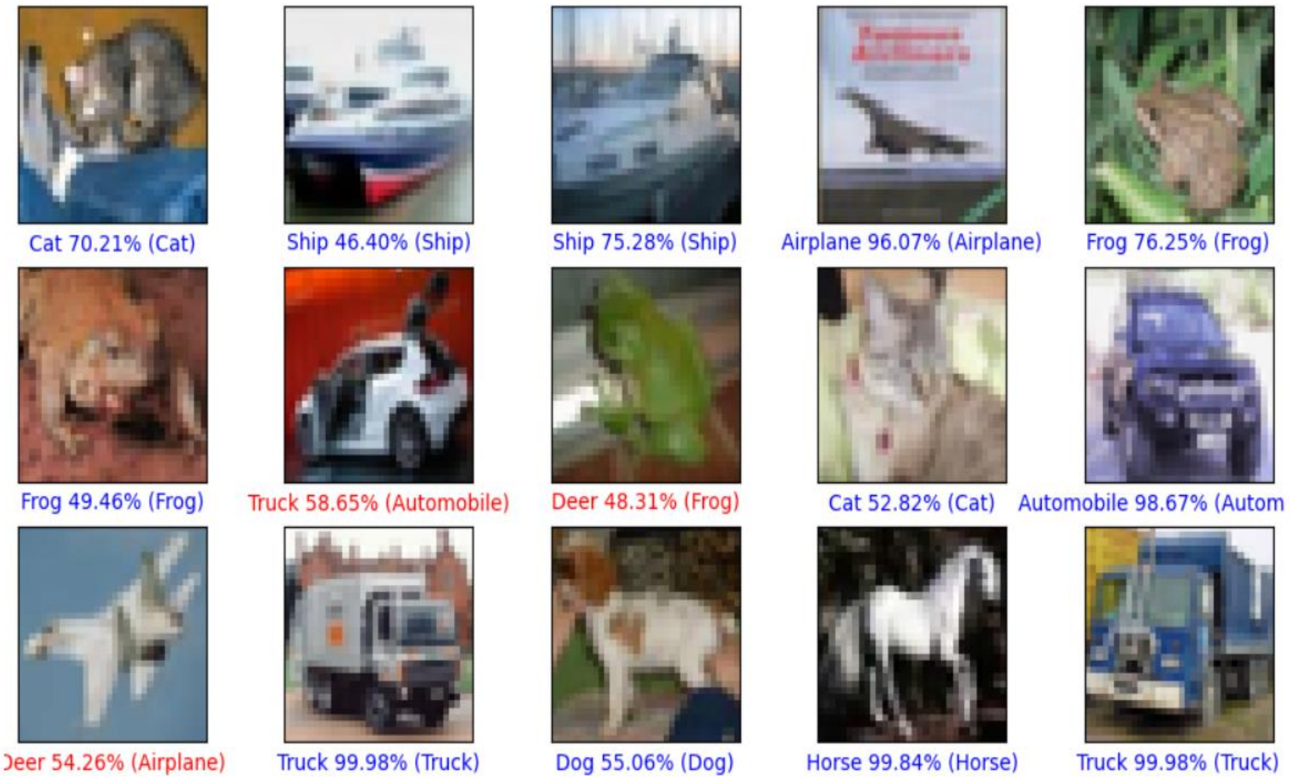
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

# Load the CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images = train_images / 255.0
test_images = test_images / 255.0

# Define the class names in CIFAR-10
class_names = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Horse', 'Ship', 'Truck']

# Build the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10)
])

# Compile the model
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Train the model
model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print('\nTest accuracy:', test_acc)

# Make predictions
predictions = model.predict(test_images)

# Visualize some predictions
def plot_image(i, predictions_array, true_label, img):
    predictions_array, true_label, img = predictions_array[i], true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img)
    predicted_label = class_names[np.argmax(predictions_array)]
    true_label = class_names[true_label[0]]
    color = 'blue' if predicted_label == true_label else 'red'
    plt.xlabel(f'{predicted_label} ({true_label})', color=color)
# Plot the first 15 images, their predicted labels, and the true labels
plt.figure(figsize=(15, 5))
for i in range(15):
    plt.subplot(3, 5, i + 1)
    plot_image(i, predictions, test_labels, test_images)
plt.show()
```

Output:



```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 64s 38ms/step - accuracy: 0.3568 - loss: 1.7473 - val_accuracy: 0.5432 - val_loss: 1.2695
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 86s 41ms/step - accuracy: 0.5819 - loss: 1.1891 - val_accuracy: 0.6040 - val_loss: 1.1213
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 71s 33ms/step - accuracy: 0.6437 - loss: 1.0165 - val_accuracy: 0.6413 - val_loss: 1.0165
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 62s 39ms/step - accuracy: 0.6858 - loss: 0.9026 - val_accuracy: 0.6485 - val_loss: 1.0120
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 67s 43ms/step - accuracy: 0.7110 - loss: 0.8297 - val_accuracy: 0.6750 - val_loss: 0.9381
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 59s 37ms/step - accuracy: 0.7275 - loss: 0.7830 - val_accuracy: 0.6940 - val_loss: 0.8983
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 64s 41ms/step - accuracy: 0.7719 - loss: 0.6412 - val_accuracy: 0.7019 - val_loss: 0.8817
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 56s 36ms/step - accuracy: 0.7924 - loss: 0.5842 - val_accuracy: 0.6970 - val_loss: 0.8984
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - accuracy: 0.6982 - loss: 0.8914

Test accuracy: 0.6970000267028809
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step
```


Cat 70.21% (Cat)
Ship 46.40% (Ship)
Ship 75.28% (Ship)
Airplane 96.07% (Airplane)
Frog 76.25% (Frog)
Frog 49.46% (Frog)
Truck 58.65% (Automobile)
Deer 48.31% (Frog)
Cat 52.82% (Cat)
Automobile 98.67% (Autom
Deer 54.26% (Airplane)
Truck 99.98% (Truck)
Dog 55.06% (Dog)
Horse 99.84% (Horse)
Truck 99.98% (Truck)

Result:

Exp No: 10 **Date:**

DESIGN FRAUD DETECTION ALGORITHM

Aim: To design a fraud detection algorithm using Orange tool

Introduction to Orange Tool

Orange is an open-source data visualization and analysis tool designed for both novice and expert users. It provides a user-friendly graphical interface for data mining and machine learning, allowing users to create workflows by connecting various data processing and modeling components. Orange's flexibility makes it suitable for tasks such as data visualization, classification, clustering, and regression, and it supports various machine learning algorithms, including Random Forest, SVM, KNN, and more.

Orange is particularly beneficial for educational purposes, enabling students and researchers to explore machine learning concepts without extensive coding knowledge. Its visual programming approach simplifies the experimentation process, allowing users to focus on the analysis and results.

Step-by-Step Procedure to Upload Credit Card Fraud CSV and Apply Random Forest Algorithm in Orange

Step 1: Install Orange

Download and Install:

- Go to the Orange website and download the installer for your operating system (Windows, macOS, or Linux).
- Follow the installation instructions to set up Orange on your computer.

Step 2: Launch Orange

Open Orange:

- Start the Orange application after installation.

Step 3: Upload the Credit Card Fraud CSV File

Add the File Widget:

- In the Orange canvas, locate the "File" widget in the left sidebar under the "Data" category.
- Drag and drop the File widget onto the canvas.

Load the CSV File:

- Double-click the File widget to open its settings.
- Click on "Browse" and navigate to the location of your Credit Card Fraud CSV file.
- Select the CSV file and click Open.
- Click OK to load the dataset.

Step 4: Data Preprocessing (Optional)

Check Data Quality:

- You can use the Data Table widget to visualize the loaded dataset.
- Add a Data Table widget to the canvas and connect it to the File widget.
- Double-click the Data Table widget to view the data and check for missing values or inconsistencies.

Preprocess Data:

- If necessary, you can use the Select Columns, Edit Domain, or Impute widgets to preprocess your data, such as selecting relevant features or handling missing values.

Step 5: Apply Random Forest Algorithm
Add the Random Forest Widget:

- From the left sidebar, locate the "Random Forest" widget under the “Model” category.
- Drag and drop the Random Forest widget onto the canvas.

Connect the Widgets:

- Connect the Data Table (or the File widget if you didn’t use a Data Table) to the Random Forest widget.

Set Target Variable:

- Double-click the Random Forest widget to open its settings.
- Select the target variable (e.g., "fraud" or "is_fraud") from your dataset to train the model on.

Step 6: Evaluate the Model with Confusion Matrix
Add a Test & Score Widget:

- Locate the "Test & Score" widget in the left sidebar under the “Evaluate” category.
- Drag and drop the Test & Score widget onto the canvas.

Connect Widgets:

- Connect the Random Forest widget to the Test & Score widget.

Configure Test & Score:

- Double-click the Test & Score widget to open its settings.
- Ensure that the Random Forest is selected as the model for evaluation.
- Select an evaluation method, such as Cross Validation or Train/Test Split.

Add a Confusion Matrix Widget:

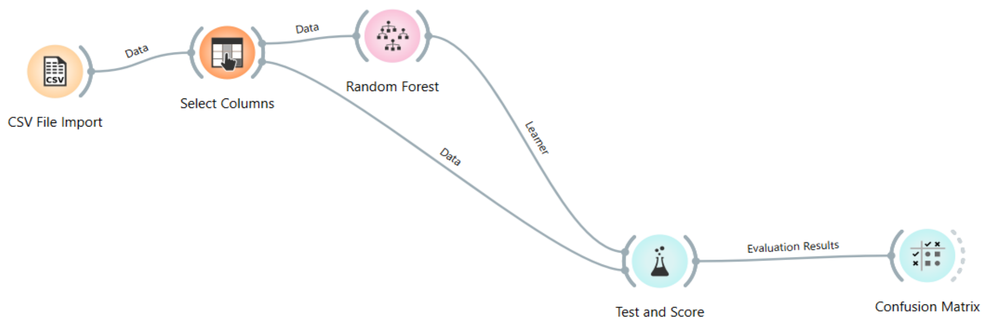
- From the Evaluate category, find the Confusion Matrix widget and drag it onto the canvas.
- Connect the Test & Score widget to the Confusion Matrix widget.

Step 7: Run the Workflow
Execute the Workflow:

- Click the Run button (the small play icon) in the top right corner of the Orange interface to execute the workflow.

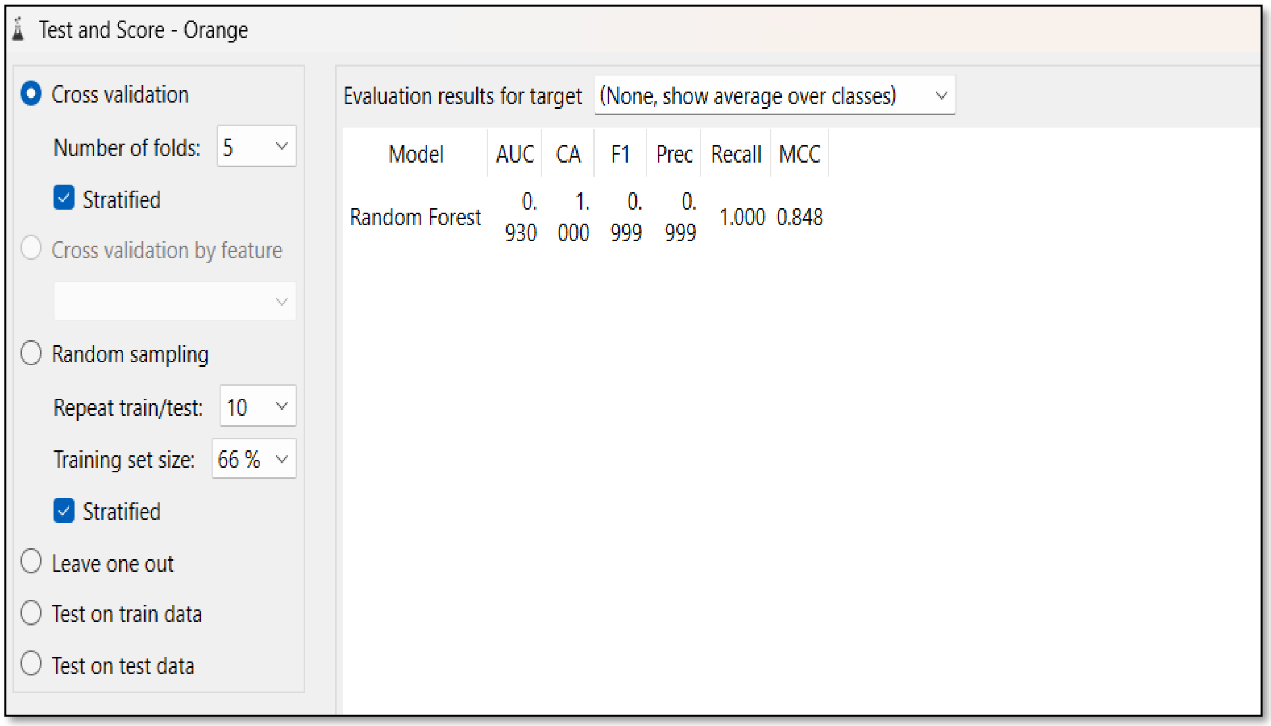
View Results:

- After execution, you can click on the Test & Score widget to see the evaluation metrics (accuracy, precision, recall, etc.).
- Double-click the Confusion Matrix widget to view the confusion matrix, which shows the model's predictions compared to the actual labels.



Workflow in Orange Tool

Output: Test and Score Widget



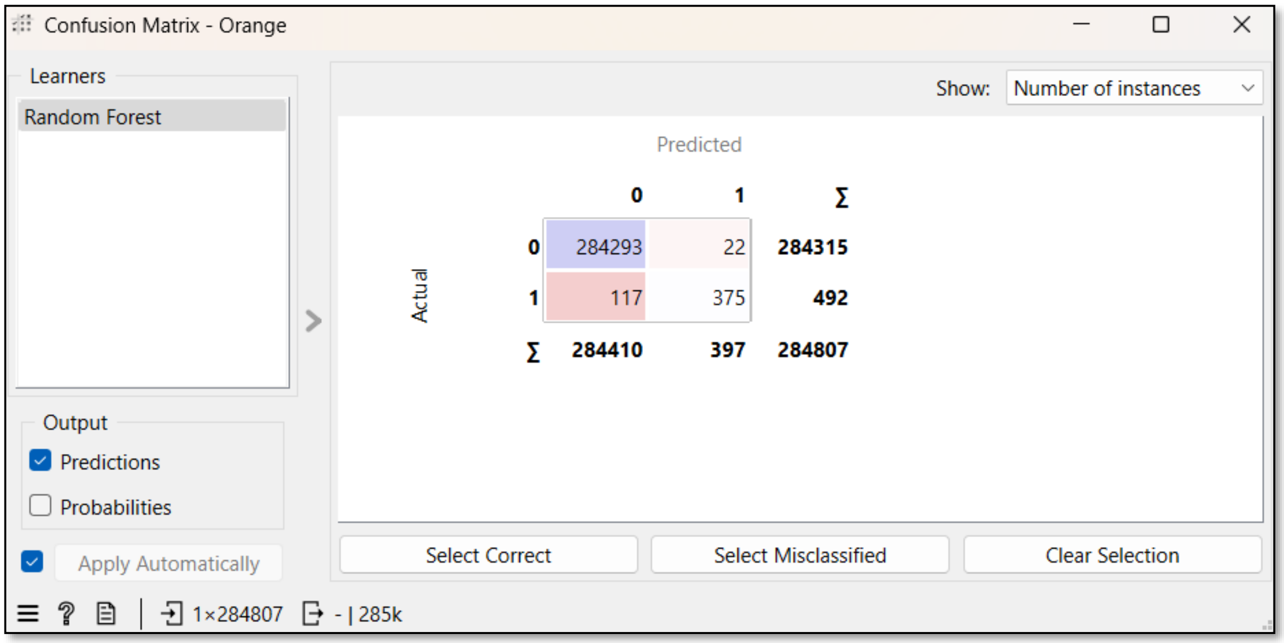
Test and Score - Orange

- ● Cross validation
  - Number of folds: 5
  - [✓] Stratified
- ○ Cross validation by feature
- ○ Random sampling
  - Repeat train/test: 10
  - Training set size: 66 %
  - [✓] Stratified
- ○ Leave one out
- ○ Test on train data
- ○ Test on test data

Evaluation results for target (None, show average over classes)

Model	AUC	CA	F1	Prec	Recall	MCC
Random Forest	0.930	1.000	0.999	0.999	1.000	0.848

Confusion Matrix



Confusion Matrix - Orange

Learners: Random Forest

Show: Number of instances

Actual \ Predicted	0	1	Σ
0	284293	22	284315
1	117	375	492
Σ	284410	397	284807

Output
- [✓] Predictions
- [ ] Probabilities

[✓] Apply Automatically

Result:

Exp No: 11 **Date:**

EXPLORE BAGGING AND BOOSTING ENSEMBLE CLASSIFIERS FOR CLASSIFICATIONS

Aim: To Explore Bagging and Boosting Ensemble Classifiers for Classifications using Python and Orange Tool

Algorithm:

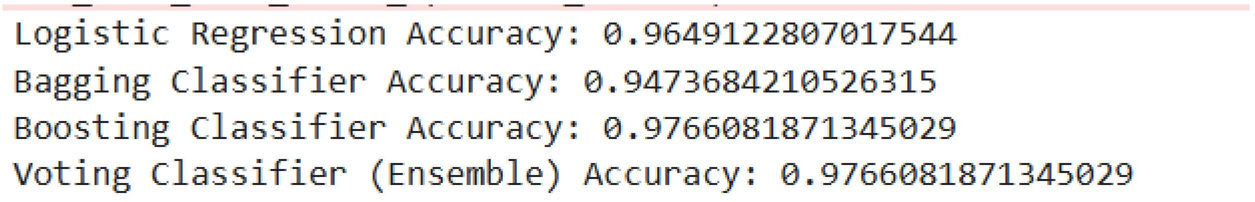
1. Load Dataset: Load the Breast Cancer dataset and split it into X (features) and y (target).
2. Train-Test Split: Divide data into training and testing sets.
3. Define and Train Classifiers:
4. Logistic Regression: Train on the dataset and predict the test set.
5. Bagging (BaggingClassifier with Decision Tree): Train using bagging, with 10 trees, and predict the test set.
6. Boosting (AdaBoost with Decision Tree): Train using AdaBoost, with 50 estimators, and predict the test set.
7. Voting (VotingClassifier): Combine Logistic Regression, Decision Tree, and K-Nearest Neighbors with majority voting; predict the test set.
8. Evaluate Models: Calculate and print the accuracy for each model.

Python Program

```
 # Import necessary libraries
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier, VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split data 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)

# 1. Simple Supervised Classifier (Logistic Regression)
log_reg_clf = LogisticRegression(max_iter=1000, random_state=42)
log_reg_clf.fit(X_train, y_train)
log_reg_pred = log_reg_clf.predict(X_test)
log_reg_accuracy = accuracy_score(y_test, log_reg_pred)
```

```
# 2. Bagging Classifier
bagging_clf = BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=10, random_state=42)
bagging_clf.fit(X_train, y_train)

bagging_pred = bagging_clf.predict(X_test)
bagging_accuracy = accuracy_score(y_test, bagging_pred)

# 3. Boosting Classifier (AdaBoost)
boosting_clf = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1), n_estimators=50, random_state=42)
boosting_clf.fit(X_train, y_train)
boosting_pred = boosting_clf.predict(X_test)
boosting_accuracy = accuracy_score(y_test, boosting_pred)

# 4. Ensemble Learner (Voting Classifier)
# Using Logistic Regression, Decision Tree, and K-Nearest Neighbors as base models
knn_clf = KNeighborsClassifier()
voting_clf = VotingClassifier(
    estimators=[
        ('lr', log_reg_clf),
        ('dt', DecisionTreeClassifier()),
        ('knn', knn_clf)
    ],
    voting='hard'  # 'hard' for majority voting; 'soft' would use probabilities
)
voting_clf.fit(X_train, y_train)
voting_pred = voting_clf.predict(X_test)
voting_accuracy = accuracy_score(y_test, voting_pred)

# Print the accuracies
print("Logistic Regression Accuracy:", log_reg_accuracy)
print("Bagging Classifier Accuracy:", bagging_accuracy)
print("Boosting Classifier Accuracy:", boosting_accuracy)
print("Voting Classifier (Ensemble) Accuracy:", voting_accuracy)
```

Output:

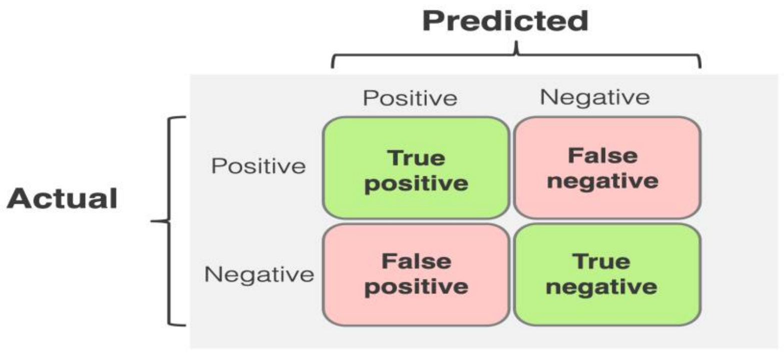


```
Logistic Regression Accuracy: 0.9649122807017544
Bagging Classifier Accuracy: 0.9473684210526315
Boosting Classifier Accuracy: 0.9766081871345029
Voting Classifier (Ensemble) Accuracy: 0.9766081871345029
```

Performance Metrics used:

1. **Confusion Matrix**

The Confusion Matrix is a table used to evaluate the performance of a classification model, especially in binary classification tasks. It summarizes the outcomes of predictions and compares them to the actual labels. The matrix provides a more detailed look at what errors a model is making, breaking down predictions into four key categories:



Components of the Confusion Matrix

1. True Positives (TP): Correctly predicted positive cases (e.g., the model predicted "positive," and the actual label was also "positive").
2. False Positives (FP): Incorrectly predicted positive cases (e.g., the model predicted "positive," but the actual label was "negative"). Also called a Type I Error.
3. True Negatives (TN): Correctly predicted negative cases (e.g., the model predicted "negative," and the actual label was also "negative").
4. False Negatives (FN): Incorrectly predicted negative cases (e.g., the model predicted "negative," but the actual label was "positive"). Also called a Type II Error.

Example

For instance, if a confusion matrix is given as:

	Predicted Positive	Predicted Negative
Actual Positive	50	10
Actual Negative	5	100

Here:

- **TP = 50**: Correct positive predictions.
- **FN = 10**: Positive cases incorrectly predicted as negative.
- **FP = 5**: Negative cases incorrectly predicted as positive.
- **TN = 100**: Correct negative predictions.

2. **Area Under Curve (AUC)**

Definition: AUC, or Area Under the Curve, is a performance measurement for classification models at various threshold settings. It represents the area under the ROC (Receiver Operating Characteristic) curve, which plots the true positive rate (sensitivity) against the false positive rate (1-specificity).

Interpretation:

AUC = 1.0: Perfect classifier.
AUC = 0.5: Model performs no better than random chance.
Usage: AUC is useful for comparing classifiers, especially in imbalanced datasets, as it reflects the model's ability to differentiate between positive and negative classes across different thresholds.

3. Classification Accuracy (CA)

Definition: Classification accuracy is the ratio of correctly predicted observations to the total observations.

$$\text{Classification Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100$$

Higher accuracy indicates better model performance.
However, it can be misleading for imbalanced datasets, as it doesn't differentiate between types of errors (false positives and false negatives).

4. Precision

Definition: Precision, also known as the positive predictive value, is the ratio of correctly predicted positive observations to the total predicted positive observations.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

High precision indicates that the model has a low false positive rate, meaning that when it predicts a positive, it's likely correct.
Usage: Precision is especially useful in situations where false positives are more costly than false negatives (e.g., in spam detection).

5. Recall

Definition: Recall, also known as sensitivity or true positive rate, is the ratio of correctly predicted positive observations to all observations in the actual class.

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

High recall indicates that the model can capture most of the actual positives.
Usage: Recall is crucial when false negatives are costly (e.g., in disease diagnosis, where missing a positive case can have severe consequences).

6. Matthews Correlation Coefficient (MCC)

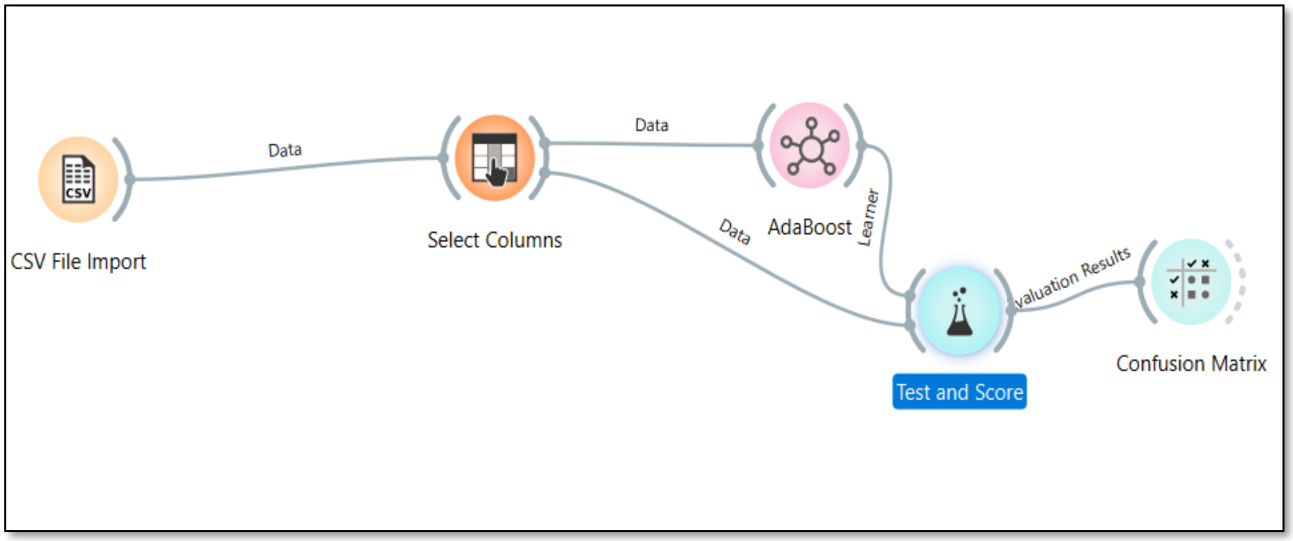
Definition: The Matthews Correlation Coefficient (MCC) is a metric that measures the quality of binary classifications, particularly useful for imbalanced datasets. It considers true and false positives and negatives to provide a comprehensive evaluation.

$$\mathrm{MCC} = \frac{(\mathrm{TP} \cdot \mathrm{TN}) - (\mathrm{FP} \cdot \mathrm{FN})}{\sqrt{(\mathrm{TP}+\mathrm{FP}) \cdot (\mathrm{TP}+\mathrm{FN}) \cdot (\mathrm{TN}+\mathrm{FP}) \cdot (\mathrm{TN}+\mathrm{FN})}}$$

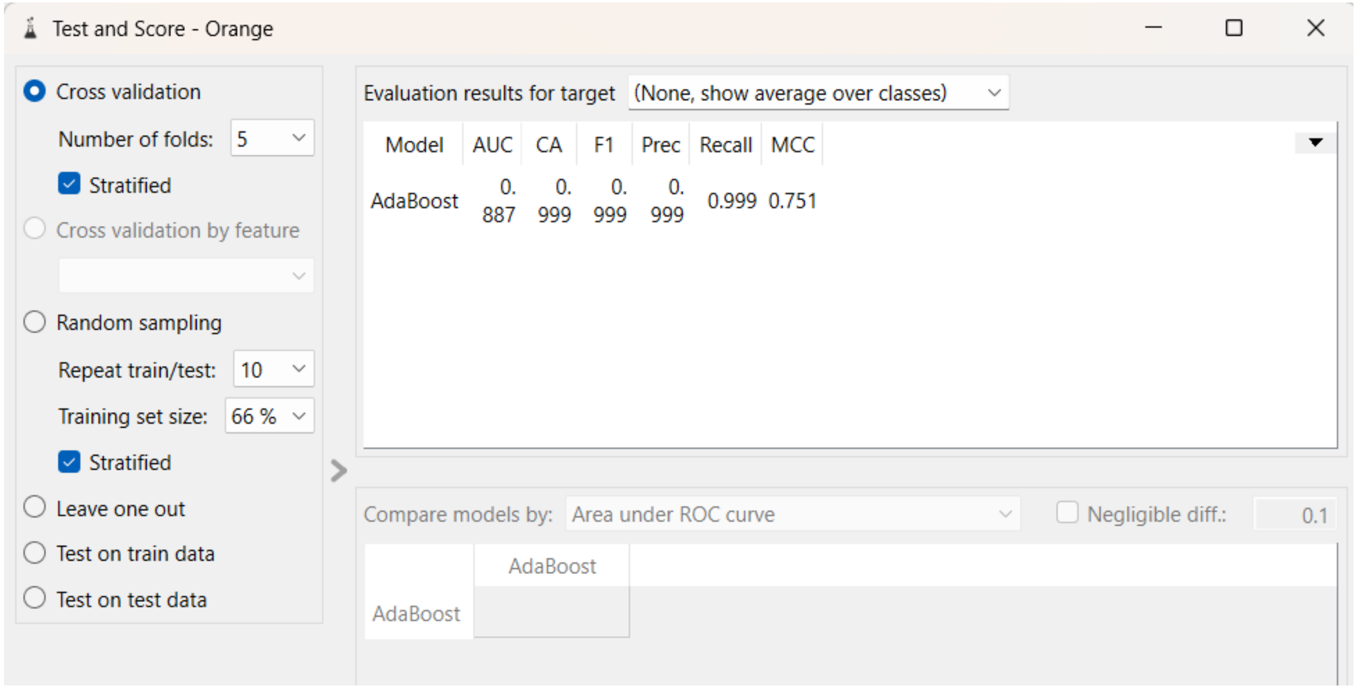
MCC = +1: Perfect prediction.
MCC = 0: Model performs no better than random.
MCC = -1: Total disagreement between prediction and ground truth.
Usage: MCC is a balanced measure even when classes are of very different sizes, providing a more reliable evaluation than accuracy in such cases.

Using Orange Tool

Boosting: Adaboost Implementation in Orange



Workflow in Orange:Adaboost



Cross validation [✓]
Number of folds: 5
Stratified ✓
Cross validation by feature
Random sampling
Repeat train/test: 10
Training set size: 66 %
Stratified ✓
Leave one out
Test on train data
Test on test data

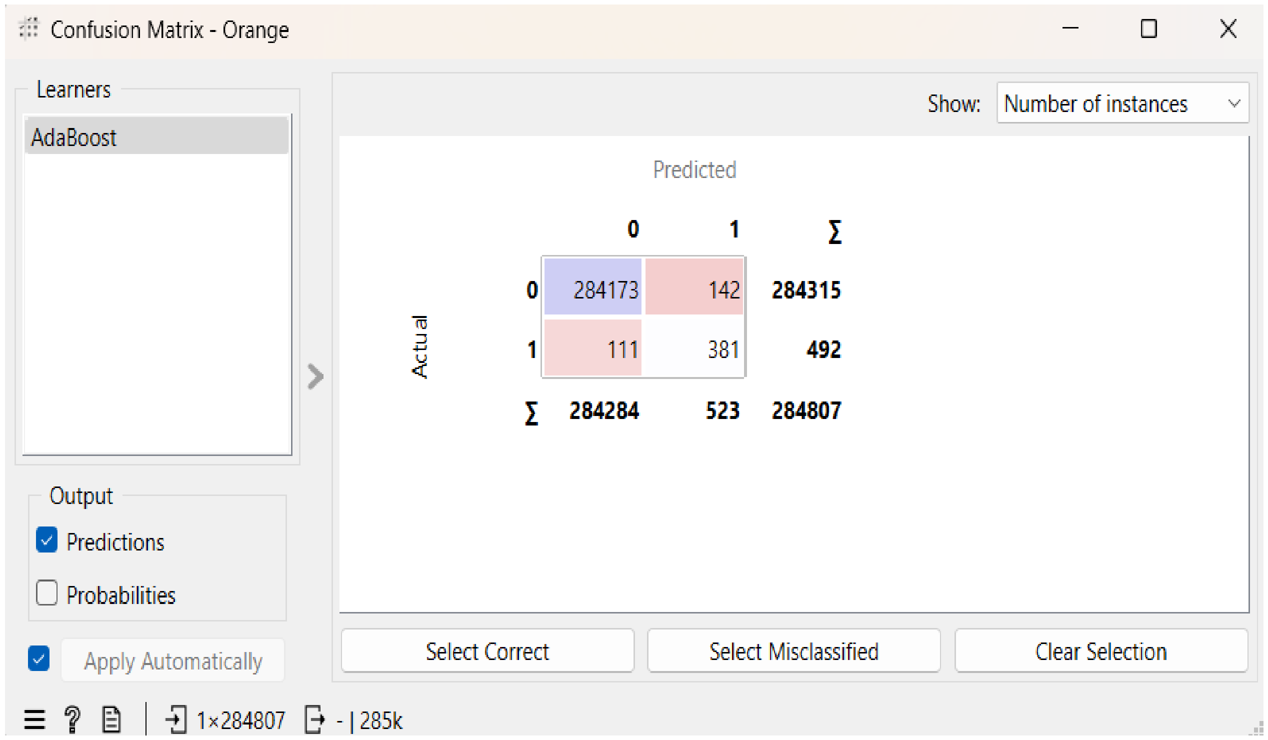
Evaluation results for target (None, show average over classes)

Model	AUC	CA	F1	Prec	Recall	MCC
AdaBoost	0.887	0.999	0.999	0.999	0.999	0.751

Compare models by: Area under ROC curve   Negligible diff.: 0.1

	AdaBoost
AdaBoost	

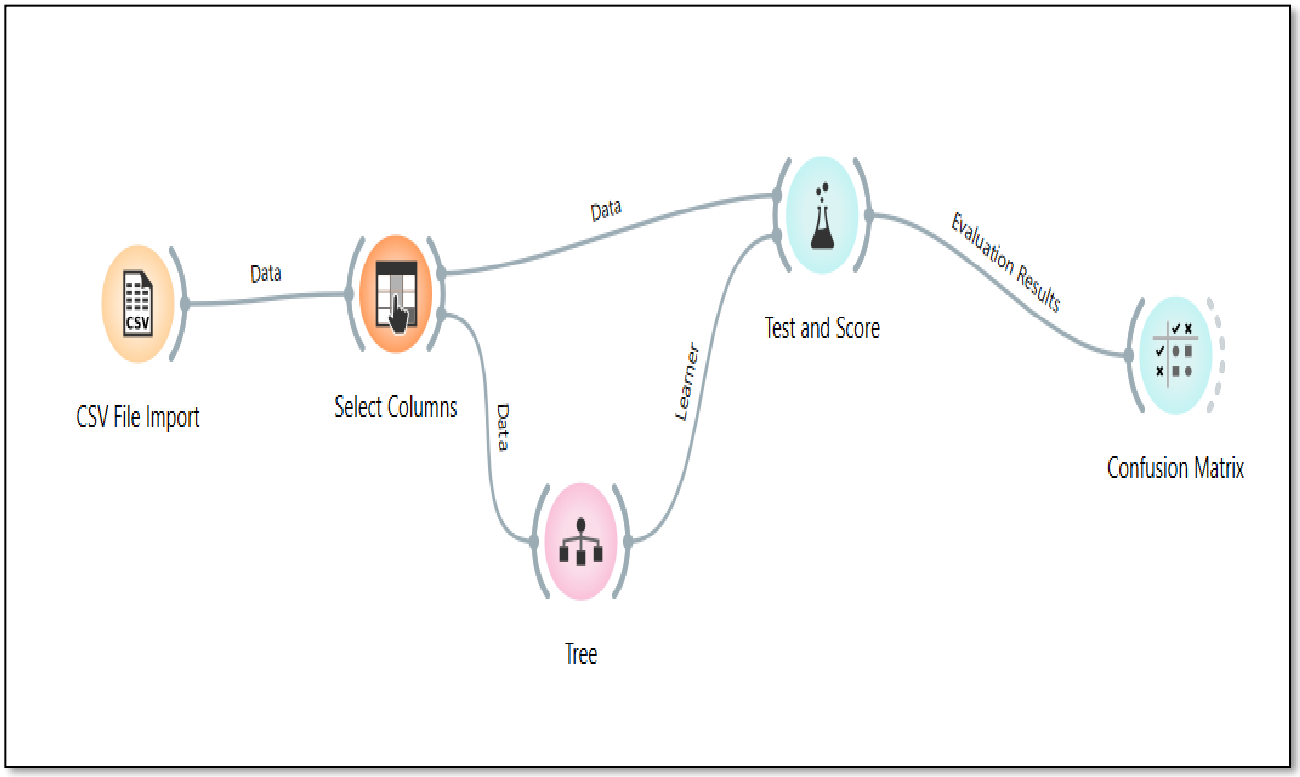
Output: Area under Curve(AUC),Classification Accuracy(CA),Precision, Recall, MCC(Matthews Correlation Coefficient)



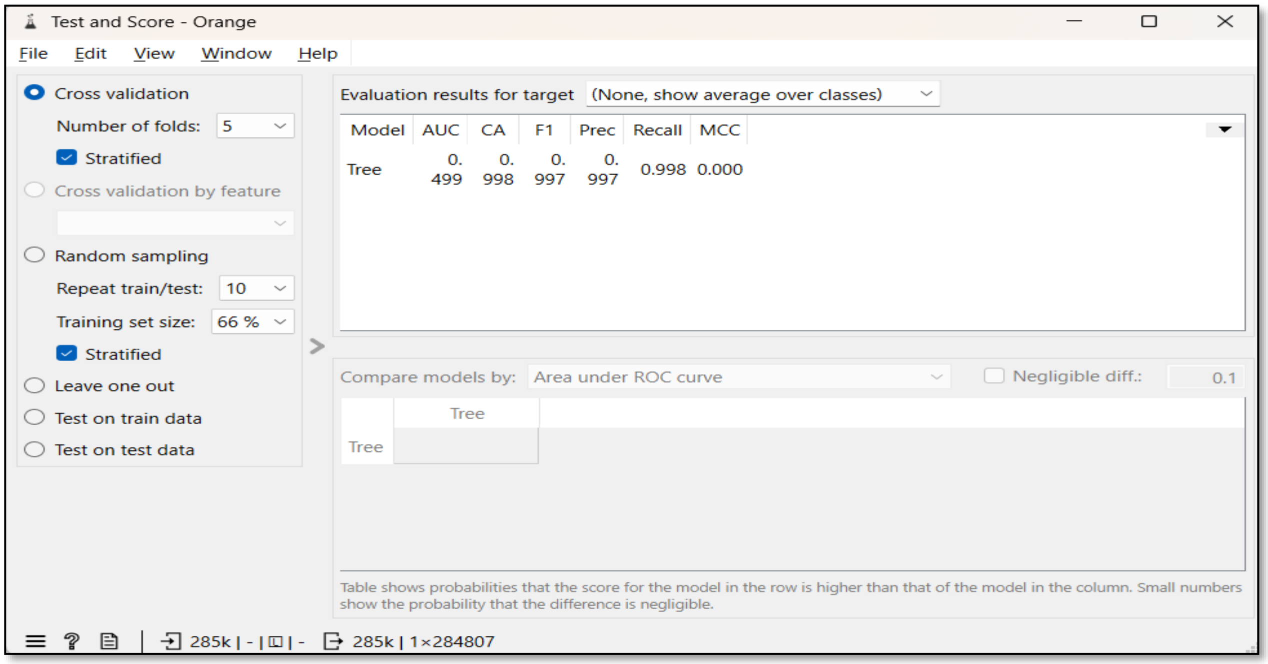
		Predicted		
		0	1	∑
Actual	0	284173	142	284315
	1	111	381	492
	∑	284284	523	284807

Output: Confusion Matrix-Adaboost

<u>Bagging: Decision Tree Implementation in Orange</u>



Workflow in Orange: Decision Tree



Cross validation [✓]
Number of folds: 5
Stratified ✓
Cross validation by feature
Random sampling
Repeat train/test: 10
Training set size: 66 %
Stratified ✓
Leave one out
Test on train data
Test on test data

Evaluation results for target (None, show average over classes)

Model	AUC	CA	F1	Prec	Recall	MCC
Tree	0.499	0.998	0.997	0.997	0.998	0.000

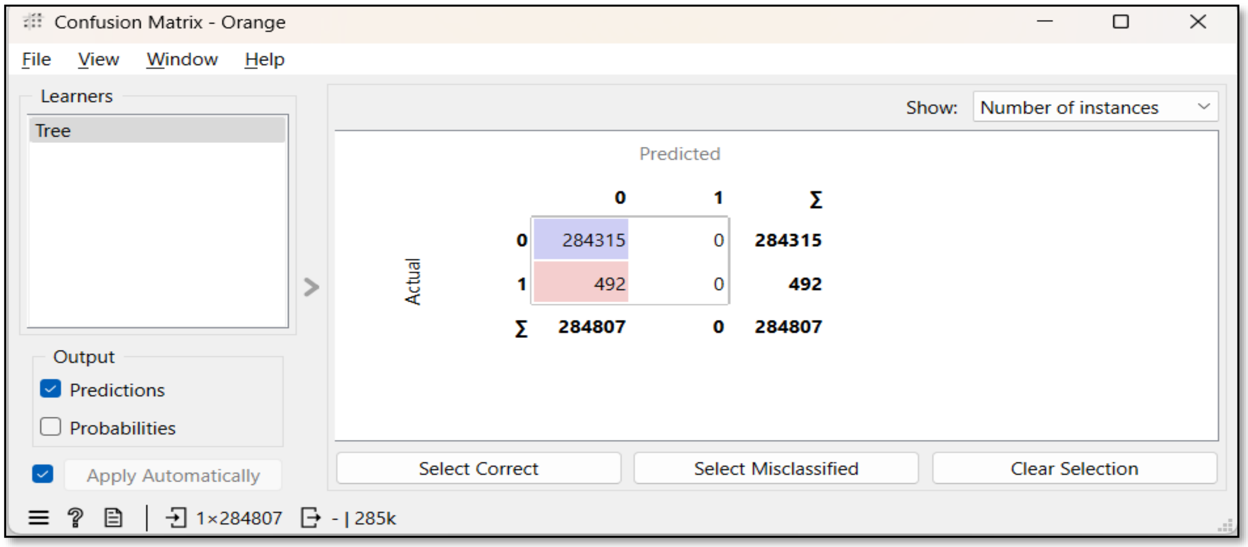
Compare models by: Area under ROC curve   Negligible diff.: 0.1

	Tree
Tree	

Table shows probabilities that the score for the model in the row is higher than that of the model in the column. Small numbers show the probability that the difference is negligible.

285k | - | ◫ | -   285k | 1×284807

Output: Area under Curve(AUC),Classification Accuracy(CA),Precision, Recall, MCC(Matthews Correlation Coefficient)



Learners
Tree

Show: Number of instances

Predicted

Actual	0	1	Σ
0	284315	0	284315
1	492	0	492
Σ	284807	0	284807

Output
Predictions ✓
Probabilities
Apply Automatically ✓

1×284807   - | 285k

Output: Confusion Matrix-Bagging-Decision Tree

Result:

Exp No: 12 **Date:**

CREATE A FAKE NEWS TRACKER PROGRAM TO COLLECT, DETECT AND HELP VISUALIZE FAKE NEWS DATA

Aim: To create a fake news tracker program to collect, detect and help visualize fake news data

Algorithm:

1. Load the Dataset: Reads the CSV file and loads it into a Pandas DataFrame.
2. Data Preprocessing:
   - Drops any rows with missing values.
   - Converts text to lowercase to standardize the input.
3. Split the Dataset: Splits data into training and test sets.
4. Text Vectorization: Uses TfidfVectorizer to convert the text data into numerical features.
5. Train the Model: Trains a logistic regression model on the TF-IDF features.
6. Make Predictions: Predicts labels on the test data.
7. Evaluate the Model: Calculates accuracy, confusion matrix, and classification report.

Program

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Step 1: Load the dataset
data = pd.read_csv('C:/Users/dhari/OneDrive/Desktop/AI IN CS/WELFake_Dataset.csv')

# Step 2: Preprocess the data
# Drop any null rows
data = data.dropna()

# Converting text to lowercase for consistency
data['text'] = data['text'].str.lower()

# Step 3: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data['text'], data['label'], test_size=0.2, random_state=42)

# Step 4: Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer(max_df=0.7, stop_words='english')
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Step 5: Train a logistic regression model
model = LogisticRegression()
```

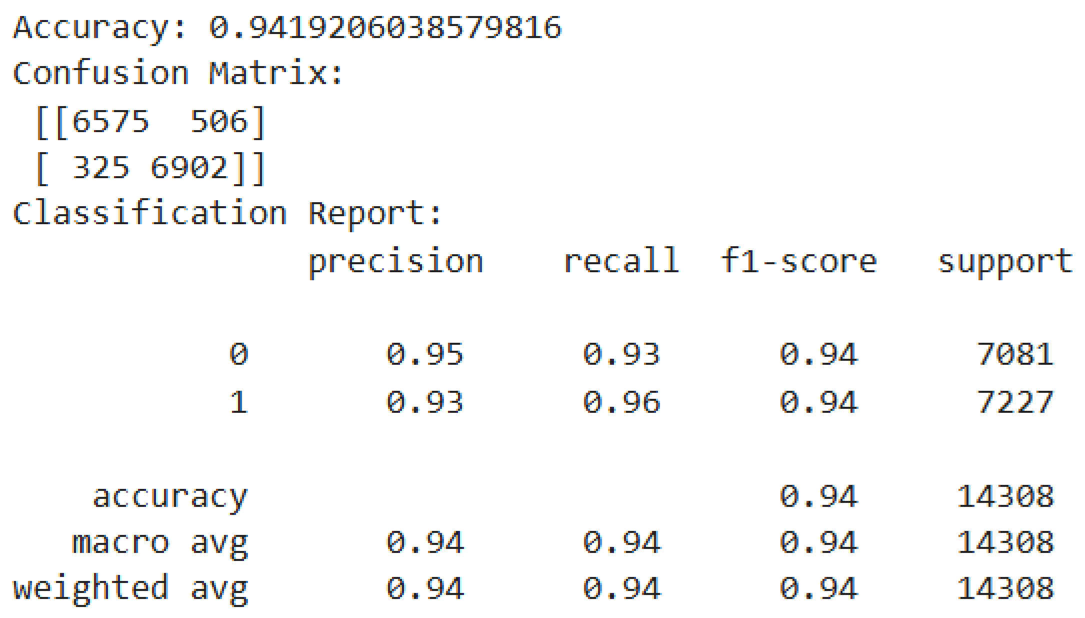
model.fit(X_train_tfidf, y_train)

# Step 6: Make predictions on the test set
y_pred = model.predict(X_test_tfidf)

# Step 7: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)

Output:



```
Accuracy: 0.9419206038579816
Confusion Matrix:
 [[6575  506]
 [ 325 6902]]
Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.93      0.94      7081
           1       0.93      0.96      0.94      7227

    accuracy                           0.94     14308
   macro avg       0.94      0.94      0.94     14308
weighted avg       0.94      0.94      0.94     14308
```

Result: