Practical No. 1

CODE:BREADTH FIRST SEARCH

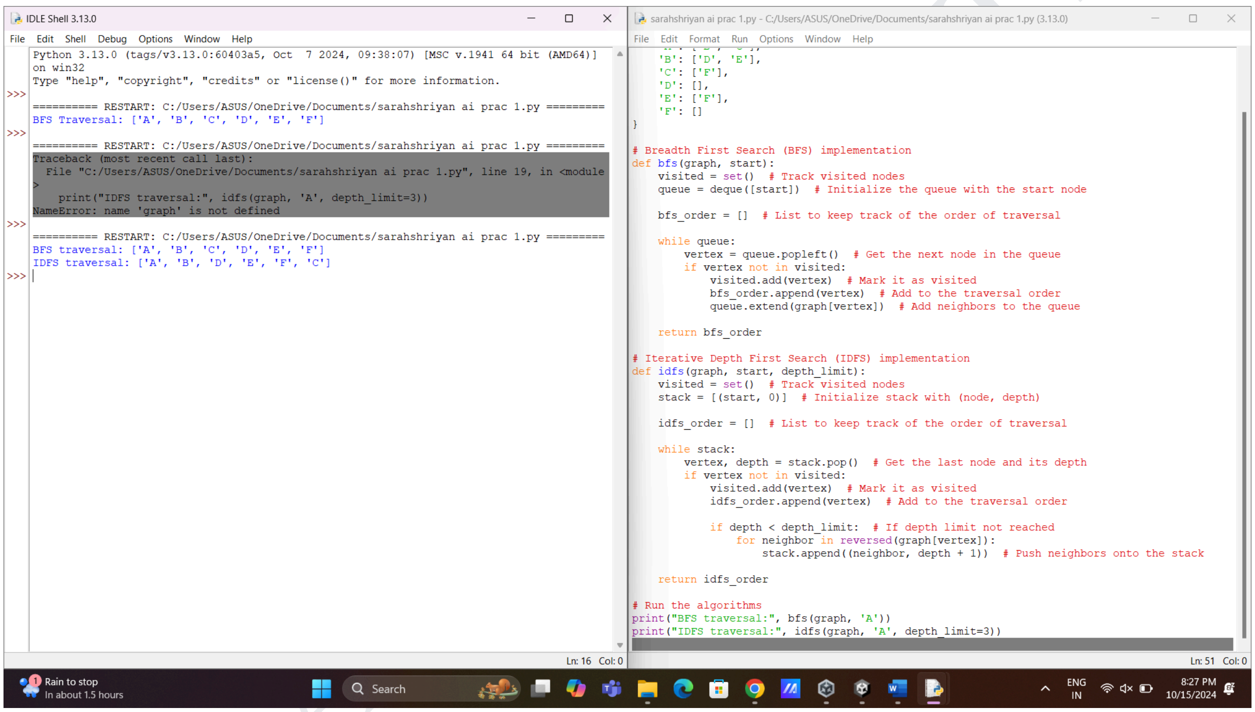
```
from collections import deque

def bfs(graph, start_node):
    visited = set()
    queue = deque([start_node])
    bfs_order = []

    while queue:
        node = queue.popleft()
        if node not in visited:
            visited.add(node)
            bfs_order.append(node)
            queue.extend(graph[node])

    return bfs_order

# Example graph (Adjacency List)
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': [],
    'F': []
}

print("BFS Traversal:", bfs(graph, 'A'))
```

CODE:**Iterative Depth First Search (IDFS) Code**

```
def idfs(graph, start_node, depth_limit):
    def dfs_limited(node, depth):
        if depth == 0:
            return
        visited.add(node)
        dfs_order.append(node)
        for neighbor in graph[node]:
            if neighbor not in visited:
                dfs_limited(neighbor, depth - 1)
```

```
    dfs_order = []
    for depth in range(1, depth_limit + 1):
        visited = set()
        dfs_limited(start_node, depth)

    return dfs_order

# Using the same graph
print("IDFS traversal:", idfs(graph, 'A', depth_limit=3))
```

OUTPUT;



Practical No. 2

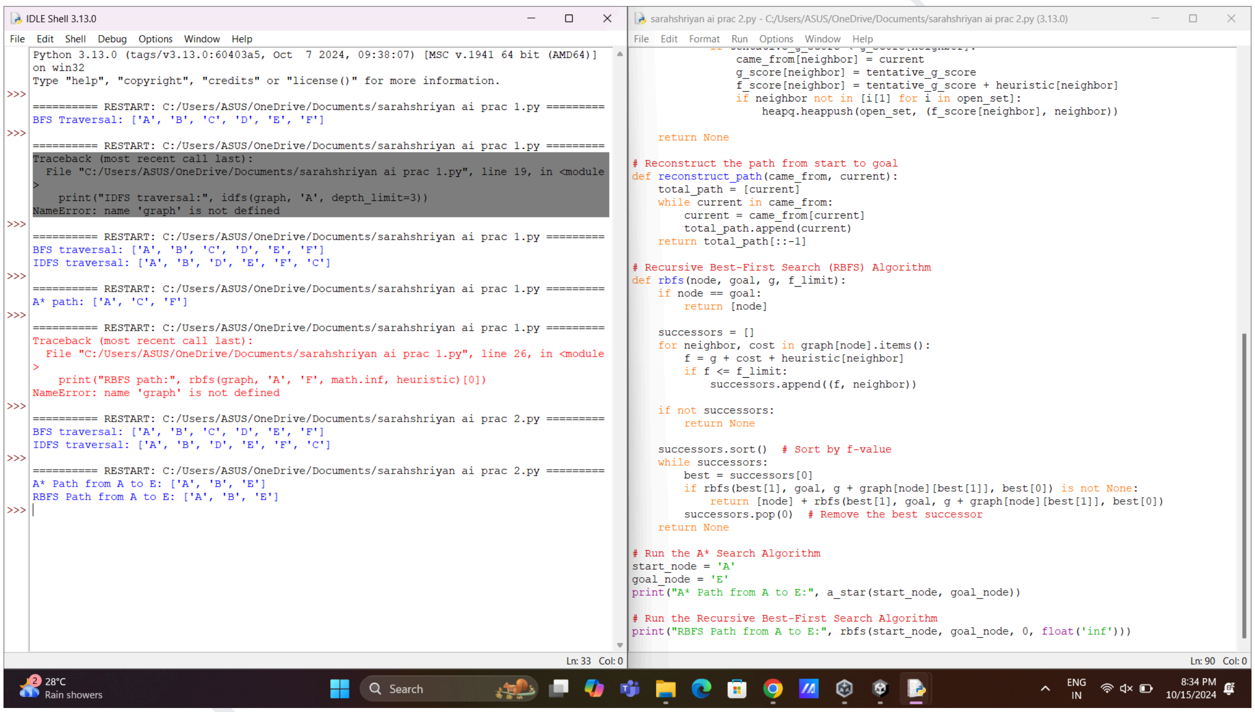
CODE:

```
import heapq

# Define the graph as a dictionary with costs and heuristic values
graph = {
    'A': {'B': 1, 'C': 4},
    'B': {'A': 1, 'D': 2, 'E': 5},
    'C': {'A': 4, 'F': 3},
    'D': {'B': 2, 'G': 3},
    'E': {'B': 5, 'G': 2},
    'F': {'C': 3, 'G': 1},
    'G': {'D': 3, 'E': 2, 'F': 1}
}

# Heuristic values (straight-line distances to the goal)
heuristic = {
    'A': 7,
    'B': 6,
    'C': 2,
    'D': 1,
    'E': 0,
    'F': 1,
    'G': 0
}

# A* Search Algorithm
def a_star(start, goal):
    open_set = []
    heapq.heappush(open_set, (0 + heuristic[start], start))  # (f(n), node)
    came_from = {}
    g_score = {node: float('inf') for node in graph}
    g_score[start] = 0
    f_score = {node: float('inf') for node in graph}
    f_score[start] = heuristic[start]

    while open_set:
        current = heapq.heappop(open_set)[1]
```

```
        if current == goal:
            return reconstruct_path(came_from, current)

        for neighbor, cost in graph[current].items():
            tentative_g_score = g_score[current] + cost

            if tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = tentative_g_score + heuristic[neighbor]
                if neighbor not in [i[1] for i in open_set]:
                    heapq.heappush(open_set, (f_score[neighbor], neighbor))

    return None

# Reconstruct the path from start to goal
def reconstruct_path(came_from, current):
    total_path = [current]
    while current in came_from:
        current = came_from[current]
        total_path.append(current)
    return total_path[::-1]

# Recursive Best-First Search (RBFS) Algorithm
def rbfs(node, goal, g, f_limit):
    if node == goal:
        return [node]

    successors = []
    for neighbor, cost in graph[node].items():
        f = g + cost + heuristic[neighbor]
        if f <= f_limit:
            successors.append((f, neighbor))

    if not successors:
        return None

    successors.sort()  # Sort by f-value
    while successors:
        best = successors[0]
        if rbfs(best[1], goal, g + graph[node][best[1]], best[0]) is not None:
            return [node] + rbfs(best[1], goal, g + graph[node][best[1]], best[0])
```

```
        successors.pop(0)  # Remove the best successor
    return None

# Run the A* Search Algorithm
start_node = 'A'
goal_node = 'E'
print("A* Path from A to E:", a_star(start_node, goal_node))

# Run the Recursive Best-First Search Algorithm
print("RBFS Path from A to E:", rbfs(start_node, goal_node, 0, float('inf')))
```

OUTPUT:



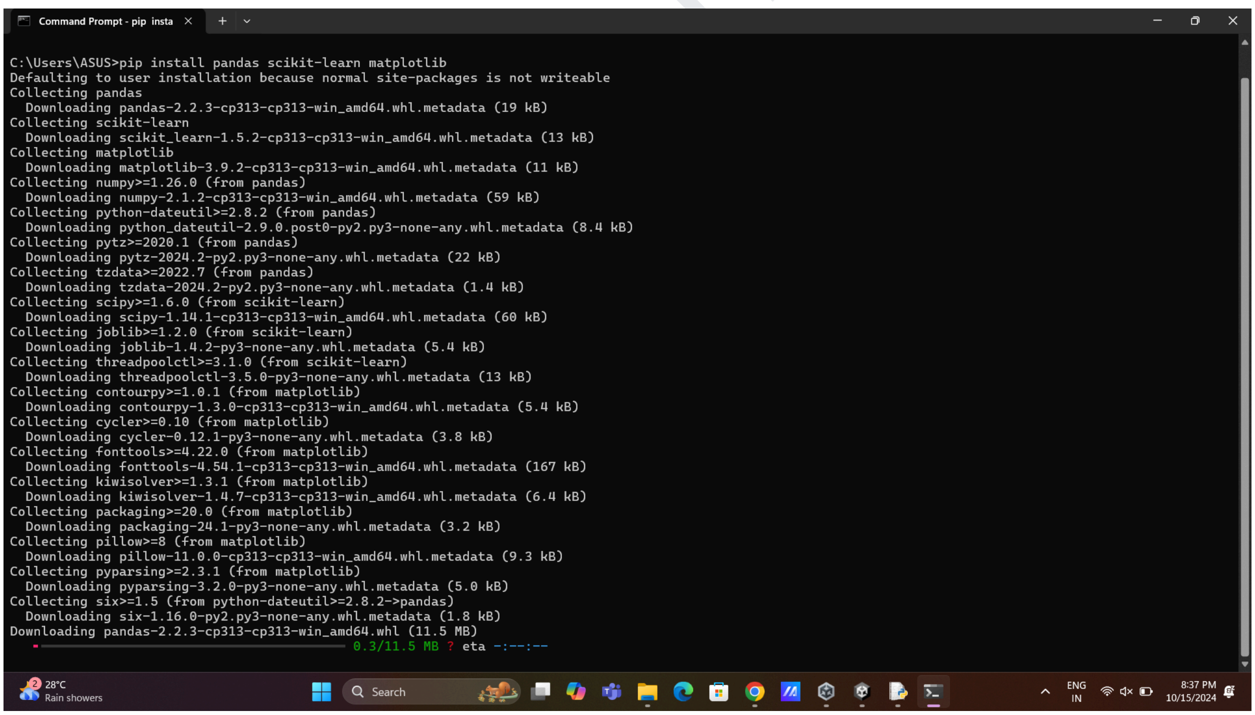
Practical No. 3

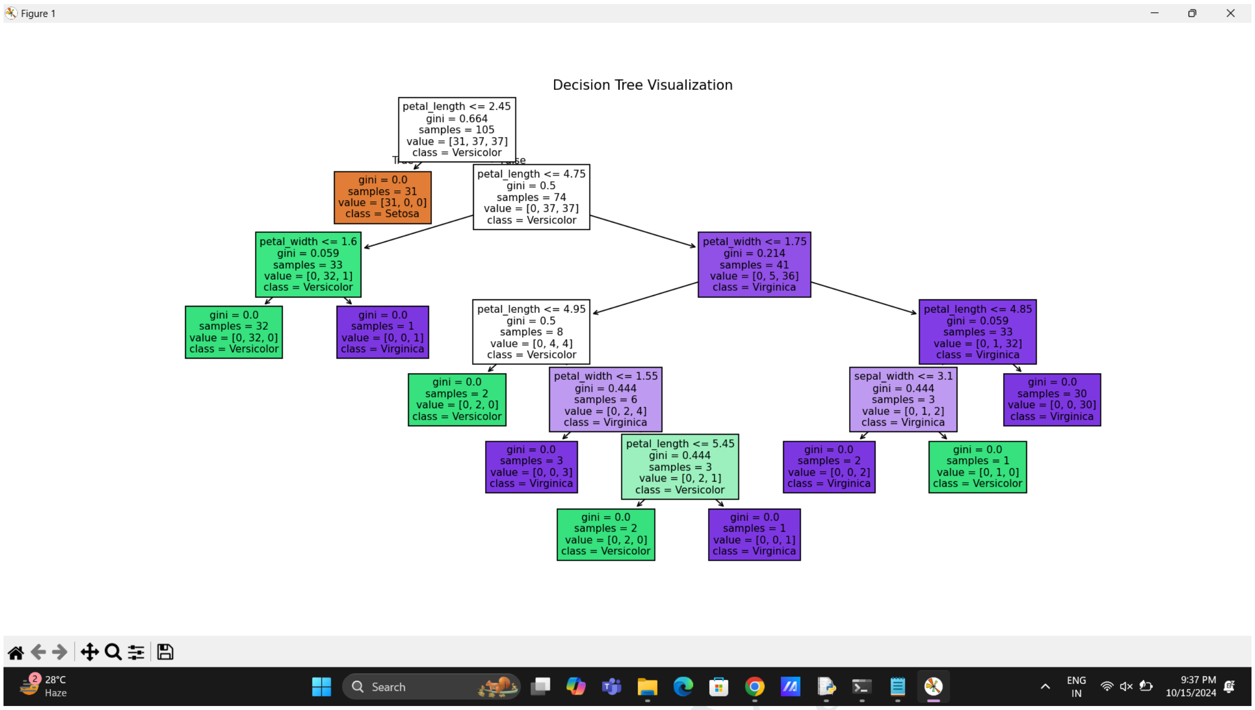
CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

# Load dataset (using the Iris dataset as an example)
# You can replace this with your own dataset or download it from a CSV file
try:
    url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
    column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
    data = pd.read_csv(url, header=None, names=column_names)

    # Check if the data is loaded correctly
    print("Data Loaded Successfully")
    print(data.head())  # Display the first few rows of the dataset

except Exception as e:
    print("Error loading the dataset:", e)

# Preprocess the data
# Map species to numerical values
data['species'] = data['species'].map({'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2})

# Split the dataset into features and target
X = data.drop('species', axis=1)  # Features
y = data['species']  # Target variable

# 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)

# Initialize and train the Decision Tree model
try:
    dtree = DecisionTreeClassifier(random_state=42)
    dtree.fit(X_train, y_train)

    # Make predictions
    y_pred = dtree.predict(X_test)
```

```
    # Evaluate the accuracy of the model
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy of the Decision Tree model: {accuracy:.2f}")

except Exception as e:
    print("Error during model training or prediction:", e)

# Visualize the Decision Tree
try:
    plt.figure(figsize=(12, 8))
    plot_tree(dtree, filled=True, feature_names=X.columns, class_names=['Setosa', 'Versicolor',
'Virginica'])
    plt.title("Decision Tree Visualization")
    plt.show()

except Exception as e:
    print("Error during tree visualization:", e)
```

OUTPUT






```
>>>
========== RESTART: C:/Users/ASUS/OneDrive/Documents/sarahshriyan ai prac 3.py =========
Data Loaded Successfully
   sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
Accuracy of the Decision Tree model: 1.00
```

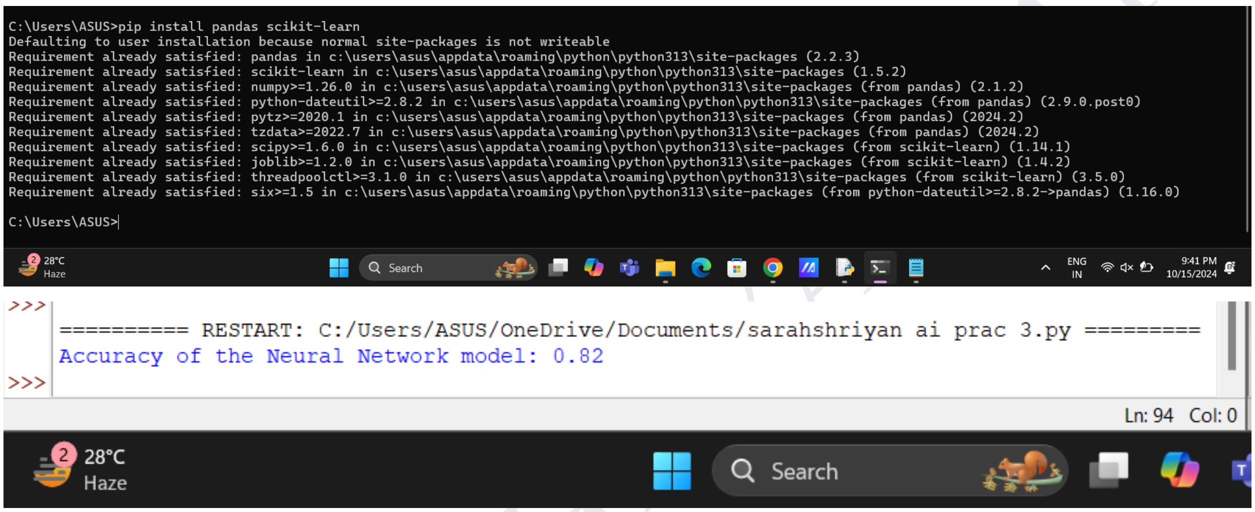
Practical No. 4

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Load dataset (using the Iris dataset as an example)
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
data = pd.read_csv(url, header=None, names=column_names)

# Preprocess the data
# Map species to numerical values
data['species'] = data['species'].map({'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2})

# Split the dataset into features and target
X = data.drop('species', axis=1)  # Features
y = data['species']  # Target variable

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# Initialize and train the Feed Forward Backpropagation Neural Network
mlp = MLPClassifier(hidden_layer_sizes=(5, 5),  # Two hidden layers with 5 neurons each
                    max_iter=2000,  # Increased maximum iterations
                    random_state=42,
                    solver='adam',  # Default solver
                    learning_rate_init=0.01,  # Set the learning rate
                    early_stopping=True,  # Enable early stopping
                    n_iter_no_change=10)  # Stop if no improvement for 10 iterations

# Fit the model on the training data
mlp.fit(X_train, y_train)

# Make predictions
```

```
y_pred = mlp.predict(X_test)

# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Neural Network model: {accuracy:.2f}")
```

OUTPUT:



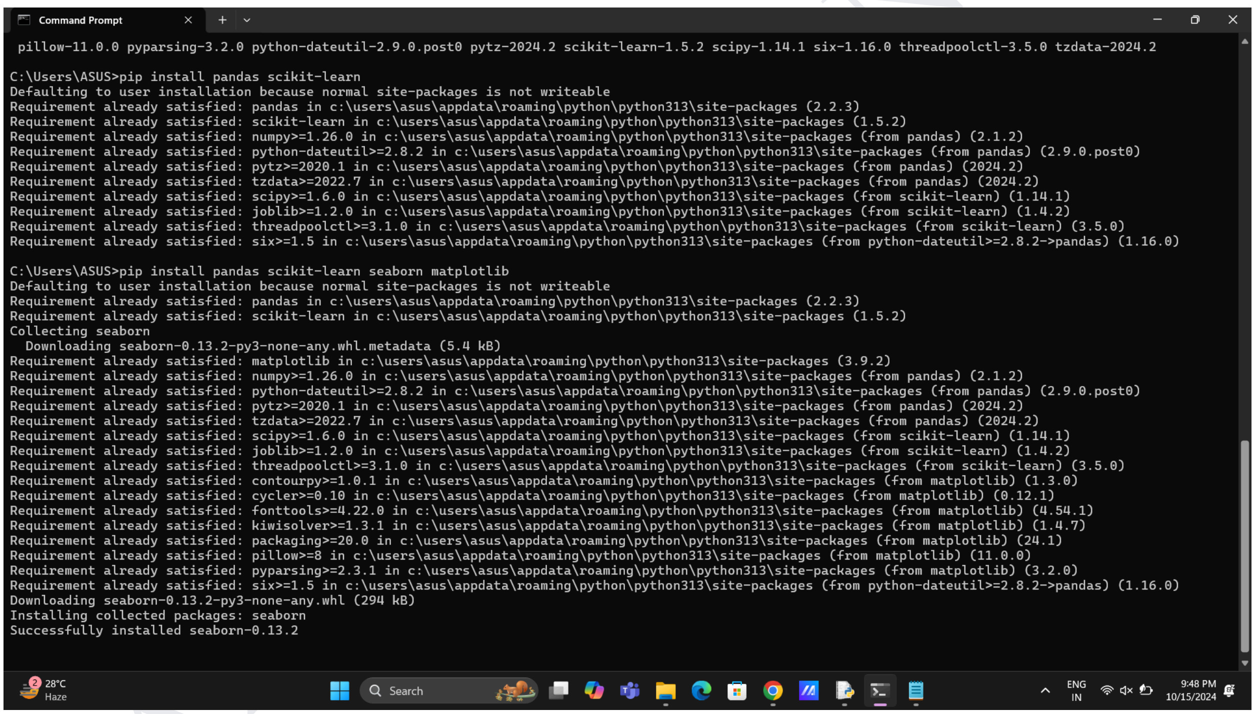
Practical No. 5

CODE:

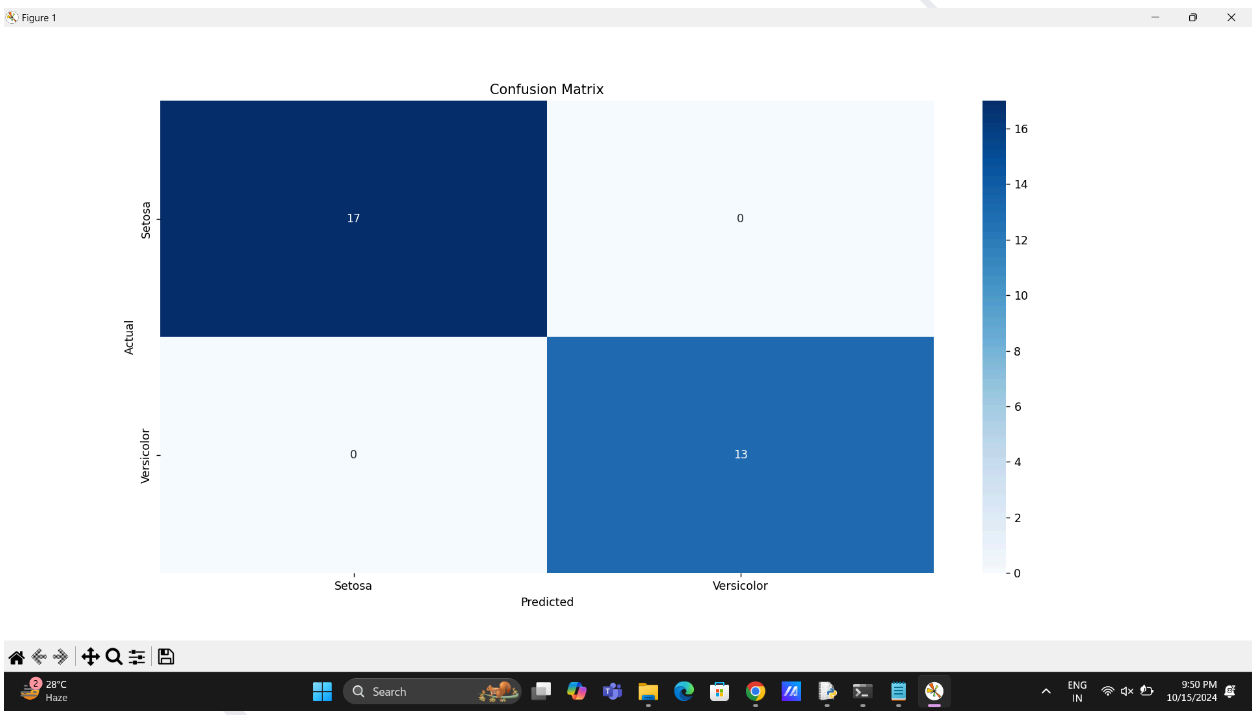
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset (using the Iris dataset as an example)
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
data = pd.read_csv(url, header=None, names=column_names)

# Preprocess the data
# Map species to numerical values (Iris-setosa: 0, Iris-versicolor: 1, Iris-virginica: 2)
data['species'] = data['species'].map({'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2})

# For binary classification, let's select only two classes (setosa and versicolor)
data_binary = data[data['species'] < 2]

# Split the dataset into features and target
X = data_binary.drop('species', axis=1)  # Features
y = data_binary['species']  # Target variable

# 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)

# Initialize and train the SVM model
svm_model = SVC(kernel='linear', random_state=42)  # You can change the kernel to 'rbf', 'poly', etc.
svm_model.fit(X_train, y_train)

# Make predictions
y_pred = svm_model.predict(X_test)

# Evaluate the performance of the SVM model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the SVM model: {accuracy:.2f}")

# Print classification report and confusion matrix
```

```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix visualization
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Setosa', 'Versicolor'],
yticklabels=['Setosa', 'Versicolor'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

OUTPUT:



```
========== RESTART: C:/Users/ASUS/OneDrive/Documents/sarahshriyan ai prac 5.py =========
Accuracy of the SVM model: 1.00

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       1.00      1.00      1.00        13

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```



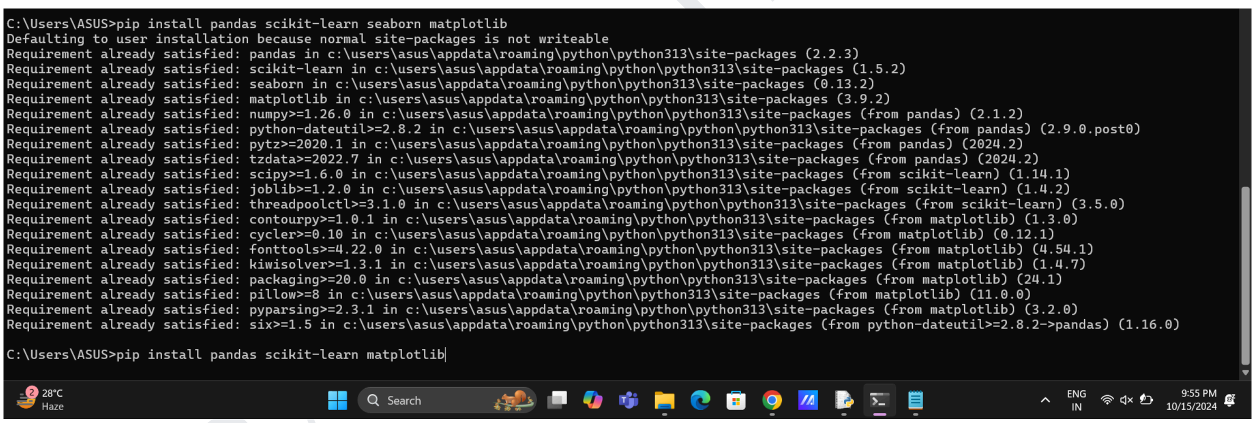
Practical No. 6

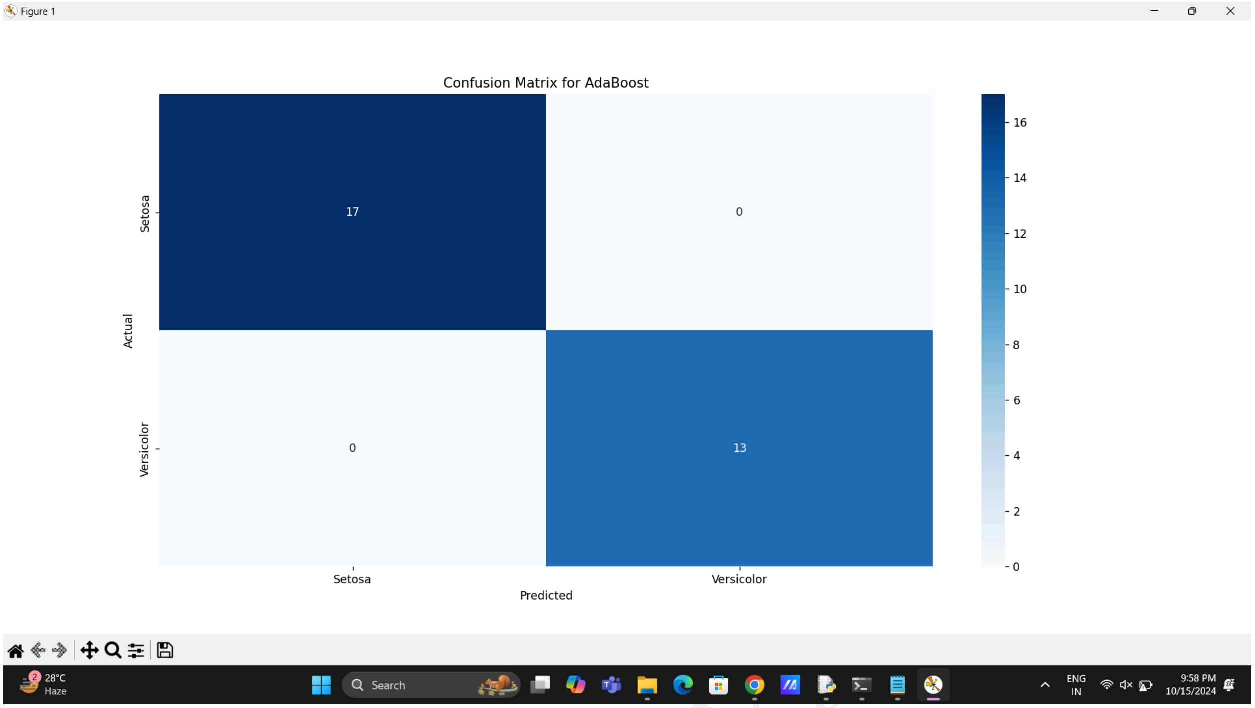
CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset (using the Iris dataset as an example)
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
data = pd.read_csv(url, header=None, names=column_names)

# Preprocess the data
# Map species to numerical values (Iris-setosa: 0, Iris-versicolor: 1, Iris-virginica: 2)
data['species'] = data['species'].map({'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2})

# For binary classification, let's select only two classes (setosa and versicolor)
data_binary = data[data['species'] < 2]

# Split the dataset into features and target
X = data_binary.drop('species', axis=1)  # Features
y = data_binary['species']  # Target variable

# 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)

# Initialize the base classifier (Decision Tree)
base_classifier = DecisionTreeClassifier(max_depth=1)  # Stump

# Initialize and train the AdaBoost classifier
ada_model = AdaBoostClassifier(estimator=base_classifier, n_estimators=50, random_state=42)  #
Change here
ada_model.fit(X_train, y_train)

# Make predictions
y_pred = ada_model.predict(X_test)
```

```
# Evaluate the performance of the AdaBoost model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the AdaBoost model: {accuracy:.2f}")

# Print classification report and confusion matrix
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix visualization
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Setosa', 'Versicolor'],
yticklabels=['Setosa', 'Versicolor'])
plt.title('Confusion Matrix for AdaBoost')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

OUTPUT:






```
========== RESTART: C:/Users/ASUS/OneDrive/Documents/sarahshriyan ai prac 6.py =========

Warning (from warnings module):
  File "C:\Users\ASUS\AppData\Roaming\Python\Python313\site-packages\sklearn\ensemble\_w
eight_boosting.py", line 527
    warnings.warn(
FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in
1.6. Use the SAMME algorithm to circumvent this warning.
Accuracy of the AdaBoost model: 1.00

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       1.00      1.00      1.00        13

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Ln: 166 Col: 0

28°C
Haze
Search

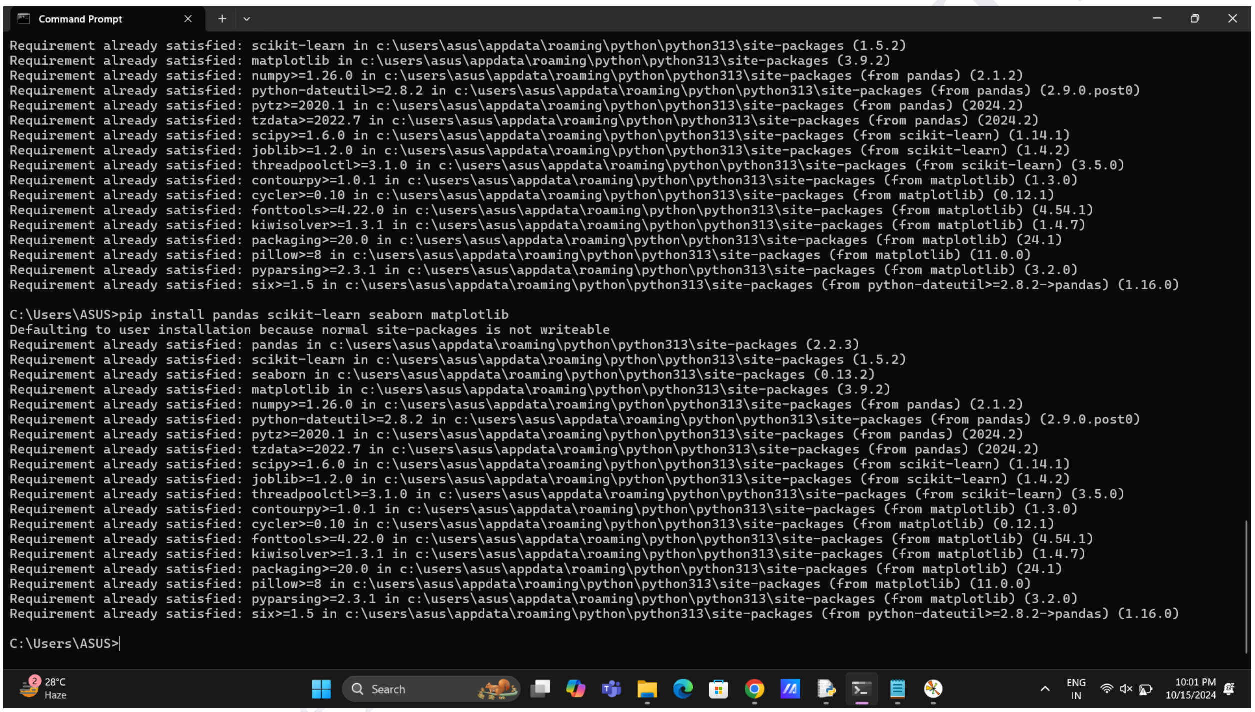
Practical No. 7

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset (using the Iris dataset as an example)
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
data = pd.read_csv(url, header=None, names=column_names)

# Preprocess the data
# Map species to numerical values (Iris-setosa: 0, Iris-versicolor: 1, Iris-virginica: 2)
data['species'] = data['species'].map({'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2})

# Split the dataset into features and target
X = data.drop('species', axis=1)  # Features
y = data['species']  # Target variable

# 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)

# Initialize and train the Naive Bayes classifier
naive_bayes_model = GaussianNB()
naive_bayes_model.fit(X_train, y_train)

# Make predictions
y_pred = naive_bayes_model.predict(X_test)

# Evaluate the performance of the Naive Bayes model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Naive Bayes model: {accuracy:.2f}")

# Print classification report and confusion matrix
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
# Confusion matrix visualization
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Setosa', 'Versicolor',
'Virginica'], yticklabels=['Setosa', 'Versicolor', 'Virginica'])
plt.title('Confusion Matrix for Naive Bayes Classifier')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

OUTPUT:




```
========== RESTART: C:/Users/ASUS/OneDrive/Documents/sarahshriyan ai prac 7.py =========
Accuracy of the Naive Bayes model: 0.98

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.92      0.96        13
           2       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45
```

Ln: 183 Col: 0

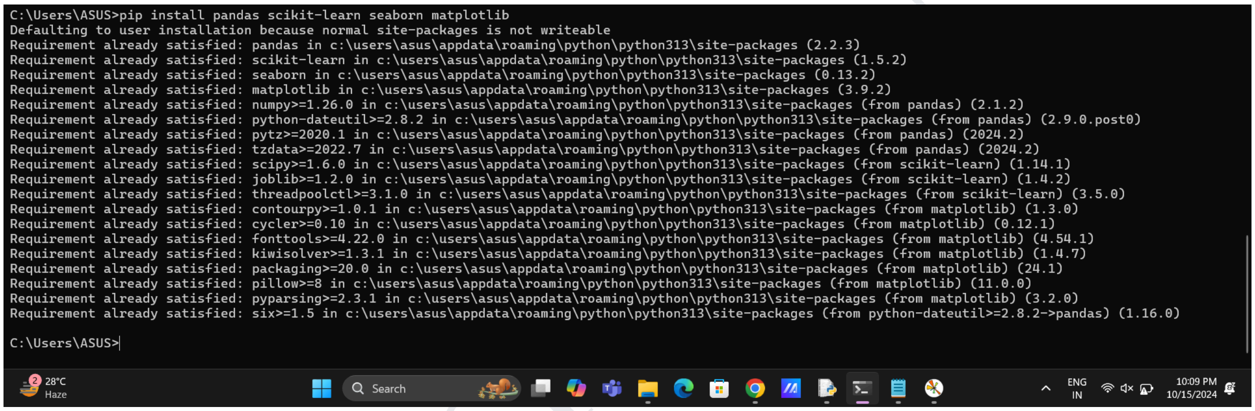
Practical No. 8

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset (using the Iris dataset as an example)
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
data = pd.read_csv(url, header=None, names=column_names)

# Preprocess the data
# Map species to numerical values (Iris-setosa: 0, Iris-versicolor: 1, Iris-virginica: 2)
data['species'] = data['species'].map({'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2})

# Split the dataset into features and target
X = data.drop('species', axis=1)  # Features
y = data['species']  # Target variable

# 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)

# Initialize the K-NN classifier with K=3
k = 3
knn_model = KNeighborsClassifier(n_neighbors=k)

# Train the K-NN model
knn_model.fit(X_train, y_train)

# Make predictions
y_pred = knn_model.predict(X_test)

# Evaluate the performance of the K-NN model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the K-NN model: {accuracy:.2f}")

# Print classification report and confusion matrix
print("\nClassification Report:")
```

```
print(classification_report(y_test, y_pred))

# Confusion matrix visualization
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Setosa', 'Versicolor',
'Virginica'], yticklabels=['Setosa', 'Versicolor', 'Virginica'])
plt.title('Confusion Matrix for K-NN Classifier')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

OUTPUT:




```
========== RESTART: C:/Users/ASUS/OneDrive/Documents/sarahshriyan ai prac 8.py =========
Accuracy of the K-NN model: 1.00

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Ln: 198 Col: 0

28°C
Mostly cloudy
Search

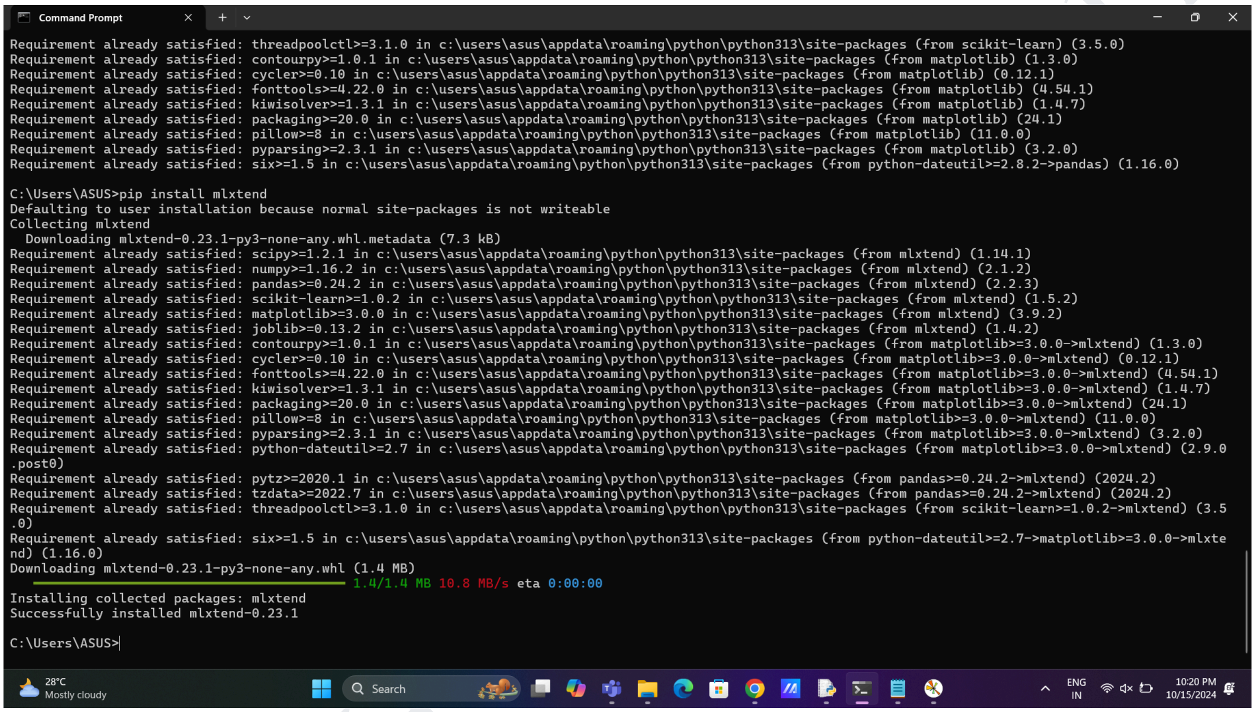
Practical No. 9

CODE

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Sample transaction data
# Each row represents a transaction and each column represents an item.
data = {
    'TransactionID': [1, 2, 3, 4, 5, 6],
    'Items': [
        ['Milk', 'Bread', 'Diaper'],
        ['Bread', 'Diaper', 'Beer'],
        ['Milk', 'Diaper', 'Beer'],
        ['Bread', 'Milk'],
        ['Milk', 'Diaper', 'Beer'],
        ['Bread', 'Milk']
    ]
}

# Convert the transaction data into a DataFrame
df = pd.DataFrame(data)

# Convert the list of items into a one-hot encoded DataFrame
one_hot = df['Items'].str.join('|').str.get_dummies()

# Convert the one-hot DataFrame to boolean type
one_hot = one_hot.astype(bool)

# Display the one-hot encoded DataFrame
print("One-Hot Encoded DataFrame:")
print(one_hot)

# Apply Apriori algorithm to find frequent itemsets with a minimum support of 0.4
frequent_itemsets = apriori(one_hot, min_support=0.4, use_colnames=True)

# Display the frequent itemsets
print("\nFrequent Itemsets:")
print(frequent_itemsets)

# Generate association rules with a minimum confidence of 0.6
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.6)
```

```
# Display the generated association rules
print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

OUTPUT



```
>>>
========== RESTART: C:/Users/ASUS/OneDrive/Documents/sarahshriyan ai prac 9.py =========
One-Hot Encoded DataFrame:
    Beer  Bread  Diaper   Milk
0  False   True    True   True
1   True   True    True  False
2   True  False    True   True
3  False   True   False   True
4   True  False    True   True
5  False   True   False   True

Frequent Itemsets:
    support         itemsets
0  0.500000           (Beer)
1  0.666667          (Bread)
2  0.666667         (Diaper)
3  0.833333           (Milk)
4  0.500000  (Diaper, Beer)
5  0.500000   (Bread, Milk)
6  0.500000  (Diaper, Milk)

Association Rules:
  antecedents consequents  support  confidence  lift
0    (Diaper)      (Beer)      0.5        0.75   1.5
1      (Beer)    (Diaper)      0.5        1.00   1.5
2     (Bread)      (Milk)      0.5        0.75   0.9
3      (Milk)     (Bread)      0.5        0.60   0.9
4    (Diaper)      (Milk)      0.5        0.75   0.9
5      (Milk)    (Diaper)      0.5        0.60   0.9
>>>
```

Ln: 261 Col: 0

28°C
Mostly cloudy
Search

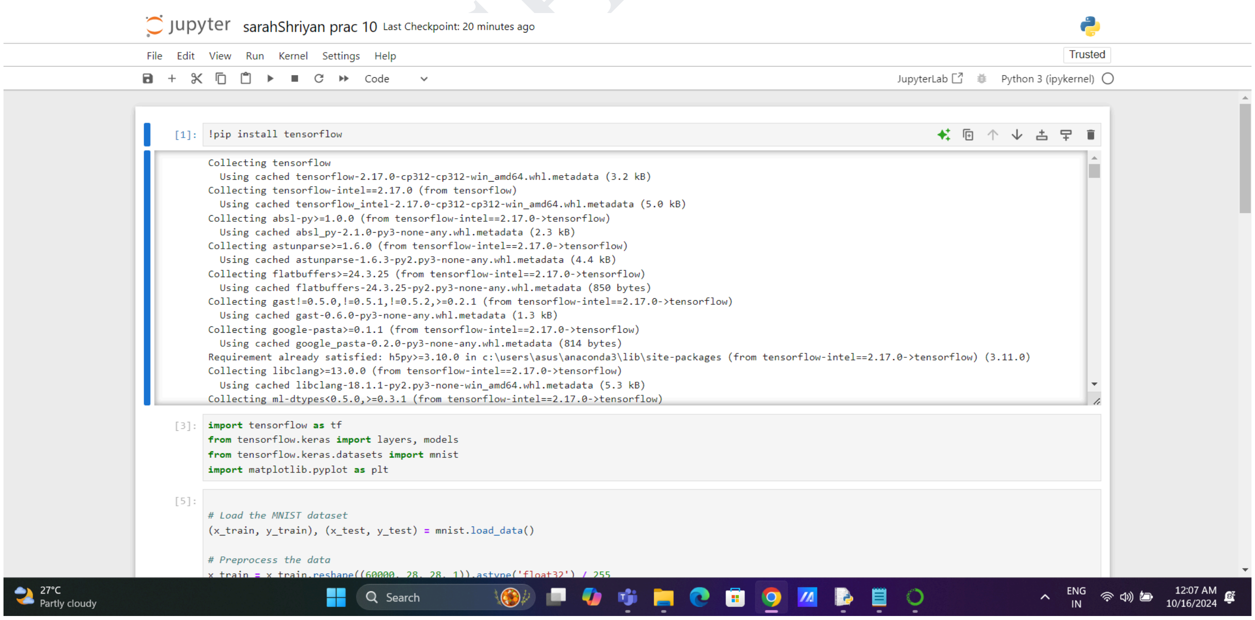
Practical No. 10

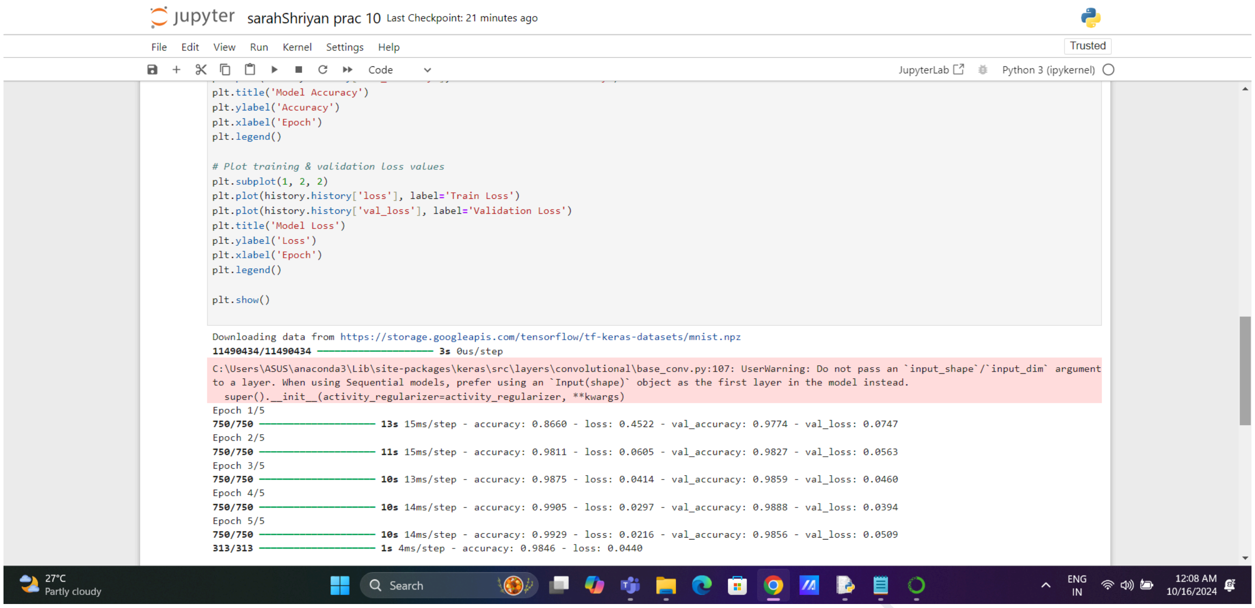
CODE:

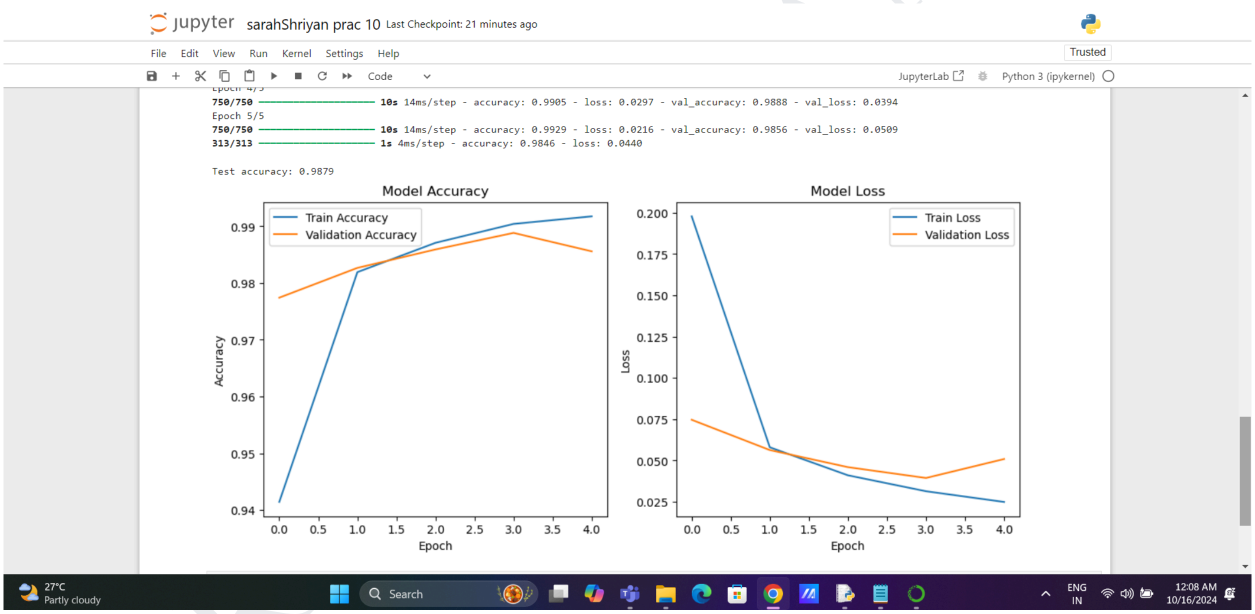
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess the data
x_train = x_train.reshape((60000, 28, 28, 1)).astype('float32') / 255
x_test = x_test.reshape((10000, 28, 28, 1)).astype('float32') / 255

# Convert labels to categorical one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Build the neural network model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

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

# Train the model
history = model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'\nTest accuracy: {test_acc:.4f}')
```

```
# Plot training & validation accuracy values
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()

plt.show()
```

OUTPUT:




jupyter sarahShriyan prac 10 Last Checkpoint: 21 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
C:\Users\ASUS\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 13s 15ms/step - accuracy: 0.8660 - loss: 0.4522 - val_accuracy: 0.9774 - val_loss: 0.0747
Epoch 2/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy: 0.9811 - loss: 0.0605 - val_accuracy: 0.9827 - val_loss: 0.0563
Epoch 3/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.9875 - loss: 0.0414 - val_accuracy: 0.9859 - val_loss: 0.0460
Epoch 4/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 10s 14ms/step - accuracy: 0.9905 - loss: 0.0297 - val_accuracy: 0.9888 - val_loss: 0.0394
Epoch 5/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 10s 14ms/step - accuracy: 0.9929 - loss: 0.0216 - val_accuracy: 0.9856 - val_loss: 0.0509
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9846 - loss: 0.0440
27°C
Partly cloudy
Search
ENG
IN
12:08 AM
10/16/2024


jupyter sarahShriyan prac 10 Last Checkpoint: 21 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
750/750 ━━━━━━━━━━━━━━━━━━━━ 10s 14ms/step - accuracy: 0.9905 - loss: 0.0297 - val_accuracy: 0.9888 - val_loss: 0.0394
Epoch 5/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 10s 14ms/step - accuracy: 0.9929 - loss: 0.0216 - val_accuracy: 0.9856 - val_loss: 0.0509
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9846 - loss: 0.0440
Test accuracy: 0.9879
Model Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch
Model Loss
Train Loss
Validation Loss
Loss
Epoch
27°C
Partly cloudy
Search
ENG
IN
12:08 AM
10/16/2024