**SUPERVISED and DEEP LEARNING**

**PRACTICAL FILE**



**Submitted to: Ms. Meenakshi Sihag**

**Name: SARTHAK GUPTA**

**Enrolment Number: 01876803121**

**Branch: IT-3 (MLDA)**

**Index**

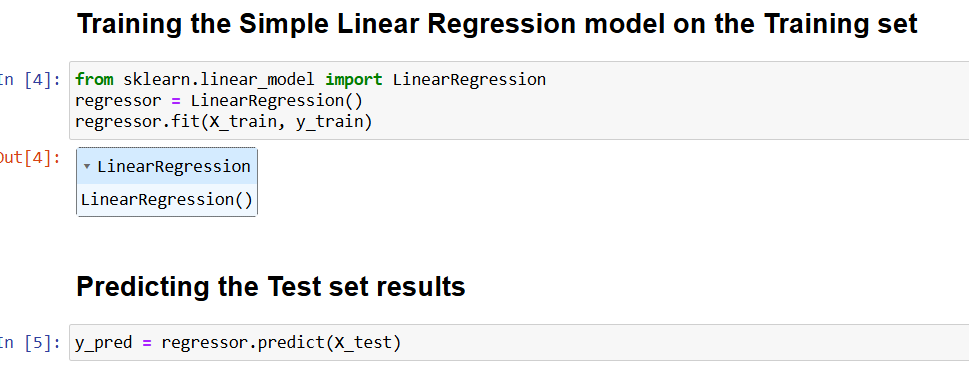
|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **Title** | **Date** | **Sign.** |
|  | Linear regression: Implement linear regression on a dataset and evaluate the model's performance. |  |  |
| 2. | Logistic regression: Implement logistic regression on a binary classification dataset and evaluate the model's performance. |  |  |
| 3. | k-Nearest Neighbors (k-NN): Implement k-NN algorithm on a dataset and evaluate the model's performance. |  |  |
| 4. | Decision Trees: Implement decision trees on a dataset and evaluate the model's performance |  |  |
| 5. | Random Forest: Implement random forest algorithm on a dataset and evaluate the model's performance. |  |  |
| 6. | Support Vector Machines (SVM): Implement SVM on a dataset and evaluate the model's performance |  |  |
| 7. | Naive Bayes: Implement Naive Bayes algorithm on a dataset and evaluate the model's performance. |  |  |
| 8. | Gradient Boosting: Implement gradient boosting algorithm on a dataset and evaluate the model's performance. |  |  |
| 9. | Convolutional Neural Networks (CNN): Implement CNN on an image classification dataset and evaluate the model's performance |  |  |
| 10. | Recurrent Neural Networks (RNN): Implement RNN on a text classification dataset and evaluate the model's performance. |  |  |
| 11. | Long Short-Term Memory Networks (LSTM): Implement LSTM on a time-series dataset and evaluate the model's performance. |  |  |
| 12. | Autoencoders: Implement autoencoders on an image dataset and evaluate the model's performance. |  |  |
| 13. | Generative Adversarial Networks (GANs): Implement GANs on an image dataset and evaluate the model's performance. |  |  |
| 14. | Transfer Learning: Implement transfer learning on an image dataset and evaluate the model's performance |  |  |
| 15. | Implement reinforcement learning in a game environment and evaluate performance . |  |  |

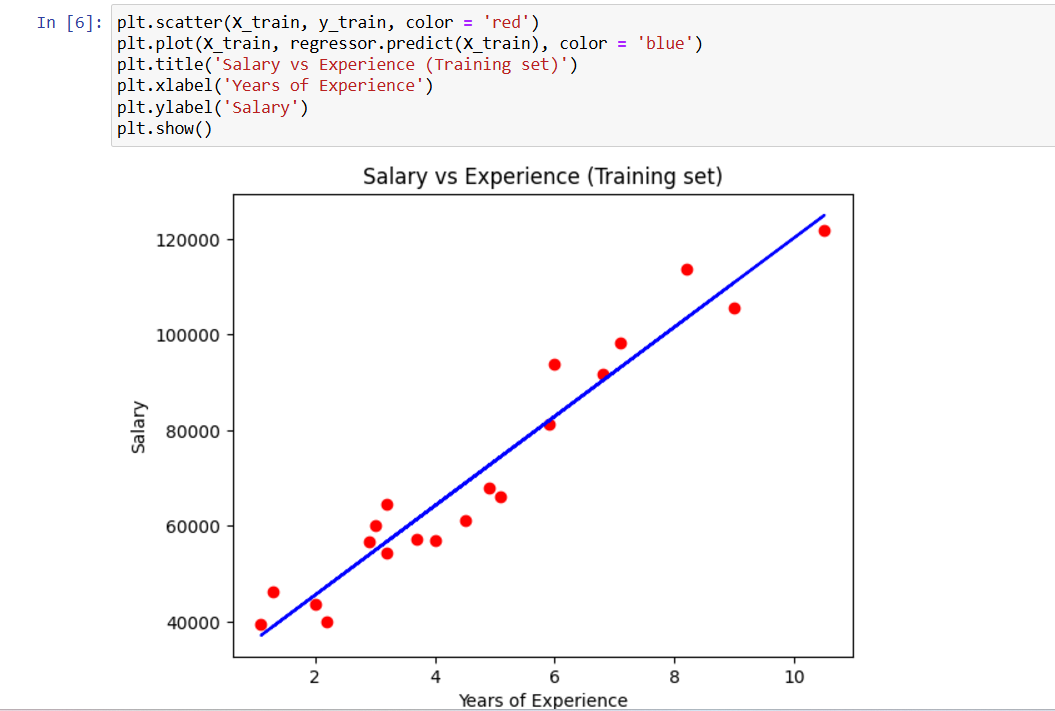
**Practical -1**

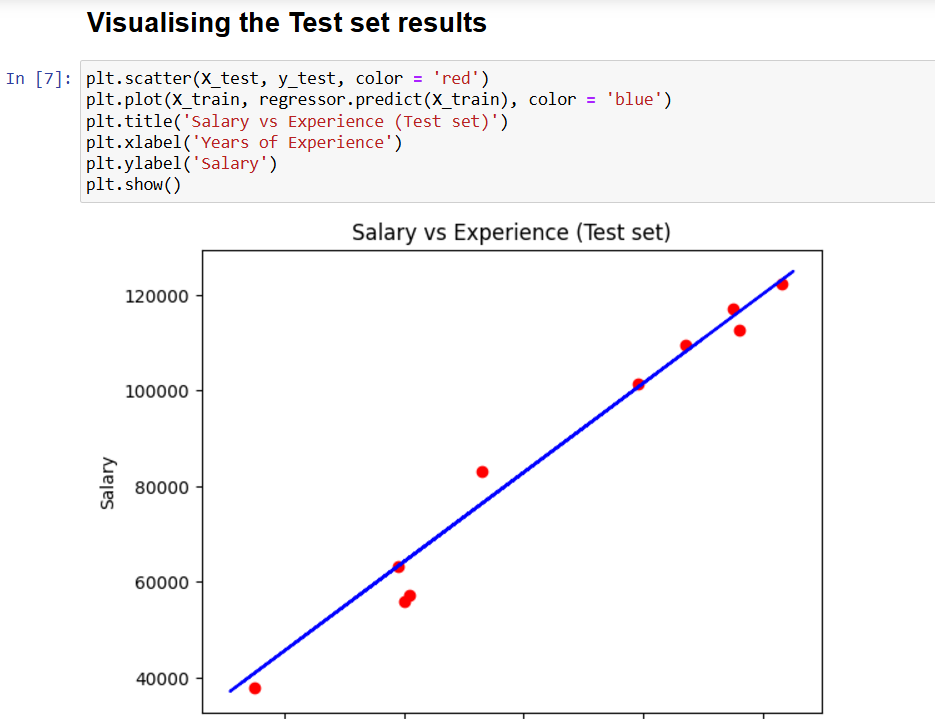
**AIM :** Linear regression: Implement linear regression on a dataset and evaluate the model's performance.

**Code :**

****

****

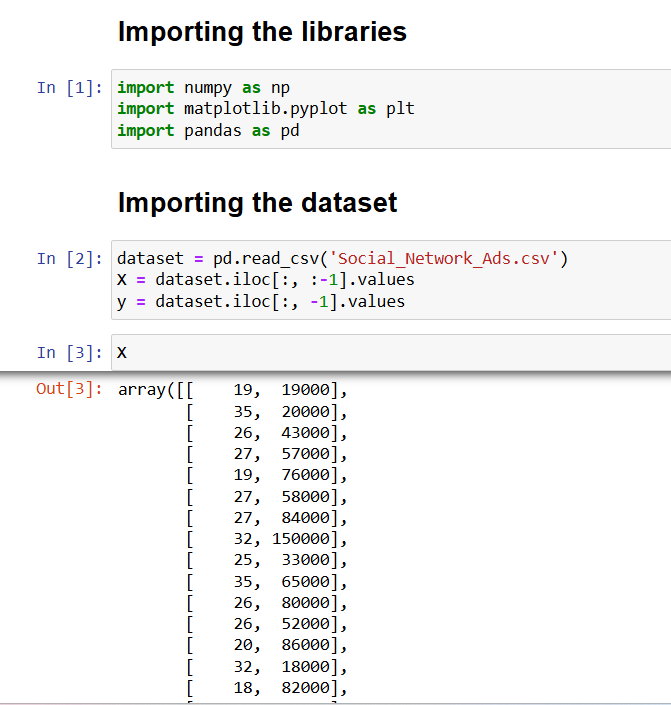
****

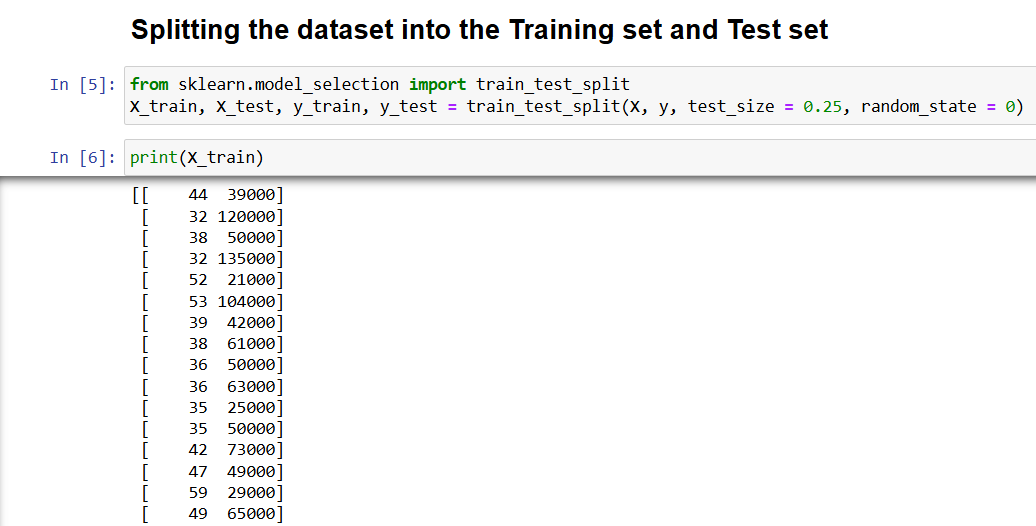
****

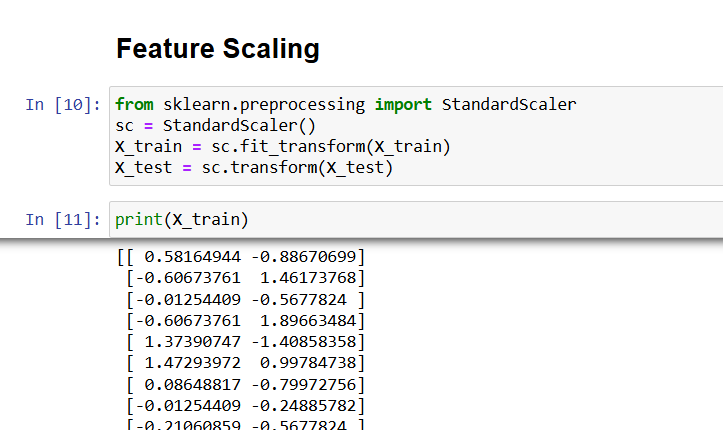
**Practical - 2**

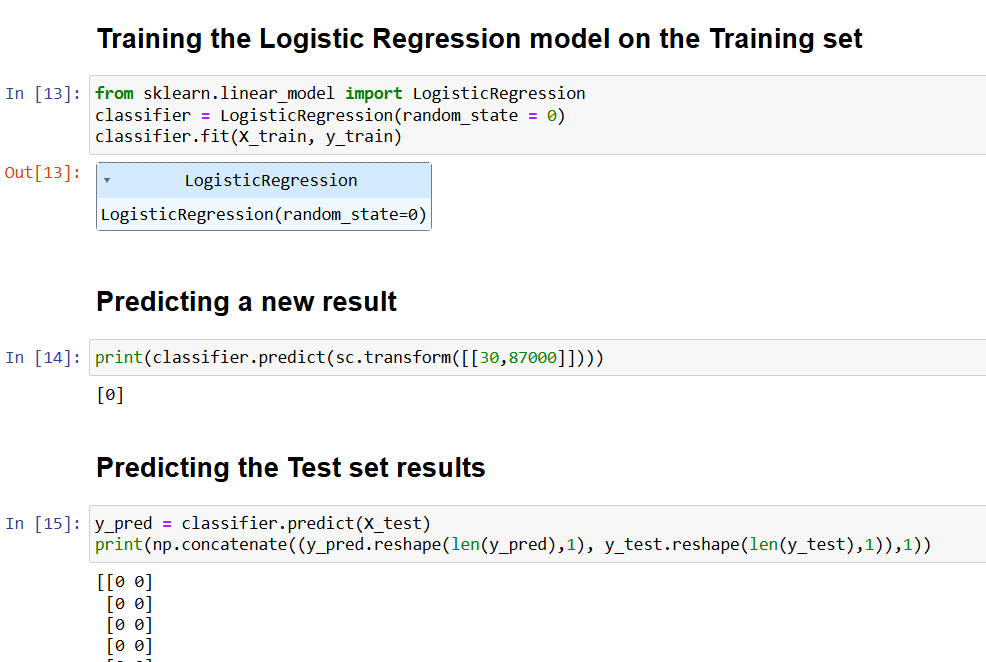
**AIM :** Logistic regression: Implement logistic regression on a binary classification dataset and evaluate the model's performance.

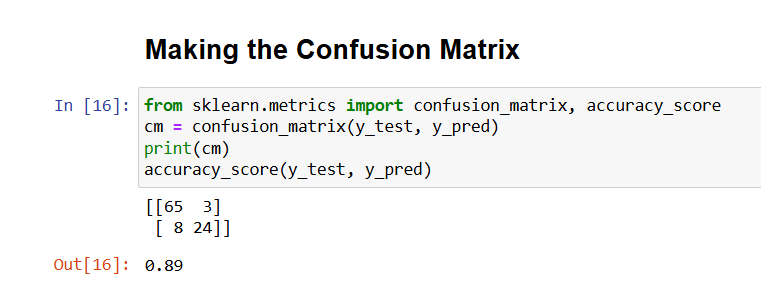
**Code :**

****

****

****

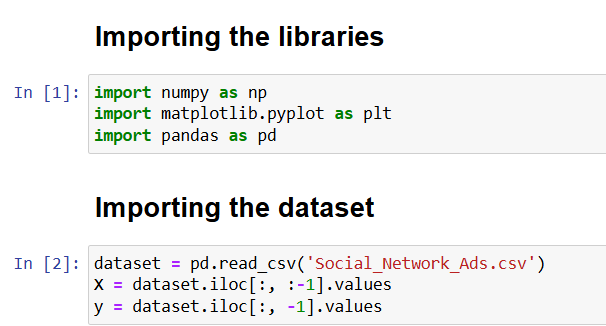
****

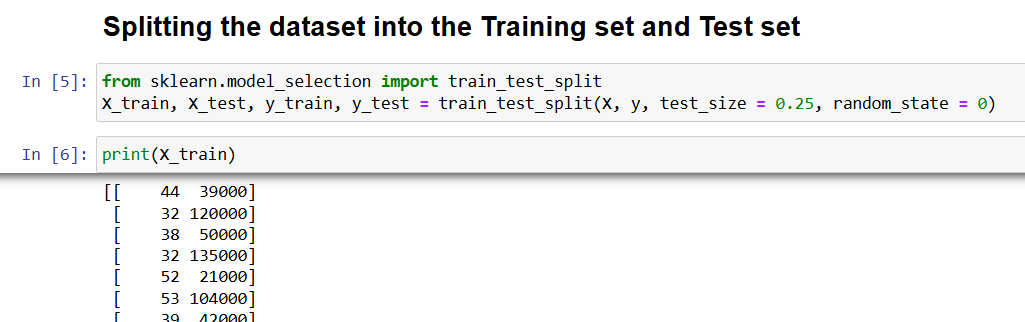
****

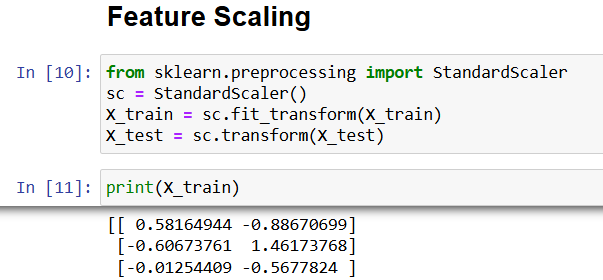
**Practical - 3**

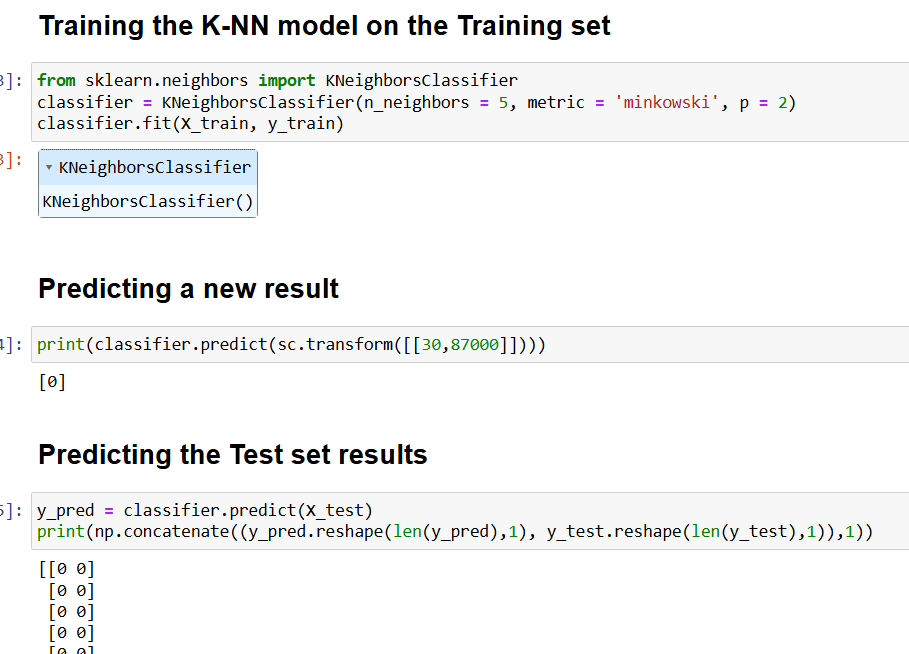
**AIM :** k-Nearest Neighbors (k-NN): Implement k-NN algorithm on a dataset and evaluate the model's performance.

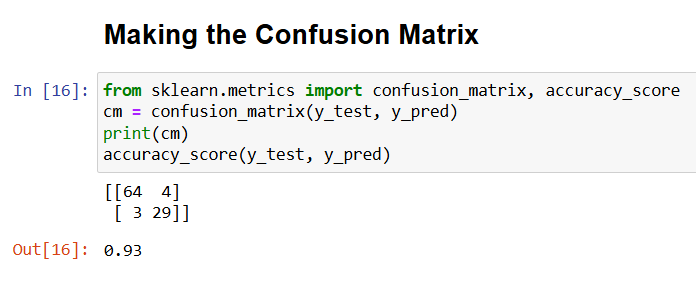
**Code :**

****

****

****

****

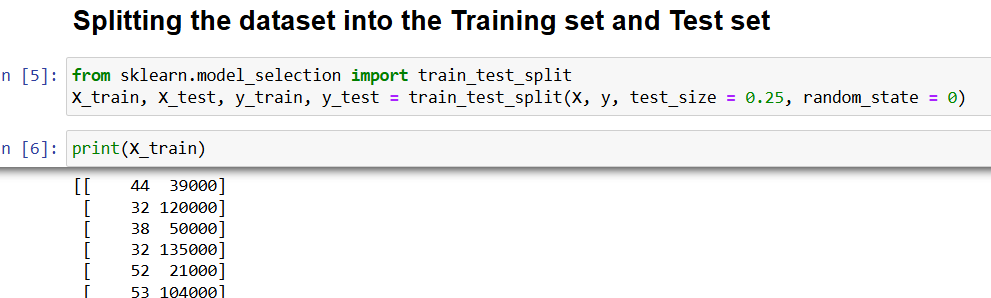
****

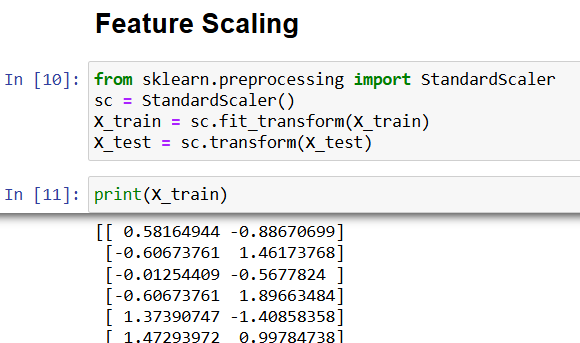
**Practical - 4**

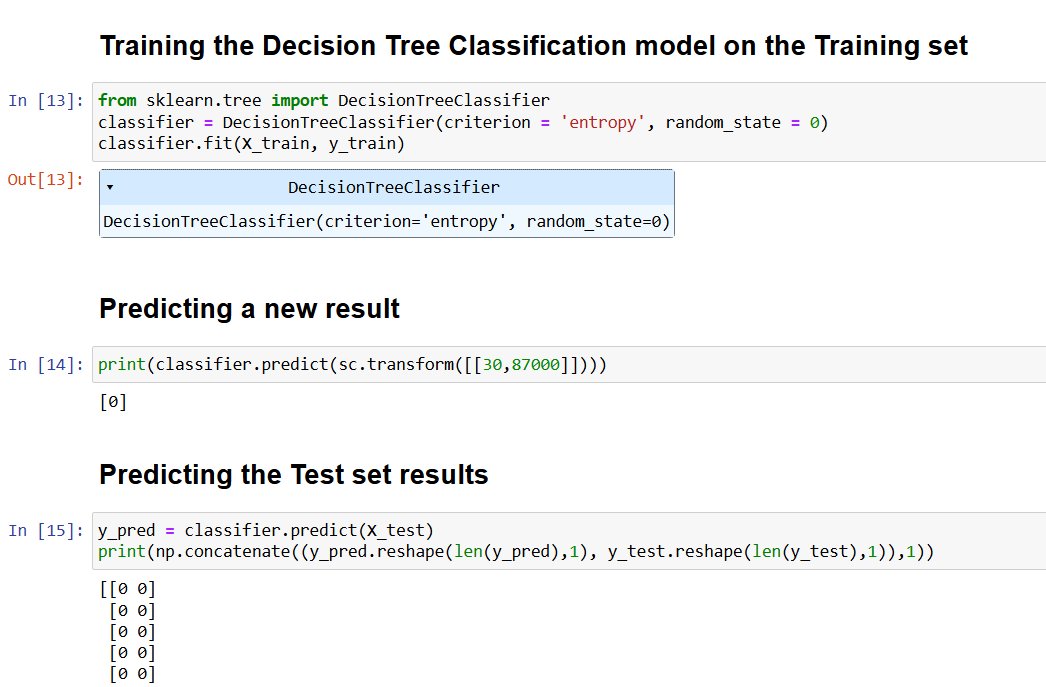
**AIM :** Decision Trees: Implement decision trees on a dataset and evaluate the model's performance.

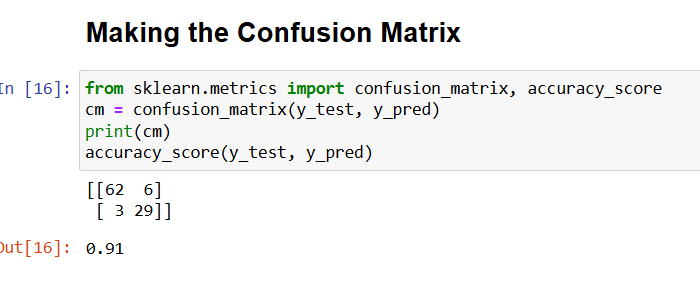
**Code :**

****

****

****

****

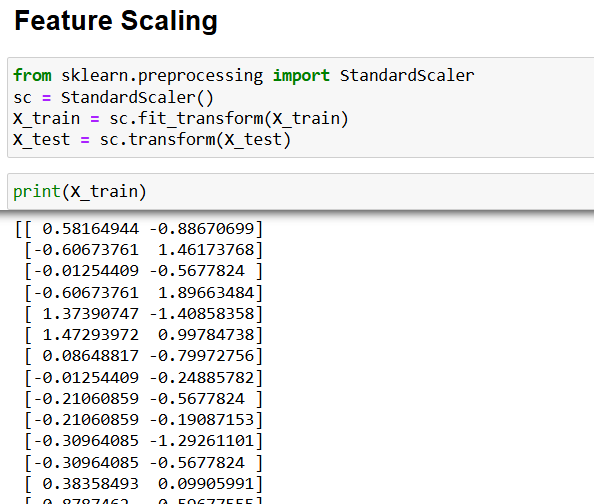
****

**Practical - 5**

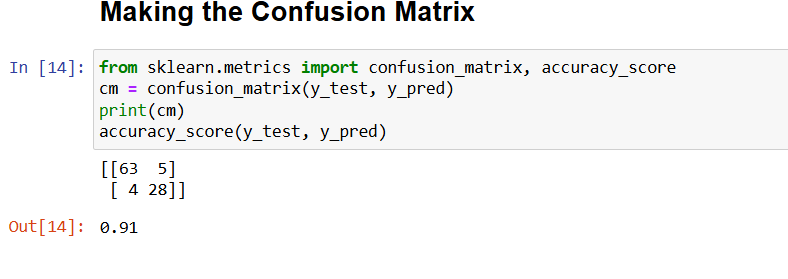
**Aim :** Random Forest: Implement random forest algorithm on a dataset and evaluate the model's performance.

**Code :**

****

****

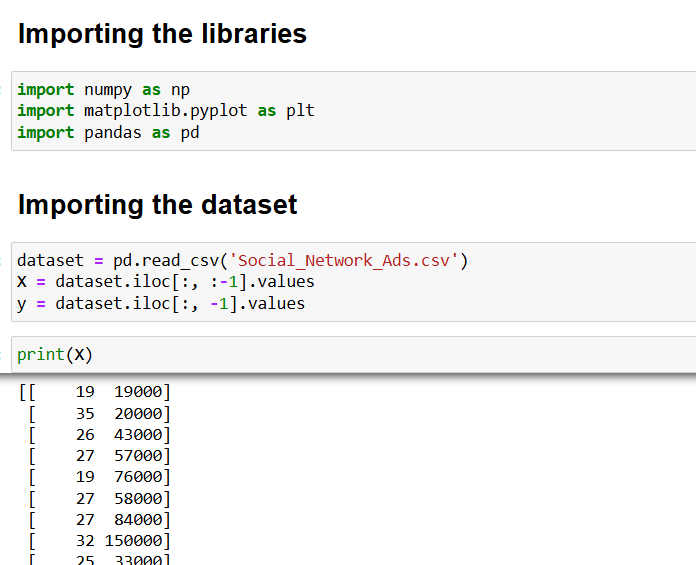
****

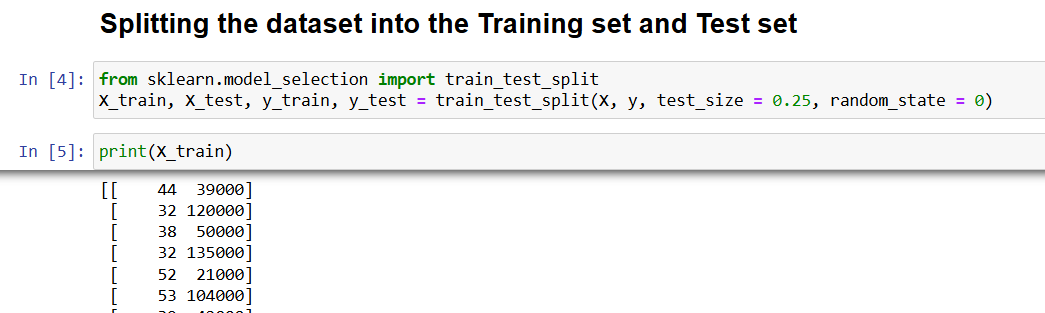
****

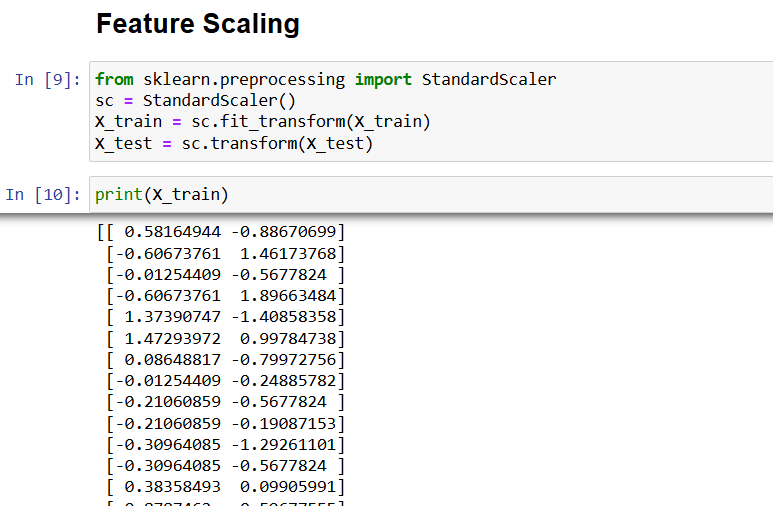
**Practical - 6**

**AIM:** Support Vector Machines (SVM): Implement SVM on a dataset and evaluate the model's performance.

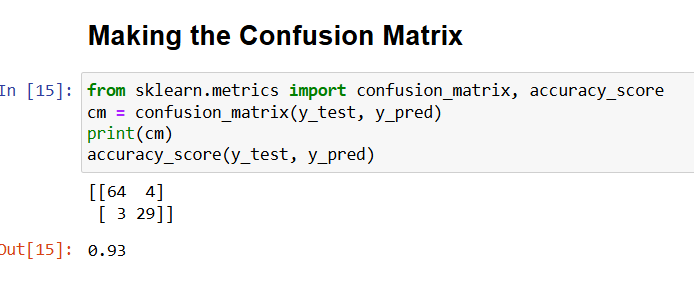
**Code :**

****

****

****

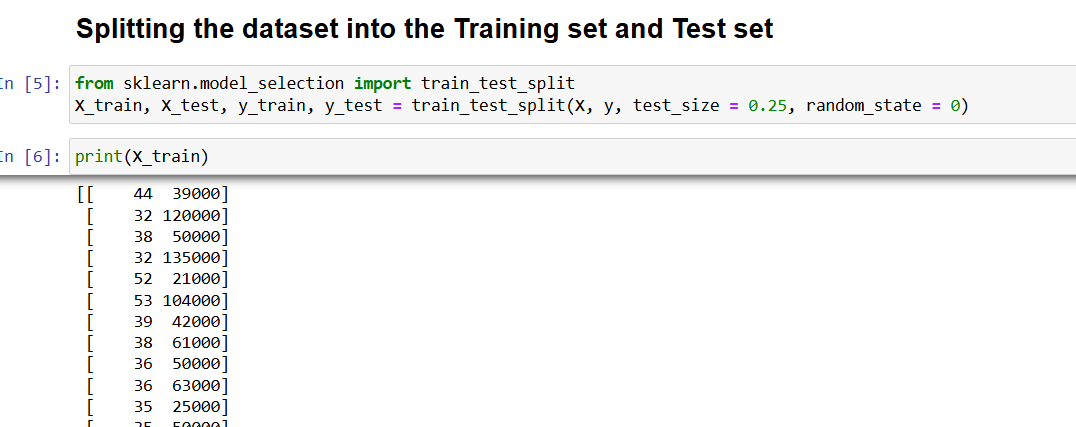
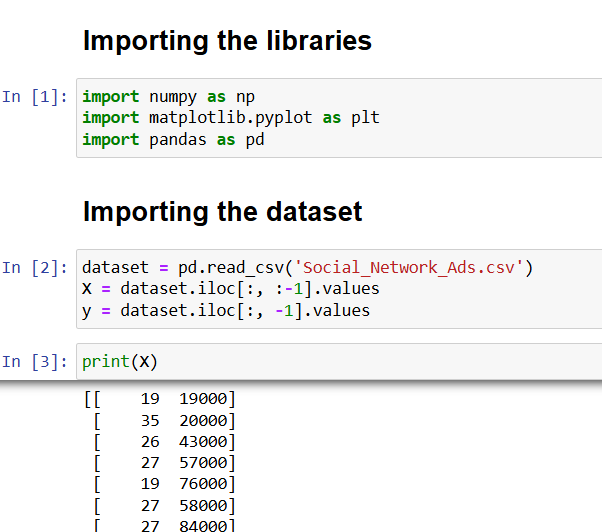
****

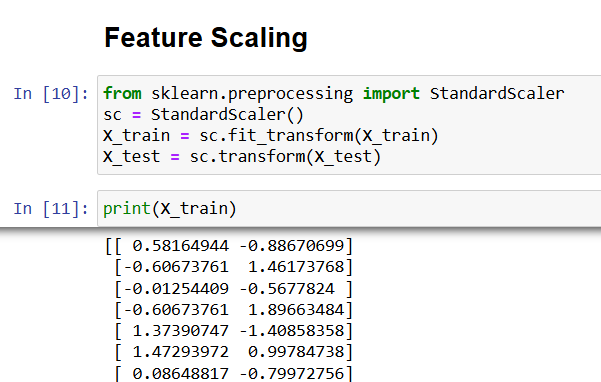
****

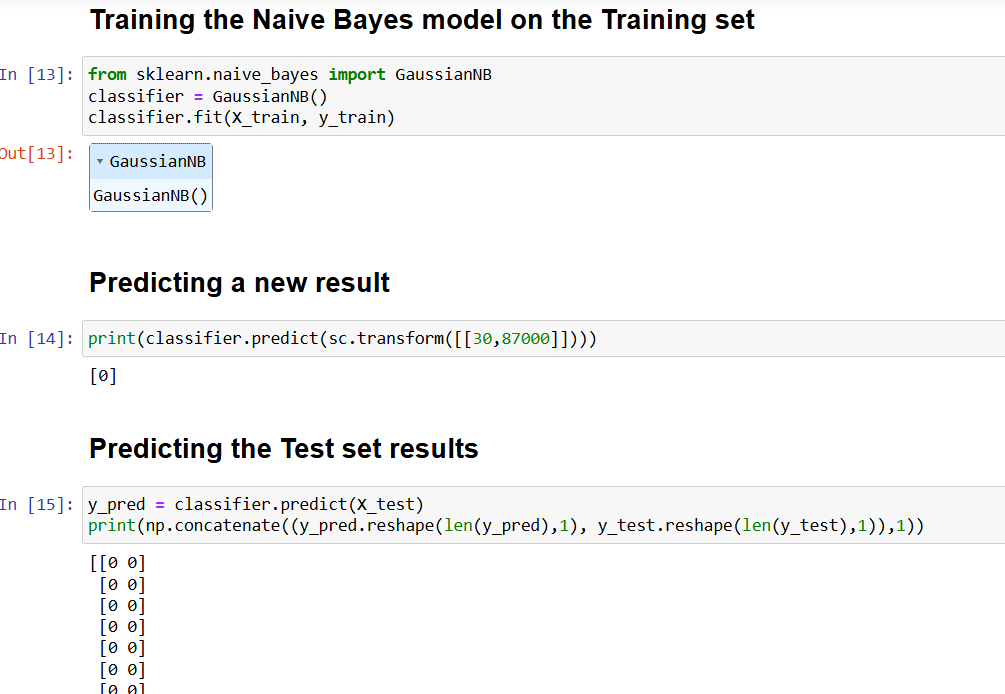
**Practical - 7**

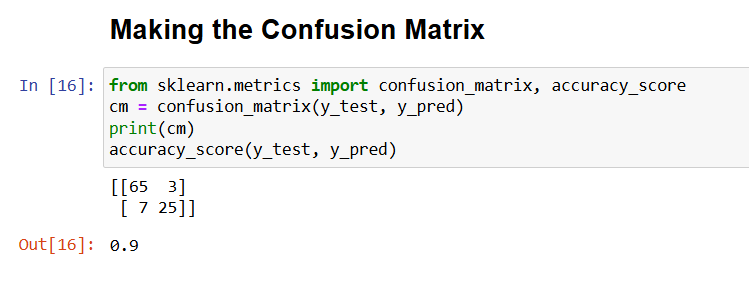
**Aim :** Naive Bayes: Implement Naive Bayes algorithm on a dataset and evaluate the model's performance .

**Code :**

****

****

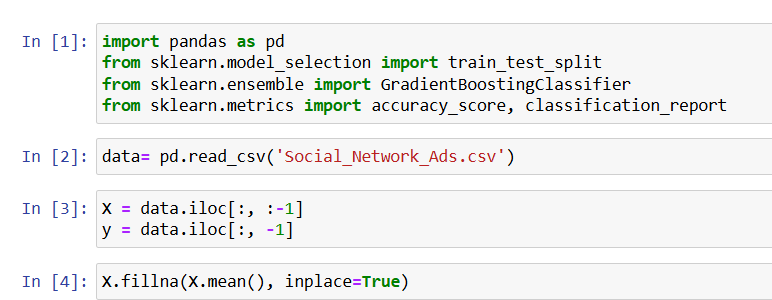
****

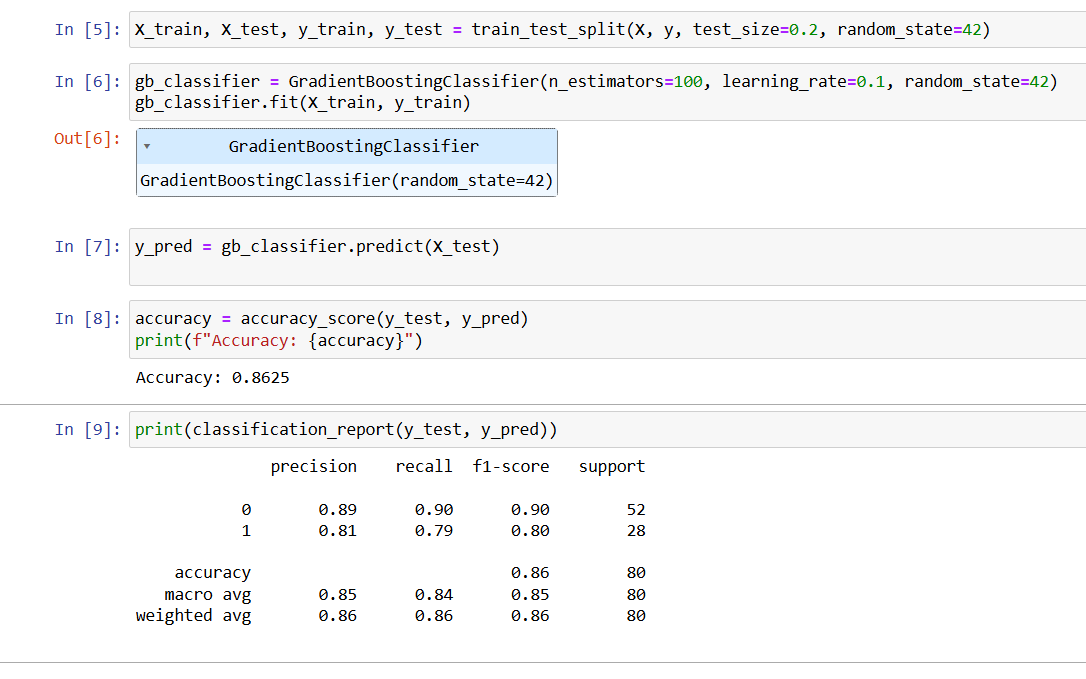
****

**Practical - 8**

**Aim :** Gradient Boosting: Implement gradient boosting algorithm on a dataset and evaluate the model's performance.

**Code :**

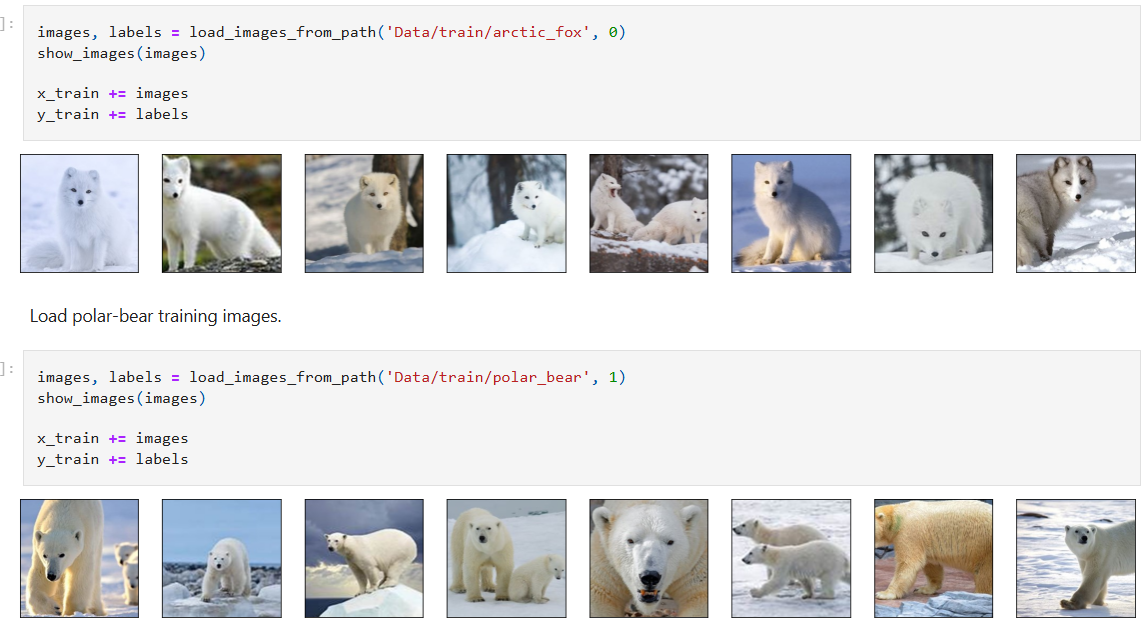
****

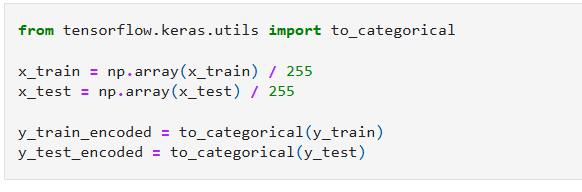
****

**Practical - 9**

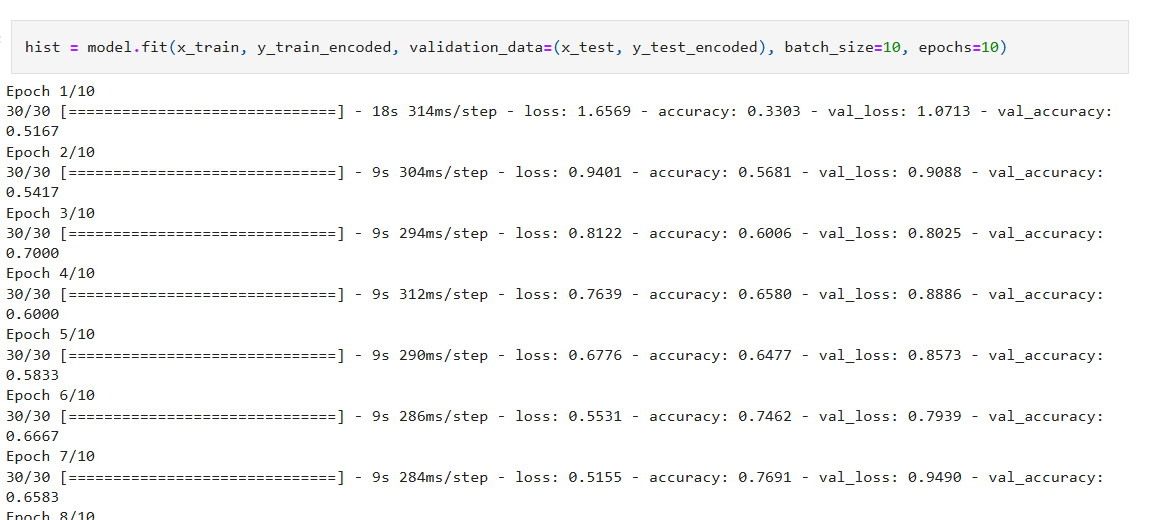
**Aim :** Convolutional Neural Networks (CNN): Implement CNN on an image classification dataset and evaluate the model's performance.

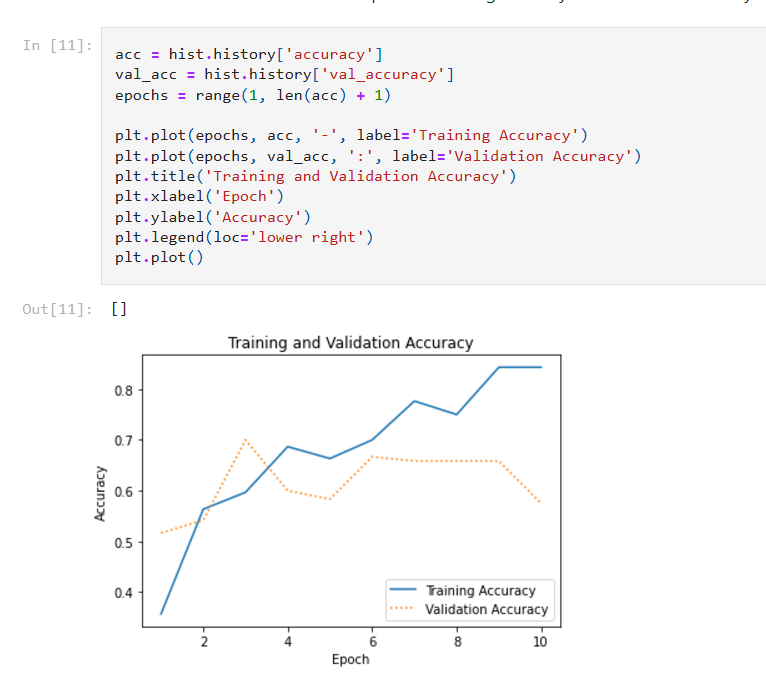
**Code :**

**  
 **

****

****

****

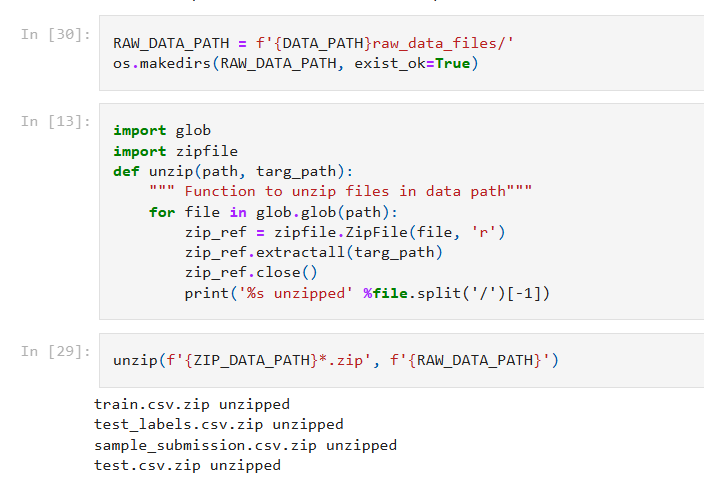
****

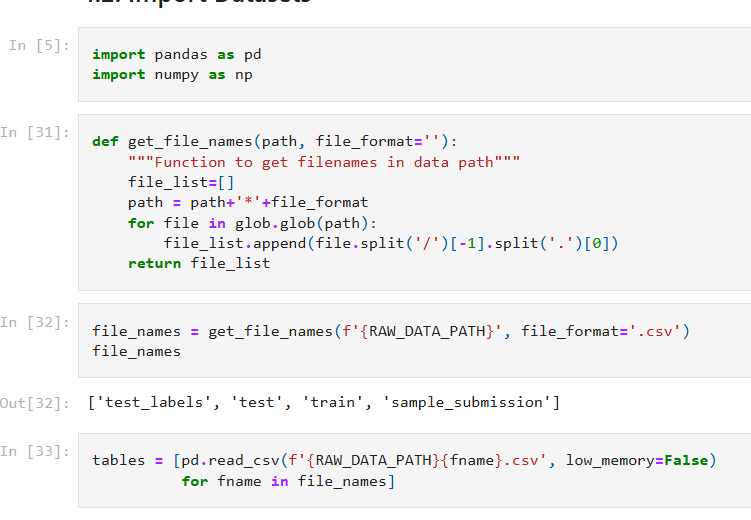
**Practical - 10**

**Aim :** Recurrent Neural Networks (RNN): Implement RNN on a text classification dataset and evaluate the model's performance.

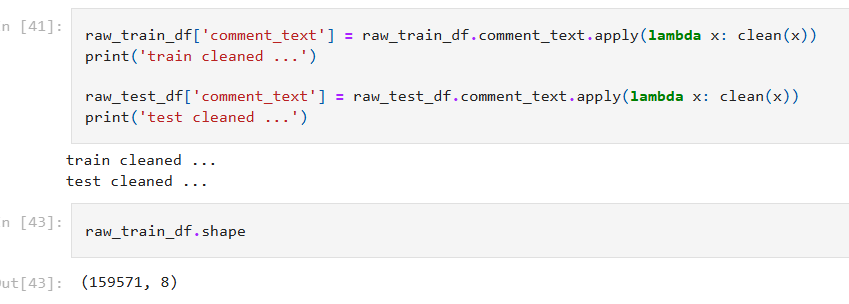
**Code :**

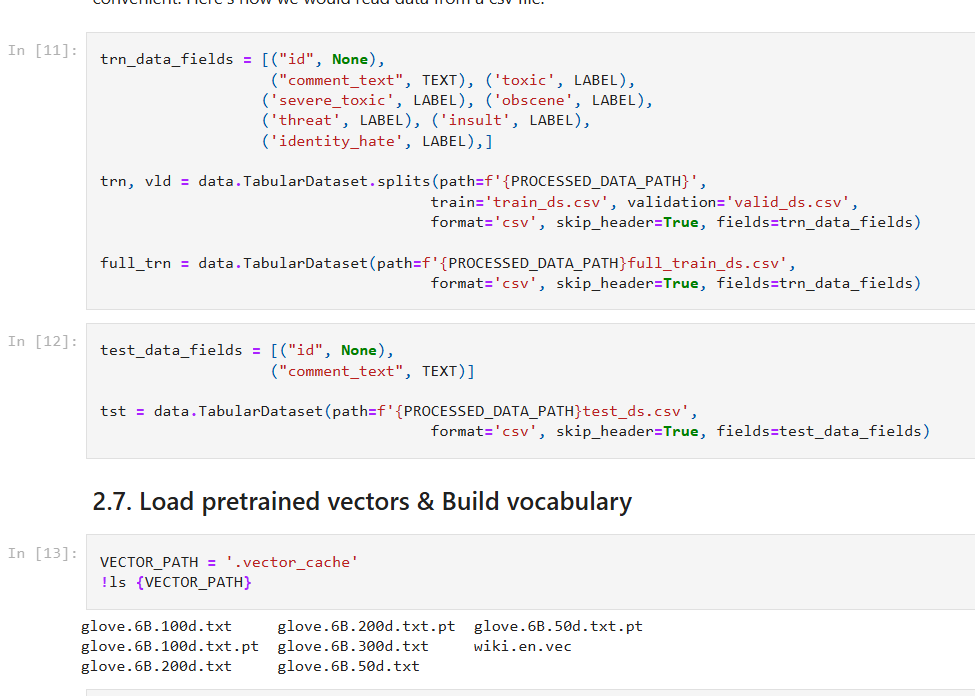
****

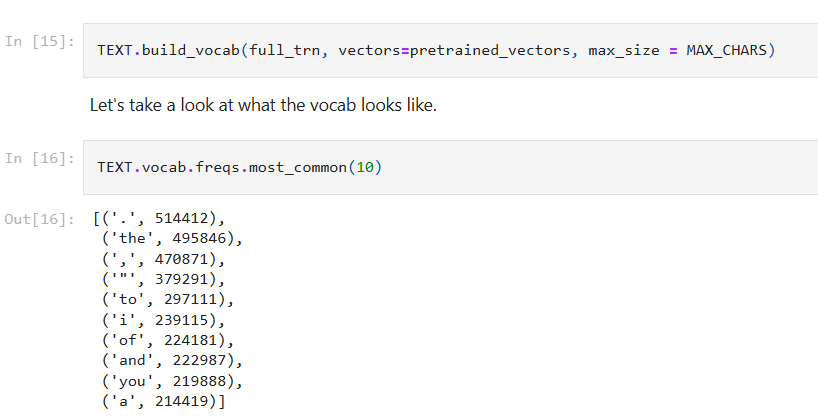
****

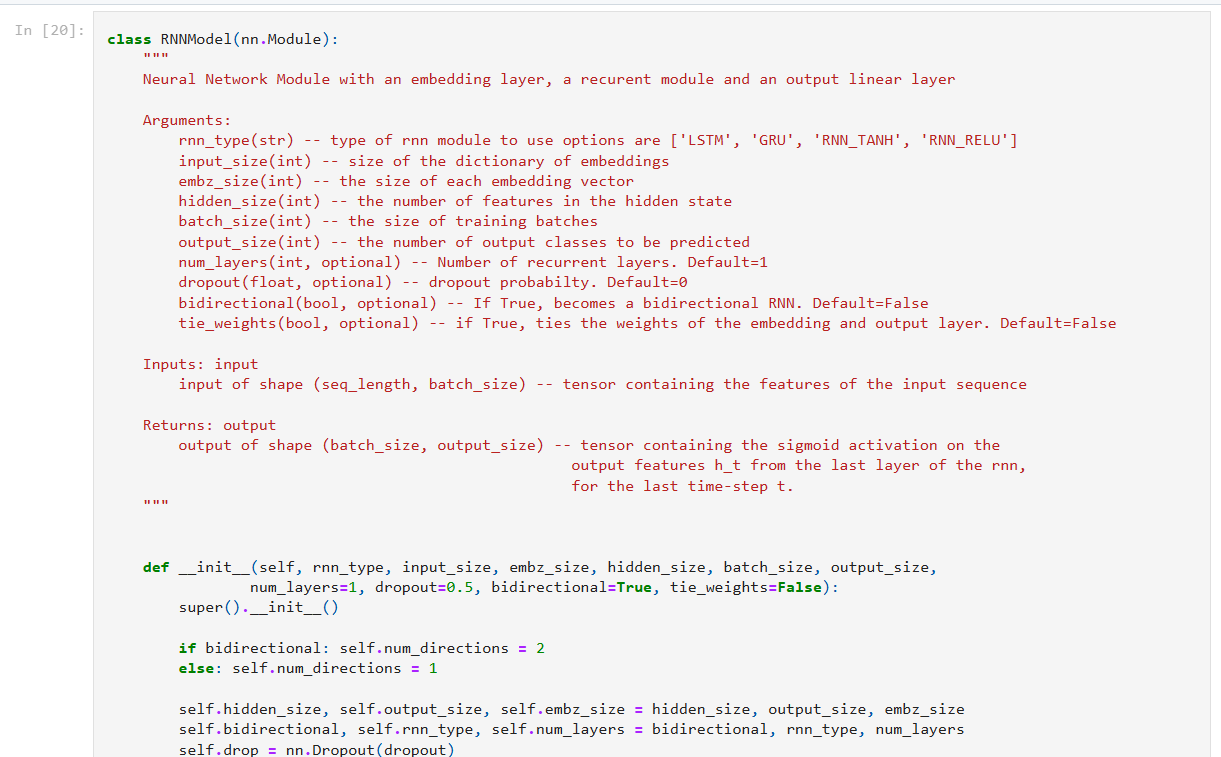
****

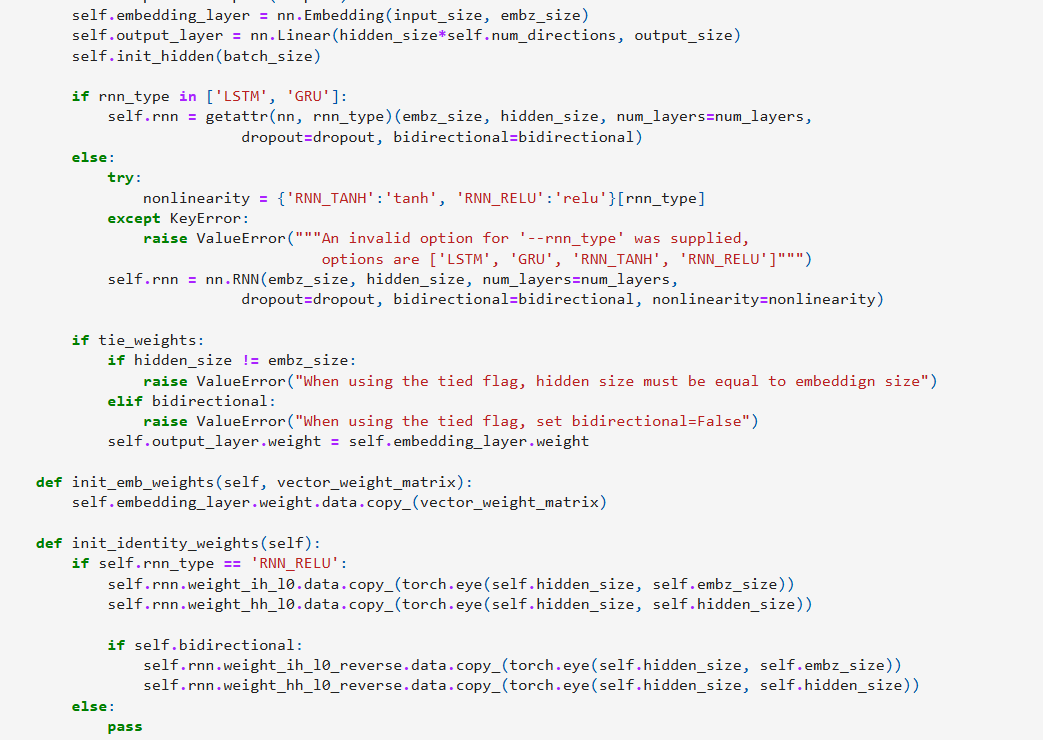
****

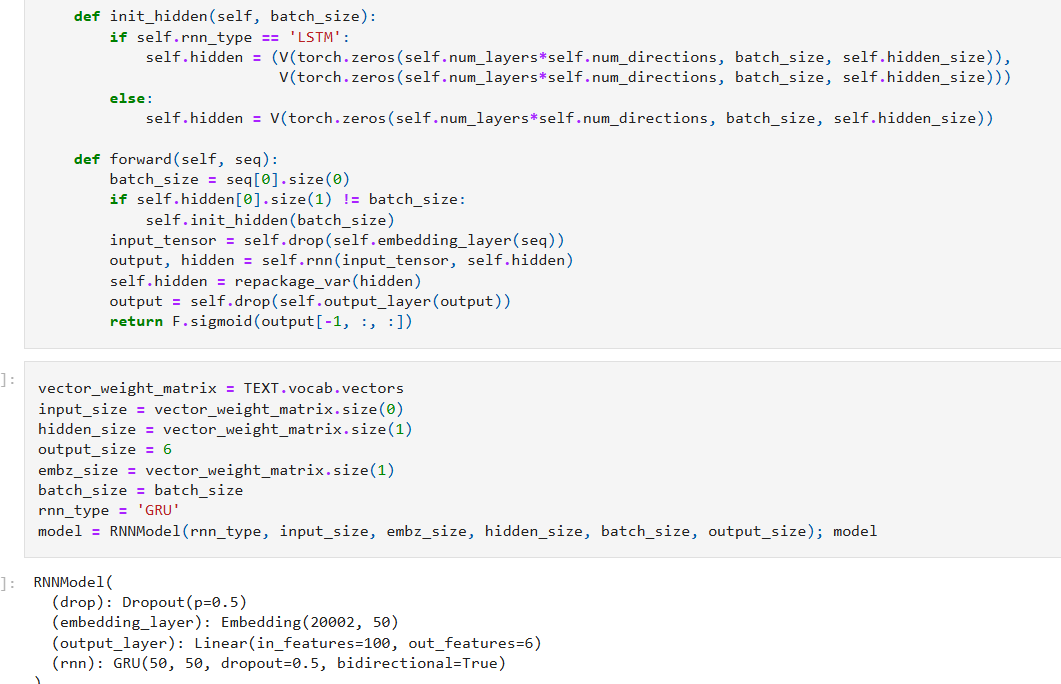
****

****

****

****

****

****

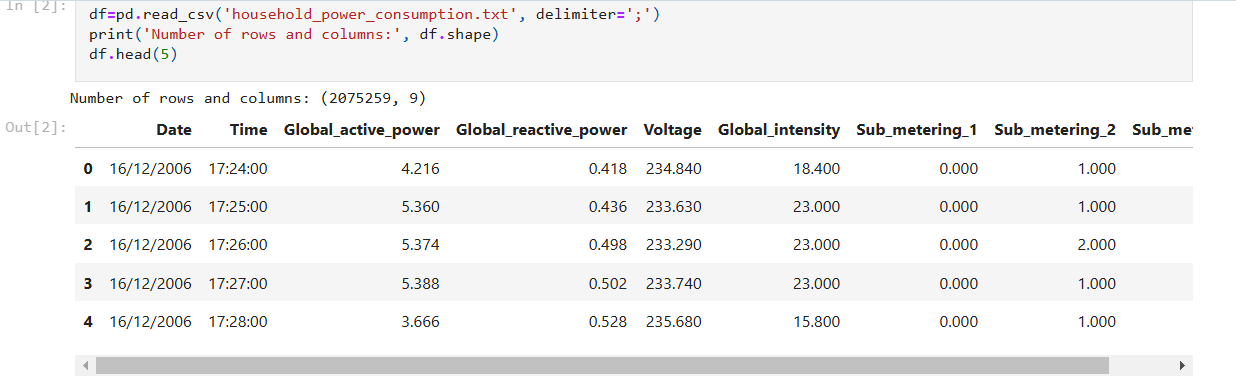
****

**Practical - 11**

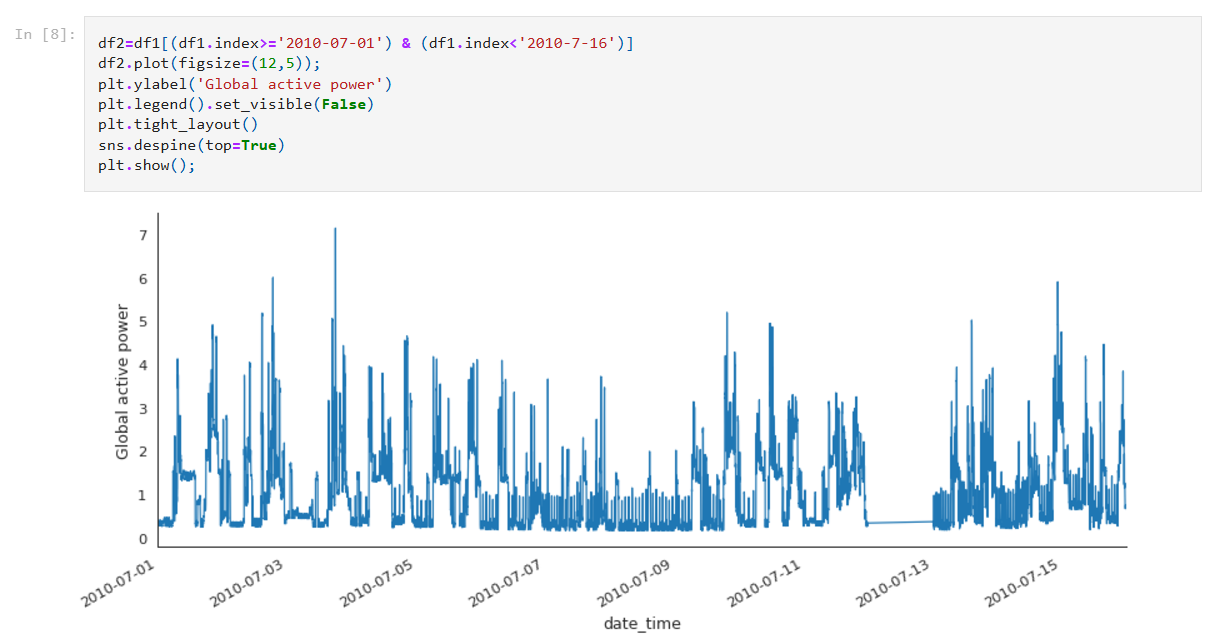
**Aim :** Long Short-Term Memory Networks (LSTM): Implement LSTM on a time-series dataset and evaluate the model's performance.

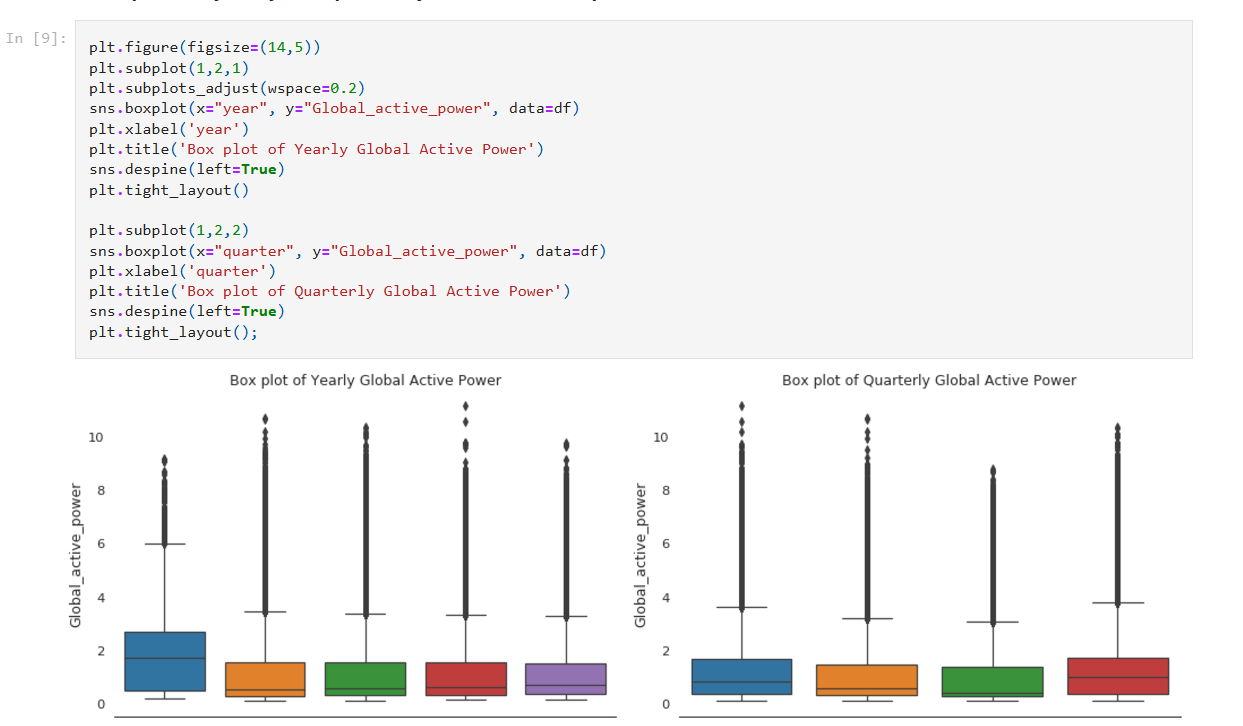
**Code :**

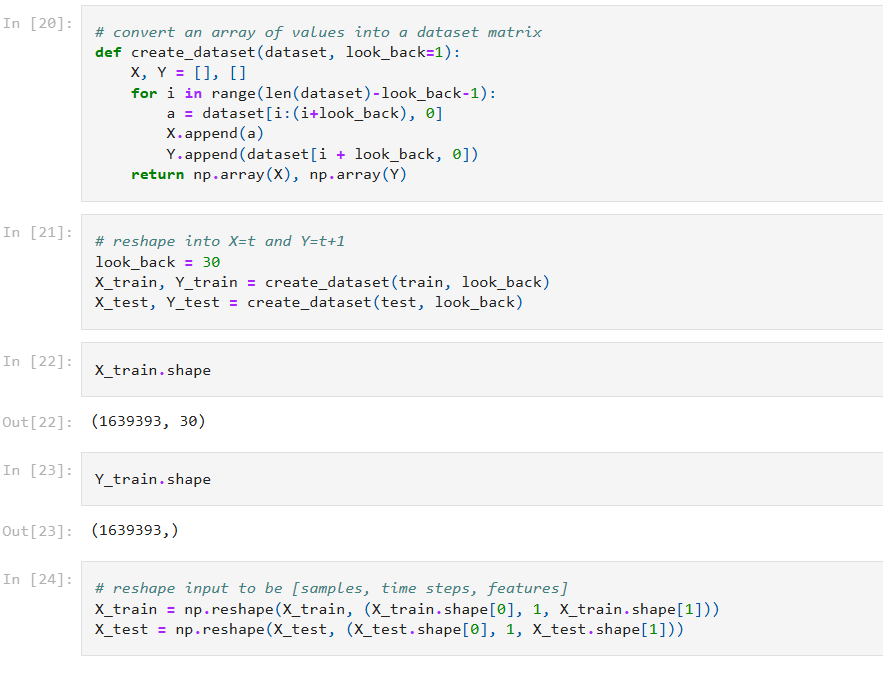
****

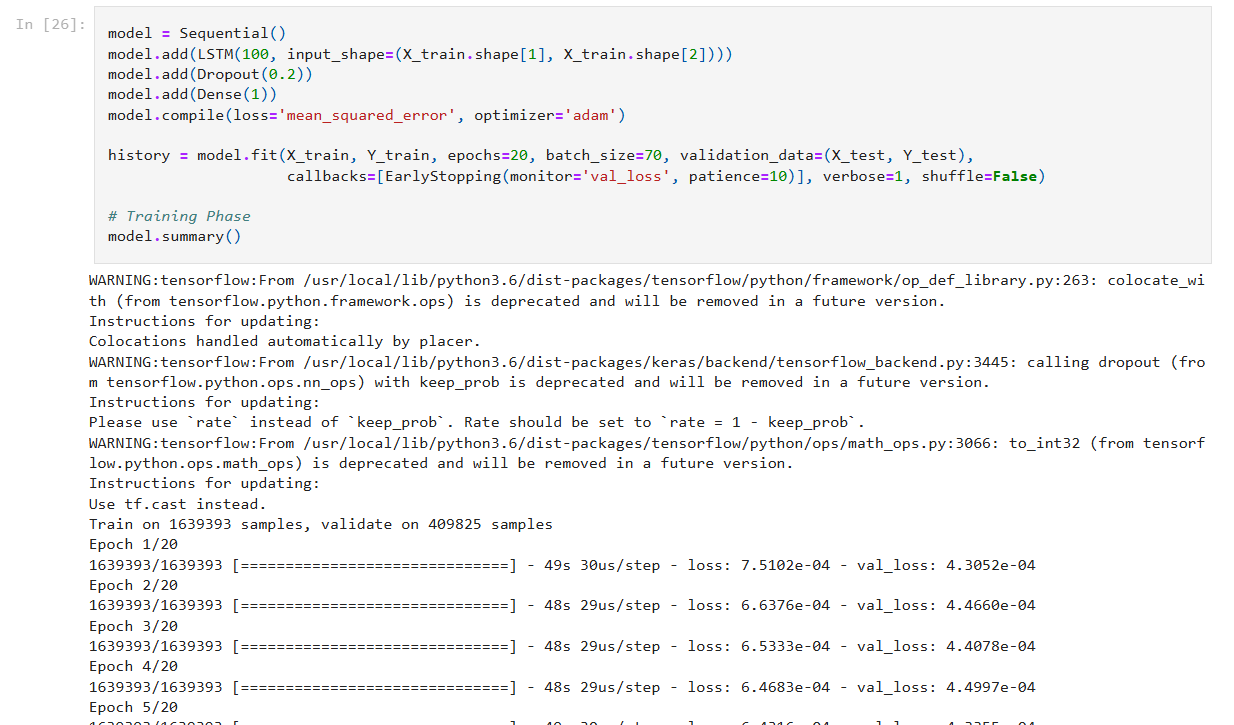
****

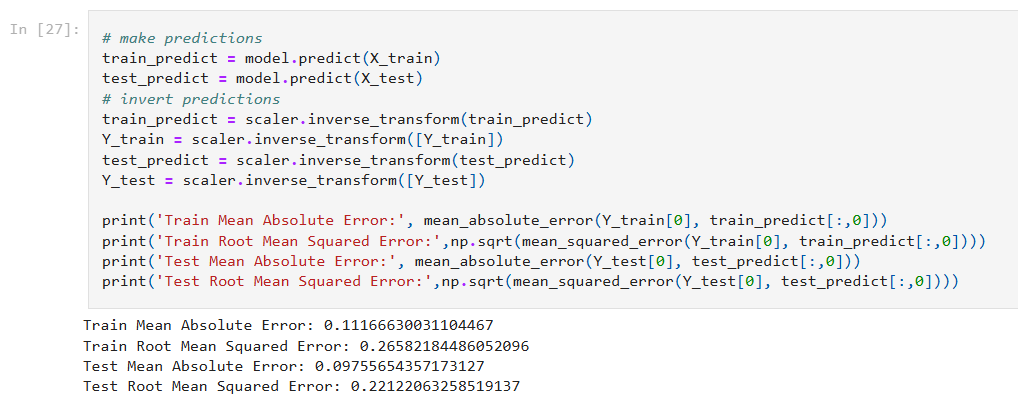
****

****

****

****

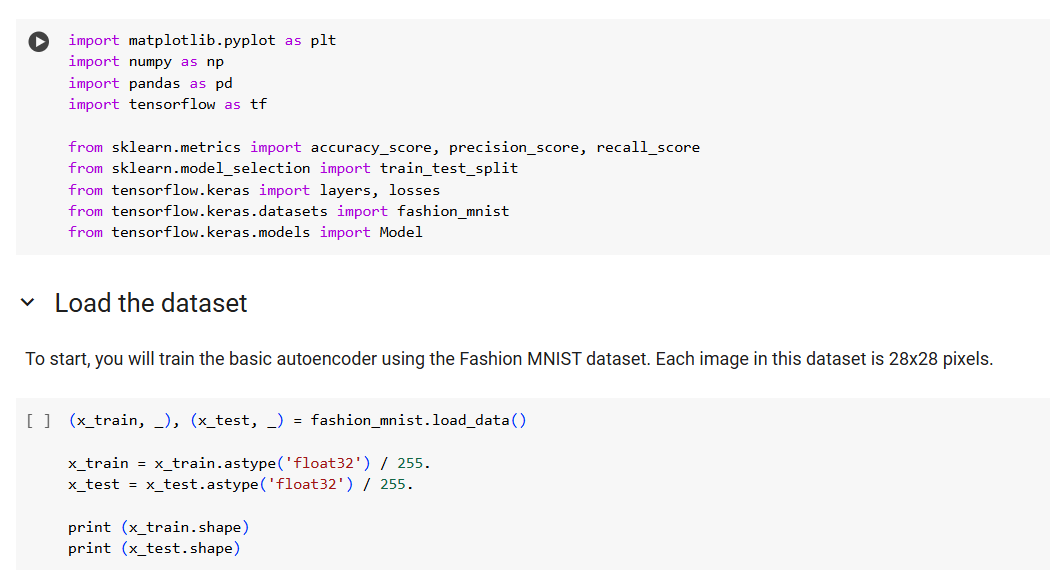
****

****

**Practical - 12**

**Aim :** Autoencoders: Implement autoencoders on an image dataset and evaluate the model's performance.

**Code :**

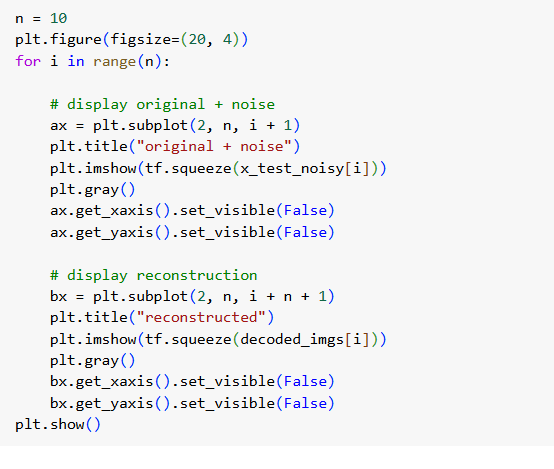
****

****

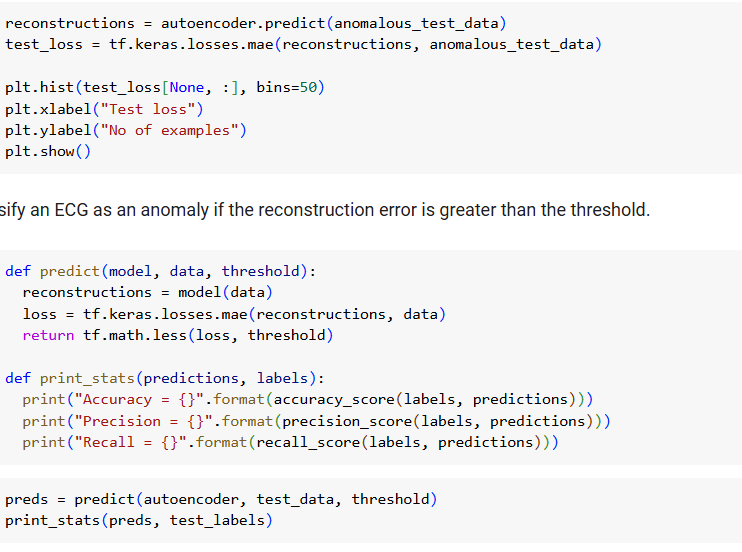
****

****

****

****

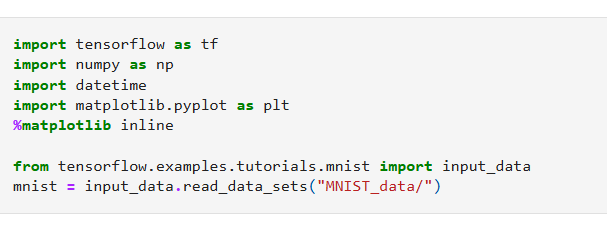
****

****

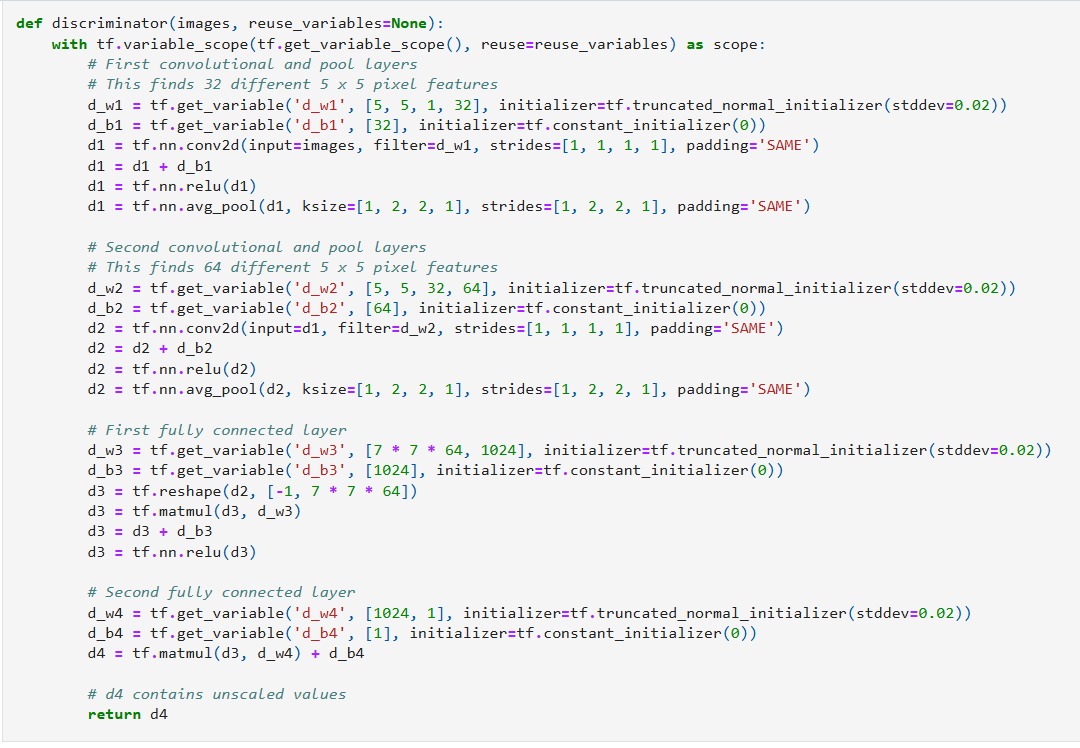
**Practical - 13**

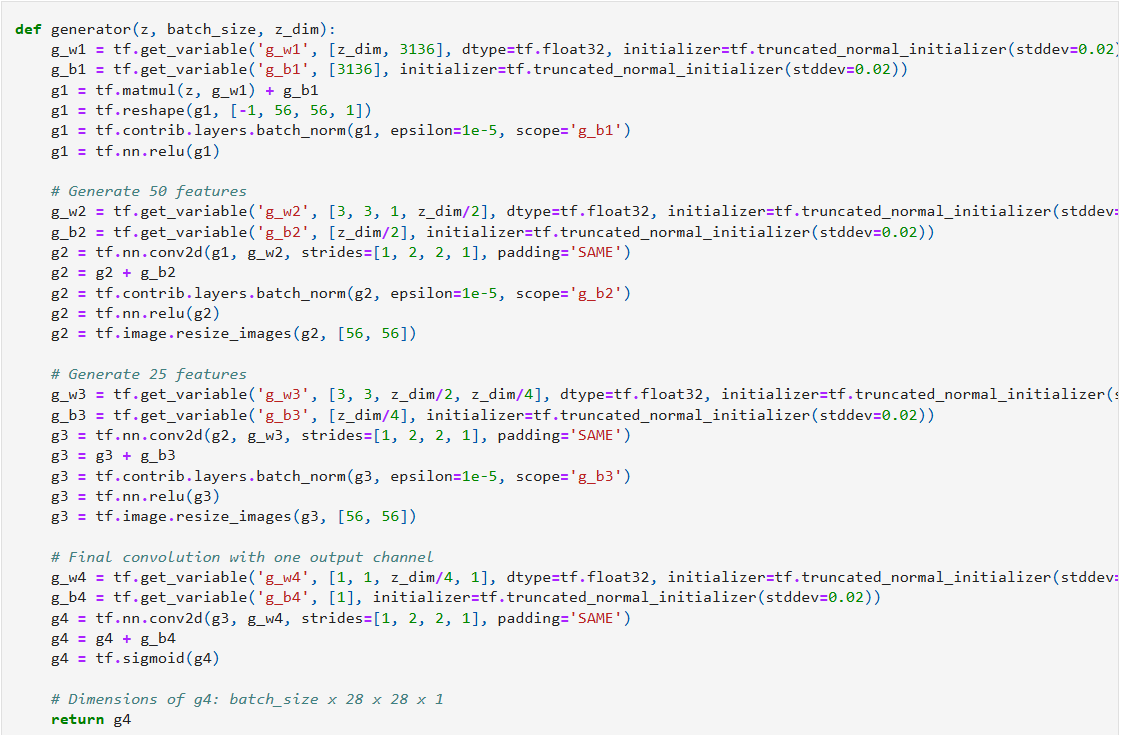
**Aim :** Generative Adversarial Networks (GANs): Implement GANs on an image dataset and evaluate the model's performance.

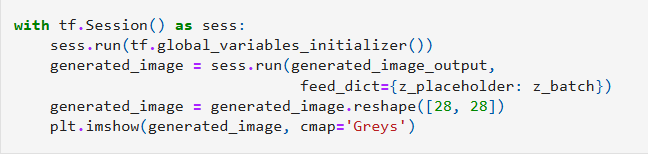
**Code :**

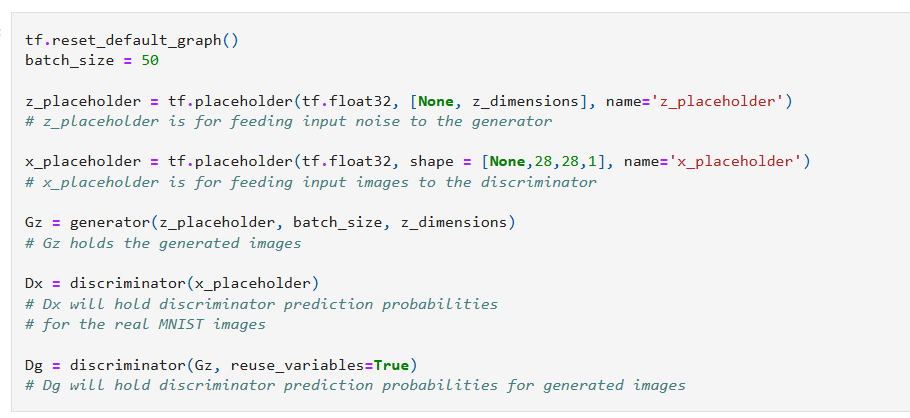
****

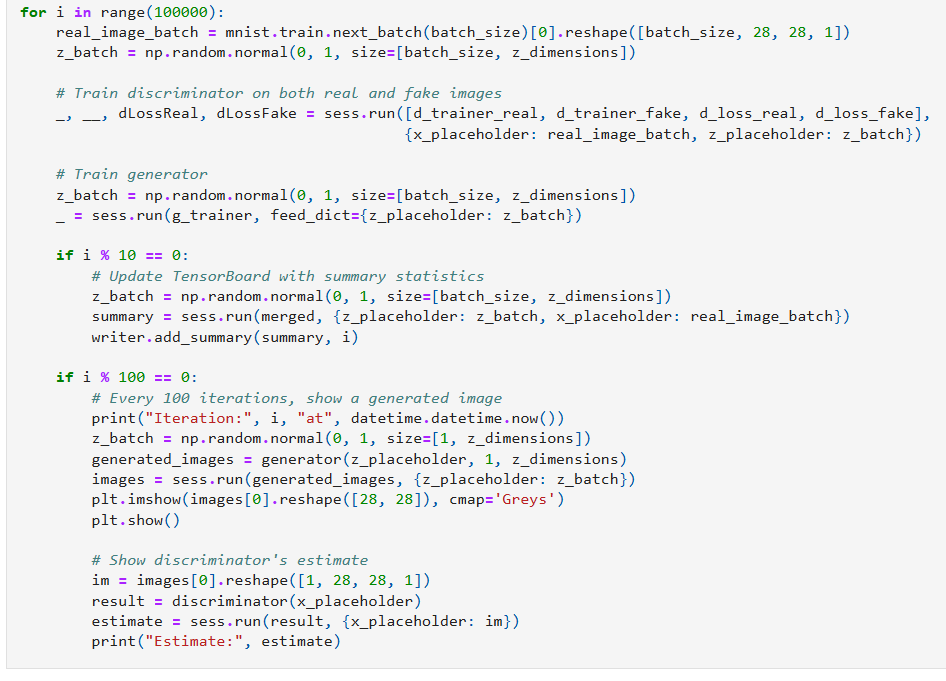
****

****

****

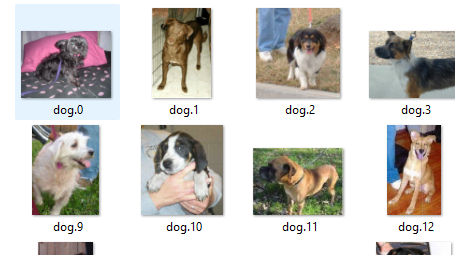
****

****

****

# Practical - 14

**Aim:** Transfer Learning: Implement transfer learning on an image dataset and evaluate the model's performance.  
  
**Dataset Used:**

****

**Code:**

import tensorflow as tf

from tensorflow.keras.applications import ResNet50

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping

import matplotlib.pyplot as plt

import os

# Set paths for the Cats vs. Dogs dataset

base\_dir = r"./cats\_and\_dogs\_filtered"

train\_dir = os.path.join(base\_dir, "train")

validation\_dir = os.path.join(base\_dir, "validation")

print(f"Validation directory: {validation\_dir}")

# Image data generators for preprocessing

train\_datagen = ImageDataGenerator(

    rescale=1./255,

    rotation\_range=20,

    width\_shift\_range=0.2,

    height\_shift\_range=0.2,

    shear\_range=0.2,

    zoom\_range=0.2,

    horizontal\_flip=True,

    fill\_mode="nearest"

)

validation\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_directory(

    train\_dir,

    target\_size=(224, 224),

    batch\_size=32,

    class\_mode='binary'

)

validation\_generator = validation\_datagen.flow\_from\_directory(

    validation\_dir,

    target\_size=(224, 224),

    batch\_size=32,

    class\_mode='binary'

)

# Load the ResNet50 model pre-trained on ImageNet

base\_model = ResNet50(weights="imagenet", include\_top=False, input\_shape=(224, 224, 3))

# Freeze all layers of the base model

base\_model.trainable = False

# Add custom layers on top

model = Sequential([

    base\_model,

    GlobalAveragePooling2D(),

    Dropout(0.5),

    Dense(128, activation='relu'),

    Dropout(0.5),

    Dense(1, activation='sigmoid')

])

# Compile the model

model.compile(

    optimizer=Adam(learning\_rate=0.0001),

    loss='binary\_crossentropy',

    metrics=['accuracy']

)

# Early stopping for better performance

early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)

# Train the model

history = model.fit(

    train\_generator,

    epochs=10,

    validation\_data=validation\_generator,

    callbacks=[early\_stopping]

)

# Evaluate the model

loss, accuracy = model.evaluate(validation\_generator)

print(f"Validation Loss: {loss:.4f}")

print(f"Validation Accuracy: {accuracy:.4f}")

# Plot training and validation accuracy

plt.figure(figsize=(8, 5))

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# Plot training and validation loss

plt.figure(figsize=(8, 5))

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Training and Validation Loss')

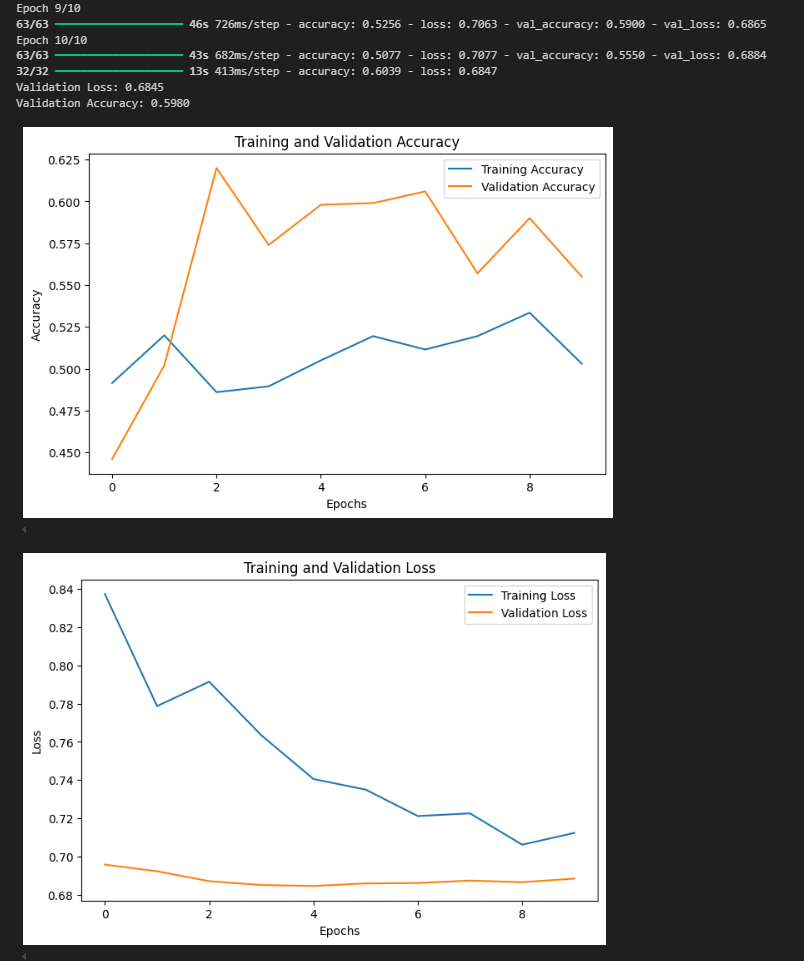
plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

**Output:**



# Practical - 15

**Aim:** Reinforcement Learning: Implement reinforcement learning on a game environment and evaluate the model's performance.

**Code:**

import gym

import numpy as np

import warnings

# Suppress specific deprecation warnings

warnings.filterwarnings("ignore", category=DeprecationWarning)

# Load the environment with render mode specified

env = gym.make('CartPole-v1', render\_mode="human")

# Initialize the environment to get the initial state

state = env.reset()

# Print the state space and action space

print("State space:", env.observation\_space)

print("Action space:", env.action\_space)

for \_ in range(10):

    env.render()

    action = env.action\_space.sample()

    step\_result = env.step(action)

    if len(step\_result) == 4:

        next\_state, reward, done, info = step\_result

        terminated = False

    else:

        next\_state, reward, done, truncated, info = step\_result

        terminated = done or truncated

    print(f"Action: {action}, Reward: {reward}, Next State: {next\_state}, Done: {done}, Info: {info}")

    if terminated:

        state = env.reset()

env.close()

**Output**  
  
