Bharat Intern-

Stock Prediction

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

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

from keras.models import Sequential

from keras.layers import LSTM, Dense

# Load the stock price data

df = pd.read\_csv('bharat\_intern/all\_stocks\_5yr.csv')

# Get the closing prices

data = df['Close'].values.reshape(-1, 1)

# Normalize the data

scaler = MinMaxScaler()

data = scaler.fit\_transform(data)

# Split the data into training and testing sets

train\_size = int(len(data) \* 0.8)

train\_data = data[:train\_size]

test\_data = data[train\_size:]

# Function to create time series dataset

def create\_dataset(data, time\_steps=1):

    X, y = [], []

    for i in range(len(data) - time\_steps):

        X.append(data[i:(i + time\_steps), 0])

        y.append(data[i + time\_steps, 0])

    return np.array(X), np.array(y)

# Set the time steps for LSTM

time\_steps = 60

# Create the time series dataset

X\_train, y\_train = create\_dataset(train\_data, time\_steps)

X\_test, y\_test = create\_dataset(test\_data, time\_steps)

# Reshape the data for LSTM (samples, time steps, features)

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

# Build the LSTM model

model = Sequential()

model.add(LSTM(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)))

model.add(LSTM(units=50))

model.add(Dense(units=1))

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

model.fit(X\_train, y\_train, epochs=10, batch\_size=32)

# Evaluate the model

train\_loss = model.evaluate(X\_train, y\_train)

test\_loss = model.evaluate(X\_test, y\_test)

print("Train Loss:", train\_loss)

print("Test Loss:", test\_loss)

# Make predictions

train\_predictions = model.predict(X\_train)

test\_predictions = model.predict(X\_test)

# Inverse the normalization

train\_predictions = scaler.inverse\_transform(train\_predictions)

y\_train = scaler.inverse\_transform([y\_train])

test\_predictions = scaler.inverse\_transform(test\_predictions)

y\_test = scaler.inverse\_transform([y\_test])

# Plot the results

plt.plot(y\_train.flatten(), label='Actual Train')

plt.plot(train\_predictions.flatten(), label='Predicted Train')

plt.plot(len(y\_train.flatten()) + np.arange(len(y\_test.flatten())), y\_test.flatten(), label='Actual Test')

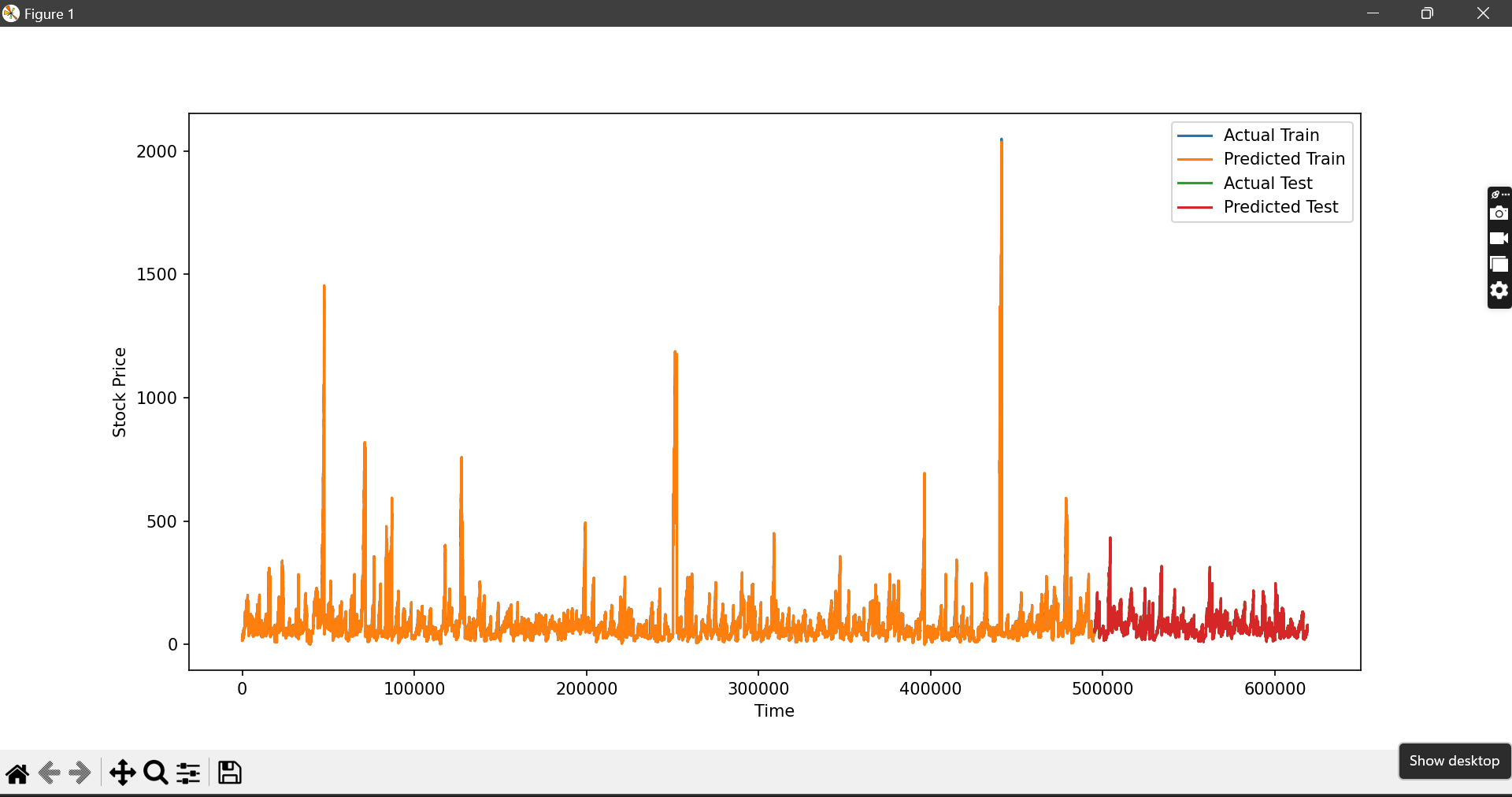
plt.plot(len(y\_train.flatten()) + np.arange(len(y\_test.flatten())), test\_predictions.flatten(), label='Predicted Test')

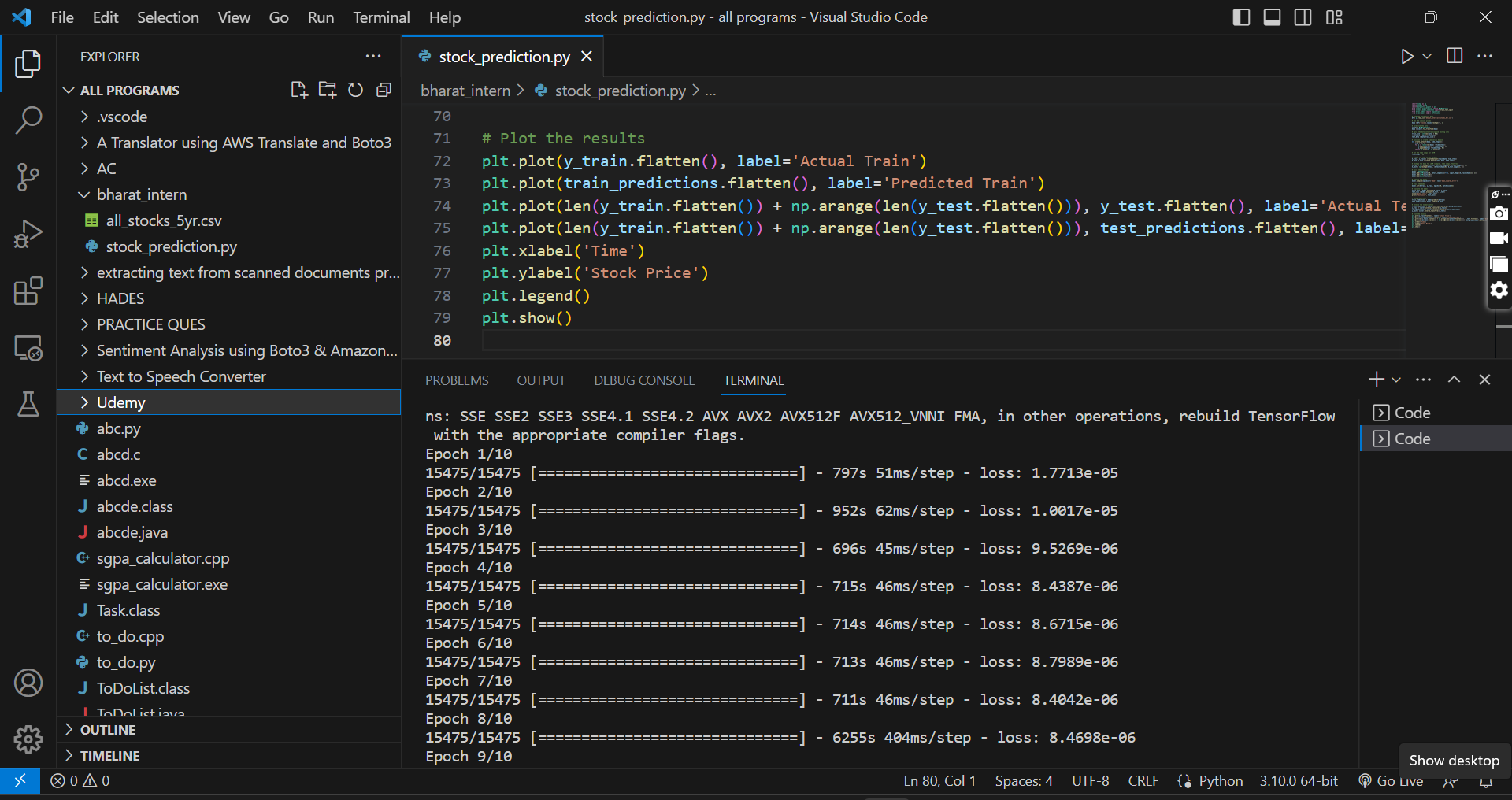
plt.xlabel('Time')

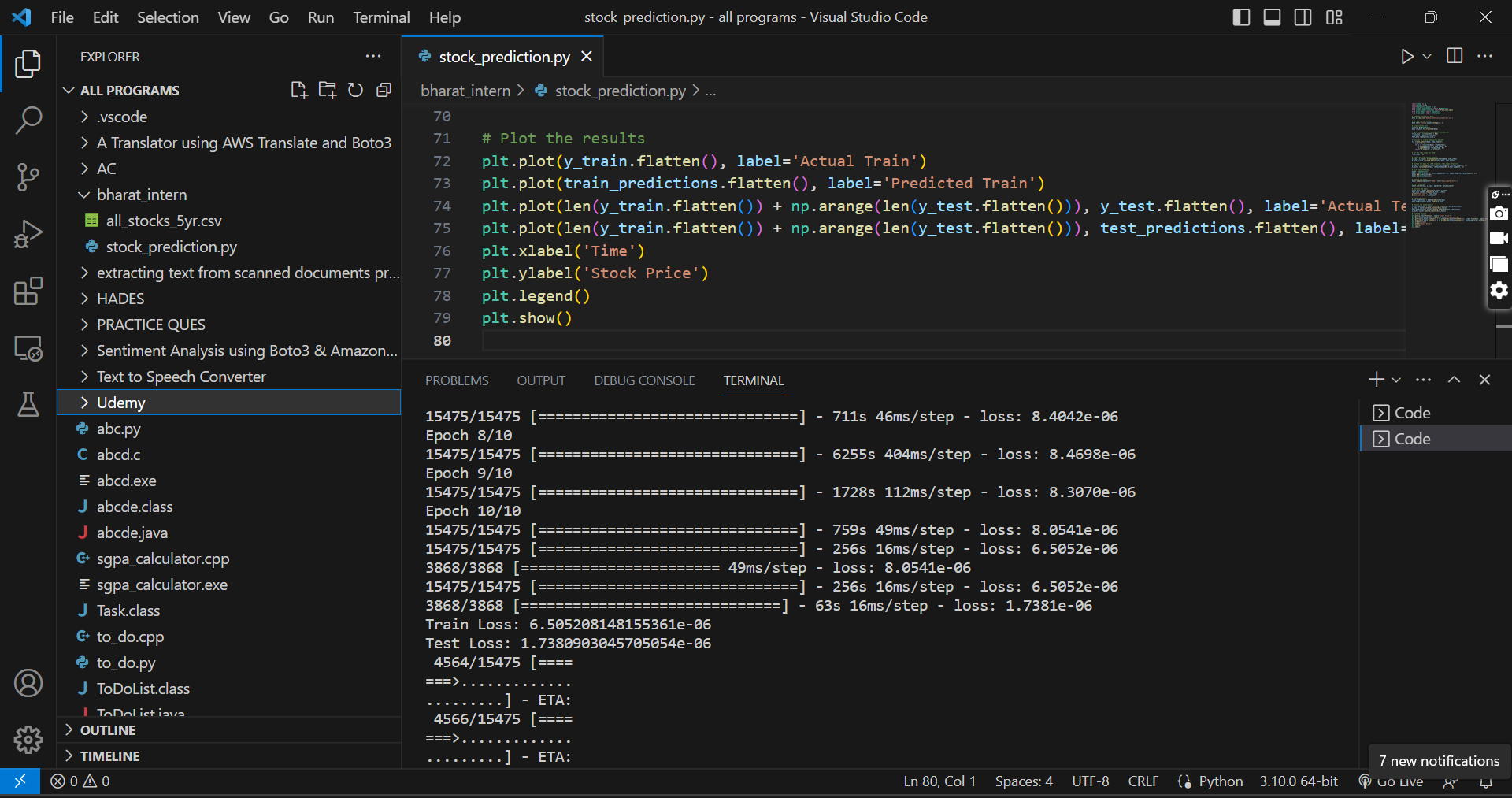
plt.ylabel('Stock Price')

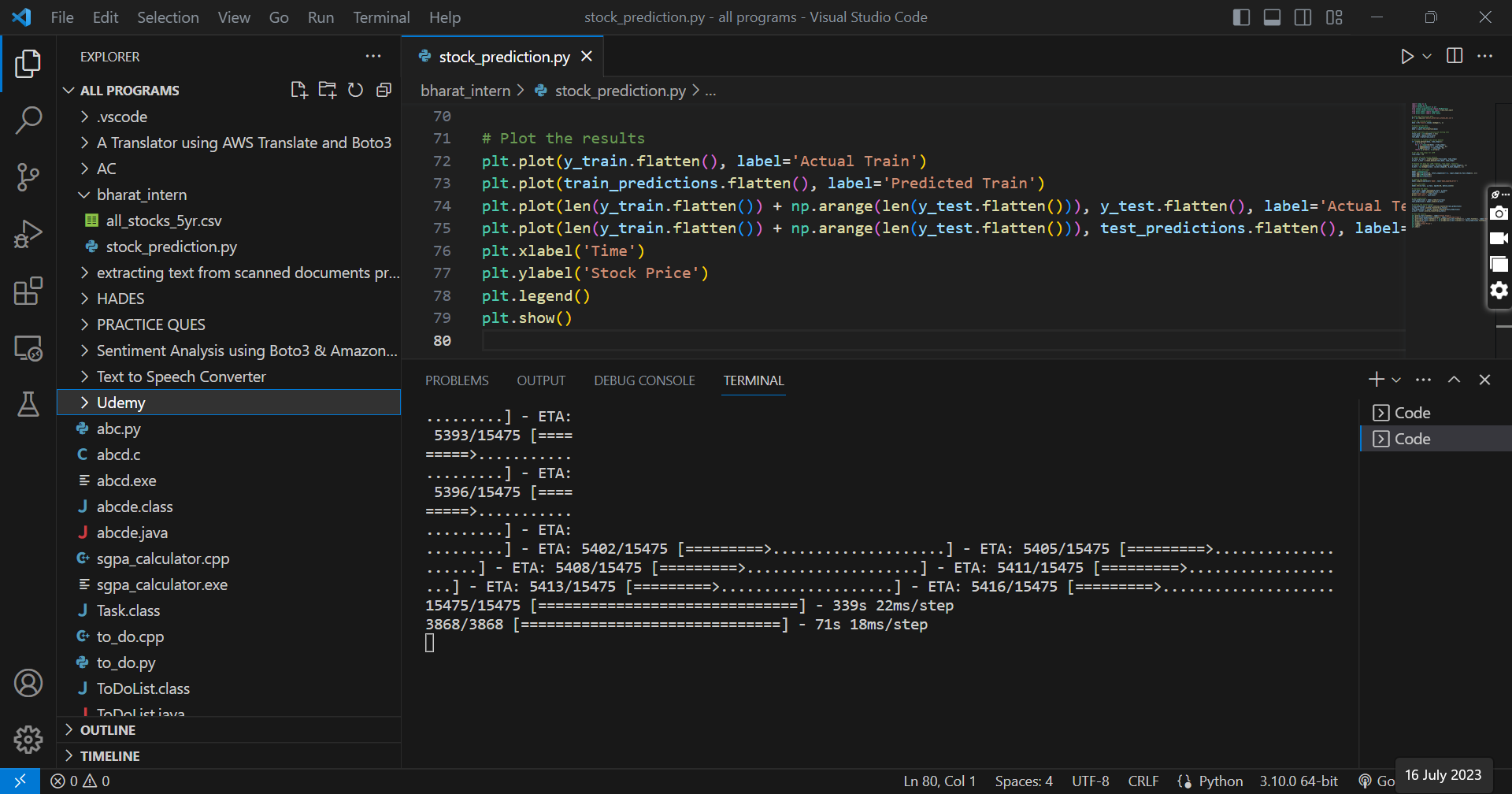
plt.legend()

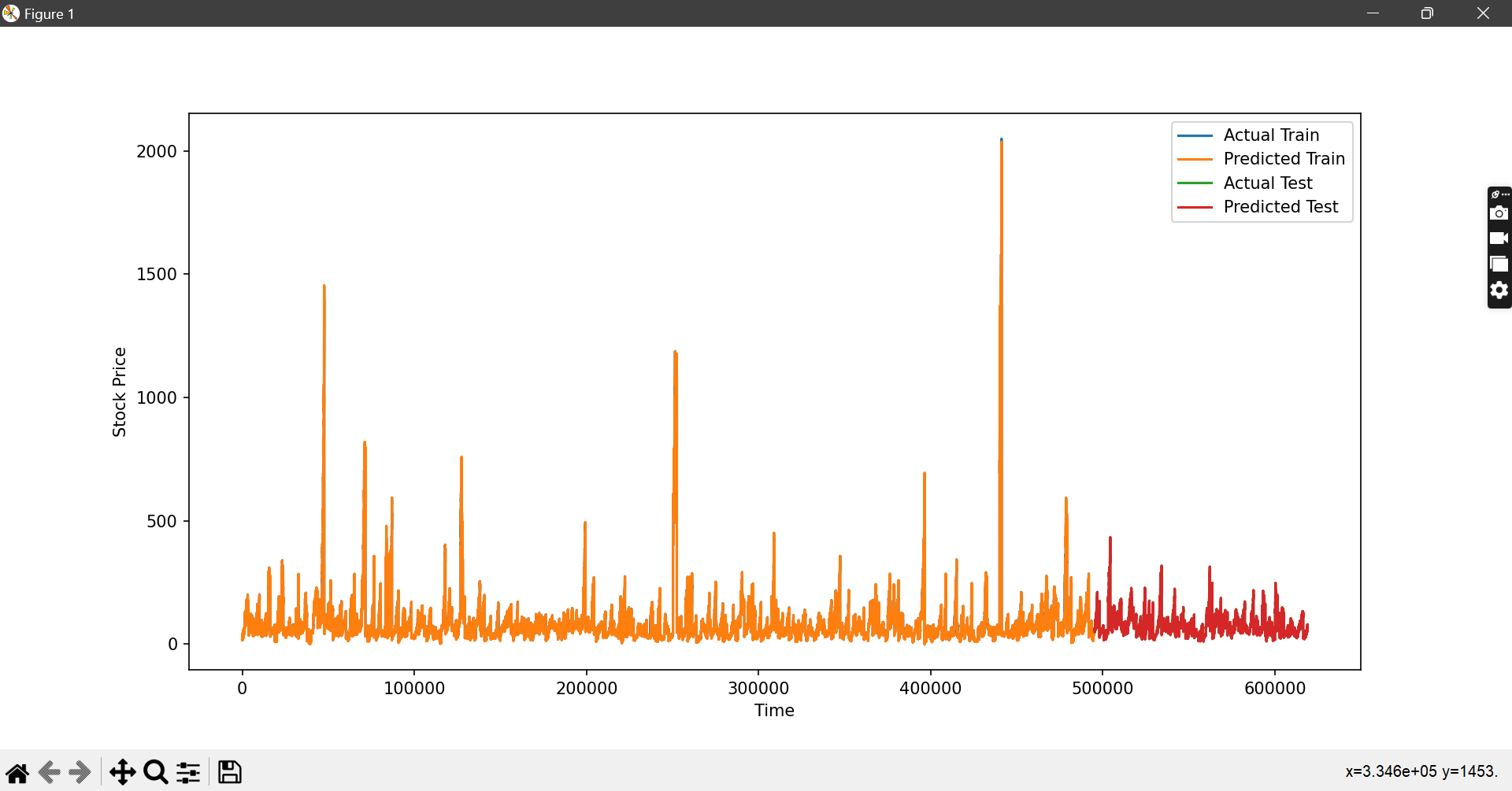
plt.show()











Text Classification-

# Import necessary libraries

import pandas as pd

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

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

# Load the Titanic dataset

data = pd.read\_csv('bharat\_intern/titanic\_train.csv')

# Select relevant features and target variable

features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']

target = 'Survived'

# Preprocess the data

data = data[features + [target]].dropna()  # Remove rows with missing values

data['Sex'] = LabelEncoder().fit\_transform(data['Sex'])  # Convert 'Sex' to numerical

# Encode 'Embarked' column using one-hot encoding

embarked\_encoded = pd.get\_dummies(data['Embarked'], prefix='Embarked')

data = pd.concat([data, embarked\_encoded], axis=1).drop('Embarked', axis=1)

# Split the data into training and testing sets

X = data.drop(target, axis=1)

y = data[target]

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

# Train the model

model = RandomForestClassifier()

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

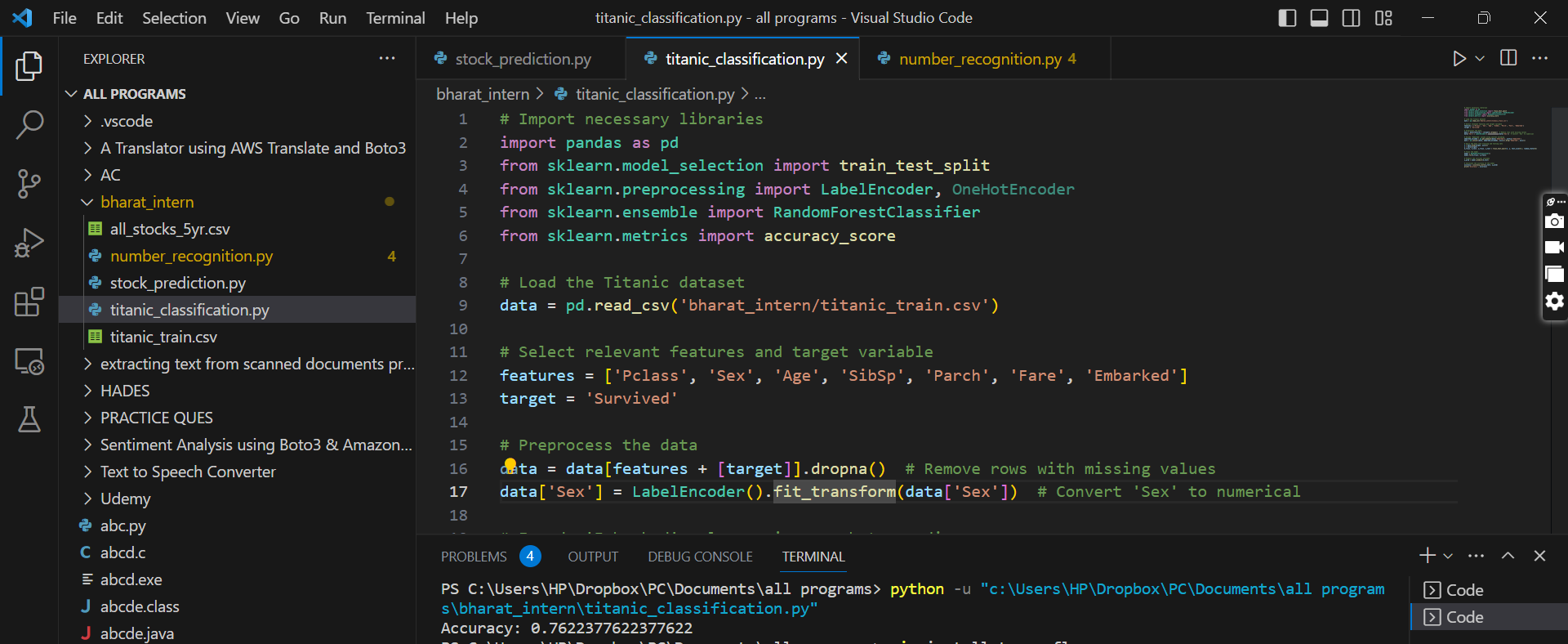
# Predict the survival outcomes

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

# Calculate the accuracy of the model

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

print('Accuracy:', accuracy)



Number Recognition

# Import necessary libraries

import tensorflow as tf

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.utils import to\_categorical

# Load the MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

# Preprocess the data

X\_train = X\_train.reshape((60000, 784))  # Flatten the images (28x28 pixels) into a single vector

X\_test = X\_test.reshape((10000, 784))

X\_train = X\_train.astype('float32') / 255.0  # Normalize pixel values between 0 and 1

X\_test = X\_test.astype('float32') / 255.0

y\_train = to\_categorical(y\_train)  # Convert labels to one-hot encoded vectors

y\_test = to\_categorical(y\_test)

# Build the neural network model

model = Sequential()

model.add(Dense(512, activation='relu', input\_shape=(784,)))

model.add(Dropout(0.2))

model.add(Dense(10, activation='softmax'))

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=10, batch\_size=128, verbose=1)

# Evaluate the model

\_, accuracy = model.evaluate(X\_test, y\_test, verbose=0)

print('Accuracy:', accuracy)

