LABORATORY REPORT

**Application Development Lab (CS33002)**

**B.Tech Program in ECSc**

Submitted By

**Name:-**Sadashray Rastogi

**Roll No:** 2230108



**Kalinga Institute of Industrial Technology (Deemed to be University)**

**Bhubaneswar, India**

Spring 2024-2025

**Table of Content**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Exp No.** | **Title** | **Date of Experiment** | **Date of Submission** | **Remarks** |
| 1. | Build a Resume using HTML/CSS | 07-01-2025 | 13-01-25 |  |
| 2. | Machine Learning and Deep Learning for Cat and Dog Classification | 14-01-2025 | 20-01-2025 |  |
| 3. | Regression Analysis for Stock Prediction | 20-01-2025 | 28-01-2025 |  |
| 4. | Conversational Chatbot with PDF Reader |  |  |  |
| 5. | Web Scraper using LLMs |  |  |  |
| 6. | User Authentication and Document Management System |  |  |  |
| 7. | Natural Language Database Interaction with LLMs |  |  |  |
| 8. | Sentiment Prediction API Using FastAPI and X (formerly Twitter) Tweets |  |  |  |
| 9. | Open Ended 1 |  |  |  |
| 10. | Open Ended 2 |  |  |  |

|  |  |
| --- | --- |
| **Experiment Number** | 3 |
| **Experiment Title** | Regression Analysis for Stock Prediction |
| **Date of Experiment** | 20-01-2025 |
| **Date of Submission** | 28-01-2025 |

## Objective:-

The objective of regression analysis for stock prediction is to model the relationship between historical stock data and future prices to predict stock trends and make informed financial decisions.

## Procedure:-

* 1. Obtain historical stock data.

2. Clean, scale, and prepare features.

3. Create relevant indicators (e.g., moving averages).

4. Choose an appropriate regression model (e.g., Linear, LSTM).

5.Train on historical data and validate performance.

6.Test with metrics (e.g., RMSE, R²) and refine.

7.Make predictions and deploy the model in an app.

## Code:- Index.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Stock Price Prediction</title>

    <link rel="stylesheet" href="/static/style.css">

</head>

<body>

    <h1>Stock Price Prediction</h1>

    <form id="prediction-form">

        <label for="data">Enter Data Points (comma-separated):</label>

        <input type="text" id="data" placeholder="e.g., 1.2, 3.4, 5.6" required>

        <button type="submit">Predict</button>

    </form>

    <div id="results" style="display:none;">

        <h2>Predictions</h2>

        <p><strong>Linear Regression:</strong> <span id="lr-result"></span></p>

        <p><strong>LSTM:</strong> <span id="lstm-result"></span></p>

    </div>

    <script>

        document.getElementById('prediction-form').addEventListener('submit', async function(*event*) {

            event.preventDefault();

            const dataInput = document.getElementById('data').value;

            if (!dataInput.trim()) {

                alert('Please enter some data points.');

                return;

            }

            const data = dataInput.split(',').map(Number);

            try {

                const response = await fetch('/predict', {

                    method: 'POST',

                    headers: {

                        'Content-Type': 'application/json',

                    },

                    body: JSON.stringify({ data }),

                });

                if (!response.ok) {

                    throw new Error(`Error: ${response.statusText}`);

                }

                const result = await response.json();

                document.getElementById('lr-result').textContent = result.linear\_regression\_prediction.join(', ');

                document.getElementById('lstm-result').textContent = result.lstm\_prediction.join(', ');

                document.getElementById('results').style.display = 'block';

            } catch (error) {

                alert(`Failed to fetch predictions: ${error.message}`);

            }

        });

    </script>

</body>

</html>

# CSS:-

# body {

# font-family: Arial, sans-serif;

# margin: 20px;

# line-height: 1.6;

# }

# h1 {

# color: #333;

# text-align: center;

# }

# form {

# margin-bottom: 20px;

# display: flex;

# flex-direction: column;

# align-items: center;

# }

# label {

# font-weight: bold;

# margin-bottom: 10px;

# }

# input[type="text"] {

# padding: 10px;

# width: 80%;

# max-width: 400px;

# margin-bottom: 20px;

# border: 1px solid #ccc;

# border-radius: 5px;

# }

# button {

# padding: 10px 20px;

# background-color: #007BFF;

# color: white;

# border: none;

# border-radius: 5px;

# cursor: pointer;

# transition: background-color 0.3s ease;

# }

# button:hover {

# background-color: #0056b3;

# }

# button:focus {

# outline: 2px solid #0056b3;

# outline-offset: 2px;

# }

# p {

# font-size: 18px;

# text-align: center;

# }

# @media (max-width: 768px) {

# input[type="text"] {

# width: 90%;

# }

# button {

# width: 100%;

# }

# }

# Models code:

# # %% [markdown]

# # # Stock Price Prediction

# # %%

# import numpy as np

# import pandas as pd

# # %%

# file\_path = r"C:\Users\KIIT\Documents\Programs\college assignment\LAB3\datasets\TESLA.csv"

# data = pd.read\_csv(file\_path)

# # %%

# data = data[['Date', 'Close']] # Select only Date and Close columns

# data['Date'] = pd.to\_datetime(data['Date'])

# data.set\_index('Date', inplace=True)

# # %%

# data

# # %%

# import matplotlib.pyplot as plt

# plt.figure(figsize=(12, 6))

# plt.plot(data['Close'], label='Tesla Stock Closing Price', color='blue')

# plt.title('Tesla Stock Price Over Time')

# plt.xlabel('Date')

# plt.ylabel('Close Price (USD)')

# plt.legend()

# plt.show()

# # %% [markdown]

# # ### Why MinMaxScaler Is Preferred for Stock Prices

# # -Stock Prices Are Not Gaussian Distributed:

# #

# # --StandardScaler assumes the data is normally distributed (bell-shaped curve). However, stock prices often exhibit non-Gaussian distributions due to market fluctuations, trends, and other anomalies.

# # Applying StandardScaler to non-Gaussian data can lead to skewed results.

# # Time Series Models (e.g., LSTM) Are Sensitive to Scale:

# #

# # --LSTM models work better when input features are in a consistent and bounded range, such as [0, 1].

# # StandardScaler does not guarantee a bounded range, and scaled values might exceed the range of the activation functions used in the model (e.g., sigmoid or tanh).

# # Preservation of Data Shape:

# #

# # --MinMaxScaler preserves the original shape and trends of the data, which is crucial for time-series prediction. Stock prices exhibit relative changes and patterns that must remain intact for accurate modeling.

# # StandardScaler alters the distribution by centering data around the mean, potentially distorting the patterns.

# # %%

# from sklearn.preprocessing import MinMaxScaler

# scaler = MinMaxScaler(feature\_range=(0, 1))

# scaled\_data = scaler.fit\_transform(data[['Close']])

# # %%

# scaled\_data

# # %%

# joblib.dump(scaler, 'scaler.pkl')

# # %% [markdown]

# # ## When Can train\_test\_split Be Used?

# # When Data Is Stationary and Uncorrelated:

# #

# # If the time-series data is stationary and has no autocorrelation, train\_test\_split might yield reasonable results. However, most financial time-series data (e.g., stock prices) are non-stationary.

# # Feature Engineering with Lagged Variables:

# #

# # After creating lagged features, the dataset can sometimes behave like standard supervised learning data. In such cases, random splits might be acceptable.

# # When Forecasting Is Not the Goal:

# #

# # If the task is classification or regression on static features derived from the time-series (e.g., predicting stock direction based on lagged indicators), train\_test\_split might work.

# # %%

# ## 5. Create Sequences for Training and Testing

# def create\_sequence(data, sequence\_length):

# X , y = [] , []

# for i in range(sequence\_length, len(data)):

# X.append(data[i-sequence\_length:i,0])

# y.append(data[i,0])

# return np.array(X), np.array(y)

# # %%

# sequence\_length = 60

# X, y = create\_sequence(scaled\_data, sequence\_length)

# # %% [markdown]

# # #### Key Points:

# # -> sequence\_length: Number of previous time steps used to predict the next time step.

# #

# # -> X: Contains sequences of the past sequence\_length values.

# #

# # -> y: Contains the target value (the next value after each sequence).

# #

# # Example: If sequence\_length = 3 and data = [1, 2, 3, 4], then X = [[1, 2, 3]] and y = [4].

# # %%

# ## 6. Split Data into Training and Testing Sets

# train\_size = int(len(X) \* 0.8)

# X\_train, X\_test = X[:train\_size], X[train\_size:]

# y\_train, y\_test = y[:train\_size], y[train\_size:]

# # %% [markdown]

# # reshape(): Flattens the sequences for linear regression, which requires input in a 2D format.

# # %%

# ## 7. Prepare Data for Linear Regression

# X\_train\_lr = X\_train.reshape(X\_train.shape[0], -1)

# X\_test\_lr = X\_test.reshape(X\_test.shape[0], -1)

# # %%

# ## 8. Linear Regression Model

# from sklearn.linear\_model import LinearRegression

# linear\_regression = LinearRegression()

# linear\_regression.fit(X\_train\_lr, y\_train)

# # %%

# lr\_pred = linear\_regression.predict(X\_test\_lr)

# # %% [markdown]

# # reshape(): Converts the sequences into a 3D format required by LSTM layers (samples, timesteps, features).

# # %%

# ## 9. Prepare Data for LSTM

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

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

# # %%

# ## 10. LSTM Model

# from tensorflow.keras.models import Sequential

# from tensorflow.keras.layers import LSTM, Dense, Dropout

# lstm\_model = Sequential([

# LSTM(50, return\_sequences=True, input\_shape=(X\_train\_lstm.shape[1], 1)),

# Dropout(0.2),

# LSTM(50, return\_sequences=False),

# Dropout(0.2),

# Dense(25),

# Dense(1)

# ])

# # %%

# from sklearn.metrics import mean\_squared\_error

# # %%

# lstm\_model.compile(optimizer='adam', loss='mean\_squared\_error')

# lstm\_model.fit(X\_train\_lstm, y\_train,epochs=20, batch\_size=32, validation\_data=(X\_test\_lstm, y\_test))

# # %%

# ## 11. Prediction

# lstm\_predictions = lstm\_model.predict(X\_test\_lstm)

# # %%

# lstm\_predictions

# # %%

# ## 12. Reverse Scaling

# lr\_pred = scaler.inverse\_transform(lr\_pred.reshape(-1, 1))

# lstm\_predictions = scaler.inverse\_transform(lstm\_predictions)

# actual\_prices = scaler.inverse\_transform(y\_test.reshape(-1, 1))

# # %% [markdown]

# # inverse\_transform: Converts scaled predictions and targets back to their original scale for evaluation and visualization.

# # %%

# ## 13. Evaluate Models

# lr\_rmse = np.sqrt(mean\_squared\_error(actual\_prices, lr\_pred))

# lstm\_rmse = np.sqrt(mean\_squared\_error(actual\_prices, lstm\_predictions))

# # %% [markdown]

# # mean\_squared\_error: Calculates the mean squared error.

# #

# # np.sqrt: Computes the root mean squared error (RMSE) for easier interpretation.

# # %%

# ## 14. Visualize Results

# plt.figure(figsize=(14, 7))

# plt.plot(actual\_prices, label='Actual Prices', color='blue')

# plt.plot(lr\_pred, label='Linear Regression Predictions', color='green')

# plt.plot(lstm\_predictions, label='LSTM Predictions', color='red')

# plt.title('Tesla Stock Price Prediction')

# plt.xlabel('Days')

# plt.ylabel('Stock Price (USD)')

# plt.legend()

# plt.show()

# # %%

# import joblib

# # Save the Linear Regression model

# joblib.dump(linear\_regression, 'linear\_regression\_model.pkl')

# # %%

# # Save the LSTM model

# lstm\_model.save('lstm\_model.h5')

## App:-

from flask import Flask, render\_template, request, jsonify

import numpy as np

import joblib

from tensorflow.keras.models import load\_model

from sklearn.preprocessing import MinMaxScaler

app = Flask(\_\_name\_\_)

*# Load the models and scaler*

linear\_regression\_model = joblib.load("linear\_regression\_model.pkl")

lstm\_model = load\_model("lstm\_model.h5")

scaler = joblib.load("scaler.pkl")  *# Pre-fit scaler for consistent transformations*

@app.route("/")

def home():

    return render\_template("index.html")

import traceback  *# Add this at the top of your app.py file*

@app.route("/predict", *methods*=["POST"])

def predict():

    try:

        data = request.json.get("data", [])

        if not data:

            return jsonify({"error": "No data provided"}), 400

*# Ensure the input data is valid*

        try:

            data = np.array(data).reshape(-1, 1)

        except ValueError:

            return (

                jsonify({"error": "Invalid data format. Provide numeric values."}),

                400,

            )

*# Scale the input data*

        scaled\_data = scaler.transform(data)

*# Prepare data for predictions*

        if len(scaled\_data) < 60:

*# Pad with zeros if less than 60 features*

            padding = np.zeros((60 - len(scaled\_data), 1))

            X\_lr = np.vstack((padding, scaled\_data)).reshape(1, -1)

        else:

*# Take only the last 60 features*

            X\_lr = scaled\_data[-60:].reshape(1, -1)

*# Prepare data for LSTM*

        X\_lstm = scaled\_data[-60:].reshape(1, -1, 1)

*# Predict using models*

        lr\_pred = linear\_regression\_model.predict(X\_lr)

        lstm\_pred = lstm\_model.predict(X\_lstm)

*# Inverse transform predictions*

        lr\_pred = scaler.inverse\_transform(lr\_pred.reshape(-1, 1)).flatten()

        lstm\_pred = scaler.inverse\_transform(lstm\_pred).flatten()

        return jsonify(

            {

                "linear\_regression\_prediction": lr\_pred.tolist(),

                "lstm\_prediction": lstm\_pred.tolist(),

            }

        )

    except Exception as e:

        error\_message = traceback.format\_exc()

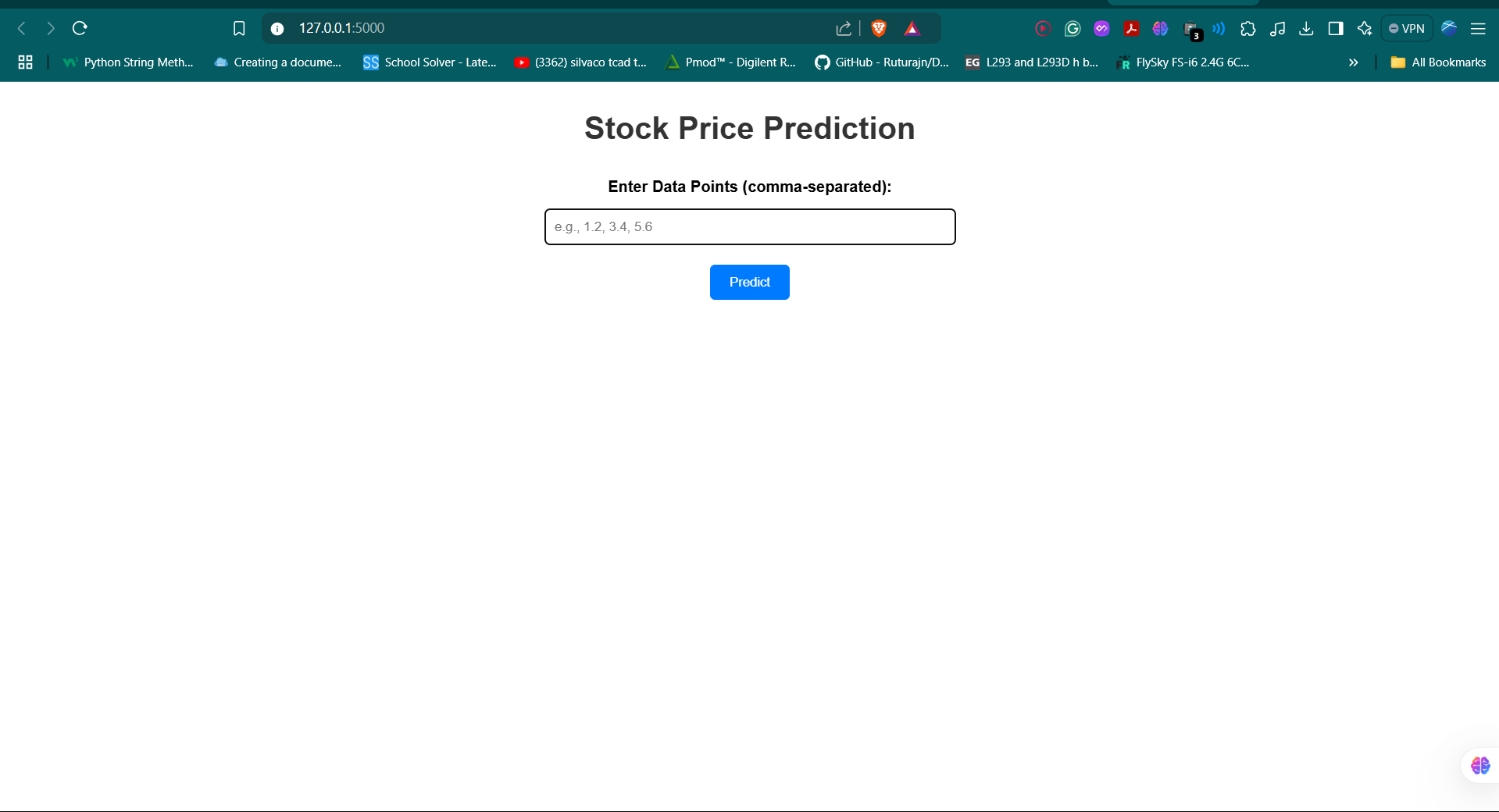
        print(error\_message)

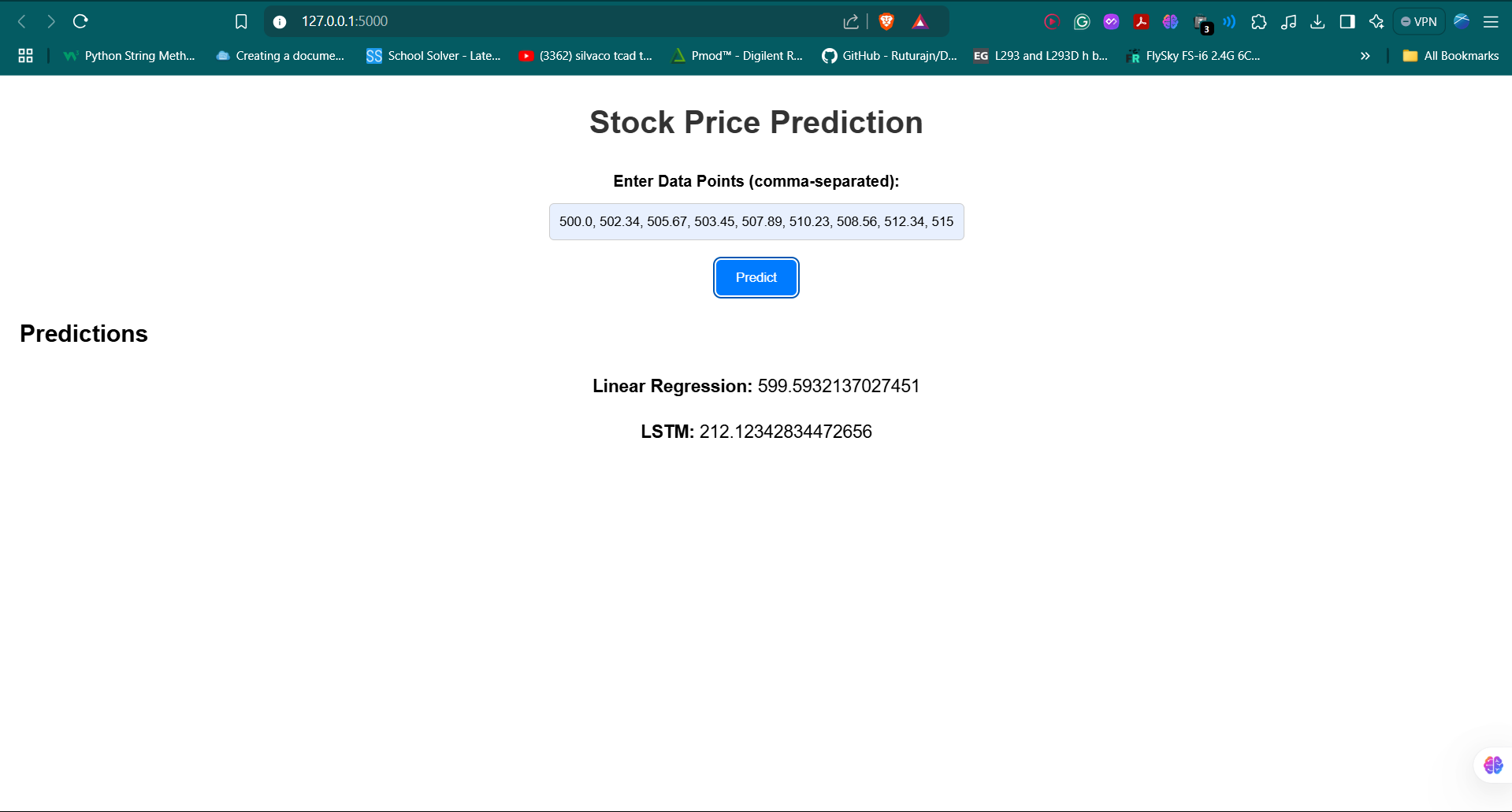
        return jsonify({"error": str(e)}), 500

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(*debug*=True)

## Results/Output:-

****

****

1. **Remarks:-**

Signature of the Student Signature of the Lab Coordinator

Sadashray Rastogi Bhargav Apassani