LABORATORY REPORT

Application Development Lab
(CS33002)

B.Tech Program in ECSc

Submitted By

Name:-Anubrata Mukhopadhyay

Roll No: 2230311

Kalinga Institute of Industrial Technology
(Deemed to be University)
Bhubaneswar, India

Spring 2024-2025

Table of Content

Experiment Number	1
Experiment Title	Stock price prediction using Linear Regression and LSTM models
Date of Experiment	21/01/25
Date of Submission	28/01/25

1. Objective:- To perform stock price prediction based on historical stock price data, using linear regression and LSTM (Long short-term model)

2. Procedure:- 1. Collect historical stock price data.

2. Preprocess the data for analysis (missing data, scaling, splitting into train/test).

3. Implement Linear Regression to predict future stock prices.

4. Design and train an LSTM model for time-series prediction.

5. Compare the accuracy of both models.

6. Create a Flask backend for model predictions.

7. Build a frontend to visualize predictions using charts and graphs.

3. Code:-

```
import numpy as np
import pandas as pd
from flask import Flask, request, jsonify, render_template
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
import os
from werkzeug.utils import secure_filename

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads'
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024  # 16MB max
file size
```

```
# Create uploads folder if it doesn't exist
os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

# Initialize global variables
linear_model = LinearRegression()
lstm_model = None
scaler = StandardScaler()
mm_scaler = MinMaxScaler()
ALLOWED_EXTENSIONS = {'csv'}

def allowed_file(filename):
        return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS

def prepare_data_linear(df):
    """Prepare data for Linear Regression"""
    # Create features
    df['SMA_5'] = df['Close'].rolling(window=5).mean()
    df['SMA_20'] = df['Close'].rolling(window=20).mean()
    df['RSI'] = calculate_rsi(df['Close'])
    df['Price_Change'] = df['Close'].pct_change()
    df['Volatility'] = df['Close'].rolling(window=10).std()

    # Drop missing values
    df.dropna(inplace=True)

    # Create features and target
    X = df[['SMA_5', 'SMA_20', 'RSI', 'Price_Change', 'Volatility']].values
    y = df['Close'].values

    return X, y

def prepare_data_lstm(df, look_back=30):
    """Prepare data for LSTM"""
    # Ensure enough data points for look_back
    if len(df) < look_back:
        look_back = len(df) // 2

    # Scale the data
    scaled_data = mm_scaler.fit_transform(df[['Close']].values)

    X, y = [], []
    for i in range(look_back, len(scaled_data)):
        X.append(scaled_data[i-look_back:i, 0])
        y.append(scaled_data[i, 0])
```

```
    X, y = np.array(X), np.array(y)
    X = np.reshape(X, (X.shape[0], X.shape[1], 1))

    return X, y

def calculate_rsi(prices, period=14):
    """Calculate RSI indicator"""
    delta = prices.diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=period).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=period).mean()
    rs = gain / loss
    return 100 - (100 / (1 + rs))

def create_lstm_model(input_shape):
    """Create and compile LSTM model"""
    model = Sequential([
                        LSTM(50, activation='relu', input_shape=input_shape,
return_sequences=True),
        Dropout(0.2),
        LSTM(50, activation='relu'),
        Dropout(0.2),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
    return model

def train_linear_model(X, y):
    """Train the linear regression model"""
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    linear_model.fit(X_train_scaled, y_train)
    return linear_model.score(X_test_scaled, y_test)

def train_lstm_model(X, y):
    """Train the LSTM model"""
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
    global lstm_model
    lstm_model = create_lstm_model((X.shape[1], 1))
                lstm_model.fit(X_train, y_train, epochs=50, batch_size=32,
validation_split=0.1, verbose=0)
    return lstm_model.evaluate(X_test, y_test)

@app.route('/')
```

```
def home():
    return render_template('index.html')

@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return jsonify({'error': 'No file part'}), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'No selected file'}), 400

    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(filepath)
        return jsonify({'success': True, 'filename': filename})

    return jsonify({'error': 'Invalid file type'}), 400

@app.route('/predict', methods=['POST'])
def predict():
    try:
        data = request.get_json()
        filename = data.get('filename')
        model_type = data.get('model_type', 'linear')

        # Load and prepare data
                df  =  pd.read_csv(os.path.join(app.config['UPLOAD_FOLDER'],
filename), parse_dates=['Date'])
        df.set_index('Date', inplace=True)

        if model_type == 'linear':
            # Linear Regression prediction
            X, y = prepare_data_linear(df)
            accuracy = train_linear_model(X, y)
            last_data = scaler.transform([X[-1]])
            prediction = linear_model.predict(last_data)[0]
        else:
            # LSTM prediction
            X, y = prepare_data_lstm(df)
            accuracy = train_lstm_model(X, y)

            # Predict next point
            last_sequence = X[-1]  # Last sequence in the training data
            last_sequence = np.expand_dims(last_sequence, axis=0)
```

```
            prediction = lstm_model.predict(last_sequence)
                    prediction = mm_scaler.inverse_transform(prediction.reshape(-1,
1))[0][0]

        response = {
            'prediction': float(prediction),
                        'accuracy': float(1 - accuracy) if model_type == 'lstm' else
float(accuracy),
            'historical_data': df['Close'].tolist(),
            'dates': df.index.strftime('%Y-%m-%d').tolist()
        }

        return jsonify(response)

    except Exception as e:
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Advanced Stock Price Prediction</title>
                                                                        <script
src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/3.7.0/chart.min.js"></script
>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 1200px;
            margin: 0 auto;
            padding: 20px;
            background-color: #f5f5f5;
        }

        .container {
            background-color: white;
```

```
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.header {
    text-align: center;
    margin-bottom: 30px;
}

.controls {
    display: flex;
    justify-content: center;
    gap: 20px;
    margin-bottom: 20px;
    align-items: center;
}

.file-upload {
    display: flex;
    align-items: center;
    gap: 10px;
}

.model-select {
    padding: 8px;
    border-radius: 4px;
    border: 1px solid #ddd;
    font-size: 16px;
}

button {
    padding: 8px 16px;
    background-color: #007bff;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    font-size: 16px;
}

button:hover {
    background-color: #0056b3;
}

button:disabled {
```

```
            background-color: #cccccc;
            cursor: not-allowed;
        }

        .results {
            margin-top: 20px;
            padding: 20px;
            border: 1px solid #ddd;
            border-radius: 4px;
            display: none;
        }

        .chart-container {
            margin-top: 20px;
            height: 400px;
        }

        .loading {
            display: none;
            text-align: center;
            margin: 20px 0;
        }

        .error {
            color: red;
            text-align: center;
            margin: 20px 0;
            display: none;
        }

        .upload-status {
            margin-top: 10px;
            text-align: center;
            color: #666;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h1>Advanced Stock Price Prediction</h1>
            <p>Upload CSV data and choose prediction model</p>
        </div>

        <div class="controls">
            <div class="file-upload">
```

```
            <input type="file" id="csvFile" accept=".csv">
            <button onclick="uploadFile()" id="uploadBtn">Upload</button>
        </div>
        <select class="model-select" id="modelSelect">
            <option value="linear">Linear Regression</option>
            <option value="lstm">LSTM</option>
        </select>
                <button onclick="predict()" id="predictBtn" disabled>Generate
Prediction</button>
    </div>

    <div class="upload-status" id="uploadStatus"></div>
    <div class="loading" id="loading">Processing... Please wait.</div>
    <div class="error" id="error"></div>

    <div class="results" id="results">
        <h2>Prediction Results</h2>
        <p>Predicted Price: $<span id="predictedPrice"></span></p>
        <p>Model Accuracy: <span id="accuracy"></span>%</p>
    </div>

    <div class="chart-container">
        <canvas id="priceChart"></canvas>
    </div>
</div>

<script>
    let chart;
    let currentFileName = '';

    async function uploadFile() {
        const fileInput = document.getElementById('csvFile');
        const file = fileInput.files[0];

        if (!file) {
            showError('Please select a file first');
            return;
        }

        const formData = new FormData();
        formData.append('file', file);

        showLoading(true);
        hideError();
        clearUploadStatus();
```

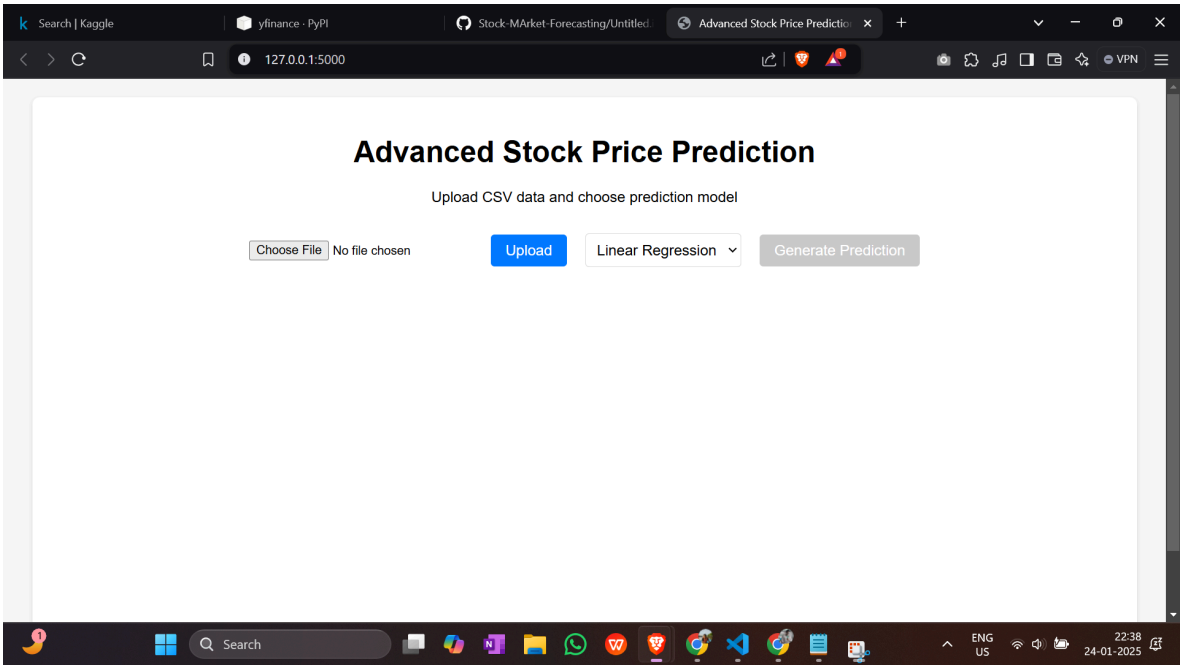
```
        try {
            const response = await fetch('/upload', {
                method: 'POST',
                body: formData
            });

            const data = await response.json();

            if (data.error) {
                showError(data.error);
                return;
            }

            currentFileName = data.filename;
                    document.getElementById('uploadStatus').textContent = 'File
uploaded successfully!';
            document.getElementById('predictBtn').disabled = false;
        } catch (error) {
            showError('Error uploading file: ' + error.message);
            document.getElementById('predictBtn').disabled = true;
        } finally {
            showLoading(false);
        }
    }

    async function predict() {
        if (!currentFileName) {
            showError('Please upload a file first');
            return;
        }

        const modelType = document.getElementById('modelSelect').value;
        showLoading(true);
        hideError();
        hideResults();

        try {
            const response = await fetch('/predict', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify({
                    filename: currentFileName,
                    model_type: modelType
                })
```

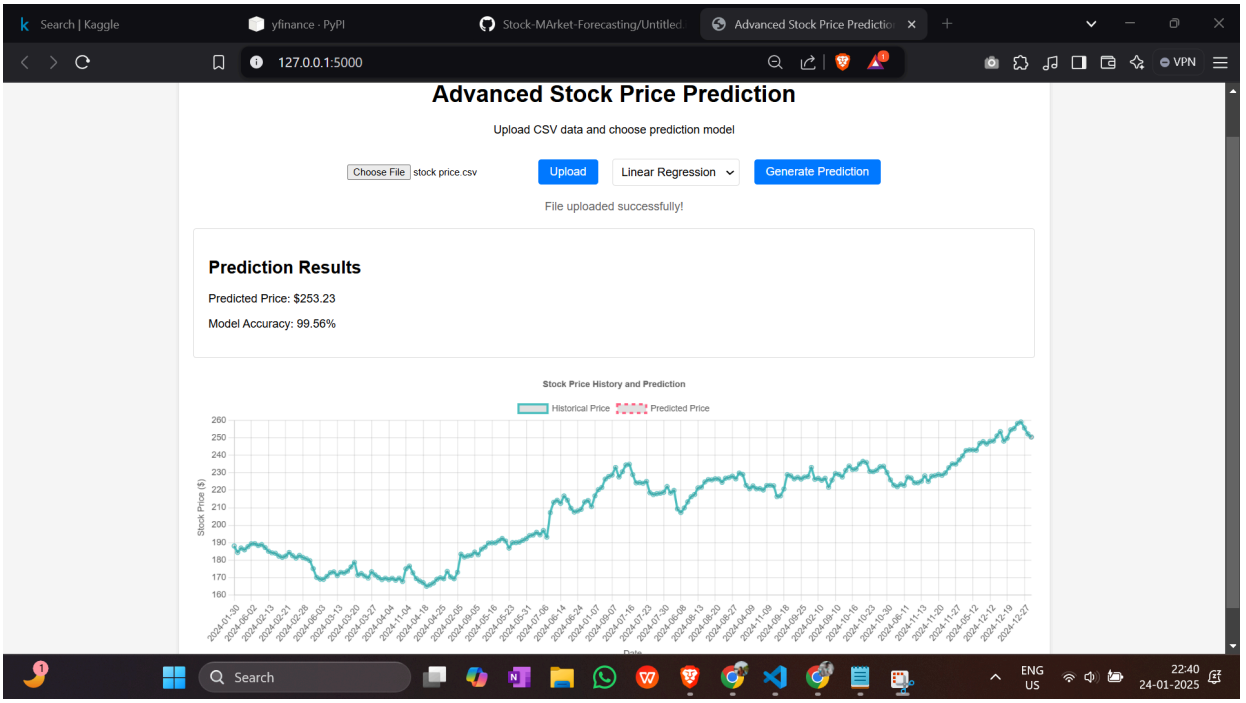
```
        });

        const data = await response.json();

        if (data.error) {
            showError(data.error);
            return;
        }

        updateResults(data);
        updateChart(data);
        showResults();
    } catch (error) {
        showError('Error generating prediction: ' + error.message);
    } finally {
        showLoading(false);
    }
}

function updateResults(data) {
    document.getElementById('predictedPrice').textContent =
        data.prediction.toFixed(2);
    document.getElementById('accuracy').textContent =
        (data.accuracy * 100).toFixed(2);
}

function updateChart(data) {
    const ctx = document.getElementById('priceChart').getContext('2d');

    if (chart) {
        chart.destroy();
    }

    chart = new Chart(ctx, {
        type: 'line',
        data: {
            labels: data.dates,
            datasets: [{
                label: 'Historical Price',
                data: data.historical_data,
                borderColor: 'rgb(75, 192, 192)',
                tension: 0.1
            }, {
                label: 'Predicted Price',
                data: [...Array(data.historical_data.length - 1).fill(null),
                    data.historical_data[data.historical_data.length - 1],
```

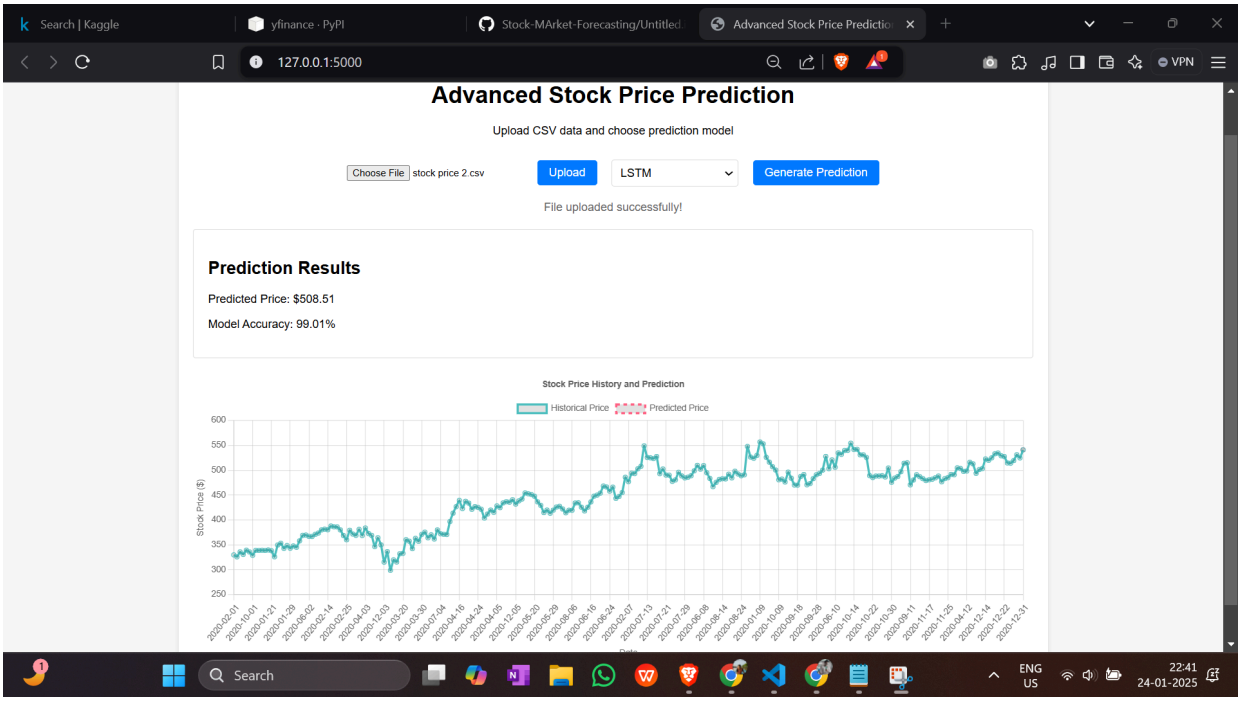
```
                    data.prediction],
                borderColor: 'rgb(255, 99, 132)',
                borderDash: [5, 5],
                tension: 0.1
            }]
        },
        options: {
            responsive: true,
            maintainAspectRatio: false,
            scales: {
                y: {
                    beginAtZero: false,
                    title: {
                        display: true,
                        text: 'Stock Price ($)'
                    }
                },
                x: {
                    title: {
                        display: true,
                        text: 'Date'
                    }
                }
            },
            plugins: {
                title: {
                    display: true,
                    text: 'Stock Price History and Prediction'
                }
            }
        }
    });
}

function showLoading(show) {
    document.getElementById('loading').style.display = show ? 'block' :
'none';
}

function showError(message) {
    const errorDiv = document.getElementById('error');
    errorDiv.textContent = message;
    errorDiv.style.display = 'block';
}

function hideError() {
```

```
        document.getElementById('error').style.display = 'none';
    }

    function clearUploadStatus() {
        document.getElementById('uploadStatus').textContent = '';
    }

    function showResults() {
        document.getElementById('results').style.display = 'block';
    }

    function hideResults() {
        document.getElementById('results').style.display = 'none';
    }
  </script>
</body>
</html>
```

4. Results/Output:-




Search | Kaggle
yfinance · PyPI
Stock-MArket-Forecasting/Untitled
Advanced Stock Price Prediction
127.0.0.1:5000
VPN
Advanced Stock Price Prediction
Upload CSV data and choose prediction model
Choose File
stock price.csv
Upload
Linear Regression
Generate Prediction
File uploaded successfully!
Prediction Results
Predicted Price: $253.23
Model Accuracy: 99.56%
Stock Price History and Prediction
Historical Price
Predicted Price
Stock Price ($)
Search
ENG
US
22:40
24-01-2025


Search | Kaggle
yfinance · PyPI
Stock-MArket-Forecasting/Untitled
Advanced Stock Price Prediction
127.0.0.1:5000
VPN
Advanced Stock Price Prediction
Upload CSV data and choose prediction model
Choose File
stock price 2.csv
Upload
LSTM
Generate Prediction
File uploaded successfully!
Prediction Results
Predicted Price: $508.51
Model Accuracy: 99.01%
Stock Price History and Prediction
Historical Price
Predicted Price
Stock Price ($)
Search
ENG
US
22:41
24-01-2025

5. Remarks:-

Signature of the Student

_____

(Name of the Student)

Signature of the Lab Coordinator

_____

(Name of the Coordinator)