LABORATORY REPORT

Application Development Lab

(CS33002)

B.Tech Program in ECSc

Submitted By

Name:-Manodeep Ray
Roll No: 2230028

Kalinga Institute of Industrial Technology

(Deemed to be University)
Bhubaneswar, India

# TABLE OF CONTENT

| | |
|---|---|
| **Experiment Number** | 3 |
| **Experiment Title** | Developing a stock prediction webapp |
| **Date of Experiment** | 21/1/25 |
| **Date of Submission** | 28/1/25 |

## 1. Objective:-

To design and develop a stock prediction webapp using machine learning and deep learning models

## 2. Procedure:-

1. Download the stocks data
2. Extract the closing value and form the rolling windows needed to train the model
3. Create your model and Train them using the data.
4. Save the trained model.
5. Create falsk , html for webapp front end.
6. Load the stored models and integrate them into the webapp backend
7. Test the models using the data from the stock prediction data split

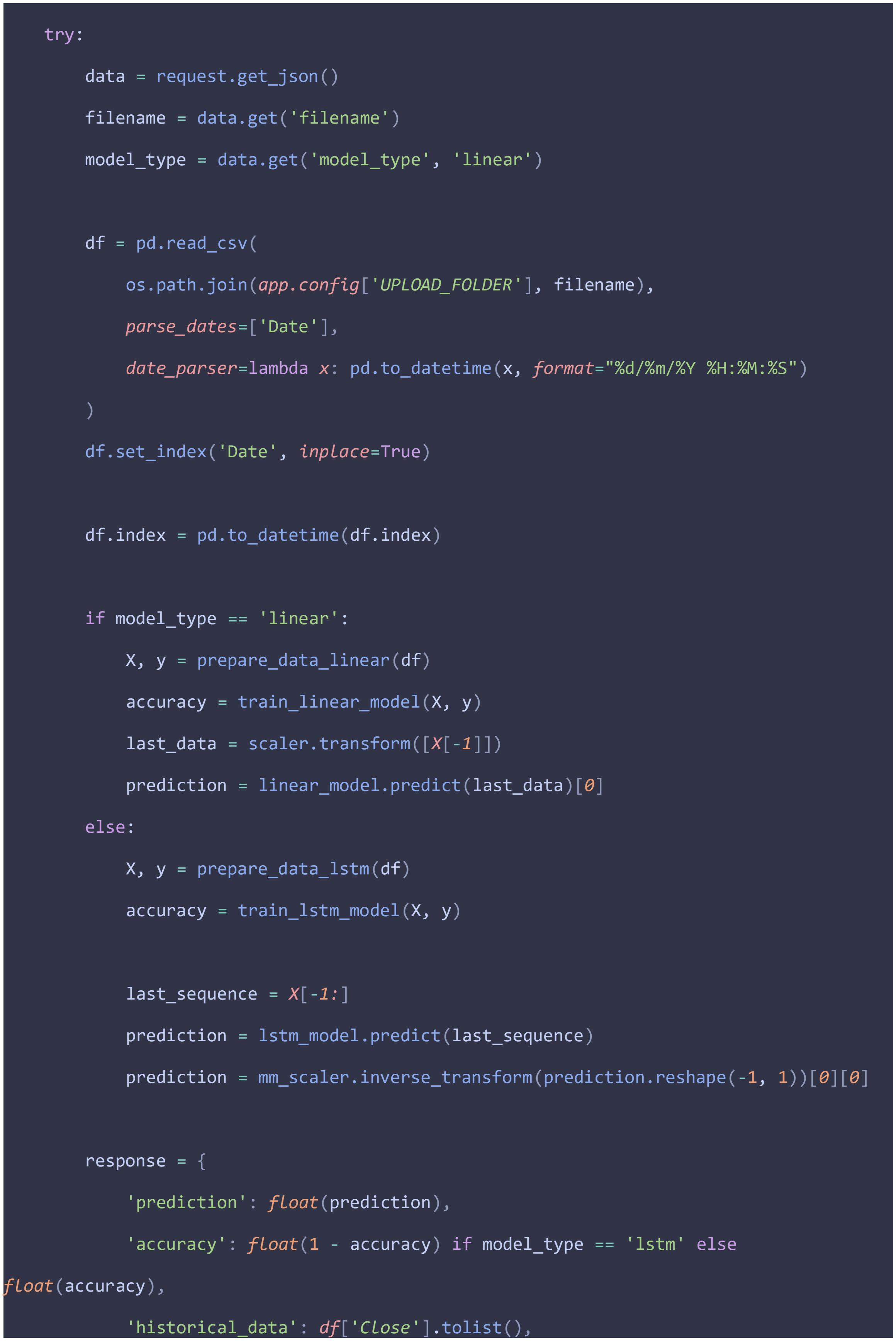
## 3. Code:-

### App

```
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
```

```
os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)
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"""
    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()

    df.dropna(inplace=True)

    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"""
    if len(df) < look_back:
        look_back = len(df) // 2
    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)
```

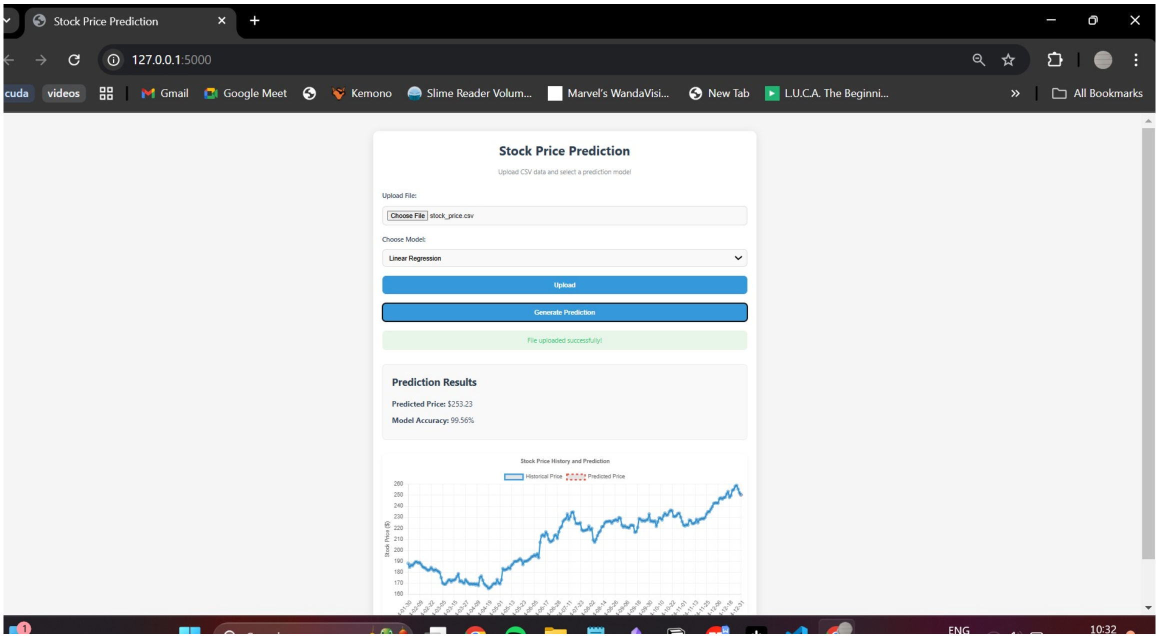
```
def train_lstm_model(X, y):
    """Train the LSTM model"""
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
    global lstm_model
    lstm_model = create_lstm_model((X.shape[1], 1))
    lstm_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.1,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

        df = pd.read_csv(
            os.path.join(app.config['UPLOAD_FOLDER'], filename),
            parse_dates=['Date'],
            date_parser=lambda x: pd.to_datetime(x, format="%d/%m/%Y %H:%M:%S")
        )
        df.set_index('Date', inplace=True)

        df.index = pd.to_datetime(df.index)

        if model_type == 'linear':
            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:
            X, y = prepare_data_lstm(df)
            accuracy = train_lstm_model(X, y)

            last_sequence = X[-1:]
            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.map(lambda x: x.strftime('%Y-%m-%d')).tolist()  # This should work now
        }

        return jsonify(response)

    except Exception as e:
        return jsonify({'error': str(e)}), 500
if __name__ == '__main__':
    app.run(debug=True)
```

## 4. Results/Output:-

For linear regressionmodel



Stock Price Prediction

Upload CSV data and select a prediction model

Upload File: Choose File stock_price.csv

Choose Model: Linear Regression

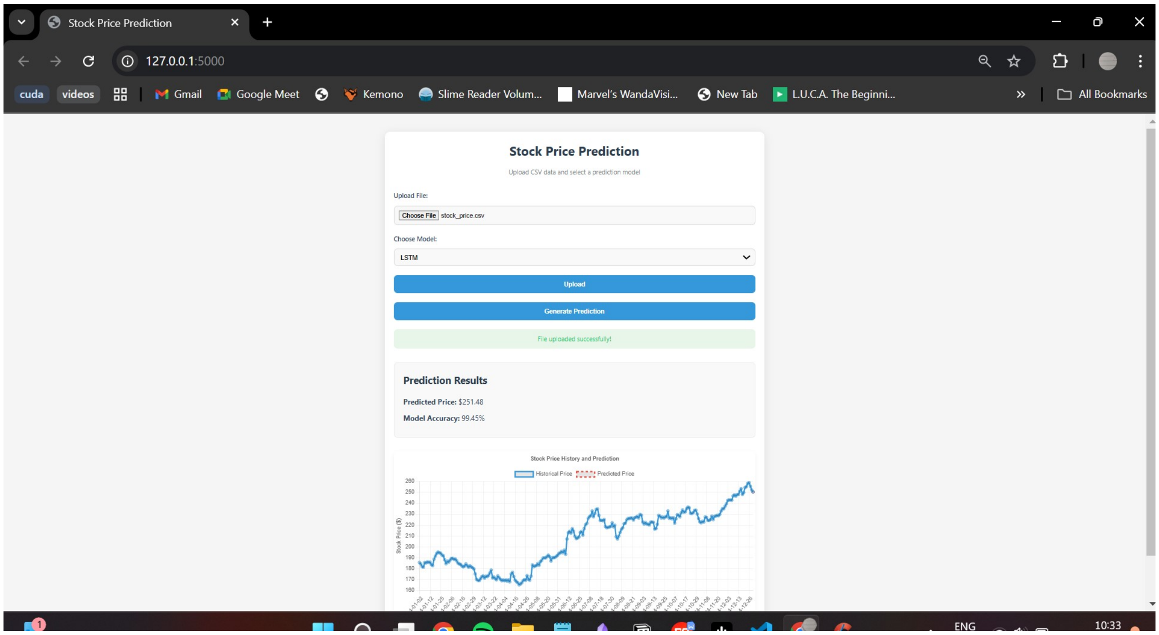
File uploaded successfully!

Prediction Results

**Predicted Price:** $253.23

**Model Accuracy:** 99.56%

For LSTMmodel



Stock Price Prediction

Upload CSV data and select a prediction model

Upload File:

Choose File stock_price.csv

Choose Model:

LSTM

Upload

Generate Prediction

File uploaded successfully!

Prediction Results

**Predicted Price:** $251.48

**Model Accuracy:** 99.45%

## 5. Remarks:-

The Flask application for stock price prediction has been successfully developed and deployed. Through this project, key concepts in deep learning, such as autoregression , recurrent neural network, model training, and inference, were effectively implemented. Additionally, the integration of flask for creating an interactive user interface demonstrated the importance of deploying machine learning models in a user-friendly and accessible manner. This exercise not only strengthened technical proficiency in frameworks like TensorFlow ,scikit-learnand flask but also emphasized the significance of real-world usability and presentation in AI applications. The project serves as a robust stepping stone for future work in deploying machine learning solutions for practical and impactful use cases.

Signature of the Student　　　　　　Signature of the Lab Coordinator

________________________　　　　______________________________

(MANODEEP RAY)