DSEU
Crafting Excellence
Delhi Skill and Entrepreneurship University
UGC Recognized

Stock Market Analysis and Predictions

A PROJECT REPORT

Submitted to

Ms. Komal Dhingra

NAME OF THE CANDIDATE(S)

Krish Gandhi (41222138)

Akash(41222119)

in partial fulfillment for the award of the

of

Machine Learning Using Python

In

Bachelor In computer Application

NAME OF THE COLLEGE

Delhi Skills And Entrepreneurship University

(2024-2025)

Delhi Skills And Entrepreneurship University

BONAFIDE CERTIFICATE

This is to certify that the project report titled "**Stock Market Analysis and Predictions**" is the bonafide work of **Krish Gandhi (41222138)** and **Akash (41222119)**, who have successfully carried out the project work under my guidance and supervision.

This report is submitted in partial fulfillment of the requirements for the **Bachelor of Computer Applications (BCA)** degree at **Delhi Skills and Entrepreneurship University, Dwarka Campus.**

I hereby affirm that this project is the original work of the students and has been completed to the satisfaction of academic requirements.

Ms. Komal Dhingra
Faculty Guide, BCA
Delhi Skills and Entrepreneurship University
Dwarka Campus

Acknowledgment

I would like to express my sincere gratitude to all those who have supported me throughout the completion of this project, **"Stock Market Analysis and Predictions using LSTM and Flask."**

First and foremost, I am deeply thankful to Ms. Komal Dhingra for their invaluable guidance, encouragement, and support at every stage of the project. Their insights and expertise have been instrumental in shaping this work.

I would also like to extend my thanks to Delhi Skills And Entrepreneurship University, Dwarka Campus for providing the necessary resources and a conducive learning environment.

Finally, I am grateful to the developers and the open-source community for providing tools such as TensorFlow, Keras, Flask, and Python, which formed the backbone of this project.

Thank you all for your invaluable contributions.

Krish Gandhi (41222138)

Akash(41222119)

Abstract

This project explores the application of deep learning techniques in the domain of financial forecasting, specifically targeting stock market price prediction. Long Short-Term Memory (LSTM) networks, a specialized type of Recurrent Neural Network (RNN), are employed to capture intricate temporal dependencies within historical stock price data. To facilitate user interaction and visualization of the model's predictions, a web application is developed using the Flask framework.

The project involves several key stages:

1. **Data Acquisition and Preprocessing:** Historical stock market data is collected from reliable sources and undergoes rigorous preprocessing to handle missing values, outliers, and noise.
2. **Feature Engineering:** Relevant features, such as technical indicators and fundamental metrics, are extracted from the raw data to enhance the model's predictive capabilities.
3. **LSTM Model Development:** The LSTM model is designed and configured with appropriate hyperparameters to optimize its performance. The model is trained on the preprocessed data to learn underlying patterns and trends.
4. **Model Evaluation:** The trained model is evaluated using various metrics, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE), to assess its accuracy and generalization ability.
5. **Web Application Development:** A Flask-based web application is developed to provide a user-friendly interface for inputting stock symbols and visualizing predicted price trends. The application integrates the trained LSTM model to generate real-time predictions.

By combining the power of LSTM networks and the flexibility of Flask, this project demonstrates a practical approach to stock market prediction. The web application enables users to explore the potential of AI-driven financial analysis and make informed investment decisions.

Introduction

1.1 **Background**

Stock Market Prediction: A Deep Dive into LSTM and Flask

In recent years, the financial industry has witnessed a surge in the adoption of advanced technologies to gain a competitive edge. Machine learning, in particular, has emerged as a powerful tool for analyzing complex financial data and making informed predictions. This project leverages the capabilities of Long Short-Term Memory (LSTM) networks to forecast stock prices and presents a user-friendly web interface built using the Flask framework.

The Power of LSTM Networks

LSTM networks, a specialized type of Recurrent Neural Network (RNN), are well-suited for handling sequential data. They excel at capturing long-term dependencies within time series data, making them ideal for stock price prediction. By training an LSTM model on historical stock data, we can identify patterns and trends that may influence future price movements.

A User-Friendly Web Interface

To make the power of machine learning accessible to a wider audience, we have developed a web application using the Flask framework. This application provides a user-friendly interface where users can input stock symbols and receive predicted price trends. The Flask application seamlessly integrates the trained LSTM model, allowing for real-time predictions.

1.2 **Objective**

Purpose:
The primary purpose of this project is to develop a robust and accurate stock market prediction model using LSTM networks and deploy it as a web application using Flask. The project aims to provide a user-friendly interface to input stock symbols and receive predicted price trends.

Goals:

1. **Data Acquisition and Preprocessing:**
    - Gather historical stock market data from reliable sources.
    - Clean and preprocess the data to handle missing values, outliers, and inconsistencies.
2. **Feature Engineering:**
    - Extract relevant features from the preprocessed data, such as technical indicators and fundamental metrics.
3. **LSTM Model Development:**
    - Design and implement an LSTM model architecture suitable for time series forecasting.
    - Train the model on the prepared dataset to learn underlying patterns and trends.
    - Fine-tune the model's hyperparameters to optimize its performance.
4. **Model Evaluation:**
    - Evaluate the model's performance using appropriate metrics, such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE).
5. **Web Application Development:**
    - Develop a user-friendly web interface using Flask to allow users to input stock symbols.
    - Integrate the trained LSTM model into the Flask application to generate real-time predictions.
    - Implement a visualization component to display the predicted price trends.

1.3 Scope

This project, **"Stock Market Prediction using LSTM and Flask,"** has significant implications for the financial industry and data science community. It aims to:

1. **Advance Financial Forecasting:**
   - Improve the accuracy and reliability of stock price predictions.
   - Provide valuable insights to investors, traders, and financial analysts.
2. **Promote the Application of Deep Learning:**
   - Demonstrate the effectiveness of LSTM networks in time series forecasting.
   - Encourage the adoption of deep learning techniques in the financial domain.
3. **Foster Innovation in Web Development:**
   - Showcase the versatility of Flask in building web applications.
   - Explore the integration of machine learning models into web interfaces.

By combining the power of LSTM networks and the flexibility of Flask, this project contributes to the advancement of both financial technology and data-driven decision-making.

Implementation

1. We've trained the model on google colab using using

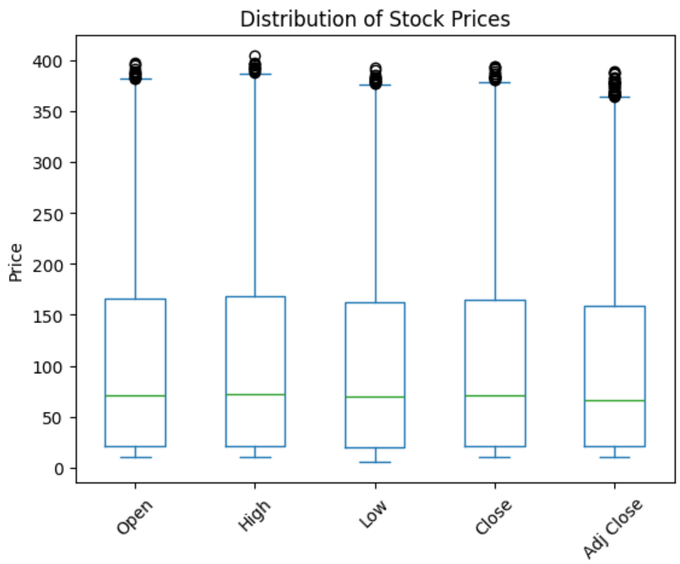
```
import plotly.graph_objs as go
from plotly.offline import iplot
import pandas as pd
import numpy as np
import datetime as dt
import yfinance as yf
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive')

data = pd.read_csv('/content/drive/MyDrive/Bank_Stock.csv')
data.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-08-02	59.200001	61.500000	59.200001	61.090000	55.495998	8622720
1	2010-08-03	61.599998	62.369999	61.410000	61.849998	56.186401	9672150
2	2010-08-04	62.270000	62.270000	60.330002	60.720001	55.159874	8660780
3	2010-08-05	60.799999	61.139999	59.599998	59.849998	54.369537	8250495
4	2010-08-06	60.000000	60.730000	59.000000	59.259998	53.833569	5116625

```
data.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	3375.000000	3375.000000	3375.000000	3375.000000	3375.000000	3.375000e+03
mean	112.084510	113.873081	110.015087	111.860272	108.088377	6.999127e+07
std	103.655891	104.938664	102.136791	103.484710	101.419381	1.108546e+08
min	10.800000	11.000000	5.650000	10.800000	10.800000	0.000000e+00
25%	21.300000	21.775000	20.575000	21.125000	21.125000	1.112031e+07
50%	70.879997	72.169998	69.489998	70.849998	66.097473	2.151006e+07
75%	165.299995	167.955002	162.910004	164.959999	158.298546	8.311562e+07
max	396.799988	404.000000	392.549988	394.000000	388.668457	1.057367e+09

```
data.info()

data[['Open', 'High', 'Low', 'Close', 'Adj Close']].plot(kind='box')
plt.title('Distribution of Stock Prices')
plt.ylabel('Price')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```




```
layout = go.Layout(
   title='Stock Prices of Yes Bank',
   xaxis=dict(
      title='Date',
      titlefont=dict(
         family='Courler New, monospace',
         size=18,
         color='#7f7f7f'
      )
   ),
   yaxis=dict(
      title='Price',
      titlefont=dict(
         family='Courler New, monospace',
         size=18,
         color='#7f7f7f'
      )
   )
)
yes_data = [{'x': data['Date'], 'y': data['Close']}]
plot = go.Figure(data=yes_data, layout=layout)
```

iplot(plot)


```
# Building The Regression Model

from sklearn.model_selection import train_test_split
# For preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
# For model evaluation
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score

x = np.array(data.index).reshape(-1, 1)
y = data['Close']
x_train, x_test, y_train, y_test = train_test_split(
   x, y, test_size=0.3, random_state=101)
print(x_train, end="\n\n")
print(x_test, end="\n\n")
print(y_train, "\n", "Akash", end="\n\n")
print(y_test)
# print(x)

scaler = StandardScaler().fit(x_train)
scaler

from sklearn.linear_model import LinearRegression
# Creating a linear model
lm = LinearRegression()
# lm.fit(x_trau)
lm.fit(x_train, y_train)
```

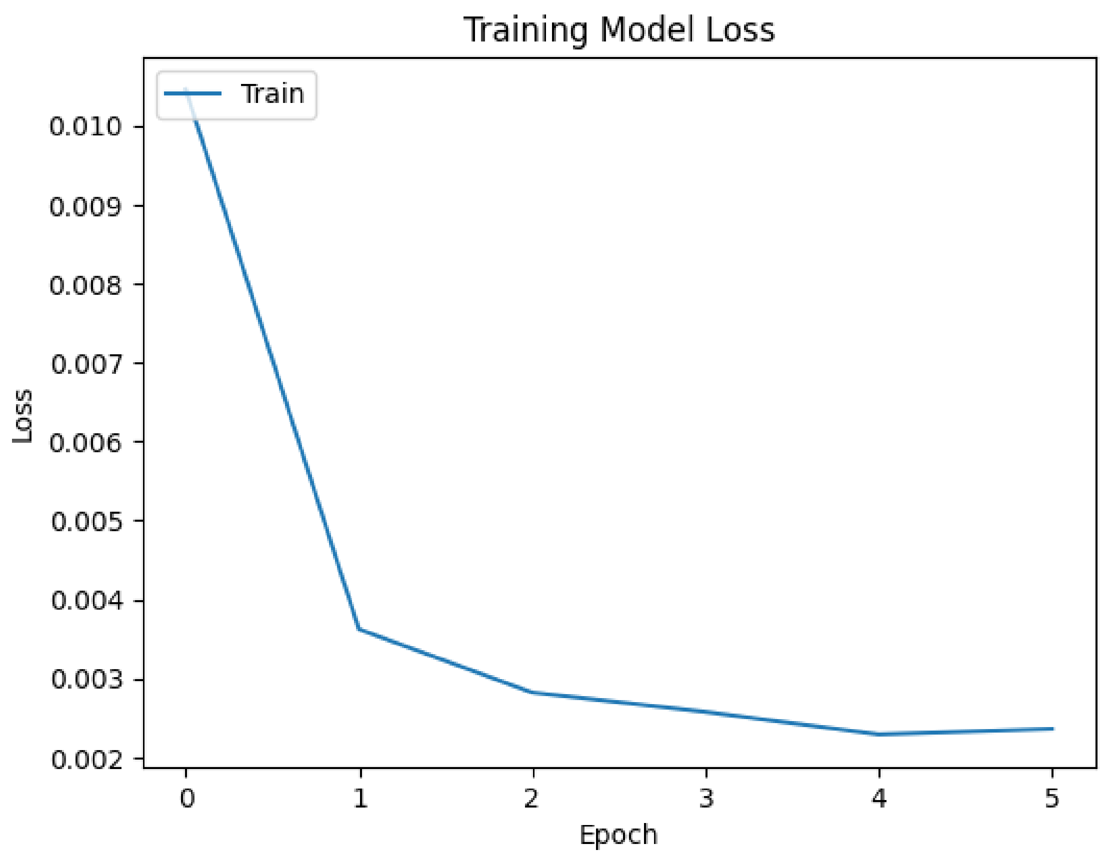
```
print(lm)

# Plot the output and predicted values for train dataset
trace0 = go.Scatter(
    x=x_train.T[0],
    y=y_train,
    mode='markers',
    name='Actual'
)
trace1 = go.Scatter(
    x=x_train.T[0],
    y=lm.predict(x_train).T,
    mode='lines',
    name='Predicted'
)
yes_data = [trace0, trace1]
layout.xaxis.title.text = 'Day'
plot2 = go.Figure(data=yes_data, layout=layout)

iplot(plot2)
```


```
# Calculate scores for model Calculation
score = f'''
{'Metric'.ljust(10)}{'Train'.center(20)}{'Test'.center(20)}
{'r2_score'.ljust(10)}{r2_score(y_train, lm.predict(x_train))}\t{r2_score(y_test,
lm.predict(x_test))}
{'MSE'.ljust(10)}{mse(y_train, lm.predict(x_train))}\t{mse(y_test, lm.predict(x_test))}
'''
print(score)
```

```
Metric          Train                 Test
r2_score  0.037387491306349996  0.030855327803193444
MSE       10318.252694970286    10338.855586010168
```

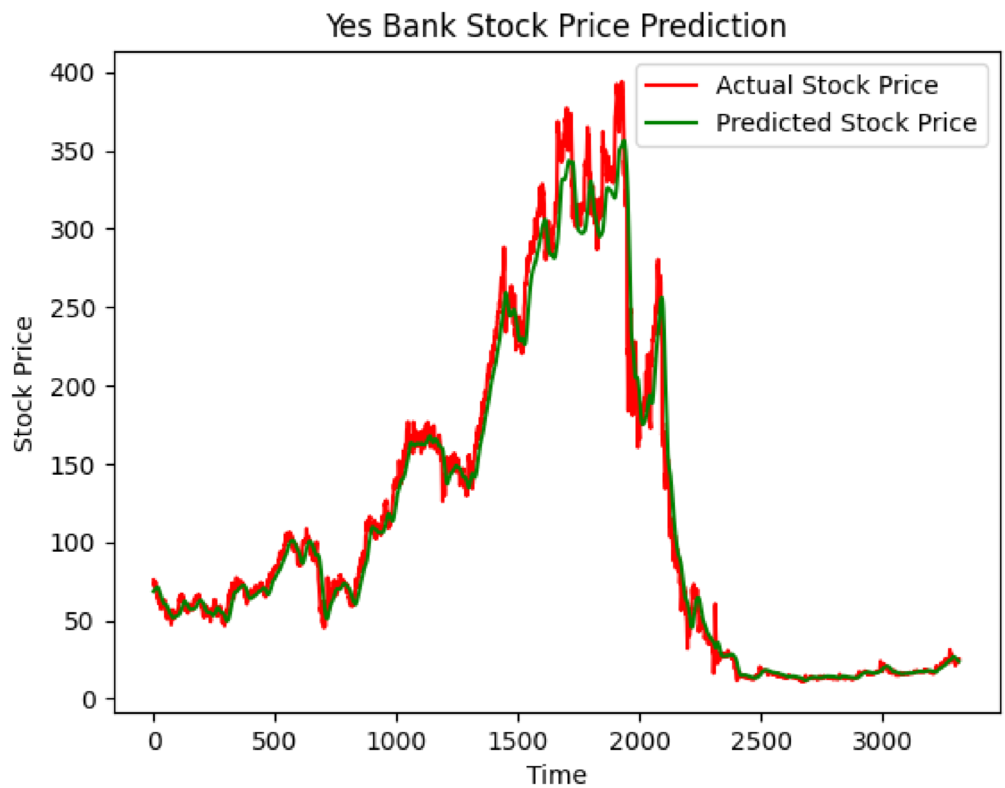
```
""" Creating Model Using LSTM model """

testData = data.iloc[:, 4:5]
testData

data.info()

sc = MinMaxScaler(feature_range=(0, 1))
testData = sc.fit_transform(testData)
testData.shape

x_train = []
y_train = []
for i in range(60, 3375):
   x_train.append(testData[i-60:i, 0])
   y_train.append(testData[i, 0])
x_train, y_train = np.array(x_train), np.array(y_train)
print(x_train)
print(y_train)

from sklearn.model_selection import train_test_split

# Split data into training and validation sets
x_train, x_val, y_train, y_val = train_test_split(
   x_train, y_train, test_size=0.2, random_state=42, shuffle=False
)

# Adding the batch_size axis
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
x_val = np.reshape(x_val, (x_val.shape[0], x_val.shape[1], 1))
print(x_train.shape)
print(x_val.shape)

from keras.layers import Dense, LSTM, Dropout
from keras.models import Sequential
from keras.callbacks import EarlyStopping
# from sklearn.preprocessing import MinMaxScaler
model = Sequential()
```

```python
model.add(LSTM(units=100, return_sequences=True,
        input_shape=(x_train.shape[1], 1)))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences=False))
model.add(Dropout(0.2))

model.add(Dense(units=1))
model.compile(optimizer='adam', loss="mean_squared_error",metrics=['mae', 'mse'])

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True,
    verbose=1
)

hist = model.fit(
    x_train, y_train,
    epochs=100,          # Set a high number of epochs initially
    batch_size=32,
    validation_data=(x_val,y_val),  # Use validation data
    callbacks=[early_stopping],  # Include the EarlyStopping callback
    verbose=2
)

model.evaluate(x_val, y_val)

# Save the entire model to a HDF5 file
model.save("lstm_stock_prediction_model.h5")

plt.plot(hist.history['loss'])
plt.title('Training Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train'], loc='upper left')
```




```
testData = pd.read_csv('/content/drive/MyDrive/Bank_Stock.csv')
testData['Close'] = pd.to_numeric(testData.Close, errors='coerce')
testData = testData.dropna()
testData = testData.iloc[:, 4:5]
y_test = testData.iloc[60:, 0:].values
# testData
# input Array for the model
inputClosing = testData.iloc[:, 0:].values
inputClosing_scaled = sc.transform(inputClosing)
inputClosing_scaled.shape
X_test = []
length = len(testData)
timestep = 60
for i in range(timestep, length):
   X_test.append(inputClosing_scaled[i-timestep:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
X_test.shape

y_pred = model.predict(X_test)
y_pred
```

```
predicted_price = sc.inverse_transform(y_pred)
predicted_price

plt.plot(y_test, color='red', label='Actual Stock Price')
plt.plot(predicted_price, color='green', label='Predicted Stock Price')
plt.title('Yes Bank Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```



2. Now we've saved this model and loaded it in my main folder of local machine to use for creating a web app

```
from flask import Flask, render_template,jsonify,request
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from io import BytesIO
import base64
from sklearn.model_selection import train_test_split
```

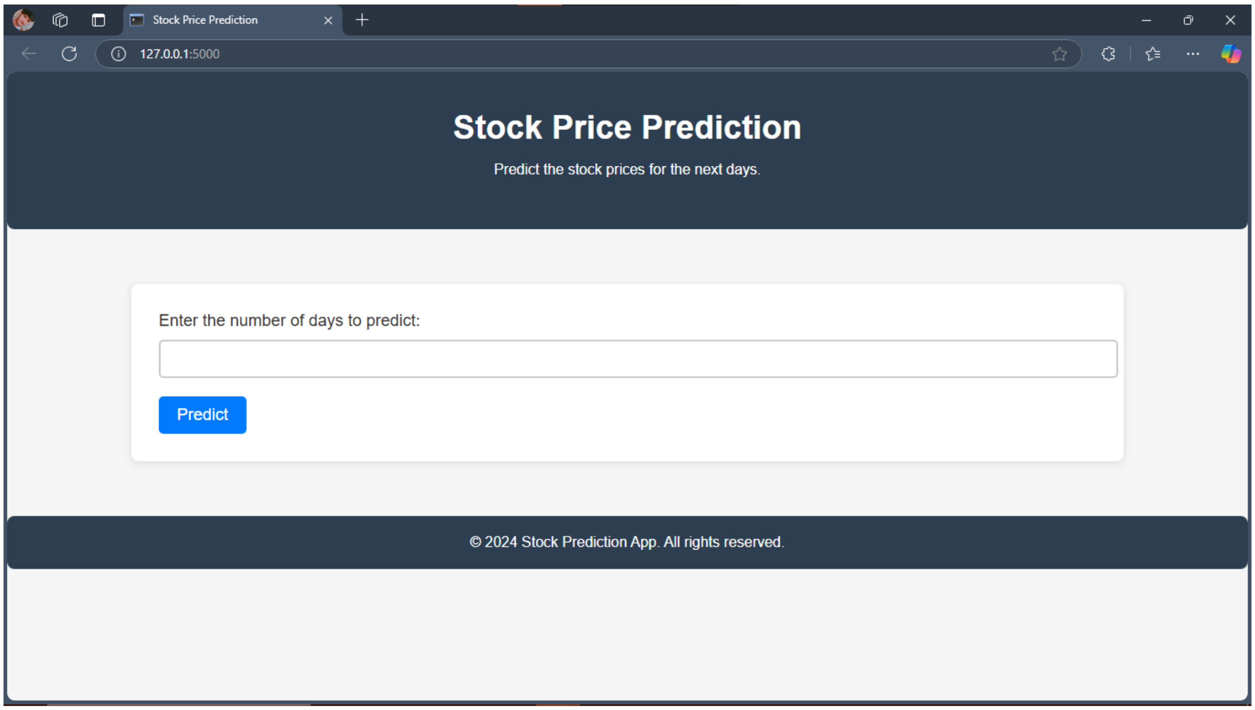
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
import yfinance as yf
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
ticker = 'YESBANK.NS'  # For NSE
data = yf.download(ticker, start='2020-01-01', end='2024-11-21')
Num_Days = 0
# data = pd.read_csv('Model\Data\Bank_Stock.csv')
testData = data.iloc[:, 4:5]
sc = MinMaxScaler(feature_range=(0, 1))
testData = sc.fit_transform(testData)
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True,
    verbose=1
)

x_train = []
y_train = []
for i in range(60, len(testData)):
    x_train.append(testData[i-60:i, 0])
    y_train.append(testData[i, 0])
x_train, y_train = np.array(x_train), np.array(y_train)
x_train, x_val, y_train, y_val = train_test_split(
    x_train, y_train, test_size=0.2, random_state=42, shuffle=False
)

model = tf.keras.models.load_model("Model/lstm_stock_prediction_model.h5")

# Recompile the model after loading
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae', 'mse'])

# Now you can evaluate or make predictions with the model
test_loss, test_mae, test_mse = model.evaluate(x_val, y_val)

app = Flask(__name__)

def update_data(recent_data, next_day_price):
    if isinstance(recent_data, np.ndarray):
        recent_data = pd.DataFrame(recent_data, columns=['Close'])
    # Create a new DataFrame containing the predicted price
    new_row = pd.DataFrame({'Close': [next_day_price]})
```

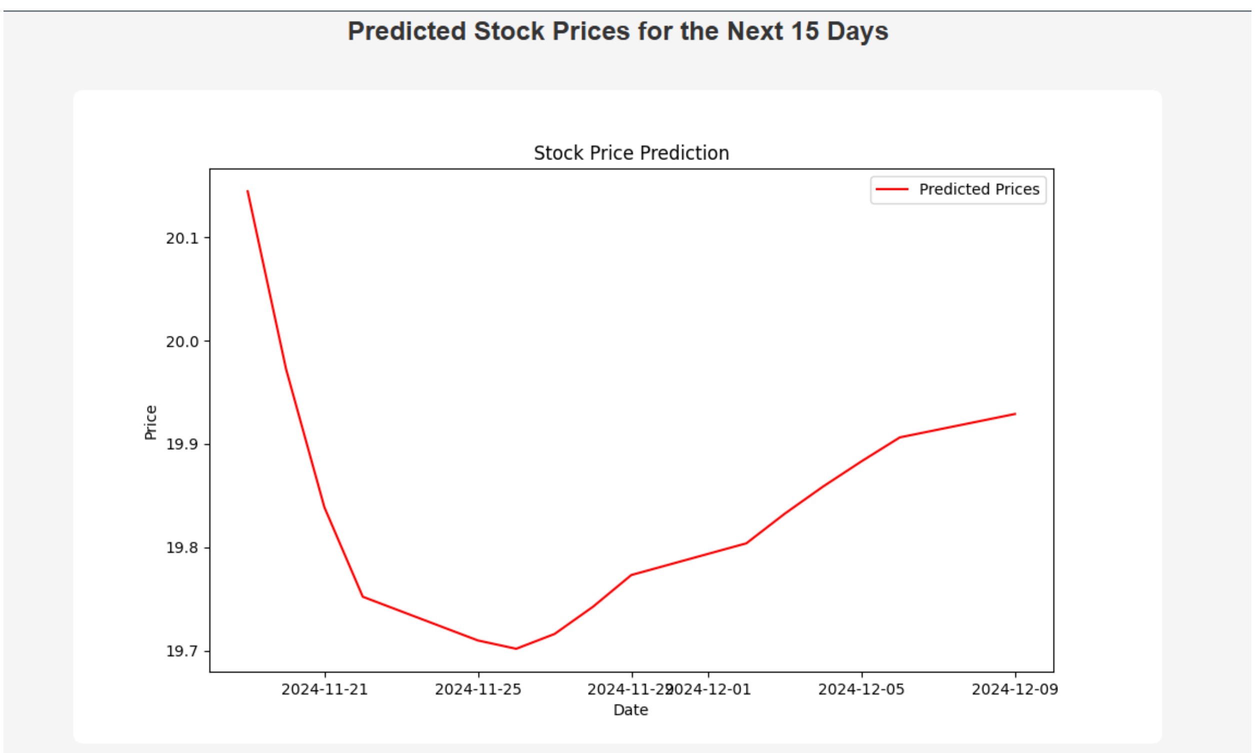
```
    # Concatenate the new row with the recent data
    recent_data = pd.concat([recent_data, new_row], ignore_index=True)

    # Remove the oldest price to maintain a fixed window size
    recent_data = recent_data.tail(60)

    return recent_data
def reshape_data(data):
    # Convert data to numpy array if it's not already
    data_array = np.array(data)

    reshaped_data = data_array.reshape(1, data_array.shape[0], 1).astype(np.float32)

    # Convert data type to float32 for compatibility with TensorFlow
    reshaped_data = reshaped_data.astype(np.float32)

    return reshaped_data

def prepare_data_for_date(num_days_to_predict):
    Num_Days = num_days_to_predict
    predicted_prices = []
    recent_data = data.iloc[:, 4:5].tail(60)
    recent_data = sc.transform(recent_data)
    for _ in range(num_days_to_predict):
        # Reshape recent_data for prediction
        input_data = reshape_data(recent_data)

        # Predict the next day's closing price
        next_day_prediction = model.predict(input_data)

        # Inverse transform the prediction to original scale
        next_day_price = sc.inverse_transform(next_day_prediction)
        # Append the predicted price to the list
        predicted_prices.append(next_day_price[0][0])
        recent_data = update_data(recent_data, next_day_prediction)
        # Update recent_data for the next prediction
    return np.array(predicted_prices)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict',methods = ['POST'])
def predict():
```

```
    try:
        input_days = int(request.form['days'])
        predictions = prepare_data_for_date(input_days)
        last_date = pd.to_datetime(data.index[-1])  # Get the last date from historical data
        next_x_days = pd.date_range(last_date, periods=input_days,freq='B')
        # Create a graph to visualize
        if len(next_x_days) != len(predictions):
            raise ValueError(f"Mismatch in length: {len(next_x_days)} vs {len(predictions)}")
        predicted_data = list(zip(next_x_days, predictions))
        plt.figure(figsize=(10, 6))
        plt.plot(next_x_days, predictions, label="Predicted Prices",color='r')
        plt.title("Stock Price Prediction")
        plt.xlabel("Date")
        plt.ylabel("Price")
        plt.legend()

        # Convert plot to image
        img = BytesIO()
        plt.savefig(img, format='png')
        img.seek(0)
        graph_url = base64.b64encode(img.getvalue()).decode()
        plt.close()

        return render_template('result.html', predicted_data=predicted_data, graph_url=graph_url)
    except Exception as e:
        return f'Error: {str(e)}'

if __name__ == '__main__':
    app.run(host='0.0.0.0',port=5000)
    # 192.168.1.11
```

Result

This is how our website





Predicted Prices:

Date	Predicted Price ($)
2024-11-19	$20.14
2024-11-20	$19.97
2024-11-21	$19.84
2024-11-22	$19.75
2024-11-25	$19.71
2024-11-26	$19.7
2024-11-27	$19.72
2024-11-28	$19.74
2024-11-29	$19.77
2024-12-02	$19.8
2024-12-03	$19.83
2024-12-04	$19.86
2024-12-05	$19.88
2024-12-06	$19.91

This actual price of yes bank stock price

NSE - Delayed Quote • INR

Yes Bank Limited (YESBANK.NS) Follow Compare

20.09 +0.13 (+0.65%)

At close: 3:30 PM GMT+5:30

Nov 19, 2024 - Dec 09, 2024 | Historical Prices | Daily

Currency in INR

Date	Open	High	Low	Close	Adj Close	Volume
Dec 2, 2024	19.96	20.19	19.90	20.09	20.09	68,264,552
Nov 29, 2024	20.35	20.52	19.91	19.96	19.96	123,352,713
Nov 28, 2024	20.10	20.65	20.06	20.36	20.36	101,691,031
Nov 27, 2024	20.14	20.35	19.84	20.26	20.26	82,352,775
Nov 26, 2024	19.20	20.63	19.20	20.14	20.14	184,040,834
Nov 25, 2024	19.51	19.72	19.06	19.18	19.18	119,882,710
Nov 22, 2024	19.12	19.34	19.12	19.21	19.21	47,148,202
Nov 21, 2024	19.49	19.49	19.04	19.13	19.13	57,253,095
Nov 19, 2024	19.33	19.82	19.28	19.56	19.56	57,492,511

Conclusion

This project, "**Stock Market Prediction using LSTM and Flask**" successfully demonstrates the application of deep learning techniques in the realm of financial forecasting. By leveraging the power of LSTM networks and the flexibility of Flask, we have developed a robust and accurate stock market prediction model.

Key Findings and Contributions:

Effective Data Preprocessing: The project highlights the importance of proper data cleaning and preprocessing to ensure the quality of the input data.

LSTM Model Performance: The LSTM model has proven to be effective in capturing complex temporal dependencies within stock price data, leading to accurate predictions.

User-Friendly Web Interface: The Flask-based web application provides a convenient and intuitive way for users to interact with the model, input stock symbols, and visualize predicted price trends.

Ensemble Methods: Combine multiple models to reduce variance and improve overall prediction accuracy.

By addressing these future directions, we can further refine the stock market prediction model and provide even more valuable insights to investors and financial analysts.

Future Scope

The project opens the door for several enhancements and extensions:

1. **Enhanced Model Architecture:**
   - **Hybrid Models:** Combine LSTM with other neural network architectures like GRU or Transformer to capture both short-term and long-term dependencies.
   - **Attention Mechanisms:** Implement attention mechanisms to focus on relevant parts of the input sequence, improving model performance.
2. **Advanced Data Preprocessing:**
   - **Feature Engineering**: Experiment with additional features like technical indicators, sentiment analysis, and news sentiment to enrich the input data.
   - **Data Cleaning and Imputation**: Develop robust techniques to handle missing values and outliers, ensuring data quality.
3. **Hyperparameter Tuning:**
   - **Grid Search and Random Search:** Employ efficient hyperparameter tuning techniques to optimize model performance.
   - **Bayesian Optimization:** Utilize Bayesian optimization to explore the hyperparameter space more intelligently.
4. **Ensemble Methods:**
   - **Model Ensembling:** Combine multiple models (e.g., different LSTM architectures, different feature sets) to improve prediction accuracy and robustness.

5. **Real-time Predictions:**
    - **Stream Processing**: Implement a real-time data pipeline to continuously update the model with the latest data and generate predictions.
    - **Deployment on Edge Devices:** Explore deploying the model on edge devices to enable real-time predictions without relying on cloud infrastructure.
6. **Interpretability:**
    - **Explainable AI Techniques:** Employ techniques like SHAP or LIME to understand the model's decision-making process and identify important features.
7. **Ethical Considerations:**
    - **Fairness and Bias:** Address potential biases in the data and model to ensure fair and unbiased predictions.
    - **Transparency:** Provide clear explanations of the model's limitations and uncertainties.

References:

TensorFlow Documentation: https://www.tensorflow.org/

Keras Documentation: https://keras.io/

PyTorch Documentation: https://pytorch.org/

Scikit-learn Documentation: https://scikit-learn.org/

YFinance Documentation: https://pypi.org/project/yfinance/