

**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.

1. **Feature Engineering:**

* Extract relevant features from the preprocessed data, such as technical indicators and fundamental metrics.

1. **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**.**

1. **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).

1. **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

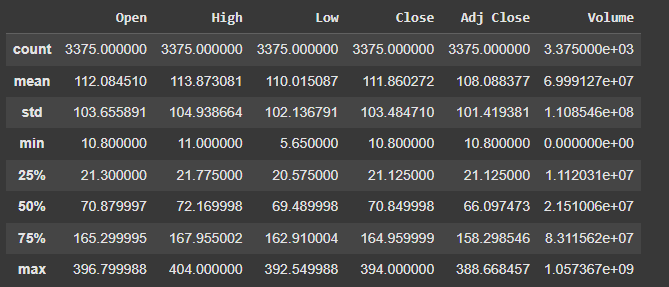
from google.colab import drive

drive.mount('/content/drive')

data = pd.read\_csv('/content/drive/MyDrive/Bank\_Stock.csv')

data.head()

data.describe()



data.info()

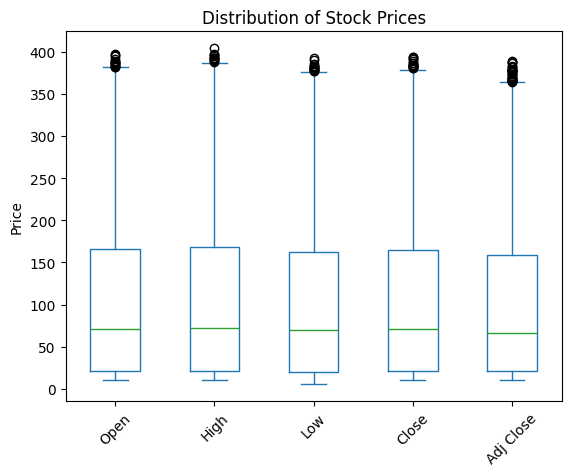
data[['Open', 'High', 'Low', 'Close', 'Adj Close']].plot(kind='box')

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'''

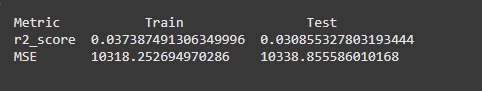
{'Metric'.ljust(10)}{'Train'.center(20)}{'Test'.center(20)}

{'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)



""" 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()

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")

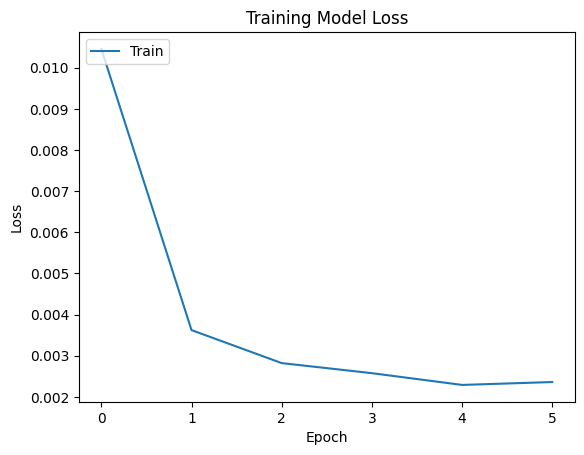
plt.plot(hist.history['loss'])

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')

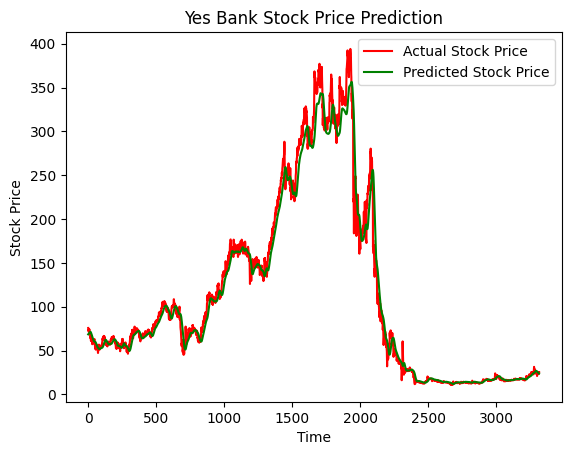
plt.title('Yes Bank Stock Price Prediction')

plt.xlabel('Time')

plt.ylabel('Stock Price')

plt.legend()

plt.show()



1. 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)}'

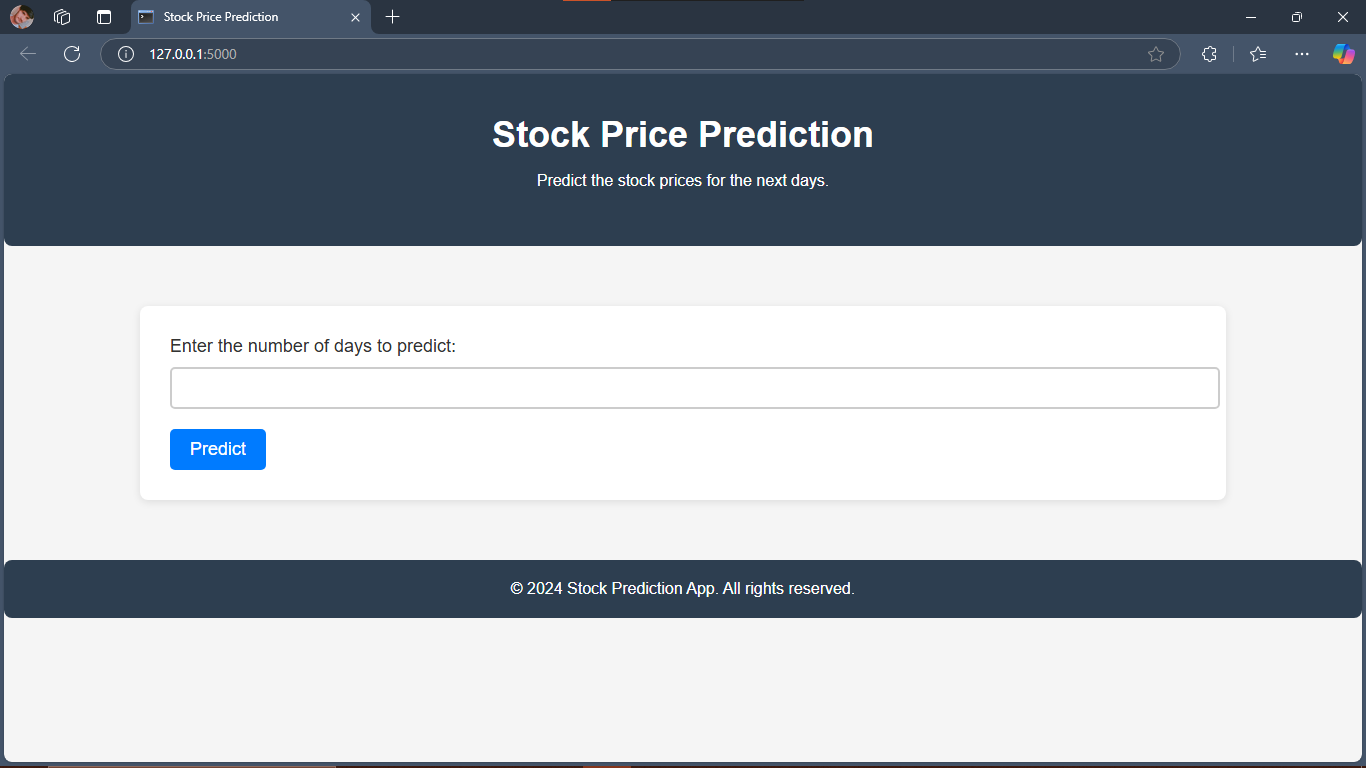
if \_\_name\_\_ == '\_\_main\_\_':

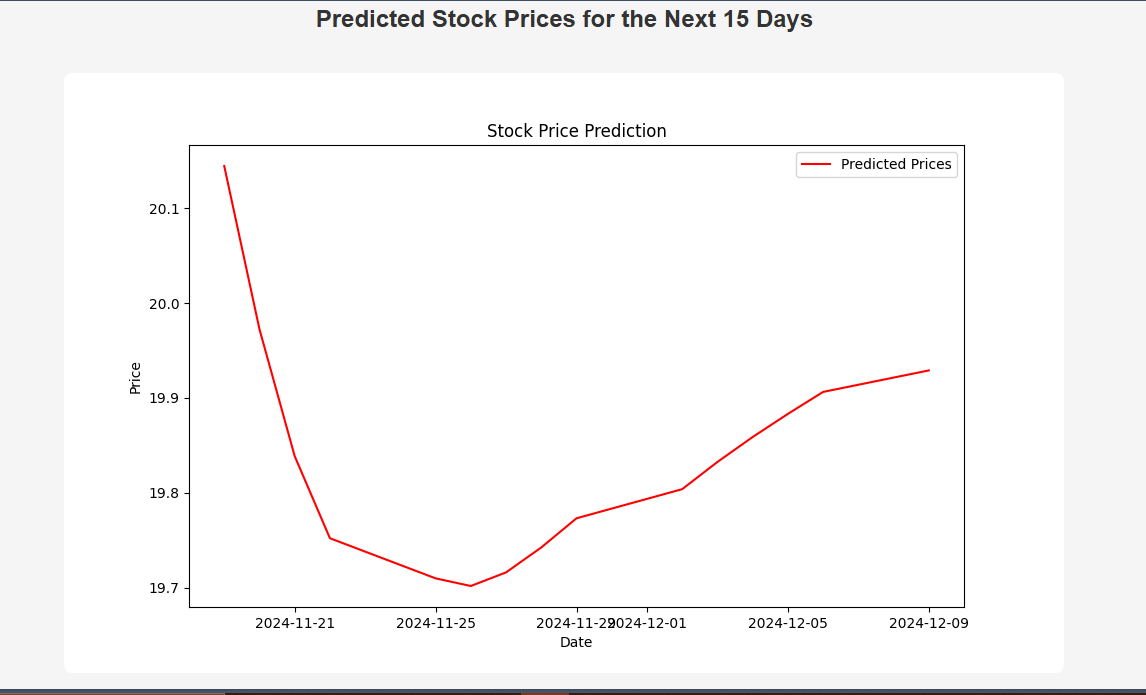
app.run(host='0.0.0.0',port=5000)

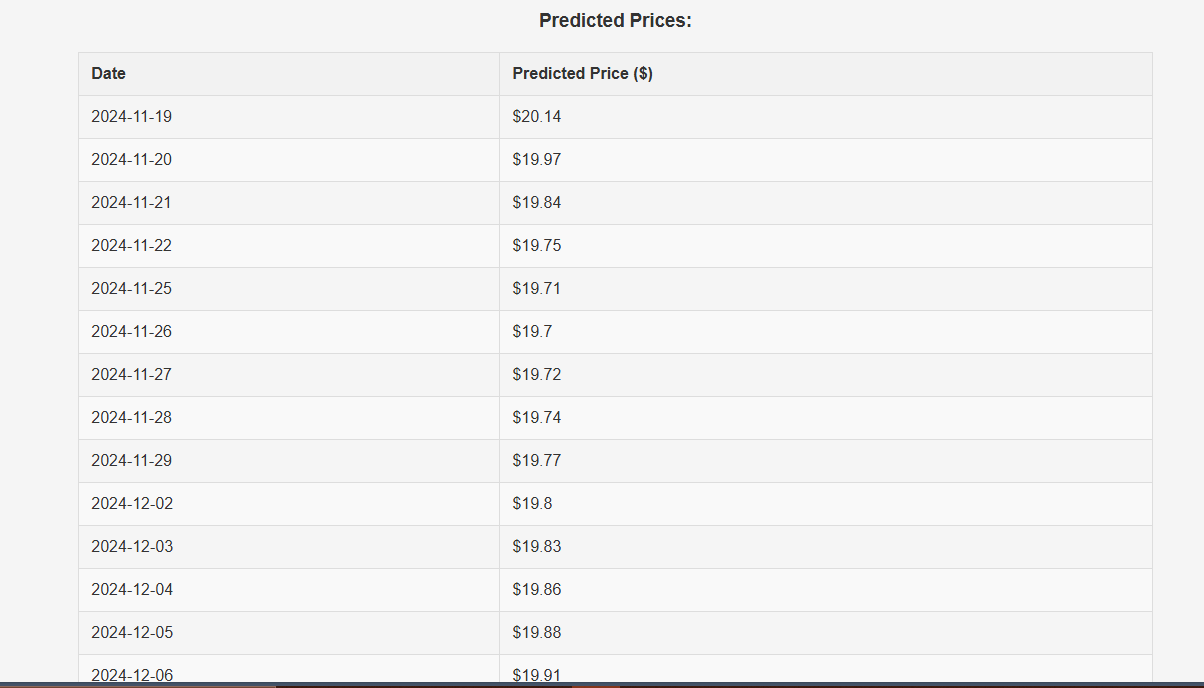
# 192.168.1.11

**Result**

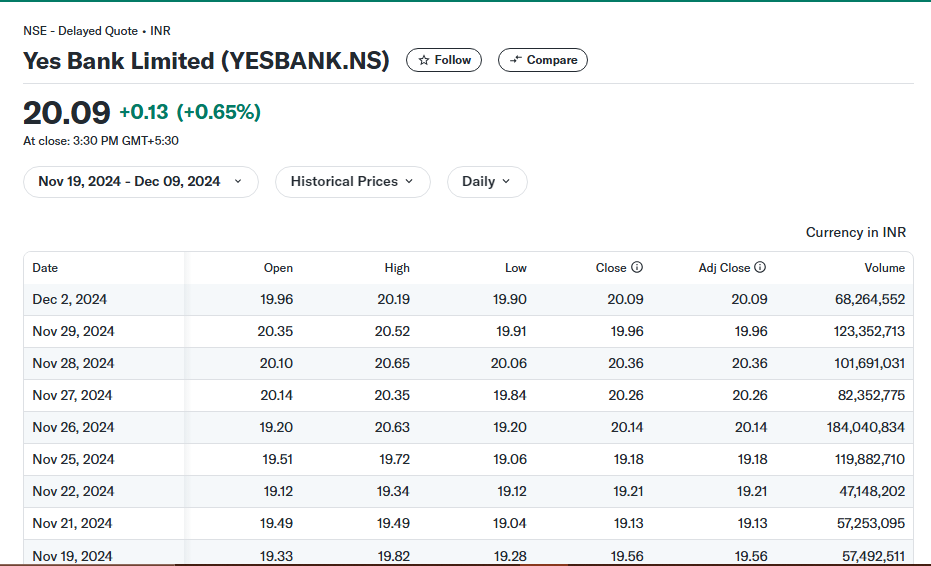
This is how our website

****

****

****

This actual price of yes bank stock price

****

**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.

1. **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.

1. **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.

1. **Ensemble Methods:**

* **Model Ensembling:** Combine multiple models (e.g., different LSTM architectures, different feature sets) to improve prediction accuracy and robustness.

1. **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.

1. **Interpretability:**

* **Explainable AI Techniques:** Employ techniques like SHAP or LIME to understand the model's decision-making process and identify important features.

1. **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/>