Experiment No. 10	
BE (AI&DS)	ROLL NO : 9742
Date of Implementation: 13/11/2024	
Aim: Design and implement GRU for any real life applications	
Programming Language Used : Python	
Upon completion of this experiment, students will be able to LO3: Build and train deep learning models for given problem	

Indicator			
Timeline Maintains submission deadline (2)	On time (2)	Late by a week (1)	Otherwise (0)
Report submission (2)	Completed fully and as per format (2)	Somewhat as per format(1)	Not at all in format (0)
Work done (3)	Excellent(3)	Substantial (2)	Satisfactory(1)
Presentation(5)	In the range 1-5		
Viva (3)	In the range 1-3		

Assessment Marks :

Timeline(2)	
Report submission (2)	
Work done (3)	
Presentation(5)	
Viva (3)	
Total (15)	

EXPERIMENT	***10***
Aim	Design and implement GRU for any real life applications

Instructions :

- Make a group of 2-3 students.
- Select any real life application suitable for experiment
- Collect database
- Design and implement GRU for that application
- Submit the report before deadline
- Give presentation and viva

Report Format

1. Project title:
2. Name of students with roll number
3. Problem statement
4. Database description
5. Program Code
6. Output in form of images/ graph/chart/evaluation measures
7. Conclusion

1. **Project Title**: Stock Price Prediction using GRU

2.

Name	**Roll No**
Aryan Sarang	9741
Akshat Sarraf	9742
Ashvini Chauhan	9807

3. Problem Statement

Stock price prediction is a complex task due to the inherent volatility and unpredictability of financial markets. Traditional methods often fall short in capturing the complex, time-dependent patterns in stock price data. This project aims to leverage **Gated Recurrent Units (GRU)**, a type of Recurrent Neural Network (RNN), to improve stock price forecasting by modeling these sequential dependencies more effectively.
The goal is to develop a predictive model that uses historical stock price data and relevant financial indicators to forecast future stock prices. The model will be deployed as a user-friendly web application using **Streamlit**, allowing users to input stock ticker symbols and view real-time predictions.

This project has the potential to help investors and analysts make more informed decisions by providing AI-powered insights into stock price trends, with the added benefit of making these tools accessible through an intuitive, no-code interface.

4. Database Description

We are using the yfinance library in our project to get our database about stock prices. `yfinance` is a popular Python library that provides an easy-to-use interface for accessing financial data from Yahoo Finance. It allows users to retrieve historical stock prices, real-time market data, dividends, splits, and other financial indicators for various publicly traded companies. The library makes it simple to download stock price data, clean and preprocess it for further analysis, and integrate it with machine learning or data visualization workflows. Some of its usages in our project are:

1. **Loading Stock Data:** We use the `yfinance.download()` function to download historical stock data for the selected stock ticker symbol. The data includes daily stock prices, such as **Open**, **High**, **Low**, **Close**, **Volume**, and **Adj Close**, for a specified time period (in this case, from January 1, 2010, to January 1, 2023).

2. **Stock Data for Model Prediction:** After fetching the data, it is used as input to

preprocess the stock's **closing prices** for the machine learning model (GRU). The model uses this data to make predictions on future stock prices.

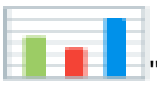
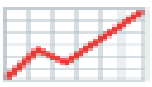
3. **Data Preprocessing:** The raw stock price data obtained through yfinance is processed using **MinMaxScaler** from **sklearn** to normalize the values into a range between 0 and 1. This helps the neural network (GRU model) learn better by reducing the impact of varying scales in the data.

4. **Interactive Visualizations:** yfinance also provides the historical stock data used to generate visualizations such as the pie chart and bar chart in the frontend. This allows users to explore the stock's past performance visually (e.g., comparing positive vs. negative price changes, average monthly closing prices).

5. **Program code**

Frontend Code

```
import streamlit as st
import pickle
import yfinance as yf
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import plotly.graph_objects as go
import plotly.express as px

# Load the trained model for the selected stock
def load_model(stock_ticker):
    filename = f"{stock_ticker}_model.pkl"  # Load model corresponding to the selected stock
    with open(filename, 'rb') as file:
        model = pickle.load(file)
    return model

# Predict next day's price for the selected stock
def predict_next_day_price(model, data, scaler):
    last_60_days = data['Close'].values[-60:].
    last_60_days_scaled = scaler.transform(last_60_days)
    X_predict = np.array([last_60_days_scaled]
    X_predict = np.reshape(X_predict, (X_predict.shape[0], X_predict.shape[1], 1))
    predicted_price = model.predict(X_predict)
    return scaler.inverse_transform(
```

```
# Function to create a pie chart showing stock performance
def create_pie_chart(data):
    # Calculate percentage change over the last 30 days
    data['Pct_Change'] = data['Close'].pct_change() * 100
    positive = len(data[data['Pct_Change'] > 0])
    negative = len(data[data['Pct_Change'] < 0])

    # Pie chart labels and values
    labels = ['Positive', 'Negative']
    values = [positive, negative]

    # Create pie chart using Plotly
    fig = go.Figure(data=[go.Pie(labels=
    fig.update_layout(title_text="
                showlegend=True)
    return fig

# Function to create a bar chart of monthly average closing prices
def create_bar_chart(data):
    # Resample data to get monthly average closing prices
    monthly_data = data['Close'].resample('M').

    # Rename columns for clarity
    monthly_data.columns = ['Month', 'Avg Close Price']

    # Create bar chart using Plotly
    fig = px.bar(monthly_data, x='Month', y='Avg Close Price',
            title="Monthly Average Closing Price",
            labels={'Month': 'Month', 'Avg Close Price': 'Average Close Price'})
    fig.update_layout(xaxis_title=
                xaxis_tickformat='%b %Y')
    return fig

# Streamlit UI

st.set_page_config(page_title=  ", layout="centered")

# Title and Introduction

st.title("  Stock Price Prediction App")
st.markdown("""
    Welcome to the **Stock Price Prediction App**!
    Use this app to predict the next day's stock price for major Indian companies.
```

```
    Choose a stock from the dropdown and see the prediction along with its recent performance breakdown!
    """)

# Select a stock from the list
stocks = ['INFY.NS', 'TCS.NS', 'HDFCBANK.NS', 'SUNPHARMA.NS', 'ONGC.NS', 'HINDUNILVR.NS']
selected_stock = st.selectbox("Select a Stock", stocks)

# Load stock data from Yahoo Finance
data = yf.download(selected_stock, start='2010-01-01', end='2023-01-01')

# Check if the data is empty
if data.empty:
    st.error(f"  No data available for **{selected_stock}**. Please check the ticker symbol and try again.")
else:
    # Show the first few rows of data for debugging
    st.write("### Stock Data Preview", data.head())

    # Display pie chart showing stock performance
    st.plotly_chart(create_pie_

    # Display bar chart showing monthly average closing prices
    st.plotly_chart(create_bar_

    # Load the trained model for the selected stock
    model = load_model(selected_stock)

    # Make a prediction when the button is clicked
    if st.button("  Predict Next Day's Price"):
        scaler = MinMaxScaler(feature_range=(0, 1)).fit(data['Close'].values.
        predicted_price = predict_next_day_price(model, data, scaler)
        st.markdown(f"### Predicted price for **{selected_stock}** next day: **₹{predicted_price:.2f}**")

        # Display additional information or fun facts
        st.markdown(f"#### About **{selected_stock}**:")
        st.write(f"Model used: GRU")

# Add footer or notes
st.markdown("---")
```

st.markdown("Made with  by Team Gambler")

```
# Optionally, add custom CSS to enhance design
st.markdown("""
    <style>
        .css-1d391kg {font-size: 18px;}
        .css-1v0mbd9 {font-size: 16px; font-weight: bold;}
        .stButton>button {background-color: #4CAF50; color: white;}
    </style>
""", unsafe_allow_html=True)
```

Back end Code

```
import pickle
import yfinance as yf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dense, Dropout
from tensorflow.keras.models import load_model as keras_load_model

# Load data for a specific stock
def load_data(ticker, start_date='2010-01-01', end_date='2023-01-01'):
    return yf.download(ticker, start=start_date, end=end_date)

# Preprocess the data (normalization and creating sequences)
def preprocess_data(data):
    scaler = MinMaxScaler(feature_range=(0, 1))
    data_scaled = scaler.fit_transform(data['

    sequence_length = 60
    X, y = [], []
    for i in range(sequence_length, len(data_scaled)):
        X.append(data_scaled[i-
        y.append(data_scaled[i, 0])

    X, y = np.array(X), np.array(y)
    X = np.reshape(X, (X.shape[0], X.shape[1], 1))
    split = int(len(X) * 0.8)  # 80% for training
    return X[:split], y[:split], X[split:], y[split:], scaler
```

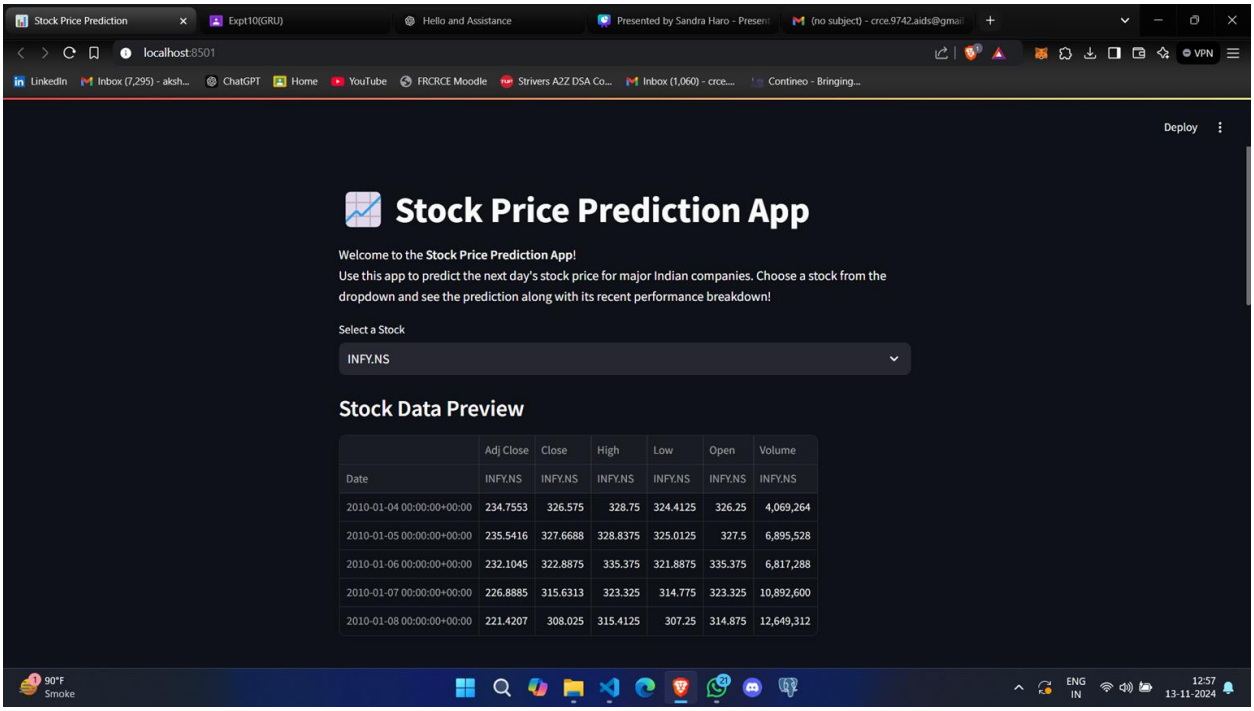
```
# Model building
def build_model(input_shape):
    model = Sequential()
    model.add(GRU(units=50, return_sequences=True, input_shape=input_shape))
    model.add(Dropout(0.2))
    model.add(GRU(units=50, return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))
    model.compile(optimizer='adam'
    return model

# Save the trained model and scaler to a file
def save_model(model, scaler, stock_name, filename):
    model.save(filename)  # Save the Keras model separately
    with open(f"{stock_name}_scaler.
        pickle.dump(scaler, file)  # Save the scaler in a separate file
    print(f"Model and scaler saved as {filename} and {stock_name}_scaler.pkl")

# Load the model and scaler from file
def load_model(stock_name, model_filename):
    model = keras_load_model(model_
    with open(f"{stock_name}_scaler.
        scaler = pickle.load(file)
    return model, scaler

# Predict future stock price based on latest data
def predict_next_day(model, scaler, data, sequence_length=60):
    data_scaled = scaler.transform(data['Close']
    X_input = data_scaled[-sequence_length:]
    scaled_prediction = model.predict(X_input)
    prediction = scaler.inverse_transform(
    return prediction[0][0]  # Return the unscaled prediction value

# Plotting the predicted vs actual stock prices
def plot_predictions(actual, predicted, stock_name):
    plt.figure(figsize=(10, 6))
    plt.plot(actual, color='blue', label='Actual Price')
    plt.plot(predicted, color='red', label='Predicted Price')
    plt.title(f'{stock_name} Stock Price Prediction')
    plt.xlabel('Time')
    plt.ylabel('Stock Price')
    plt.legend()
    plt.show()
```

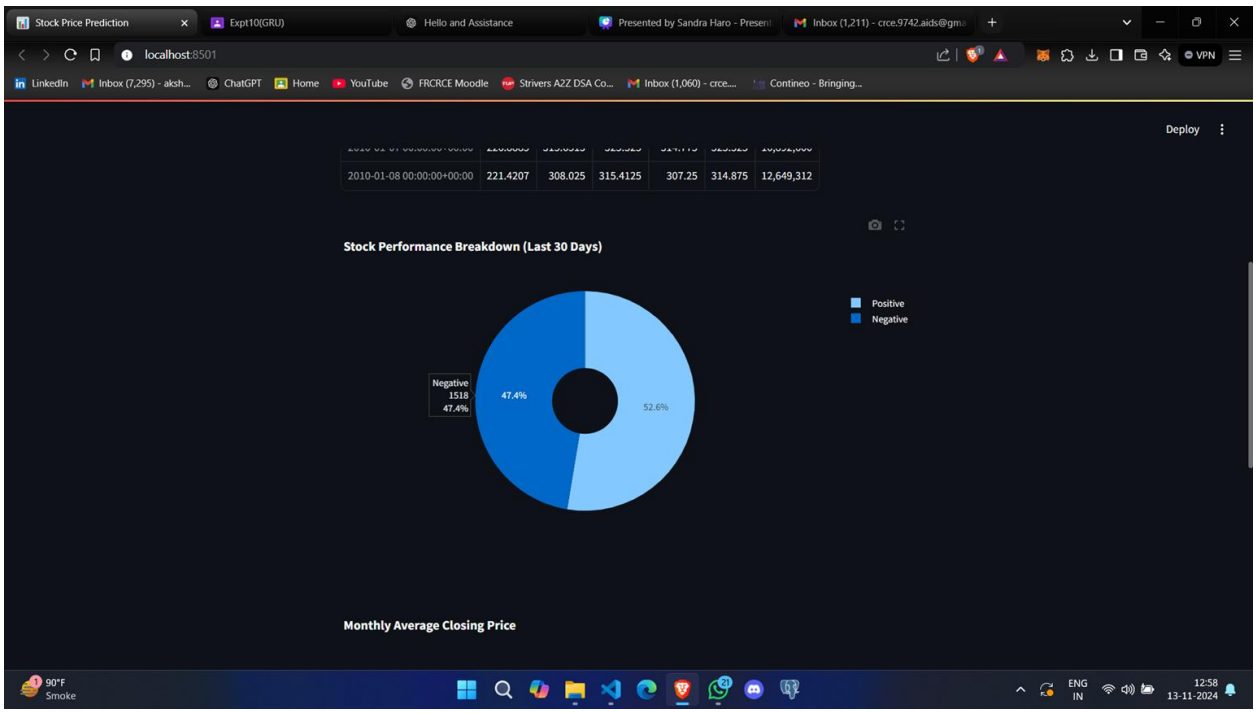
```
# Evaluate model performance
def evaluate_model(y_test, y_pred):
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    return mse, rmse, mae, r2

# Train and save model for a given stock
def train_and_save_model(stock_
    # Load data
    data = load_data(stock_ticker)

    # Preprocess data
    X_train, y_train, X_test, y_test, scaler = preprocess_data(data)

    # Build the model
    model = build_model((X_train.shape[1], 1))

    # Train the model
    model.fit(X_train, y_train, epochs=15, batch_size=32, validation_split=0.1)

    # Save the model and scaler
    save_model(model, scaler, stock_ticker, f"{stock_ticker}_model.h5")

    # Make predictions on the test set
    y_pred = model.predict(X_test)
    y_pred = scaler.inverse_transform(y_
    y_test_actual = scaler.inverse_transform(y_

    # Plot predictions vs actual values
    plot_predictions(y_test_

    # Evaluate model performance
    mse, rmse, mae, r2 = evaluate_model(y_test_actual, y_pred)
    print(f"Performance metrics for {stock_ticker}:")
    print(f"Mean Squared Error (MSE): {mse}")
    print(f"Root Mean Squared Error (RMSE): {rmse}")
    print(f"Mean Absolute Error (MAE): {mae}")
    print(f"R-squared (R²): {r2}")

    # Predict next day's stock price for the given stock
    next_day_prediction = predict_next_day(model, scaler, data)
    print(f"Predicted next day's stock price for {stock_ticker}: {next_day_prediction}")
```

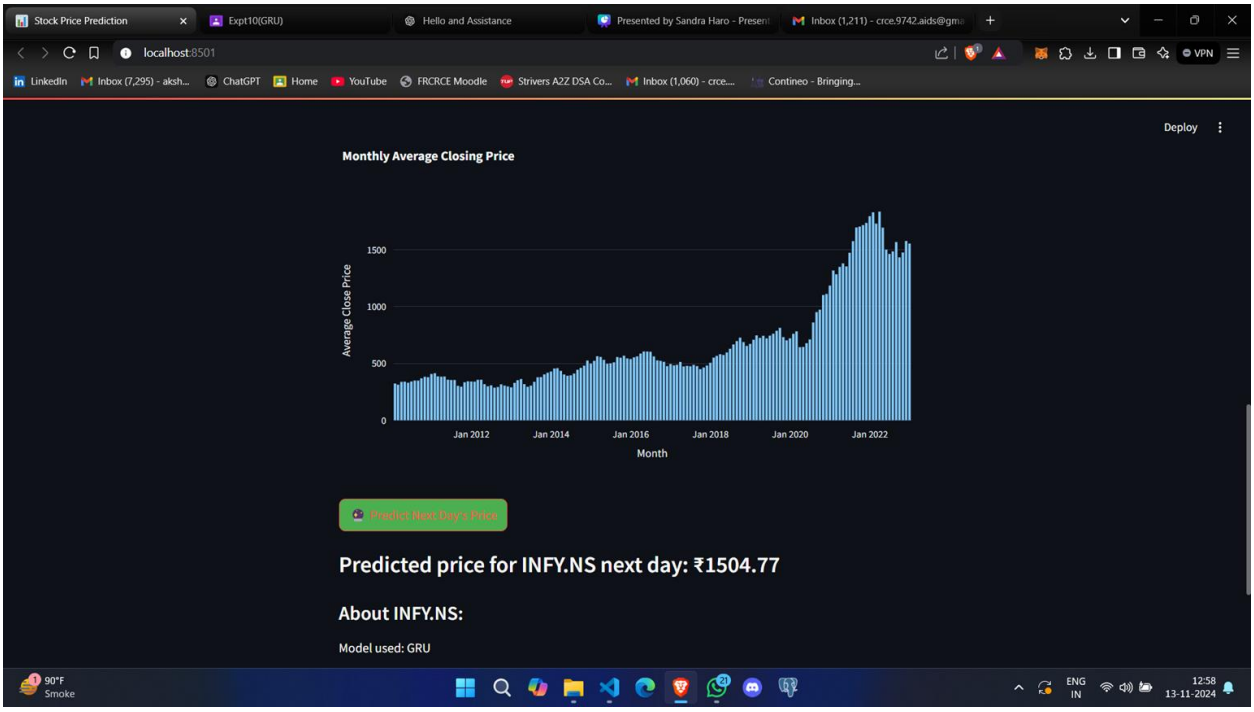
```
# Example for multiple stocks
stocks = ['TCS.NS', 'INFY.NS', 'HDFCBANK.NS', 'SUNPHARMA.NS', 'ONGC.NS',
'HINDUNILVR.NS']
for stock in stocks:
    train_and_save_model(stock)
```

6. Output Screenshots

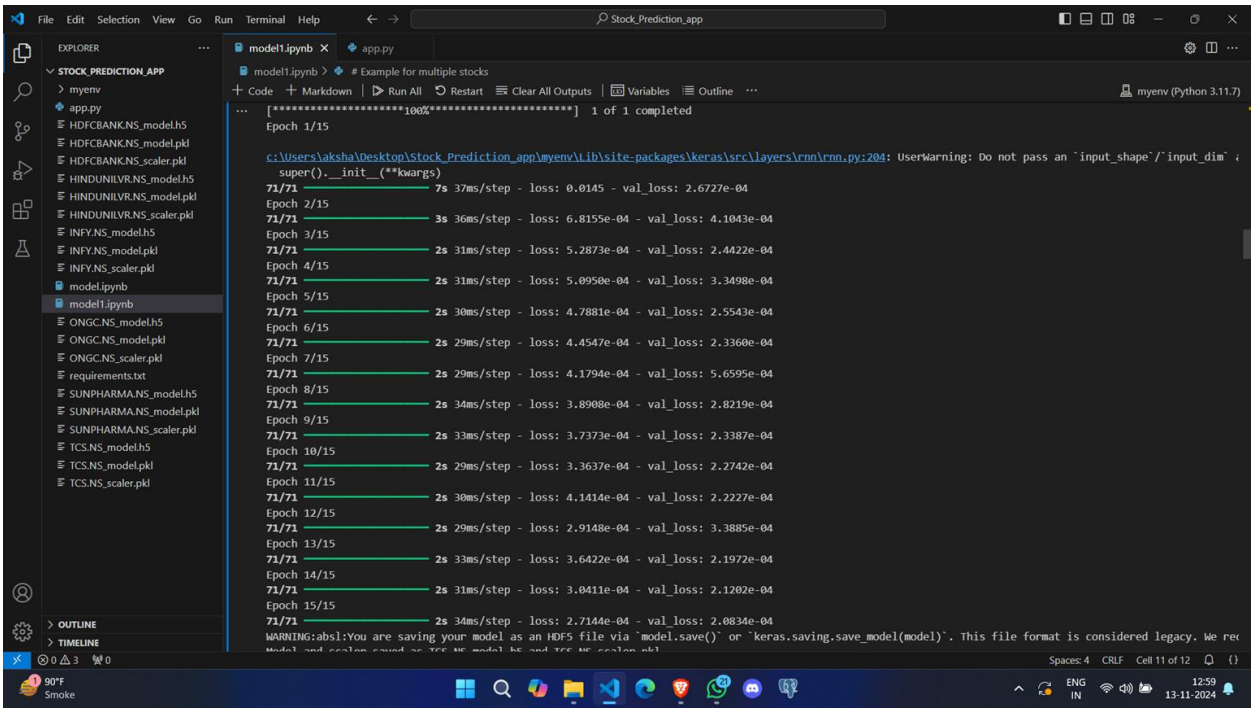
Frontend

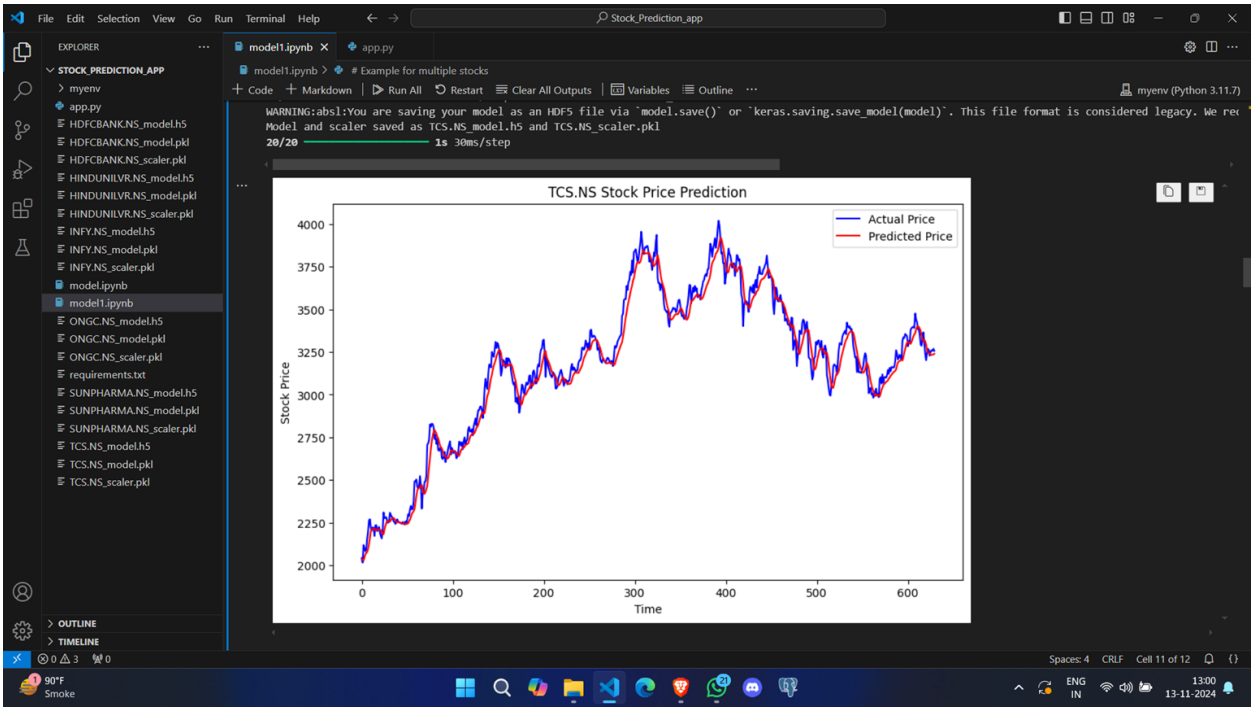



Stock Price Prediction
Expt10(GRU)
Hello and Assistance
Presented by Sandra Haro - Present
Inbox (1,211) - crce.9742.aids@gma
localhost:8501
LinkedIn
Inbox (7,295) - aksh...
ChatGPT
Home
YouTube
FRCRCE Moodle
Strivers A2Z DSA Co...
Inbox (1,060) - crce....
Contineo - Bringing...
Deploy
2010-01-08 00:00:00+00:00 221.4207 308.025 315.4125 307.25 314.875 12,649,312
Stock Performance Breakdown (Last 30 Days)
Positive
Negative
Negative
1518
47.4%
47.4%
52.6%
Monthly Average Closing Price
90°F
Smoke
ENG
IN
12:58
13-11-2024


Stock Price Prediction
Expt10(GRU)
Hello and Assistance
Presented by Sandra Haro - Present
Inbox (1,211) - crce.9742.aids@gma
localhost:8501
LinkedIn
Inbox (7,295) - aksh...
ChatGPT
Home
YouTube
FRCRCE Moodle
Strivers A2Z DSA Co...
Inbox (1,060) - crce....
Contineo - Bringing...
Deploy
Monthly Average Closing Price
Average Close Price
1500
1000
500
0
Jan 2012
Jan 2014
Jan 2016
Jan 2018
Jan 2020
Jan 2022
Month
Predict Next Day's Price
Predicted price for INFY.NS next day: ₹1504.77
About INFY.NS:
Model used: GRU
90°F
Smoke
ENG
IN
12:58
13-11-2024

Backend


File Edit Selection View Go Run Terminal Help
Stock_Prediction_app
EXPLORER
STOCK_PREDICTION_APP
myenv
app.py
HDFCBANK.NS_model.h5
HDFCBANK.NS_model.pkl
HDFCBANK.NS_scaler.pkl
HINDUNILVR.NS_model.h5
HINDUNILVR.NS_model.pkl
HINDUNILVR.NS_scaler.pkl
INFY.NS_model.h5
INFY.NS_model.pkl
INFY.NS_scaler.pkl
model.ipynb
model1.ipynb
ONGC.NS_model.h5
ONGC.NS_model.pkl
ONGC.NS_scaler.pkl
requirements.txt
SUNPHARMA.NS_model.h5
SUNPHARMA.NS_model.pkl
SUNPHARMA.NS_scaler.pkl
TCS.NS_model.h5
TCS.NS_model.pkl
TCS.NS_scaler.pkl
model1.ipynb
app.py
model1.ipynb > # Example for multiple stocks
Code Markdown Run All Restart Clear All Outputs Variables Outline
myenv (Python 3.11.7)
[**********100%**********] 1 of 1 completed
Epoch 1/15
c:\Users\aksha\Desktop\Stock_Prediction_app\myenv\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim`
super().__init__(**kwargs)
71/71 7s 37ms/step - loss: 0.0145 - val_loss: 2.6727e-04
Epoch 2/15
71/71 3s 36ms/step - loss: 6.8155e-04 - val_loss: 4.1043e-04
Epoch 3/15
71/71 2s 31ms/step - loss: 5.2873e-04 - val_loss: 2.4422e-04
Epoch 4/15
71/71 2s 31ms/step - loss: 5.0950e-04 - val_loss: 3.3498e-04
Epoch 5/15
71/71 2s 30ms/step - loss: 4.7881e-04 - val_loss: 2.5543e-04
Epoch 6/15
71/71 2s 29ms/step - loss: 4.4547e-04 - val_loss: 2.3360e-04
Epoch 7/15
71/71 2s 29ms/step - loss: 4.1794e-04 - val_loss: 5.6595e-04
Epoch 8/15
71/71 2s 34ms/step - loss: 3.8908e-04 - val_loss: 2.8219e-04
Epoch 9/15
71/71 2s 33ms/step - loss: 3.7373e-04 - val_loss: 2.3387e-04
Epoch 10/15
71/71 2s 29ms/step - loss: 3.3637e-04 - val_loss: 2.2742e-04
Epoch 11/15
71/71 2s 30ms/step - loss: 4.1414e-04 - val_loss: 2.2227e-04
Epoch 12/15
71/71 2s 29ms/step - loss: 2.9148e-04 - val_loss: 3.3885e-04
Epoch 13/15
71/71 2s 33ms/step - loss: 3.6422e-04 - val_loss: 2.1972e-04
Epoch 14/15
71/71 2s 31ms/step - loss: 3.0411e-04 - val_loss: 2.1202e-04
Epoch 15/15
71/71 2s 34ms/step - loss: 2.7144e-04 - val_loss: 2.0834e-04
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We rec
OUTLINE
TIMELINE
Spaces: 4 CRLF Cell 11 of 12
90°F Smoke
ENG IN
12:59 13-11-2024


File Edit Selection View Go Run Terminal Help
Stock_Prediction_app
EXPLORER
STOCK_PREDICTION_APP
myenv
app.py
HDFCBANK.NS_model.h5
HDFCBANK.NS_model.pkl
HDFCBANK.NS_scaler.pkl
HINDUNILVR.NS_model.h5
HINDUNILVR.NS_model.pkl
HINDUNILVR.NS_scaler.pkl
INFY.NS_model.h5
INFY.NS_model.pkl
INFY.NS_scaler.pkl
model.ipynb
model1.ipynb
ONGC.NS_model.h5
ONGC.NS_model.pkl
ONGC.NS_scaler.pkl
requirements.txt
SUNPHARMA.NS_model.h5
SUNPHARMA.NS_model.pkl
SUNPHARMA.NS_scaler.pkl
TCS.NS_model.h5
TCS.NS_model.pkl
TCS.NS_scaler.pkl
model1.ipynb
app.py
model1.ipynb > # Example for multiple stocks
Code Markdown Run All Restart Clear All Outputs Variables Outline
myenv (Python 3.11.7)
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We rec
Model and scaler saved as TCS.NS_model.h5 and TCS.NS_scaler.pkl
20/20 1s 30ms/step
TCS.NS Stock Price Prediction
Actual Price
Predicted Price
Stock Price
Time
4000
3750
3500
3250
3000
2750
2500
2250
2000
0
100
200
300
400
500
600
OUTLINE
TIMELINE
Spaces: 4 CRLF Cell 11 of 12
90°F Smoke
ENG IN
13:00 13-11-2024

```
Performance metrics for TCS.NS:
Mean Squared Error (MSE): 4912.900194856548
Root Mean Squared Error (RMSE): 70.09208368180067
Mean Absolute Error (MAE): 55.21796468098958
R-squared (R²): 0.9749410309281067
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
[*********************100%**********************]  1 of 1 completed
Predicted next day's stock price for TCS.NS: 3240.3818359375
```

7. Conclusion

In this project, we developed a machine learning-based application for predicting stock prices using a Gated Recurrent Unit (GRU) model, which is well-suited for time-series forecasting tasks like stock price prediction. The project aimed to provide a robust and user-friendly solution for predicting future stock prices of major Indian companies, leveraging historical data from Yahoo Finance.

The key steps in this project included:

- **Data Collection**: Using the yfinance library to fetch historical stock data, which served as the foundation for training the model. The data was preprocessed and normalized using techniques like **MinMaxScaler** to ensure better model performance.
- **Model Development**: We built and trained a **GRU-based neural network** to capture temporal dependencies in stock price movements. The model was trained on historical data, and its performance was evaluated using standard metrics like **Mean Squared Error (MSE)** and **Root Mean Squared Error (RMSE)**.
- **Prediction**: The trained model was used to predict the next day's stock price, allowing users to interactively input stock ticker symbols and receive predictions.
- **Frontend Deployment**: We used **Streamlit** to create an interactive web interface that allows users to select a stock, view its historical data and performance metrics, and obtain predictions about future stock prices in real-time.