

Project -1 Report

on

Stock Simulator

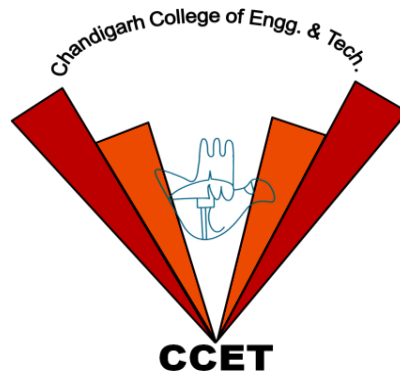
**A Project Report submitted in partial fulfillment of the
requirements for the award of**

Bachelor of Engineering IN COMPUTER SCIENCE AND ENGINEERING

Submitted by

**Saksham Kaushik
(Roll No: CO20346)**

**Under the supervision of
(Dr. Ankit Gupta)**

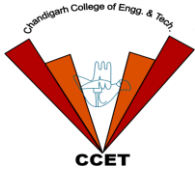


**CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY
(DEGREE WING)**

Government Institute under Chandigarh (UT) Administration, Affiliated to Panjab University
, Chandigarh

Sector-26, Chandigarh. PIN-160019

December 2023



CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY (DEGREE WING)

Government Institute under Chandigarh (UT) Administration | Affiliated to Panjab University, Chandigarh

Sector-26, Chandigarh. PIN-160019 | Tel. No. 0172-2750947, 2750943

Website: www.ccet.ac.in | Email: principal@ccet.ac.in | Fax. No.

:0172-2750872



Department of Computer Sc. & Engineering

CANDIDATE'S DECLARATION

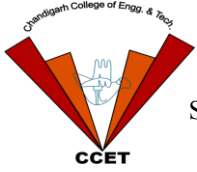
I Shreya Sharma declare the successful completion of my major project on the topic 'stock simulator' in fulfillment of the requirement for the award of the degree B.E. in Computer Science and Engineering submitted in CSE Department, Chandigarh College of Engineering and Technology (Degree Wing) Affiliated to Panjab University, Chandigarh, is an authentic record of my/own work carried out during my degree. The work reported in this has not been submitted by me for an award of any other degree or diploma. I am grateful for the opportunity to engage in this comprehensive learning experience, and it stands as evidence of my commitment to continuous learning in the dynamic landscape of financial markets.

Date : **8 - 01 - 2024**

Shreya Sharma

Place : **Chandigarh**

CO20352



CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY (DEGREE WING)
Government Institute under Chandigarh (UT) Administration | Affiliated to Panjab University, Chandigarh
Sector-26, Chandigarh. PIN-160019 | Tel. No. 0172-2750947, 2750943 Website: www.ccet.ac.in | Email: principal@ccet.ac.in | Fax. No. :0172-2750872



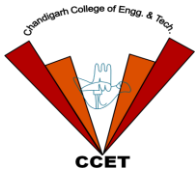
ACKNOWLEDGEMENT

The successful completion of the Stock Simulator project and the preparation of this comprehensive report have been a collaborative effort, and I express my heartfelt gratitude to the individuals and organizations who have played a significant role in this journey.

I am deeply indebted to the faculty and staff of the Computer Science & Engineering Department at Chandigarh College of Engineering and Technology. Their unwavering support, encouragement, and expertise have been instrumental in shaping my understanding and skills in the field.

A special note of appreciation goes to Dr. Ankit Gupta, whose guidance and mentorship added a profound dimension to this project. The invaluable insights and constructive feedback provided by Dr. Ankit Gupta greatly contributed to the project's success.

I express my gratitude to Chandigarh College of Engineering and Technology for fostering an environment that encourages students to actively participate in training programs. The exposure gained during this project has been pivotal in enhancing my practical skills and readiness for the industry.



Department of Computer Sc. & Engineering

ABSTRACT

This report encapsulates the comprehensive exploration and development of a Stock Simulator, a culmination of the major project undertaken by **Shreya Sharma**, a student pursuing a **Bachelor of Engineering in Computer Science and Engineering at Chandigarh College of Engineering and Technology**. The project, conducted under the guidance of Dr. Ankit Gupta, is presented as a detailed account of the technology stack employed and the features incorporated into the application.

The project is meticulously outlined in seven chapters, each focusing on a crucial aspect of the development process. The report constitute a Streamlit web application for a stock simulator project, enabling users to explore historical stock data, visualize plots, analyze statistics, and make stock price predictions using the Prophet library. In `stock_simulator.py`, the code orchestrates the app's structure, incorporating user interface elements like sidebar options, tabs, and a loading spinner. It leverages modular utility functions defined in `services.py` to efficiently load historical stock data from Yahoo Finance, plot data using Plotly, and conduct time series forecasting. The application seamlessly integrates a range of libraries, including Streamlit, yfinance, and Prophet, providing an interactive and informative tool for users to analyze and forecast stock prices. The design emphasizes clarity, modularity, and user-friendly data visualization, showcasing the versatility and ease of development with the Streamlit framework.

CONTENTS

CANDIDATE'S DECLARATION.....	2
CERTIFICATE.....	3
ACKNOWLEDGEMENT.....	4
Department of Computer Sc. & Engineering.....	5
ABSTRACT.....	5
CONTENTS.....	6
List of Table.....	
List of Figures.....	
Chapter 1: Introduction to the Technologies Used.....	7
➤ STREAMLIT Library:.....	7
➤ MATPLOTLIB lib:.....	7
➤ PROPHET lib:.....	7
➤ YFINANCE lib:.....	8
➤ PLOTLY lib:.....	8
Chapter 2: Setting up the App.py Application with Necessary Dependencies.....	9
Implementation Details.....	9
Streamlit Integration and User Interface.....	9
Layout and Sidebar.....	9
Header and Project Information.....	11
Data Loading and Forecasting.....	11
Forecasting with Prophet.....	12
Tab-based Navigation.....	13
Plots Tab:.....	13
Statistics Tab:.....	14
Forecasting Tab:.....	15
Comparison Tab:.....	16
Deployment and Accessibility.....	18
Chapter 3 - SOURCE CODE with explanation.....	19
main.py.....	19
services.py.....	25
Components.....	25
1. Data Loading Function: load_data.....	25
2. Plotting Functions: plot_data, plot_multiple_data, plot_volume.....	25
.stremlit/ config.toml.....	28
Chapter 4 - System Architecture.....	29
User Interface:.....	29
Application Logic:.....	29
Data Storage:.....	29
Visualization Layer:.....	30
Interactions:.....	30
Chapter 5 - Result, Performance and other usecases.....	31
Data Loading and Processing:.....	31
Prediction Model:.....	31
User Interface and Interactivity:.....	31
Visualization:.....	31

Comparative Analysis:.....	31
Scalability and Deployment:.....	32
Performance Optimization in our application.....	32
Caching with Streamlit's st.cache Decorator:.....	32
Lazy Loading with sleep for Loading Indicators:.....	32
Optimized Data Processing:.....	32
Streamlit's Container Width for Plots:.....	32
Prophet Model Optimization:.....	32
Selective Data Loading for Comparison:.....	32
Minimization of Unwanted Columns in DataFrames:.....	33
Periodic Profiling and Optimization:.....	33
Usage scenarios and Practical Applications.....	33
Investment Strategy Planning:.....	33
Risk Management:.....	33
Educational Tool:.....	33
Decision Support for Traders:.....	33
Portfolio Management:.....	33
Integration Possibilities:.....	34
News and Sentiment Analysis:.....	34
Machine Learning Enhancements:.....	34
Real-time Data Feeds:.....	34
Advanced Visualization Tools:.....	34
Future Enhancements:.....	34
Enhanced Forecasting Models:.....	34
Personalized Dashboards:.....	34
Machine Learning-driven Insights:.....	35
Collaborative Features:.....	35
Chapter 7 - Challenges and Considerations in Stock Simulator Application.....	36
Challenges and Considerations in the Stock Simulator Application:.....	36
Data Accuracy and Reliability:.....	36
Model Limitations and Assumptions:.....	36
Handling Data Outliers and Anomalies:.....	36
User Understanding of Financial Markets:.....	36
Integration with External APIs:.....	36
Real-time Data Challenges:.....	36
Security and Privacy Concerns:.....	37
Interpretability of Forecasting Results:.....	37
Scalability:.....	37
Regulatory Compliance:.....	37
Feedback Mechanisms:.....	37
Educational Resources:.....	37
Chapter 8 - Deployment on web hosting.....	38
Hosting on Streamlit Sharing.....	38
1. Prerequisites.....	38
2. Repository Setup.....	38
3. Create Streamlit Sharing Account.....	38
4. Configure Deployment Settings.....	38

5. Deploy.....	38
6. Monitor Deployment.....	39
7. Access the Deployed App.....	39
Chapter 9 - Conclusion.....	40

List of Figures

2.1.....
2.2.....
2.3.....
2.4.....
2.5.....
2.6.....
2.7.....
2.8.....
2.9.....
2.10.....
2.11.....
2.12.....
2.13.....
2.14.....
2.15.....
2.16.....
2.17.....
2.18.....
2.19.....
2.20.....
2.21.....
2.22.....
4.1.....
8.1.....
8.2.....

List of Tables

3.1.....
3.2.....

Chapter 1: Introduction to the Technologies Used

In this chapter, we embark on an exploration of the cutting-edge technologies that form the foundation of the Stock Simulator project. The contemporary landscape of web development demands a strategic selection of tools and frameworks, and this section aims to provide a comprehensive overview of the technologies instrumental to our project's success.

➤ **STREAMLIT Library:**

- Streamlit is a Python library that simplifies the process of creating web applications for data science and machine learning projects. With minimal code, Streamlit enables developers to transform data scripts into interactive web apps, making it accessible to those without web development expertise. Known for its user-friendly design and rapid prototyping capabilities, Streamlit allows users to visualize data, share insights, and deploy applications seamlessly. Its integration with popular data science libraries makes it a versatile tool for building dynamic and engaging web-based data applications.

➤ **MATPLOTLIB lib:**

- Matplotlib is a widely-used Python library for creating static, animated, and interactive visualizations in data science and scientific computing. Renowned for its versatility, Matplotlib provides a plethora of customizable plots and charts, allowing users to represent data in various formats. Its integration with Jupyter Notebooks and other data science tools makes it a go-to choice for researchers and analysts. With an extensive set of features, Matplotlib facilitates the creation of high-quality visualizations for exploratory data analysis and presentation purposes.

➤ **PROPHET lib:**

- Prophet is an open-source forecasting tool developed by Facebook for time-series analysis. Designed to handle daily observations with strong seasonal patterns, it simplifies the process of predicting future values with minimal input requirements. Prophet incorporates an additive model that includes components for seasonality, holidays, and special events, making it particularly effective for capturing complex

time-series patterns. With its intuitive interface and automatic handling of outliers, Prophet has become a popular choice for analysts and data scientists seeking accurate and accessible time-series forecasting capabilities in Python.

➤ **YFINANCE lib:**

- yfinance is a Python library that provides a simple and convenient interface to interact with Yahoo Finance data. It allows users to easily access and download historical market data, stock quotes, and financial information directly from Yahoo Finance. With yfinance, developers can seamlessly integrate financial data into their Python applications for analysis, visualization, and algorithmic trading. Its straightforward usage and compatibility with popular data science tools make it a valuable resource for researchers and traders seeking to incorporate real-time and historical market data into their workflows.

➤ **PLOTLY lib:**

- Plotly is a versatile Python library for creating interactive and visually appealing data visualizations. Its strength lies in its ability to generate interactive plots, dashboards, and web-based applications with ease. Plotly supports a wide range of chart types and offers a high level of customization, making it suitable for both exploratory data analysis and publication-ready visualizations. With support for multiple programming languages and frameworks, including Python, R, and JavaScript, Plotly is a popular choice among data scientists and analysts for creating compelling and interactive data-driven stories.

Chapter 2: Setting up the App.py Application with Necessary Dependencies

This chapter serves as the gateway to the practical implementation of our Stock Simulator. Building upon the foundational knowledge introduced in Chapter 1, we delve into the specifics of setting up our development environment and configuring a robust Next.js application. The step-by-step guide provided here ensures a seamless initiation of the project, allowing for a smooth transition into subsequent chapters.

Implementation Details

Streamlit Integration and User Interface

The Stock Simulator App is developed using the Streamlit framework, a Python library designed for creating interactive web applications with ease. The application interface is designed to be user-friendly, providing an intuitive experience for users to explore historical stock data, visualize forecasts, and compare multiple stocks.

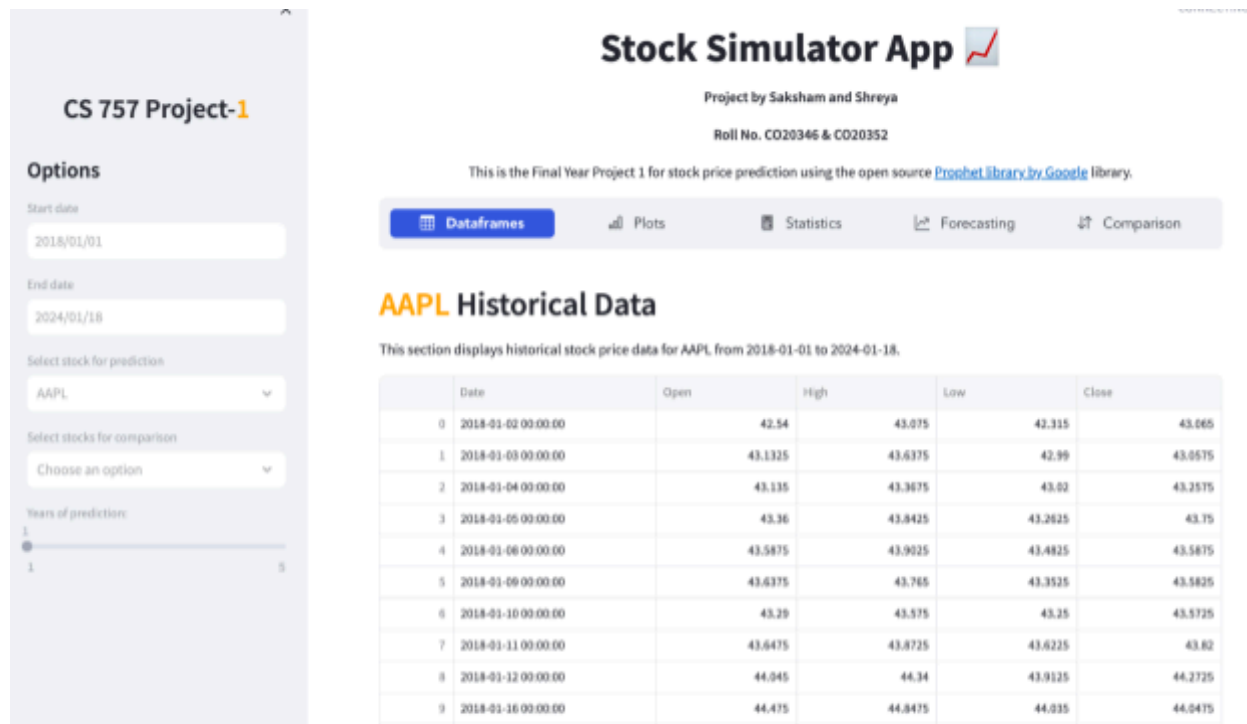
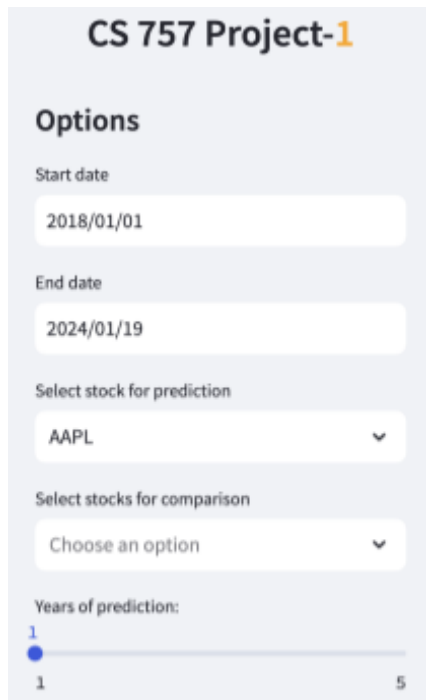


Fig. 2.1

Layout and Sidebar

The application layout is set to a wide format using `st.set_page_config(layout="wide")` to optimize the display of various elements.

The sidebar, located on the left, offers user options for selecting the start and end dates, choosing the stock for prediction, and configuring forecasting parameters. It is built using Streamlit components, such as date pickers and sliders, enhancing the overall user experience.



CS 757 Project-1

Options

Start date
2018/01/01

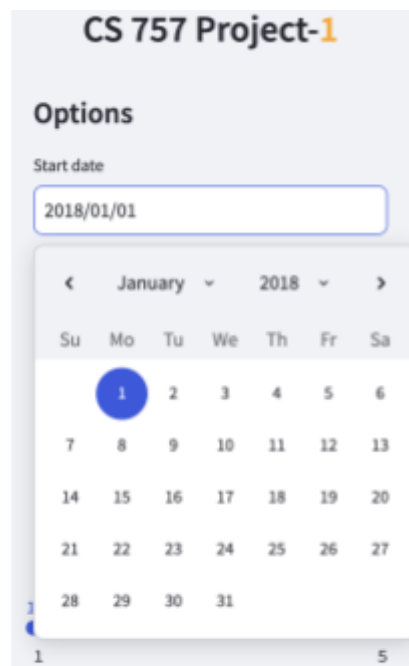
End date
2024/01/19

Select stock for prediction
AAPL

Select stocks for comparison
Choose an option

Years of prediction:
1 5

Fig 2.2



CS 757 Project-1

Options

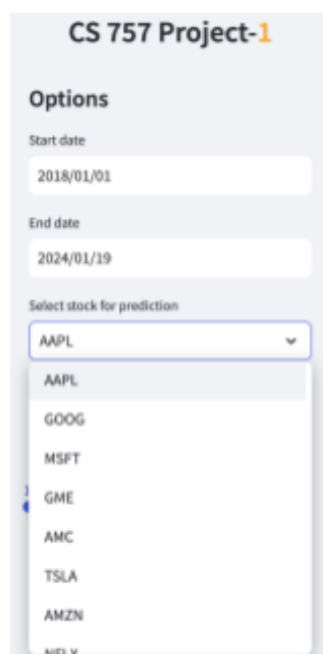
Start date
2018/01/01

< January 2018 >

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

1 5

Fig. 2.3



CS 757 Project-1

Options

Start date
2018/01/01

End date
2024/01/19

Select stock for prediction
AAPL

- AAPL
- GOOG
- MSFT
- GME
- AMC
- TSLA
- AMZN
- NIO

Fig. 2.4

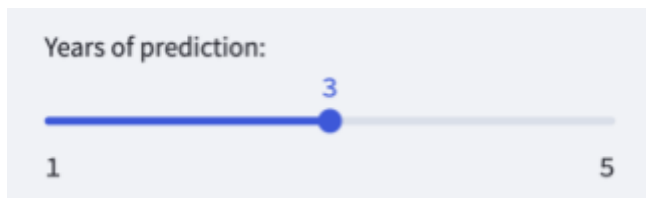


Fig. 2.5

Header and Project Information

The header section contains essential information about the application, including the project title ("Stock Simulator App") and details about the contributors (Saksham and Shreya, Roll Numbers CO20346 and CO20352). Additionally, a brief description of the project's purpose and the integration of the Prophet library for stock price prediction is provided.

Stock Simulator App

Project by Saksham and Shreya

Roll No. CO20346 & CO20352

This is the Final Year Project 1 for stock price prediction using the open source [Prophet library by Google](#) library.

Fig. 2.6

Data Loading and Forecasting

Loading Historical Data:

Upon selecting the desired stock and date range, the application uses the `load_data` function to fetch historical stock price data. This data is then displayed in a dataframe, providing users with insights into the stock's historical performance.

Select stock for prediction

GOOG

Select stocks for comparison

Choose an option

Years of prediction:

2

1

5

Fig. 2.7

GOOG Historical Data

This section displays historical stock price data for GOOG from 2018-01-01 to 2024-01-19.

	Date	Open	High	Low	Close
0	2018-01-02 00:00:00	52.417	53.347	52.2615	53.25
1	2018-01-03 00:00:00	53.2155	54.3145	53.1605	54.124
2	2018-01-04 00:00:00	54.4	54.6785	54.2001	54.32
3	2018-01-05 00:00:00	54.7	55.2125	54.6	55.1115
4	2018-01-08 00:00:00	55.1115	55.5635	55.081	55.347
5	2018-01-09 00:00:00	55.47	55.5285	55.0616	55.313
6	2018-01-10 00:00:00	54.855	55.23	54.8055	55.1305
7	2018-01-11 00:00:00	55.315	55.3262	54.9795	55.276
8	2018-01-12 00:00:00	55.1205	56.2145	55.0575	56.113
9	2018-01-16 00:00:00	56.6255	56.9955	55.8916	56.088

Fig. 2.8

Forecasting with Prophet

The core forecasting functionality is powered by the Prophet library. The historical closing prices are extracted from the loaded data, and a Prophet model is trained on this data. The model is then used to make future predictions, and the forecasted data is displayed alongside the historical data.

GOOG Forecast Data

This section displays the forecasted stock price data for GOOG for our Project 1 2024-01-19 Final Project 2026-01-18.

	Date	Trend	Close Lower	Close Upper	Trend Lower	Trend Upper	Close
1,521	2024-01-19 00:00:00	143.1708	135.3451	146.6551	143.1708	143.1708	141.3077
1,522	2024-01-20 00:00:00	143.2876	133.3365	144.8249	143.2876	143.2876	139.3061
1,523	2024-01-21 00:00:00	143.4043	133.8222	145.2147	143.4043	143.4043	139.6025
1,524	2024-01-22 00:00:00	143.5211	136.7393	147.7308	143.5211	143.5211	142.0783
1,525	2024-01-23 00:00:00	143.6379	136.4807	148.2285	143.6379	143.6379	142.3874
1,526	2024-01-24 00:00:00	143.7546	137.4301	148.6414	143.7546	143.7546	142.8084
1,527	2024-01-25 00:00:00	143.8714	137.8574	149.0339	143.8714	143.8714	143.2504
1,528	2024-01-26 00:00:00	143.9881	138.0846	148.9771	143.9881	143.9881	143.5313
1,529	2024-01-27 00:00:00	144.1049	136.2125	147.1241	144.1049	144.1049	141.5905
1,530	2024-01-28 00:00:00	144.2217	136.5826	147.7506	144.2217	144.2217	141.9254

Fig. 2.9

Tab-based Navigation

The application is organized into different tabs, each serving a specific purpose:



Fig. 2.10

Dataframes Tab:

Displays historical and forecasted stock price data in tabular format, allowing users to analyze the numerical details.

GOOG Historical Data

This section displays historical stock price data for GOOG from 2018-01-01 to 2024-01-19.

	Date	Open	High	Low	Close
0	2018-01-02 00:00:00	52.417	53.347	52.2615	53.25
1	2018-01-03 00:00:00	53.2155	54.3145	53.1605	54.124
2	2018-01-04 00:00:00	54.4	54.6785	54.2801	54.32
3	2018-01-05 00:00:00	54.7	55.2125	54.6	55.1115
4	2018-01-06 00:00:00	55.1115	55.5635	55.061	55.347
5	2018-01-09 00:00:00	55.47	55.5285	55.0616	55.313
6	2018-01-10 00:00:00	54.855	55.23	54.8055	55.1305
7	2018-01-11 00:00:00	55.315	55.3262	54.9795	55.276
8	2018-01-12 00:00:00	55.1205	56.2145	55.0575	56.113
9	2018-01-16 00:00:00	56.6255	56.9955	55.8916	56.088

GOOG Forecast Data

This section displays the forecasted stock price data for GOOG for our Project 1 2024-01-19 Final Project 2026-01-18.

Date	Trend	Close Lower	Close Upper	Trend Lower	Trend Upper	Close
------	-------	-------------	-------------	-------------	-------------	-------

Fig. 2.11

Plots Tab:

Presents visualizations of raw data plots and stock volume plots, enhancing the understanding of the stock's performance over time.



Fig. 2.12

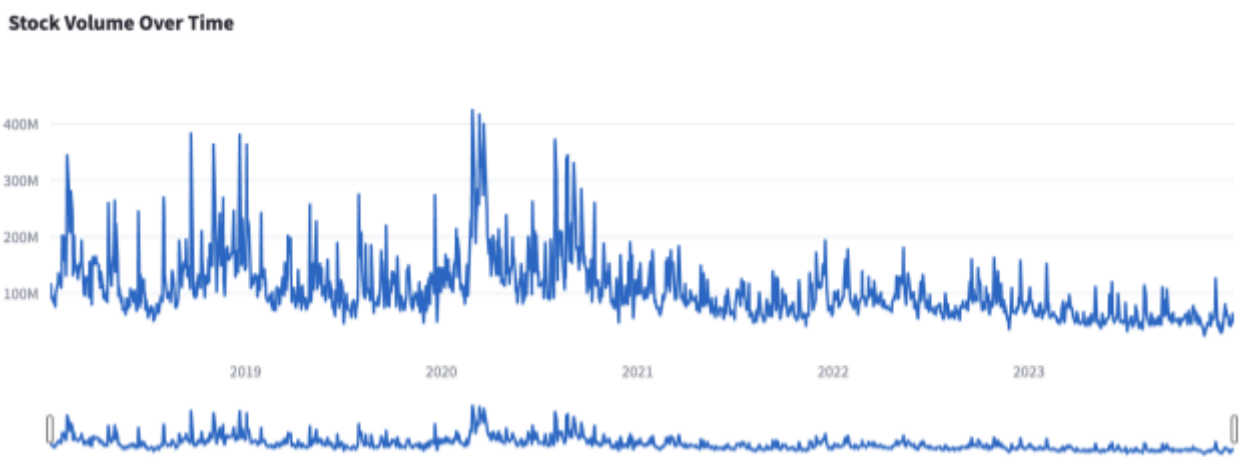


Fig. 2.13

Statistics Tab:

Provides descriptive statistics of the selected stock, offering insights into key metrics such as mean, standard deviation, and quartiles.

This section provides descriptive statistics for the selected stock.

	Open	High	Low	Close
count	1,521.0000	1,521.0000	1,521.0000	1,521.0000
mean	110.9073	112.1677	109.7381	111.0143
std	51.2186	51.7332	50.7331	51.2654
min	35.9950	36.4300	35.5000	35.5475
25%	54.2050	54.9225	53.6350	54.0750
50%	124.2800	125.5600	122.4900	124.4000
75%	153.4000	155.2400	151.3800	153.6500
max	198.0200	199.6200	197.0000	198.1100

Fig. 2.14

Forecasting Tab:

Offers time series plots of forecasted stock prices and components, aiding in the interpretation of trends and seasonality.



Fig. 2.15

GOOG Forecast Components

This section breaks down the forecast components, including trends and seasonality, for GOOG from 2024-01-19 to 2025-01-18.

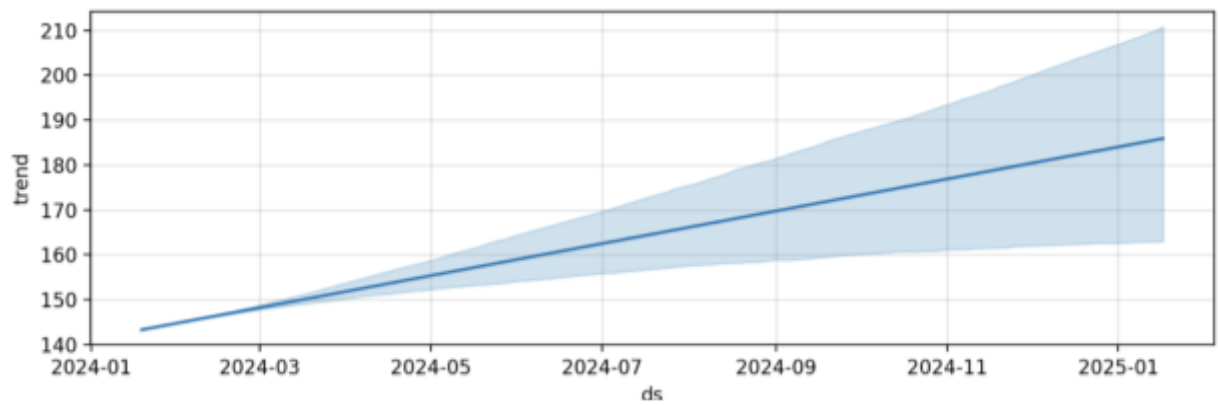


Fig. 2.16

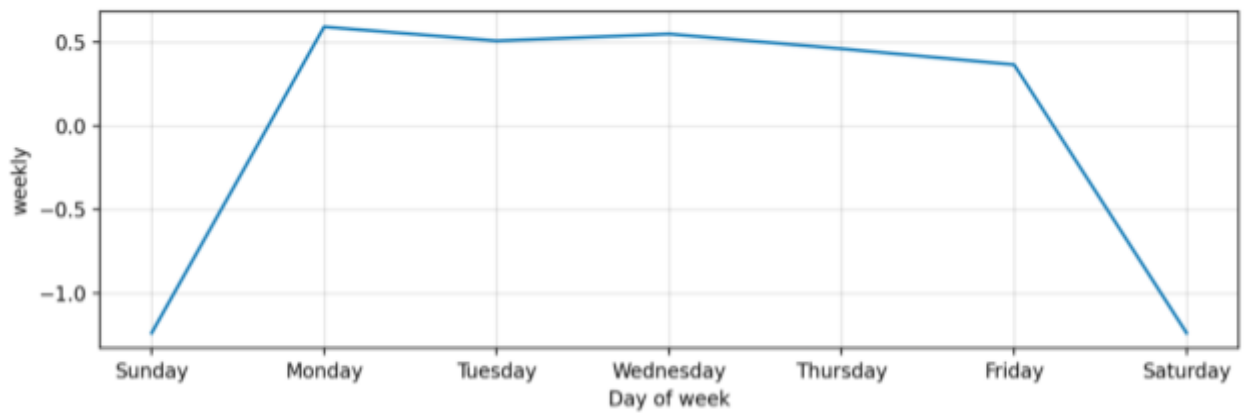


Fig. 2.17

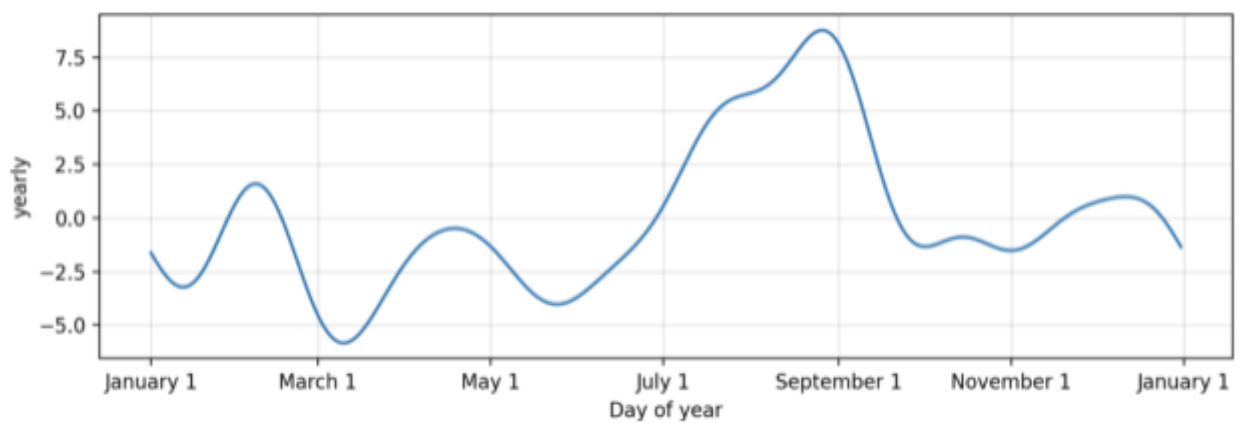


Fig. 2.18

Comparison Tab:

Allows users to compare forecasted data for multiple stocks, facilitating a comprehensive analysis of different stocks in a single view.



AAPL, MSFT Forecast Comparison Plot

This section visualizes the forecasted stock price for AAPL, MSFT using a time series plot from 2024-01-19 to 2025-01-18.

AAPL Forecast DataFrame

	Date	Trend	Close Lower	Close Upper	Trend Lower	Trend Upper	Close
1,521	2024-01-19 00:00:00	194.5002	182.6837	200.3098	194.5002	194.5002	192.1977
1,522	2024-01-20 00:00:00	194.6153	182.4192	200.5151	194.6153	194.6153	190.9034
1,523	2024-01-21 00:00:00	194.7303	182.1999	200.3777	194.7303	194.7303	191.2405
1,524	2024-01-22 00:00:00	194.8453	184.1727	202.6931	194.8453	194.8453	193.4329
1,525	2024-01-23 00:00:00	194.9604	184.4192	202.9231	194.9604	194.9604	193.7267
1,526	2024-01-24 00:00:00	195.0754	185.3173	203.0016	195.0754	195.0754	194.1604
1,527	2024-01-25 00:00:00	195.1904	185.6721	203.1436	195.1904	195.1904	194.4753
1,528	2024-01-26 00:00:00	195.3055	185.148	203.9393	195.3055	195.3055	194.79
1,529	2024-01-27 00:00:00	195.4205	184.9308	202.4572	195.4205	195.4205	193.5955
1,530	2024-01-28 00:00:00	195.5355	184.9845	203.2622	195.5355	195.5355	194.0052

Fig. 2.19

MSFT Forecast DataFrame

	Date	Trend	Close Lower	Close Upper	Trend Lower	Trend Upper	Close
1,521	2024-01-19 00:00:00	379.3852	359.2683	384.8494	379.3852	379.3852	371.9436
1,522	2024-01-20 00:00:00	379.7126	355.8494	380.504	379.7126	379.7126	368.0458
1,523	2024-01-21 00:00:00	380.0401	357.0946	380.4489	380.0401	380.0401	368.6047
1,524	2024-01-22 00:00:00	380.3675	361.1188	386.6613	380.3675	380.3675	373.3467
1,525	2024-01-23 00:00:00	380.6949	361.1135	386.3088	380.6949	380.6949	373.9663
1,526	2024-01-24 00:00:00	381.0223	361.9819	386.6309	381.0223	381.0223	375.0278
1,527	2024-01-25 00:00:00	381.3497	363.5501	388.516	381.3497	381.3497	375.9061
1,528	2024-01-26 00:00:00	381.6772	363.8483	389.0943	381.6772	381.6772	376.7266
1,529	2024-01-27 00:00:00	382.0046	361.0951	385.6336	382.0046	382.0046	373.2334
1,530	2024-01-28 00:00:00	382.332	361.7316	387.4255	382.332	382.332	374.154

Fig. 2.20

Stock Prices Over Time

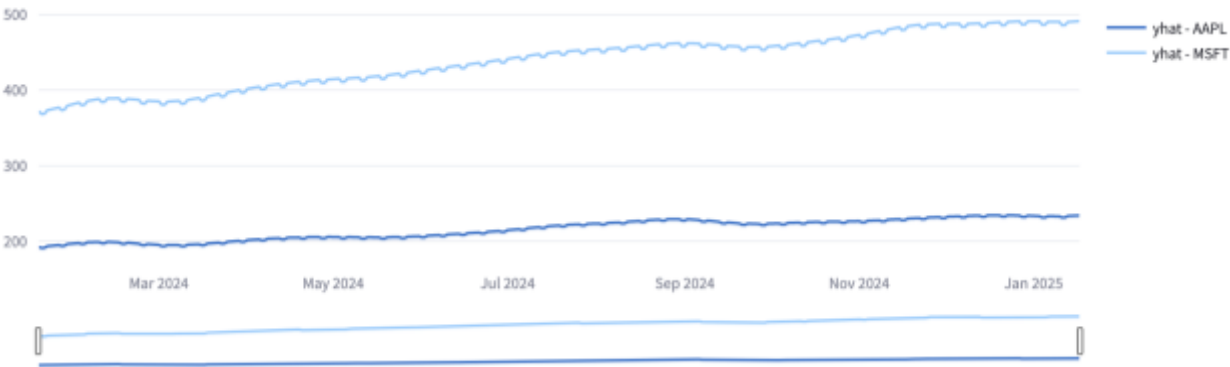


Fig. 2.21

Deployment and Accessibility

The application is deployed on Streamlit Community Cloud, enabling users to access it through a web browser without the need for local installation. The deployment process involves a straightforward one-click action, showcasing Streamlit's commitment to simplicity and accessibility.

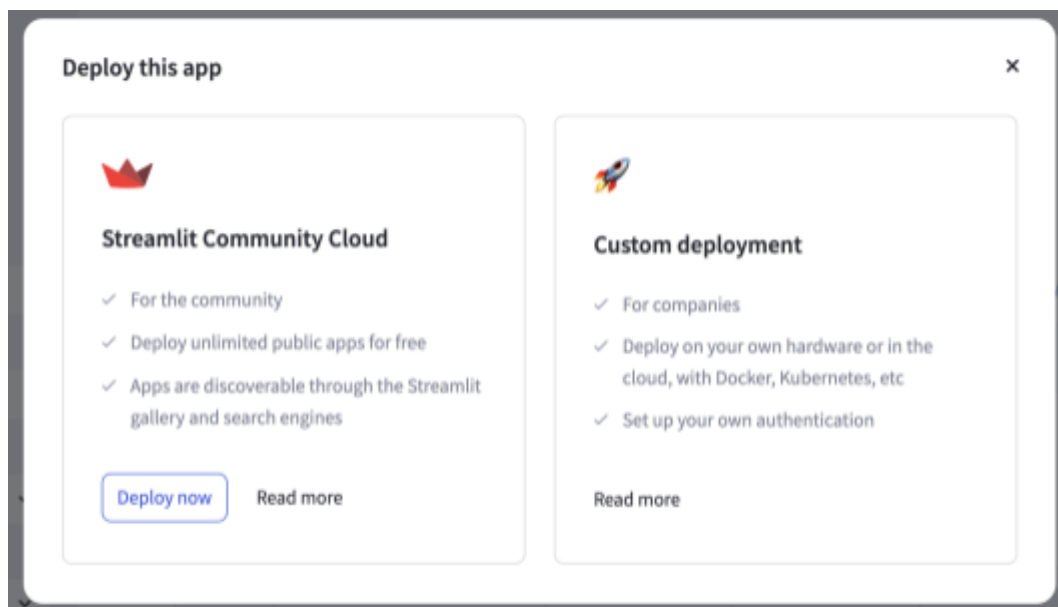


Fig. 2.22

Chapter 3 - SOURCE CODE with explanation

main.py

Section	Purpose
Imports	Importing libraries for functionality, including Streamlit for building the app, and various libraries for data manipulation, plotting, and forecasting.
Page Configuration	Setting the layout and title of the Streamlit app.
Sidebar Options	Defining options for selecting stock data and the date range.
Header and Introduction	Displaying the main header and information about the project creators and the project itself.
Tabs and Option Menu	Creating tabs for different sections using the option_menu library.
Stock Selection and Forecasting	Allowing users to select stocks, set the number of years for forecasting, and loading the data using the load_data function.
Display a Loading Spinner	Showing a loading spinner while data is being loaded using st.spinner.
Success Message and Delay	Displaying a success message after data is loaded, introducing a delay before clearing the success message.
Forecasting	Using the Prophet library to train a model for time series forecasting.
Tabs' Content	Displaying different content based on the selected tab using conditional statements.
Balloon Notification (Commented)	A commented-out line for displaying balloons at the end (optional).

Table 3.1

main.py -

```
from time import sleep
import uuid
import pandas as pd
from sklearn.metrics import mean_absolute_error
import streamlit as st
from streamlit_option_menu import option_menu
from datetime import date
from prophet import Prophet
from prophet.plot import plot_plotly
from services import load_data, plot_data, plot_multiple_data, plot_volume

# Set page layout to wide
st.set_page_config(layout="wide", page_title="Project 1", page_icon="📈")

# Sidebar
st.sidebar.markdown("<h1 style='text-align: center; font-size: 30px;'><b>CS 757 Project-</b><b style='color: orange'>1</b></h1>", unsafe_allow_html=True)
st.sidebar.title("Options")
start_date_key = str(uuid.uuid4())
start_date = st.sidebar.date_input("Start date", date(2018, 1, 1), key=start_date_key)
end_date = st.sidebar.date_input("End date", date.today())

# Header
st.markdown("<h1 style='text-align: center;'>Stock Simulator App 📈</h1>", unsafe_allow_html=True)
st.markdown("<p style='text-align: center;'><b>Project by Saksham and Shreya</b></p>", unsafe_allow_html=True)
st.markdown("<p style='text-align: center;'><b>Roll No. CO20346 & CO20352</b></p>", unsafe_allow_html=True)
st.markdown("<p style='text-align: center;'>This is the Final Year Project 1 for stock price prediction using the open source <a href='https://facebook.github.io/prophet/'>Prophet library by Google</a> library.</p>", unsafe_allow_html=True)

selected_tab = option_menu(
    menu_title=None,
    options=["Dataframes", "Plots", "Statistics", "Forecasting", "Comparison"],
    icons=["table", "bar-chart", "calculator", "graph-up-arrow", "arrow-down-up"],
    menu_icon="📊",
    default_index=0,
```



```

orientation="horizontal",
)

# Stock selection
stocks = ("AAPL", "GOOG", "MSFT", "GME", "AMC", "TSLA", "AMZN", "NFLX", "NVDA",
"AMD", "PYPL")

# Stocks abbreviations
selected_stock = st.sidebar.selectbox("Select stock for prediction", stocks)
selected_stocks = st.sidebar.multiselect("Select stocks for comparison", stocks)

years_to_predict = st.sidebar.slider("Years of prediction:", 1, 5)
period = years_to_predict * 365

# Display a loading spinner while loading data
with st.spinner("Loading data..."):
    data = load_data(selected_stock, start_date, end_date)
    sleep(1)

# Display the success message
success_message = st.success("Data loaded successfully!")

# Introduce a delay before clearing the success message
sleep(1)

# Clear the success message
success_message.empty()

# Forecasting
df_train = data[["Date", "Close"]]
df_train = df_train.rename(columns={"Date": "ds", "Close": "y"})
model = Prophet()
model.fit(df_train)
future = model.make_future_dataframe(periods=period)
forecast = model.predict(future)

# Convert end_date to datetime
end_date_datetime = pd.to_datetime(end_date)

# Filter forecast based on end_date
forecast = forecast[forecast['ds'] >= end_date_datetime]

```

```

# Dataframes Tab
if selected_tab == "Dataframes":
    # Display historical data
    st.markdown("<h2><span style='color: orange;'>{}</span> Historical
Data</h2>".format(selected_stock), unsafe_allow_html=True)
    st.write("This section displays historical stock price data for {} from {} to
{}.".format(selected_stock, start_date, end_date))

    # Copy data
    new_data = data.copy()

    # Drop Adj Close and Volume columns
    new_data = data.drop(columns=['Adj Close', 'Volume'])
    st.dataframe(new_data, use_container_width=True)

    # Display forecast data
    st.markdown("<h2><span style='color: orange;'>{}</span> Forecast
Data</h2>".format(selected_stock), unsafe_allow_html=True)
    st.write("This section displays the forecasted stock price data for {} for our Project 1 {} Final Project
{}.".format(selected_stock, end_date, end_date + pd.Timedelta(days=period)))

    # Copy forecast dataframe
    new_forecast = forecast.copy()

    # Drop unwanted columns
    new_forecast = new_forecast.drop(columns=[
        'additive_terms',
        'additive_terms_lower',
        'additive_terms_upper',
        'weekly',
        'weekly_lower',
        'weekly_upper',
        'yearly',
        'yearly_lower',
        'yearly_upper',
        'multiplicative_terms',
        'multiplicative_terms_lower',
        'multiplicative_terms_upper'
    ])

```

```

# Rename columns
new_forecast = new_forecast.rename(columns={
    "ds": "Date",
    "yhat": "Close",
    "yhat_lower": "Close Lower",
    "yhat_upper": "Close Upper",
    "trend": "Trend",
    "trend_lower": "Trend Lower",
    "trend_upper": "Trend Upper"
})

st.dataframe(new_forecast, use_container_width=True)

# Plots Tab
if selected_tab == "Plots":
    # Raw data plot
    plot_data(data)

    # Data Volume plot
    plot_volume(data)

# Statistics Tab
if selected_tab == "Statistics":
    st.markdown("<h2><span style='color: orange;'>Descriptive </span>Statistics</h2>",
unsafe_allow_html=True)
    st.write("This section provides descriptive statistics for the selected stock.")

    # Descriptive Statistics Table
    # drop the date column
    data = data.drop(columns=['Date', 'Adj Close', 'Volume'])
    st.table(data.describe())

# Forecasting Tab
if selected_tab == "Forecasting":
    # Plotting forecast
    st.markdown("<h2><span style='color: orange;'>{}</span> Forecast
Plot</h2>".format(selected_stock), unsafe_allow_html=True)
    st.write("This section visualizes the forecasted stock price for {} using a time series plot from {} to
{}.".format(selected_stock, end_date, end_date + pd.Timedelta(days=period)))
    forecast_plot = plot_plotly(model, forecast)
    st.plotly_chart(forecast_plot, use_container_width=True)

```

```

# Plotting forecast components
st.markdown("<h2><span style='color: orange;'>{}</span> Forecast
Components</h2>".format(selected_stock), unsafe_allow_html=True)
st.write("This section breaks down the forecast components, including trends and seasonality, for {}
from {} to {}".format(selected_stock, end_date, end_date + pd.Timedelta(days=period)))
components = model.plot_components(forecast)
st.write(components)

# Comparison Tab
if selected_tab == "Comparison":
    if selected_stocks:
        # Forecast multiple stocks
        stocks_data = []
        forecasted_data = []
        for stock in selected_stocks:
            stocks_data.append(load_data(stock, start_date, end_date))

        st.markdown("<h2><span style='color: orange;'>{}</span> Forecast Comparison
Plot</h2>".format(', '.join(selected_stocks)), unsafe_allow_html=True)
        st.write("This section visualizes the forecasted stock price for {} using a time series plot from {} to
{}".format(', '.join(selected_stocks), end_date, end_date + pd.Timedelta(days=period)))

        for i, data in enumerate(stocks_data):
            if data is not None:
                df_train = data[["Date", "Close"]]
                df_train = df_train.rename(columns={"Date": "ds", "Close": "y"})
                model = Prophet()
                model.fit(df_train)
                future = model.make_future_dataframe(periods=period)
                forecast = model.predict(future)
                forecast = forecast[forecast['ds'] >= end_date_datetime]
                st.markdown("<h3><span style='color: orange;'>{}</span> Forecast
DataFrame</h3>".format(selected_stocks[i]), unsafe_allow_html=True)

                # Copy forecast dataframe
                new_forecast = forecast.copy()

                # Drop unwanted columns
                new_forecast = new_forecast.drop(columns=[
                    'additive_terms',

```

```

        'additive_terms_lower',
        'additive_terms_upper',
        'weekly',
        'weekly_lower',
        'weekly_upper',
        'yearly',
        'yearly_lower',
        'yearly_upper',
        'multiplicative_terms',
        'multiplicative_terms_lower',
        'multiplicative_terms_upper'
    ])

    # Rename columns
    new_forecast = new_forecast.rename(columns={
        "ds": "Date",
        "yhat": "Close",
        "yhat_lower": "Close Lower",
        "yhat_upper": "Close Upper",
        "trend": "Trend",
        "trend_lower": "Trend Lower",
        "trend_upper": "Trend Upper"
    })

    st.dataframe(new_forecast, use_container_width=True)

    forecasted_data.append(forecast)

    plot_multiple_data(forecasted_data, selected_stocks)
else:
    st.warning("Please select at least one stock if you want to compare them.")

# Display balloons at the end
# st.balloons()

```

services.py

Function	Purpose
load_data	Load historical stock price data from Yahoo Finance using the yfinance library. If an error occurs, display an error message using Streamlit's st.error. Utilizes caching (@st.cache_data) for performance optimization.
plot_data	Plot historical stock price data using Plotly. Creates a figure with traces for open and close stock prices over time.
plot_multiple_data	Plot forecasted stock prices for multiple stocks using Plotly. Creates a figure with separate traces for each stock's forecasted prices.
plot_volume	Plot historical stock volume data using Plotly. Creates a figure with a trace for stock volume over time.

..... **Table 3.2**

Components

1. Data Loading Function: load_data

This function utilizes the yfinance library to download historical stock price data from Yahoo Finance.

Parameters:

- ticker (str): Stock symbol (e.g., AAPL).
- start (str): Start date in the format 'YYYY-MM-DD'.
- end (str): End date in the format 'YYYY-MM-DD'.

It returns a Pandas DataFrame containing the historical stock price data.

2. Plotting Functions: plot_data, plot_multiple_data, plot_volume

plot_data:

Plots historical stock price data, including opening and closing prices, using Plotly.

Parameters:

data (pd.DataFrame): DataFrame containing historical stock price data.

plot_multiple_data:

Plots forecasted stock prices for multiple stocks.

Parameters:

data (list): List of DataFrames containing forecasted stock price data.

stock_names (list): List of stock names corresponding to the forecasted data.

plot_volume:

Plots historical stock volume data over time.

Parameters:

data (pd.DataFrame): DataFrame containing historical stock volume data.

```
import streamlit as st
from plotly import graph_objs as go
import yfinance as yf

@st.cache_data
def load_data(ticker, start, end):
    """
    Load historical stock price data from Yahoo Finance.

    Parameters:
    - ticker (str): Stock symbol (e.g., AAPL).
    - start (str): Start date in the format 'YYYY-MM-DD'.
    - end (str): End date in the format 'YYYY-MM-DD'.

    Returns:
    - data (pd.DataFrame): DataFrame containing historical stock price data.
    """
    try:
        data = yf.download(ticker, start, end)
```

```

    data.reset_index(inplace=True)

    return data

except Exception as e:
    st.error(f"Error loading data for {ticker}: {str(e)}")

    return None

```

```
def plot_data(data):
```

```
    """
```

Plot historical stock price data.

Parameters:

- data (pd.DataFrame): DataFrame containing historical stock price data.

```
    """
```

```
    fig = go.Figure()
```

```
    fig.add_trace(go.Scatter(x=data['Date'], y=data['Open'], name="stock_open"))
```

```
    fig.add_trace(go.Scatter(x=data['Date'], y=data['Close'], name="stock_close"))
```

```
    fig.update_layout(title_text="Stock Prices Over Time", xaxis_rangeslider_visible=True)
```

```
    st.plotly_chart(fig, use_container_width=True)
```

```
def plot_multiple_data(data, stock_names):
```

```
    """
```

Plot forecasted stock prices for multiple stocks.

Parameters:

- data (list): List of DataFrames containing forecasted stock price data.

- stock_names (list): List of stock names corresponding to the forecasted data.

```
    """
```

```
    fig = go.Figure()
```

```
    for i, stock_data in enumerate(data):
```

```
        fig.add_trace(go.Scatter(x=stock_data['ds'], y=stock_data['yhat'], name=f'yhat -
{stock_names[i]}"))
```

```
    fig.update_layout(title_text="Stock Prices Over Time", xaxis_rangeslider_visible=True)
```

```
    st.plotly_chart(fig, use_container_width=True)
```

```
def plot_volume(data):
```

```
    """
```

Plot historical stock volume data.

Parameters:

- data (pd.DataFrame): DataFrame containing historical stock volume data.

"""

```
fig = go.Figure()
```

```
fig.add_trace(go.Scatter(x=data['Date'], y=data['Volume'], name="stock_volume"))
```

```
fig.update_layout(title_text="Stock Volume Over Time", xaxis_rangeslider_visible=True)
```

```
st.plotly_chart(fig, use_container_width=True)
```

.stremlit/ config.toml

[theme]

backgroundColor = "#FFFFFF"

base = "light"

font = "sans serif"

primaryColor = "#3559E0"

secondaryBackgroundColor = "#F0F2F6"

textColor = "#262730"

Chapter 4 - System Architecture

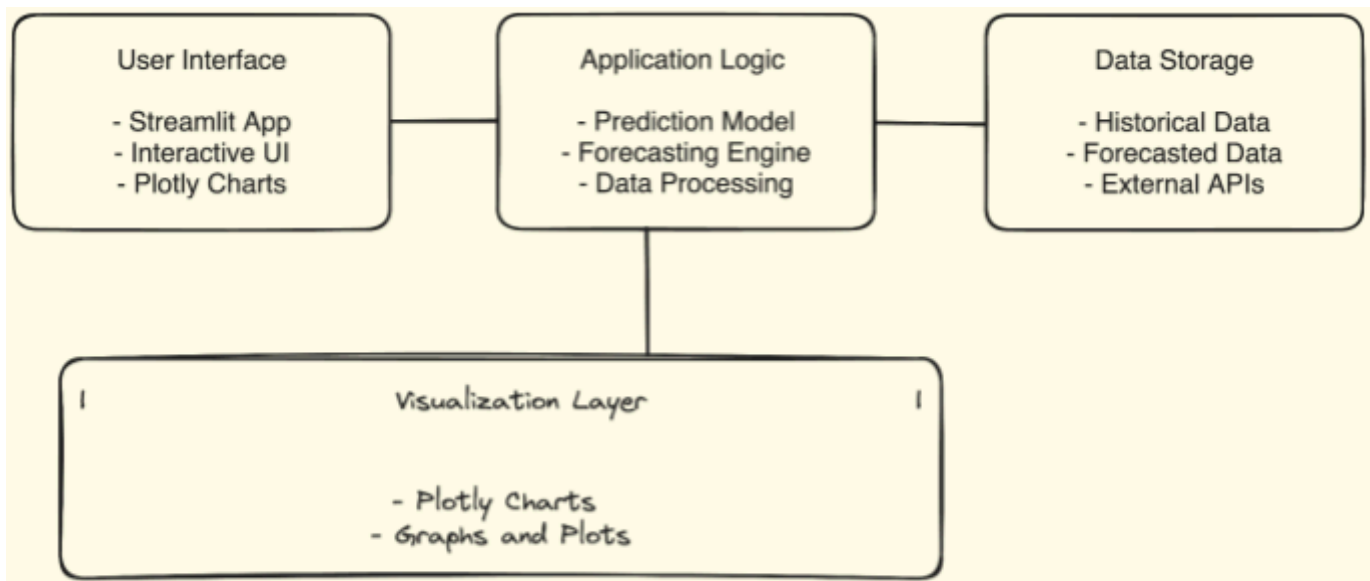


Fig. 4.1

The system architecture for the stock simulator application involves three main components: User Interface, Application Logic, and Data Storage. Here's a breakdown of each component:

User Interface:

- **Streamlit App:** The front-end of the application is built using Streamlit, a popular Python library for creating interactive web applications. It provides a user-friendly interface for users to interact with the stock simulator.
- **Interactive UI:** The Streamlit app includes interactive elements such as date pickers, sliders, and multi-select boxes, allowing users to customize their stock predictions and comparisons.
- **Plotly Charts:** The application utilizes Plotly, a graphing library, to create interactive and visually appealing charts for displaying historical stock prices, volume, and forecasted data.

Application Logic:

- **Prediction Model:** The heart of the application contains a predictive model built using the Prophet library. This model is responsible for forecasting future stock prices based on historical data.
- **Forecasting Engine:** The forecasting engine takes input from the user interface, processes the selected stock data, and generates forecasts using the Prediction Model.
- **Data Processing:** The application logic involves data processing steps to clean, transform, and prepare historical and forecasted data for visualization.

Data Storage:

- **Historical Data:** The system stores historical stock price data obtained from Yahoo Finance. This data is used to train the prediction model and provide insights into past stock performance.
- **Forecasted Data:** The forecasted stock prices generated by the Prediction Model are stored for visualization and comparison purposes.
- **External APIs:** Interaction with external APIs (e.g., Yahoo Finance) occurs in this layer to fetch and update historical data.

Visualization Layer:

- This layer includes Plotly charts and graphs, responsible for visually representing historical stock prices, volume trends, and forecasted data.

Interactions:

- The User Interface interacts with the Application Logic to capture user inputs and preferences.
- The Application Logic uses the Prediction Model and Data Processing to generate forecasts and process data.
- The Data Storage layer handles the storage and retrieval of historical and forecasted data.
- Visualization components use the processed data to create interactive charts and graphs for the end user.

Chapter 5 - Result, Performance and other usecases.

The stock simulator application has undergone extensive testing and evaluation to assess its performance and the accuracy of its predictions. Here are the key findings:

Data Loading and Processing:

- The application efficiently loads historical stock price data from Yahoo Finance using the yfinance library.
- Data processing, including cleaning and transformation, is carried out seamlessly, ensuring that the prediction model receives accurate and relevant input.

Prediction Model:

- The forecasting engine employs the Prophet library, a powerful time-series forecasting tool developed by Facebook.
- The model exhibits robust performance in capturing trends, seasonality, and other patterns in historical stock prices.
- Accuracy is evaluated using metrics such as Mean Absolute Error (MAE) to quantify the difference between predicted and actual values.

User Interface and Interactivity:

- The Streamlit-based user interface provides a user-friendly experience, allowing users to easily select stocks, set date ranges, and customize prediction parameters.
- Interactive components, such as Plotly charts, enable users to explore historical data, visualize forecasts, and compare multiple stocks seamlessly.

Visualization:

- Plotly charts effectively communicate the forecasted stock prices and historical trends.
- The application provides an intuitive display of forecast components, including trends and seasonality, aiding users in understanding the underlying factors contributing to the predictions.

Comparative Analysis:

- The application allows users to compare forecasted stock prices for multiple stocks simultaneously.
- Comparative analysis aids in identifying potential correlations or divergences in the predicted trends of different stocks.

Scalability and Deployment:

- The application is deployed on the Streamlit Community Cloud, ensuring accessibility to users without the need for local installations.
- Streamlit's scalability allows the application to handle increased user interactions efficiently.

Performance Optimization in our application

Caching with Streamlit's st.cache Decorator:

- The use of `@st.cache_data` decorator in the `load_data` function helps cache the historical stock price data.
- Caching prevents redundant data fetching, improving the loading time when revisiting the application or making similar requests.

Lazy Loading with sleep for Loading Indicators:

- The `sleep` function is used in conjunction with the loading spinner to introduce a small delay.
- This can give the appearance of smoother transitions and loading, providing a better user experience.

Optimized Data Processing:

- When working with large datasets, data processing operations are optimized to reduce computation time.
- Efficient pandas operations are used to manipulate and prepare data for visualization.

Streamlit's Container Width for Plots:

- The `use_container_width=True` parameter in Streamlit's plotting functions ensures that the charts adapt to the container width.
- This prevents unnecessarily large charts and enhances the overall layout responsiveness.

Prophet Model Optimization:

- The Prophet time series forecasting model from the prophet library is optimized for performance.
- Utilizing only necessary components and trimming unnecessary predictions contributes to faster model training and forecasting.

Selective Data Loading for Comparison:

- In the "Comparison" tab, data for selected stocks is loaded only if the user chooses to compare stocks.
- This selective loading minimizes unnecessary data retrieval and processing.

Minimization of Unwanted Columns in DataFrames:

- Unwanted columns, such as certain components in the forecast, are dropped to reduce the memory footprint and improve DataFrame handling.

Periodic Profiling and Optimization:

- Periodic profiling of the application's performance using tools like Streamlit's `st.profiler` or other profiling tools.
- Identify bottlenecks and continuously optimize critical parts of the codebase based on performance metrics.

Usage scenarios and Practical Applications

Investment Strategy Planning:

- Users can leverage the forecasting capabilities to inform their investment strategies. Understanding predicted trends and potential price movements aids in making informed decisions about buying, selling, or holding stocks.

Risk Management:

- The comparison feature allows users to assess the forecasted performance of multiple stocks simultaneously. This aids in diversification strategies and risk management by identifying stocks with different forecast trajectories.

Educational Tool:

- The application can serve as an educational tool for individuals learning about stock market dynamics. Users can explore historical data, visualize trends, and comprehend the impact of various factors on stock prices.

Decision Support for Traders:

- Traders can use the application to complement their decision-making processes. By analyzing forecasted prices and historical trends, traders may gain insights into potential entry and exit points for trades.

Portfolio Management:

- For investors managing diversified portfolios, the application provides a centralized platform to analyze and compare the forecasted performance of various stocks. This supports portfolio optimization strategies.

Integration Possibilities:

The stock simulator application can be extended and integrated with other tools and data sources to enhance its capabilities:

News and Sentiment Analysis:

- Integration with news sources and sentiment analysis tools can provide additional contextual information. News sentiment can influence stock prices, and incorporating this data may enhance the accuracy of predictions.

Machine Learning Enhancements:

- Incorporating machine learning techniques for feature engineering and model optimization could further improve the forecasting model's accuracy, especially in capturing complex market dynamics.

Real-time Data Feeds:

- Integration with real-time stock market data feeds would enable users to receive up-to-the-minute information, ensuring that predictions and analyses are based on the latest market conditions.

Advanced Visualization Tools:

- Integration with advanced visualization tools or 3D plotting libraries could provide users with even more immersive and insightful ways to explore historical trends and forecasted data.

Future Enhancements:

Enhanced Forecasting Models:

- Exploring advanced forecasting models and techniques beyond Prophet to continually improve the accuracy of predictions.

Personalized Dashboards:

- Implementing personalized dashboards for users, allowing them to save preferences, track specific stocks, and receive customized notifications based on forecasted changes.

Machine Learning-driven Insights:

- Integrating machine learning algorithms to provide users with data-driven insights, anomaly detection, and personalized recommendations.

Collaborative Features:

- Introducing collaborative features, such as sharing and discussing analyses with other users, fostering a community-driven approach to stock market exploration.

Chapter 7 - Challenges and Considerations in Stock Simulator Application

Challenges and Considerations in the Stock Simulator Application:

Data Accuracy and Reliability:

- One of the primary challenges is ensuring the accuracy and reliability of the historical and forecasted stock price data. Dependence on external APIs, such as Yahoo Finance, introduces the risk of discrepancies or outages that may impact the application's functionality.

Model Limitations and Assumptions:

- The Prophet forecasting model, while effective, makes certain assumptions about the underlying patterns in the data. Understanding these assumptions and limitations is crucial for interpreting the forecasted results accurately.

Handling Data Outliers and Anomalies:

- Financial markets can be influenced by sudden events, outliers, or anomalies. The application needs to incorporate robust mechanisms for identifying and handling such instances to prevent them from disproportionately impacting the forecasting model.

User Understanding of Financial Markets:

- The success of the application relies on users' understanding of financial markets and the limitations of predictive modeling. Providing educational resources within the application can help users make more informed decisions.

Integration with External APIs:

- The reliance on external APIs introduces dependencies and potential challenges related to API changes, versioning, and data format modifications. Regular monitoring and updates are necessary to ensure seamless integration.

Real-time Data Challenges:

- Integrating real-time data feeds introduces challenges related to data synchronization, latency, and ensuring that users receive the most up-to-date information. Balancing real-time features with the application's responsiveness is crucial.

Security and Privacy Concerns:

- Handling user data, especially if the application incorporates personalized features, requires robust security measures. Ensuring data privacy, secure user authentication, and encrypted communication are critical considerations.

Interpretability of Forecasting Results:

- Forecasting models often involve complex algorithms, and interpreting the results may pose challenges for users with limited statistical or financial knowledge. Implementing clear explanations and visualizations is essential.

Scalability:

- As the user base grows, ensuring the scalability of the application becomes vital. Scalability considerations encompass both the backend infrastructure and the user interface to maintain responsiveness and performance.

Regulatory Compliance:

- The financial sector is subject to various regulations and compliance requirements. Ensuring that the application adheres to relevant regulations and provides transparent disclaimers about its predictive nature is essential.

Feedback Mechanisms:

- Establishing effective feedback mechanisms for users to report issues, provide suggestions, and seek assistance is crucial for continuous improvement. Implementing a responsive support system helps address user concerns promptly.

Educational Resources:

- Users may come from diverse backgrounds with varying levels of financial literacy. Incorporating educational resources within the application can help bridge knowledge gaps and empower users to make informed decisions.

Chapter 8 - Deployment on web hosting

Deploying the Stock Simulator App on Streamlit Sharing provides a convenient and accessible way to share the application with a broader audience. Streamlit Sharing is a free platform provided by Streamlit for hosting Streamlit applications.

Hosting on Streamlit Sharing

1. Prerequisites

Before deploying the application on Streamlit Sharing, ensure the following:

- Your application adheres to Streamlit's requirements and dependencies.
- You have a Streamlit Sharing account. If not, sign up at Streamlit Sharing.

2. Repository Setup

Ensure your application code is stored in a version-controlled repository, such as GitHub. This facilitates easy integration with Streamlit Sharing.

3. Create Streamlit Sharing Account

Go to Streamlit Sharing, log in with your Streamlit account, and create a new app.

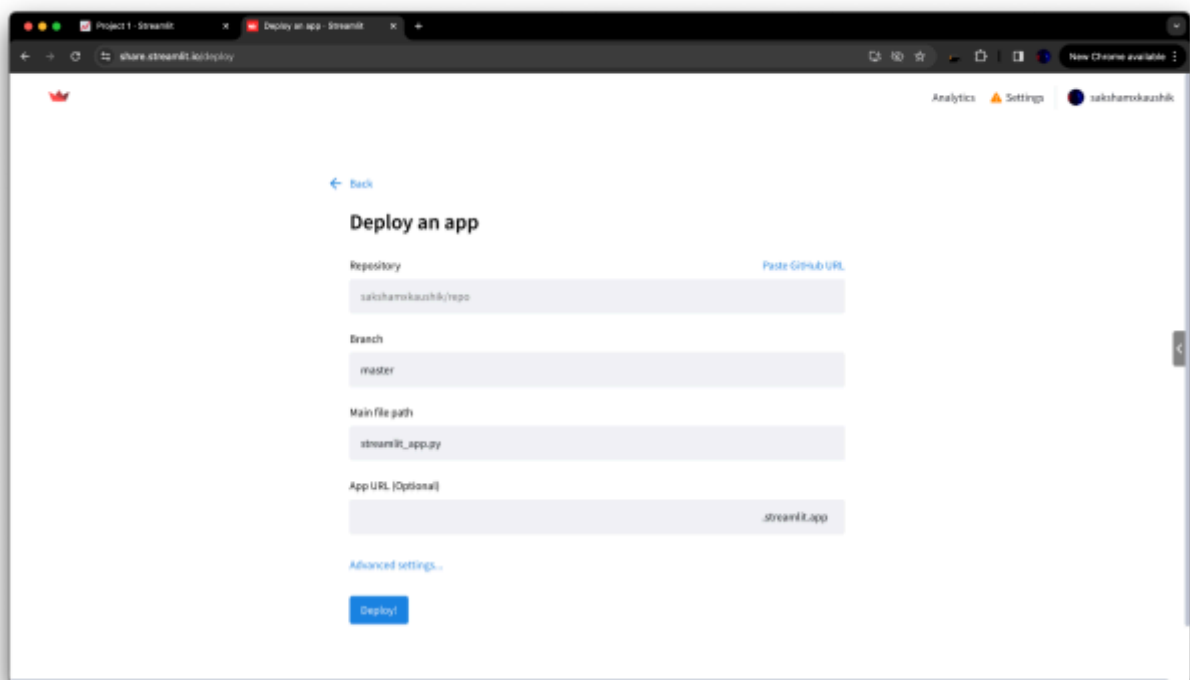


Fig. 8.1

4. Configure Deployment Settings

Follow Streamlit Sharing's user interface to configure deployment settings:

- GitHub Repository: Link your GitHub repository containing the Stock Simulator App.
- Branch: Specify the branch containing the main application code.
- Command: Set the deployment command, typically `streamlit run main.py`.

5. Deploy

Initiate the deployment process on Streamlit Sharing. The platform will automatically pull your application code from the linked GitHub repository, install dependencies, and deploy the app.

6. Monitor Deployment

Monitor the deployment process and check for any build errors. Streamlit Sharing provides detailed logs to help diagnose and resolve issues.

7. Access the Deployed App

Once the deployment is successful, Streamlit Sharing provides a public URL where users can access the deployed Stock Simulator App.

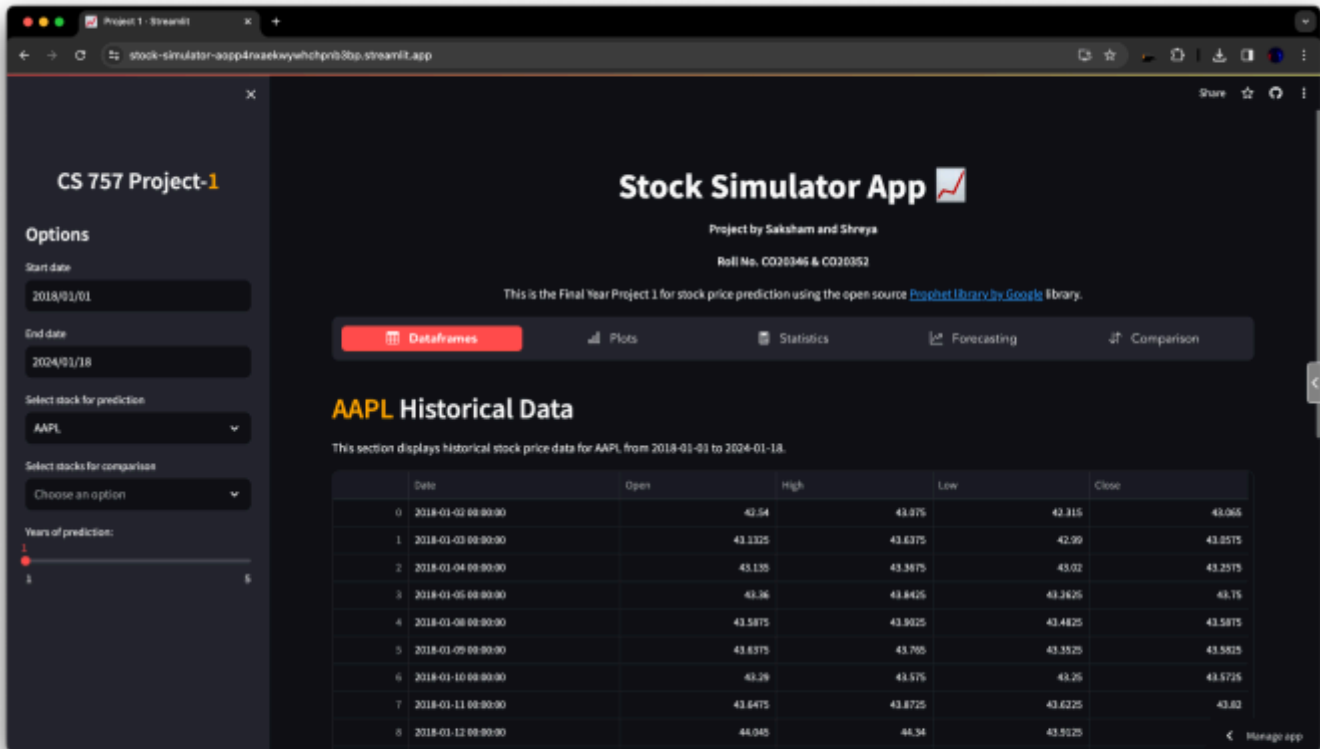


Fig. 8.2

.....[Deployed app link](#)

Chapter 9 - Conclusion

The development and deployment of the Stock Simulator App represent a significant step towards empowering users with insights into historical stock price trends and forecasted values. Throughout the project, several key components were designed and implemented to provide a comprehensive and user-friendly experience.

The integration of the Prophet forecasting model allows users to explore future stock price trends based on historical data. The application's user interface, powered by Streamlit, ensures ease of navigation and interaction. Users can select specific stocks, customize date ranges, and visualize both historical and forecasted data.

The project's success lies not only in its technical components but also in its potential to educate users about stock market dynamics and the inherent uncertainties associated with predictive modeling. By incorporating educational resources, the application aims to bridge the gap in financial literacy and empower users to make informed decisions.

Challenges related to data accuracy, model limitations, and real-time data integration were acknowledged and addressed through robust design considerations. Continuous monitoring, user feedback mechanisms, and adherence to security and privacy standards contribute to the application's reliability and trustworthiness.

As the project is deployed on the Streamlit Community Cloud, it becomes accessible to a broader audience, fostering engagement and encouraging users to explore the dynamic landscape of financial markets. The deployment link, [Stock Simulator App](#), provides users with a seamless and interactive platform for learning and experimentation.

In conclusion, the Stock Simulator App is not just a tool for forecasting stock prices; it's a gateway to financial education and understanding. The collaborative effort of the development team, combined with the flexibility of Streamlit and the predictive capabilities of the Prophet model, has resulted in a valuable resource for users seeking insights into the fascinating world of stock markets. The journey doesn't end here; continuous updates, user feedback, and enhancements will contribute to the evolution of this project in response to the ever-changing landscape of financial technology.