Stock Prophet

PROJECT REPORT

Submitted by

Shail

(2210993837-G2)

BE-CSE (Artificial Intelligence)

Guided by

Dr. Rajan Kumar

CHITKARA UNIVERSITY INSITUTE OF ENGINEERING & TECHNOLOGY

CHITKARA UNIVERSITY, RAJPURA

CONTENTS

ACKNOWLEDGEMENT

With immense please **I, Mr. Shail Sharma** presenting the "Stock Prophet" project report as part of the curriculum of 'BE-CSE (AI)'.

I would like to express my sincere thanks to **Dr. Rajan Kumar** for their valuable guidance and support in completing my project.

I would also like to express my gratitude towards our dean **Dr. Sushil Kumar Narang** for giving me this great opportunity to do a project on **this website**. Without their support and suggestions, this project would not have been completed.

Signature……….

Name: Shail Sharma

Roll No: 2210993837

ABSTRACT

Welcome to StockProphet, a groundbreaking project at the forefront of financial innovation, poised to transform the way investors analyze and predict stock market trends. By seamlessly integrating Django, the robust web framework, with live data streams from Yahoo Finance, StockProphet delivers a cutting-edge platform for real-time stock prediction and analysis. Through sophisticated machine learning algorithms, statistical models, and intuitive user interfaces, StockProphet empowers investors with actionable insights, enabling them to navigate the complexities of the stock market with confidence and precision. With features such as live data integration, predictive analytics, user authentication, interactive visualizations, and responsive design, StockProphet represents a fusion of advanced technology and financial expertise, offering users the tools they need to thrive in today's dynamic financial landscape.

Enter the world of StockProphet, a revolutionary project that marries the power of Django, a versatile web framework, with the real-time data streams from Yahoo Finance to create a dynamic platform for stock market prediction. In an era where information is currency, StockProphet stands as a beacon of innovation, offering investors a sophisticated toolset to navigate the ever-changing landscape of financial markets. Through advanced machine learning algorithms, statistical analysis, and user-friendly interfaces, StockProphet equips users with the insights they need to make informed investment decisions and stay ahead of market trends. With its focus on live data integration, predictive analytics, user-centric features, and responsive design, StockProphet redefines the way investors interact with stock market data, empowering them to unlock new opportunities and achieve financial success in today's fast-paced environment.

CHAPTER 1

Introduction

1.1 OBJECTIVES:

- The Develop a user-friendly web application that integrates seamlessly with Yahoo Finance's live data feeds.
- Implement Django's robust authentication system to ensure secure user access and data privacy.
- Utilize machine learning algorithms to analyze historical stock data and identify trends and patterns.
- Provide predictive analytics features to forecast future stock movements with accuracy.
- Enable users to create and customize portfolios based on their investment preferences and risk tolerance.
- Implement interactive visualizations, including dynamic charts and graphs, to facilitate data interpretation.
- Integrate real-time stock alerts and notifications to keep users informed of significant market developments.
- Optimize the platform for responsive design, ensuring accessibility across various devices and screen sizes.
- Implement user-friendly interfaces for seamless navigation and intuitive user experience.
- Provide comprehensive documentation and user guides to assist users in utilizing the platform effectively.
- Conduct thorough testing to ensure the reliability, performance, and security of the application.
- Continuously monitor and update the platform to incorporate new features and enhancements.
- Foster a supportive community around the project through forums, discussion groups, and user feedback channels.
- Collaborate with financial experts and industry professionals to ensure the accuracy and relevance of predictive models and analytics.

1.2 LIMITATIONS:

- Data accuracy and reliability from external sources.
- Predictive accuracy of machine learning models.
- Risk of overfitting predictive models to historical data.
- Inherent market risks associated with stock investments.
- Compliance with financial regulations and data privacy laws.
- Constraints on computational resources and server capacities.
- Technical issues such as server downtime or software bugs.
- User expertise required for interpreting predictive analytics.
- Scope limitations regarding covered stocks, markets, or strategies.
- Dependency on external APIs for live data feeds.

CHAPTER 2

PROPOSED SYSTEM

The proposed system, StockProphet, is a web-based platform designed to provide users with real-time stock market insights and predictions. Leveraging Django as the backend framework and integrating with Yahoo Finance's live data feeds, StockProphet offers a comprehensive suite of features to assist investors in making informed decisions.

SOFTWARE REQUIREMENTS:

- Django (backend framework)
- PostgreSQL, MySQL, or SQLite (database management system)
- Apache or Nginx (web server)
- Python (programming language)
- HTML5, CSS3, JavaScript, Bootstrap, jQuery, Plotly (frontend frameworks and libraries)
- Yahoo Finance API (live data integration)
- Scikit-learn, TensorFlow, or PyTorch (machine learning libraries)
- Django's built-in authentication system or Django REST framework (user authentication)
- Integrated Development Environment (IDE) like PyCharm, Visual Studio Code, or Sublime Text
- Git (version control)
- Platform as a Service (PaaS) providers like Heroku, AWS, or Azure (deployment platform)
- Testing frameworks like Django's built-in testing framework or pytest
- Documentation tools like Markdown, reStructuredText, and Sphinx
- Dependency management tools like pip or conda.

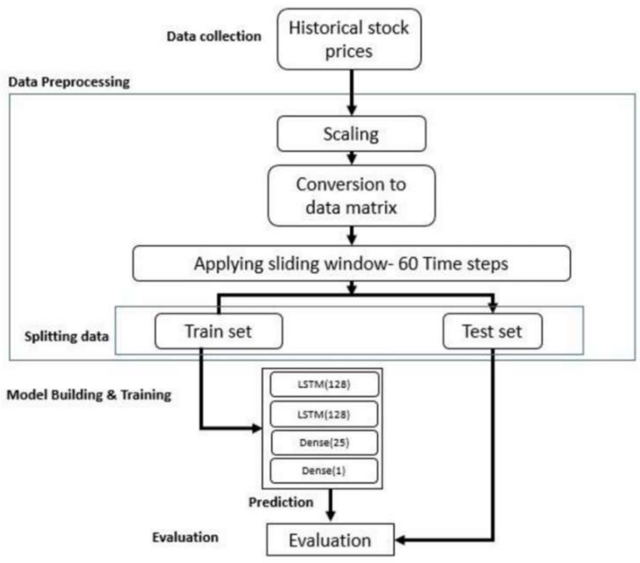
HARDWARE REQUIREMENTS:

- Server infrastructure with sufficient CPU and RAM.
- Adequate storage space for data storage.
- Reliable internet connectivity and bandwidth.
- Backup and redundancy systems for data protection.
- Security measures such as firewalls and SSL certificates.
- Scalable server architecture for handling increased traffic.
- Monitoring tools for performance tracking and management.

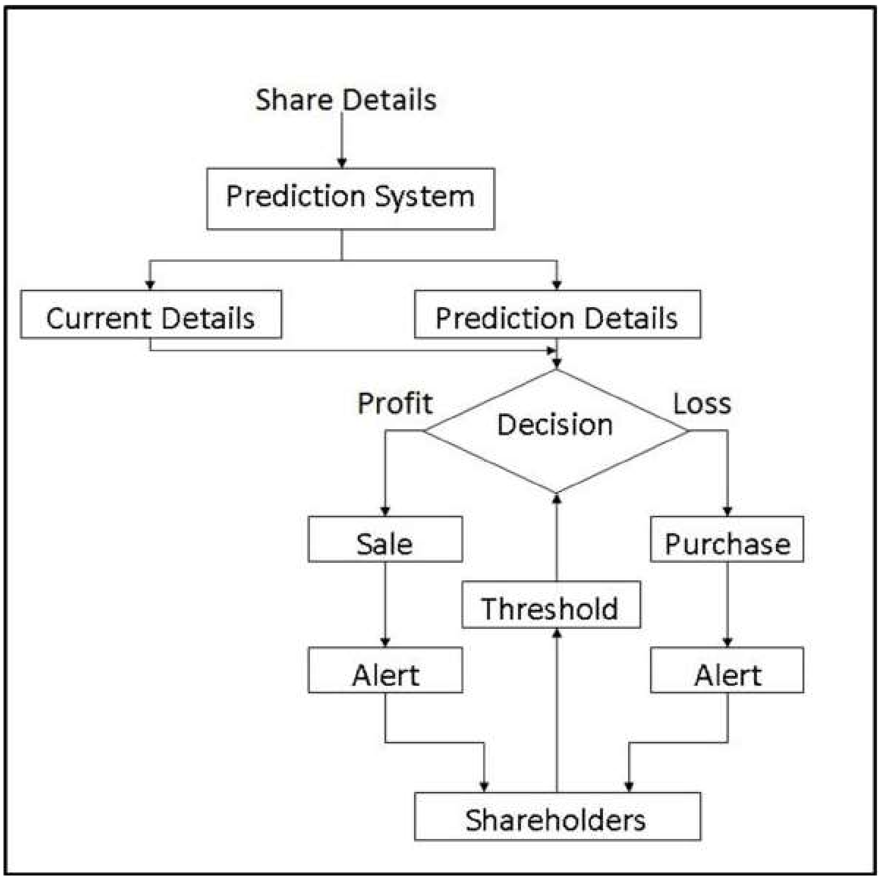
CHAPTER 3

Database Design

Flowchart:



Flowchart - 1



Flowchart-2

CHAPTER 4

Technical Details

Backend Database

A Database Management System (DBMS) is essential for managing databases, structured data, and performing operations on that data. Examples include Oracle, MySQL, PostgreSQL, SQLite, etc. DBMSs organize, store, manage, and retrieve data, offering modeling languages to define database schemas, data security features, and transaction mechanisms to ensure data integrity. SQL (Structured Query Language) is integral for data manipulation in relational databases, with SQL statements used for data definition, manipulation, and control.

In the relational model, data is stored in structures called relations or tables. SQL statements are issued for:

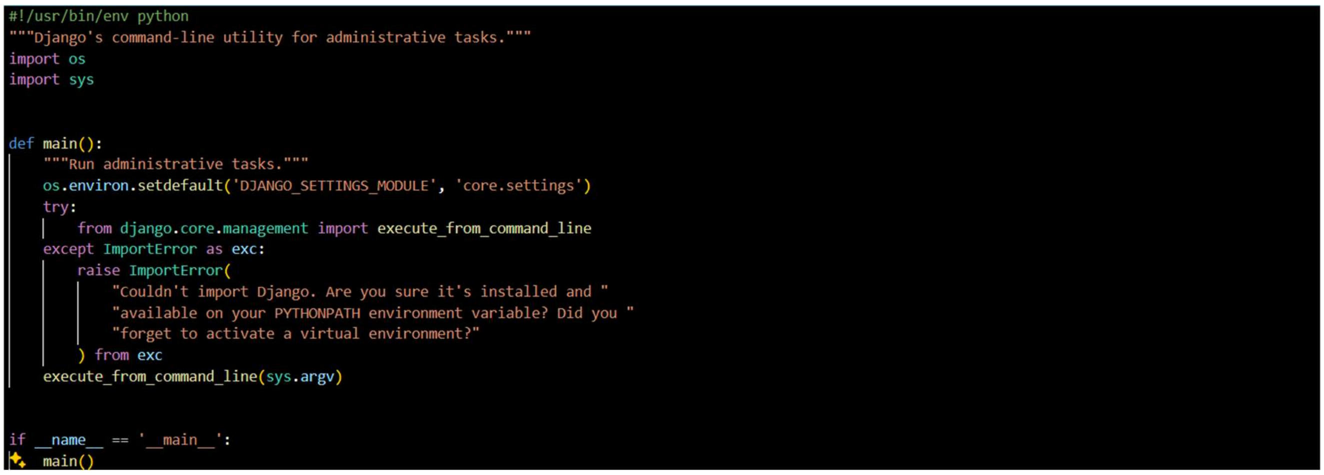
1. Data Definition: Defining tables and structures in the database (DDL used to create, alter, and drop schema objects such as tables and indexes).

The DBMS provides:
- A modeling language for defining database schema.
- Data structures optimized for dealing with large datasets.
- A database query language and report writer for interactive interrogation and analysis.
- Data security features to prevent unauthorized access.
- A transaction mechanism ensuring ACID properties for data integrity.
- Support for concurrent user accesses and fault tolerance.

DBMSs are vital for organizations, allowing for easy adaptation to changing information requirements and facilitating various types of data processing and analysis. SQL plays a central role in relational databases, providing a standardized language for data manipulation and control.

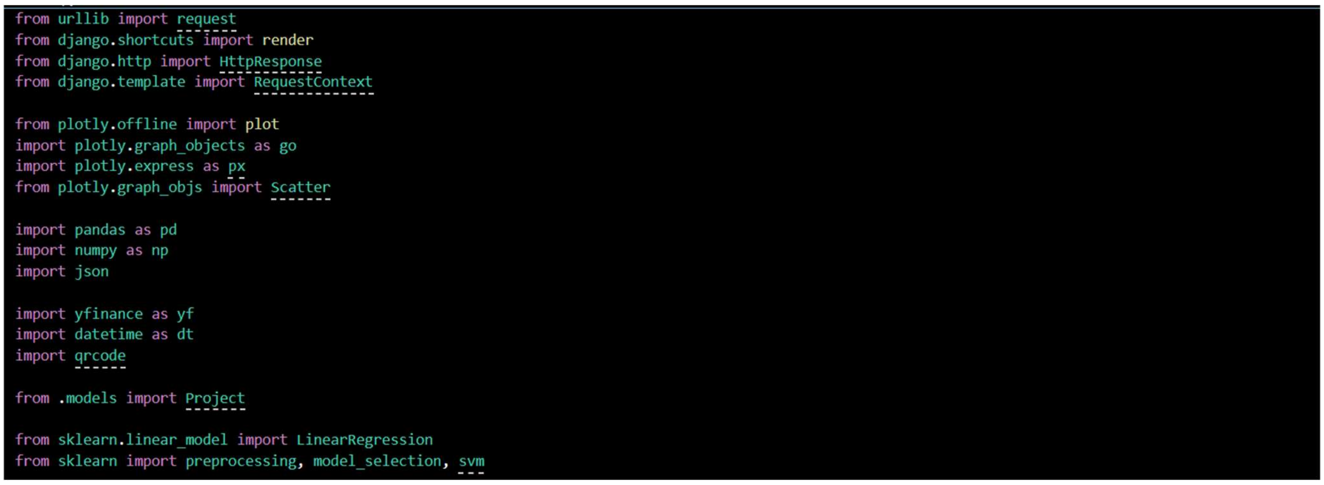
CHAPTER 5

CODE



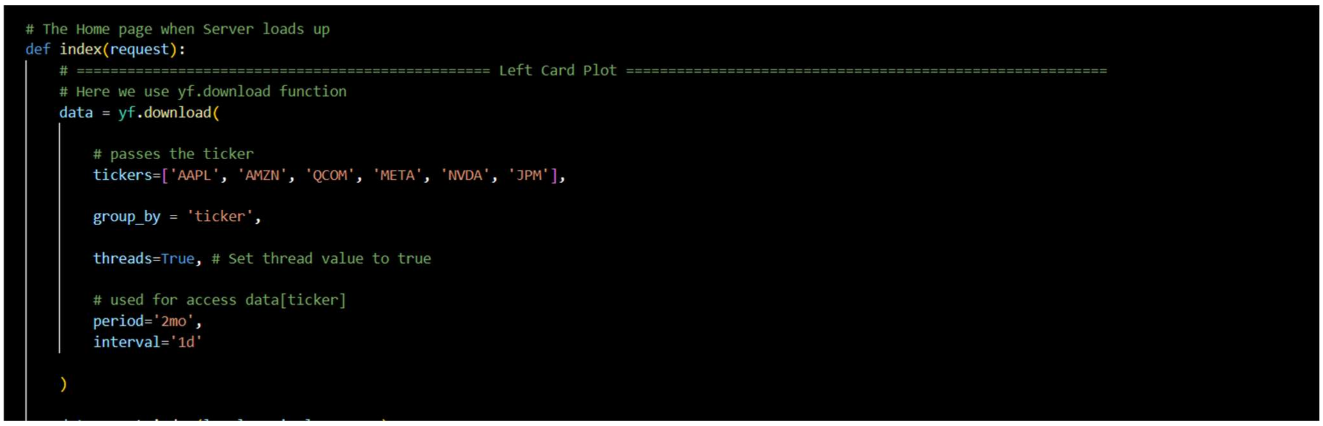
```python
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'core.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

Fig-1: This Python script is Django's command-line utility for administrative tasks. It sets up the Django environment and executes management commands specified via the command line.



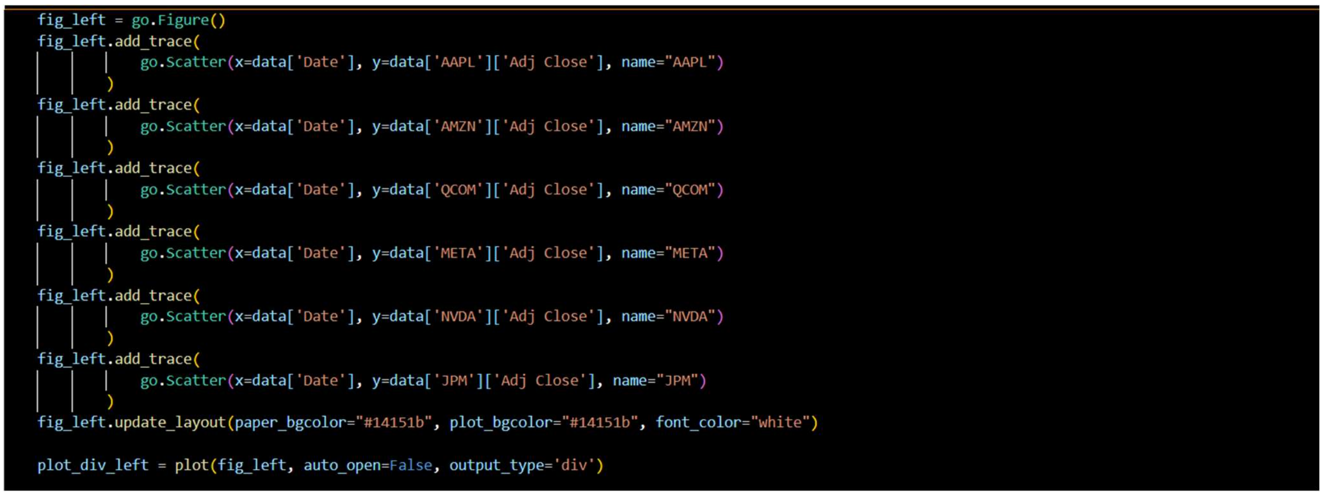
```python
from urllib import request
from django.shortcuts import render
from django.http import HttpResponse
from django.template import RequestContext

from plotly.offline import plot
import plotly.graph_objects as go
import plotly.express as px
from plotly.graph_objs import Scatter

import pandas as pd
import numpy as np
import json

import yfinance as yf
import datetime as dt
import qrcode

from .models import Project

from sklearn.linear_model import LinearRegression
from sklearn import preprocessing, model_selection, svm
```

Fig-2: This Python script imports Django components, Plotly, pandas, numpy, yfinance, datetime, qrcode, and scikit-learn, and also imports a model named `Project` from the local Django application.



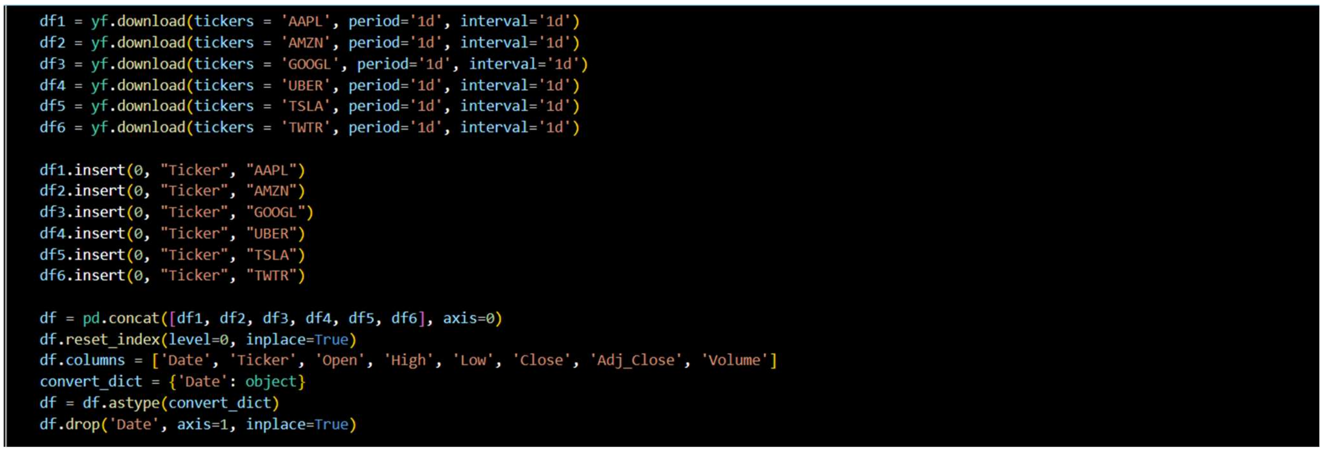
```python
# The Home page when Server loads up
def index(request):
    # ================================================ Left Card Plot =========================================================
    # Here we use yf.download function
    data = yf.download(

        # passes the ticker
        tickers=['AAPL', 'AMZN', 'QCOM', 'META', 'NVDA', 'JPM'],

        group_by = 'ticker',

        threads=True, # Set thread value to true

        # used for access data[ticker]
        period='2mo',
        interval='1d'

    )
```

Fig-3: This Django view function retrieves stock data for tickers AAPL, AMZN, QCOM, META, NVDA, and JPM using the yfinance library, then resets the index of the data DataFrame.



```python
fig_left = go.Figure()
fig_left.add_trace(
            go.Scatter(x=data['Date'], y=data['AAPL']['Adj Close'], name="AAPL")
        )
fig_left.add_trace(
            go.Scatter(x=data['Date'], y=data['AMZN']['Adj Close'], name="AMZN")
        )
fig_left.add_trace(
            go.Scatter(x=data['Date'], y=data['QCOM']['Adj Close'], name="QCOM")
        )
fig_left.add_trace(
            go.Scatter(x=data['Date'], y=data['META']['Adj Close'], name="META")
        )
fig_left.add_trace(
            go.Scatter(x=data['Date'], y=data['NVDA']['Adj Close'], name="NVDA")
        )
fig_left.add_trace(
            go.Scatter(x=data['Date'], y=data['JPM']['Adj Close'], name="JPM")
        )
fig_left.update_layout(paper_bgcolor="#14151b", plot_bgcolor="#14151b", font_color="white")

plot_div_left = plot(fig_left, auto_open=False, output_type='div')
```

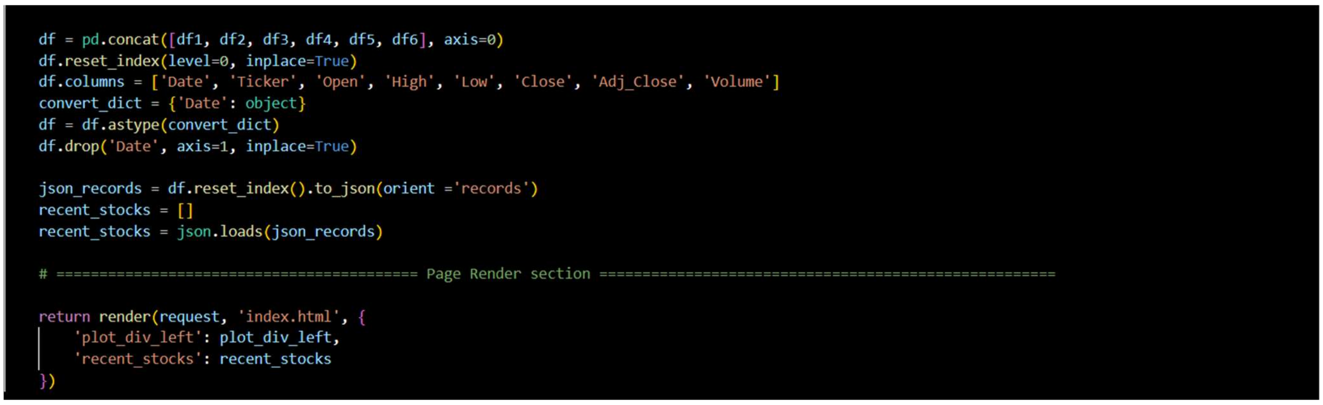
Fig-4: This code segment generates a Plotly line plot (`fig_left`) for the adjusted close prices of stocks AAPL, AMZN, QCOM, META, NVDA, and JPM, updates the layout, and generates a HTML div (`plot_div_left`) for embedding.



```python
df1 = yf.download(tickers = 'AAPL', period='1d', interval='1d')
df2 = yf.download(tickers = 'AMZN', period='1d', interval='1d')
df3 = yf.download(tickers = 'GOOGL', period='1d', interval='1d')
df4 = yf.download(tickers = 'UBER', period='1d', interval='1d')
df5 = yf.download(tickers = 'TSLA', period='1d', interval='1d')
df6 = yf.download(tickers = 'TWTR', period='1d', interval='1d')

df1.insert(0, "Ticker", "AAPL")
df2.insert(0, "Ticker", "AMZN")
df3.insert(0, "Ticker", "GOOGL")
df4.insert(0, "Ticker", "UBER")
df5.insert(0, "Ticker", "TSLA")
df6.insert(0, "Ticker", "TWTR")

df = pd.concat([df1, df2, df3, df4, df5, df6], axis=0)
df.reset_index(level=0, inplace=True)
df.columns = ['Date', 'Ticker', 'Open', 'High', 'Low', 'Close', 'Adj_Close', 'Volume']
convert_dict = {'Date': object}
df = df.astype(convert_dict)
df.drop('Date', axis=1, inplace=True)
```

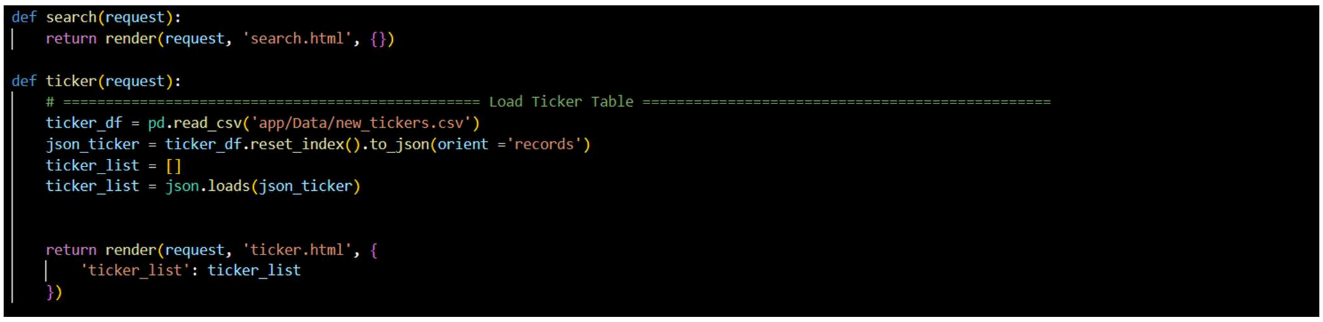
9.

Fig-5: This code retrieves daily stock data for tickers AAPL, AMZN, GOOGL, UBER, TSLA, and TWTR using the yfinance library, then concatenates the dataframes and formats the data into JSON records stored in the list `recent_stocks`.



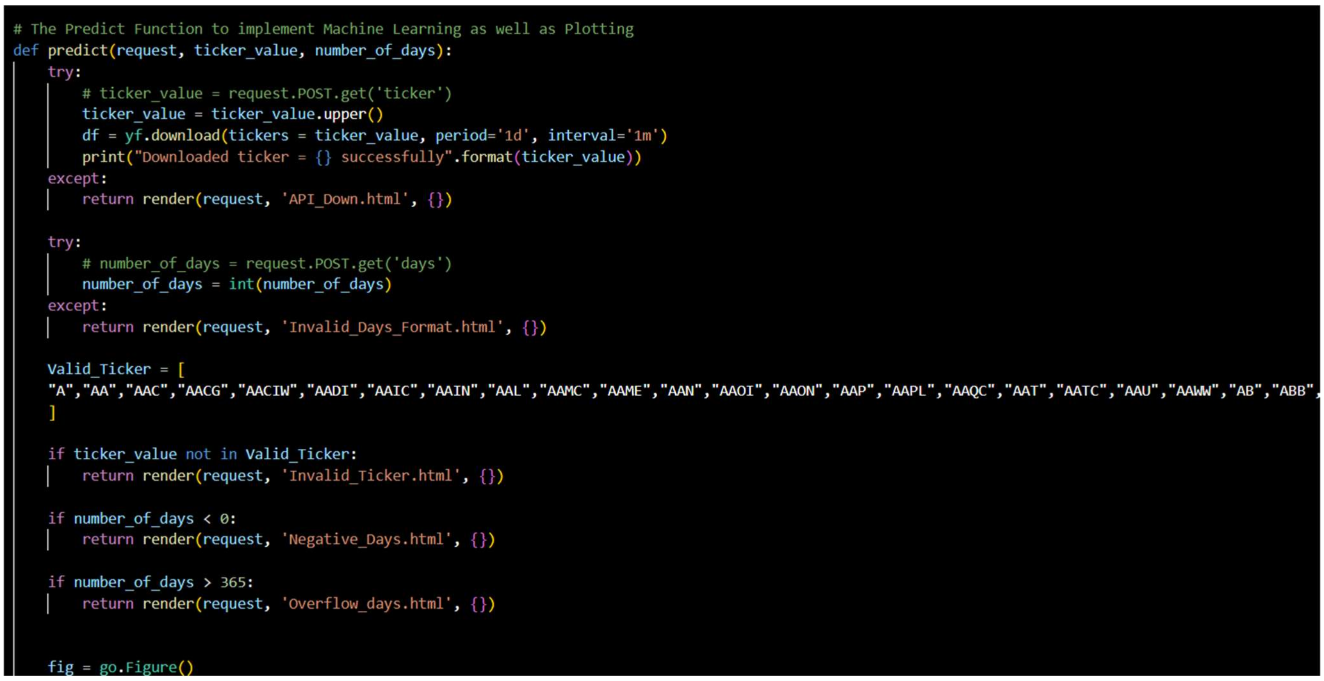
```python
df = pd.concat([df1, df2, df3, df4, df5, df6], axis=0)
df.reset_index(level=0, inplace=True)
df.columns = ['Date', 'Ticker', 'Open', 'High', 'Low', 'Close', 'Adj_Close', 'Volume']
convert_dict = {'Date': object}
df = df.astype(convert_dict)
df.drop('Date', axis=1, inplace=True)

json_records = df.reset_index().to_json(orient ='records')
recent_stocks = []
recent_stocks = json.loads(json_records)

# ========================================= Page Render section ==================================================

return render(request, 'index.html', {
    'plot_div_left': plot_div_left,
    'recent_stocks': recent_stocks
})
```

Fig-6: This code concatenates the dataframes `df1`, `df2`, `df3`, `df4`, `df5`, and `df6`, resets the index, renames the columns, converts the 'Date' column to object type, drops the 'Date' column, converts the dataframe to JSON records, and stores them in the list `recent_stocks`.



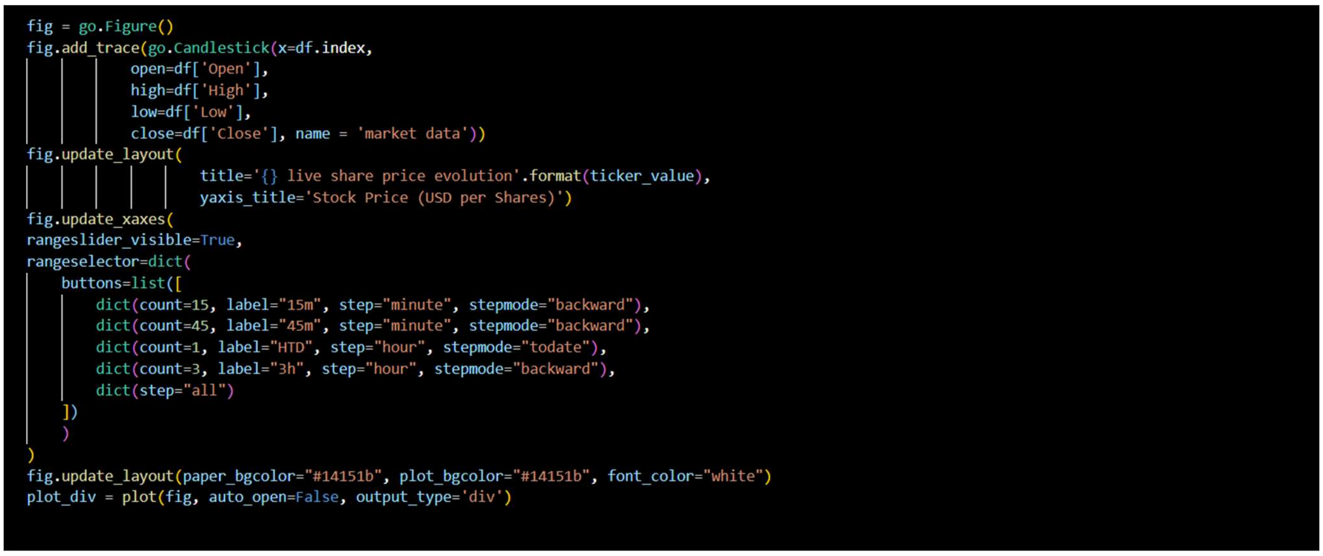
```python
def search(request):
    return render(request, 'search.html', {})

def ticker(request):
    # ================================================ Load Ticker Table ===============================================
    ticker_df = pd.read_csv('app/Data/new_tickers.csv')
    json_ticker = ticker_df.reset_index().to_json(orient ='records')
    ticker_list = []
    ticker_list = json.loads(json_ticker)


    return render(request, 'ticker.html', {
        'ticker_list': ticker_list
    })
```

Fig-7: This Django view function loads ticker data from a CSV file, converts it to JSON records, and passes it to the 'ticker.html' template for rendering.



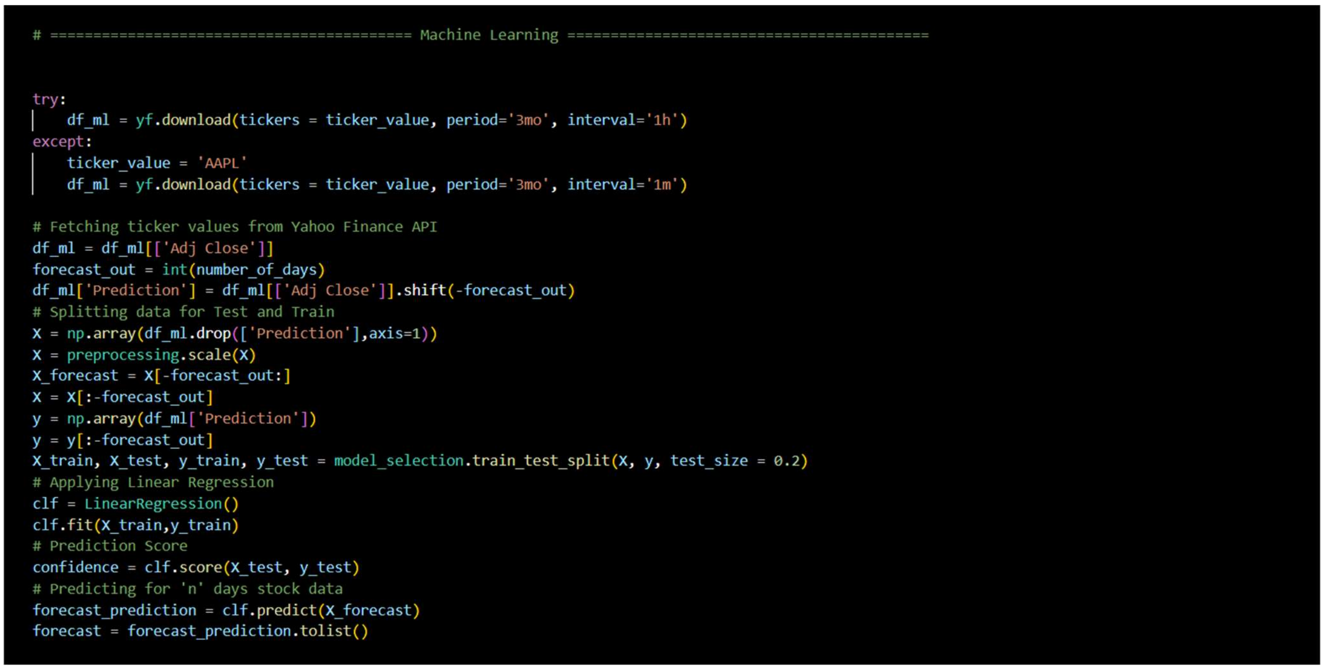
```python
# The Predict Function to implement Machine Learning as well as Plotting
def predict(request, ticker_value, number_of_days):
    try:
        # ticker_value = request.POST.get('ticker')
        ticker_value = ticker_value.upper()
        df = yf.download(tickers = ticker_value, period='1d', interval='1m')
        print("Downloaded ticker = {} successfully".format(ticker_value))
    except:
        return render(request, 'API_Down.html', {})

    try:
        # number_of_days = request.POST.get('days')
        number_of_days = int(number_of_days)
    except:
        return render(request, 'Invalid_Days_Format.html', {})

    Valid_Ticker = [
    "A","AA","AAC","AACG","AACIW","AADI","AAIC","AAIN","AAL","AAMC","AAME","AAN","AAOI","AAON","AAP","AAPL","AAQC","AAT","AATC","AAU","AAWW","AB","ABB",
    ]

    if ticker_value not in Valid_Ticker:
        return render(request, 'Invalid_Ticker.html', {})

    if number_of_days < 0:
        return render(request, 'Negative_Days.html', {})

    if number_of_days > 365:
        return render(request, 'Overflow_days.html', {})


    fig = go.Figure()
```

Fig-8: This Django view function takes a ticker value and the number of days as input parameters, retrieves minute-level stock data for the specified ticker using the yfinance library, and performs error handling for invalid ticker values, negative or overly large number of days.



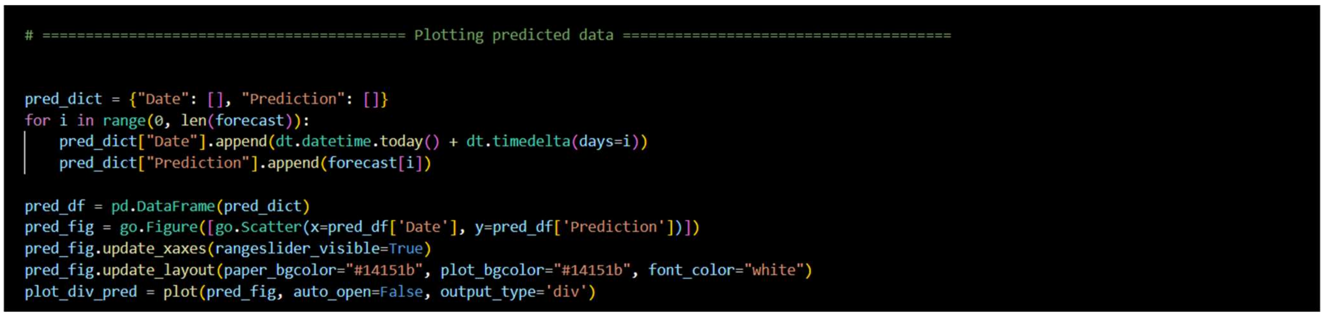
```python
fig = go.Figure()
fig.add_trace(go.Candlestick(x=df.index,
                open=df['Open'],
                high=df['High'],
                low=df['Low'],
                close=df['Close'], name = 'market data'))
fig.update_layout(
                        title='{} live share price evolution'.format(ticker_value),
                        yaxis_title='Stock Price (USD per Shares)')
fig.update_xaxes(
rangeslider_visible=True,
rangeselector=dict(
    buttons=list([
        dict(count=15, label="15m", step="minute", stepmode="backward"),
        dict(count=45, label="45m", step="minute", stepmode="backward"),
        dict(count=1, label="HTD", step="hour", stepmode="todate"),
        dict(count=3, label="3h", step="hour", stepmode="backward"),
        dict(step="all")
    ])
    )
)
fig.update_layout(paper_bgcolor="#14151b", plot_bgcolor="#14151b", font_color="white")
plot_div = plot(fig, auto_open=False, output_type='div')
```

Fig-9: This code segment creates a Plotly candlestick chart (`fig`) using minute-level stock data, updates the layout with the title and y-axis label based on the ticker value, adds range sliders and selectors for time intervals, and generates a HTML div (`plot_div`) for embedding the candlestick chart in a web page.



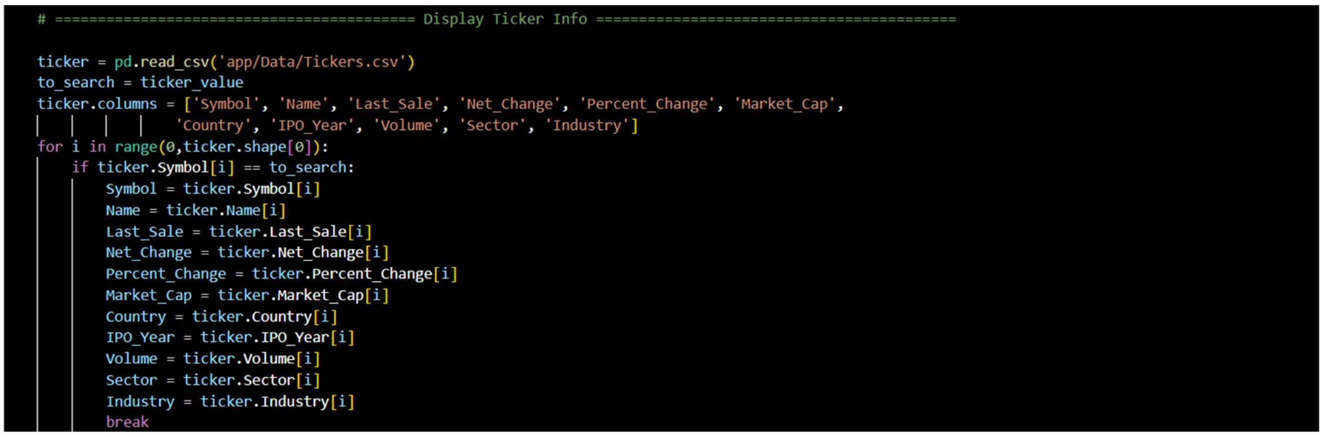
```python
# ======================================== Machine Learning ========================================


try:
    df_ml = yf.download(tickers = ticker_value, period='3mo', interval='1h')
except:
    ticker_value = 'AAPL'
    df_ml = yf.download(tickers = ticker_value, period='3mo', interval='1m')

# Fetching ticker values from Yahoo Finance API
df_ml = df_ml[['Adj Close']]
forecast_out = int(number_of_days)
df_ml['Prediction'] = df_ml[['Adj Close']].shift(-forecast_out)
# Splitting data for Test and Train
X = np.array(df_ml.drop(['Prediction'],axis=1))
X = preprocessing.scale(X)
X_forecast = X[-forecast_out:]
X = X[:-forecast_out]
y = np.array(df_ml['Prediction'])
y = y[:-forecast_out]
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size = 0.2)
# Applying Linear Regression
clf = LinearRegression()
clf.fit(X_train,y_train)
# Prediction Score
confidence = clf.score(X_test, y_test)
# Predicting for 'n' days stock data
forecast_prediction = clf.predict(X_forecast)
forecast = forecast_prediction.tolist()
```

Fig-10: This code segment retrieves historical stock data for a given ticker, calculates a prediction for a specified number of days using linear regression, and stores the prediction results in the list `forecast`.



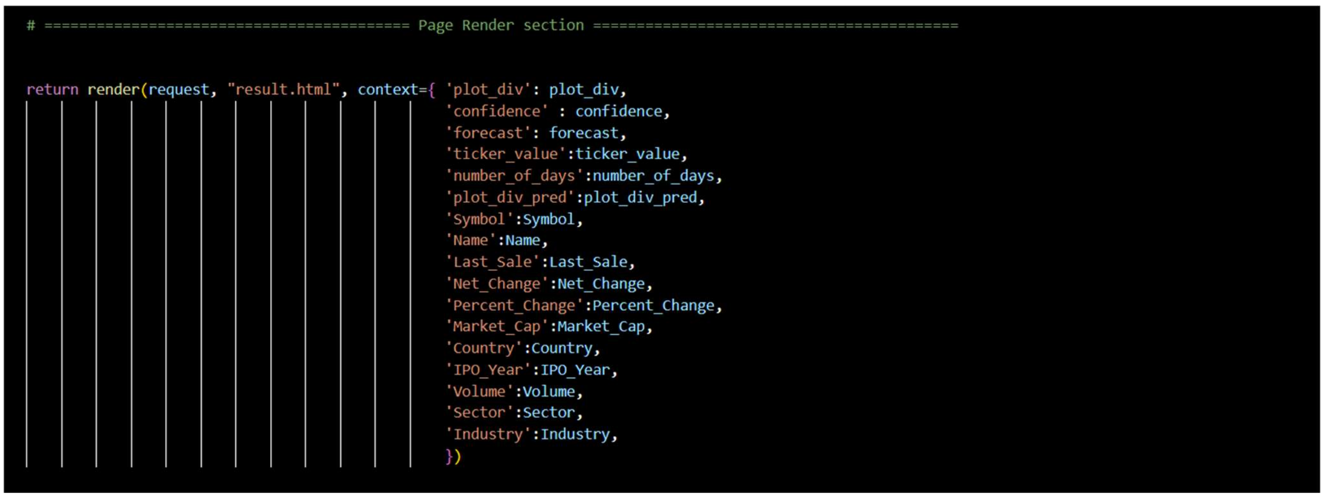
```python
# ========================================== Plotting predicted data ======================================


pred_dict = {"Date": [], "Prediction": []}
for i in range(0, len(forecast)):
    pred_dict["Date"].append(dt.datetime.today() + dt.timedelta(days=i))
    pred_dict["Prediction"].append(forecast[i])

pred_df = pd.DataFrame(pred_dict)
pred_fig = go.Figure([go.Scatter(x=pred_df['Date'], y=pred_df['Prediction'])])
pred_fig.update_xaxes(rangeslider_visible=True)
pred_fig.update_layout(paper_bgcolor="#14151b", plot_bgcolor="#14151b", font_color="white")
plot_div_pred = plot(pred_fig, auto_open=False, output_type='div')
```

Fig-11: This code segment creates a dictionary `pred_dict` containing dates and corresponding prediction values, converts it into a DataFrame `pred_df`, creates a Plotly figure `pred_fig` with a scatter plot of the predictions, updates the layout and range slider of the figure, and generates a HTML div `plot_div_pred` for embedding the scatter plot in a web page.



```python
# ========================================== Display Ticker Info ==========================================

ticker = pd.read_csv('app/Data/Tickers.csv')
to_search = ticker_value
ticker.columns = ['Symbol', 'Name', 'Last_Sale', 'Net_Change', 'Percent_Change', 'Market_Cap',
                'Country', 'IPO_Year', 'Volume', 'Sector', 'Industry']
for i in range(0,ticker.shape[0]):
    if ticker.Symbol[i] == to_search:
        Symbol = ticker.Symbol[i]
        Name = ticker.Name[i]
        Last_Sale = ticker.Last_Sale[i]
        Net_Change = ticker.Net_Change[i]
        Percent_Change = ticker.Percent_Change[i]
        Market_Cap = ticker.Market_Cap[i]
        Country = ticker.Country[i]
        IPO_Year = ticker.IPO_Year[i]
        Volume = ticker.Volume[i]
        Sector = ticker.Sector[i]
        Industry = ticker.Industry[i]
        break
```

Fig-12: This code segment reads ticker data from a CSV file, searches for a specific ticker value, and assigns its corresponding details such as symbol, name, last sale price, net change, percent change, market capitalization, country, IPO year, volume, sector, and industry to variables.

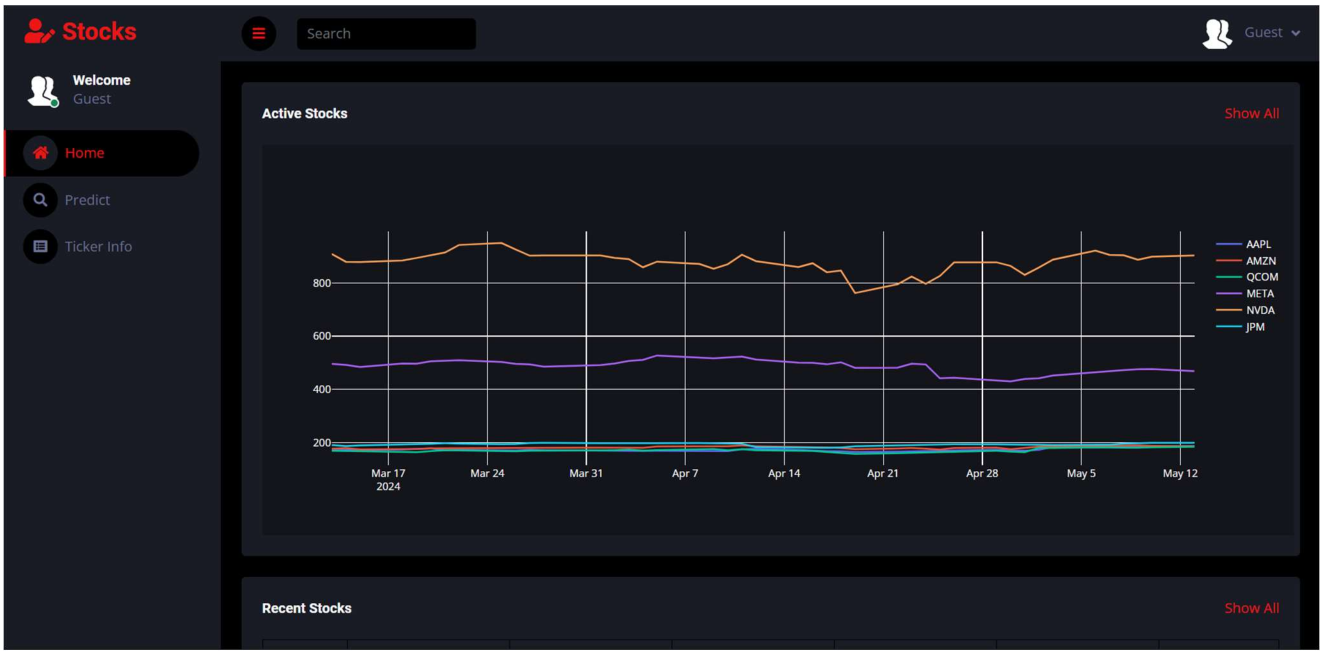


```
# ======================================== Page Render section ========================================


return render(request, "result.html", context={ 'plot_div': plot_div,
                                                'confidence' : confidence,
                                                'forecast': forecast,
                                                'ticker_value':ticker_value,
                                                'number_of_days':number_of_days,
                                                'plot_div_pred':plot_div_pred,
                                                'Symbol':Symbol,
                                                'Name':Name,
                                                'Last_Sale':Last_Sale,
                                                'Net_Change':Net_Change,
                                                'Percent_Change':Percent_Change,
                                                'Market_Cap':Market_Cap,
                                                'Country':Country,
                                                'IPO_Year':IPO_Year,
                                                'Volume':Volume,
                                                'Sector':Sector,
                                                'Industry':Industry,
                                                })
```

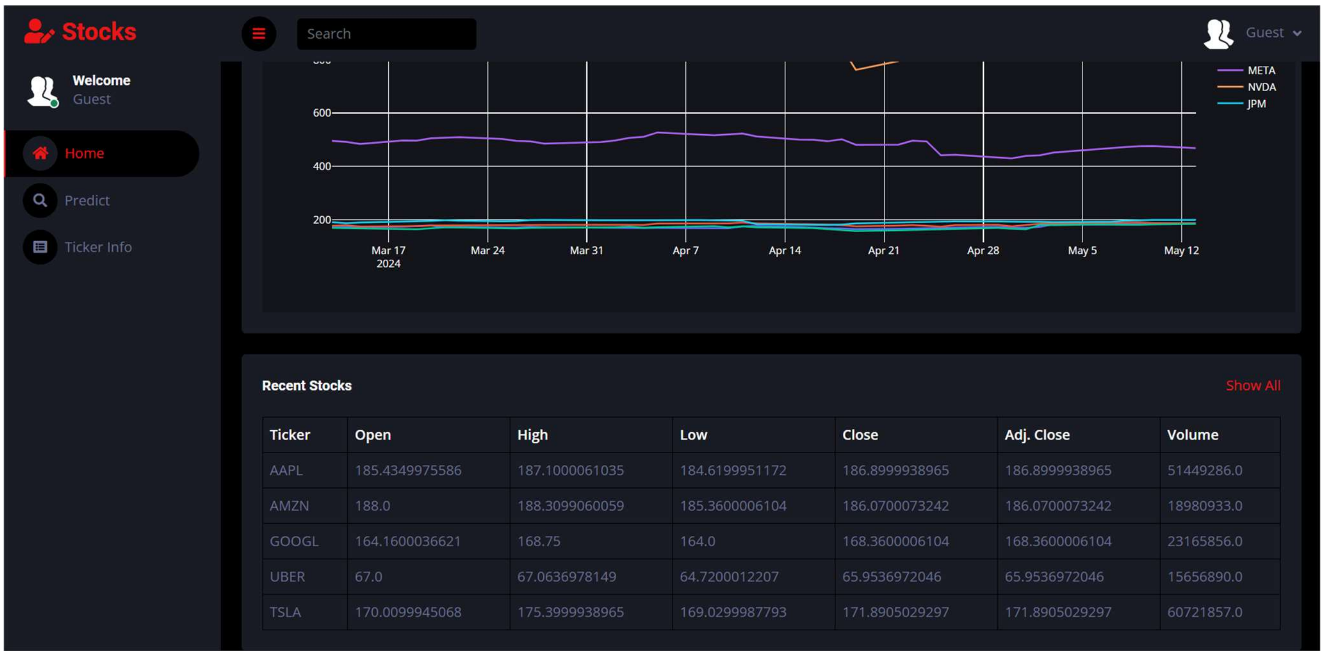
Fig-13: This code renders the 'result.html' template with various context variables including plot divs for the candlestick chart and prediction scatter plot, confidence score, forecast data, ticker details such as symbol, name, last sale price, net change, percent change, market capitalization, country, IPO year, volume, sector, and industry.

CHAPTER 6

Results (Screenshot)

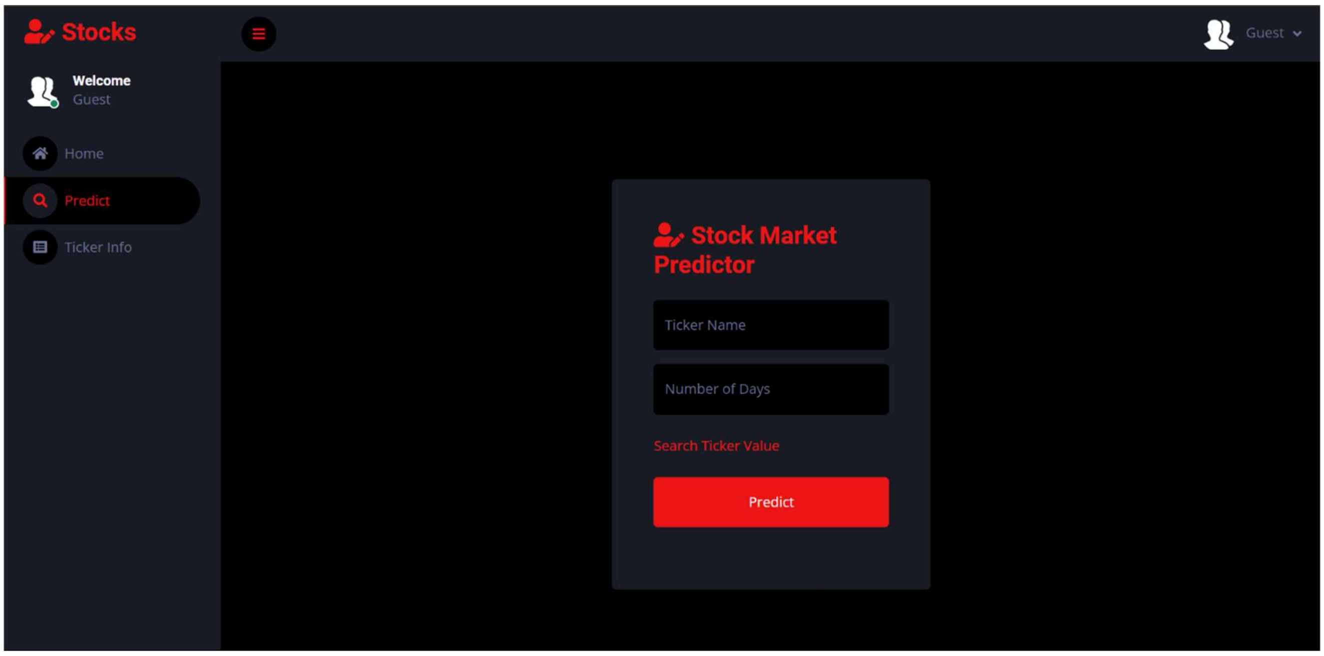


Results-1: The "predict" view function offers a comprehensive stock analysis, including data retrieval, trend prediction, visualization through candlestick and scatter plots, and stock details, all conveniently accessible on a single page.

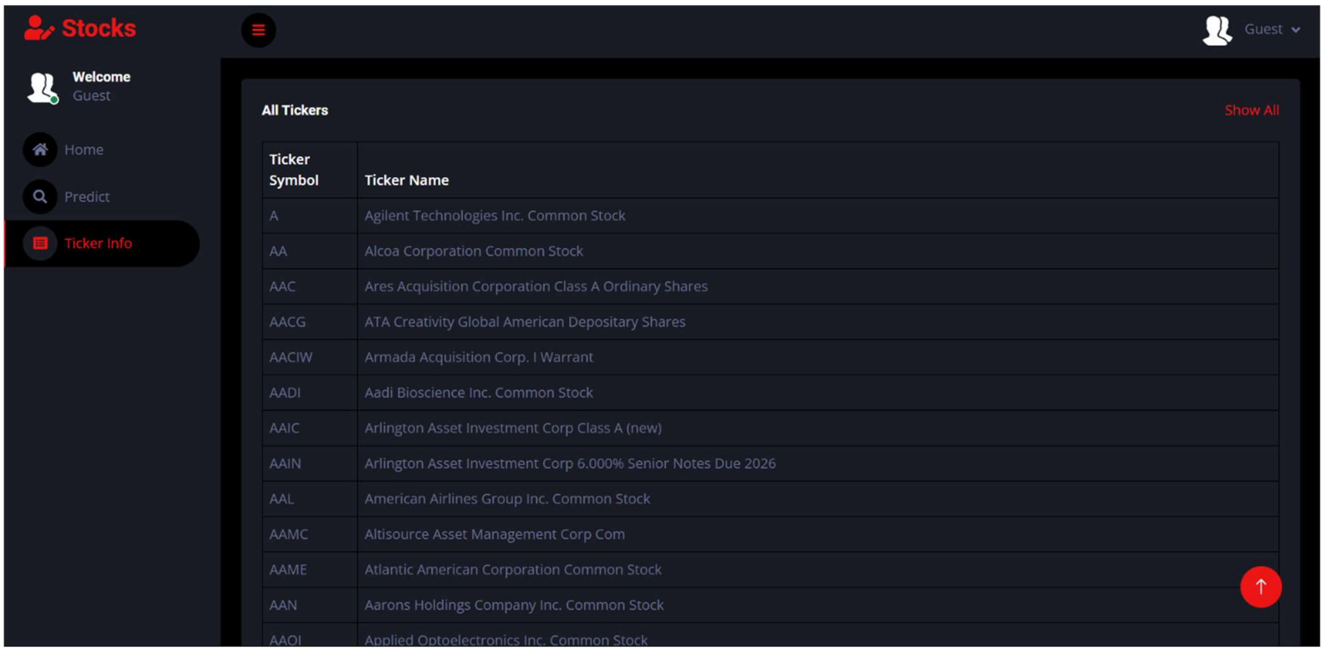


Ticker	Open	High	Low	Close	Adj. Close	Volume
AAPL	185.4349975586	187.1000061035	184.6199951172	186.8999938965	186.8999938965	51449286.0
AMZN	188.0	188.3099060059	185.3600006104	186.0700073242	186.0700073242	18980933.0
GOOGL	164.1600036621	168.75	164.0	168.3600006104	168.3600006104	23165856.0
UBER	67.0	67.0636978149	64.7200012207	65.9536972046	65.9536972046	15656890.0
TSLA	170.0099945068	175.3999938965	169.0299987793	171.8905029297	171.8905029297	60721857.0

Results-2: The "predict" view function provides a holistic stock analysis, presenting ticker data including open, high, low, close, adjusted close prices, and volume in both tabular format and numerical values, all conveniently accessible on a single page.



Results-3: The "predict" view function solicits user input for a stock's name and the desired number of days, then generates predictions for the stock's future trends, presenting the results on the same page for user convenience.



All Tickers

Ticker Symbol	Ticker Name
A	Agilent Technologies Inc. Common Stock
AA	Alcoa Corporation Common Stock
AAC	Ares Acquisition Corporation Class A Ordinary Shares
AACG	ATA Creativity Global American Depositary Shares
AACIW	Armada Acquisition Corp. I Warrant
AADI	Aadi Bioscience Inc. Common Stock
AAIC	Arlington Asset Investment Corp Class A (new)
AAIN	Arlington Asset Investment Corp 6.000% Senior Notes Due 2026
AAL	American Airlines Group Inc. Common Stock
AAMC	Altisource Asset Management Corp Com
AAME	Atlantic American Corporation Common Stock
AAN	Aarons Holdings Company Inc. Common Stock
AAOI	Applied Optoelectronics Inc. Common Stock

Results-4: The "predict" view function prompts users to input the stock's abbreviated name, along with the desired number of forecasted days, and subsequently displays predictions for the stock's future trends, all while conveniently showcasing the abbreviated forms of all available stocks.

CHAPTER 7

CONCLUSION

the StockProphet project presents an innovative solution for investors seeking to navigate the complexities of the stock market with confidence and precision. By leveraging Django as the backend framework and integrating with Yahoo Finance's live data feeds, StockProphet offers users a powerful platform for real-time stock prediction and analysis.

Throughout this project, we have explored the technical details, including the backend database requirements and the role of SQL in data manipulation. We have highlighted the importance of a robust Database Management System (DBMS) in managing structured data efficiently, providing essential features such as data modeling, security, and transaction management.

Furthermore, we have discussed the significance of SQL in relational databases, serving as a standardized language for defining, querying, and controlling database structures and data. SQL plays a pivotal role in enabling users to interact with the database, perform data analysis, and make informed investment decisions based on predictive analytics.

Overall, StockProphet aims to empower investors with the tools and insights needed to thrive in today's dynamic financial landscape. By embracing cutting-edge technologies and best practices in database management and data manipulation, StockProphet stands as a testament to innovation and excellence in the field of stock market analysis.

CHAPTER 8

Future enhancement

- Expanded Data Sources: Integrating additional data sources beyond Yahoo Finance.
- Advanced Predictive Models: Improving machine learning algorithms for better accuracy.
- Customizable Alerts: Allowing users to create personalized alert systems.
- Social Collaboration Features: Adding user forums or community-driven analysis.
- Mobile Application: Developing a dedicated app for convenient access.
- Algorithmic Trading Integration: Allowing users to automate trading strategies.
- Enhanced Visualization Tools: Providing more interactive dashboards.
- Natural Language Processing (NLP): Analyzing unstructured textual data for insights.
- Robust Backtesting Framework: Evaluating model performance against historical data.
- Global Market Coverage: Expanding market coverage to include various asset classes.

REFERENCES

- Django Documentation: https://docs.djangoproject.com/en/stable/ PostgreSQL/
- Documentation: https://www.postgresql.org/docs/
- MySQL Documentation: https://dev.mysql.com/doc/
- SQLite Documentation: https://www.sqlite.org/docs.html
- Apache Documentation: https://httpd.apache.org/docs/
- Nginx Documentation: https://nginx.org/en/docs/
- Python Documentation: https://docs.python.org/3/
- Bootstrap Documentation: https://getbootstrap.com/docs/5.1/getting-started/introduction/
- jQuery Documentation: https://api.jquery.com/
- Plotly Documentation: https://plotly.com/python/
- Yahoo Finance API Documentation: https://www.yahoofinanceapi.com/
- Scikit-learn Documentation: https://scikit-learn.org/stable/documentation.html
- TensorFlow Documentation: https://www.tensorflow.org/guide
- PyTorch Documentation: https://pytorch.org/docs/stable/index.html
- SQL Tutorial: https://www.w3schools.com/sql/
- Relational Database Management Systems (RDBMS) Tutorial: https://www.tutorialspoint.com/dbms/index.html