JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

B.TECH SEMESTER-V

OPEN SOURCE SOFTWARE LAB

PROJECT REPORT

STOCK MARKET PREDICTOR

Supervision of:

Prof. Deepti
Dr. Indu Chawla

ASSISTANT PROFESSOR
DEPARTMENT OF CSE & IT.
JIIT, SECTOR-62, NOIDA

Submitted by:

NAME	ENROLL NO.
SAI RAJ SINGH	22103216
VINEET KUMAR	22103222
HARSHAL RELAN	22103235

INTRODUCTION

The stock market is highly dynamic, with prices fluctuating every second based on market conditions, news, and various other factors. Investors and traders need real-time data and insights to make informed decisions. While APIs offer one way to access stock data, some websites do not provide APIs or have limited data access. Web scraping becomes a useful alternative for gathering stock information directly from web pages.

This project integrates web scraping using Beautiful Soup to extract stock market data from websites in real-time and stores the data in MongoDB, a NoSQL database that allows for efficient storage and retrieval of large amounts of data. Once the data is collected and stored, we will perform in-depth analysis and visualization using Pandas, NumPy, and Matplotlib. This integrated solution will enable users to: - Scrape stock data from websites in real time. – Store and manage large datasets efficiently.

- Analyze stock trends, price movements, and other financial metrics.
- Visualize data for better decision-making.

By combining web scraping with data storage and analysis, this project aims to create a comprehensive stock market analysis tool that is both scalable and flexible.

MODELLING and IMPLEMENTATION DETAILS

Website Scraped: https://finance.yahoo.com/quote/HAL.NS/history/

- **Data Extracted**: Stock historical data for the company HAL (Halliburton Company) from Yahoo Finance, specifically including:
  - Dates
  - Open Prices
  - High Prices
  - Low Prices
  - Close Prices
  - Adjusted Close Prices
  - Trading Volumes

1.Data Collection:

The project began with collecting stock data from Yahoo Finance using the requests library. A GET request was sent to the website with the necessary headers to bypass restrictions. The returned HTML content was then parsed using BeautifulSoup to extract the relevant stock data from specific table rows. The extracted data included dates, open prices, high prices, low prices, close prices, adjusted close prices, and trading volumes.

2. Data Storage and Manipulation

Libraries Involved: pandas, pymongo, csv

- **pandas:**
  - The pandas library played a crucial role in organizing the stock data into a structured format. Once the raw data was collected from Yahoo Finance, it was stored in a Pandas DataFrame, which is a tabular data structure similar to an Excel spreadsheet.
  - The use of pandas made it easy to clean, manipulate, and perform statistical calculations on the data. The describe()

function was called to get a summary of key statistics like mean, max, and standard deviation of the prices.
    - Additionally, moving averages (MA for 10, 20, and 50 days) and other stock indicators were calculated and stored as new columns in the DataFrame.
- **pymongo:**
    - After processing the data, pymongo was used to store the stock data in a MongoDB database so that it could be accessed and queried later. The data from the Pandas DataFrame was converted to dictionary format and inserted into a MongoDB collection.
    - This approach ensured that the stock data was not only analyzed locally but also saved persistently in a database for future retrieval and analysis.

csv:

- As part of the workflow, the stock data was also saved into a CSV file (stock_data.csv) using the csv.writer() method. This step allowed for easy data storage and ensured that the dataset could be accessed outside the Python environment.

```
HAL=pd.read_csv('stock_data.csv')
HAL.describe()
```

	Open	High	Low	Close	Adjusted Close	Volume
count	247.00000	247.000000	247.000000	247.000000	247.000000	2.470000e+02
mean	3910.15749	3967.414980	3839.856883	3904.360324	3890.966437	2.418579e+06
std	978.07588	988.528263	954.071222	967.053054	972.718769	1.977929e+06
min	2026.95000	2048.000000	1995.000000	2030.800000	2010.480000	1.701530e+05
25%	3016.50000	3045.400000	2980.000000	3003.800000	2981.270000	1.287220e+06
50%	4035.35000	4086.250000	3966.350000	4074.700000	4063.520000	1.801363e+06
75%	4740.00000	4781.050000	4666.400000	4712.450000	4707.190000	2.792774e+06
max	5638.90000	5674.750000	5560.500000	5621.950000	5606.520000	1.548668e+07

```
HAL=HAL.iloc[::-1]
```

```
HAL
```

	Date	Open	High	Low	Close	Adjusted Close	Volume
246	11/08/2023	2026.95	2048.00	2008.75	2044.55	2024.09	1748876
245	11/09/2023	2045.00	2049.75	2026.00	2030.80	2010.48	679939
244	11/10/2023	2041.65	2110.95	1995.00	2058.90	2038.30	4755794
243	11/13/2023	2072.00	2075.00	2041.80	2060.35	2039.74	1183911
242	11/15/2023	2066.05	2105.00	2040.55	2091.00	2070.08	1634396
...	...	...	...	...	...	...	...
4	11/04/2024	4287.95	4287.95	4196.00	4208.25	4208.25	1013884
3	11/05/2024	4200.00	4280.00	4130.00	4261.95	4261.95	1215686
2	11/06/2024	4276.00	4420.00	4270.05	4390.15	4390.15	1228872
1	11/07/2024	4404.00	4454.00	4381.10	4433.80	4433.80	1185134
0	11/08/2024	4435.55	4476.85	4380.00	4400.60	4400.60	1073330

3.Data Visualization:

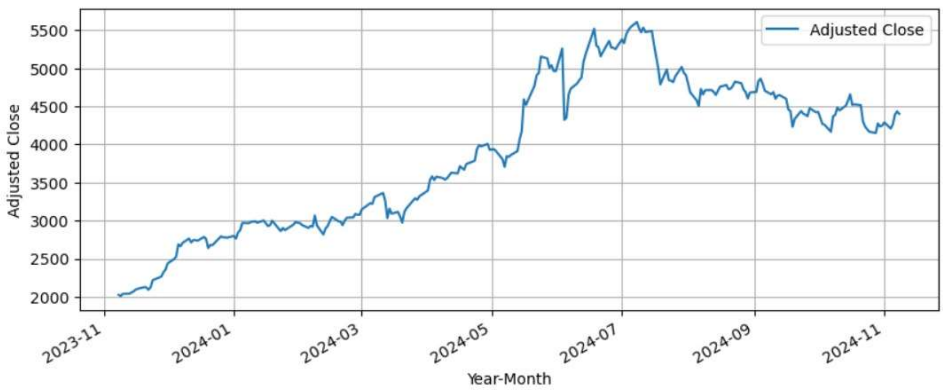
matplotlib:

The matplotlib library was essential for visualizing the stock data. Multiple plots were created to provide insights into the stock's behaviour over time. For instance:

- A time-series plot was created to show the adjusted close prices of the stock over time.
- Moving averages were also plotted to show how the stock's price smoothed over different windows (10 days, 20 days, 50 days).
- The library was further used to plot technical indicators such as MACD and RSI.

- The plots were customized with titles, legends, and gridlines to make them more interpretable.

```
HAL.plot(x='Date',y='Adjusted Close',figsize=(10,4))
plt.grid()
plt.xlabel("Year-Month")
plt.ylabel("Adjusted Close");
```



seaborn:

- seaborn was used for enhancing the aesthetic of the plots. For instance, the distplot function was used to create a distribution plot of the daily returns, which allowed us to visualize the distribution and volatility of the stock's returns.

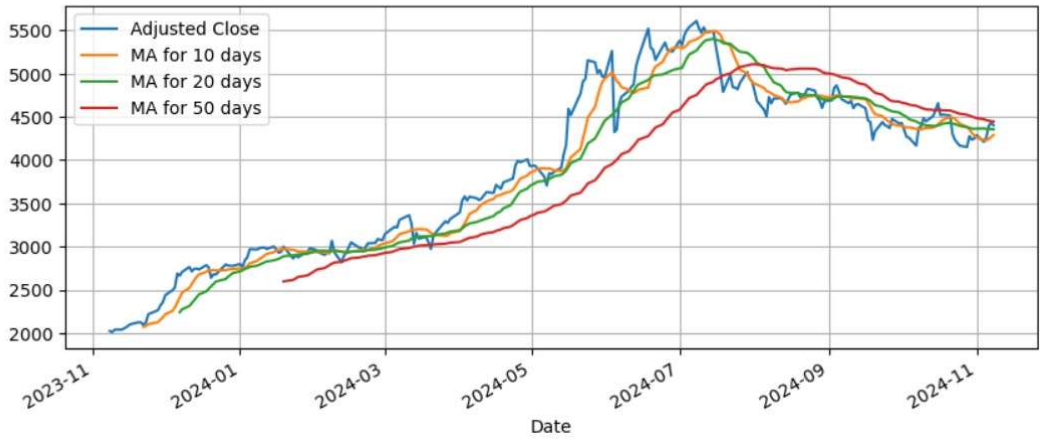
4.Technical Analysis and Indicator Calculation

Libraries Involved: pandas, numpy

Moving Averages (MA):

- Process: Moving averages for different windows (10, 20, 50 days) were calculated using the rolling() function available in pandas. This function was used to compute the average of the adjusted close prices over a sliding window.
- Purpose: Moving averages help smoothen stock price fluctuations, making it easier to identify trends

```
[59]: HAL.plot(x='Date',y=['Adjusted Close','MA for 10 days','MA for 20 days','MA for 50 days'],figsize=(10,4))
      plt.grid()
```

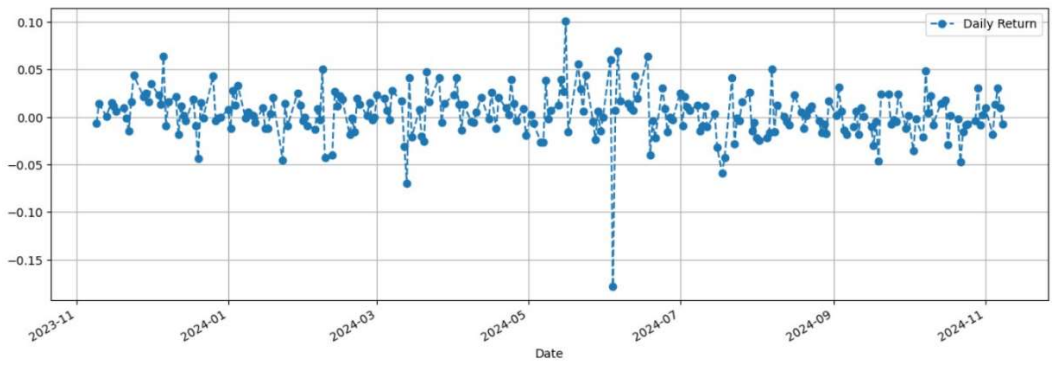


Daily Return:

- The daily return is calculated using the pct_change() function in pandas, which computes the percentage change in price from one day to the next. This allows us to understand how much the stock price has increased or decreased on a daily basis.

```
[64]: HAL.plot(x='Date',y='Daily Return',linestyle='--',marker='o',figsize=(15,5))
      plt.grid()
      plt.legend()

[64]: <matplotlib.legend.Legend at 0x2693617d010>
```

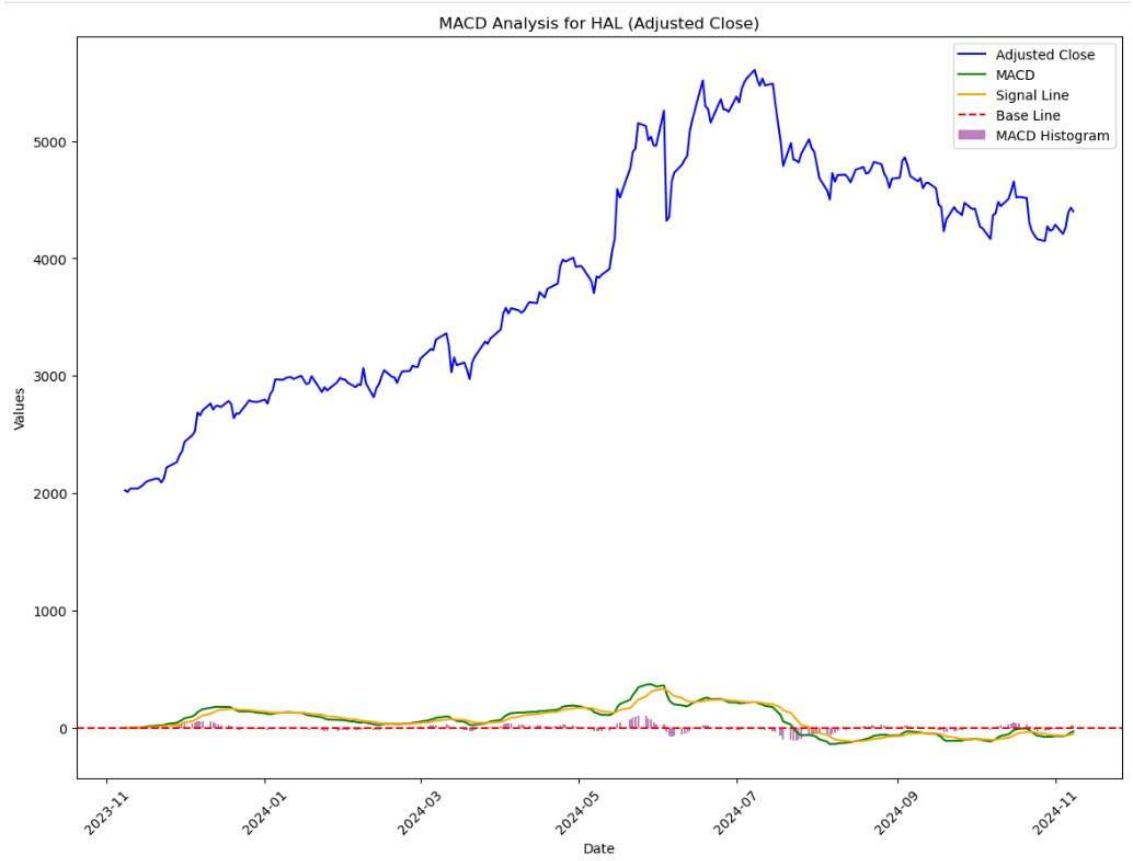


Relative Strength Index (RSI):

- Process: A custom function calculate_rsi() was defined to calculate the RSI, which measures the magnitude of recent price changes to evaluate overbought or oversold conditions.
- Purpose: RSI helps identify potential buy or sell signals based on whether the asset is overbought (RSI > 70) or oversold (RSI < 30).

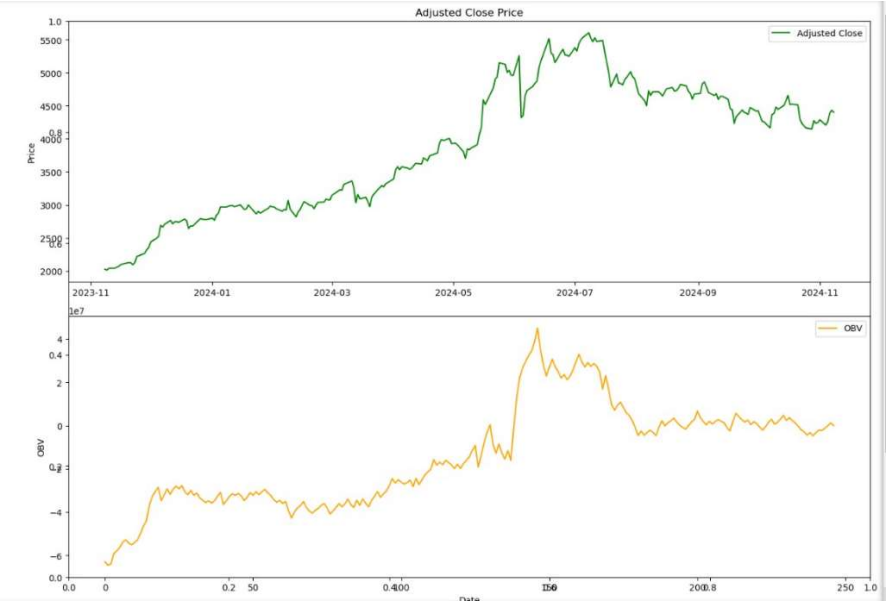
MACD (Moving Average Convergence Divergence):

- Process: The MACD was calculated as the difference between the 12-day and 26-day exponential moving averages (EMAs) of the adjusted close price. The signal line (9-day EMA of MACD) and MACD histogram were also plotted.
- Purpose: MACD is used as a momentum indicator to spot changes in the strength, direction, and duration of a trend.



On-Balance Volume (OBV):

- Process: OBV was calculated to measure buying and selling pressure. The cumulative volume was adjusted based on whether the stock price increased or decreased.
- Purpose: OBV helps confirm price trends by analyzing volume changes alongside price changes.

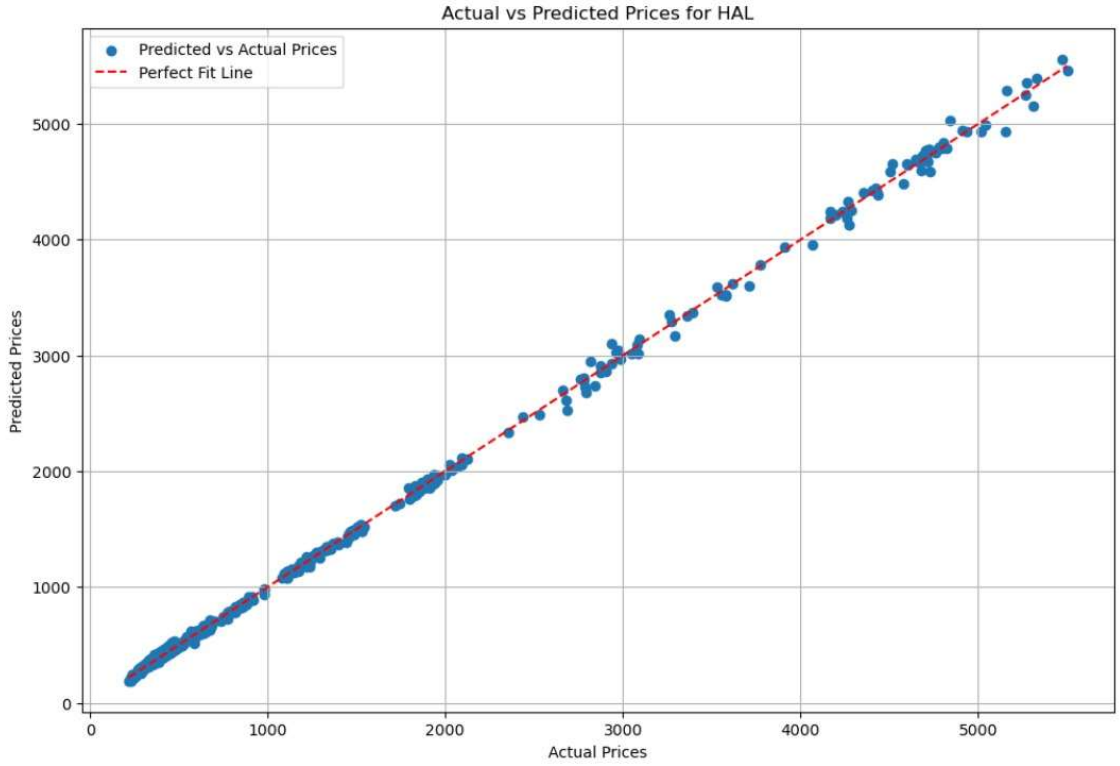


5.Predictive Modeling and Machine Learning

Libraries Involved: numpy, scikit-learn

- **Linear Regression:**
  - Process: We used the LinearRegression model from the sklearn library to predict future stock prices based on past data. The independent variable (X) was the opening price, and the dependent variable (Y) was the adjusted close price.
  - The dataset was split into training and testing sets using train_test_split(). The model was trained on 60% of the data and tested on 40%. The accuracy of the model was measured using the score() function, and the actual vs predicted stock prices were plotted.

```
plt.figure(figsize=(12, 8))
plt.scatter(y_test, lr.predict(x_test), label='Predicted vs Actual Prices')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--', label='Perfect Fit Line')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('Actual vs Predicted Prices for HAL')
plt.grid(True)
plt.legend()
plt.show()
```



TECHNOLOGY USED

- **Beautiful Soup(bs4):** A Python library for web scraping, used to extract stock data from financial websites.

- **MongoDB(pymongo library):** A NoSQL database for storing large amounts of real-time and historical stock data.

- **Pandas(pd):** For importing, cleaning, manipulating, and analyzing stock data from MongoDB.

- **NumPy(np):** For performing numerical calculations and computing financial indicators like moving averages and volatility.
- **Matplotlib(plt):** For visualizing stock trends, price movements, and technical indicators.
- **Scikit-Learn (sklearn):** scikit-learn is a machine learning library offering simple tools for data analysis and modelling.
- The combination of technical indicators (**MA, DR, RSI, MACD, OBV**) and machine learning (linear regression) provided a comprehensive toolkit for analyzing the stock's historical performance and predicting future trends.

This project combines real-time data collection, storage, and analysis, creating a dynamic system for tracking and analyzing stock market performance. The use of web scraping and MongoDB ensures that the tool can handle large datasets, while the dashboard provides a user-friendly interface for exploring and visualizing stock data.

CONCLUSION

In conclusion, this project demonstrated the use of various Python libraries for collecting, processing, and analyzing stock data. The libraries worked together seamlessly to achieve the project's objectives. The requests and BeautifulSoup libraries collected the data, the pandas and NumPy libraries processed and analyzed the data, the matplotlib and seaborn libraries visualized the data, and the scikit-learn library implemented machine learning techniques. The project provided a comprehensive analysis of the stock data and demonstrated the potential of using Python libraries for stock market analysis.

FUTURE WORK

Future work could include:

- Collecting data from multiple sources to improve the accuracy of the predictions.
- Considering other factors that may affect the stock prices.
- Using other machine learning techniques to improve the accuracy of the predictions.