Stock Recommendation System using Machine Learning

MACHINE LEARNING PROJECT

TEAM MEMBERS

01
2262
SHUBHAM GAONKAR

02
2257
SHREEVESH NAIK

03
2259
GRISHMA CHODANKER

04
2246
AMBERLY SILVA

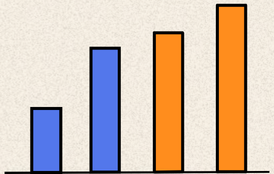
OVERVIEW

OBJECTIVE

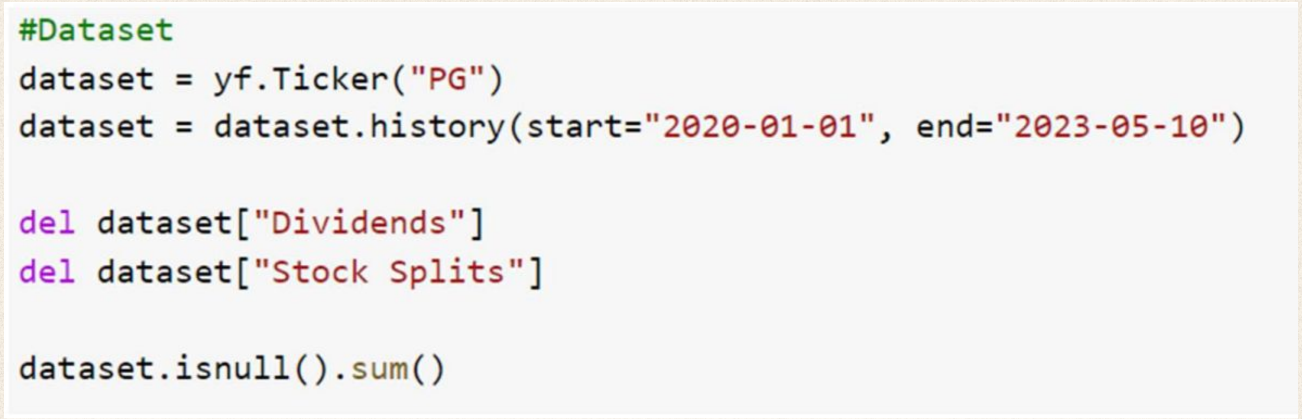
To develop a recommendation system for investors to suggest the best 4 stocks out of a set of 10 based on historical prices, technical indicators, and market trends and also to analyze stock price predictions using machine learning based on past returns. Overall, it helps to create a user-friendly tool to optimize investment portfolios and maximize returns.

LIBRARIES USED

1. **yfinance:** library for downloading historical market data from Yahoo Finance
2. **pandas:** library for data manipulation and analysis
3. **mplfinance:** library for plotting financial charts and technical indicators
4. **plotly:** library for creating interactive visualizations
5. **sklearn:** machine learning library for data mining and analysis
6. **Math:** used for simple math and basic statistical analysis.



DATA PROCESSING



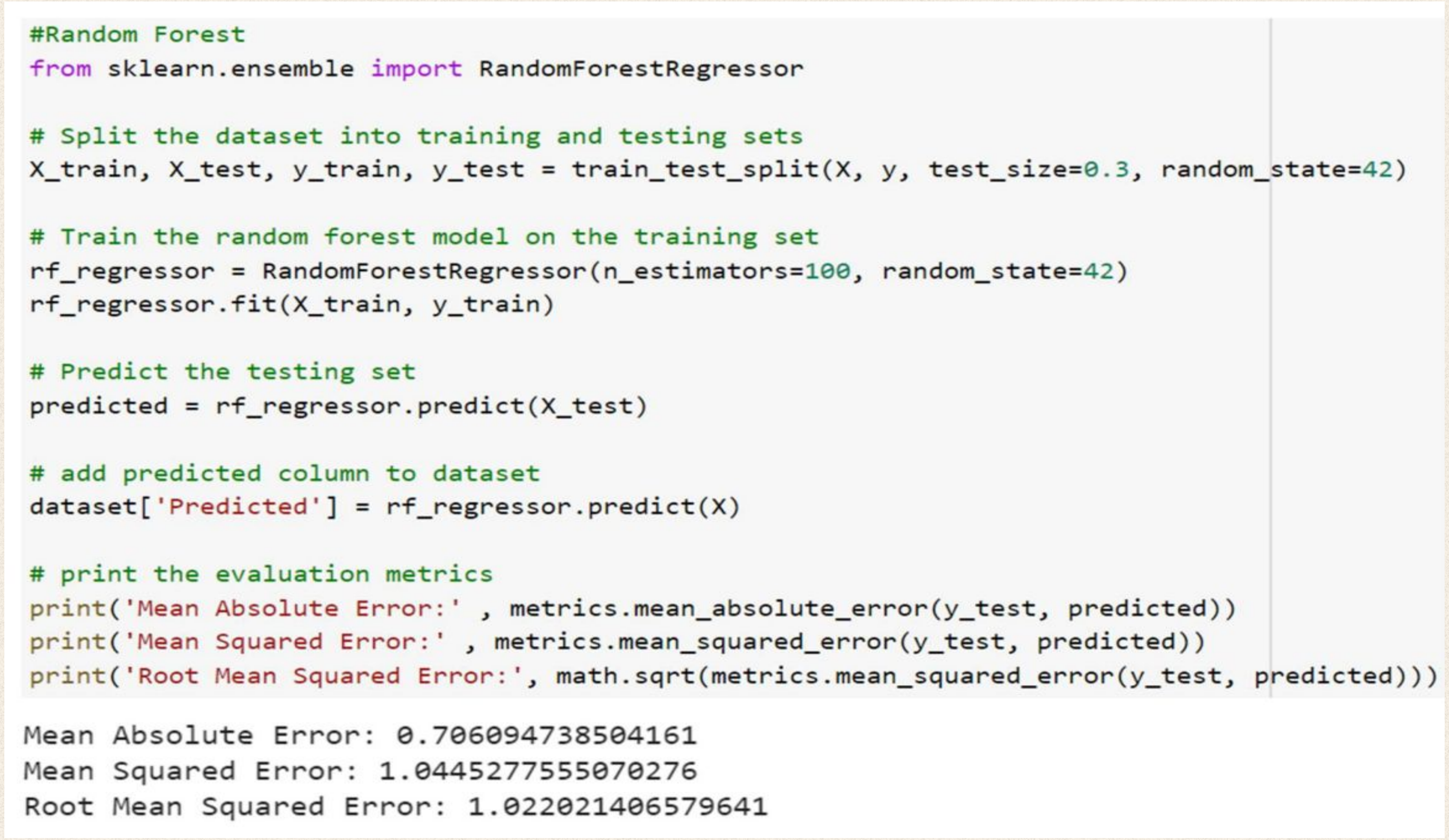
```
#Dataset
dataset = yf.Ticker("PG")
dataset = dataset.history(start="2020-01-01", end="2023-05-10")

del dataset["Dividends"]
del dataset["Stock Splits"]

dataset.isnull().sum()
```

1. The above code retrieves historical stock data of Procter & Gamble from Yahoo Finance.
2. Data stored in pandas DataFrame named "dataset".
3. "Dividends" and "Stock Splits" columns removed because not relevant.
4. "isnull().sum()" method used to check for missing values in dataset.
5. Method calculates total missing values for each column and adds them up.
6. Code prepares stock data by removing unnecessary columns.

MODELS USED

1. RANDOM FOREST FOREST
2. LINEAR REGRESSION

1. RANDOM TREE FOREST

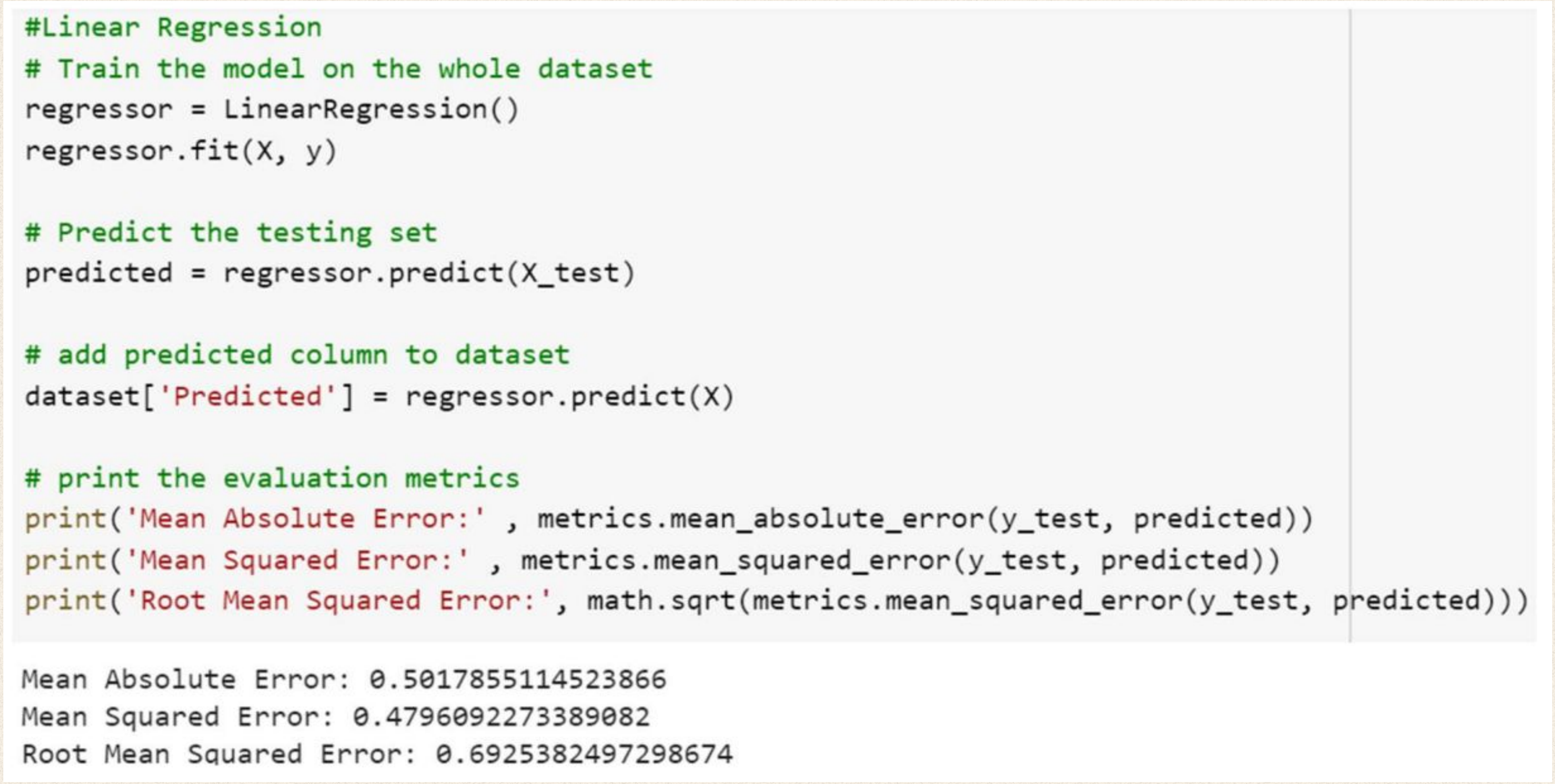


```
#Random Forest
from sklearn.ensemble import RandomForestRegressor

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the random forest model on the training set
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train, y_train)

# Predict the testing set
predicted = rf_regressor.predict(X_test)

# add predicted column to dataset
dataset['Predicted'] = rf_regressor.predict(X)

# print the evaluation metrics
print('Mean Absolute Error:' , metrics.mean_absolute_error(y_test, predicted))
print('Mean Squared Error:' , metrics.mean_squared_error(y_test, predicted))
print('Root Mean Squared Error:', math.sqrt(metrics.mean_squared_error(y_test, predicted)))
```

```
Mean Absolute Error: 0.706094738504161
Mean Squared Error: 1.0445277555070276
Root Mean Squared Error: 1.022021406579641
```

The above code implements a Random Forest Regression model for stock price prediction.

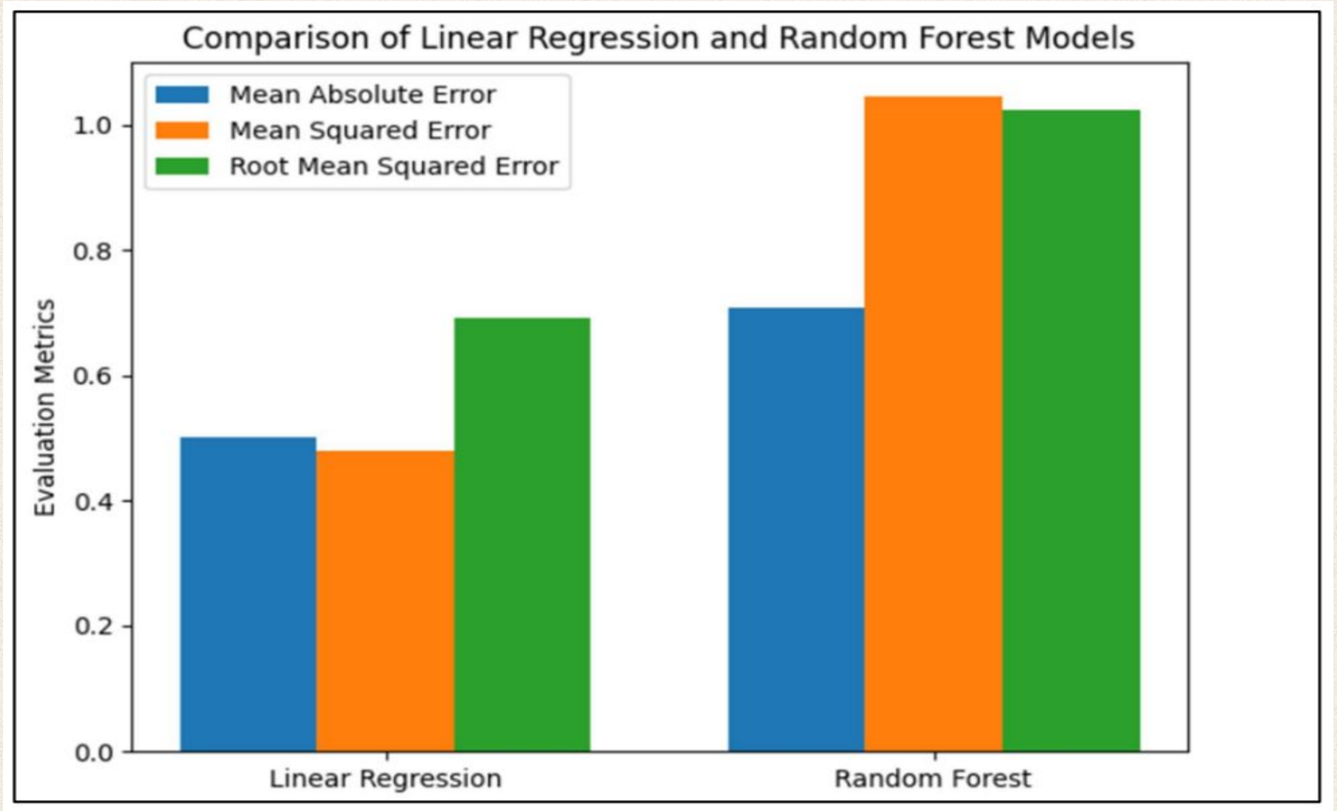
2. LINEAR REGRESSION



```
#Linear Regression
# Train the model on the whole dataset
regressor = LinearRegression()
regressor.fit(X, y)

# Predict the testing set
predicted = regressor.predict(X_test)

# add predicted column to dataset
dataset['Predicted'] = regressor.predict(X)

# print the evaluation metrics
print('Mean Absolute Error:' , metrics.mean_absolute_error(y_test, predicted))
print('Mean Squared Error:' , metrics.mean_squared_error(y_test, predicted))
print('Root Mean Squared Error:', math.sqrt(metrics.mean_squared_error(y_test, predicted)))
```

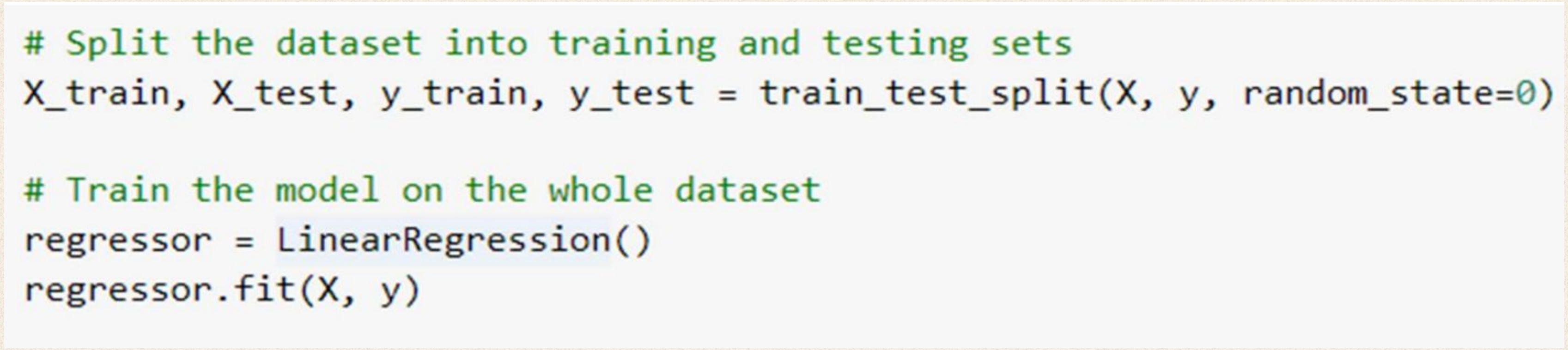
```
Mean Absolute Error: 0.5017855114523866
Mean Squared Error: 0.4796092273389082
Root Mean Squared Error: 0.692538249729 8674
```

This code performs linear regression analysis on the dataset.



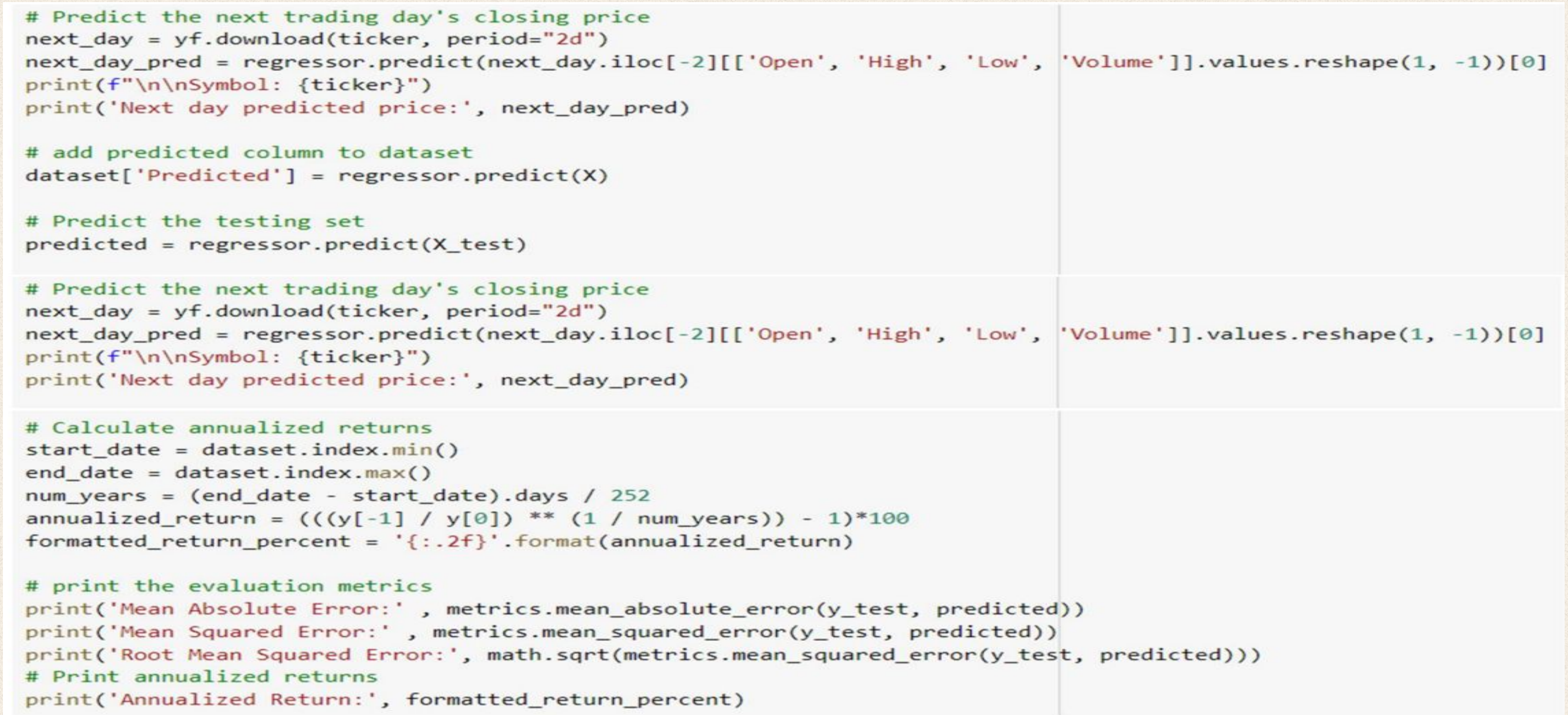
Based on the parameters, the Linear Regression model has a lower MAE, MSE, and RMSE than the Random Forest model. This suggests that the Linear Regression model is slightly more accurate in predicting the stock prices than the Random Forest model.

TRAINING


```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# Train the model on the whole dataset
regressor = LinearRegression()
regressor.fit(X, y)
```

1. The dataset is split into training and testing sets using train_test_split function.
2. The Linear Regression model is initialized and fit method is called on training set to train the model.
3. The entire dataset is also used for training the model by calling fit method on input features and target variable of the entire dataset.
4. The model is trained to predict the next trading day's closing price using predict method on the Open, High, Low, and Volume of the latest trading day.

TESTING AND EVALUATION

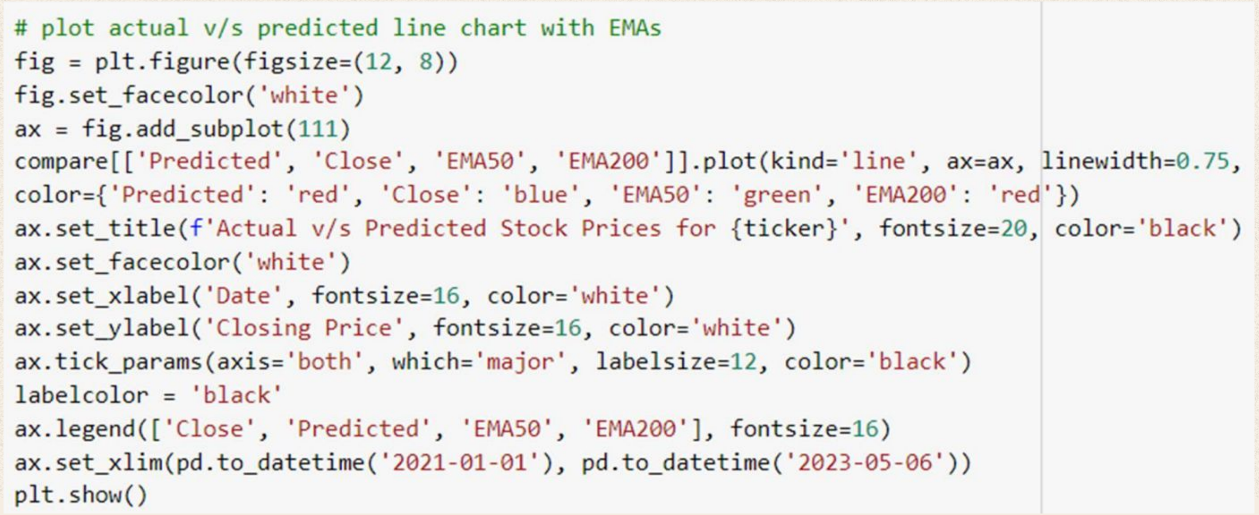


```
# Predict the next trading day's closing price
next_day = yf.download(ticker, period="2d")
next_day_pred = regressor.predict(next_day.iloc[-2][['Open', 'High', 'Low', 'Volume']].values.reshape(1, -1))[0]
print(f"\n\nSymbol: {ticker}")
print('Next day predicted price:', next_day_pred)

# add predicted column to dataset
dataset['Predicted'] = regressor.predict(X)

# Predict the testing set
predicted = regressor.predict(X_test)
```

```
# Predict the next trading day's closing price
next_day = yf.download(ticker, period="2d")
next_day_pred = regressor.predict(next_day.iloc[-2][['Open', 'High', 'Low', 'Volume']].values.reshape(1, -1))[0]
print(f"\n\nSymbol: {ticker}")
print('Next day predicted price:', next_day_pred)
```

```
# Calculate annualized returns
start_date = dataset.index.min()
end_date = dataset.index.max()
num_years = (end_date - start_date).days / 252
annualized_return = (((y[-1] / y[0]) ** (1 / num_years)) - 1)*100
formatted_return_percent = '{:.2f}'.format(annualized_return)

# print the evaluation metrics
print('Mean Absolute Error:' , metrics.mean_absolute_error(y_test, predicted))
print('Mean Squared Error:' , metrics.mean_squared_error(y_test, predicted))
print('Root Mean Squared Error:', math.sqrt(metrics.mean_squared_error(y_test, predicted)))
# Print annualized returns
print('Annualized Return:', formatted_return_percent)
```

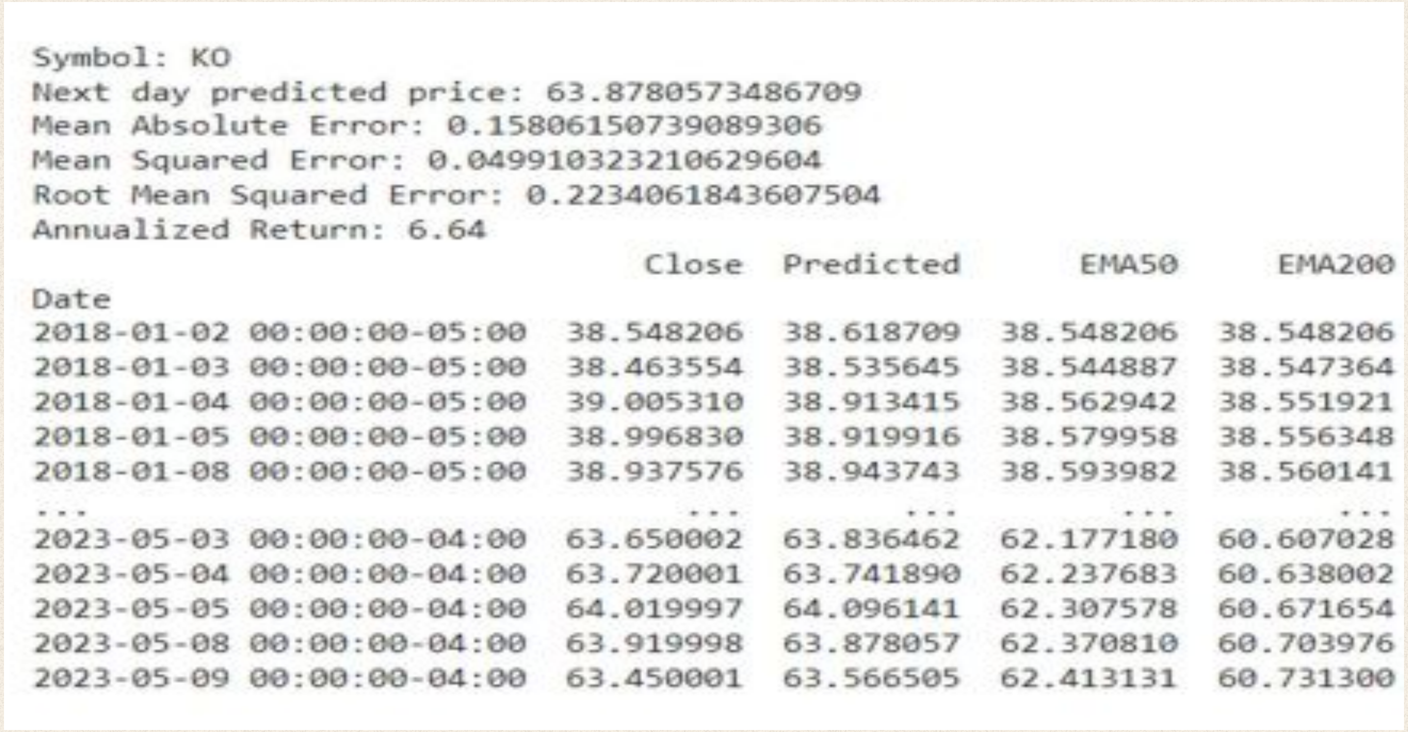
EVALUATION OF MODELS USING PLOTS



```python
# plot actual v/s predicted line chart with EMAs
fig = plt.figure(figsize=(12, 8))
fig.set_facecolor('white')
ax = fig.add_subplot(111)
compare[['Predicted', 'Close', 'EMA50', 'EMA200']].plot(kind='line', ax=ax, linewidth=0.75,
color={'Predicted': 'red', 'Close': 'blue', 'EMA50': 'green', 'EMA200': 'red'})
ax.set_title(f'Actual v/s Predicted Stock Prices for {ticker}', fontsize=20, color='black')
ax.set_facecolor('white')
ax.set_xlabel('Date', fontsize=16, color='white')
ax.set_ylabel('Closing Price', fontsize=16, color='white')
ax.tick_params(axis='both', which='major', labelsize=12, color='black')
labelcolor = 'black'
ax.legend(['Close', 'Predicted', 'EMA50', 'EMA200'], fontsize=16)
ax.set_xlim(pd.to_datetime('2021-01-01'), pd.to_datetime('2023-05-06'))
plt.show()
```

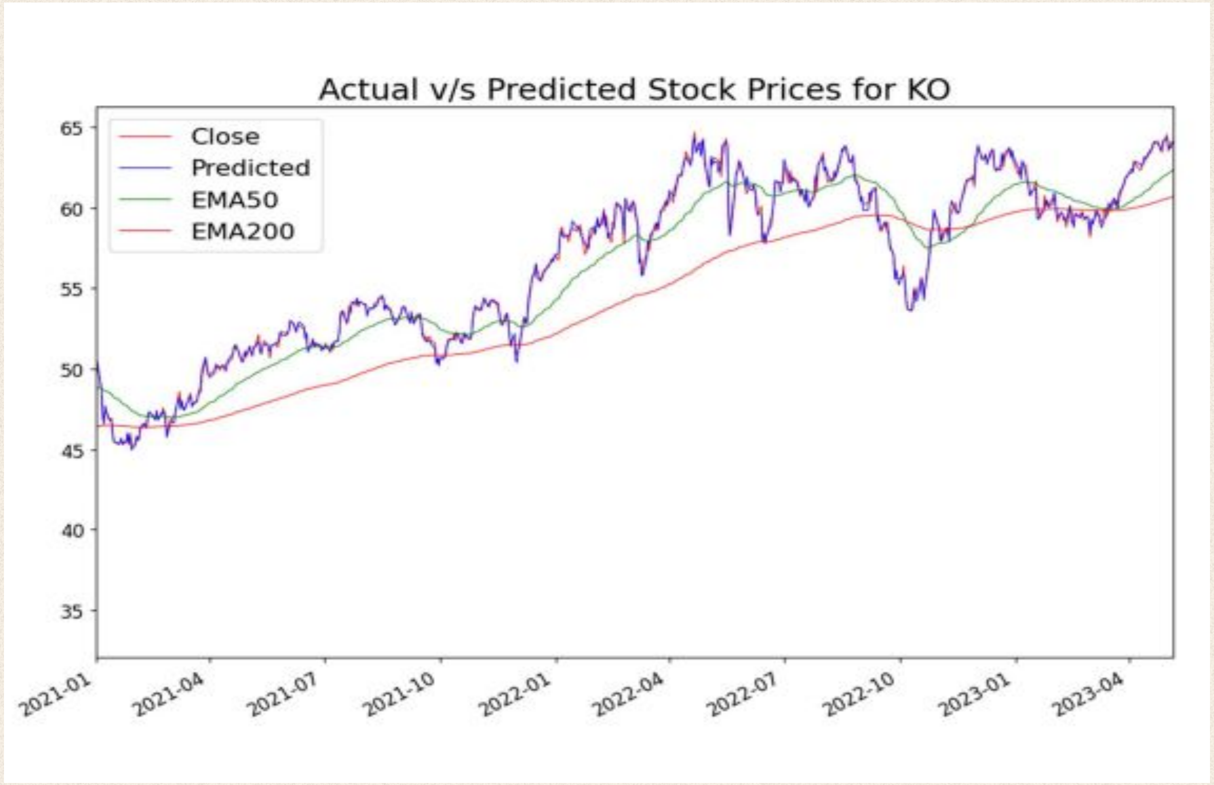
1. The above code creates a compare dataframe containing the Close, Predicted, EMA50, and EMA200 columns from the dataset.
2. It plots a line chart of the actual and predicted closing prices of the stock on a line chart with the 50-day and 200-day EMAs.
3. Plot is created using the plot method with the kind parameter set to 'line' and linewidth parameter set to 0.75.
4. The colors of the lines are set to blue for the Close, red for the Predicted, and green for the EMA50 and EMA200.
5. The title, x-axis label, y-axis label, and legend are added using the set_title, set_xlabel, set_ylabel, and legend methods.
6. The x-axis limits are set to show data from January 1, 2021, to May 6, 2023.
7. Finally, the line chart is displayed using the show method of the plt module.

Coca-Cola Company (KO)



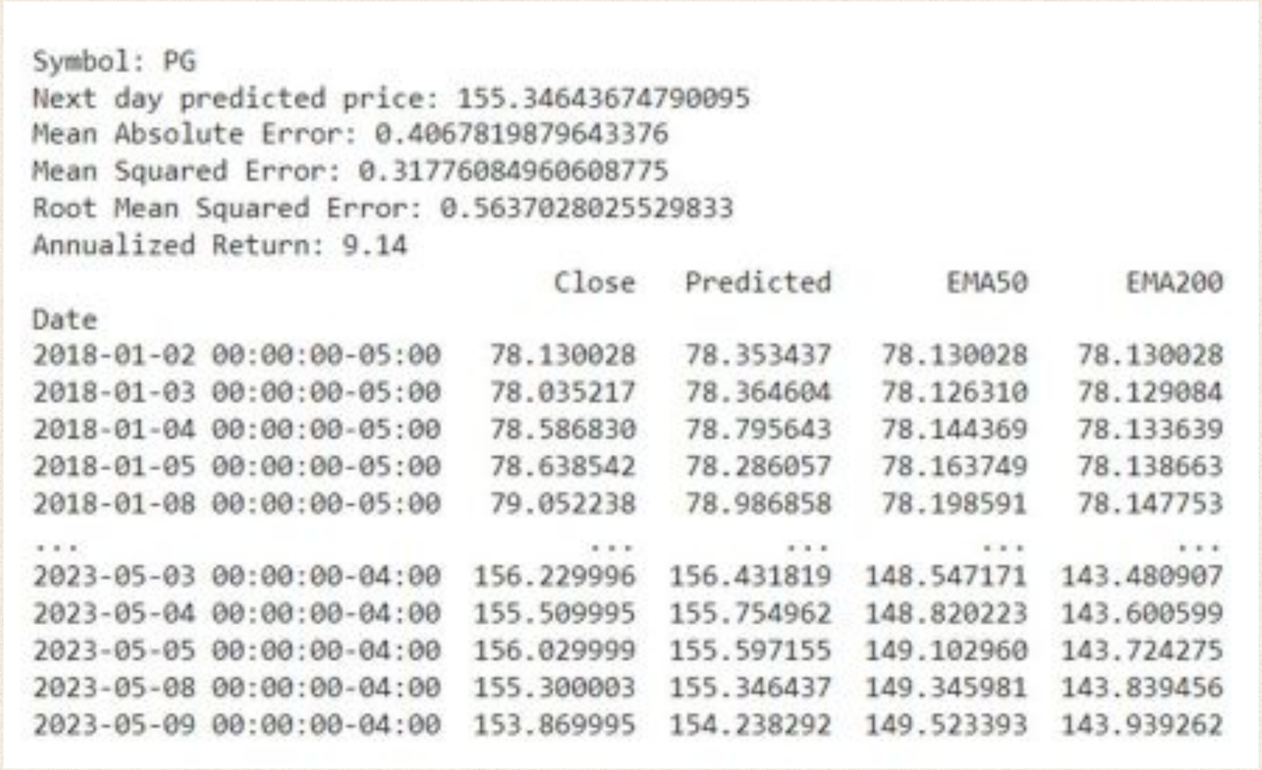
```
Symbol: KO
Next day predicted price: 63.8780573486709
Mean Absolute Error: 0.15806150739089306
Mean Squared Error: 0.04991032321062960 4
Root Mean Squared Error: 0.2234061843607504
Annualized Return: 6.64
                                Close  Predicted      EMA50     EMA200
Date
2018-01-02 00:00:00-05:00   38.548206  38.618709  38.548206  38.548206
2018-01-03 00:00:00-05:00   38.463554  38.535645  38.544887  38.547364
2018-01-04 00:00:00-05:00   39.005310  38.913415  38.562942  38.551921
2018-01-05 00:00:00-05:00   38.996830  38.919916  38.579958  38.556348
2018-01-08 00:00:00-05:00   38.937576  38.943743  38.593982  38.560141
...                               ...        ...        ...        ...
2023-05-03 00:00:00-04:00   63.650002  63.836462  62.177180  60.607028
2023-05-04 00:00:00-04:00   63.720001  63.741890  62.237683  60.638002
2023-05-05 00:00:00-04:00   64.019997  64.096141  62.307578  60.671654
2023-05-08 00:00:00-04:00   63.919998  63.878057  62.370810  60.703976
2023-05-09 00:00:00-04:00   63.450001  63.566505  62.413131  60.731300
```

GRAPH



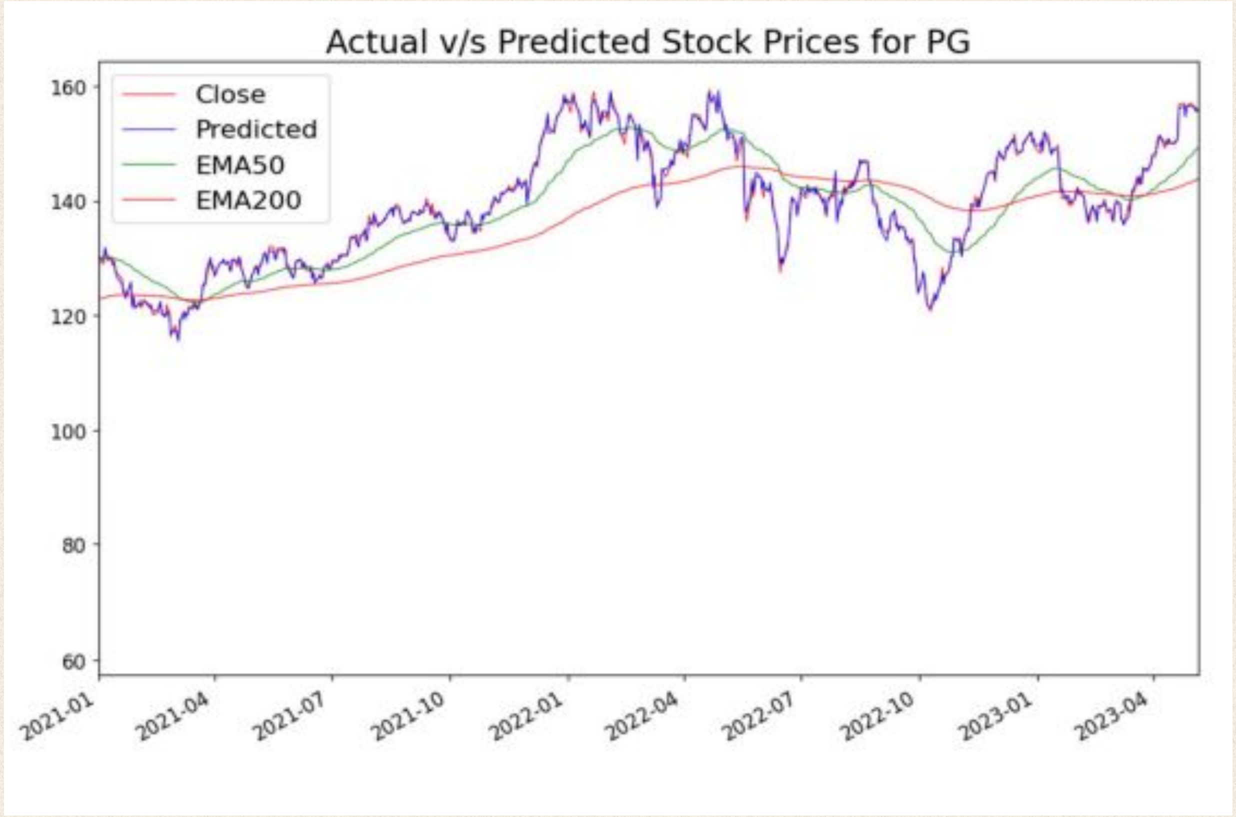
Actual v/s Predicted Stock Prices for KO
Close
Predicted
EMA50
EMA200
65
60
55
50
45
40
35
2021-01
2021-04
2021-07
2021-10
2022-01
2022-04
2022-07
2022-10
2023-01
2023-04

Procter & Gamble (PG)

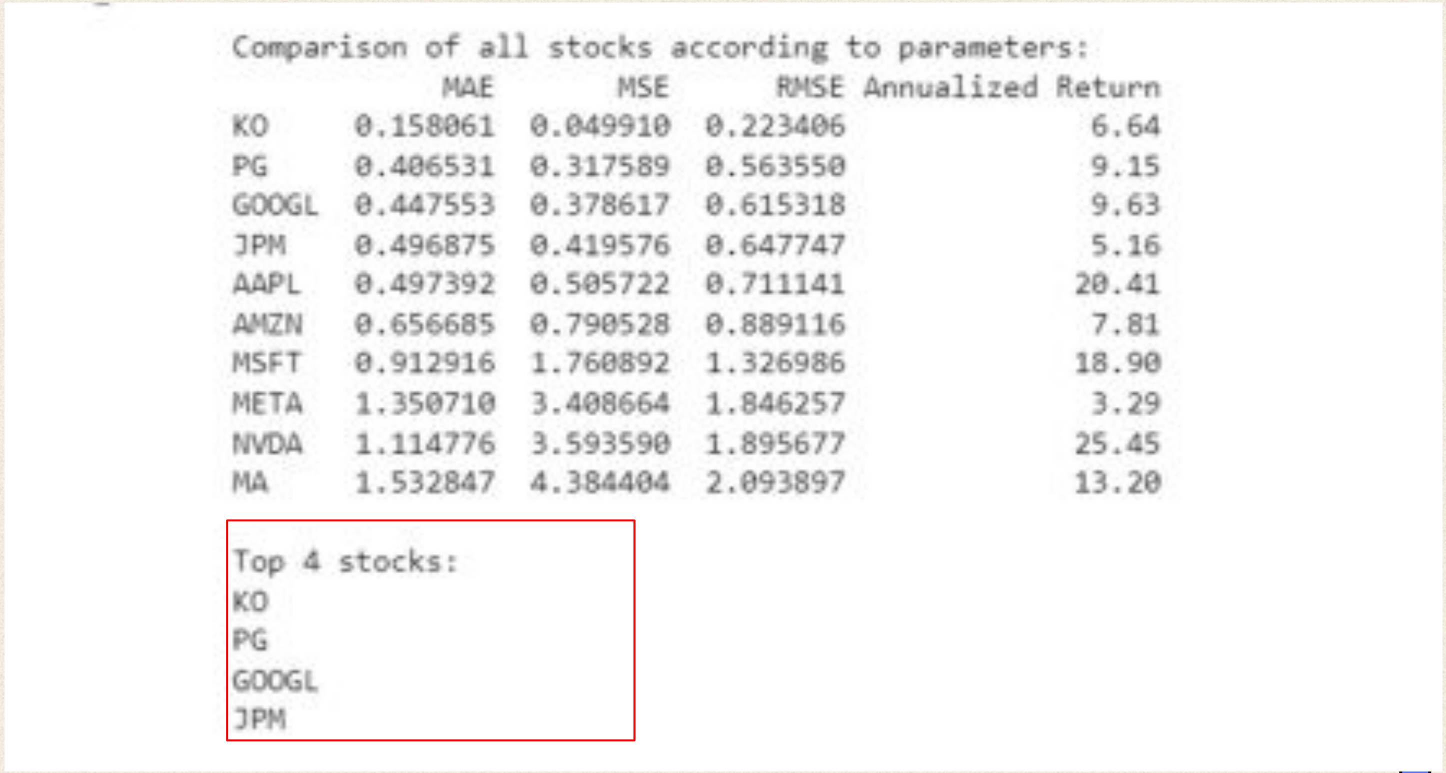


```
Symbol: PG
Next day predicted price: 155.34643674790095
Mean Absolute Error: 0.4067819879643376
Mean Squared Error: 0.31776084960608775
Root Mean Squared Error: 0.5637028025529833
Annualized Return: 9.14
                                Close   Predicted       EMA50      EMA200
Date
2018-01-02 00:00:00-05:00   78.130028   78.353437   78.130028   78.130028
2018-01-03 00:00:00-05:00   78.035217   78.364604   78.126310   78.129084
2018-01-04 00:00:00-05:00   78.586830   78.795643   78.144369   78.133639
2018-01-05 00:00:00-05:00   78.638542   78.286057   78.163749   78.138663
2018-01-08 00:00:00-05:00   79.052238   78.986858   78.198591   78.147753
...                               ...         ...         ...         ...
2023-05-03 00:00:00-04:00  156.229996  156.431819  148.547171  143.480907
2023-05-04 00:00:00-04:00  155.509995  155.754962  148.820223  143.600599
2023-05-05 00:00:00-04:00  156.029999  155.597155  149.102960  143.724275
2023-05-08 00:00:00-04:00  155.300003  155.346437  149.345981  143.839456
2023-05-09 00:00:00-04:00  153.869995  154.238292  149.523393  143.939262
```

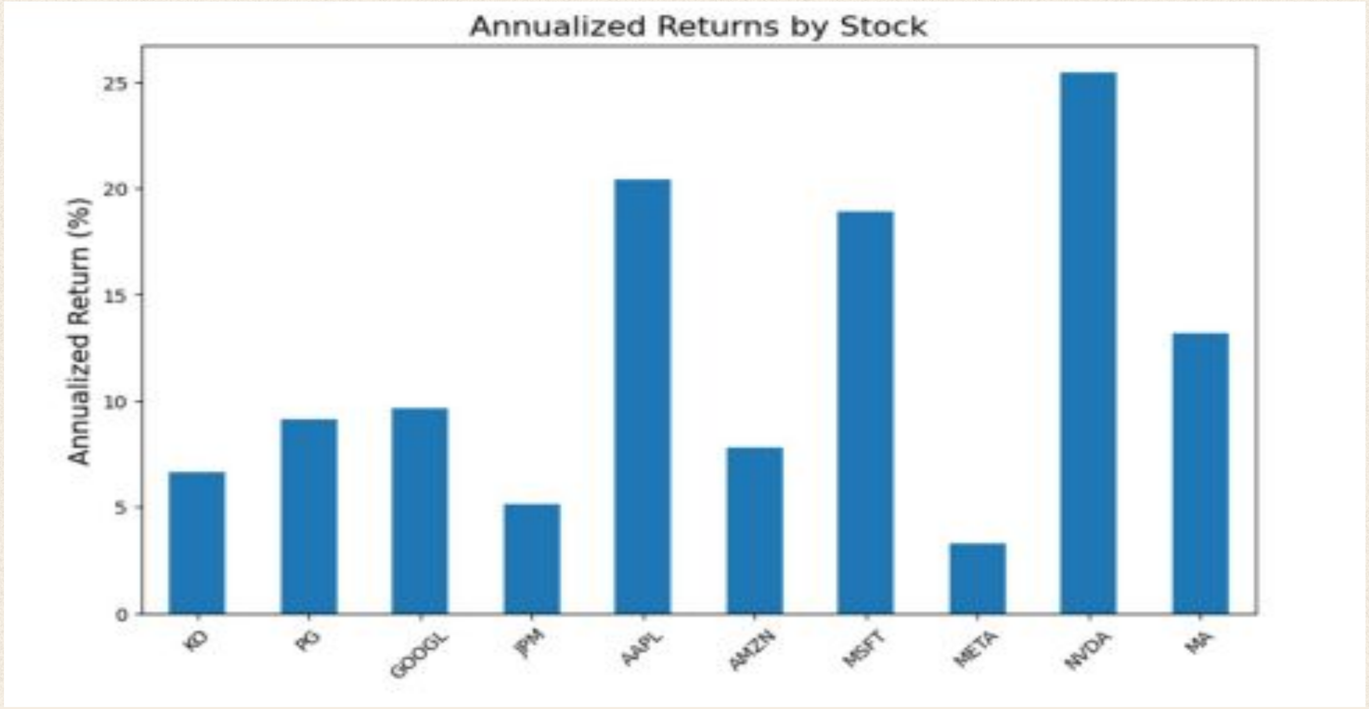
GRAPH


Actual v/s Predicted Stock Prices for PG
Close
Predicted
EMA50
EMA200
160
140
120
100
80
60
2021-01
2021-04
2021-07
2021-10
2022-01
2022-04
2022-07
2022-10
2023-01
2023-04

Comparison and Selection of Top Performing Stocks



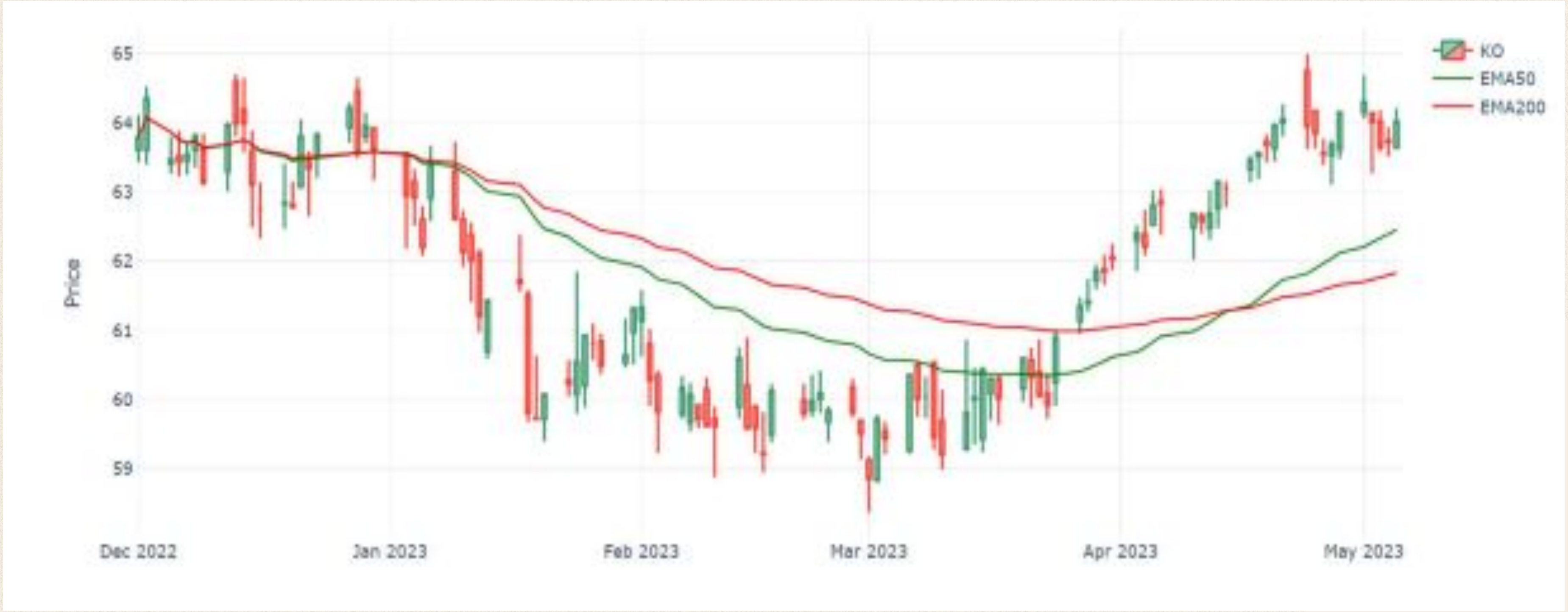
```
Comparison of all stocks according to parameters:
            MAE       MSE      RMSE  Annualized Return
KO     0.158061  0.049910  0.223406               6.64
PG     0.406531  0.317589  0.563550               9.15
GOOGL  0.447553  0.378617  0.615318               9.63
JPM    0.496875  0.419576  0.647747               5.16
AAPL   0.497392  0.505722  0.711141              20.41
AMZN   0.656685  0.790528  0.889116               7.81
MSFT   0.912916  1.760892  1.326986              18.90
META   1.350710  3.408664  1.846257               3.29
NVDA   1.114776  3.593590  1.895677              25.45
MA     1.532847  4.384404  2.093897              13.20

Top 4 stocks:
KO
PG
GOOGL
JPM
```

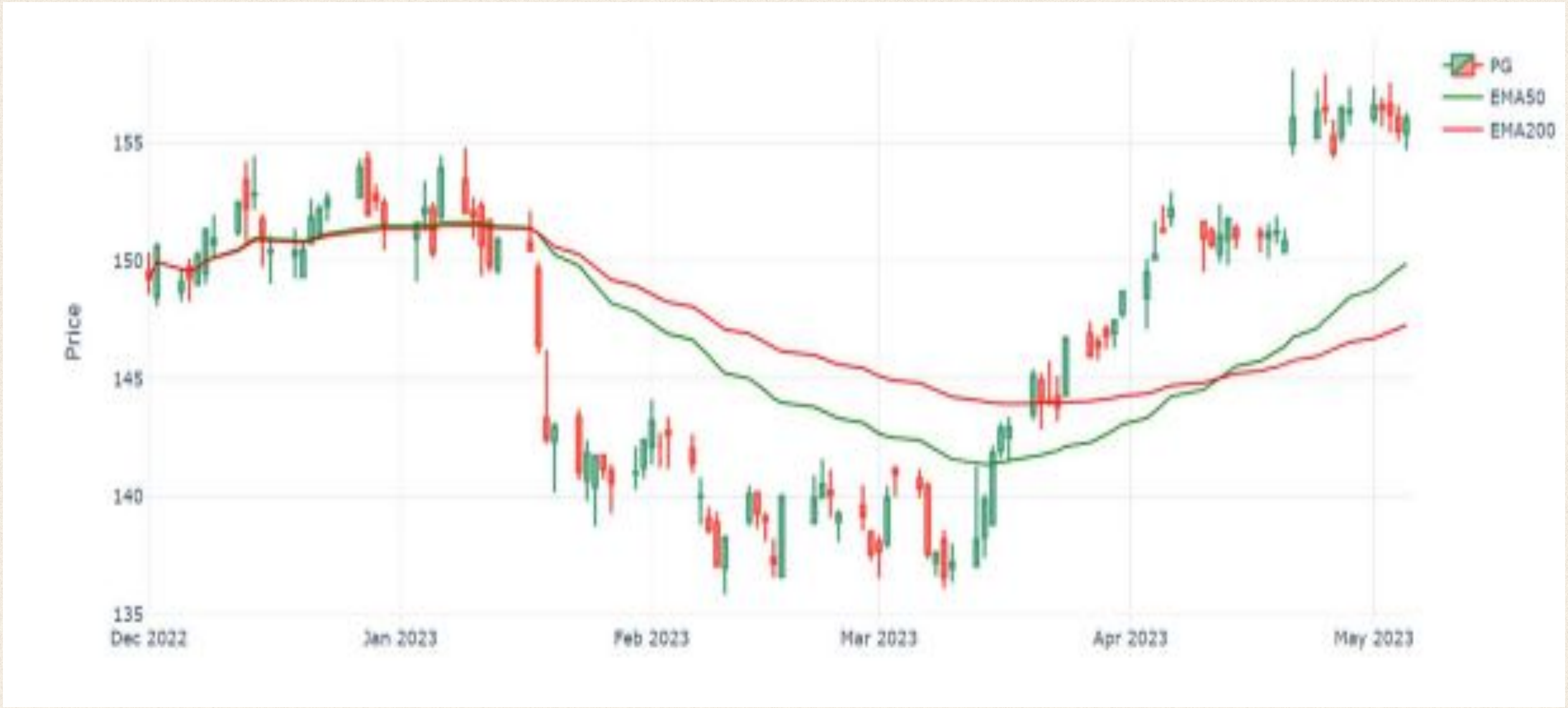


1. Comparing the results of different stocks based on evaluation metrics.
2. The evaluation metrics used are root mean square error (RMSE), mean absolute error (MAE), mean absolute percentage error (MAPE), and annualized return.
3. The results are presented in a Pandas DataFrame sorted in ascending order based on the RMSE values and the top 4 stocks based on the RMSE values are printed.
4. This can help identify stocks that perform well according to the specified evaluation metrics, which can be useful for further analysis or investment purposes.

Creating Candlestick charts with EMA lines

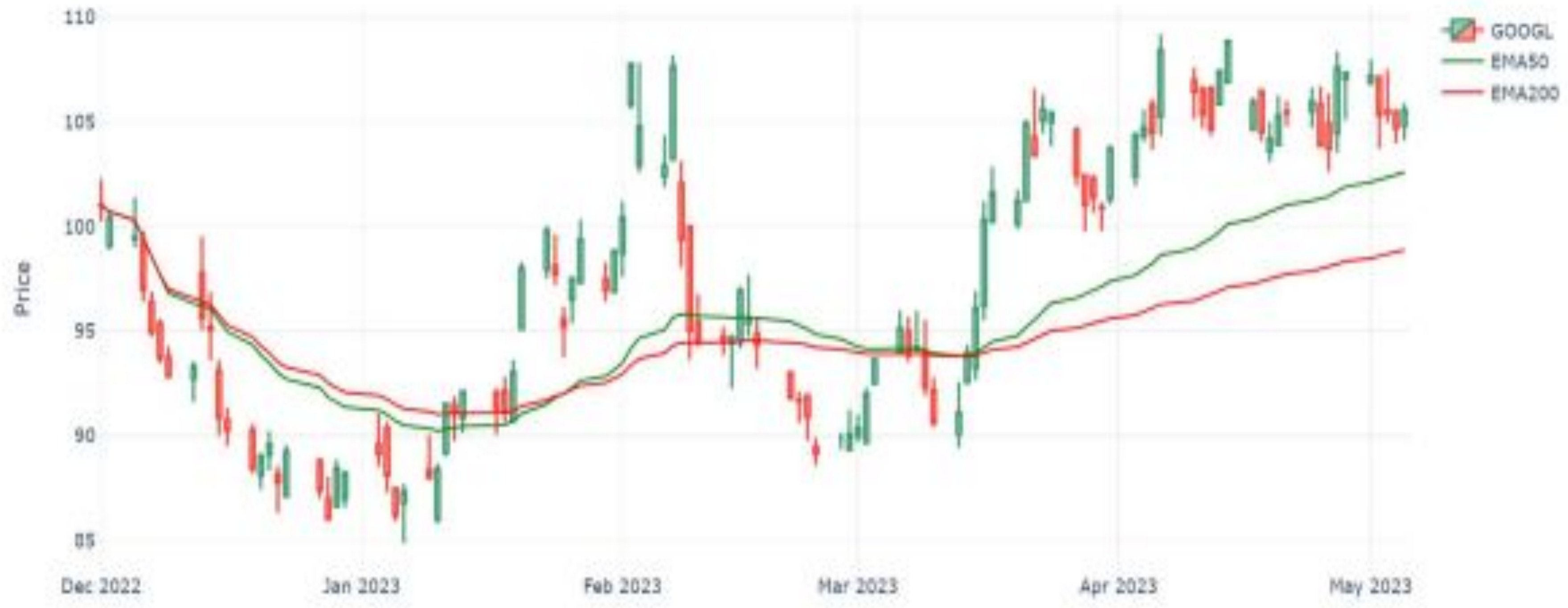
I. KO





PG
EMA50
EMA200
Price
155
150
145
140
135
Dec 2022
Jan 2023
Feb 2023
Mar 2023
Apr 2023
May 2023

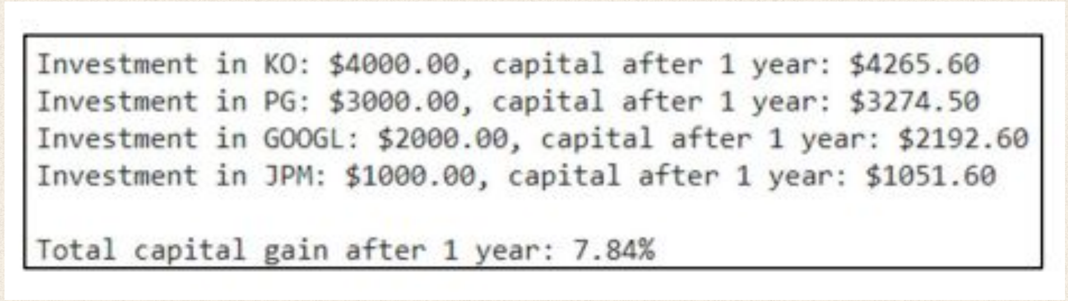
II. PG



110
105
100
95
90
85
Price
Dec 2022
Jan 2023
Feb 2023
Mar 2023
Apr 2023
May 2023
GOOGL
EMA50
EMA200

III. GOOGL

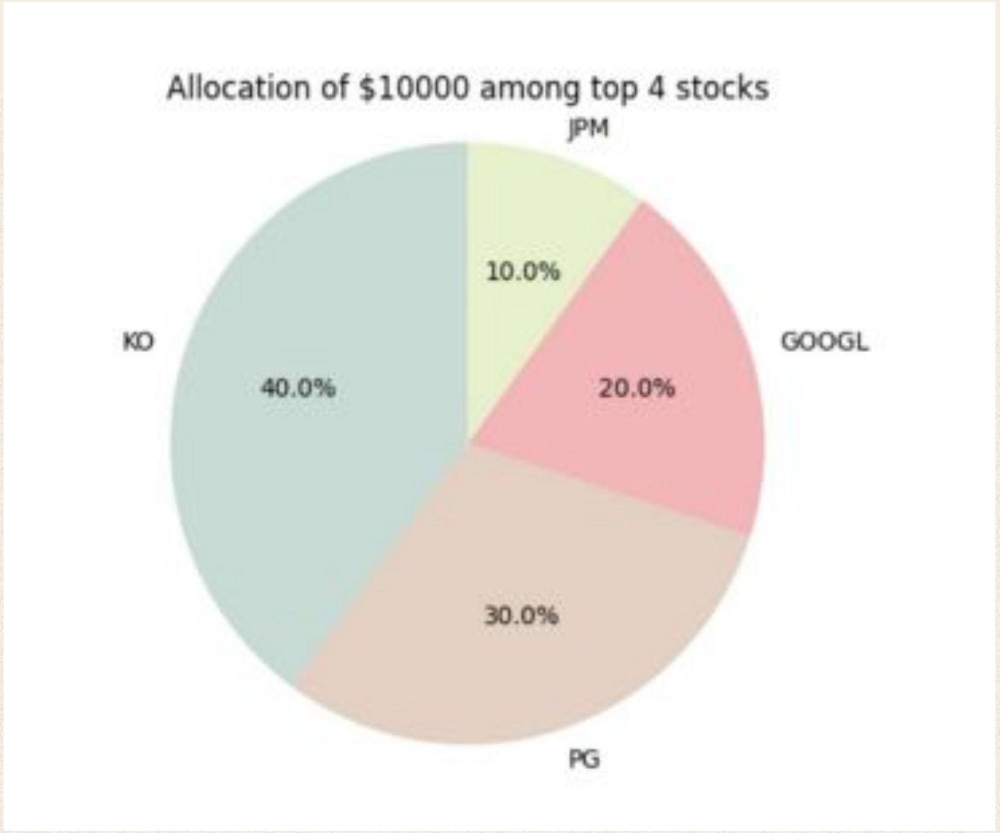
Allocation and Visualization of Investment in Top 4 Stocks



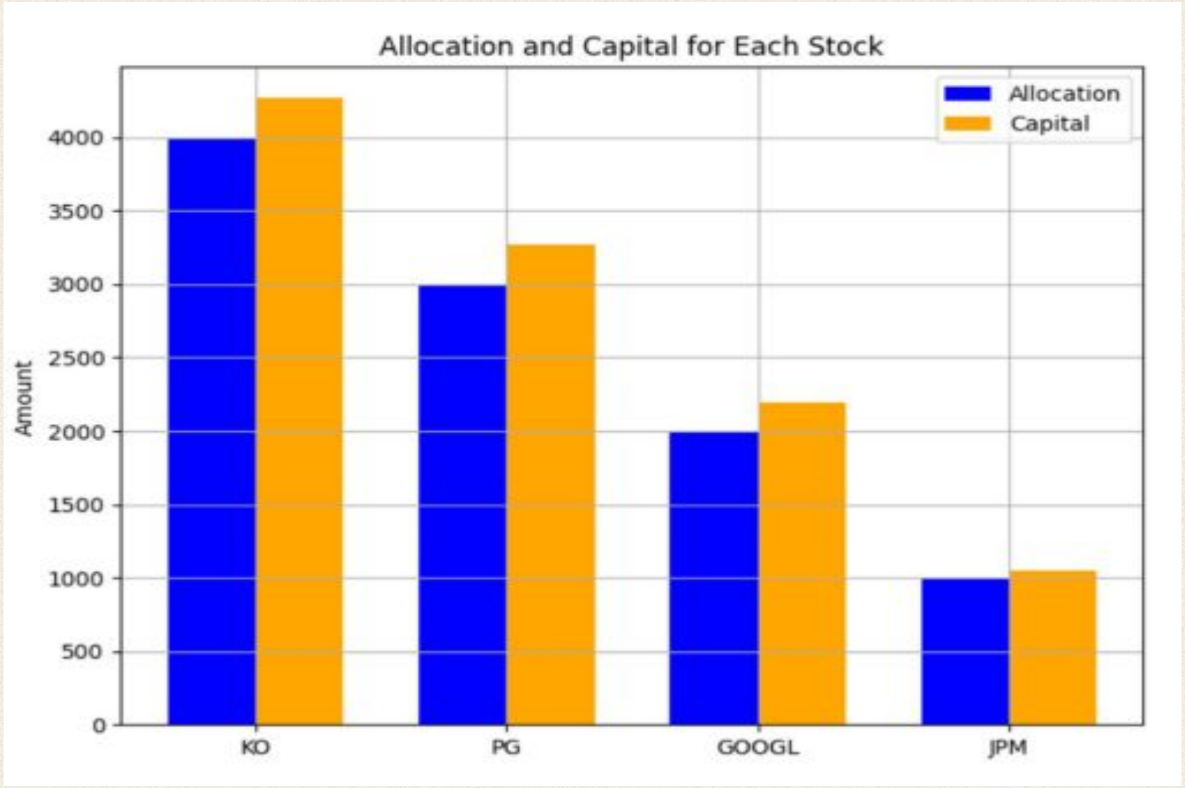
```
Investment in KO: $4000.00, capital after 1 year: $4265.60
Investment in PG: $3000.00, capital after 1 year: $3274.50
Investment in GOOGL: $2000.00, capital after 1 year: $2192.60
Investment in JPM: $1000.00, capital after 1 year: $1051.60

Total capital gain after 1 year: 7.84%
```

1. This code implements a portfolio allocation strategy for a set of stocks.
2. It selects the top 4 stocks based on their annualized returns.
3. It allocates a certain percentage of a $10,000 investment to each of the top 4 stocks, which are hardcoded as 0.4, 0.3, 0.2, and 0.1, respectively.
4. The code calculates the capital gain after one year for each stock based on the allocated investment, the annualized return of the stock, and the compounding effect.
5. It then prints the investment and capital gain for each stock and the total capital gain for the portfolio after one year.

Finally, it creates a pie chart to visualize the allocation of the $10,000 investment among the top 4 stocks.



Creating a bar plot for Allocation and Capital for Each Stock



1. The code creates a portfolio allocation strategy for the top 4 performing stocks based on annualized returns.
2. It calculates the capital gain after one year for each stock.
3. It visualizes the allocation and capital for each stock in a bar plot using matplotlib.
4. This can aid in investment decision-making and visualization of the performance of the selected stocks.

FUTURE IMPROVEMENTS

1. Implement a web application
2. Incorporate more data sources
3. Optimize the algorithm
4. Implement trading strategies
5. Expand to other markets

SUMMARY

1. The project creates recommendation system suggesting best 4 stocks out of 10.
2. Historical data and machine learning algorithms used to identify market trends.
3. Candlestick charts used to visualize stock data and identify trading opportunities.
4. Charts display moving averages of 50 and 200 and color-coded to highlight trends.
5. Aimed at aiding investors and traders in making informed decisions and improving strategies.
6. Future work includes integrating additional technical indicators and expanding dataset.

REFERENCES

1. Yahoo Finance API: https://finance.yahoo.com/
2. Pandas documentation: https://pandas.pydata.org/pandas-docs/stable/index.html
3. Matplotlib documentation: https://matplotlib.org/stable/contents.html
4. mplfinance documentation: https://github.com/matplotlib/mplfinance
5. Plotly documentation: https://plotly.com/python/
6. Technical Analysis Library in Python (TA-Lib): https://github.com/mrjbq7/ta-lib
7. Chat-gpt: https://chat.openai.com/

Thank You