A COMPREHENSIVE MARKET ANALYSIS MODEL USING PYTHON

Jahnavi Ravi

Table of Contents

Introduction

With the increasing interest in the stock market, investors are continuously looking for ways to analyze data and make informed investment decisions. The vast amounts of financial data that are generated every day require advanced tools and techniques to extract valuable insights. To address this need, our team developed a comprehensive market analysis program in Python that will offer users a range of functions to analyze and learn from stock data of a particular stock.

Python has become one of the most popular programming languages for data analysis, thanks to its simplicity, flexibility, and a large number of libraries that cater to data analysis and visualization. The program is written in Python, and uses various libraries such as Pandas, Matplotlib, mpl_finance, and Scipy. In this report, we will analyze the code that deals with data retrieval, data visualization, technical analysis, and financial metrics, and demonstrate how it can be used to analyze stock data effectively.

This model intends to offer a powerful combination of data analysis, visualization, and machine learning capabilities, making it a preferred choice for investment firms, hedge funds, and individual investors. A Comprehensive Market Analysis Model will enable investors to analyze a wide range of financial data and provide them with the tools necessary to make informed investment decisions.

The model will enable investors to load csv files containing stock data and financial information of a stock, visualize the data through charts, and analyze the data using various technical indicators and financial metrics. This will not only help investors to identify market trends and patterns but also to identify risks and opportunities in the market.

In addition, Python's object-oriented programming paradigm makes the code modular, reusable, and easy to maintain, which is crucial for complex market analysis models. Moreover, the error handling mechanisms that we have included in the code will ensure that the model is reliable and robust, reducing the chances of potential errors that could lead to incorrect analysis and decision-making.

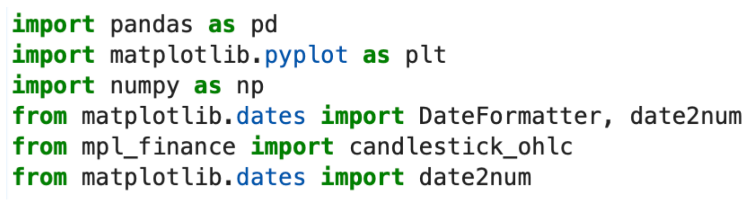
The market analysis model will enable investors to make informed decisions, reduce risks, and capitalize on market opportunities. It will provide investors with a competitive edge in the dynamic and ever-changing financial markets, making it a crucial tool for investors looking to maximize their returns.

Methodology

In today's world, data visualization has become an essential aspect of decision-making in various fields, including finance. The main agenda of our model is to provide a comprehensive analysis of stock data, including visualizations and financial metrics. The code can read data from a CSV file and perform various data visualization tasks such as plotting line charts, candlestick charts, and visualizing features. The code also performs technical analysis tasks such as calculating moving averages, RSI, and MACD. Additionally, the code calculates financial metrics such as P/E ratio and dividend yield.

Once we have created a basic code for all the packages that we require, we then made sure that our model could handle various errors that might arise when a user is analyzing data. Finally, we have created instances of usage for the code and used the stock market prices of Apple Inc. over the past year and its financial data to check if our code was working as intended. The plots and output that will be seen further in the report are the outputs from our testing.

Libraries and functions used



```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.dates import DateFormatter, date2num
from mpl_finance import candlestick_ohlc
from matplotlib.dates import date2num
```

1. import pandas as pd: This imports the pandas library and assigns it an alias pd. Pandas is a powerful data manipulation library that provides data structures for efficiently storing and manipulating data in Python.
2. import matplotlib.pyplot as plt: This imports the pyplot module from the matplotlib library and assigns it an alias plt. Matplotlib is a plotting library for creating static, animated, and interactive visualizations in Python. The pyplot module provides a convenient interface for creating basic plots and visualizations.
3. import numpy as np: This imports the numpy library and assigns it an alias np. Numpy is a numerical computing library for Python that provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions.
4. from matplotlib.dates import DateFormatter, date2num: This imports the DateFormatter and date2num functions from the dates module of the matplotlib library. These functions are used for formatting dates in matplotlib plots.
5. from mpl_finance import candlestick_ohlc: This imports the candlestick_ohlc function from the mpl_finance module. This function is used to plot financial charts, such as candlestick charts, in matplotlib.

6. from matplotlib.dates import date2num: This imports the date2num function from the dates module of the matplotlib library. This function is used for converting dates to numerical values that can be plotted on a graph.

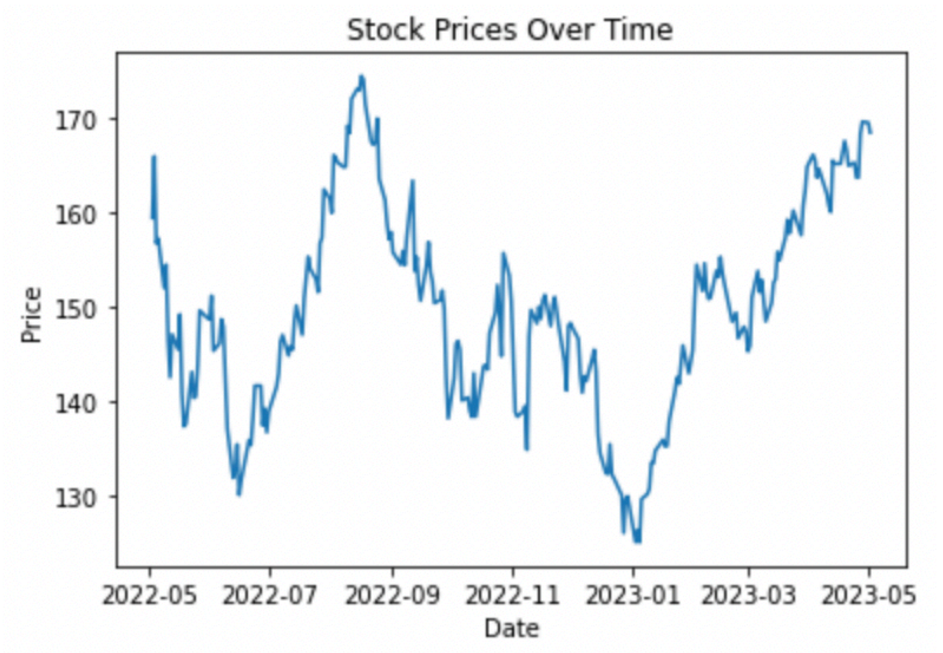
Data Visualization

Data visualization is a crucial aspect of any market analysis model. With the vast amount of data available, it can be challenging for analysts to draw insights and make informed decisions. Data visualization techniques help to represent data in a way that is easy to understand and interpret, enabling analysts to identify trends and patterns quickly.

The constructor of the class **DataVisualization** takes a CSV file containing stock data and reads it into a Pandas DataFrame. The date column is converted to a DateTime object and set as the index of the DataFrame, making it easy to manipulate and plot time-series data. The class then provides several visualization methods to plot line charts, candlestick charts, and simple feature visualizations.

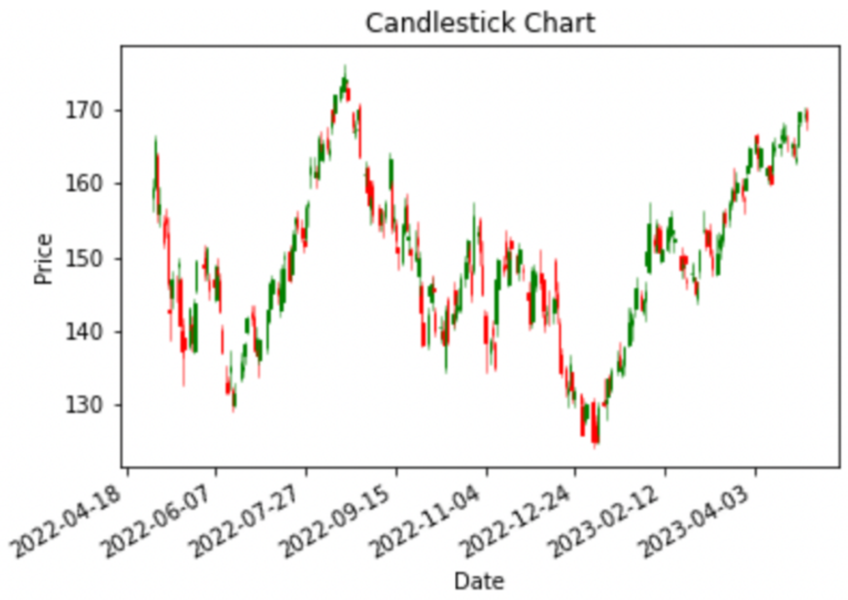
Line Chart

The plot_line_chart() method plots a line chart of the closing prices of the stock over time. This method is essential in providing an overview of how the stock has performed over time, allowing investors to track trends and identify patterns.



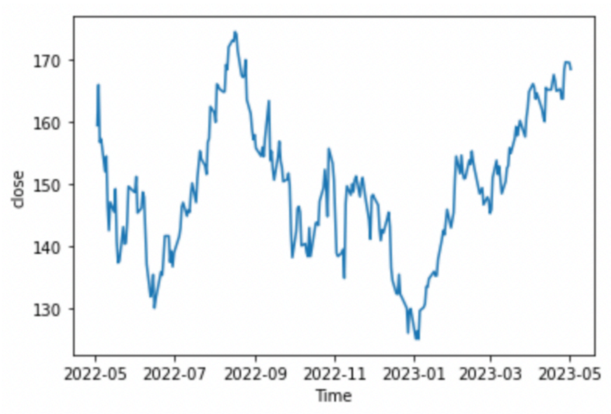
Candlestick Chart

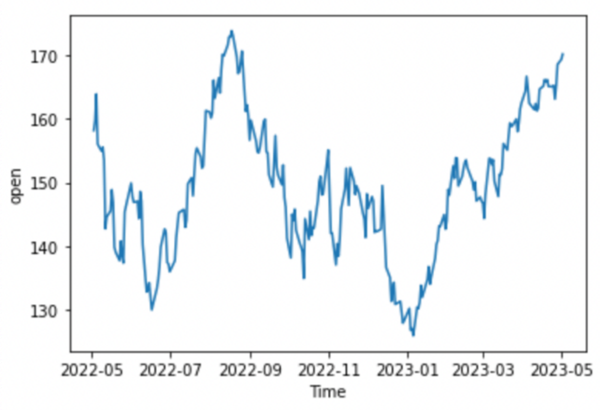
The plot_candlestick_chart() method plots a candlestick chart, which is a popular chart type used to visualize stock prices. This method is important because it provides a comprehensive view of the stock's price action over time, showing the opening and closing prices as well as the highs and lows.



Visualize Method Charts

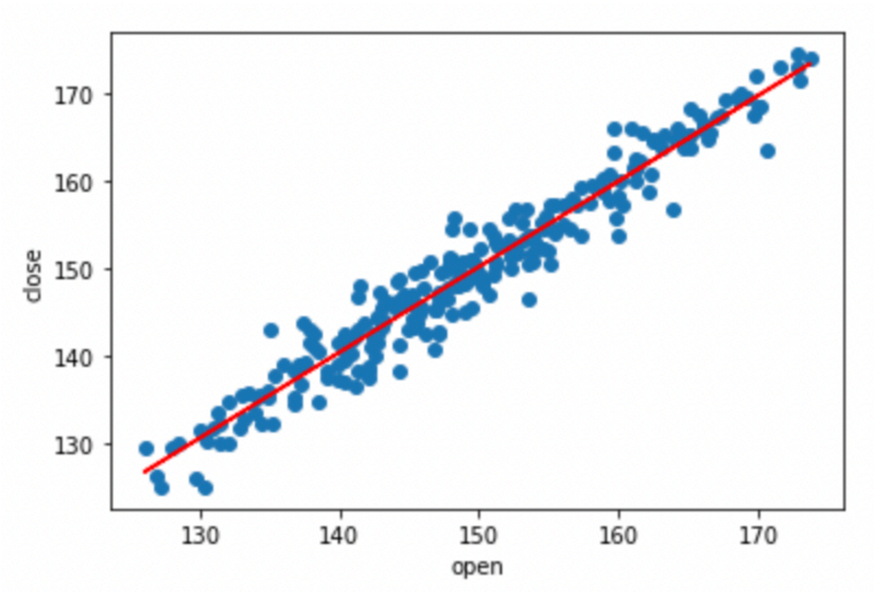
The visualize() method allows users to plot any feature of the stock data over time, giving them the flexibility to explore the data and identify patterns that may be useful in their investment decisions.





Linear Regression Plot

The linear_regression() method fits a linear regression model to two columns of the stock data and plots the resulting line along with the scatter plot of the data points. This method is useful in identifying the relationship between two features of the stock data and can be used to make predictions about future prices.



Calculate MSE

The calculate_mse() method calculates the mean squared error between two columns of the stock data, which is a measure of how well the linear regression model fits the data. This method is important in assessing the accuracy of the model and determining whether it is appropriate for making predictions.

```
MSE: 6.718447171822981
```

Perform a t-Test

Finally, the t_test() method performs a t-test on two columns of the stock data, which is a statistical test used to determine whether there is a significant difference between the means of the two columns. This method is important in assessing whether two features of the stock data are related and can be used to inform investment decisions.

```
T-Statistic: -0.24867162398384282, P-Value: 0.8037169065419512
```

In summary, the DataVisualization class provides users with a clear understanding of stock price trends and relationships between features, this model can help investors make informed decisions about where to invest their money.

Technical Analysis

We have then included a class to perform technical analysis tasks on the stock data, including calculating moving averages, RSI, and MACD. The code consists of a TechnicalAnalysis class that contains methods for calculating moving averages, relative strength index (RSI), and moving average convergence divergence (MACD) of stock data.

The first important aspect of this code is the use of the Pandas library. Pandas is a powerful library for data manipulation and analysis in Python. It provides data structures for efficiently storing and manipulating large datasets, making it an ideal choice for analyzing stock data.

The TechnicalAnalysis class is defined with an **init** method that takes in a Pandas DataFrame containing the stock data. It checks if the input data is a Pandas DataFrame and then stores it in a class variable. This approach makes the code more modular and easier to use since the user only needs to provide the stock data to an instance of the TechnicalAnalysis class.

The main functionalities of the TechnicalAnalysis class are as follows along with the example output that we have generated using stock prices of Apple Inc.

Calculating the Moving Average

The calculate_moving_average method uses the rolling method of the Pandas DataFrame to calculate the moving average of the 'close' column over a specified window size. Moving averages are used to smooth out fluctuations in price data and are commonly used to identify trends in the market. They are calculated by taking the average closing price of a stock over a specified time period, such as 10 days, 20 days, or 50 days.

In this case, the moving average with a window size of 20 days is calculated for the 'close' column of the stock_data dataframe. The first few rows of the moving average output contain NaN values because it takes some time for the moving average to be calculated.

Specifically, the first 19 rows of the moving average output are NaN because the window size is set to 20 days. For each day in the first 19 days, there are not enough previous days' data available to calculate the 20-day moving average. Therefore, the result is NaN for these days.

```
# Calculate the moving average with a window size of 20 days
moving_average = ta.calculate_moving_average(20)
moving_average
```

```
0             NaN
1             NaN
2             NaN
3             NaN
4             NaN
          ...
246    164.290501
247    164.672501
248    165.038500
249    165.273000
250    165.391500
Name: close, Length: 251, dtype: float64
```

Calculating the Relative Strength Index

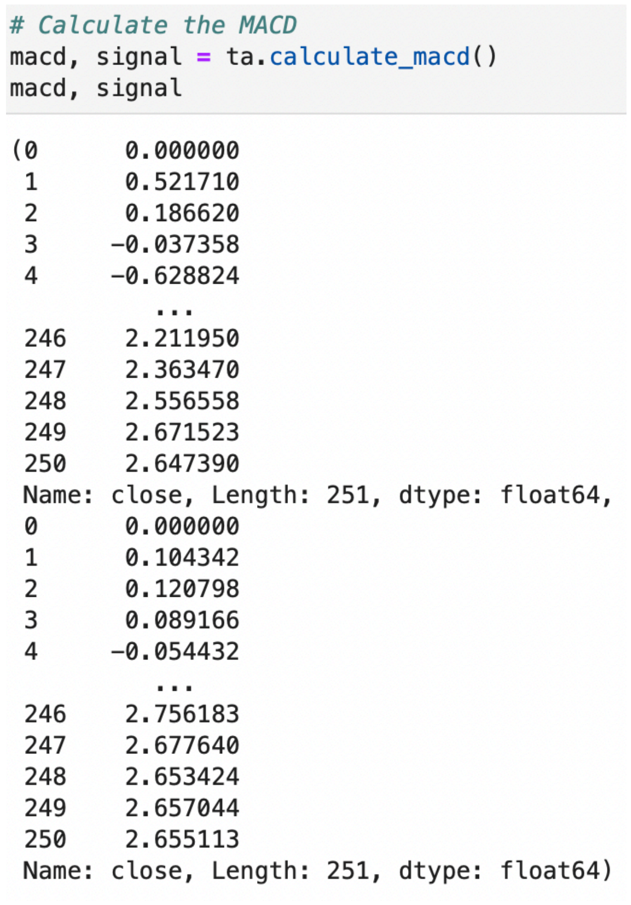
The calculate_rsi method calculates the RSI of the stock data using the difference between the closing prices and the rolling averages of gains and losses. RSI, or Relative Strength Index, is a momentum indicator that compares the magnitude of recent gains to recent losses in an attempt to determine overbought and oversold conditions of an asset.

```
# Calculate the RSI with a window size of 14 days
rsi = ta.calculate_rsi(14)
rsi
```

```
0            NaN
1            NaN
2            NaN
3            NaN
4            NaN
         ...
246    50.000000
247    58.549934
248    68.595047
249    72.619667
250    71.334665
Name: close, Length: 251, dtype: float64
```

Calculating the Moving Average Convergence Divergence

The calculate_macd method calculates the MACD using the exponential moving averages of the closing prices over specified time periods. MACD, or Moving Average Convergence Divergence, is another momentum indicator that uses moving averages to identify changes in trend.



```python
# Calculate the MACD
macd, signal = ta.calculate_macd()
macd, signal
```

```
(0      0.000000
 1      0.521710
 2      0.186620
 3     -0.037358
 4     -0.628824
          ...
 246    2.211950
 247    2.363470
 248    2.556558
 249    2.671523
 250    2.647390
 Name: close, Length: 251, dtype: float64,
 0      0.000000
 1      0.104342
 2      0.120798
 3      0.089166
 4     -0.054432
          ...
 246    2.756183
 247    2.677640
 248    2.653424
 249    2.657044
 250    2.655113
 Name: close, Length: 251, dtype: float64)
```

The code also contains error handling for cases where the stock data is not in the correct format, or if there is an error while performing the calculations. The code checks for the existence of the 'close' column and raises a KeyError if it is not found. It also raises a TypeError if the input data is not a Pandas DataFrame.

Financial Metrics

In addition to data visualization and technical analysis, the FinancialMetrics class that provides methods to calculate different financial metrics such as the P/E ratio, P/S ratio, and total return. These metrics are widely used in the finance industry to evaluate the performance and valuation of companies.

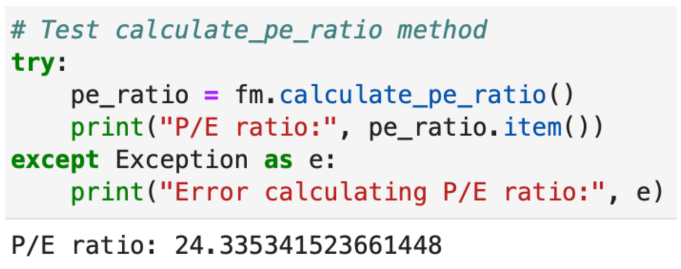
The FinancialMetrics class is initialized with two parameters, stock_data, and financial_statements, both of which must be pandas DataFrames. These DataFrames are assumed to contain the relevant financial data for the analysis. The class methods then use this data to calculate the desired metrics.

Calculating the Price-to-Earnings (P/E) Ratio

The calculate_pe_ratio method calculates the price-to-earnings (P/E) ratio, a widely used valuation ratio in the finance industry. It is calculated by dividing the average stock price by the basic earnings per share (EPS) of the company.

This metric is commonly used to determine the relative value of a stock and can be used to compare companies within the same industry. A higher P/E ratio indicates that investors are willing to pay more for each dollar of earnings, which may indicate a growth opportunity or market optimism. Conversely, a lower P/E ratio may indicate that investors have less confidence in the company's future prospects.

The method first calculates the average close price of the stock_data DataFrame and then divides it by the Basic EPS column of the financial_statements DataFrame. If any required column is missing, a KeyError is raised. If any other error occurs during the calculation, an Exception is raised.



```
# Test calculate_pe_ratio method
try:
    pe_ratio = fm.calculate_pe_ratio()
    print("P/E ratio:", pe_ratio.item())
except Exception as e:
    print("Error calculating P/E ratio:", e)
```

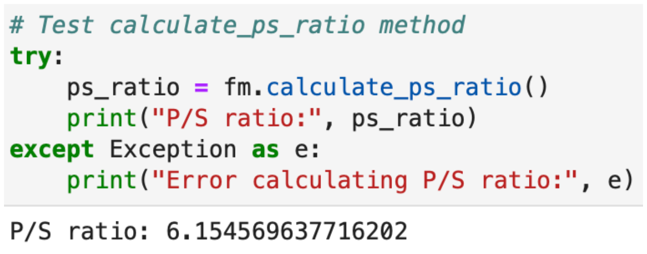
```
P/E ratio: 24.335341523661448
```

Calculating the Price-to-Sales (P/S) Ratio

The calculate_ps_ratio method calculates the price-to-sales (P/S) ratio, another widely used valuation ratio in the finance industry. It is calculated by dividing the market capitalization by the total revenue of the company.

This metric is used to determine the value that the market places on a company's sales. A higher P/S ratio may indicate that investors are optimistic about the company's future sales growth, while a lower P/S ratio may indicate that investors have less confidence in the company's sales prospects.

The method first calculates the average close price of the stock_data DataFrame and the basic average shares from the financial_statements DataFrame. It then multiplies the two to get the market capitalization and divides it by the Total Revenue column of the financial_statements DataFrame. If any required column is missing, a KeyError is raised, and if the DataFrame is empty, an IndexError is raised. If any other error occurs during the calculation, an Exception is raised.



```
# Test calculate_ps_ratio method
try:
    ps_ratio = fm.calculate_ps_ratio()
    print("P/S ratio:", ps_ratio)
except Exception as e:
    print("Error calculating P/S ratio:", e)
```

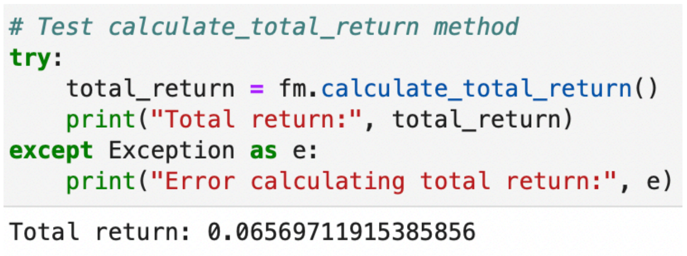
```
P/S ratio: 6.154569637716202
```

Calculating the Total Return of a Stock

The calculate_total_return method calculates the total return of the stock over a given period. It is calculated by dividing the difference between the final price and the initial price by the initial price. The method first gets the initial and final prices from the open and close columns of the stock_data DataFrame, respectively. If any error occurs during the calculation, an Exception is raised.

This metric is used to evaluate the overall performance of a stock over a given period of time. A higher total return indicates that the investment has performed well, while a lower total return indicates that the investment has underperformed.

In summary, the FinancialMetrics class provides a convenient and reusable way to calculate financial metrics for different stocks. These financial metrics provide a comprehensive view of a company's financial performance and valuation, allowing investors to make informed decisions about whether to buy, hold, or sell a stock. The code also handles errors that might occur during the calculation of the metrics, making it more robust and reliable.



```
# Test calculate_total_return method
try:
    total_return = fm.calculate_total_return()
    print("Total return:", total_return)
except Exception as e:
    print("Error calculating total return:", e)
```

```
Total return: 0.06569711915385856
```

Conclusion

In conclusion, the Comprehensive Market Analysis Model in Python offers a wide range of functions for analyzing and learning from stock data of a particular stock. The program is written in Python and uses various libraries such as Pandas, Matplotlib, mpl_finance, and Scipy.

The model enables investors to load csv files containing stock data and financial information of a stock, visualize the data through charts, and analyze the data using various technical indicators and financial metrics. Python's object-oriented programming paradigm makes the code modular, reusable, and easy to maintain.

Moreover, the error handling mechanisms included in the code ensure that the model is reliable and robust, reducing the chances of potential errors that could lead to incorrect analysis and decision-making.

This market analysis model provides investors with a competitive edge in the dynamic and ever-changing financial markets, making it a crucial tool for investors looking to maximize their returns.