Title

Standard & Poor’s 500 Index: A Trading Forecasting Analysis through Generative Artificial Intelligence

Journal

UC Merced Undergraduate Research Journal, 17(1)

Authors

Vagish, Adhya
Rao, Aditya

Publication Date

2024

DOI

10.5070/M417164617

Issue 17, Volume 1 December 2024

Standard & Poor's 500 Index: A Trading Forecasting Analysis through Generative Artificial Intelligence

Adhya Vagish and Aditya Rao

ACKNOWLEDGEMENTS

This paper was written for WRI 010 with Phillip Lovas, originally submitted on November 28, 2023.

Standard & Poor's 500 Index: A Trading Forecasting Analysis through Generative Artificial Intelligence

Aditya Rao and Adhya Vagish

University of California, Merced

Writing 010

Phillip Lovas

November 28th, 2023

Abstract

In November 2022, the world of artificial intelligence, programming, and efficiency changed forever, as OpenAI created the first-ever publicly accessible large language generative chatbot: Generative Pre-trained Transformed (GPT)-3.5 (Open AI, 2022). The bot passed several Advanced Placement course exams, which are tests high school students can take to obtain college credit. It passed graduate-level exams such as the GRE, and even the BAR exam required to become a professional lawyer (Open AI, 2023). With all of GPT's success, the specific issue with OpenAI's model, GPT-3.5, is that it cannot access the internet or fetch real-time data (Open AI, 2022). The challenge we undertook was to use the GPT chatbot to create a stock-prediction trading algorithm, guiding the model to provide a conclusive output, and limiting our influence on the model—outside of errors—as much as possible. From November 2 to November 17 of 2023, we manually compared GPT's predictions to the actual results of ten stocks within a trading day (6:30 AM PST – 1:00 PM PST). It has been widely concluded in the past that GPT models are unable to make daily stock predictions accurately. Past researchers suggest, "It is too soon to claim AI can beat the stock markets" (Mokhtari, 2021) as they perform better in the long term, which is why we are testing their short-term and long-term capabilities. From the tests we ran to evaluate GPT's capabilities, we conclude there is great value in incorporating artificial intelligence into current trading models. However, generative AI models like GPT cannot be solely relied upon for accurate predictions. Models built with AI can help advise full-time stock traders, casual investors, and large trading firms about the effectiveness of AI models in their technical analysis before investing in a stock.

Keywords: Artificial Intelligence, stock prediction, GPT (Generative Pre-trained Model), quantitative analysis

Standard & Poor's 500 Index: A Trading Forecasting Analysis Through Generative Artificial Intelligence

Imagine if doctors could detect brain tumors by looking at hundreds of scans. That idea seems far-fetched for humans, but not so much for Artificial Intelligence (AI). AI is an emerging industry in technology, taking the modern world by storm with its ability to perform mundane and complex tasks arguably more efficiently than humans. It can be used for home improvement, programming assistance, text classification, image generation, disease prevention—tasks of all sorts. What if it could be somewhat accurate in providing financial stock predictions? Our research aims to delve deeper into the relationship between the stock market and LLMs (Large Language Models), a form of AI, identifying the places it succeeds and struggles in.

The stock market is a platform where buyers and sellers exchange stocks with one another for a monetary value (Bae, 2017). This monetary value shifts up and down based on a variety of conditions that impact the market. Some examples of these conditions are the news, public opinion, and supply/demand. To the average person and business, the stock market is a way to make money, either passively or actively. This is where trading models can be utilized, to give individuals and companies the ability to predict the stocks to a certain degree by considering various conditions that affect the market, such as the news and public opinion. This can be accomplished through the Yahoo Finance library, which can be implemented into Python code.

Trading models made without AI tools generally rely on traditional methods (looking at financial statements, earning reports, price chart patterns, economic indicators, market news, and statistical analysis) with barely any room for uniqueness because of traditions in the business space (Wu, 2023). The same financial metrics are being used by all professional traders, making it more challenging to grow as a trader without immense competition. The introduction of

ChatGPT disregards these business practices because of the amount of information at their disposal and the computational power in handling the data (Chandanshive, 2023). The amount of information traders know pales in comparison to the amount of information some AI models can store and analyze in their training data. These models can process and analyze millions of data points within a few seconds. The use of prompts also gives users the ability to easily test and experiment with new ideas they might have when developing a model. For instance, they can prompt the bot to add a specific metric when creating a model or show parts of their current model to receive suggestions on what other factors to implement and change. This gives way to fascinating combinations that would be challenging for humans to come up with themselves (Chandanshive, 2023). Due to this, AI has been proven to be useful from an educational standpoint, by seeing trends in grades to make shifts in lesson plans. However, what we aim to accomplish with this research is to determine the usefulness of LLMs in the application of a complex test case and creating predictive trading models for short-term and long-term stock options like Dr. Geoff Warren (2014) researched, except with limited human interference.

The volatile nature of stocks causes day trading to be a risky endeavor for most investors, and long-term investments are considered a safer bet. Past research with GPT-3.5 models has appeared to be successful in both long-term and short-term investments, as the researchers Mr. Glasserman and Mr. Lin found, "long-short trading strategies implemented using the original scraped and original TR headlines as input and GPT-3.5's output substantially outperform market returns over the out-of-sample period [seeing how the model works with data the model has not seen or been trained on]" (Glasserman, 2023). Within our research parameters, we will test if a trading model, generated solely by artificial intelligence, echoes this sentiment about long-term trading by comparing it with the results of short-term trading. This will be done by comparing long-term (week-to-week) and short-term trading (day-to-day) within the bounds of Open AI's Chat GPT-4's model. The accuracy of their ability to effectively read and filter unbiased historical stock data, fundamental data, technical data, and news and events (with the YahooFinance API) will be tested through the comparison of its output to the market for that selected period. Historical stock data consists of stock prices, trading volume, and other financial metrics. Fundamental data consists of revenue, earnings, and debt levels. Technical data consists of moving averages, Bollinger Bands, and Moving Average Convergence/Divergence (refer to Concepts and Equations for further information) (Mancini, 2023). Finally, news and events will be found through using the Yahoo Finance API, which will check psychological indicators, only if GPT-4 deems it is appropriate. These psychological indicators might consist of various measures such as politics and public opinion (Janková, 2019). It might succeed primarily using historical data or any other method it comes up with. Through our research, we plan to use ChatGPT to create a trading model using Python to determine whether or not large language model technology can be trusted in investing. Because Artificial Intelligence has not been widely implemented into current trading models, the research we performed can help inform the public whether or not AI models are beneficial for stock analysis, and if beneficial, whether their reliability is practical in use.

Methods

Long-term predictions will take place over seven days, not limited to trading days (Monday–Friday). Short-term predictions will take place daily, not like day trading. Day trading is where buyers purchase and sell stocks on the same trading day to capitalize on small movements or momentum. Short-term is where a prediction will be made before a trading day starts and then be compared to the actual results of the stock market at the end of the trading day (1 PM PST).

Unlike past research, we are giving the generative AI GPT-4 significantly more control and decisiveness in how it wants to formulate its predictions. GPT-4 is the most advanced model offered by Open-AI as of November 2023. Numerous other companies are working with OpenAI to integrate GPT-4 into their operations/products. Our test will truly push the limitations previously thought to be imposed on generative artificial intelligence models. To comprehensively test its stock-related prediction capabilities, we tested how well the model's stock predictions would do with daily trading and weekly trading. The short-term prediction would lead the bot to only analyze stocks from the Standard & Poor's 500 Index (commonly referred to as the S&P 500 Index). A complication we encountered constantly is updating the S&P 500 index with the most recent stocks since the index changes regularly. To have a consistent set of data, we created a CSV file with all the stocks in the index present on March 16th, 2023 (Xela, 2023). Since stock indexes are constantly being updated, a few stocks such as Signature Bank (SBNY) have been removed, and others have been added. Working with a consistent set of stocks would help the model with the following: a) limit overfitting the model, which means the model takes in too much information from newly added and removed stocks and b) being able to visualize and analyze the data more effectively because of the same stocks being analyzed each time.

With the constant changing of the S&P 500, one might wonder: What comprises this index? The answer is the most publicly traded stocks on the market. When developing the idea for this project, our research team decided to approach utilizing AI from a different perspective: maximizing its control to limit bias as much as possible. Ethically, it made sense to have the model dictate the majority of the stock analysis and prediction means and not us, because any influence would not provide authentic results (Mancini, 2023). Our goal was to find out the bigger picture of generative artificial intelligence models like GPT, and if they can create these prediction algorithms that are somewhat accurate, on their own, without any assistance.

Initially, we used the free, publicly available version of GPT-3.5 that was released in November of 2022 by OpenAI. The model had severe difficulties when producing the model and was unable to take our instructions or use our follow-up questions to help itself. The prompt we developed, after careful consideration, is depicted in Figure A1.

In Figure A1, we prompted the model with specific instructions to make a detailed stock prediction model program. There is a stock list called "500_stock_list.csv." The ".csv" refers to a comma-separated format, similar to a spreadsheet. This file has around 500 stocks that are formatted for YahooFinance searching, allowing them to be individually analyzed (Xela, 2023). The prompt in Figure A1, is answered by GPT-4, to output the code in Figure B1–G1.

The issue with this code in Figure B1–G1, however, was that there were mismatches in the number of data features. The process called *StandardScalar* expected five features but did not receive them because it was not provided with columns. To fix this issue, we asked the GPT-4 model to diagnose the problem with the code and fix itself in Figure A2, and the code was fixed in Figure B2.

After this, we combined the changes the bot made and then finally got a program (turn code into successfully executing something, regardless of the output). Next, in Figure A3–C3, we manually asked GPT to add the current prices of the stock to understand the prediction better, and to calculate the final percent change in a separate, prediction-evaluation program, detailed in a later step (Figure A10–E10). The code in Figure A3–C3 was generated by GPT-3.5, but it merely edited the print statements and format in the data, not influencing the prediction section of the code whatsoever.

After modifying the code, it fixed the error, as shown in Figure A4–D4. The model started giving predictions of actual stocks. There were still ways to go in accuracy and realness, but the output needed to be easier to read and analyze. As a result, we prompted the bot to print the current prices along with the prediction and percent change, in a neat format, to be used for data visualization later (Figure 5). This way, the user can understand the price and percent deviations from the actual market values.

After testing the program, the outputs were given, and they were highly unrealistic. Most stocks in the S&P 500 do not usually exceed 55% changes, and the predictions were given 4–5 stocks like this. If we continued like this, the model would be highly inaccurate for all the trials we performed; to limit our bias and improve the model, we asked GPT-4 how it is making its predictions to understand its process, and then how to make the prediction model more accurate (Figure A6–B6). Following this, in Figure A7–C7, GPT-4 used the steps it provided to update the code, and provided a brief explanation as to what was updated. The final part of the prediction program was created to ensure the program was thoroughly complete and working, so we asked GPT-4 to give the entire working prediction model (Open AI 2023), along with the extended functions it suggested to improve the accuracy (Figure A8–G8).

To see how the predictions performed, we created a prediction-evaluation program, to compare how the predictions from the model performed compared to the actual market. The steps detailed in Figure A9–E9 shows how this evaluation program was created and then optimized for readability and analysis. This was done in GPT-3.5, but since it was an evaluation program that drew from real-time data, it did not affect the prediction model whatsoever.

Finally, we added a less-guided weekly prediction model from GPT-4, using a similar prompt to the day-to-day predictions (Figure B11–D11). This model merely uses the Linear Regression model, detailed in section C of the Concepts and Equations Section. The proof of code generation from GPT-4 can be found in the References section (*Stock Prediction Model in Colab*).

Figure B11

Prepares the 500 stocks for data analysis.

1. Read the stock symbols from the provided CSV file.
2. Fetch historical stock data for each symbol.
3. Build and train a machine learning model on this data.
4. Predict the stock prices for the specified time frame.
5. Calculate the percent change and identify the top 10 gainers.
6. Output the results in a formatted way.

For the stock data, we can use the `yfinance` library, which allows us to fetch historical market data from Yahoo Finance. We'll use a simple machine learning model, such as a linear regressor, due to its ease of use and speed, although more complex models might yield more accurate results.

Here is the complete script:

python Copy code

```python
import pandas as pd
import yfinance as yf
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from datetime import datetime, timedelta
import numpy as np

# Function to fetch historical data for a stock symbol
def fetch_stock_data(symbol, start_date, end_date):
    stock_data = yf.download(symbol, start=start_date, end=end_date)
    return stock_data['Adj Close']

# Function to prepare data for model training
def prepare_data(stock_data):
    stock_data = stock_data.reset_index()
    stock_data['Day'] = stock_data.index
    X = stock_data[['Day']]  # Features
    y = stock_data['Adj Close']  # Target
    return train_test_split(X, y, test_size=0.2, random_state=0)
```

Figure C11

Splits the model into testing and training datasets, and uses the linear regression model mentioned in Concepts and Equations section G.

```
# Function to predict future stock price
def predict_future_price(model, future_day):
    return model.predict(np.array([[future_day]]))

# Load S&P 500 stock symbols from CSV
sp500_stocks = pd.read_csv('/content/500_stock_List.csv', header=None)
symbols = sp500_stocks[0].tolist()

# Define the time frame for data fetching and prediction
end_date = datetime.now()
start_date = end_date - timedelta(days=365)  # Last year
predict_date = end_date + timedelta(days=7)  # Predict for 7 days later

# Dictionary to store predictions
predictions = {}

# Process each stock symbol
for symbol in symbols:
    try:
        # Fetch historical data
        stock_data = fetch_stock_data(symbol, start_date, end_date)

        # Prepare data for model
        X_train, X_test, y_train, y_test = prepare_data(stock_data)

        # Create and train the model
        model = LinearRegression()
        model.fit(X_train, y_train)

        # Predict future price
        future_price = predict_future_price(model, len(stock_data) + 7)
        last_price = stock_data.iloc[-1]
        percent_change = ((future_price - last_price) / last_price) * 1

        # Store prediction
        predictions[symbol] = (future_price[0], percent_change[0])
    except Exception as e:
        print(f"Error processing {symbol}: {e}")
```

Figure D11

Prints each of the stocks in the weekly predictions.

```
# Sort predictions by percent change
top_gainers = sorted(predictions.items(), key=lambda x: x[1][1], revers

# Print top 10 predictions
print("Top 10 Stock Predictions for the Next Week:")
for symbol, (predicted_price, percent_change) in top_gainers:
    print(f"{symbol}: Predicted Price - ${predicted_price:.2f}, Predict
```

Instructions for Google Colab:

1. Copy the script into a new Google Colab notebook.
2. Upload your `500_stock_List.csv` file to the Colab environment.
3. Install the required libraries (like `yfinance`) by running `!pip install yfinance`.
4. Run the script.

Notes:

- The script uses a linear regression model for simplicity and demonstration purposes. This is not typically used for accurate stock prediction in real-world scenarios.
- Stock market predictions are inherently uncertain and should be approached with caution.
- The model does not consider external factors like market news, global events, or company-specific developments, which can significantly impact stock prices.
- The prediction is made for a week ahead, but keep in mind that stock markets are influenced by many unpredictable factors, so the accuracy of such predictions can be limited.

Concepts and Equations

The steps A–F are the equations and theory ChatGPT used when developing its prediction model.

A. Relative Strength Index (RSI):

RSI = 100 – (100/1+RS)

RS (relative strength) = (average gain of up periods during specified time/average loss of down periods during specified time)

The rate and variation of price movements are measured by the RSI as a number between 0 and 100.

B. Moving Average Convergence Divergence (MACD):

MACD = (EMA refers to the Exponential Moving Average)

Signal Line = EMA(short) – EMA(long) (EMA refers to the Exponential Moving Average)

The link between two moving averages of a stock's price is displayed by the trend-following momentum indicator known as the MACD. The 26-period EMA is subtracted from the 12-period EMA to get the MACD. The MACD line is the outcome of the equation. Plotting the "signal line," which is the nine-day EMA of the MACD, is then done on top of the MACD line.

C. Simple Moving Average (SMA):

SMA = (Sum of closing prices over 'n' periods/n)

SMA determines the average of a chosen range of closing prices by dividing it by the total number of periods in the range.

D. Random Forest Regressor:

A Random Forest Regressor operates by constructing decision trees at running time and outputting the mean prediction of the individual trees for regression tasks. Through the code ChatGPT provided we can point out places where regression tasks are occurring:

i. Predicting Stock Price Changes in the form of Delta:

- Predicting the change in stock prices, as indicated by the "Change" column in the data set, is the main objective of this regression task. The difference in closing prices between two consecutive days is used to construct this column. The Random Forest Regressor model predicts the numerical value of this change, which is a continuous variable.

ii. Independent Variables Regression:

- To calculate the stock price change there are independent variables such as the opening price of the day, highest price of the day, Relative Strength Index (RSI), Moving Average Convergence and Divergence (MACD), etc. which are calculated. The Random Forest Regressor model learns from these variables to predict the stock price change.

E. Standard Scaler:

Standard Scaler preprocesses data and conforms it to standardization by subtracting the mean and scaling it to unit variance by dividing the values by the standard deviation.

F. Grid Search Cross-Validation (GridSearchCV):

The GridSearchCV looks for the best combinations of hyperparameters for a given model (in our case the RandomForestRegressor model) by performing cross-validation. Cross-validation is needed to evaluate each combination of the parameters ensuring that the model can be trained with the correct hyperparameters. Hyperparameters are specific configurations needed to structure the model and are important to its performance.

G. Linear Regression:

Linear Regression is more statistical in nature than other training models such as the Random Forest Regressor because it assumes a linear relationship between dependent and independent variables. Linear Regression is more effective in long-term cases because it is hard to create other models such as the Random Forest Regressor without more variables being taken into account. These variables are difficult to obtain because they are always changing.

Code Explanations

Here are the two new functions GPT-4 used, as a result of implementing the steps it suggested to improve itself in Figure A6:

Figure A10

The function, "compute_rsi," is given by GPT-4 to improve the accuracy of the model.

```
# Define function to calculate RSI
def compute_rsi(data, window):
    delta = data.diff()
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)
    avg_gain = gain.rolling(window=window, min_periods=1).mean()
    avg_loss = loss.rolling(window=window, min_periods=1).mean()
    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))
    return rsi
```

Here ChatGPT is defining a function called "compute_rsi," which takes in two variables/parameters: data and window, to compute the RSI (refer to Concepts and Equations section A). In this function, we are defining a delta variable which equals the difference of comparing an element in the data with another element in the data. The gain and loss variables are dependent on the delta variable, when the delta (the change) is below 0 it defines the loss, and above 0 defines the gain. The initialization of the "avg_gain" and "avg_loss" variables needed for the RS calculation are defined with the "rolling()" and "mean()" functions. The "rolling()" function used takes in 2 parameters, window (the amount of numbers it takes in when calculating) which is defined as "window" (the function parameter GPT defined in "compute_rsi"), and "min_period" (the minimum number of observations needed to perform a calculation) which is defined as 1. The "rolling()" function then uses the "mean()" function to calculate the mean of the results. The RSI is then calculated when the RS is given in the RSI equation. The "compute_rsi" function ultimately stores the RSI.

Figure B10

The "computer_macd" function given by GPT-4 to improve the accuracy of the model.

```
# Define function to calculate MACD
def compute_macd(data, short_window, long_window):
    ema12 = data.ewm(span=short_window, adjust=False).mean()
    ema26 = data.ewm(span=long_window, adjust=False).mean()
    macd = ema12 - ema26
    signal = macd.ewm(span=9, adjust=False).mean()
    return macd - signal
```

In this code snippet, ChatGPT is defining a function called "compute_macd," which takes in three variables/parameters: data, "short_window" and "long_window," to compute the MACD (refer to Concepts and Equations section B). The ema12 variable in the function is defined with the "ewm()" function (Exponentially Weighted Moving) and the "mean()" function to calculate the EMA (Exponential Moving Average) or EWMA (Exponentially Weighted Moving Average). These terms, EMA and EWMA, are often used interchangeably. The "ewm()" function takes in two parameters, span (a higher span number means that older observations keep their influence for a longer period) which is defined as "short_window" (the function parameter GPT-4 defined in "compute_macd"), and adjust (True means the weighted averages are calculated with the assumption that weights are applied to a fully observed time series and False means that the calculation is done cumulatively without any assumption) which is defined as False. The "ema26" variable is defined similarly but instead of the parameter span being "short_window" it is “long_window” (another function parameter GPT-4 defined in compute_macd) instead. The MACD equation then calculates the MACD variable. After that, the signal variable is calculated by the Signal Line logic (in Concepts and Equations section B), and finally, stored in the “compute_macd” function is the $MACD\ variable\ -\ signal\ variable$.

The figure below depicts the section of the model that uses training and testing features to predict stocks.

Figure 11

```
    for symbol in stock_symbols:
        try:
            # Fetch historical data from Yahoo Finance
            data = yf.download(symbol, period="180d", interval="1d")

            # Prepare the data for the model
            data['Change'] = data['Close'].diff()
            data['SMA_10'] = data['Close'].rolling(window=10).mean()
            data['SMA_50'] = data['Close'].rolling(window=50).mean()
            data['RSI'] = compute_rsi(data['Close'], window=14)
            data['MACD'] = compute_macd(data['Close'], 12, 26)
            data.dropna(inplace=True)

            X = data[['Open', 'High', 'Low', 'Volume', 'SMA_10', 'SMA_50', 'RSI', 'MA
            y = data['Change']

            # Split the dataset into the train set and the test set
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

            # Initialize and fit the StandardScaler
            scaler = StandardScaler()
            X_train_scaled = scaler.fit_transform(X_train)

            # Define the parameter grid for hyperparameter tuning
            param_grid = {
                'n_estimators': [50, 100, 150],
                'max_depth': [10, 20, 30]
            }

            # Initialize and fit the RandomForest model
            model = RandomForestRegressor(random_state=0)
            grid_search = GridSearchCV(model, param_grid, cv=5)
            grid_search.fit(X_train_scaled, y_train)
            best_model = grid_search.best_estimator_

            # Predict the price change for the next trading day
            latest_features = X.iloc[-1:].values
            latest_features_scaled = scaler.transform(latest_features)
            predicted_change = best_model.predict(latest_features_scaled)[0]
            current_price = data['Close'].iloc[-1]
            predicted_price = current_price + predicted_change
            predicted_change_percent = (predicted_change / current_price) * 100
```

There are six steps to this model: 1) Data collection, 2) Feature engineering, 3) Data splitting, 4) Compatibility, 5) RandomForestRegressor model and GridSearchCV, 6) Predictions.

1. Data collection: The data must be collected somehow. In this case, we utilized the “yfinance” library, which stands for Yahoo Finance. Yahoo Finance has the historical data of all the stocks, which is key for calculating technical metrics and analyzing advanced data (Janková 2019). This finance library takes the past 180 days of stock data.
2. Feature Engineering: There are five important feature engineering functions present: “Change,” “SMA_10,” “SMA_15,” “RSI,” and “MACD.” Change measures the difference between a stock’s opening price (price at 6:30 AM PST on a trading day) and closing price (price at 1:00 PM on a trading day). The SMA functions to find the moving averages over 10 and 50 days respectively. Moving averages are simply averages over a specific amount of time. The RSI measures how fast price movements occur: the MACD also does the same, but it finds convergence and divergences in the price charts.
3. Data splitting: In machine learning models, 80% of the data is used to train the model, and 20% is used to test the model, which is what the accuracy represents. This is standard across most machine learning disciplines.
4. Compatibility: Due to the errors encountered earlier with the StandardScalar, the regression model has to take a certain amount of features (in this case five), in order to standardize it with the linear regression model which utilizes the StandardScalar’s data as an input.
5. RandomForestRegressor model and GridSearchCV: Tuning the RandomForestRegressor model is finding out what combinations work best for the training data, by using decision trees, to make itself function at a high standard instead of relying on randomness (). The

RandomForest model uses the training data, past stock data, and the individual variable functions to learn and calculate data through regression tasks. GridSearchCV takes in the parameter grid and model and then trains the model by cross-validating multiple combinations of the parameters. While doing this GridSearchCV also checks the model's performance to make sure it yields the best results.

6. Predictions: In the last section of the code, the latest features are selected and scaled so it can be fed into the training model function ("best_model.predict()") as a parameter and saved in the variable "predicted_change" as a prediction. The current price of the stock is then found at the closing time and is defined as "current_price" which is added to the "predicted_change" to define the "predicted_price variable." At the end, we can finally get the predicted change percent in a "predicted_change_percent" variable by setting it equal to the $(predicted\_change/current\_price) * 100$.

Results and Discussion

Initially, we tried using the GPT-3.5 model to create our prediction model. Even though GPT-3.5 is significantly faster at generating responses, it did not have the updated training data and added capabilities the Open AI team added to the GPT-4 model. When we tried using the same prompt detailed in Figure A1 for the GPT-3.5 version:

1. The bot refused to generate the prediction model and instead gave functions with comments that read "to be implemented."
2. The bot created a prediction model but had compilation errors. When asked to fix its errors either logical errors appeared, or more compilation errors occurred.
3. The bot did not provide any pseudocode (a simplified programming outline), and instead gave written instructions on how to go about building a stock prediction model.

We purchased the monthly GPT-4 subscription advertised by OpenAI because of the limitations and difficulties of using GPT-3.5. As of November 2023, the GPT-4 plan can be purchased for $20 a month, which is more accurate and has more features compared to GPT-3.5.

Disclaimer

All the conversations and outputs from ChatGPT can be verified by visiting the References section (*Final* and *Stock Prediction Model in Colab*). We turned on the Advanced-data analysis feature for GPT-4 from the drop-down section; however, we would like to disclaim that GPT-4 did not create the entirety of the prediction model. In the middle of the research, the model resorted back to GPT-3.5 when fixing the timing, though that was solely to fix the time issue. In the appendix, where the prompts are listed, the statements that end with "The previous model used in this conversation is unavailable. We've switched you to the latest default model" are all created by GPT-3.5; all the other statements (including the prediction model and feature engineering) were by the GPT-4 Advanced Data Analysis feature. GPT-3.5 was mainly to do with fixing the timing of the predictions and had nothing to do with fixing historical data. It did not influence the calculations of the predictions. The above statements can be verified by clicking the link to the chat log. We have decided to use all these images to not omit information from the reader, so they can understand the full process we went through.

Visualization

In total, there were 12 day-to-day predictions made, and 1 weekly prediction made. A day-to-day prediction involved running the stock at 9 PM PST, the day before an open trading day. All of the daily predictions with their comparison can be viewed in the Appendix, in Table A1–Table A12.

To gain an understanding of how the model performed over time, we created a line graph with a trendline using linear regression. The metric we used to show the accuracy of the days over time is the Mean Absolute Error (MAE), which is the average difference between the predicted values and the actual values in the data tables. Since there are 10 predictions made per trading day, the subtraction involving the difference in the actual percent change and the predicted percent change was performed for each stock's comparison and then averaged to form the MAE.

The lower the MAE is, the more accurate the predictions are. Individual MAE points are labeled over each one, though one concept is clear: the MAE reduces over time in this case. This is a line graph with a trendline of the 12 day-to-day stock predictions:

Figure 12

Date vs. Mean Absolute Error for day-to-day predictions

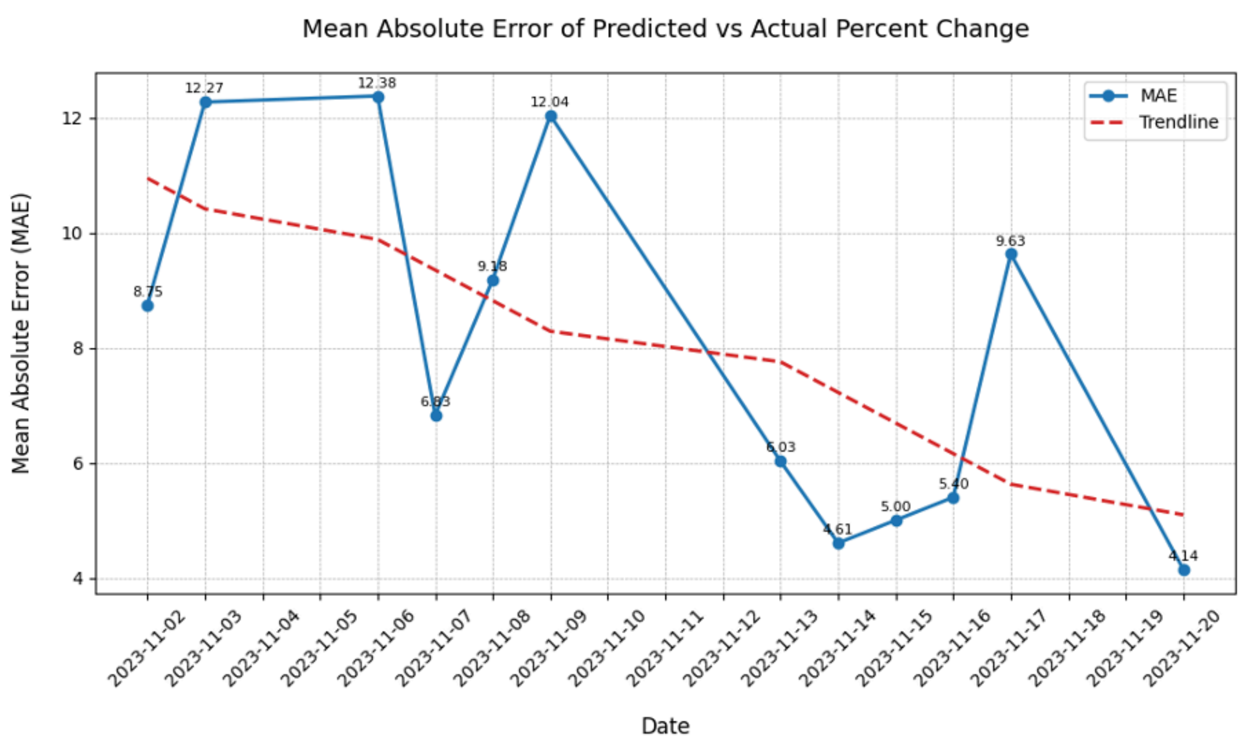
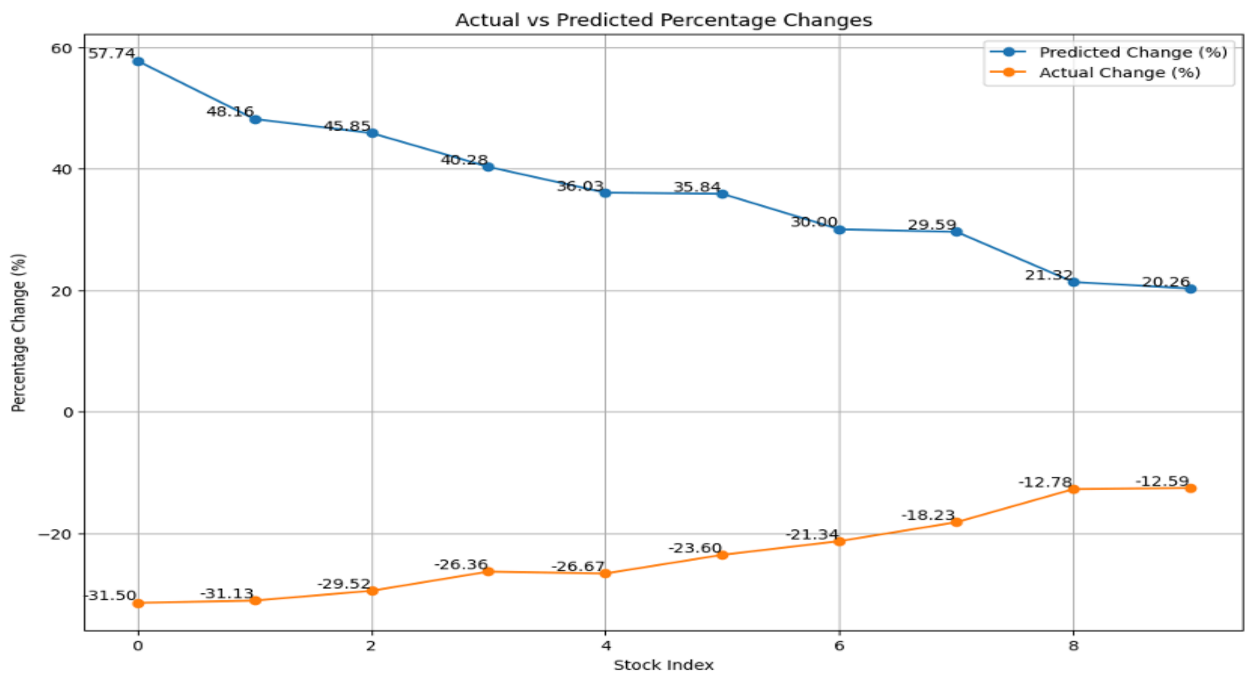


Figure 13

Actual Percent Changes vs. Predicted Percent Changes for weekly predictions



One interesting point to note is the predictions appeared widely inaccurate in the first few days; however, the last 6 predictions were very accurate with the exception of one of the days. Because the model was given the same stocks over and over, it was able to cut down on compiling time. The increase in accuracy might be explained by the increase in data for a stock over time. The more data available, the more accurate the predictions became, and that can be partially attributed to the technical metric functions used in Figure A10, Figure B10, and Figure 11. In Figure 13, the model was highly inaccurate. This was because the stock predictions over the long term are difficult to calculate since much can change with stock over a week compared to a day. There is more potential for stock breakouts, and breaking news, such as earning reports that drastically change a stock's value. This suggests that the daily prediction model was handled significantly better than the longer-term predictions. There were possibly not enough trials run on the weekly prediction, and perhaps more functions instead could have been used to assist the linear regression model; however, we expected the model to be somewhat accurate with a few positive predictions with little assistance because long-term predictions are safer for investors and net more positive returns. It was quite surprising to see all the predictions do so poorly, that not even one of the predicted ones did well.

The last point about this section involves the legality of implementing AI. Can artificial intelligence-generated code be defined as intellectual property? Currently, there are no concrete laws defining this because AI is a new space. With plagiarism detectors growing, it is easier to detect AI-generated content that copies what has been said elsewhere. Algorithmic plagiarism with the stock prediction programs can be much different, however. Since the algorithms are intricate and use specific code, it is easy to define them as intellectual property and copyright. These boundaries are undefined in some regions and still have ways to go to develop legally. Using artificial intelligence chatbots like GPT can benefit society if used ethically and with respect to each other's work.

Conclusion

When developing the process for this research problem, we considered the rise of Artificial Intelligence in society as of 2023. Large-language models like ChatGPT, one of the first large-language models accessible to the public, are one of the reasons for this AI boom. Artificial Intelligence and chatbots are useful for automated tasks like customer support and data entry; and on a much larger scale, we tested if Artificial Intelligence can be accurate with limited human intervention and guidance to predict stocks.

From our results, it is evident that the prediction models have a long way to go on their own to give accurate predictions. Perhaps the data it is trained on could be expanded, which spells the trade-off of efficiency versus accuracy. If the prediction model takes several hours, and cannot compute before a trading day opens, it becomes less useful. This issue could potentially be resolved with a more powerful system.

Overall, day-to-day trading led to higher yields in results than long-term trading. One interesting thing to note is the improvement of the model over time; the predictions became more accurate as the predicted change percent deviated from the actual percent changes. Our research aims to help further innovation in prompting and stock prediction with AI at this moment by promoting the integration of functions with AI to save time, manual calculation, and human resources to perform these processes more effectively.

References

Bae, K., and Kang, J. (2017). Does the Stock Market Benefit the Economy? *European Financial Management Association*. https://www.efmaefm.org/0EFMSYMPOSIUM/2017/papers/Does%20the%20Stock%20Market%20Benefit%20the%20Economy%20-%20updated.pdf

Chandanshive, G., and Ansurkar, G. (2023). A Comparative Study of Artificial Intelligence (AI) Techniques for Stock Market Prediction, *International Journal of Scientific Research in Science and Technology*, 842–851. https://doi.org//10.32628/IJSRST523102122

Glasserman, P., and Lin, C.(2023). Assessing Look-Ahead Bias in Stock Return Predictions Generated By GPT Sentiment Analysis. https://doi.org/10.48550/arxiv.2309.17322

Janková, Z., and Dostál, P. (2019). Utilization of Artificial Intelligence for Sensitivity Analysis in the Stock Market, *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis*, *67*(5), 1269–1283. https://doi.org/10.11118/actaun201967051269

Lim, C., Bourke, L. Fouracre, K., Pallaras, L., and Barbaro, A. (2023, April 14). *ChatGPT and the Importance of AI Governance*. Lexology. http://www.lexology.com/library/detail.aspx?g=014f36ad-dcd8-4135-a045-1986a963f241

Mancini, J. (2023, April 25). *ChatGPT can predict stock market moves? New study shows ChatGPT surprisingly accurate in predicting stock moves and could even replace investment analysts.* Yahoo!Finance. https://finance.yahoo.com/news/chatgpt-predict-stock-market-moves-185307674.html

Mokhtari, S., Yen, K. K., and Liu, J. (2021). "Effectiveness of Artificial Intelligence in Stock Market Prediction based on Machine Learning," *International Journal of Computer Applications, 183*(7), 1–8. https://doi.org/10.5120/ijca2021921347

ChatGPT. (2022). ChatGPT. https://chat.openai.com/

OpenAI. (2023). *GPT-4 Technical Report.* OpenAI. https://cdn.openai.com/papers/gpt-4.pdf

OpenAI. (2023). *Final*. https://bit.ly/49WdmWq

OpenAI. (2023). *Stock Prediction Model in Colab*. https://bit.ly/3SWhn75

Warren, Geoffrey J., Long-Term Investing: What Determines Investment Horizon? (October 21, 2014). CIFR Paper No. 39, Available at SSRN: https://ssrn.com/abstract=2513088 or http://dx.doi.org/10.2139/ssrn.2513088

Wu, B. (2023). Is GPT4 a Good Trader? *arXiv*. https://doi.org/10.48550/arxiv.2309.10982

Xela, A. (2023, March 16). *S&P 500 Companies*. Kaggle. https://www.kaggle.com/datasets/alexanderxela/sp-500-companies

Appendix A

Prompts and Results

Figure A1

Initial prompt to create a prediction model in Python.

You

Create a machine learning model using any programming language that can be run in google colab. The program should take all of the individual stocks in the S&P 500, which are listed is in the first column of this csv path: /content/500_stock_List.csv
Keep in mind that the stocks are just listed and there is NO column heading with the word "symbol"
Predict the top 10 stock gainers for the following trading day. For example, if the program runs after the trading day starts on Monday, predict the stock prices and percent changes for the end of the day on Tuesday. Keep in mind that if the program is run after Friday's trading time starts, you should predict the stock prices for the end of the day on Monday. Trading days start at 6:30 AM PST and end at 1:00 PM PST.
Print out the top 10 stocks' predictions in a neat format, along with their predicted percent changes. Do not assume any code will be finished or written later. Make sure the program is as accurate as possible. Only respond to me with the coding programs I can run that accomplish these tasks, do not give me an outline or unusable code.

Figure B1

GPT-4 explaining its thought process.

ChatGPT

Creating a machine learning model to predict the stock market accurately is quite complex and generally not reliable due to the random nature of the markets. Stock prices are influenced by countless factors that can't be accurately captured by historical price data alone. However, for the sake of demonstration, I will create a simple model using Python, TensorFlow, and Keras which you can run in Google Colab.

Keep in mind that this is just a simplified example for educational purposes, and it's not intended to be used as financial advice or a real trading strategy.

Here's a script that you can use:

Figure C1

Setting the trading day to a day of the week.

python Copy code

```python
# Install the yfinance package which will be used to fetch historical stock data
!pip install yfinance

import yfinance as yf
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
import datetime
import os

# Check if today is a trading day and adjust the prediction date accordingly
def get_next_trading_day(today):
    if today.weekday() == 4: # Friday
        next_trading_day = today + datetime.timedelta(days=3) # Monday
    elif today.weekday() == 5: # Saturday
        next_trading_day = today + datetime.timedelta(days=2) # Monday
    else:
        next_trading_day = today + datetime.timedelta(days=1)
    return next_trading_day
```

Figure D1

Downloading the stocks; using feature analysis for predictions; storing the predictions.

```
# Predict the Adjusted Close price for the next trading day
def predict_stock(stock_symbol, model, scaler, days=30):
    # Fetch historical data from Yahoo Finance
    data = yf.download(stock_symbol, period=f"{days}d", interval="1d")
    if data.empty:
        return None, None
    # We'll use 'Close' price for prediction
    X = data['Close'].values.reshape(-1, 1)
    # Scale the features
    X_scaled = scaler.transform(X)
    # Predict
    predicted_price = model.predict(X_scaled[-1].reshape(1, -1))
    predicted_change_percent = (predicted_price - X[-1]) / X[-1] * 100
    return predicted_price[0], predicted_change_percent[0]

# Read the list of S&P 500 stock symbols
df_stocks = pd.read_csv('/content/500_stock_List.csv', header=None)
stock_symbols = df_stocks.iloc[:, 0].values

# Initialize lists to store predictions
predictions = []
change_percents = []

# Today's date
today = datetime.datetime.now()

# Get the next trading day
next_trading_day = get_next_trading_day(today)

# Use RandomForest as the prediction model
model = RandomForestRegressor()
```

Figure E1

Splitting into training and testing data for further analysis.

```
# Feature scaling
scaler = StandardScaler()

# Loop over each stock symbol and make a prediction
for symbol in stock_symbols:
    try:
        # Get historical stock data
        stock_data = yf.download(symbol, period="60d", interval="1d")

        # Prepare the data for RandomForest
        stock_data['Change'] = stock_data['Close'].diff()
        stock_data.dropna(inplace=True)
        X = stock_data.drop(['Close', 'Change'], axis=1)
        y = stock_data['Change']

        # Split the dataset into the train set and the test set
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ranc

        # Scale the features
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Fit the RandomForest model
        model.fit(X_train_scaled, y_train)

        # Predict the price change for the next trading day
        predicted_price, predicted_change_percent = predict_stock(symbol, model, scal
```

Figure F1

Error checking and print statements

```
        # Store the results
        if predicted_price and predicted_change_percent:
            predictions.append((symbol, predicted_price, predicted_change_percent))

    except Exception as e:
        print(f"An error occurred for {symbol}: {e}")

# Sort the predictions by the percent change
predictions.sort(key=lambda x: x[2], reverse=True)

# Print the top 10 predictions
print(f"Top 10 Stock Predictions for {next_trading_day.strftime('%Y-%m-%d')}")
for i, prediction in enumerate(predictions[:10]):
    print(f"{i+1}. {prediction[0]}: Predicted Price - ${prediction[1]:.2f}, Predicted
```

Figure G1

GPT-4 program notes and disclaimers

Please note the following points regarding this script:

1. The predictions made by this script are based on historical price changes and don't take into account other critical market factors.
2. The model uses the Random Forest algorithm which is not necessarily the best algorithm for time-series predictions like stock prices.
3. The script does not include any feature engineering which is crucial for improving the model's predictions.
4. Real stock trading algorithms use much more sophisticated methods and data sources to predict stock prices.
5. It's recommended to have a good understanding of financial markets and machine learning before attempting to trade based on algorithmic predictions.

Make sure you have the file `/content/500_stock_List.csv` with the correct format available in your Google Colab environment before running this script.

Figure A2

Sending a compiler error that occurs back into GPT-4 for fixation.

You

*********100%%**********************] 1 of 1 completed
An error occurred for DVN: X has 1 features, but StandardScaler is expecting 5 features as input.
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
 warnings.warn(
[*********************100%%**********************] 1 of 1 completed
[*********************100%%**********************] 1 of 1 completed
An error occurred for DXC: X has 1 features, but StandardScaler is expecting 5 features as input.
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
 warnings.warn(
[*********************100%%**********************] 1 of 1 completed
[*********************100%%**********************] 1 of 1 completed
An error occurred for DXCM: X has 1 features, but StandardScaler is expecting 5 features as input.
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
 warnings.warn(

ChatGPT

The error you are encountering suggests that there's a mismatch in the number of features between the training phase and the prediction phase. The `StandardScaler` is expecting 5 features, which means during training it was fit on data with 5 features, but during prediction, only 1 feature is being provided.

The error arises because when predicting, we are only using the 'Close' prices (`X = data['Close'].values.reshape(-1, 1)`), which is a single feature, whereas the model was trained with more features. To fix this, we need to ensure that the prediction phase uses the same features as the training phase.

Figure B2

GPT-4 fixes the error

Here is the corrected version of the prediction part of the script:

python Copy code

```python
# Predict the Adjusted Close price for the next trading day
def predict_stock(stock_symbol, model, scaler, days=60):
    # Fetch historical data from Yahoo Finance
    data = yf.download(stock_symbol, period=f"{days}d", interval="1d")
    if data.empty:
        return None, None

    # We use the same features as used in training
    data['Change'] = data['Close'].diff()
    features = data.drop(['Close', 'Change'], axis=1)
    latest_features = features.iloc[-1].values.reshape(1, -1)

    # Scale the features using the scaler fit on the training data
    latest_features_scaled = scaler.transform(latest_features)

    # Predict
    predicted_change = model.predict(latest_features_scaled)
    predicted_price = data['Close'].iloc[-1] + predicted_change[0]
    predicted_change_percent = (predicted_change / data['Close'].iloc[-1]) * 100

    return predicted_price, predicted_change_percent[0]
```

Make sure to adjust this function in the main loop where you are calling the `predict_stock` function.

Also, ensure that you save and reuse the scaler that was fitted during training. Each stock symbol will have its own scaler, so you may need to save each scaler and retrieve the correct one for the corresponding stock when making predictions. The script provided does not do this and would need to be modified to accommodate saving and retrieving scalers for each stock.

Lastly, this is still a very simplistic approach to stock prediction and is likely to not perform well. Real-world stock prediction is much more complex and often not reliably accurate. It's highly recommended to seek proper financial advice and perform rigorous backtesting before using any kind of automated system for trading.

Figure A3

Adding categories to the table for prediction evaluation later.

Anonymous

good. Now when the program is run get the current stock prices and also print the predicted percent changes, not just the predicted price. print it in a neat stylistic format

The previous model used in this conversation is unavailable. We've switched you to the latest default model.

ChatGPT

Understood. To achieve this, we will modify the function to fetch the current stock prices for the day and calculate the predicted percent changes relative to those prices. We will also enhance the output formatting to make it more readable and well-presented.

Below is the complete revised code:

python Copy code

```python
!pip install yfinance

import yfinance as yf
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
import datetime

# Define constants for the trading hours
TRADING_START_HOUR = 6  # 6:30 AM PST
TRADING_END_HOUR = 13   # 1:00 PM PST

# Function to check if the current time is after the trading day's end
def is_after_trading_hours():
    now = datetime.datetime.now()
    trading_day_end = datetime.time(hour=TRADING_END_HOUR, minute=0)
    return now.time() > trading_day_end

# Function to get the next trading day's date
def get_next_trading_day():
    today = datetime.datetime.now()
    # If it's Friday after trading hours, or it's Saturday or Sunday, skip to Monday
    if (today.weekday() == 4 and is_after_trading_hours()) or today.weekday() > 4:
        next_trading_day = today + datetime.timedelta(days=(7 - today.weekday()))
    else:
        next_trading_day = today if not is_after_trading_hours() else today + datetim
    return next_trading_day.date()
```

Figure B3

Prints entire working prediction program.

```
# Predict the stock prices and percent changes for the next trading day
def predict_stock_prices(stock_symbols):
    # Initialize lists to store predictions
    predictions = []

    for symbol in stock_symbols:
        try:
            # Fetch historical data from Yahoo Finance
            data = yf.download(symbol, period="60d", interval="1d")

            # Prepare the data for the model
            data['Change'] = data['Close'].diff()
            data.dropna(inplace=True)
            X = data.drop(['Close', 'Change'], axis=1)
            y = data['Change']

            # Split the dataset into the train set and the test set
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

            # Initialize and fit the StandardScaler
            scaler = StandardScaler()
            X_train_scaled = scaler.fit_transform(X_train)

            # Initialize and fit the RandomForest model
            model = RandomForestRegressor(random_state=0)
            model.fit(X_train_scaled, y_train)

            # Predict the price change for the next trading day
            latest_features_scaled = scaler.transform(X.iloc[-1:].values)
            predicted_change = model.predict(latest_features_scaled)[0]
            current_price = data['Close'].iloc[-1]
            predicted_price = current_price + predicted_change
            predicted_change_percent = (predicted_change / current_price) * 100

            # Store the results
            predictions.append({
                "symbol": symbol,
                "current_price": current_price,
                "predicted_price": predicted_price,
                "predicted_change_percent": predicted_change_percent
            })

        except Exception as e:
            print(f"An error occurred for {symbol}: {e}")
```

Figure C3

Prints the entire working prediction program.

```
    # Sort the predictions by the predicted percent change
    predictions.sort(key=lambda x: x["predicted_change_percent"], reverse=True)

    return predictions[:10]  # Return only the top 10 predictions

# Read the list of S&P 500 stock symbols from a CSV file
df_stocks = pd.read_csv('/content/500_stock_List.csv', header=None)
stock_symbols = df_stocks.iloc[:, 0].values

# Get the next trading day
next_trading_day = get_next_trading_day()

# Predict stock prices and percent changes for the next trading day
predictions = predict_stock_prices(stock_symbols)

# Neatly print the predictions
print(f"Top 10 Stock Predictions for {next_trading_day.strftime('%Y-%m-%d')} (End of
print("---------------------------------------------------------")
print(f"{'Symbol':<10} | {'Current Price':>15} | {'Predicted Price':>15} | {'Change (
print("---------------------------------------------------------")
for prediction in predictions:
    print(f"{prediction['symbol']:<10} | "
          f"${prediction['current_price']:>14.2f} | "
          f"${prediction['predicted_price']:>14.2f} | "
          f"{prediction['predicted_change_percent']:>10.2f}%")
print("---------------------------------------------------------")
```

This code will:

1. Check if the current time is after trading hours.
2. Adjust the prediction day to the next trading day accordingly.
3. Fetch the last close price of the stock.
4. Predict the price change and the next day's price.
5. Calculate the predicted percent change

Figure A4

Logic error occurs with the trading times; GPT-4 attempts to fix this issue.

You

I believe something is off with the timing. When I ran the program at 9 PM PST on November 1st, it gave me the predictions for november 3rd end of the day. I want predictions for november 2nd end of the trading day.

essentially, If i run the program after the trading day ends on monday, it should give predictions for end of the trading day (1 PM PST) on Tuesday.

ChatGPT

I see what you mean. To predict the stock prices for the end of the trading day on Tuesday when running the program after Monday's trading day ends, you should consider a different approach to calculate the prediction date. You can use the current time and adjust it to the next trading day if needed. Here's the modified code:

Figure B4

Fixed time error in the code.

python Copy code

```python
# Import the necessary libraries
import yfinance as yf
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
import datetime

# Function to check if the current time is after the trading day's end (1:00 PM PST)
def is_after_trading_day_end():
    now = datetime.datetime.now()
    trading_day_end = datetime.time(13, 0)  # 1:00 PM PST
    return now.time() >= trading_day_end

# Function to get the next trading day's date
def get_next_trading_day():
    today = datetime.date.today()
    if today.weekday() == 4:  # If today is Friday, move to Monday
        next_trading_day = today + datetime.timedelta(days=3)
    else:
        next_trading_day = today + datetime.timedelta(days=1)
    return next_trading_day

# Predict the stock prices for the next trading day
def predict_stock_prices(stock_symbols):
    # Initialize lists to store predictions
    predictions = []
```

Figure C4

Prints the rest of the program.

```
for symbol in stock_symbols:
    try:
        # Fetch historical data from Yahoo Finance
        data = yf.download(symbol, period="60d", interval="1d")

        # Prepare the data for the model
        data['Change'] = data['Close'].diff()
        data.dropna(inplace=True)
        X = data.drop(['Close', 'Change'], axis=1)
        y = data['Change']

        # Split the dataset into the train set and the test set
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

        # Initialize the StandardScaler
        scaler = StandardScaler()

        # Scale the features
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Initialize the RandomForest model
        model = RandomForestRegressor()

        # Fit the model
        model.fit(X_train_scaled, y_train)

        # Predict the price change for the next trading day
        latest_features = X_test_scaled[-1].reshape(1, -1)
        predicted_change = model.predict(latest_features)
        predicted_price = data['Close'].iloc[-1] + predicted_change[0]
```

Figure D4

Prints the rest of the program.

```
    # Sort the predictions by predicted price
    predictions.sort(key=lambda x: x[1], reverse=True)

    return predictions

# Read the list of S&P 500 stock symbols from a CSV file
df_stocks = pd.read_csv('/content/500_stock_List.csv', header=None)
stock_symbols = df_stocks.iloc[:, 0].values

# Check if it's after the trading day's end, and adjust the prediction date if needed
if is_after_trading_day_end():
    next_trading_day = get_next_trading_day()
else:
    next_trading_day = datetime.date.today()

# Predict stock prices for the next trading day
predictions = predict_stock_prices(stock_symbols)

# Print the top 10 predictions
print(f"Top 10 Stock Predictions for {next_trading_day.strftime('%Y-%m-%d')}")
for i, prediction in enumerate(predictions[:10]):
    print(f"{i+1}. {prediction[0]}: Predicted Price - ${prediction[1]:.2f}")
```

This code will give you predictions for the end of the trading day on November 2nd when you run it after the trading day ends on November 1st. It checks the current time to determine if it's after the trading day's end and adjusts the prediction date accordingly.

Figure 5

More format and neatly formatted print statement.

```
# Neatly print the predictions
print(f"Top 10 Stock Predictions for {next_trading_day.strftime('%Y-%m-%d')} (End of
print("----------------------------------------------------------")
print(f"{'Symbol':<10} | {'Current Price':>15} | {'Predicted Price':>15} | {'Change (
print("----------------------------------------------------------")
for prediction in predictions:
    print(f"{prediction['symbol']:<10} | "
          f"${prediction['current_price']:>14.2f} | "
          f"${prediction['predicted_price']:>14.2f} | "
          f"{prediction['predicted_change_percent']:>10.2f}%")
print("----------------------------------------------------------")
```

This code will:

1. Check if the current time is after trading hours.
2. Adjust the prediction day to the next trading day accordingly.
3. Fetch the last close price of the stock.
4. Predict the price change and the next day's price.
5. Calculate the predicted percent change

Figure A6

GPT-4 suggests ways to improve the accuracy of its predictions.

You

Good. Now tell me in words what else can be done to make the initial prediction model more accurate.

ChatGPT

To make the initial stock price prediction model more accurate, you can consider the following strategies and techniques:

1. **More Feature Engineering:**
   - Explore additional features like technical indicators (e.g., moving averages, RSI, MACD), fundamental data (e.g., P/E ratio, market capitalization), and news sentiment analysis to incorporate more information into the model.
2. **Longer Historical Data:**
   - Use a longer historical dataset, beyond the 60 days, to capture more long-term trends and patterns that can affect stock prices.
3. **Better Feature Selection:**
   - Identify the most relevant features for prediction and remove noise from the data. Use techniques like feature importance analysis to select the most informative features.
4. **Hyperparameter Tuning:**
   - Fine-tune the hyperparameters of the machine learning model. For the Random Forest Regressor, you can adjust parameters like the number of trees, depth of trees, and the minimum number of samples required to split a node.
5. **Cross-Validation:**
   - Implement cross-validation techniques like k-fold cross-validation to assess the model's performance and identify potential overfitting.
6. **Ensemble Methods:**
   - Consider ensemble methods like stacking or blending multiple machine learning models to combine their predictions for improved accuracy.
7. **Feature Scaling and Normalization:**
   - Experiment with different scaling and normalization techniques to preprocess the data. Ensure that the chosen method is appropriate for the features and the model.

Figure B6

GPT-4 suggests ways to improve the accuracy of its predictions.

8. **Handling Imbalanced Data:**
   - Address any class imbalance issues in the dataset. If the dataset has significantly more examples of one class, it can lead to biased predictions.
9. **News and Event Data:**
   - Integrate real-time news and event data that can influence stock prices. This can help the model react to breaking news and events in real-time.
10. **Sentiment Analysis:**
    - Incorporate sentiment analysis of financial news and social media data to gauge market sentiment, as market sentiment can strongly impact stock prices.
11. **Data Preprocessing:**
    - Clean and preprocess the data more thoroughly, handling missing values, outliers, and anomalies in a way that best suits the dataset.
12. **Time Series Analysis:**
    - Consider using time series analysis techniques, such as autoregressive integrated moving average (ARIMA) or Prophet, for modeling and predicting time-dependent stock price patterns.
13. **Additional Data Sources:**
    - Explore other data sources, such as economic indicators, sector-specific data, or global market data, which can provide valuable insights into the stock market.
14. **Advanced Models:**
    - Experiment with more advanced machine learning models, such as recurrent neural networks (RNNs) and long short-term memory networks (LSTMs), which are well-suited for sequential data like time series.
15. **Risk Management:**
    - Implement risk management strategies, such as stop-loss orders and position sizing, to mitigate losses when trading based on predictions.
16. **Continuous Model Updating:**
    - Regularly retrain and update the model with new data to adapt to changing market conditions.
17. **Backtesting:**
    - Conduct extensive backtesting to evaluate the model's historical performance and identify areas for improvement.
18. **Expert Advice:**
    - Seek guidance from financial experts or data scientists with domain knowledge in finance to refine the model.

Figure A7

GPT uses its own words to make the model more accurate, displaying our limitation of human intervention. Here it adds the RSI function to improve accuracy.

You

Using what you have just told me, make the initial prediction model more accurate

ChatGPT

Improving the accuracy of a stock price prediction model is a complex task that requires a combination of strategies and fine-tuning. While I can provide general guidance, it's important to understand that stock price prediction remains a challenging and uncertain endeavor. Here's an extended version of the initial prediction model with some enhancements:

python Copy code

```python
# Import additional libraries
from datetime import timedelta
from pandas_datareader import data as pdr
import fix_yahoo_finance as yf

# Extend historical data to capture more long-term trends
data = yf.download(symbol, period="180d", interval="1d")

# Feature Engineering
data['SMA_10'] = data['Close'].rolling(window=10).mean()  # 10-day Simple Moving Aver
data['SMA_50'] = data['Close'].rolling(window=50).mean()  # 50-day Simple Moving Aver
data['RSI'] = compute_rsi(data['Close'], window=14)  # Relative Strength Index (RSI)
data['MACD'] = compute_macd(data['Close'], 12, 26)  # Moving Average Convergence Dive

# Define function to calculate RSI
def compute_rsi(data, window):
    delta = data.diff()
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)
    avg_gain = gain.rolling(window=window, min_periods=1).mean()
    avg_loss = loss.rolling(window=window, min_periods=1).mean()
    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))
    return rsi
```

Figure B7

GPT-4 adds the MACD function and implements cross-validation to improve accuracy.

```
# Define function to calculate MACD
def compute_macd(data, short_window, long_window):
    ema12 = data.ewm(span=short_window, adjust=False).mean()
    ema26 = data.ewm(span=long_window, adjust=False).mean()
    macd = ema12 - ema26
    signal = macd.ewm(span=9, adjust=False).mean()
    return macd - signal

# Feature Selection
X = data[['Open', 'High', 'Low', 'Close', 'Volume', 'SMA_10', 'SMA_50', 'RSI', 'MACD'
y = data['Change']

# Tune Hyperparameters (use GridSearchCV or RandomizedSearchCV)
# Example: optimize the number of trees and maximum depth
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [10, 20, 30]
}

# Implement Cross-Validation
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

model = RandomForestRegressor(random_state=0)
grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X, y)
best_model = grid_search.best_estimator_

# Fetch and predict current stock data
current_data = pdr.get_data_yahoo(symbol, end=datetime.now())
current_features = current_data.iloc[-1][['Open', 'High', 'Low', 'Close', 'Volume', '
current_features = current_features.values.reshape(1, -1)
predicted_change = best_model.predict(current_features)
current_price = current_data['Close'].iloc[-1]
predicted_price = current_price + predicted_change[0]

# Print the comparison
print(f"Comparison of Predicted and Actual Stock Prices (End of Day)")
print("--------------------------------------------------------------")
print(f"{'Symbol':<8} | {'Predicted Price':>17} | {'Actual Price':>14} | {'Change (%
print("--------------------------------------------------------------")
```

Figure C7

GPT-4 explains what it changed to improve accuracy, in five steps.

```
if current_price is not None:
    price_change = (current_price - predicted_price) / predicted_price * 100
    print(f"{symbol:<8} | ${predicted_price:.4f} | ${current_price:.4f} | {price_chan
else:
    print(f"{symbol:<8} | ${predicted_price:.4f} | {'N/A':>12} | {'N/A':>16}")

print("------------------------------------------------------------")
```

In this extended model:

1. **Extended Historical Data**: We've increased the historical data to 180 days to capture more long-term trends.
2. **Feature Engineering**: Additional features like Simple Moving Averages (SMA), Relative Strength Index (RSI), and Moving Average Convergence Divergence (MACD) are computed, providing more information to the model.
3. **Hyperparameter Tuning**: The model's hyperparameters are optimized using GridSearchCV. You can fine-tune other parameters such as `min_samples_split`, `min_samples_leaf`, and more.
4. **Cross-Validation**: Cross-validation is applied to assess the model's performance and reduce overfitting.
5. **Real-Time Data**: We use `pandas_datareader` to fetch real-time data, and the model predicts the change in the current trading session. The comparison with the predicted and actual prices is printed.

These enhancements aim to provide a more comprehensive and accurate prediction. However, it's crucial to remember that even with these improvements, stock price prediction remains inherently uncertain due to the dynamic and unpredictable nature of financial markets. The model's predictions should be used in conjunction with thorough risk management and diversified investment strategies.

Figure A8

Prompted to give a working program.

R **You**

Now give me the entire working initial prediction program, with the changes in the previous response (extended model). here is the intial trading model:

```
!pip install yfinance

import yfinance as yf
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
import datetime

# Define constants for the trading hours
TRADING_START_HOUR = 6  # 6:30 AM PST
TRADING_END_HOUR = 13   # 1:00 PM PST

# Function to check if the current time is after the trading day's end
def is_after_trading_hours():
  now = datetime.datetime.now()
  trading_day_end = datetime.time(hour=TRADING_END_HOUR, minute=0)
  return now.time() > trading_day_end

# Function to get the next trading day's date
def get_next_trading_day():
  today = datetime.datetime.now()
  # If it's Friday after trading hours, or it's Saturday or Sunday, skip to Monday
  if (today.weekday() == 4 and is_after_trading_hours()) or today.weekday() > 4:
    next_trading_day = today + datetime.timedelta(days=(7 - today.weekday()))
  else:
    next_trading_day = today if not is_after_trading_hours() else today +
datetime.timedelta(days=1)
  return next_trading_day.date()

# Predict the stock prices and percent changes for the next trading day
def predict_stock_prices(stock_symbols):
  # Initialize lists to store predictions
  predictions = []

  for symbol in stock_symbols:
    try:
      # Fetch historical data from Yahoo Finance
      data = yf.download(symbol, period="60d", interval="1d")
```

Figure B8

Prompted to give the entire working program.

```
        # Prepare the data for the model
        data['Change'] = data['Close'].diff()
        data.dropna(inplace=True)
        X = data.drop(['Close', 'Change'], axis=1)
        y = data['Change']

        # Split the dataset into the train set and the test set
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

        # Initialize and fit the StandardScaler
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)

        # Initialize and fit the RandomForest model
        model = RandomForestRegressor(random_state=0)
        model.fit(X_train_scaled, y_train)

        # Predict the price change for the next trading day
        latest_features_scaled = scaler.transform(X.iloc[-1:].values)
        predicted_change = model.predict(latest_features_scaled)[0]
        current_price = data['Close'].iloc[-1]
        predicted_price = current_price + predicted_change
        predicted_change_percent = (predicted_change / current_price) * 100

        # Store the results
        predictions.append({
            "symbol": symbol,
            "current_price": current_price,
            "predicted_price": predicted_price,
            "predicted_change_percent": predicted_change_percent
        })

    except Exception as e:
        print(f"An error occurred for {symbol}: {e}")

  # Sort the predictions by the predicted percent change
  predictions.sort(key=lambda x: x["predicted_change_percent"], reverse=True)

  return predictions[:10]  # Return only the top 10 predictions

# Read the list of S&P 500 stock symbols from a CSV file
df_stocks = pd.read_csv('/content/500_stock_List.csv', header=None)
stock_symbols = df_stocks.iloc[:, 0].values
```

Figure C8

Prompted to give the entire working program.

```
# Get the next trading day
next_trading_day = get_next_trading_day()

# Predict stock prices and percent changes for the next trading day
predictions = predict_stock_prices(stock_symbols)

# Neatly print the predictions
print(f"Top 10 Stock Predictions for {next_trading_day.strftime('%Y-%m-%d')} (End of Day)")
print("--------------------------------------------------------")
print(f"{'Symbol':<10} | {'Current Price':>17} | {'Predicted Price':>17} | {'Change (%)':>11}")
print("--------------------------------------------------------")
for prediction in predictions:
   print(f"{prediction['symbol']:<10} | "
       f"${prediction['current_price']:.4f} | "
       f"${prediction['predicted_price']:.4f} | "
       f"{prediction['predicted_change_percent']:.4f}%")
print("--------------------------------------------------------")
```

‹ 5 / 5 ›

The previous model used in this conversation is unavailable. We've switched you to the latest default model.

ChatGPT

Here's the entire initial prediction program with the enhancements mentioned in the previous response. This includes feature engineering, hyperparameter tuning, and cross-validation. The code expects the stock symbol data to be read from a CSV file, but you would need to provide the actual path to that CSV file, or otherwise adapt the code to fetch the symbols from another source.

python Copy code

```python
!pip install yfinance
!pip install pandas_datareader

import yfinance as yf
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
import datetime
from pandas_datareader import data as pdr
import fix_yahoo_finance as yf
```

Figure D8

GPT-4 gives the entire updated program.

```
# Define constants for the trading hours
TRADING_START_HOUR = 6  # 6:30 AM PST
TRADING_END_HOUR = 13   # 1:00 PM PST

# Function to check if the current time is after the trading day's end
def is_after_trading_hours():
    now = datetime.datetime.now()
    trading_day_end = datetime.time(hour=TRADING_END_HOUR, minute=0)
    return now.time() > trading_day_end

# Function to get the next trading day's date
def get_next_trading_day():
    today = datetime.datetime.now()
    # If it's Friday after trading hours, or it's Saturday or Sunday, skip to Monday
    if (today.weekday() == 4 and is_after_trading_hours()) or today.weekday() > 4:
        next_trading_day = today + datetime.timedelta(days=(7 - today.weekday()))
    else:
        next_trading_day = today if not is_after_trading_hours() else today + datetim
    return next_trading_day.date()

# Define function to calculate RSI
def compute_rsi(data, window):
    delta = data.diff()
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)
    avg_gain = gain.rolling(window=window, min_periods=1).mean()
    avg_loss = loss.rolling(window=window, min_periods=1).mean()
    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))
    return rsi

# Define function to calculate MACD
def compute_macd(data, short_window, long_window):
    ema12 = data.ewm(span=short_window, adjust=False).mean()
    ema26 = data.ewm(span=long_window, adjust=False).mean()
    macd = ema12 - ema26
    signal = macd.ewm(span=9, adjust=False).mean()
    return macd - signal

# Predict the stock prices and percent changes for the next trading day
def predict_stock_prices(stock_symbols):
    # Initialize lists to store predictions
    predictions = []
```

Figure E8

GPT-4 gives the entire updated program.

```
    for symbol in stock_symbols:
        try:
            # Fetch historical data from Yahoo Finance
            data = yf.download(symbol, period="180d", interval="1d")

            # Prepare the data for the model
            data['Change'] = data['Close'].diff()
            data['SMA_10'] = data['Close'].rolling(window=10).mean()
            data['SMA_50'] = data['Close'].rolling(window=50).mean()
            data['RSI'] = compute_rsi(data['Close'], window=14)
            data['MACD'] = compute_macd(data['Close'], 12, 26)
            data.dropna(inplace=True)

            X = data[['Open', 'High', 'Low', 'Volume', 'SMA_10', 'SMA_50', 'RSI', 'MA
            y = data['Change']

            # Split the dataset into the train set and the test set
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

            # Initialize and fit the StandardScaler
            scaler = StandardScaler()
            X_train_scaled = scaler.fit_transform(X_train)

            # Define the parameter grid for hyperparameter tuning
            param_grid = {
                'n_estimators': [50, 100, 150],
                'max_depth': [10, 20, 30]
            }

            # Initialize and fit the RandomForest model
            model = RandomForestRegressor(random_state=0)
            grid_search = GridSearchCV(model, param_grid, cv=5)
            grid_search.fit(X_train_scaled, y_train)
            best_model = grid_search.best_estimator_

            # Predict the price change for the next trading day
            latest_features = X.iloc[-1:].values
            latest_features_scaled = scaler.transform(latest_features)
            predicted_change = best_model.predict(latest_features_scaled)[0]
            current_price = data['Close'].iloc[-1]
            predicted_price = current_price + predicted_change
            predicted_change_percent = (predicted_change / current_price) * 100
```

Figure F8

GPT-4 gives the entire updated program.

```
            # Store the results
            predictions.append({
                "symbol": symbol,
                "current_price": current_price,
                "predicted_price": predicted_price,
                "predicted_change_percent": predicted_change_percent
            })

        except Exception as e:
            print(f"An error occurred for {symbol}: {e}")

    # Sort the predictions by the predicted percent change
    predictions.sort(key=lambda x: x["predicted_change_percent"], reverse=True)

    return predictions[:10]  # Return only the top 10 predictions

# Read the list of S&P 500 stock symbols from a CSV file
df_stocks = pd.read_csv('path_to_your_csv/500_stock_List.csv', header=None)
stock_symbols = df_stocks.iloc[:, 0].values

# Get the next trading day
next_trading_day = get_next_trading_day()

# Predict stock prices and percent changes for the next trading day
predictions = predict_stock_prices(stock_symbols)

# Neatly print the predictions
print(f"Top 10 Stock Predictions for {next_trading_day.strftime('%Y-%m-%d')} (End of
print("---------------------------------------------------------")
print(f"{'Symbol
```

R **You**

continue

ChatGPT

Certainly, here is the continuation and completion of the code block to neatly print the predictions:

Figure G8

GPT-4 gives the entire updated program.

python Copy code

```python
print(f"{'Symbol':<10} | {'Current Price':>17} | {'Predicted Price':>17} | {'Change
print("-----------------------------------------------------------")
for prediction in predictions:
    print(f"{prediction['symbol']:<10} | "
          f"${prediction['current_price']:.4f} | "
          f"${prediction['predicted_price']:.4f} | "
          f"{prediction['predicted_change_percent']:.4f}%")
print("-----------------------------------------------------------")
```

Make sure you change `'path_to_your_csv/500_stock_List.csv'` to the actual path where the CSV containing the stock symbols is located on your machine.

In this program, the prediction of stock prices has been done using historical stock data. The model is a RandomForestRegressor, and we've used a GridSearchCV for hyperparameter tuning. Feature engineering has been applied to include a simple moving average, the relative strength index (RSI), and the moving average convergence divergence (MACD) as additional features for the model, which could potentially improve the predictions.

The final output will list the top 10 stocks, sorted by the predicted percent change in price, in a neatly formatted table with current and predicted prices as well as the percent change. The prices and percent changes are displayed up to four decimal places, as requested.

Figure A9

GPT-4 creates the evaluation program, to see how accurate the predictions are.

You

Now create another program that takes the stock predictions and prices from the program you gave me, and compare them to the actual stock prices at the end of the trading day that was predicted. Here is an example of the output from the other program. Assume there is no csv file. Use the yfinance library to get the current stock prices:

Top 10 Stock Predictions for 2023-11-02 (End of Day)

```
------------------------------------------------------------
Symbol   |   Current Price |  Predicted Price |  Change (%)
------------------------------------------------------------
FRCB     | $0.0300 | $0.0337 | 12.3333%
GNRC     | $96.1300 | $103.0088 | 7.1557%
AIZ      | $166.4800 | $176.3050 | 5.9016%
TT       | $213.5000 | $225.6986 | 5.7136%
AMD      | $108.0400 | $114.0358 | 5.5496%
GRMN     | $113.5300 | $119.8055 | 5.5276%
ANET     | $210.8400 | $221.5104 | 5.0609%
DXCM     | $94.1300 | $97.3817 | 3.4545%
MLM      | $439.1700 | $453.9540 | 3.3664%
STX      | $71.2400 | $73.5934 | 3.3035%
```

‹ 6 / 6 ›

The previous model used in this conversation is unavailable. We've switched you to the latest default model.

ChatGPT

To compare the predicted stock prices and changes with the actual stock prices at the end of the trading day, you can create a program that fetches the current stock prices using the `yfinance` library and compares them with the predictions from the previous program. Here's a Python program that does just that:

python Copy code

```python
import yfinance as yf
import pandas as pd
import datetime

# Read the predicted stock data from the previous program
predicted_data = [
    {"symbol": "FRCB", "predicted_price": 0.0337},
    {"symbol": "GNRC", "predicted_price": 103.0088},
    {"symbol": "AIZ", "predicted_price": 176.3050},
    {"symbol": "TT", "predicted_price": 225.6986},
    {"symbol": "AMD", "predicted_price": 114.0358},
```

Figure B9

GPT-4 prints several categories: current price, predicted price, actual price, predicted percent change, and actual percent change.

```
    {"symbol": "GRMN", "predicted_price": 119.8055},
    {"symbol": "ANET", "predicted_price": 221.5104},
    {"symbol": "DXCM", "predicted_price": 97.3817},
    {"symbol": "MLM", "predicted_price": 453.9540},
    {"symbol": "STX", "predicted_price": 73.5934}
]

# Function to fetch the actual current stock prices using yfinance
def fetch_current_prices(predicted_data):
    current_prices = []
    for entry in predicted_data:
        symbol = entry["symbol"]
        try:
            # Fetch the current stock data
            data = yf.Ticker(symbol)
            current_price = data.history(period="1d").iloc[-1]['Close']
            entry["current_price"] = current_price
        except Exception as e:
            entry["current_price"] = None
            print(f"An error occurred for {symbol}: {e}")

# Fetch current stock prices
fetch_current_prices(predicted_data)

# Print the comparison between predicted and actual prices
print(f"Comparison of Predicted and Actual Stock Prices (End of Day)")
print("----------------------------------------------------------------")
print(f"{'Symbol':<8} | {'Predicted Price':>17} | {'Actual Price':>14} | {'Change (%
print("----------------------------------------------------------------")

for entry in predicted_data:
    symbol = entry["symbol"]
    predicted_price = entry["predicted_price"]
    current_price = entry.get("current_price", None)

    if current_price is not None:
        price_change = (current_price - predicted_price) / predicted_price * 100
        print(f"{symbol:<8} | ${predicted_price:.4f} | ${current_price:.4f} | {price_
    else:
        print(f"{symbol:<8} | ${predicted_price:.4f} | {'N/A':>12} | {'N/A':>16}")

print("----------------------------------------------------------------")
```

Figure 9C

Prompting GPT-4 to compare predictions to prices at 1 PM PST of trading day (end of trading day).

You

adjust the comparison model from the prevous response to compare the end of the trading day (1 PM PST) stock prices. Keep this part so I know that I can keep these stock inputs:

or result in comparison_results:

```
{"symbol": "FRCB", "predicted_price": 0.0337},
{"symbol": "GNRC", "predicted_price": 103.0088},
{"symbol": "AIZ", "predicted_price": 176.3050},
{"symbol": "TT", "predicted_price": 225.6986},
{"symbol": "AMD", "predicted_price": 114.0358},
{"symbol": "GRMN", "predicted_price": 119.8055},
{"symbol": "ANET", "predicted_price": 221.5104},
{"symbol": "DXCM", "predicted_price": 97.3817},
{"symbol": "MLM", "predicted_price": 453.9540},
{"symbol": "STX", "predicted_price": 73.5934}
```

Also make sure the user can manually enter the date to check the stock prices at the end of the day.

< 9 / 9 >

The previous model used in this conversation is unavailable. We've switched you to the latest default model.

ChatGPT

To adjust the comparison program to compare the end of the trading day (1 PM PST) stock prices and allow the user to manually enter the date, you can modify the code as follows:

python Copy code

```python
import yfinance as yf
import pandas as pd
import datetime

# Define a function to compare predicted and actual prices
def compare_predicted_actual(predictions, target_date):
    # Initialize list to store comparison results
    comparison_results = []
```

Figure 9D

Prediction program printed.

```python
    # Get the actual closing prices for the specified date
    for prediction in predictions:
        symbol = prediction['symbol']
        try:
            # Fetch actual data for the stock on the specified date
            start_date = target_date.replace(hour=13, minute=0, second=0)  # Set to 1
            end_date = target_date.replace(hour=13, minute=0, second=0) + datetime.ti
            actual_data = yf.download(symbol, start=start_date, end=end_date)
            actual_close_price = actual_data['Close'].iloc[-1] if not actual_data.emp

            if actual_close_price is not None:
                change_percent_actual = (actual_close_price - prediction['current_pri
                comparison_results.append({
                    "symbol": symbol,
                    "current_price": prediction['current_price'],
                    "predicted_price": prediction['predicted_price'],
                    "actual_price": actual_close_price,
                    "predicted_change_percent": prediction['predicted_change_percent'
                    "actual_change_percent": change_percent_actual
                })
        except Exception as e:
            print(f"An error occurred for {symbol}: {e}")

    return comparison_results

# Sample predictions for demonstration (you can replace with your actual predictions)
predictions = [
    {"symbol": "FRCB", "current_price": 0.0300, "predicted_price": 0.0337, "predicted
    {"symbol": "GNRC", "current_price": 96.1300, "predicted_price": 103.0088, "predic
    {"symbol": "ATZ", "current_price": 166.4800, "predicted_price": 176.3050, "predic
    {"symbol": "TT", "current_price": 213.5000, "predicted_price": 225.6986, "predict
    {"symbol": "AMD", "current_price": 108.0400, "predicted_price": 114.0358, "predic
    {"symbol": "GRMN", "current_price": 113.5300, "predicted_price": 119.8055, "predi
    {"symbol": "ANET", "current_price": 210.0400, "predicted_price": 221.5104, "predi
    {"symbol": "DXCM", "current_price": 94.1300, "predicted_price": 97.3817, "predict
    {"symbol": "MLM", "current_price": 439.1700, "predicted_price": 453.9540, "predic
    {"symbol": "STX", "current_price": 71.2400, "predicted_price": 73.5934, "predicte
]
```

Figure E9

Prediction program given by GPT-4, with an explanation on how to input the evaluation date.

```
# Allow the user to manually enter the date for comparison
while True:
    try:
        target_date_str = input("Enter the target date in YYYY-MM-DD format: ")
        target_date = datetime.datetime.strptime(target_date_str, '%Y-%m-%d')
        break
    except ValueError:
        print("Invalid date format. Please enter a valid date in YYYY-MM-DD format.")

# Run the comparison for the specified date
comparison_results = compare_predicted_actual(predictions, target_date)

# Neatly print the comparison results
print(f"Comparison of Predicted and Actual Prices for {target_date.strftime('%Y-%m-%d
print("-----------------------------------------------------------------------------
print(f"{'Symbol':<10} | {'Current Price':>15} | {'Predicted Price':>17} | {'Actual P
print("-----------------------------------------------------------------------------
for result in comparison_results:
    print(f"{result['symbol']:<10} | "
          f"${result['current_price']:.4f} | "
          f"${result['predicted_price']:.4f} | "
          f"${result['actual_price']:.4f} | "
          f"{result['predicted_change_percent']:.4f}% | "
          f"{result['actual_change_percent']:.4f}%")
print("-----------------------------------------------------------------------------
```

This adjusted code allows the user to enter the target date in YYYY-MM-DD format and then fetches the actual closing prices for the specified date (end of the trading day, 1 PM PST) to compare them with the predicted prices. The comparison results are printed in a formatted table. You can replace the sample predictions with your actual predictions.

Figure A10

Explained in the Code Explanations section.

Figure B10

Explained in the Code Explanations section.

Figure A11

Weekly prediction program prompt.

Anonymous

Create a machine learning model using any programming language that can be run in google colab. The program should take all of the individual stocks in the S&P 500, which are listed is in the first column of this csv path: /content/500_stock_List.csv
Keep in mind that the stocks are just listed and there is NO column heading with the word "symbol"
Predict the top 10 stock gainers for the start and end date given, which is suppose to be a weekly time frame. For example, if the program runs after the trading day starts on Monday, predict the stock prices and percent changes for the end of the day on the next week's Monday. Keep in mind that the stock market is not open for trading on the weekends. Trading days start at 6:30 AM PST and end at 1:00 PM PST.
Print out the top 10 stocks' predictions in a neat format, along with their predicted percent changes. Do not assume any code will be finished or written later. Make sure the program is as accurate as possible. Only respond to me with the coding programs I can run that accomplish these tasks, do not give me an outline or unusable code.
Make your predictions as accurate as possible, use anything you need.

change the csv file content path to this:
/content/500_stock_List.csv

also print the predicted price and predicted percent change.

Data Analysis

To create a machine learning model for predicting the top 10 stock gainers from the S&P 500, we'll need to follow several steps. Since we're aiming to run this in Google Colab, I'll provide you with a Python script that you can directly use there. The script will:

Figure B11

Instructions on the program's function, along with the code output are given.

1. Read the stock symbols from the provided CSV file.
2. Fetch historical stock data for each symbol.
3. Build and train a machine learning model on this data.
4. Predict the stock prices for the specified time frame.
5. Calculate the percent change and identify the top 10 gainers.
6. Output the results in a formatted way.

For the stock data, we can use the `yfinance` library, which allows us to fetch historical market data from Yahoo Finance. We'll use a simple machine learning model, such as a linear regressor, due to its ease of use and speed, although more complex models might yield more accurate results.

Here is the complete script:

python Copy code

```python
import pandas as pd
import yfinance as yf
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from datetime import datetime, timedelta
import numpy as np

# Function to fetch historical data for a stock symbol
def fetch_stock_data(symbol, start_date, end_date):
    stock_data = yf.download(symbol, start=start_date, end=end_date)
    return stock_data['Adj Close']

# Function to prepare data for model training
def prepare_data(stock_data):
    stock_data = stock_data.reset_index()
    stock_data['Day'] = stock_data.index
    X = stock_data[['Day']]  # Features
    y = stock_data['Adj Close']  # Target
    return train_test_split(X, y, test_size=0.2, random_state=0)
```

Figure C11

Program finishes printing. This is less advanced but uses the Linear Regression model, which is explained in the Concepts and Equations section.

```
# Function to predict future stock price
def predict_future_price(model, future_day):
    return model.predict(np.array([[future_day]]))

# Load S&P 500 stock symbols from CSV
sp500_stocks = pd.read_csv('/content/500_stock_List.csv', header=None)
symbols = sp500_stocks[0].tolist()

# Define the time frame for data fetching and prediction
end_date = datetime.now()
start_date = end_date - timedelta(days=365)  # Last year
predict_date = end_date + timedelta(days=7)  # Predict for 7 days later

# Dictionary to store predictions
predictions = {}

# Process each stock symbol
for symbol in symbols:
    try:
        # Fetch historical data
        stock_data = fetch_stock_data(symbol, start_date, end_date)

        # Prepare data for model
        X_train, X_test, y_train, y_test = prepare_data(stock_data)

        # Create and train the model
        model = LinearRegression()
        model.fit(X_train, y_train)

        # Predict future price
        future_price = predict_future_price(model, len(stock_data) + 7)
        last_price = stock_data.iloc[-1]
        percent_change = ((future_price - last_price) / last_price) * 1

        # Store prediction
        predictions[symbol] = (future_price[0], percent_change[0])
    except Exception as e:
        print(f"Error processing {symbol}: {e}")
```

Figure D11

Prints top gainers.

```
# Sort predictions by percent change
top_gainers = sorted(predictions.items(), key=lambda x: x[1][1], revers

# Print top 10 predictions
print("Top 10 Stock Predictions for the Next Week:")
for symbol, (predicted_price, percent_change) in top_gainers:
    print(f"{symbol}: Predicted Price - ${predicted_price:.2f}, Predict
```

Instructions for Google Colab:

1. Copy the script into a new Google Colab notebook.
2. Upload your `500_stock_List.csv` file to the Colab environment.
3. Install the required libraries (like `yfinance`) by running `!pip install yfinance`.
4. Run the script.

Notes:

- The script uses a linear regression model for simplicity and demonstration purposes. This is not typically used for accurate stock prediction in real-world scenarios.
- Stock market predictions are inherently uncertain and should be approached with caution.
- The model does not consider external factors like market news, global events, or company-specific developments, which can significantly impact stock prices.
- The prediction is made for a week ahead, but keep in mind that stock markets are influenced by many unpredictable factors, so the accuracy of such predictions can be limited.

Appendix B

Tables for Day-to-day Predictions

Table A1

Comparison of Predicted and Actual Prices for 2023-11-02 (End of Day 1)

Symbol	Current Price	Predicted Price	Actual Price	Pred. Change	Act. Change
FRCB	$0.030	$0.0337	$0.020	12.3333%	-33.3333%
GNRC	$96.130	$103.0088	$99.360	7.1557%	3.3600%
AIZ	$166.480	$176.3050	$163.790	5.9016%	-1.6158%
TT	$213.500	$225.6986	$213.110	5.7136%	-0.1827%
AMD	$108.040	$114.0358	$107.830	5.5496%	-0.1944%
GRMN	$113.530	$119.8055	$114.450	5.5276%	0.8104%
ANET	$210.840	$221.5104	$211.680	5.0609%	0.3984%
DXCM	$94.130	$97.3817	$93.670	3.4545%	-0.4887%
MLM	$439.170	$453.9540	$440.740	3.3664%	0.3575%
STX	$71.240	$73.5934	$71.810	3.3035%	0.8001%

Table A2.

Comparison of Predicted and Actual Prices for 2023-11-03(End of Day 2)

Symbol	Current Price	Predicted Price	Actual Price	Pred. Change	Act. Change

SBNY	$0.020	$0.0319	$0.015	59.3167%	-25.0000%
SBUX	$101.290	$108.0368	$102.650	6.6609%	1.3427%
TFX	$207.125	$220.5410	$211.030	6.4773%	1.8853%
TYL	$409.160	$435.4585	$416.250	6.4274%	1.7328%
WBD	$10.960	$11.6270	$11.770	6.0857%	7.3905%
PH	$404.156	$426.3609	$401.190	5.4942%	-0.7338%
GNRC	$101.840	$107.3458	$104.920	5.4063%	3.0244%
SEE	$33.110	$34.8812	$34.620	5.3496%	4.5606%
CDAY	$66.790	$70.1806	$65.200	5.0765%	-2.3806%

Table A3.

Comparison of Predicted and Actual Prices for 2023-11-06 (End of Day 3)

Symbol	Current Price	Predicted Price	Actual Price	Pred. Change	Act. Change
EXPE	$112.910	$125.9099	$111.5900	11.5135%	-1.1691%
PARA	$13.745	$15.1147	$12.6900	9.9651%	-7.6755%
IT	$387.280	$419.2664	$392.5400	8.2592%	1.3582%
SBNY	$0.014	$0.0151	$0.0200	8.2000%	42.8571%
MRNA	$78.065	$82.9687	$72.0700	6.2816%	-7.6795%
CZR	$44.402	$47.1506	$43.9000	6.1894%	-1.1315%

WBD	$11.775	$12.4021	$11.4500	5.3257%	-2.7601%
AAP	$57.150	$60.1575	$56.6300	5.2624%	-0.9099%
AES	$16.685	$17.5543	$16.3375	5.2098%	-2.0827%

Table A4.

Comparison of Predicted and Actual Prices for 2023-11-07 (End of Day 4)

Symbol	Current Price	Predicted Price	Actual Price	Pred. Change	Act. Change
SBNY	$0.020	$0.0266	$0.020	33.2119%	0.0000%
LUMN	$1.320	$1.4490	$1.300	9.7712%	-1.5152%
CEG	$124.730	$129.8805	$119.870	4.1293%	-3.8964%
D	$45.590	$47.0996	$45.580	3.3113%	-0.0219%
LLY	$595.190	$6114655	$599.930	2.7345%	0.7964%
VFC	$15.950	$16.3374	$15.990	2.4290%	0.2508%
BKNG	$2,971.430	$3,041.9767	$3,011.900	2.3742%	1.3620%
WYNN	$95.140	$97.3478	$93.180	2.3206%	-2.0601%
MOH	$348.530	$356.4882	$349.460	2.2834%	0.2668%

Table A5.

Comparison of Predicted and Actual Prices for 2023-11-08 (End of Day 5)

Symbol	Current Price	Predicted Price	Actual Price	Pred. Change	Act. Change
SBNY	$0.020	$0.0224	$0.010	11.9867%	-50.0000%
GEN	$18.930	$19.9612	$19.320	5.4474%	2.0602%
EXPE	$118.050	$122.8595	$116.560	4.0741%	-1.2622%
WAT	$260.670	$267.5762	$255.820	2.6494%	-1.8606%
BIO	$302.820	$310.4496	$305.140	2.5195%	0.7661%
CLX	$132.520	$135.6595	$131.930	2.3691%	-0.4452%
PAYC	$167.840	$171.7249	$167.520	2.3147%	-0.1907%
AAP	$58.380	$59.7151	$57.640	2.2870%	-1.2676%
MTCH	$29.990	$30.6617	$29.840	2.2397%	-0.5002%
A	$110.540	$112.9669	$109.390	2.1955%	-1.0403%

Table A6.

Comparison of Predicted and Actual Prices for 2023-11-09 (End of Day 6)

Symbol	Current Price	Predicted Price	Actual Price	Pred. Change	Act. Change
FRCB	$0.040	$0.0513	$0.020	28.1667%	-50.0000%
EXR	$118.530	$126.3878	$118.100	6.6294%	-0.3628%
DVA	$84.810	$88.1150	$84.310	3.8970%	-0.5896%
TTWO	$143.470	$148.4022	$146.320	3.4378%	1.9865%
LLY	$619.130	$639.1039	$591.320	3.2261%	-4.4918%
JKHY	$149.940	$154.5998	$149.210	3.1078%	-0.4869%
GEN	$19.320	$19.9101	$19.000	3.0543%	-1.6563%
WST	$345.300	$355.6693	$336.040	3.0030%	-2.6817%
FIS	$53.000	$54.4687	$51.780	2.7711%	-2.3019%
PWR	$169.760	$173.8290	$169.570	2.3969%	-0.1119%

Table A7.

Comparison of Predicted and Actual Prices for 2023-11-13 (End of Day 7)

Symbol	Current Price	Predicted Price	Actual Price	Pred. Change	Act. Change
SBNY	$0.014	$0.0205	$0.018	43.9162%	25.8741%
HOLX	$72.130	$75.3526	$71.400	4.4677%	-1.0121%
LRCX	$685.430	$711.1783	$673.430	3.7565%	-1.7507%
WBD	$10.130	$10.5027	$9.890	3.6795%	-2.3692%
MCHP	$77.560	$80.3397	$76.620	3.5839%	-1.2120%
AMAT	$150.680	$155.6505	$149.740	3.2987%	-0.6238%
KLAC	$534.250	$551.7923	$528.250	3.2835%	-1.1231%
AMD	$118.590	$122.4451	$116.790	3.2508%	-1.5178%
QRVO	$91.660	$94.6063	$90.370	3.2144%	-1.4074%
MPWR	$504.580	$520.5932	$507.120	3.1736%	0.5034%

Table A8.

Comparison of Predicted and Actual Prices for 2023-11-14 (End of Day 8)

Symbol	Current Price	Predicted Price	Actual Price	Pred. Change	Act. Change
SBNY	$0.018	$0.0223	$0.017	24.0175%	-5.5556%
DVA	$88.380	$91.8750	$92.120	3.9545%	4.2317%
HSIC	$67.500	$69.8445	$68.900	3.4733%	2.0741%
DXCM	$98.350	$101.7618	$101.380	3.4691%	3.0808%
BSX	$53.710	$55.2244	$54.020	2.8196%	0.5772%
BA	$204.540	$210.3013	$207.470	2.8167%	1.4325%
TSLA	$223.710	$228.4147	$237.410	2.1030%	6.1240%
RCL	$97.810	$99.7475	$102.810	1.9809%	5.1119%
MOS	$34.500	$35.1294	$35.910	1.8243%	4.0870%
SYK	$281.840	$286.9014	$283.030	1.7959%	0.4222%

Table A9.

Comparison of Predicted and Actual Prices for 2023-11-15 (End of Day 9)

Symbol	Current Price	Predicted Price	Actual Price	Pred. Change	Act. Change
ENPH	$91.390	$99.3748	$92.810	8.7371%	1.5538%
LUMN	$1.300	$1.4100	$1.380	8.4628%	6.1538%
SEDG	$79.350	$85.1574	$80.040	7.3187%	0.8696%
BXP	$56.690	$60.7847	$57.020	7.2230%	0.5821%
ARE	$105.320	$112.8484	$105.710	7.1481%	0.3703%
EXR	$129.550	$138.2878	$130.000	6.7447%	0.3474%
NWL	$7.360	$7.8520	$7.660	6.6851%	4.0761%
FSLR	$149.140	$158.6778	$152.420	6.3952%	2.1993%
CCL	$13.890	$14.7294	$14.570	6.0432%	4.8956%
MHK	$86.730	$91.8022	$86.330	5.8483%	-0.4612%

Table A10.

Comparison of Predicted and Actual Prices for 2023-11-16 (End of Day 10)

Symbol	Current Price	Predicted Price	Actual Price	Pred. Change	Act. Change
TGT	$130.460	$144.1118	$129.890	10.4644%	-0.4369%
VFC	$17.770	$19.3428	$17.210	8.8509%	-3.1514%
CTLT	$39.560	$41.8922	$39.870	5.8953%	0.7836%
EXPE	$130.330	$137.0379	$136.380	5.1469%	4.6421%
ETSY	$71.740	$74.9120	$72.530	4.4215%	1.1012%
PARA	$13.170	$13.7448	$13.190	4.3643%	0.1519%
CRL	$187.570	$195.6387	$189.040	4.3017%	0.7837%
NWL	$7.660	$7.9816	$7.550	4.1983%	-1.4360%
CCL	$14.570	$15.1677	$14.790	4.1023%	1.5100%
LUMN	$1.380	$1.4356	$1.350	4.0290%	-2.1739%

Table A11.

Comparison of Predicted and Actual Prices for 2023-11-17 (End of Day 11)

Symbol	Current Price	Predicted Price	Actual Price	Pred. Change	Act. Change
SIVBQ	$0.030	$0.0420	$0.020	39.9105%	-33.3333%
INTC	$43.350	$45.7277	$43.810	5.4848%	1.0611%
SBNY	$0.017	$0.0178	$0.017	4.6286%	0.0000%
GEN	$20.400	$20.9855	$20.660	2.8703%	1.2745%
ISRG	$303.950	$312.1209	$305.280	2.6882%	0.4376%
DXCM	$104.770	$107.3922	$104.970	2.5028%	0.1909%
TGT	$129.940	$132.8336	$129.890	2.2269%	-0.0385%
SHW	$269.300	$275.0610	$270.280	2.1392%	0.3639%
GNRC	$113.480	$115.6829	$114.120	1.9412%	0.5640%
ALL	$134.859	$137.4563	$134.190	1.9260%	-0.4961%

Table A12.

Comparison of Predicted and Actual Prices for 2023-11-20 (End of Day 12)

Symbol	Current Price	Predicted Price	Actual Price	Pred Change	Act Change
SBNY	$0.017	$0.0188	$0.016	10.6049%	-5.8823%
ROST	$128.820	$134.8374	$129.430	4.6712%	0.4735%
EXPE	$136.380	$142.5671	$134.900	4.5366%	-1.0852%
DISH	$3.570	$3.7253	$3.650	4.3511%	2.2409%
ALB	$127.390	$131.1294	$130.360	2.9354%	2.3314%
PARA	$13.190	$13.5261	$13.930	2.5482%	5.6103%
MTCH	$32.430	$33.1995	$32.590	2.3729%	0.4934%
VTRS	$9.450	$9.6741	$9.470	2.3718%	0.2116%
ETSY	$72.530	$74.2216	$72.070	2.3323%	-0.6342%
SYF	$29.880	$30.5237	$29.820	2.1543%	-0.2008%

Appendix C

Table for Week-long Prediction

Table B1.

Comparison of Predicted and Actual Prices from 2023-11-17 to 2023-11-24

Symbol	Predicted Price	Actual Price	Predicted Change	Actual Change
ALGN	$320.400	$219.4800	57.74%	-31.5000%
SEDG	$113.370	$78.0800	48.16%	-31.1300%
PAYC	$253.260	$178.4900	45.85%	-29.5200%
ILMN	$133.550	$98.3400	40.28%	-26.3600%
ON	$94.480	$69.2800	36.03%	-26.6700%
OGN	$14.870	$11.3600	35.84%	-23.6000%
HAS	$58.670	$46.1500	30.00%	-21.3400%
FTNT	$65.340	$53.4300	29.59%	-18.2300%
XRAY	$35.910	$31.3200	21.32%	-12.7800%
ALB	$147.430	$128.8700	20.26%	-12.5900%