Name: Endri Lohja
Matriculation number: 29251
University mail: endri.lohja@hsrw.org
Personal Mail: endrilohja11@gmail.com
Project github link: https://github.com/EndriLohja11/amas_23_project

Stock Price Prediction Analysis in Python

Table of contents

General Introduction

The trading market has been a staple of the economy since forever, and as such stocks are highly important. From the valuation of the company and its future to people speculating to become rich, because they think they got a better idea, or they are smarter. Being a highly talked topic and an issue of great importance, many funds, ETFs and investing firms have been founded.
It is to no surprise that us humans like to gamble, however investing in the stock market is not gambling. Every determined price comes from very complex mathematical, economical and statistical models, and these computations give us a better understanding of the stock.
Although technically it is mathematics, the market itself is so complex and volatile that most investor prefer to bet long-term rather than short-term. This indicates that trying to predict a price on the near future not only is more a gamble than a calculation, but it is highly unsafe and not recommended if you are not a professional trader.

In this stock analysis we will undergo a simple yet very helpful overview on knowing how to value a stock, key indicators, price history and a set of methods to speculate on the market.

This project is mainly done on Jupyter Notebook, however it can also be done in any text editor of choice. Keep in mind that in Jupyter we can print and graph line by line, which we cannot do in running a simple python file, that would output everything when interpreted.

It is recommended to use a python virtual environment, mentioning venv (virtualenv), conda, pyenv etc. The job of the virtual environment is to manage our dependencies and libraries, also we can specifically choose the python version we want to run. In this case I went with venv, however I would recommend conda for most users, as it is a bit more friendly on its syntax.

The machine I am using is running Fedora 38, but you should be comfortable following on Windows and Mac as well. The first thing that is required is having a version of python installed, you can check by writing on your terminal (cmd for windows): `python -V`

If you do not have python installed on your system, which that would be the case only for windows, then search for https://www.python.org/downloads/ and download the appropriate file for you system, as per UNIX systems simply install it via you package manager whatever that might be.

In this report there is python code, so basic knowledge in python and programming paradigm is encouraged.

The paper will be divided on sections according to their functionality and usage.

Understanding our Data

After we make sure the python is in our system and we also have created a virtual environment, we can start installing the necessary libraries. We need to install 7 libraries and we do so by using **pip** the package manager for python. We can run **pip install** with the names of the libraries we need divided by space and no comma. We will be using **yfinance**, **pandas**, **numpy**, **sklearn**, **matplotlib**, **statsmodels**.

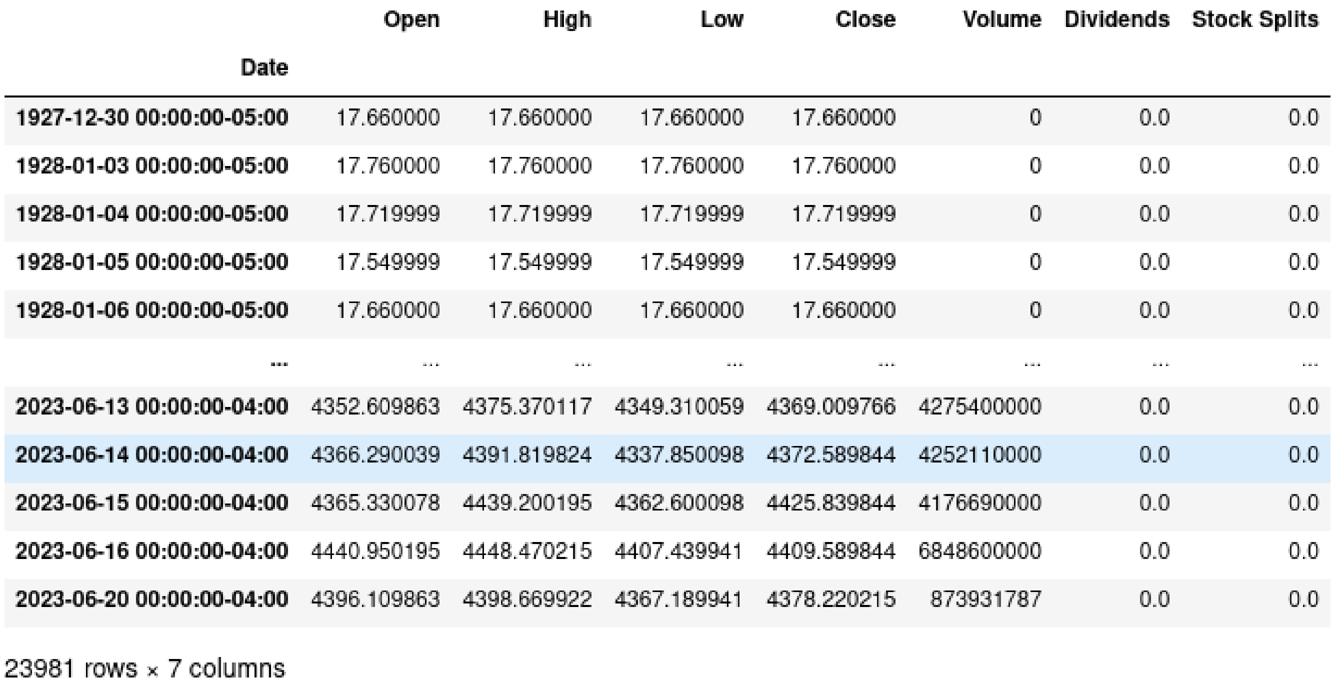
After we install them we open up our notebook and import them in a cell.

```
import yfinance as yf
import pandas as pd
import numpy as np
```

Using the Ticker method from yfinance library I assign the GSPC which is the code name for SP500 in Yahoo Finance to the sp500 variable, and then set the history limit to maximum

```
# Retrieve S&P 500 data
sp500 = yf.Ticker('^GSPC')
sp500 = sp500.history(period="max")
```

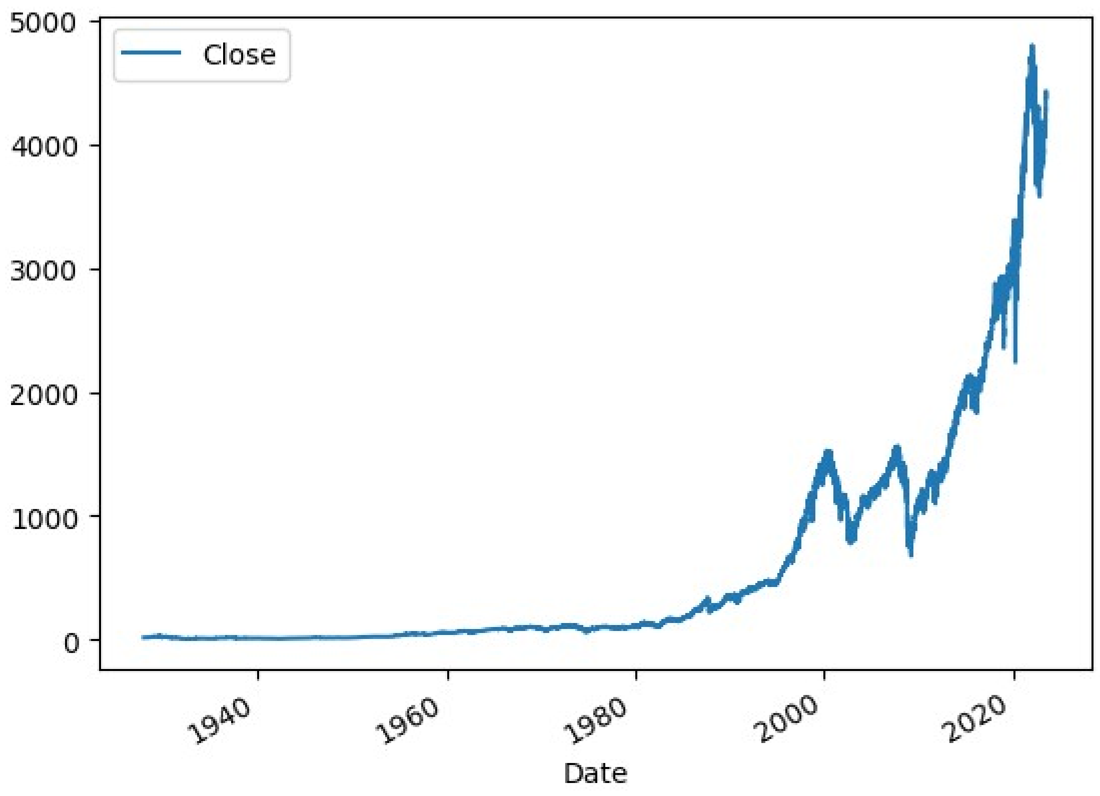
Now the variable sp500 contains the historical data of the real S&P500.



	Open	High	Low	Close	Volume	Dividends	Stock Splits
Date							
1927-12-30 00:00:00-05:00	17.660000	17.660000	17.660000	17.660000	0	0.0	0.0
1928-01-03 00:00:00-05:00	17.760000	17.760000	17.760000	17.760000	0	0.0	0.0
1928-01-04 00:00:00-05:00	17.719999	17.719999	17.719999	17.719999	0	0.0	0.0
1928-01-05 00:00:00-05:00	17.549999	17.549999	17.549999	17.549999	0	0.0	0.0
1928-01-06 00:00:00-05:00	17.660000	17.660000	17.660000	17.660000	0	0.0	0.0
...	...	...	...	...	...	...	...
2023-06-13 00:00:00-04:00	4352.609863	4375.370117	4349.310059	4369.009766	4275400000	0.0	0.0
2023-06-14 00:00:00-04:00	4366.290039	4391.819824	4337.850098	4372.589844	4252110000	0.0	0.0
2023-06-15 00:00:00-04:00	4365.330078	4439.200195	4362.600098	4425.839844	4176690000	0.0	0.0
2023-06-16 00:00:00-04:00	4440.950195	4448.470215	4407.439941	4409.589844	6848600000	0.0	0.0
2023-06-20 00:00:00-04:00	4396.109863	4398.669922	4367.189941	4378.220215	873931787	0.0	0.0

23981 rows × 7 columns

By running sp500.plot.line(y="Close", use_index=True), we can see the price over the years on a line graph.



I delete the columns named Dividends and Stock Splits since they are not relevant for our case, however if we want to track individual stocks such as INTEL or MICROSOFT those can be very valuable. Let me explain the names of the columns. The column Open represents the price on the first moment the market opened on that day, the High represents the highest price during the whole day, the Low represents the lowest price during the whole day, the Close represents the price at the end of the day, and the Volume represents how much trading volume was going on that particular day. Although the Volume column won't make much of a difference on our analysis we will still keep it.
We create another column which is called tomorrow and holds the value of the Close column of the next day.

```
sp500["Tomorrow"] = sp500["Close"].shift(-1)
```

Now we create yet another column which is called Target and its a boolean value on whether the Tomorrow value is bigger or not than the Today's value. We use type int, because is easier for usage on Machine Learning top work with integer

```
sp500["Target"] = (sp500["Tomorrow"] > sp500["Close"]).astype(int)
```

For a couple of reasons such as the ".com" boom on the 90's and the low trading volume on tech, also the time efficiency we use the subset since 1987.

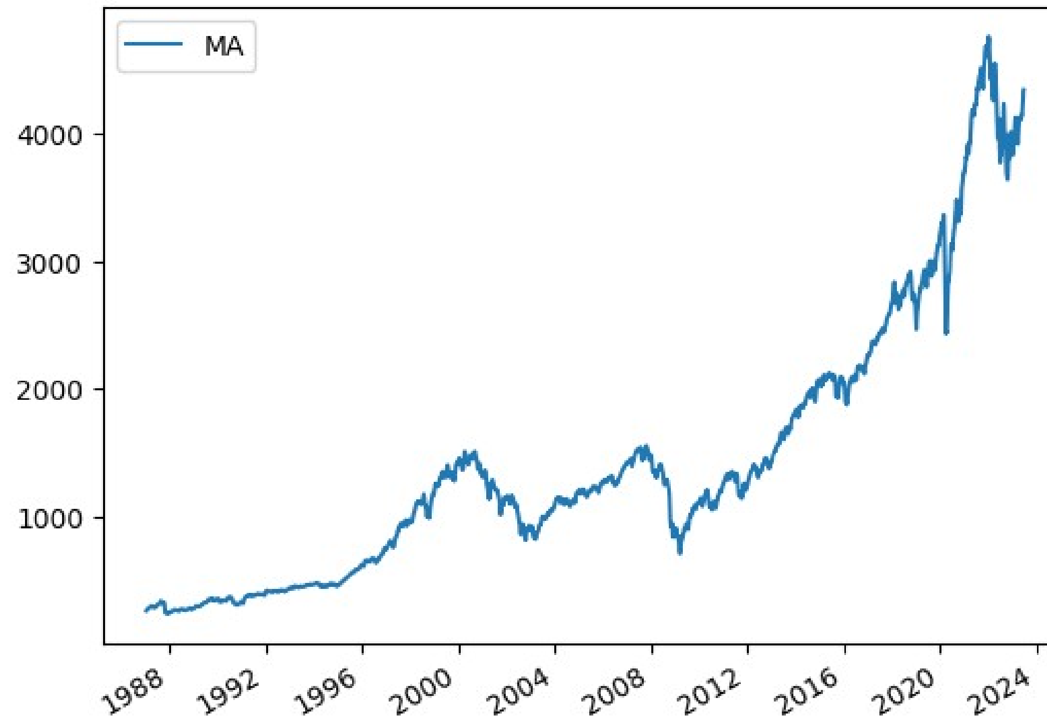
```
sp500 = sp500.loc["1987-01-01":].copy()
```

Moving Average (MA) Model

Moving Average (MA) is a widely used technique for time series analysis and forecasting, including stock price prediction. It calculates the average of past data points over a specified window size to identify trends and smooth out noise in the data.

As many strategies it has its pros and cons, on the pro side we can mention that it is a relatively easy technique, and it does not require complex computation, even for very large datasets simple parallelism can be used to enhance the algorithms capability. It also serves as a trend indicator and test for support and resistance levels, since it is based on previous data. However, we should be careful on our window size, because the market often has systematical changes, and not all the data is valuable.
On the cons, it is necessary to mention its lagging indicator, meaning that once the algorithm detects a price jump it may be already too late. Sensitivity to window size its also something that should be taken into consideration, together with the false signals it may create.

```
window_size = 10
sp500["MA"] = sp500['Close'].rolling(window=window_size).mean()
sp500.plot.line(y="MA", use_index=True)
```



This graph indicates the price increase of the sp500 from 1988, as you can see there are some dips noticeable there, the famous 2008 and 2020.

```
last_ma = sp500["MA"].iloc[-1]  # Get the last moving average value
next_prediction = last_ma       # Consider it as the forecast for the ne
print("Next prediction:", next_prediction)
```

```
Next prediction: 4343.833984375
```

The computation is fairly easy. We get the one position before last moving average and compare it to the stock now, and as you can see it detected a slight decrease, although the stock in fact increased.

Monte Carlo Simulation

Monte Carlo simulation is a statistical technique used to model and analyze complex systems, such as stock price predictions. It involves generating random samples of possible future price paths for a stock based on historical data and a statistical model. This approach incorporates uncertainty, provides flexible modeling options, and offers probabilistic estimates of future prices. However, it relies on assumptions that may oversimplify reality and historical data that may not capture all market dynamics.

The pros of Monte Carlo simulation include its ability to account for uncertainty, its flexibility in modeling various factors, and its provision of probabilistic estimates for different scenarios. On the other hand, the cons include reliance on simplified assumptions, potential biases from historical data, and the exclusion of certain market behaviors. Despite these limitations, Monte Carlo simulation is essential in modeling as it enables a comprehensive understanding of potential outcomes and associated probabilities, assisting in making informed investment decisions and managing risks effectively.

```python
sp500['Log_Return'] = np.log(sp500['Close'] / sp500['Close'].shift(1))

mean_return = sp500['Log_Return'].mean()
std_return = sp500['Log_Return'].std()

num_simulations = 1000
num_days = 30
simulations = np.random.normal(mean_return, std_return, size=(num_simulations, num_days))

initial_price = sp500['Close'].iloc[-1]
simulated_prices = initial_price * np.exp(np.cumsum(simulations, axis=1))
```

```python
# Assuming you have historical prices in the "sp500" dataframe and simulated prices in the

# Get the actual prices for the specific period
actual_prices = sp500['Close'].tail(num_days)

# Calculate the accuracy percentage
accuracy_count = sum(simulated_prices[:, -1] >= actual_prices.iloc[-1])
accuracy_percentage = (accuracy_count / num_simulations) * 100

print("Accuracy percentage: {:.2f}%".format(accuracy_percentage))
```

```
Accuracy percentage: 56.40%
```

Firstly, the code calculates the logarithmic returns of the stock prices in the 'sp500' dataframe. This is done by taking the natural logarithm of the ratio between the closing prices of consecutive days. The logarithmic returns are stored in a new column called 'Log_Return'.

The mean and standard deviation of the logarithmic returns are then computed using the 'mean()' and 'std()' functions. These values will be used in the Monte Carlo simulation.

Next, the code specifies the number of simulations to be performed (num_simulations) and the number of days to predict (num_days).

The simulations are generated using the 'np.random.normal()' function, which generates random numbers from a normal distribution. The mean and standard deviation used for the random number generation are the mean_return and std_return calculated earlier. The size parameter is set to (num_simulations, num_days) to generate a matrix of random numbers representing the simulated returns.

The initial price of the stock is extracted from the 'Close' column of the 'sp500' dataframe using 'iloc[-1]', which retrieves the last value. This initial price is then used to calculate the simulated prices by applying the exponential of the cumulative sum of the simulated returns along the rows of the simulations matrix.

In order to assess the accuracy of the simulation, the actual prices for the specified period are extracted from the 'Close' column of the 'sp500' dataframe using 'tail(num_days)'. This captures the prices corresponding to the last num_days in the historical data.

The code compares the simulated prices for the last day of the simulation period with the actual prices. The number of simulations where the simulated price is greater than or equal to the actual price is counted using 'sum(simulated_prices[:, -1] >= actual_prices.iloc[-1])'.

Finally, the accuracy percentage is calculated by dividing the count of accurate simulations by the total number of simulations, multiplying by 100, and printing the result.

This code allows for the estimation of the accuracy of the Monte Carlo simulation in predicting stock prices based on the simulated returns. The accuracy percentage provides an indication of how well the simulation performed in capturing the direction of the actual prices for the specified period. And as seen the calculations is not as accurate as we can blindly trust this algorithm.

	Open	High	Low	Close	Volume	Tomorrow	Target	MA	Log_Return
Date									
1987-01-02 00:00:00-05:00	242.169998	246.449997	242.169998	246.449997	91880000	252.190002	1	NaN	NaN
1987-01-05 00:00:00-05:00	246.449997	252.570007	246.449997	252.190002	181900000	252.779999	1	NaN	0.023024
1987-01-06 00:00:00-05:00	252.199997	253.990005	252.139999	252.779999	189300000	255.330002	1	NaN	0.002337
1987-01-07 00:00:00-05:00	252.779999	255.720001	252.649994	255.330002	190900000	257.279999	1	NaN	0.010037
1987-01-08 00:00:00-05:00	255.360001	257.279999	254.970001	257.279999	194500000	258.730011	1	NaN	0.007608
...	...	...	...	...	...	...	...	...	...
2023-06-13 00:00:00-04:00	4352.609863	4375.370117	4349.310059	4369.009766	4275400000	4372.589844	1	4280.911035	0.006909
2023-06-14 00:00:00-04:00	4366.290039	4391.819824	4337.850098	4372.589844	4252110000	4425.839844	1	4300.187012	0.000819
2023-06-15 00:00:00-04:00	4365.330078	4439.200195	4362.600098	4425.839844	4176690000	4409.589844	0	4320.668994	0.012105
2023-06-16 00:00:00-04:00	4440.950195	4448.470215	4407.439941	4409.589844	6848600000	4388.709961	0	4333.390967	-0.003678
2023-06-20 00:00:00-04:00	4396.109863	4400.149902	4367.189941	4388.709961	4055790000	NaN	0	4344.882959	-0.004746

Autoregressive Integrated Moving Average (ARIMA) Model

The Autoregressive Integrated Moving Average (ARIMA) model is a commonly used time series analysis technique for stock trading and price predictions. It combines the concepts of autoregression (AR), differencing (I), and moving average (MA) to capture the patterns and trends in historical stock price data.

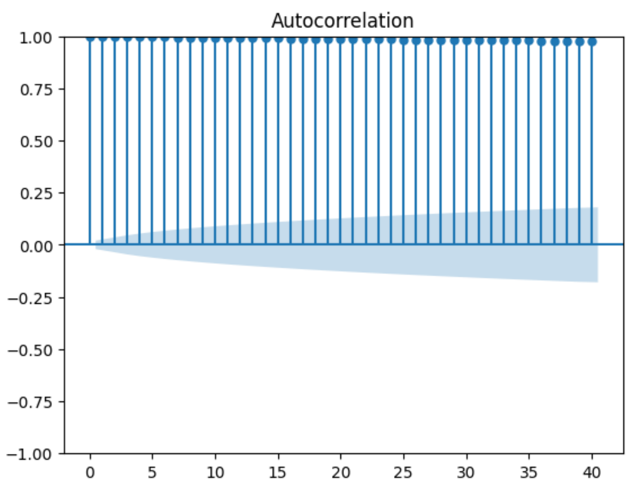
In the context of stock trading and price predictions, the ARIMA model can be applied as follows:

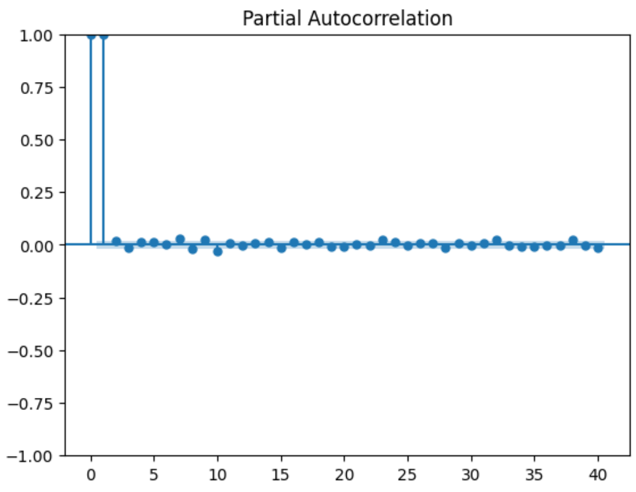
Autoregression (AR): The AR component of the model considers the relationship between an observation and a certain number of lagged observations (previous prices). It assumes that the current value of a stock price depends on its past values. By analyzing the autocorrelation of the historical stock price data, the AR component helps capture the inherent serial dependence in the time series.

Differencing (I): The differencing component of ARIMA is used to remove the trend or seasonality present in the stock price data. Differencing involves subtracting the current observation from a previous observation to obtain the difference. This helps make the time series stationary, meaning that its statistical properties, such as mean and variance, remain constant over time. Stationarity is important for accurate modeling and prediction.

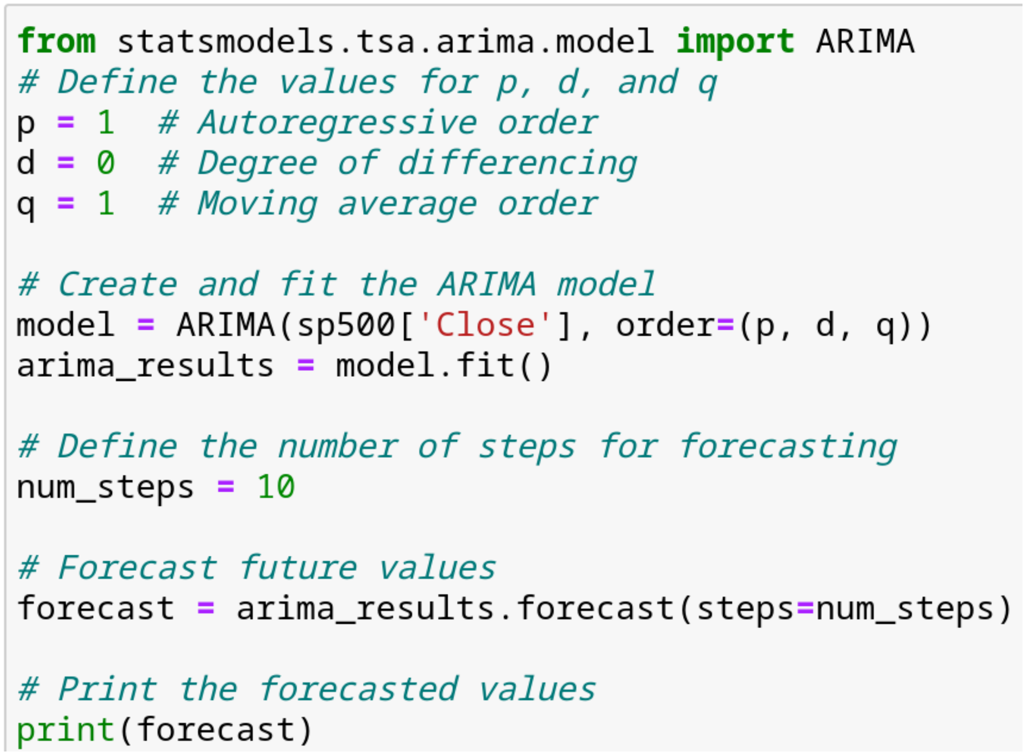
Moving Average (MA): The MA component of ARIMA considers the dependency between an observation and a residual error from a moving average model applied to lagged observations. It helps capture the short-term fluctuations and random shocks in the stock price data. The MA component smooths out the noise and identifies the underlying patterns in the time series.

By combining these three components, the ARIMA model can provide insights into future stock price movements. It takes into account both the historical price data and the patterns identified through autoregression, differencing, and moving average calculations. ARIMA models are widely used in stock trading and price predictions as they can help forecast potential price trends, identify buy/sell signals, and inform investment decisions.





The provided graphs demonstrate the use of the statsmodels library to generate plots of the autocorrelation function (ACF) and partial autocorrelation function (PACF) for time series analysis. By importing the plot_acf and plot_pacf functions from the statsmodels.graphics.tsaplots module, users can easily visualize the correlation patterns in their time series data. The plot_acf function creates a plot that showcases the correlation values between the time series and its lagged values at different time intervals. Similarly, the plot_pacf function generates a plot illustrating the partial correlation values while accounting for the effects of shorter lags. These plots are valuable in identifying any significant autocorrelation or partial autocorrelation patterns in the time series data. By using the matplotlib.pyplot module and calling plt.show(), the generated plots are displayed for easy interpretation and analysis. Overall, this provides a convenient and efficient way to assess the autocorrelation and partial autocorrelation characteristics of a time series.



```
from statsmodels.tsa.arima.model import ARIMA
# Define the values for p, d, and q
p = 1  # Autoregressive order
d = 0  # Degree of differencing
q = 1  # Moving average order

# Create and fit the ARIMA model
model = ARIMA(sp500['Close'], order=(p, d, q))
arima_results = model.fit()

# Define the number of steps for forecasting
num_steps = 10

# Forecast future values
forecast = arima_results.forecast(steps=num_steps)

# Print the forecasted values
print(forecast)
```

ARIMA class from the statsmodels.tsa.arima.model module is used to perform an Autoregressive Integrated Moving Average (ARIMA) analysis on a time series data represented by the 'Close' column of the sp500 variable.

The values for the order of the ARIMA model are defined: p for the autoregressive order, d for the degree of differencing, and q for the moving average order.

An ARIMA model is created and fitted to the 'Close' column using the specified order.

The variable num_steps is defined to determine the number of future steps to forecast.

The forecast variable stores the forecasted values for the future steps based on the fitted ARIMA model.

Finally, the forecasted values are printed using the print function.

Overall, this code segment demonstrates how to define the order of an ARIMA model, fit the model to the data, forecast future values, and print the forecasted results.

Machine Learning Approach

Machine learning models provide several advantages when it comes to stock price prediction. One of the major benefits is automation. These models can efficiently analyze vast amounts of data and make predictions automatically, eliminating the need for manual analysis. This not only saves time but also reduces effort compared to traditional methods.

Another advantage is pattern recognition. Machine learning algorithms excel at identifying complex patterns and relationships within historical data. They can uncover intricate correlations that may be challenging for humans to discern. By uncovering these patterns, machine learning models can provide valuable insights for predicting future stock prices.

Flexibility is another strength of machine learning models. They can handle a wide range of input features, allowing them to adapt to changing market conditions. This adaptability enables the models to capture new trends and adjust their predictions accordingly, enhancing their accuracy and relevance.

Furthermore, machine learning models provide a quantitative framework for decision-making. They enable systematic and data-driven predictions by leveraging mathematical algorithms and statistical techniques. This quantitative analysis enhances the objectivity and reliability of stock price predictions, reducing reliance on subjective assessments.

Despite their advantages, machine learning approaches for stock price prediction also face certain challenges. One significant hurdle is data quality and availability. To train reliable machine learning models, accurate and comprehensive historical data is essential. Incomplete or noisy data can lead to inaccurate predictions and undermine the effectiveness of these models.

Another challenge is overfitting. Machine learning models may become overly focused on the training data, capturing noise and specific patterns that do not generalize well to unseen data. This phenomenon can result in poor performance when applied to new data, diminishing the reliability of the predictions.

Model complexity and interpretability pose additional difficulties. Some machine learning algorithms, such as ensemble methods like Random Forest, can be complex and difficult to interpret. The complexity of these models makes it challenging to understand the underlying factors driving the predictions, limiting their transparency and interpretability.

Lastly, market unpredictability poses a significant challenge for stock price prediction. Various external factors, including economic conditions, geopolitical events, and investor sentiment, can influence stock prices. The inherent unpredictability of financial markets makes accurate predictions challenging, even with advanced machine learning techniques.

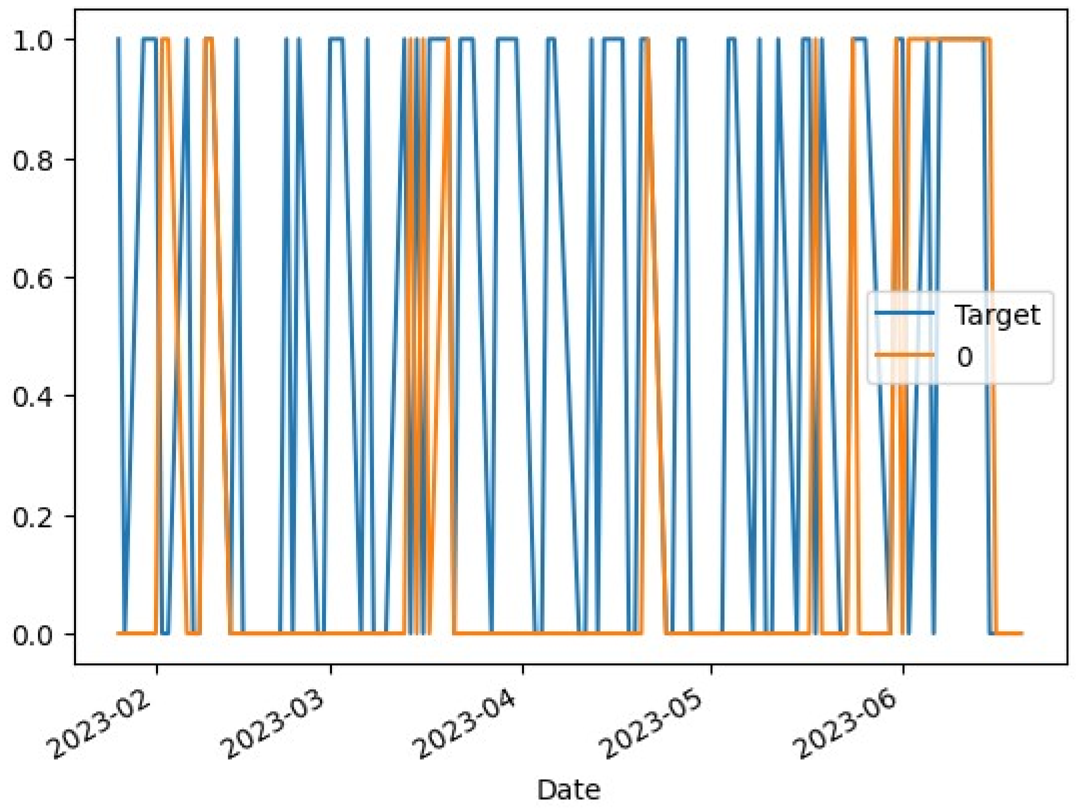
In summary, utilizing machine learning for stock price prediction offers the advantages of automation, pattern recognition, flexibility, and quantitative analysis. However, it also faces challenges related to data quality, overfitting, model complexity, interpretability, and the inherent unpredictability of financial markets.

```python
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100, min_samples_split=100, random_state=1)

train = sp500.iloc[:-100]
test = sp500.iloc[-100:]

predictors = ["Close", "Volume", "Open", "High", "Low"]
model.fit(train[predictors], train["Target"])
```

```
RandomForestClassifier
RandomForestClassifier(min_samples_split=100, random_state=1)
```

```python
from sklearn.metrics import precision_score

preds = model.predict(test[predictors])
preds = pd.Series(preds, index=test.index)
precision_score(test["Target"], preds)
```

```
0.6190476190476191
```



The graph visually shows how well the predictions align with the actual values. If the predicted values closely follow the actual values, it suggests that the model's predictions are accurate and capture the underlying patterns in the data. Conversely, if there are significant differences or discrepancies between the predicted and actual values, it indicates that the model's predictions may not align well with the true values.

The code implements a machine learning approach, specifically using a Random Forest classifier, for stock price prediction. It trains the model on historical stock data, splitting it into a training set and a test set. The model is then used to make predictions on the test set. The precision score is calculated to evaluate the accuracy of the predictions. The code also includes functions for backtesting, which involves iteratively training the model on past data and making predictions on subsequent windows of data. This allows for assessing the model's performance over time. The code further incorporates additional predictors derived from rolling averages and trends over different time horizons to potentially improve the prediction accuracy. The updated model is backtested again, and the precision score is calculated accordingly. Overall, the code showcases the process of training a Random Forest model, making predictions, backtesting, and evaluating the precision of the predictions, while also exploring the impact of additional predictors on the model's performance.

Conclusion

In this stock price prediction analysis, we explored various techniques and approaches to understand and speculate on stock market data. We started by discussing the importance of stocks in the economy and the complexity involved in determining stock prices. It was emphasized that predicting short-term stock prices can be risky and challenging, often requiring the expertise of professional traders.

We then proceeded to analyze the stock data using different methods. We employed moving average (MA) models to identify trends and support/resistance levels in the data. While MA models provide a relatively simple approach, they have limitations such as being lagging indicators and sensitivity to window size.

We also utilized Monte Carlo simulation to generate random samples and simulate possible future price paths based on historical data. This approach incorporated uncertainty and provided probabilistic estimates of future prices. However, it relied on assumptions and historical data that may not capture all market dynamics, indicating the need for caution when interpreting the results.

The Autoregressive Integrated Moving Average (ARIMA) model was applied to capture patterns and trends in the stock price data. By combining autoregression, differencing, and moving average components, the ARIMA model offered insights into future price movements. It is a widely used technique for time series analysis and stock price prediction.

Additionally, we briefly touched upon the application of machine learning approaches in stock price prediction. Machine learning models automate the analysis of large amounts of data, identify complex patterns, and provide quantitative frameworks for decision-making. However, challenges such as data quality, overfitting, and model complexity need to be addressed for reliable predictions.

Overall, this analysis provided an overview of different methods and techniques for stock price prediction. It highlighted the importance of understanding data, considering limitations, and being cautious when making predictions in the dynamic and complex stock market.