Time Series Forecasting with ARIMA

Time Series Forecasting is a **crucial tool** for making informed decisions in various fields such as **weather prediction**, **sales analysis**, **business planning**, and **stock market analysis**. One popular method for Time Series Forecasting is the **ARIMA model**, short for **Autoregressive Integrated Moving Average**. In this article, we'll delve into the process of Time Series Forecasting using ARIMA with the **Python programming language**.

Understanding ARIMA: **ARIMA** is an acronym for **Autoregressive Integrated Moving Average**. It's a **powerful algorithm** specifically designed for forecasting **Time Series Data**. ARIMA models are defined by three key parameters denoted as **ARIMA(p, d, q)**, where:

- 'p' signifies the **number of lagged observations** included in the model, capturing the **autoregressive component**.
- 'd' represents the **degree of differencing** needed to make the time series **stationary**. A value of **0** indicates stationary data, while **1** suggests **seasonal data**.
- 'q' indicates the **size of the moving average window**, capturing the **moving average component** of ARIMA.

By grasping these parameters and understanding how they interact, we can effectively employ ARIMA models to generate accurate forecasts for **time-dependent datasets**. Let's explore this further by implementing ARIMA in Python for Time Series Forecasting.

To embark on Time Series Forecasting with ARIMA, our initial step involves retrieving historical data on Google's stock prices using the Yahoo Finance API. This API offers a comprehensive source of financial data, enabling us to collect a detailed dataset spanning a defined timeframe. This dataset forms the cornerstone of our analysis, facilitating insights into Google's stock price dynamics over time.

```
#Below is a Python code snippet demonstrating how you can retrieve the latest stock price data using the Yahoo Financ
#pip install yfinance


import pandas as pd
import yfinance as yf
import datetime
from datetime import date, timedelta
today = date.today()


d1 = today.strftime("%Y-%m-%d")
end_date = d1
d2 = date.today() - timedelta(days=360)
d2 = d2.strftime("%Y-%m-%d")
start_date = d2

data = yf.download('AAPL',
                      start=start_date,
                      end=end_date,
                      progress=False)
print(data.head())
```

```
                  Open        High         Low       Close   Adj Close  \
Date
2023-03-29  159.369995  161.050003  159.350006  160.770004  159.916428
2023-03-30  161.529999  162.470001  161.270004  162.360001  161.497971
2023-03-31  162.440002  165.000000  161.910004  164.899994  164.024475
2023-04-03  164.270004  166.289993  164.220001  166.169998  165.287750
2023-04-04  166.600006  166.839996  165.110001  165.630005  164.750626

              Volume
Date
2023-03-29  51305700
2023-03-30  49501700
2023-03-31  68749800
2023-04-03  56976200
2023-04-04  46278300
```

The provided code retrieves stock price data spanning from the current date back to the previous 360 days. However, in this dataset, the Date information is not presented as a separate column; instead, it serves as the index. To facilitate the utilization of this data for various data science tasks, it's essential to transform this index into a regular column. Below are the steps demonstrating how to accomplish this:

```
data["Date"] = data.index
data = data[["Date", "Open", "High",
             "Low", "Close", "Adj Close", "Volume"]]
data.reset_index(drop=True, inplace=True)
print(data.head())
```

```
        Date        Open        High         Low       Close   Adj Close  \
0 2023-03-29  159.369995  161.050003  159.350006  160.770004  159.916428
1 2023-03-30  161.529999  162.470001  161.270004  162.360001  161.497971
2 2023-03-31  162.440002  165.000000  161.910004  164.899994  164.024475
3 2023-04-03  164.270004  166.289993  164.220001  166.169998  165.287750
4 2023-04-04  166.600006  166.839996  165.110001  165.630005  164.750626

     Volume
0  51305700
1  49501700
2  68749800
3  56976200
4  46278300
```

The resultant dataset mirrors the format typically obtained from Yahoo Finance, providing comprehensive stock price data accessible through Python. This method effectively retrieves stock price information, aligning with the structure commonly encountered in datasets acquired from Yahoo Finance.

```
import pandas as pd
import yfinance as yf
import datetime
from datetime import date, timedelta
today = date.today()



d1 = today.strftime("%Y-%m-%d")
end_date = d1
d2 = date.today() - timedelta(days=365)
d2 = d2.strftime("%Y-%m-%d")
start_date = d2


data = yf.download('GOOG',
                      start=start_date,
                      end=end_date,
                      progress=False)
data["Date"] = data.index
data = data[["Date", "Open", "High", "Low", "Close", "Adj Close", "Volume"]]
data.reset_index(drop=True, inplace=True)
print(data.tail())
```

```
          Date        Open        High         Low       Close   Adj Close  \
246 2024-03-18  149.369995  152.929993  148.139999  148.479996  148.479996
247 2024-03-19  148.979996  149.619995  147.009995  147.919998  147.919998
248 2024-03-20  148.789993  149.759995  147.664993  149.679993  149.679993
249 2024-03-21  150.320007  151.304993  148.009995  148.740005  148.740005
250 2024-03-22  150.190002  152.550003  150.089996  151.770004  151.770004

       Volume
246  47676700
247  17748400
248  17730000
249  19843900
250  19207179
```

We now proceed to select and isolate the crucial columns, 'Date' and 'Close' prices, from the dataset. This focused selection is paramount for our subsequent analysis, ensuring that we work exclusively with the essential data elements required for Time Series Forecasting and other data science tasks. By prioritizing these columns, we optimize the efficiency of our analysis, maintaining a sharp focus on the most pertinent information. This streamlined approach enhances the clarity and effectiveness of our data processing and modeling efforts, setting a solid foundation for robust insights and informed decision-making.

```
data = data[["Date", "Close"]]
print(data.head())
```

```
         Date       Close
0  2023-03-24  106.059998
1  2023-03-27  103.059998
2  2023-03-28  101.360001
3  2023-03-29  101.900002
4  2023-03-30  101.320000
```

```
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
plt.figure(figsize=(15, 10))
plt.plot(data["Date"], data["Close"])
```

```
[<matplotlib.lines.Line2D at 0x7e458339bd00>]
```

Using ARIMA for Time Series Forecasting

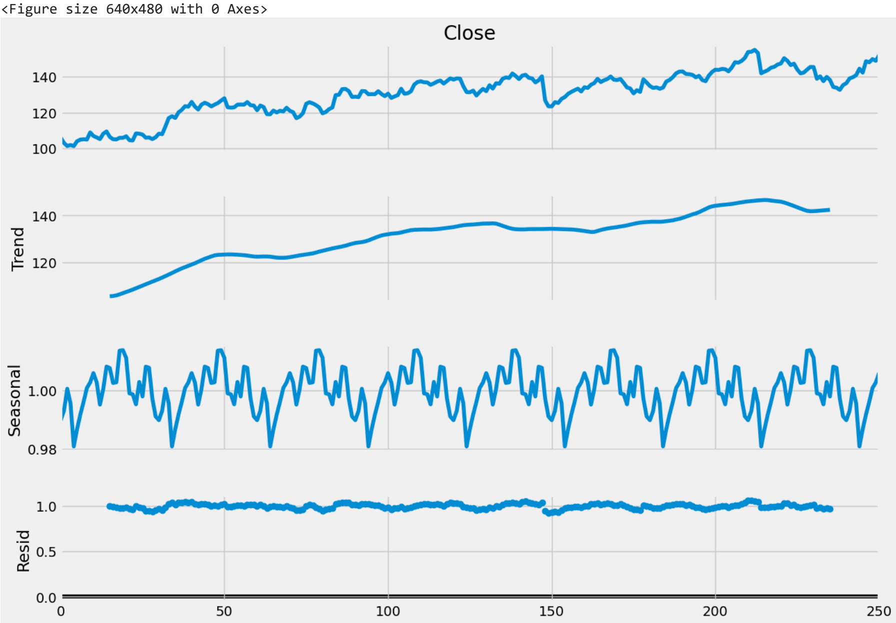
Before applying the ARIMA model for Time Series Forecasting, it's essential to determine whether our dataset exhibits stationarity or seasonality. The visualization of the closing stock prices graph above indicates that our dataset lacks stationarity. To conduct a thorough assessment of stationarity and seasonality within our dataset, we can employ the seasonal decomposition method. This technique decomposes the time series data into distinct components, namely trend, seasonality, and residuals, providing valuable insights into the underlying patterns of the time series data.

- **Assessment of Data**: Before proceeding with modeling, it's crucial to evaluate the stationarity or seasonality of the dataset.
- **Utilizing Seasonal Decomposition**: The seasonal decomposition method is employed to dissect the time series data, revealing its inherent components: trend, seasonality, and residuals.
- **Insight Generation**: By decomposing the data, we gain a deeper understanding of its underlying patterns, enabling more informed modeling decisions for Time Series Forecasting.

```
from statsmodels.tsa.seasonal import seasonal_decompose
# Determine an appropriate period for seasonality based on the frequency of the data
# For daily data, we can try periods of 7 for weekly seasonality, or 30 for monthly seasonality
result = seasonal_decompose(data["Close"], model='multiplicative', period=30)

# Plot the decomposed components
fig = plt.figure()
fig = result.plot()
fig.set_size_inches(15, 10)
```

```
<Figure size 640x480 with 0 Axes>
```



Our dataset's seasonality prompts the use of the Seasonal ARIMA (SARIMA) model for Time Series Forecasting. However, we'll initially explore the ARIMA model to grasp both approaches comprehensively.

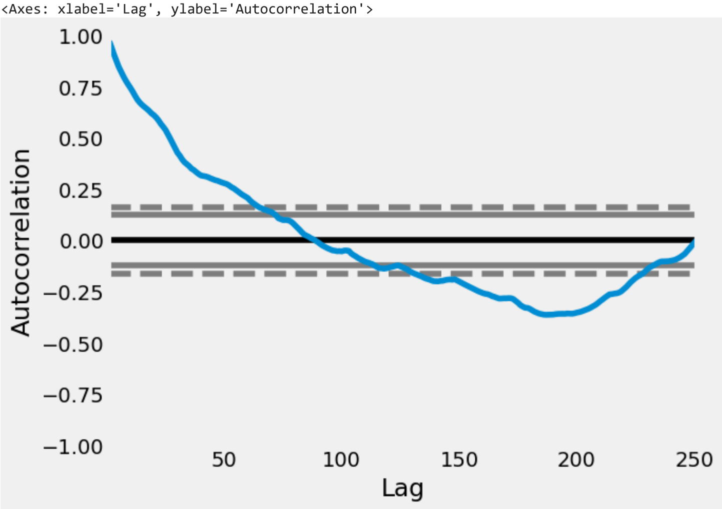
Parameter Determination:

- **p Determination**: We identify 'p' by examining the autocorrelation of the 'Close' column.
- **q Determination**: The value of 'q' is derived from the partial autocorrelation plot.
- **d Selection**: 'd' is set to 1 for seasonal data; 0 for stationary data.

This process ensures that we select suitable parameters for the ARIMA or SARIMA model, paving the way for effective Time Series Forecasting.

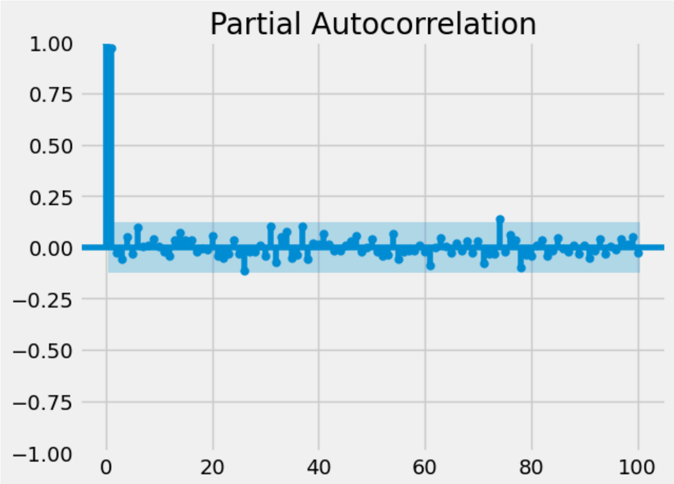
```
pd.plotting.autocorrelation_plot(data["Close"])
```

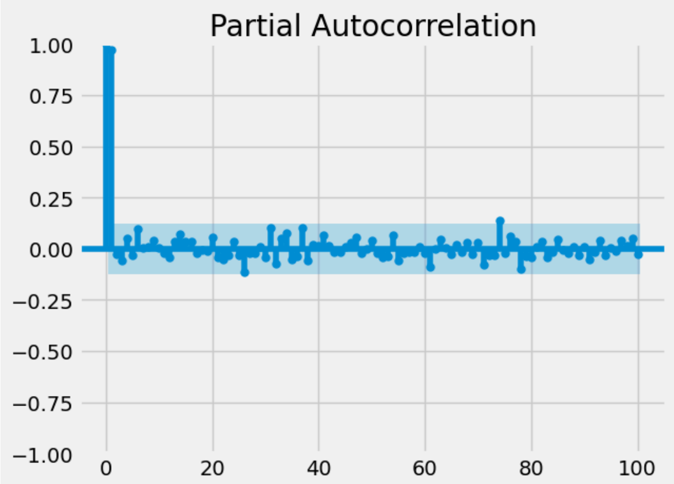
```
<Axes: xlabel='Lag', ylabel='Autocorrelation'>
```



Based on the autocorrelation plot provided, we observe the curve descending after the 5th line of the first boundary. This delineates our choice for the p-value, which is determined as 5. Now, we proceed to ascertain the value of q (moving average):

```
from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(data["Close"], lags = 100)
```





From the partial autocorrelation plot presented, we discern that only 1 point extend significantly beyond the others. This characteristic informs our determination of the q value, which we identify as 2. With both the p and q values established, let's proceed to construct an ARIMA model.

```
p, d, q = 5, 1, 2

# Import ARIMA from the new module
from statsmodels.tsa.arima.model import ARIMA

# Create ARIMA model
model = ARIMA(data["Close"], order=(p, d, q))

# Fit the model
fitted = model.fit()

# Display model summary
print(fitted.summary())
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary
  warn('Non-stationary starting autoregressive parameters'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible
  warn('Non-invertible starting MA parameters found.'
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Close   No. Observations:                  251
Model:                 ARIMA(5, 1, 2)   Log Likelihood                -557.557
Date:                Sat, 23 Mar 2024   AIC                           1131.114
Time:                        00:08:42   BIC                           1159.286
Sample:                             0   HQIC                          1142.452
                                - 251
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.2390      0.117     10.610      0.000       1.010       1.468
ar.L2         -0.8830      0.155     -5.706      0.000      -1.186      -0.580
ar.L3         -0.0964      0.134     -0.717      0.473      -0.360       0.167
ar.L4          0.1566      0.134      1.165      0.244      -0.107       0.420
ar.L5         -0.1394      0.090     -1.554      0.120      -0.315       0.036
ma.L1         -1.2306      0.096    -12.860      0.000      -1.418      -1.043
ma.L2          0.9025      0.108      8.343      0.000       0.691       1.115
sigma2         5.0591      0.270     18.759      0.000       4.531       5.588
===================================================================================
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):               591.55
Prob(Q):                              0.86   Prob(JB):                         0.00
Heteroskedasticity (H):               1.50   Skew:                            -1.25
Prob(H) (two-sided):                  0.07   Kurtosis:                        10.11
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

```
predictions = fitted.predict()
print(predictions)
```

```
0        0.000000
1      106.060008
2      103.003640
3      101.246225
4      102.103144
          ...
246    141.868696
247    147.644047
248    147.889796
249    148.955205
250    149.246772
Name: predicted_mean, Length: 251, dtype: float64
```

Building an ARIMA model on seasonal time series data often yields inaccurate predictions due to its inability to effectively capture seasonal patterns. To address this limitation, we turn to the Seasonal ARIMA (SARIMA) model, specifically designed to handle such data.

Here's the approach to construct a SARIMA model:

```
import statsmodels.api as sm
import warnings
model=sm.tsa.statespace.SARIMAX(data['Close'],
                                order=(p, d, q),
                                seasonal_order=(p, d, q, 12))
model=model.fit()
print(model.summary())
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihoo
  warnings.warn("Maximum Likelihood optimization failed to "
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                Close   No. Observations:                  251
Model:             SARIMAX(5, 1, 2)x(5, 1, 2, 12)   Log Likelihood                -543.069
Date:                              Sat, 23 Mar 2024   AIC                           1116.139
Time:                                      00:09:58   BIC                           1168.223
```

```
Sample:                                  0   HQIC                          1137.130
                                     - 251
Covariance Type:                       opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.2329      0.167      7.381      0.000       0.906       1.560
ar.L2         -0.7925      0.167     -4.753      0.000      -1.119      -0.466
ar.L3         -0.1118      0.141     -0.793      0.428      -0.388       0.165
ar.L4          0.1497      0.144      1.041      0.298      -0.132       0.431
ar.L5         -0.1598      0.099     -1.616      0.106      -0.354       0.034
ma.L1         -1.2460      0.164     -7.587      0.000      -1.568      -0.924
ma.L2          0.8350      0.132      6.329      0.000       0.576       1.094
ar.S.L12      -0.8961      2.441     -0.367      0.714      -5.681       3.889
ar.S.L24      -0.0736      0.376     -0.196      0.845      -0.811       0.664
ar.S.L36      -0.1441      0.165     -0.873      0.383      -0.468       0.179
ar.S.L48      -0.1626      0.473     -0.344      0.731      -1.090       0.765
ar.S.L60      -0.0174      0.221     -0.078      0.937      -0.451       0.417
ma.S.L12      -0.2038      2.602     -0.078      0.938      -5.304       4.896
ma.S.L24      -0.7642      2.268     -0.337      0.736      -5.209       3.681
sigma2         4.7634      2.222      2.144      0.032       0.409       9.118
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):               396.06
Prob(Q):                              0.99   Prob(JB):                         0.00
Heteroskedasticity (H):               1.38   Skew:                            -0.87
Prob(H) (two-sided):                  0.16   Kurtosis:                         9.08
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Let's proceed to forecast future stock prices for the upcoming 10 days using the SARIMA model.

```
predictions = model.predict(len(data), len(data)+10)
print(predictions)
```

```
251    151.784203
252    152.056600
253    152.043783
254    152.372362
255    151.242114
```