Unit 8 Case Study

MSDS Fall ‘19

7333 Quantify the World

Daniel Serna, Bruce Granger, and Brandon de la Houssaye

October 20, 2019

**Abstract**

In this paper we provide an in-depth time-series analysis on the four-year historical price of Walmart stock. We use both an intuition based approached and brute-force approach to determine optimal parameters for an ARIMA model.

**1 Introduction**

We have pulled four years of stock price data for Walmart (WMT). Walmart is a publicly traded company incorporated within the United States.

The company is a retailer with retail operations in well over 35 countries throughout the world from both a retail and services standpoint. The company is also globally active in its supply chain footprint and directly or indirectly responsible for vast quantities of imports/exports in a number of countries.

Walmart is a Fortune 100 company and the largest private employer in the world that operates in all 50 US states. The company first went public on Oct 1, 1970 with a stock price of $16.50 per share.

The company is often regarded as a bellwether of US financial health and growth amongst other things. For example, the 'Walmart effect' can refer to patterns utilized by the CDC for tracking flu outbreaks as well as macro consumer habits.

However, Walmart's stock price is clearly not only impacted by the company's performance or the US economy overall. Factors such as investor sentiment and competitor impacts may have a severe impact on pricing. As a result, it is useful to consider Walmart's stock price change over time in the context of whether it is stationary, cyclical, random, etc.

To begin, we will consider 4 years of financial data (Figure 1) in order to fully assess the stock market price over the last four years.

**2 Data**

The team pulled four years of Walmart stock data from Yahoo Finance for analysis. While the data contained many interesting data points such as high, low, volume, close, adjusted close, the team focused on closing price for analysis. As can be seen in Figure 1, the four-year historical close price shows a positive linear trend with a spike in first quarter 2018.

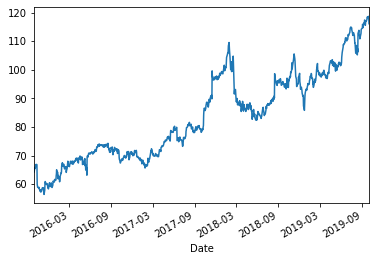


Figure 1 - Four Year Closing Price

The data is well formed and did not contain missing data points. When tested for stationarity, we can see the Dickey-Fuller test-statistic is greater than all the critical values, as seen in Figure 2.

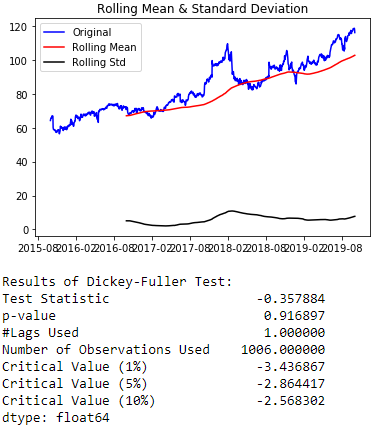


Figure 2 - Dickey-Fuller Test

The above information shows us something very basic, the Walmart data is not stationary and in fact the data is displaying an upward trend. There is clearly some cyclicality, or randomness in the data. We choose a rolling mean of two hundred fifty-three because this value represents the number of trading days in a year. The rolling mean information overlays quite well with the actual mean information (showing patterns as opposed to extreme outliers) of trend. While the std deviation results appear flat, with a slight jump. It is not perfectly smooth, which would indicate that the results carry some randomness. Thus, the data isn't stationary or purely seasonal.

**3 Data Cleansing**

Fortunately, the data is well formed and did not contain missing values which eased the data cleansing process. As mentioned, the data contained one-thousand and six (1,006) records and eight attributes, but the team choose to focus on the closing price attribute for this time series analysis. The team created a data frame consisting of closing price and date for analysis. While no NA’s were noticed in the four-year data, the team decided to add “dropna()” function calls when interacting with the data to futureproof the analysis if an expanded time window was desired.

**4 Methods**

The team decided to adopt a two-prong approach when analyzing the data. First, the team used intuition to determine optimal autoregressive (p), integrated (d), and moving average (q) parameters for an ARIMA model. A rule-based approach was followed as part of this analysis. Second, the team applied a brute force approach to determine the optimal parameters for p, d and q. Every combination of parameters was attempted and an observation of the Residual Sum of Squares and Akaike Information Criteria for each attempt was used to determine the optimal parameters. Ultimately, the team combined analysis from both of these approaches to arrive at the optimal choice of p, d, and q parameter values.

**5 Results**

The team first looked at the autocorrelation (ACF) and partial autocorrelation (PACF) plots of the data, as seen in Figure 3 and Figure 4 respectively. Autocorrelation is a form of statistical correlation (i.e., the measured strength in relationship between two variables) in the time series data (which is what we are managing here). By using the autocorrelation function, we are measuring the statistical relationship between the time series data versus the time series data from a previous time step (or periods); which are called 'lags'. That is, the time series of data is correlated over (and over, depending on defined 'lags') against the same series data from a previous period(s).

Whereas, with partial autocorrelation, we get a summary of the relationship between time series sets from different times steps - same as for autocorrelation - but without observations at intervening (or non-compared) time steps. In other words, partial autocorrelation attempts to get a more complete picture of the entire population of time series data.

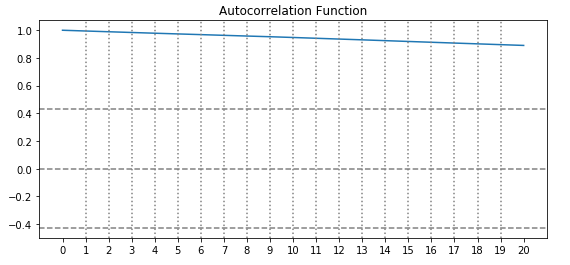


Figure 3 – Autocorrelation

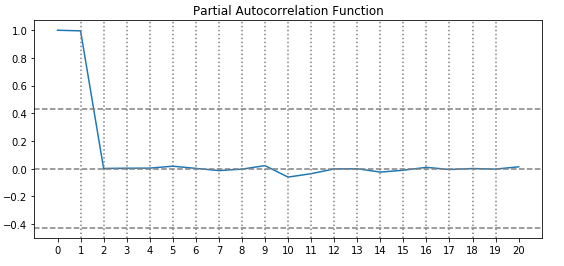


Figure 4 - Partial Autocorrelation

What does this information tell us? First, it tells us that we definitely have non-stationary data. Second, it tells us that to further our analysis we will need to incorporating differences and potentially moving averages.

What follows next is an exploration of ARIMA with focus on hyper tuning of the p, d, and q parameters. The team decided to wrap up the ARIMAS exploration with a grid search of the p, d, and q parameters in order to arrive at the best possible combination, as measured by LOSS. We also explored the AIC - a basic accuracy score - metric in order to consider the parameter sets for the ARIMAS model. Ultimately, this positioned the team to decipher past time events in order to understand/predict future events, given previous time events occurred in a pattern over time and are not readily explainable by outside of other, unprovided for variables.

We first ran our ARIMA model with p, d, and q parameters all set to 0 as seen in Figure 5.

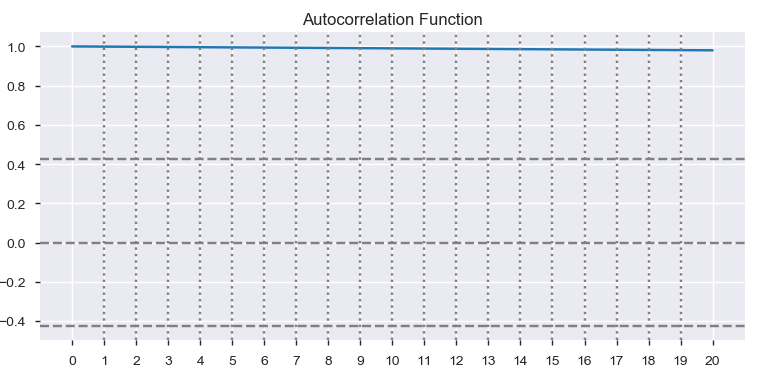


Figure 5 - P=0, D=0, Q=0

Leveraging ARMIA Rule #1, which relates to differencing, we observed that this series has positive correlations out to a high number of lags. For that reason, we introduced one higher level of differencing, with results seen in Figure 6.

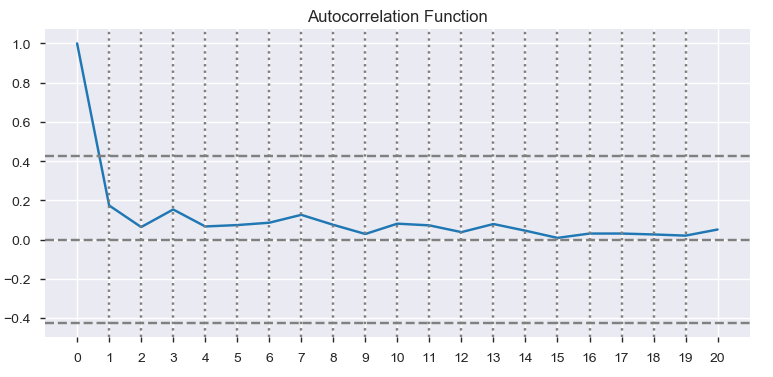


Figure 6 - P=0, D=1, Q=0

Leveraging ARINA Rule #2, the lag-1 autocorrelation is not zero or negative, but the autocorrelations are all small and there doesn't seem to be much in the way of a discernable pattern. For purposes of exploring this data deeper, we decided to see what happens when a differencing of 2 is introduced. We decided to introduced an additional lag because we did not see evidence of over differencing after lag-1. These results are seen in Figure 7.

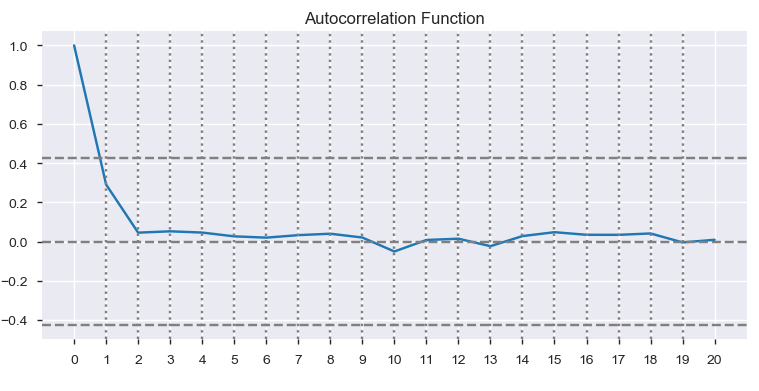


Figure 7 - P=0,D=2,Q=0

These results of an additional lag were interesting. After lag-2, the autocorrelation seems to arrive at '0'. This would indicate that any additional differencing would severely run the risk of over differencing.

As for the other rules not yet covered (for parameter d), in using two orders of differencing (thus far) it is apparent that the original series has a time varying trend (Rule #4). Simply put, the model does not seem to be stationary or constant in trend. Rules 5 and Rules 3 do not necessarily apply: this model has at least 1 order of differencing, so it does not require a constant.

We next adjusting the autoregressive parameter (p), leaving all the other parameters set to zero. We began by setting p=2, the results of which are seen in Figure 8.

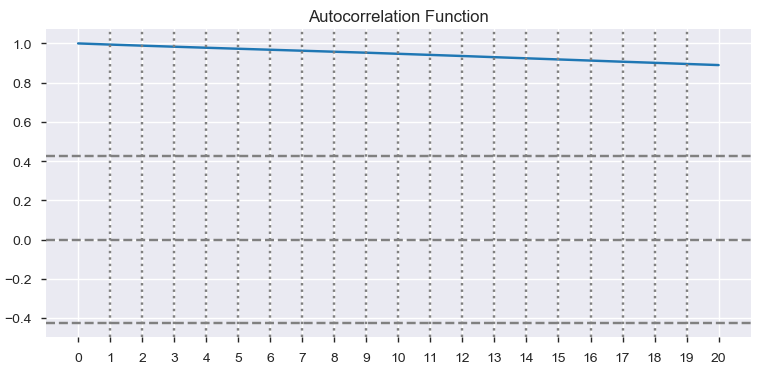


Figure 8 - P=2, D=0, Q=0

This autocorrelation has the same pattern at the original autocorrelation function. We can infer - understandably - that whether the parameter is 1 or 2, the autoregression is occurring between the current time series observations and prior time series observations. Thus, the primary parameter of consideration is really d (differences). We ran the model again setting p=2, d=2, and q=0 and these results can be seen in Figure 9.

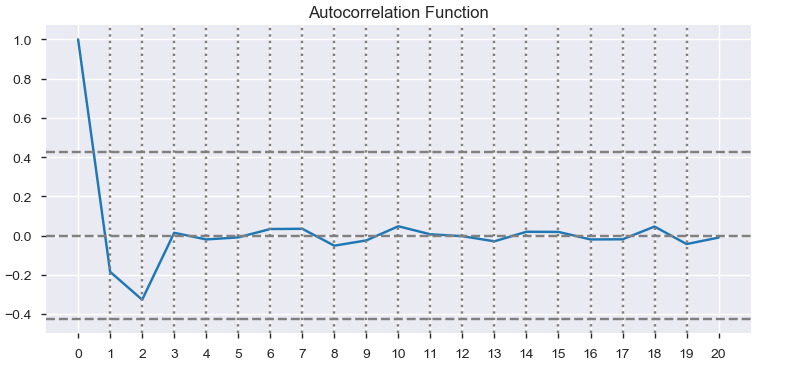


Figure 9 - P=2, D=2, Q=0

Combining the autoregression with the differences creates a problem that violates ARIMA rule #7. The problem that was created is that the time series is now over differenced. This problem is true whether p is set to 1 or 2. Next we ran the model with all parameters set to 2 and the results are seen in Figure 10.

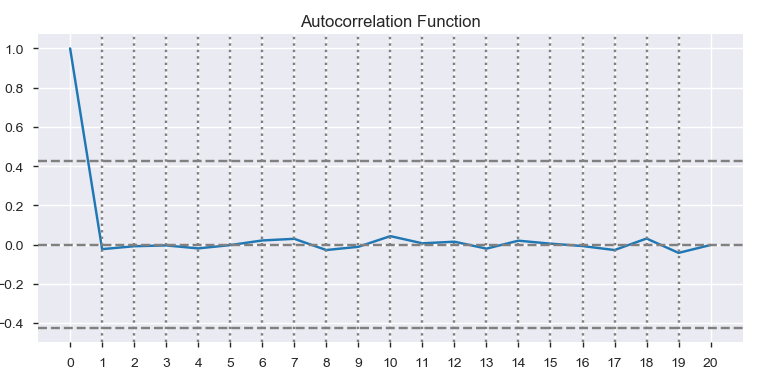


Figure - P=2, D=2, Q=2

Here we set AR and MA terms to equal each other. What is interesting is that, given prior variations of the model run, it is clear the terms cancel each other out. That is, it wouldn't matter if they were each set to "2" or "1", in either case the terms of the overall model is displaying correlation. As a result, we will revert both parameters back to zero.

After going through various iterations of the ARIMA model parameters p, d, q, it appears visually that the most appropriate p, d, q, settings are either 0, 1, 0 or 0, 2, 0. It is also possible that a value of d=2 might be slightly over differenced and d=1 might be more appropriate.

In order to assess this conclusion, we will now perform a 'brute force' analysis using Loss scores (while also pulling AIC scores). The RSS and AIC values for each run are seen in Figure 11.

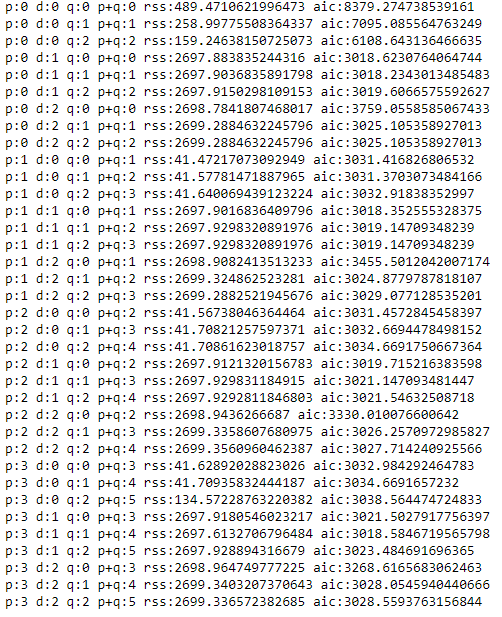


Figure - Brute Force Results

Using RSS values as the indicator of the best model, the optimal parameters determined by the brute-force method are seen in Figure 12.

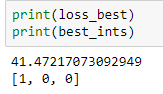


Figure - Optimal Parameters

As a determinant for the best combination of p, d, q parameter set for the ARIMA model, the team used the loss function used in the Residual Sum of Squares. Running the grid search around these parameters (p, d, q) we arrive at a set of 1,0,0. This is of course very different from the parameter combinations we arrived at when manually tuning the model. The team appreciates that the parameters do in fact interact with one another, specifically p and q, but the grid search is useful (where human intuition becomes limited) in exploring all combinations.

However, the grid search model was not constructed with 'rules' to prevent overfitting of the model with the exception of putting into the function rule 8 (if (p > 1 and q <= 1) or (q > 1 and p <= 1) or (p <= 1 and q <= 1)). This all results in the necessity to utilize the results of both the grid search (to maximize accuracy) while also considering the manual fine-tuning (to ensure that the model is not overfitted - as considered through the autocorrelation visualizations against period lags).

Considering all factors, the team would not set the parameters of p, d, q at 1,0,0 because rules 1 and 2 would be immediately violated (i.e., the data clearly remains 'unistationary'). The team questioned the sole use of RSS in the brute-force method as sole determiner of best model and decided to also look at the AIC (Akaike Information Criteria) value in combination with intuition.

Considering the AIC values produced during the brute-force exercise, the team concluded that the correct parameter settings would be 0,1,1. This means that the parameters settings of p, d, and q, satisfied all ARIMA rules and the AIC score was minimized. The team focused on setting the difference parameter to its minimum where the rules could be met in order to maximize the accuracy. Opposed to a situation where the team only focused on RSS values, no difference parameter (d) would have been included.

The AIC value for our optimal parameter setting of p=0, d=1, and q=1 is 3018.23430.

**6 Conclusion**

The team found that a combination of intuition and brute-force parameter tuning produced the best results. Each method by itself produced a combination of sub-optimal parameter values. Our intuition based approached immediately demonstrated some order of differencing would be necessary, however, we failed to determine autoregressive or moving average parameters would be necessary. In contrast, the brute-force method identified an autoregressive parameter of 1 was necessary but failed to determine differencing was needed. Combining our intuition with the brute-force approach allowed us to determine looking at additional metrics might be prudent. By also looking at AIC values produced from the brute-force method, the team determined p, d, q values of 0, 1, 1 produced the optimal results while still adhering to rules.

Lastly, the team observes the final results, in terms of the parameters selected: a moving average as well as a difference, made a lot of sense given the original dataset. The Walmart stock price was clearly trending upwards over time. The trend was not constant (so differencing was needed), and the elements that caused the trend to not be constant were seemingly random. Thus, the MA parameters were increased to smooth out these variances.

**Appendix I Code**

**from pandas import datetime**

**from matplotlib import pyplot as plt**

**import matplotlib**

**%matplotlib inline**

**import pandas\_profiling**

**import pandas as pd**

**from pathlib import Path**

**import numpy as np**

**from numpy.polynomial.polynomial import polyfit**

**from sklearn.linear\_model import LinearRegression**

**from statsmodels.tsa.seasonal import seasonal\_decompose**

**from statsmodels.tsa.stattools import acf, pacf**

**from statsmodels.tsa.arima\_model import ARIMA**

import datetime

pd.core.common.is\_list\_like = pd.api.types.is\_list\_like

from pandas\_datareader import data as web

start = datetime.datetime(2015, 10, 1)

end = datetime.datetime(2019, 10, 1)

# Change 'iex' to 'yahoo' and capitalize 'close'

wmt\_stock = web.DataReader('WMT', 'yahoo', start, end)['Close']

wmt\_stock.index = pd.to\_datetime(wmt\_stock.index)

profile = pandas\_profiling.ProfileReport(wmt\_temp)

profile

print(wmt\_stock.head())

fig = plt.figure()

wmt\_stock.plot()

plt.title("Wal-Mart Stock Price Over Time")

plt.xlabel("Time")

plt.ylabel('WMT Stock Price')

plt.show()

from statsmodels.tsa.stattools import adfuller

def test\_stationarity(timeseries):

#Determing rolling statistics

rolmean = timeseries.rolling(253).mean()

rolstd = timeseries.rolling(253).std()

#Plot rolling statistics:

orig = plt.plot(timeseries, color='blue',label='Original')

mean = plt.plot(rolmean, color='red', label='Rolling Mean')

std = plt.plot(rolstd, color='black', label = 'Rolling Std')

plt.legend(loc='best')

plt.title('Rolling Mean & Standard Deviation')

plt.show(block=False)

#Perform Dickey-Fuller test:

global dftest

print('Results of Dickey-Fuller Test:')

dftest = adfuller(timeseries, autolag='AIC')

dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])

for key,value in dftest[4].items():

dfoutput['Critical Value (%s)'%key] = value

print(dfoutput)

test\_stationarity(wmt\_stock)

def test\_stat\_vs\_crit\_val():

for key,value in dftest[4].items():

if dftest[0] > value:

print("The Test Statistic of", dftest[0], "is greater than the Critical Value", key, "value of", value)

elif dftest[0] < value:

print("The Test Statistic of", dftest[0], "is less than theCritical Value", key, "value of", value)

log\_wmt\_stock = np.log(wmt\_stock)

log\_moving\_avg = log\_wmt\_stock.rolling(12).mean()

log\_no\_trend = log\_wmt\_stock - log\_moving\_avg

test\_stationarity(log\_no\_trend.dropna())

test\_stat\_vs\_crit\_val()

# from statsmodels.tsa.seasonal import seasonal\_decompose

decomposition = seasonal\_decompose(wmt\_stock, freq=30)

trend = decomposition.trend

seasonal = decomposition.seasonal

residual = decomposition.resid

plt.figure(num=None, figsize=(8, 10), dpi=80, facecolor='w', edgecolor='k')

plt.subplot(411)

plt.plot(wmt\_stock, label='Original')

plt.legend(loc='best')

plt.subplot(412)

plt.plot(trend, label='Trend')

plt.legend(loc='best')

plt.subplot(413)

plt.plot(seasonal,label='Seasonality')

plt.legend(loc='best')

plt.subplot(414)

plt.plot(residual, label='Residuals')

plt.legend(loc='best')

plt.tight\_layout()

#from statsmodels.tsa.stattools import acf, pacf

acf\_wmt\_plot = acf(wmt\_stock.dropna(), nlags=20)

#Plot ACF:

plt.figure(figsize=(20, 4))

plt.subplot(121)

plt.plot(acf\_wmt\_plot)

plt.xticks(np.arange(21))

plt.axhline(y=0,linestyle='--',color='gray')

plt.axhline(y=-1.96/np.sqrt(len(acf\_wmt\_plot)),linestyle='--',color='gray')

plt.axhline(y=1.96/np.sqrt(len(acf\_wmt\_plot)),linestyle='--',color='gray')

for i in range(1,20):

plt.axvline(x=i,linestyle=':',color='gray')

plt.title('Autocorrelation Function')

pacf\_wmt\_plot = pacf(wmt\_stock.dropna(), nlags=20)

#Plot ACF:

plt.figure(figsize=(20, 4))

plt.subplot(121)

plt.plot(pacf\_wmt\_plot)

plt.xticks(np.arange(21))

plt.axhline(y=0,linestyle='--',color='gray')

plt.axhline(y=-1.96/np.sqrt(len(acf\_wmt\_plot)),linestyle='--',color='gray')

plt.axhline(y=1.96/np.sqrt(len(acf\_wmt\_plot)),linestyle='--',color='gray')

for i in range(1,20):

plt.axvline(x=i,linestyle=':',color='gray')

plt.title('Partial Autocorrelation Function')

def runArima(dataset, p, d, q, showRSS=False):

model = ARIMA(dataset.dropna(), order=(p, d, q))

try:

results\_ARIMA = model.fit(disp=-1)

except ValueError:

pass

except:

pass

if(showRSS):

plt.plot(dataset)

plt.plot(results\_ARIMA.fittedvalues, color='red')

x=pd.DataFrame(results\_ARIMA.fittedvalues)

x=x.join(wmt\_stock)

x['out']=(x.iloc[:,0]-x.iloc[:,1])\*\*2

loss=np.sqrt(x['out'].sum())

plt.title('RSS: %.4f'% loss)

#from statsmodels.tsa.stattools import acf, pacf

acf\_wmt\_plot = acf(results\_ARIMA.fittedvalues, nlags=20)

#Plot ACF:

plt.figure(figsize=(20, 4))

plt.subplot(121)

plt.plot(acf\_wmt\_plot)

plt.xticks(np.arange(21))

plt.axhline(y=0,linestyle='--',color='gray')

plt.axhline(y=-1.96/np.sqrt(len(acf\_wmt\_plot)),linestyle='--',color='gray')

plt.axhline(y=1.96/np.sqrt(len(acf\_wmt\_plot)),linestyle='--',color='gray')

for i in range(1,20):

plt.axvline(x=i,linestyle=':',color='gray')

plt.title('Autocorrelation Function')

print(f"p:{p} d:{d} q:{q}")

p = 0

d = 0

q = 0

runArima(wmt\_stock.dropna(),p,d,q)

d = 1

runArima(wmt\_stock.dropna(),p,d,q)

d = 2

runArima(wmt\_stock.dropna(),p,d,q)

q = 2

runArima(wmt\_stock.dropna(),p,d,q)

p = 2

d = 0

q = 0

runArima(wmt\_stock.dropna(),p,d,q)

p = 2

d = 2

q = 0

runArima(wmt\_stock.dropna(),p,d,q)

p = 2

d = 2

q = 2

runArima(wmt\_stock.dropna(),p,d,q)

pdq\_results = []

air\_pop = wmt\_stock.astype(float)

loss\_best = 1E16

best\_ints = [-1,-1,-1]

for p in range(4):

for d in range(3):

for q in range(3):

model = ARIMA(wmt\_stock.dropna(), order=(p, d, q))

try:

results\_ARIMA = model.fit(disp=-1)

except ValueError:

pass

except:

pass

plt.plot(wmt\_stock)

plt.plot(results\_ARIMA.fittedvalues, color='red')

x=pd.DataFrame(results\_ARIMA.fittedvalues)

x=x.join(wmt\_stock)

x['out']=(x.iloc[:,0]-x.iloc[:,1])\*\*2

loss=np.sqrt(x['out'].sum())

plt.title('RSS: %.4f'% loss)

pdq\_results.append(f"p:{p} d:{d} q:{q} p+q:{p+q} rss:{loss} aic:{results\_ARIMA.aic}")

if loss < loss\_best:

print(loss)

if (p > 1 and q <= 1) or (q > 1 and p <= 1) or (p <= 1 and q <= 1):

loss\_best = loss

best\_ints=[p,d,q]

plt.show()

print(p,d,q)

# print(pdq\_results)

print(len(pdq\_results))

for result in pdq\_results:

print(result)

print(loss\_best)

print(best\_ints)