## Introduction

In an interview for *Business Insider*, Stan Druckenmiller, former president of Duquesne Capital, described his success as a hedge fund manager, “I watched the stock market, how equities reacted to change in levels of economic activity and I could understand how price signals worked and how to forecast them.” [3] As Druckenmiller describes, when chosen correctly, signals are extremely beneficial in forecasting. In this case study, we use signals for frontier analysis for economic modeling.

**Background**

As of April 2018, the US stock market was $34 trillion- that was in comparison to the rest of the world’s $44 trillion capital [5]. Of those trillions of US dollars, 84% of all those stocks belong to the wealthiest 10% of American families [6]. Certainly, that leaves the other 90% of Americans wishing for a *Magic 8 Ball* (manufactured by Mattel, (MAT)) that could tell them which stocks to buy that will put them into the top 10% and provide them a share of that $34 trillion. Signal Frontier Analysis, when done well, and considering the correct variables, can provide just that “magic.” Signal Frontier analysis is an econometric technique which uses regression analysis to estimate stock prices over a period of time. Remarkably, signal forecasting has traces to Isaac Newton, who lost the equivalent of $2 million in an investment in the South Sea Company “bubble” after a rumor broke out that they had discovered large oil fields. People went crazy trying to invest in the company and Newton lost what would be millions of US dollars by today’s calculations. Of his loss, Newton said, “I can calculate the stars, but not the madness of men. [4]” What Newton was describing was a need for signal forecasting.

**Methods**

**Loading Stock Portfolio Data.** The methods used for this Signal Frontier Analysis are from the book, *Python for Data Analysis,* chapter 11. We chose to analyze the stock for Nike (NKE) for over four years worth of data. Using the ARIMA, we estimate the parameters autoregressive part (p), integrated part (d), and moving average parts (q). Following that, a grid search is conducted for parameters. Finally, we select the predictive parameters.

To begin the process, we use pandas-datareader to access *Yahoo!Finance* to download 4 years of financial data from 3/1/2015 to 3/1/2019.

stock\_list =['NKE']  
stock\_price = pd.DataFrame({stock: web.get\_data\_yahoo(stock, '3/1/2015', '3/1/2019')['Adj Close'] for stock in stock\_list})  
print (stock\_price.head())  
print (stock\_price.tail())  
print (stock\_price.shape)

NKE  
Date   
2015-03-02 46.636871  
2015-03-03 46.268517  
2015-03-04 46.055981  
2015-03-05 46.499928  
2015-03-06 45.767895  
 NKE  
Date   
2019-02-25 84.951431  
2019-02-26 85.579819  
2019-02-27 85.948868  
2019-02-28 85.510002  
2019-03-01 87.160004  
(1008, 1)  
 NKE  
Date   
2015-03-02 46.636871  
2015-03-03 46.268517  
2015-03-04 46.055981  
2015-03-05 46.499928  
2015-03-06 45.767895  
 NKE  
Date   
2019-02-25 84.951431  
2019-02-26 85.579819  
2019-02-27 85.948868  
2019-02-28 85.510002  
2019-03-01 87.160004  
(1008, 1)

**Exploratory Data Analysis.** After loading the data, we evaluate it to ensure that there are no missing values in the data (fig 1). We do so using the *msno* package. 

Fig 1. *Msno plot to reveal missing values*

With the data on-hand, the initial analysis of trends, correlations, variations, and outliers begins and enables us to better focus our ongoing analysis.

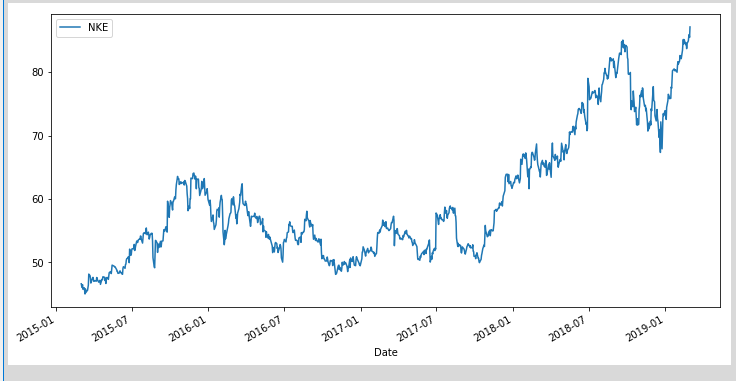
We construct two plots to visualize the Nike data. First, using a time-series graph, we view the variation in Nike stock across the four years of observations. Next, using the autocorrelation plot, we observe for randomness. Finding nothing significant with either plot, we move onto the next step.

Fig 2. *Nike time series plot*

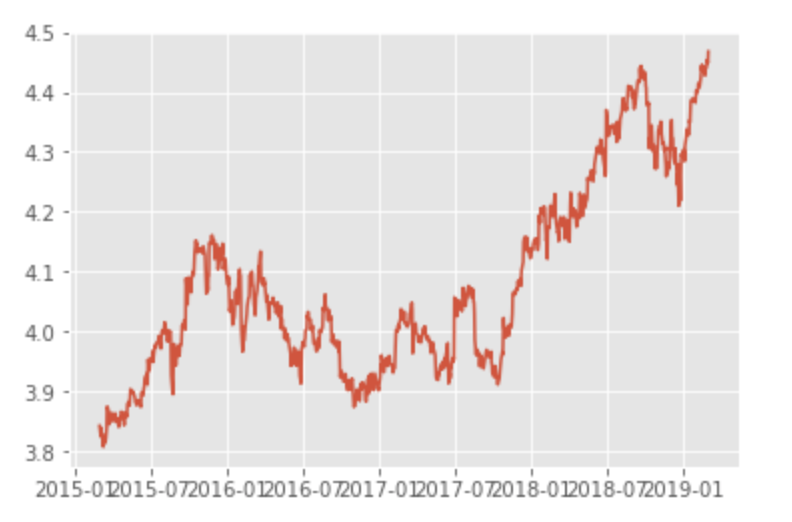
Next, we transformed the Nike stock price by taking the logarithmic format because we are essentially dealing with returns and we want to make sure our data is in the right format. Once we convert and plot the price and log format, we can see that it gives us an accurate estimate, “in terms of returns,” as it is adjusting for volatility.

Fig. 3 *Nike logarithmic format*

Before we can run the ARIMA model, we ensure our data is stationary; because if we don’t have a constant mean variance in all the correlations, we’re not going to be able to make an accurate forecast. To do this, we will use both the autocorrelation (ACF) and the partial autocorrelation (PACF) plots.

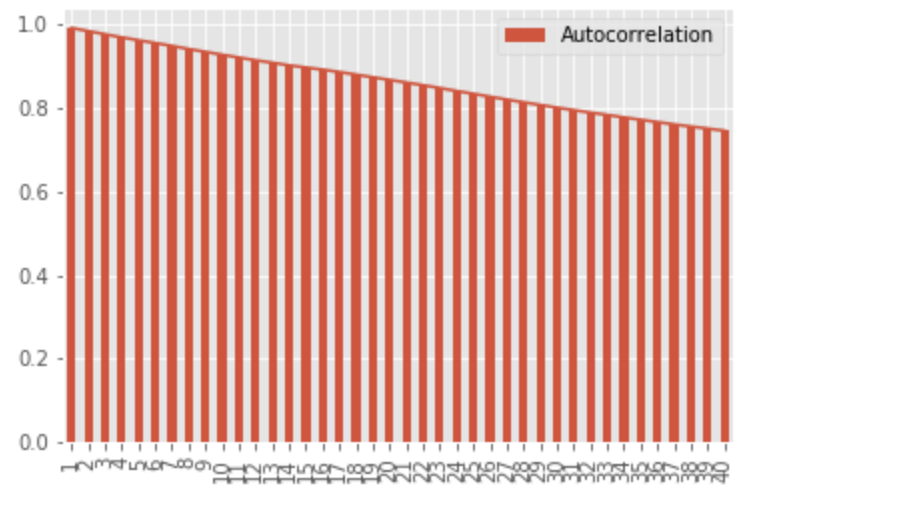


Fig 4. *Nike Autocorrelation (ACF) plot*

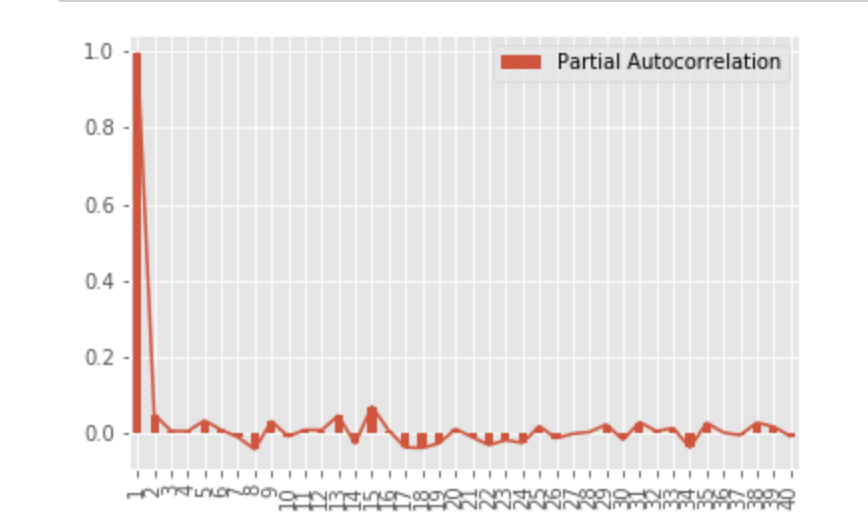
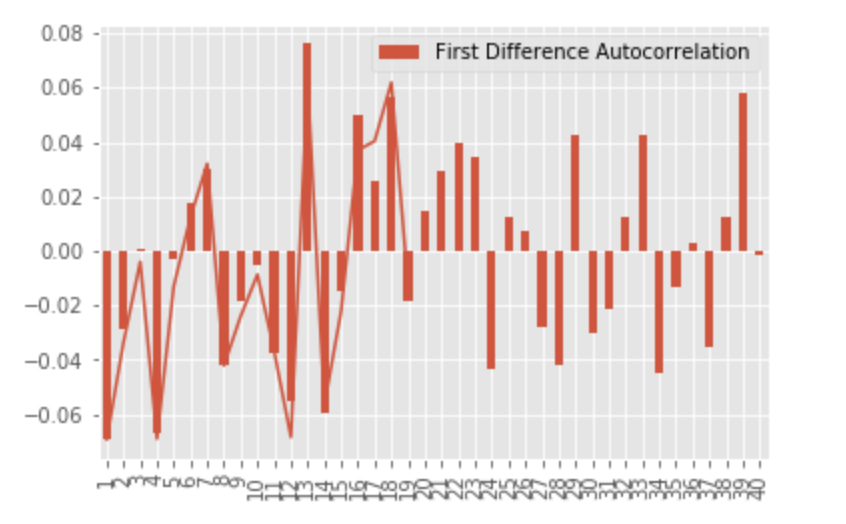


Fig 5. *Nike Partial Autocorrelation (PACF) plot*

After running both ACF and PACF plots, we see from the autocorrelation plot we have a steady decrease downwards in our correlation for each lag and from the partial autocorrelation function, we see a sudden drop in correlation for the lags. These plots indicate that we have AR one process in our time series; a stationary AR one process which coincides with the way we want our time series to essentially be structured. After screening our data, we run a more formal test, the Dickey-Fuller test.



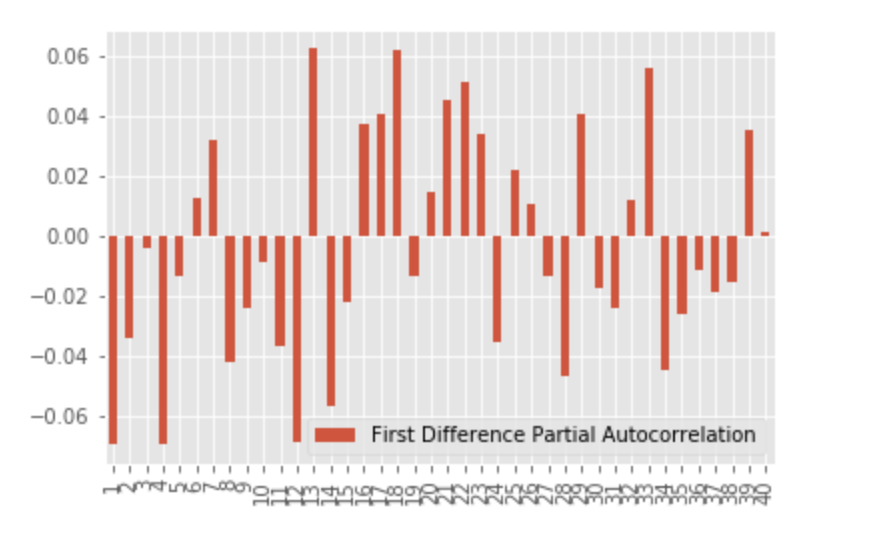


Fig. 6. *Nike Dickey-Fuller test*

From the Dickey-Fuller test, we see that the first difference autocorrelation has the up-and-down zigzag pattern that we expect in a stationary time series so once we execute the plot of the first difference autocorrelation, it coincides with what we expect. We also anticipated seeing this similar pattern for the partial autocorrelation that had also been first differenced.

Upon fully preparing our data, we run the ARIMA model, with an arrival configuration of 0,1,0. This configuration was determined from our time series data. Here we see a particular upward trend. With our ARIMA model, we also forecasted for the next 5 time periods outwards.

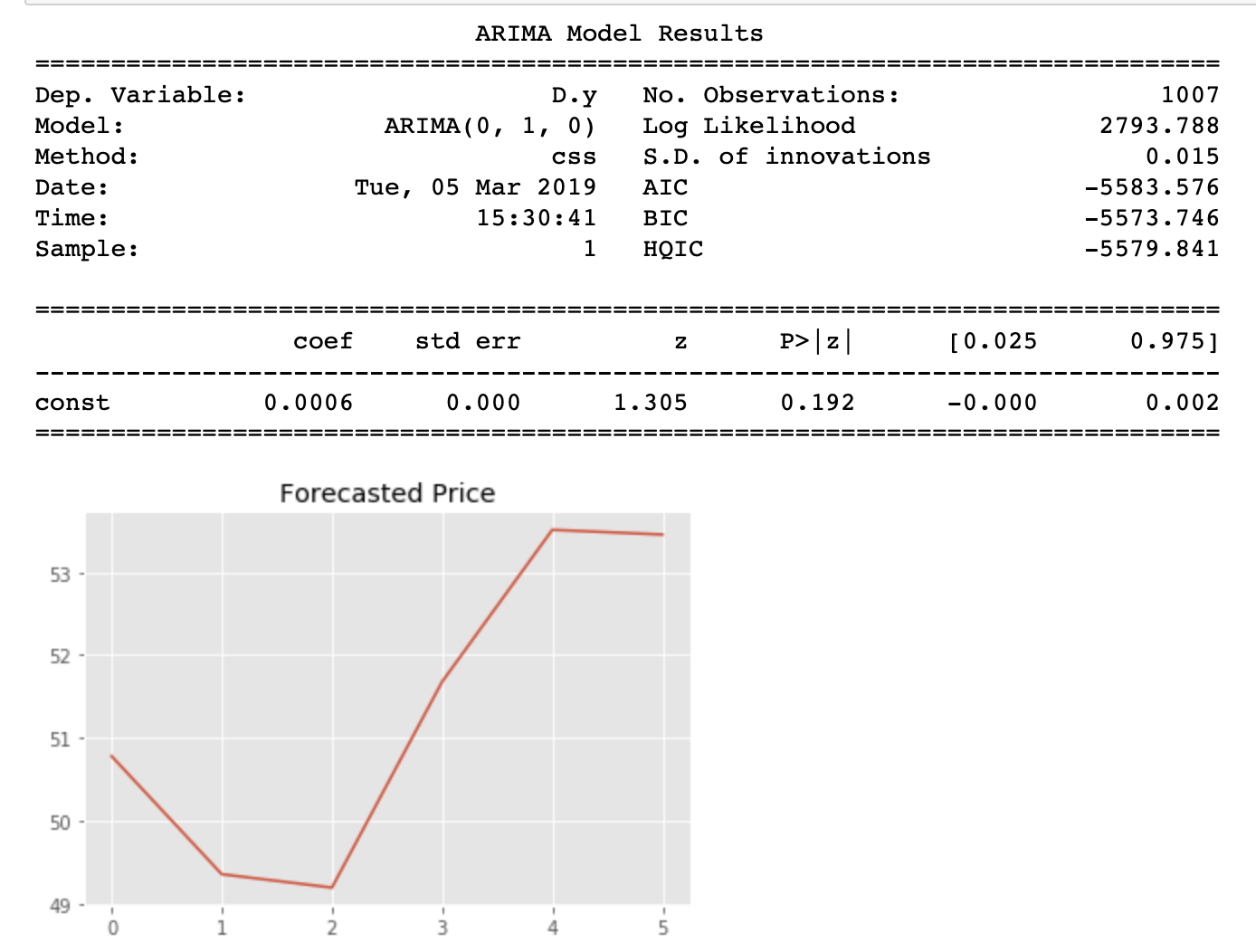


Fig. 7. *ARIMA model with a forecast for the next 5 time periods*

**Results**

From the model and our predictions, we see an upward starting at the 2nd time period until the 4th time period and then what appears to be a steady decrease when approaching time period 5.

**Conclusions**

A cursory search of Google News for “predict stock prices” will produce 8 stories published in the last 24 hours, alone, with taglines referencing predicting stock prices. Stories about predicting Target’s future stock prices [7], predicting whether or OneSpan Inc’s stocks are “on hold” [8], or if CEO characteristics predict stock prices [9]. These stories are frought with one desire, that some “magic”that will predict stocks for them. A cursory read of the articles reveals many methods and opinions to be predict stock prices, but in this case study, we’ve demonstrated one reasonable way to do so: Signal Frontier Analysis. While actual future results are yet to determine the reliability of the model, our test cases were certainly successul.

**Future Work**

Instead of using signals simply for economic modeling, we can leverage these signals for the purpose of building an investment portfolio that of which is diversified and optimized for efficiency as it relates to occupying “the ‘efficient’ parts of the risk-return spectrum” referred to as the “efficient frontier” [10]. Putting together assets that are inversely correlated can construct a portfolio that yields a lower risk for the consumer. Warren Buffett stated, “risk comes from not knowing what you are doing” [10]. One method of reducing risks is looking at return and risk of a portfolio with random weights and identify the best and/or most efficient weights. Using Python’s *scipy* package, we can use the optimizer function to retrieve optimal weights for different targeted returns. With our random weights between 0 and 1, the *scipy* optimizer can help us find the best allocation. With a plot of risk vs. returns, we can see our efficient frontier line where the best allocation is given. The Sharpe Ratio is then calculated, which is a return/risk ratio. This ratio provides the consumer with his/her portfolio performance [10].

**References**

1. McKinney, W.: Python for Data Analysis. OReilly Media, Inc, USA, Sebastopol (2012).
2. Shitoshna: Signal Processing in Finance, <https://sites.tufts.edu/eeseniordesignhandbook/2015/signal-processing-in-finance/>.
3. Lopez, L.: Stan Druckenmiller Gave A Startlingly Blunt Reason For Why Hedge Fund Managers Don't Like Bernanke, <https://www.businessinsider.com/druckenmiller-not-made-for-todays-market-2013-6>.
4. Lee, B.: From Newton to the Financial Crisis: In search of connections among physics, communication technologies and investment.
5. Surz, R.: U.S. Stock Market Is Biggest & Most Expensive In World, But U.S. Economy Is Not The Most Productive, <https://m.nasdaq.com/article/us-stock-market-is-biggest--most-expensive-in-world-but-us-economy-is-not-the-most-productive-cm942558>.
6. Cohen, P.: We All Have a Stake in the Stock Market, Right? Guess Again, <https://www.nytimes.com/2018/02/08/business/economy/stocks-economy.html>.
7. Butler, D.: I Like Target Stock at Current Prices, <https://realmoney.thestreet.com/investing/i-like-target-stock-at-current-prices-14886939>.
8. Amelie Mason, <https://postanalyst.com/2019/03/05/onespan-inc-ospn-chase-insider-institutional-changes-not-stock-prices/>.
9. Rasmussen, D., Li, H.: The MBA Myth and the Cult of the CEO, <https://www.institutionalinvestor.com/article/b1db3jy3201d38/The-MBA-Myth-and-the-Cult-of-the-CEO>.
10. Li, K. (2018). “Portfolio Optimization for Minimum Risk with Scipy--Efficient Frontier Explained”. *Medium: Towards Data Science*. Published May 4, 2018. <https://towardsdatascience.com/efficient-frontier-optimize-portfolio-with-scipy-57456428323e>

**APPENDIX**

*#!/usr/bin/env python*

*# coding: utf-8*

*# In[1]:*

**import** pandas **as** pd

**import** numpy **as** np

**import** math

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

**import** statsmodels.tsa.stattools **as** ts

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

**from** numpy.random **import** randn

**import** statsmodels.api **as** sm

**import** pandas\_datareader.data **as** web

**from** datetime **import** datetime

**from** dateutil.parser **import** parse

**from** collections **import** defaultdict

**from** pandas **import** Series, DataFrame

**import** random

**import** matplotlib.pyplot **as** plt

get\_ipython().run\_line\_magic('matplotlib', 'inline')

pd.options.display.max\_rows = 12

np.set\_printoptions(precision=4, suppress=True)

plt.rc('figure', figsize=(12,6))

**import** warnings

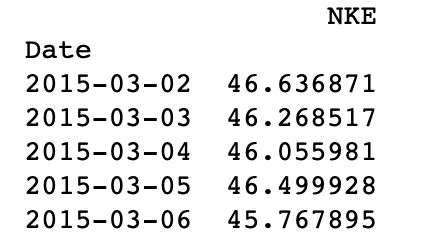
warnings.filterwarnings('ignore')

*# In[2]:*

stock\_list =['NKE']

price = pd.DataFrame({stock: web.get\_data\_yahoo(stock, '3/1/2015', '3/1/2019')['Adj Close'] **for** stock **in** stock\_list})

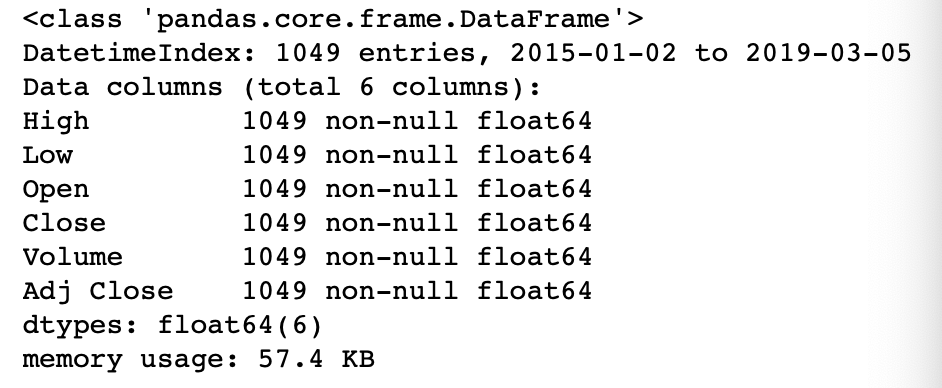
**print** (price.head())



*# In[3]:*

data = web.get\_data\_yahoo('NKE', '2015-01-01')

data.info()

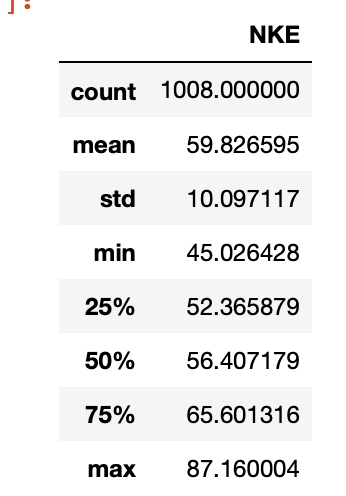


*# In[4]:*

price.ix['2015':'2019'].plot()

*# In[5]:*

price.describe()



*# In[6]:*

*#difference*

plt.figure(figsize=(15, 5))

plt.style.use('ggplot')

price\_log = np.log(price)

plt.plot(price\_log, color='b')

*# In[7]:*

lnprice=np.log(price)

lnprice

plt.plot(lnprice)

plt.show()

*# In[37]:*

acf\_1 = acf(lnprice)[1:1049]

test\_df = pd.DataFrame([acf\_1]).T

test\_df.columns = ['Autocorrelation']

test\_df.index += 1

test\_df.plot(kind='bar')

plt.plot(acf\_1)

plt.show()

*# In[38]:*

pacf\_1 = pacf(lnprice)[1:1049]

test\_df = pd.DataFrame([pacf\_1]).T

test\_df.columns = ['Partial Autocorrelation']

test\_df.index += 1

test\_df.plot(kind='bar')

plt.plot(pacf\_1)

plt.show()

*# In[39]:*

**from** statsmodels.tsa.stattools **import** adfuller

**def** test\_stationarity(timeseries):

result = ts.adfuller(lnprice.iloc[:,0].values, 1)

result

lnprice\_diff=lnprice-lnprice.shift()

diff=lnprice\_diff.dropna()

acf\_1\_diff = acf(diff)[1:1049]

test\_df = pd.DataFrame([acf\_1\_diff]).T

test\_df.columns = ['First Difference Autocorrelation']

test\_df.index += 1

test\_df.plot(kind='bar')

pacf\_1\_diff = pacf(diff)[1:20]

plt.plot(pacf\_1\_diff)

plt.show()

test\_stationarity(plt.show())

*# In[40]:*

**def** first\_difference(timeseries):

result = ts.adfuller(lnprice.iloc[:,0].values, 1)

result

lnprice\_diff=lnprice-lnprice.shift()

diff=lnprice\_diff.dropna()

pacf\_1\_diff = pacf(diff)[1:1049]

test\_df = pd.DataFrame([pacf\_1\_diff]).T

test\_df.columns = ['First Difference Partial Autocorrelation']

test\_df.index += 1

test\_df.plot(kind='bar')

plt.show()

first\_difference(plt.show())

*# In[41]:*

*#Using the (0,1,0) configuration*

price\_matrix=lnprice.as\_matrix()

model = ARIMA(price\_matrix, order=(0,1,0))

model\_fit = model.fit(disp=0)

**print**(model\_fit.summary())

predictions=model\_fit.predict(122, 127, typ='levels')

predictions

predictionsadjusted=np.exp(predictions)

predictionsadjusted

plt.plot(predictionsadjusted)

plt.title('Forecasted Price')

plt.show()

*# In[42]:*

predictions=model\_fit.predict(122, 127, typ='levels')

predictions



*# In[43]:*

predictionsadjusted=np.exp(predictions)

predictionsadjusted

