Forecasting The Interest Level of The Playstation 5

Shail Patel
CPS803
Ryerson University
Toronto, Canada
s14patel@gmail.com

***Abstract*— Looking at the interest of the unreleased Playstation 5 (PS5) through the forecasting of inherited interest of the previous console PS4**

Keyword: Forecasting, Time Series, Marketing

I. INTRODUCTION

In the 21st century Sony's biggest profit has turned into the Playstation. With nearly all products proving to be a deficit, it is important for Sony to keep striving in the gaming industry. Sony releases a new gaming console every 6 years; with it being nearly 6 years since the PlayStation 4 was released, a new console was announced last year. The PlayStation 5 is scheduled to come out in the winter of 2020.

PlayStation 4: (United States)

Month	
2019-08-01	53
2019-09-01	100
2019-10-01	35
2019-11-01	48
2019-12-01	53

The interest and hype around releasing of new console is what really spreads the word. Through groups of people buying it, the need to fit in and peer-pressure increases the sales more than any other product out there.

The 2013 "Perfect day" commercial by Playstation which aired during a football game was ranked as one of the most-liked commercials of all time. I drove sales to outstanding amounts, beating the competition by a large margin. The commercial also increased the interest which increased searches of playstation across the Web. This lets us to believe that interest and searches across the web plays a huge role on sales for the product.

As of December 2019, Playstation has sold over 100 million consoles at a cost of $400CAD. The fan base and interest in the product does not decline in fact it can be inherited from the previous model. In this project we look at the forecast of time series data for the PS4 and how it will be projected onto the PS5.

II. METHOD

A. Selecting and finding data: Time series

The idea for predicting the PS5 sales was to look at the PS4 search data over the past 6 years and use that as a a frame for predict PS5 interest and potential sales.

Selecting data was difficult. The goal set in mind was to somehow predict the success, and thus the sales, of the PS4. There was no accurate sales data for monthly sales of the PS4 rather it was more vague yearly data and the specific monthly sales data came at cost of about $50. Rather I looked for search data on google which can be directly correlated to the interest in the product resulting in sales.

To prove this, I looked at a very successful commercial "Perfect Day" which was aired in the United States during football and it drastically increased the interest; consequently searches on google were increased as well.

The data was retrieved through Google stats which lets you search any keyword and Google is able to produce a chart for the search interest over a select time frame. It is important to understand that the the points are relative to the highest search - that is what the data is for: Interest in PS4 relative to the highest searches period.

Data I retrieved was on a monthly period from January 2013 to December 2019. Data came as a CSV file with features like year, year-month-day, state, interest. I removed all features except year-month-day

Month	PlayStation 4: (United States)
2013-01	3
2013-02	34
2013-03	12
2013-04	7
2013-05	13
2013-06	41
2013-07	16
2013-08	18
2013-09	19
2013-10	26
2013-11	86
2013-12	54
2014-01	29
2014-02	29
2014-03	26
2014-04	22
2014-05	22
2014-06	26
2014-07	24

and interest(PlayStation 4: United States) level.

B. Getting stared with Machine learning

The time series data is first processed by removing unnecessary features from the data directly by directly deleting them through excel file.

The process to forecast data uses machine learning specifically the Scikit library in python. The code is processed through Jupiter note book. Anaconda is required.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from numpy import genfromtxt
from sklearn import datasets, linear_model
from matplotlib import pyplot
from sklearn.metrics import mean_squared_error, r2_score
from statsmodels.tsa.stattools import adfuller
```

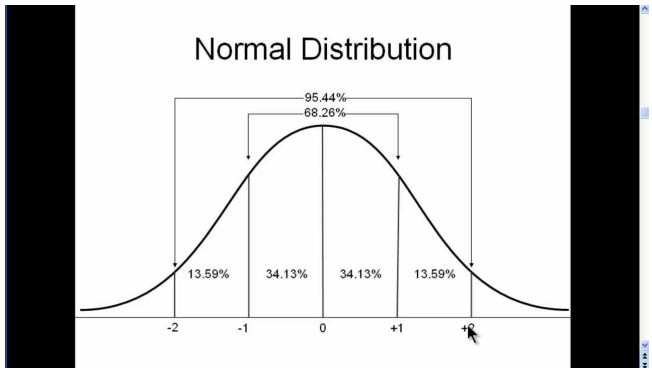
These are all the imports required. Numpy and pandas are the most basic for machine learning. The others will describes throughout the paper.

C. Deciding whether data is stationary or non stationary

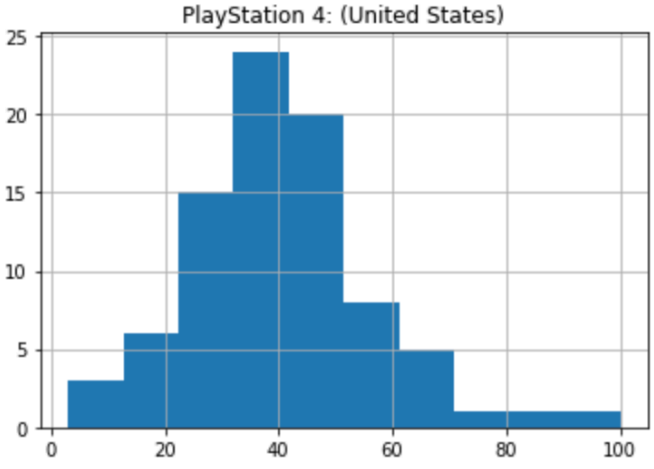
Time series data is different from other data as one of the axes is dedicated to time - the x-axis. When first starting off the data is loaded using a CSV function and then parsed using another function so pandas can read the specific months. This is built in there pandas is able to look at the foment of the year-month-date to plot a graph accordingly.

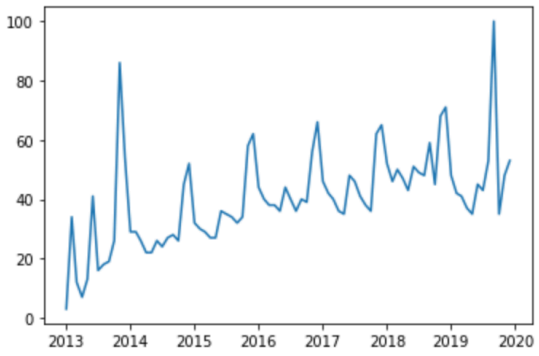
To begin forecasting, we first need stationary data. Stationary data defined as the data which the joint probably distribution, also the Gaussian distribution does not change over time; the meaning and variance do not change over time.

So this idea suggests that if our data looks close to a Gaussian distribution then it could be stationary.



So First is ta picture of a Guassian Distribution(normal distriution) and bellow that is our PS4 histogram. This looks like similar, it suggests that it might be a stationary data. However looking at the plotted graph, our eyes may suggest it is not. Bellow is a function of graphed data





When our time series is plotted, it appears not to be non-stationary. However, our histogram suggests otherwise. Further direction needs to be performed to answer the question. We use the Dickey-fuller test.

Dickey-fuller test involves proving the null hypothesis true; if a unit of present in a model then it is proven true. The alternative hypothesis is proven if the bull hypothesis is false, which suggests the data is stationary or trend-stationary; thus, no unit root present. For a mathematical explanation:

Suppose we a time series:

$$y_t = pyt\text{-}1 + \varepsilon_t$$

Where y_t is value at time t, ε_t is the error term.

To calculate yt we need yt-1.:

$$y_t\text{-}1 = py_t\text{-}2 + \varepsilon\ t\text{-}1$$

If we do that for all, the equations is:

$$y_t = p^n y_t\text{-}n + \Sigma\varepsilon_t\text{-}i * p\text{^}i$$

If p=1 then it the outcome prediction will be equal to y_{t-1} and $\Sigma\varepsilon_t$-i, the sum of all errors will increase over time meaning that the variance will increase and thus the data

```
X = df['PlayStation 4: (United States)'].values
result = adfuller(X)

print('Adf Statistic: %s' %result[0])
print('p-value: %f' %result[1])
print('critical Values:' )
for key, value in result[4].items():
    print('\t%s: %.4f' %(key,value))
```

is non-stationary.

Going to back to Juptyer, we imported a dickey-fuller method called "adfuller"- augmented dickey fuller.

We basically put in the values of the dataset and run it through the adfuller method. Then we can print "ADF

```
Adf Statistic: -1.1861253042880369
p-value: 0.679536
critical Values:
        1%: -3.5246
        5%: -2.9026
        10%: -2.5887
```

statistics", P-value and the critical values of the distribution.

These are the results:

Now the augmented dickey fuller test uses distribution and measures trend, such as seasonal trends. The closer the p-value to 1 the worse the distribution and the general rule is if p > 0.05 then it is non-stationary. Also if the critical values are lower than the ADF statistic values then it is non-stationary.

We can conclude is that even though the history gram looked normally distributed, further investigation using dickey fuller method showed that the data set is actually non-stationary. Next step would be to make it stationary so we can find the predicting values.

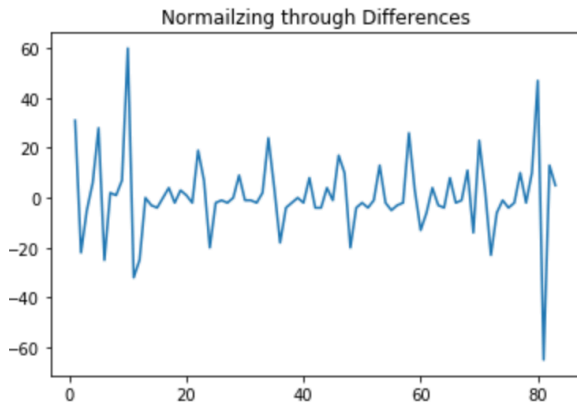
C. Implementing methods to acquire stationary data

Now we know that the data is non-stationary we need to normalize the data. We are going to apply various methods to get the best results and acquire stationary data. First we try differencing.

Differencing refers to subtracting from the previous value in the fame feature. Mathematically:

```
df["diff_values"] = df["PlayStation 4: (United States)"].diff()
```

$$y_t = y_t - y_{t-1}$$



Differencing is one way to normalize the data. It is used to get rid of varying mean. Using this idea, we use it

```
Adf Statistic: -9.5333413196743
p-value: 0.000000
critical Values:
        1%: -3.5246
        5%: -2.9026
        10%: -2.5887
```

on our data frame and assign it to a new feature "diff_values" and call the .diff() function on our original data to get the difference value.

This gives us the graph:

Visually, this graph looks stationary. We can test it through dickey fuller test. Here are the values we get when putting differenced data in adfuller() method.

The p-value = 0. Meaning there is a unit root only at 0, which means the data is stationary. Also the Ads Statistic value is lower than all Critical values. Thus the

```
moving_avg["diff_values"] = moving_avg.rolling(window=5).mean()
```

data is stationary through differencing.

Now the data being stationary we can find the moving average. The moving average is used to find the average relating to the the previous values. These

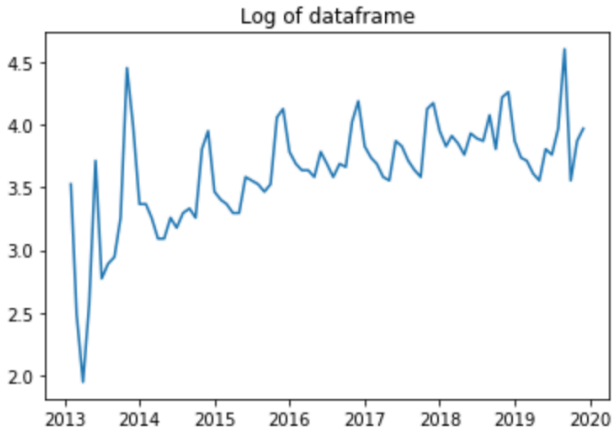
specified number of values is called lag and we use that to find the next average.

We use the method .rolling and specify window=5 which is lag. And we get the moving average line (BLUE) compared with the difference of the data.

This idea was soon found to be redundant and we will discuss it later on.

Since we have stationary data and a moving average to help us creating a model (Section E)

```
np.log(df_log["PlayStation 4: (United States)"])
```



D. Second implementation of acquiring stationary data- Log-difference method.

The second way of acquiring stationary data is the Log-difference method. Where first we normalize the data by taking the Log of it.

```
Adf Statistic: -2.357677545744799
p-value: 0.154040
critical Values:
        1%: -3.5260
        5%: -2.9032
        10%: -2.5890
```

We use this method to log our original data frame and store it in another data frame called *df_log*.

This is the normalized graph we get after taking log of

```
df_log_diff = df_log.copy()
df_log_diff["PlayStation 4: (United States)"].diff()
```

every value.

We can check again if this has made the data

```
Adf Statistic: -4.764755456840321
p-value: 0.000063
critical Values:
        1%: -3.5274
        5%: -2.9038
        10%: -2.5893
```

stationary using the dickey fuller method

p>0.05 and adf > critical values

This suggests that it is not stationary even after normalizing. Next step would be to take thee difference like the last method. Since taking log only scales down the values, we can take difference make it stationary.

After taking the difference of the data we use the Dickey fuller test to check if *p=0*

P-value =~ 0 and ADF < Critical values

The log data has not become stationary because p is very close to 0 so there is unit root=0, which we consider no unit root.

E. *ARIMA Model Explained*

After using two methods of acquiring stationary data we are now able to create model for both. We will be using the 'Auto Regression Integrated Moving Average' (ARIMA) model. It is a combination of Auto Regression and Moving Average.

Auto Regression model depends linearly on its own lag is defined as:

$$y_t = a + iy_{t-1} + iy_{t-2} + \ldots + iy_{t-n} + \varepsilon_t$$

$$y_t = a + \Sigma\, i_n y_{t-1} + \varepsilon_t$$

Where a is a bias, ε_t is the white noise, *I is some constant.*

Moving average is for univariate modelling, it relies linearly on mean of previous data, specifically lagged forecast errors(white noise). Defined as:

$$y_t = a + \varepsilon_t + i_t\, \varepsilon_{t-1} + i_2\, \varepsilon_{t-2} + \ldots + i_n \varepsilon_{t-n}$$

Where a is the means, ε_t is the white noise, *i is some constant.*

When these two are combined you are able to check for seasonal trends as well as giving extra weights after curtain lags.

This model is able to look at the past values of time series and forecast future outcomes. ARIMA model requires stationary data, in fact all time series models do.

ARIMA model using three terms to find its significant values. Lets call them (p,*d,q)*

q - number of lags of the moving average model

p - number of lags of auto regression

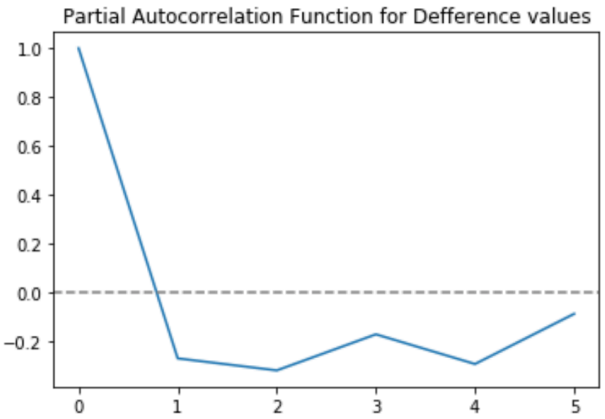
d - number of differencing required to make data stationary

d = 1 for both log-differencing and differencing-moving average. However acquiring *q* and *p* requires additional work.

```
#Finding acf and pacf to get q and p value to get ARIMA model to work
from statsmodels.tsa.stattools import acf, pacf

acf = acf(df["diff_values"], nlags=5)
pacf = pacf(df["diff_values"], nlags=5, method='ols')


plt.plot(acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.title('Autocorrelation Function')
plt.show()

plt.plot(pacf)
plt.title('Partial Autocorrelation Function')
plt.axhline(y=0,linestyle='--',color='gray')
plt.show()
plt.tight_layout()
```

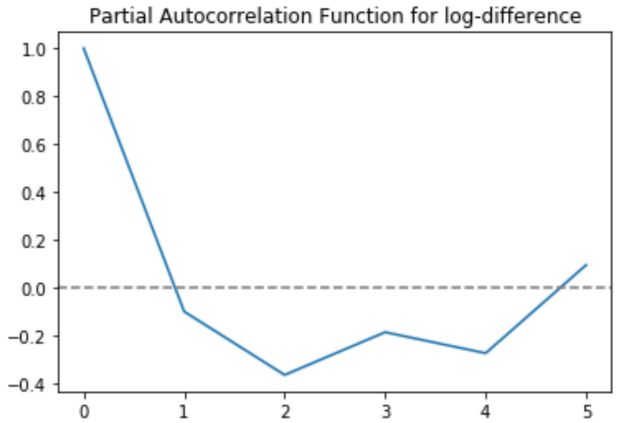
F. *Using PACF to find p*

We need to find p value for the number of lags in the auto regression model. We do this by plotting the Partial Autocorrelation(PACF) and inspecting for roots.



PACF gives a correlation of its series and the lagged values. We look at the roots for a significant value to give *p.*

We can apply this model to both of our stationary



methods. First we find *p* of our first difference method.

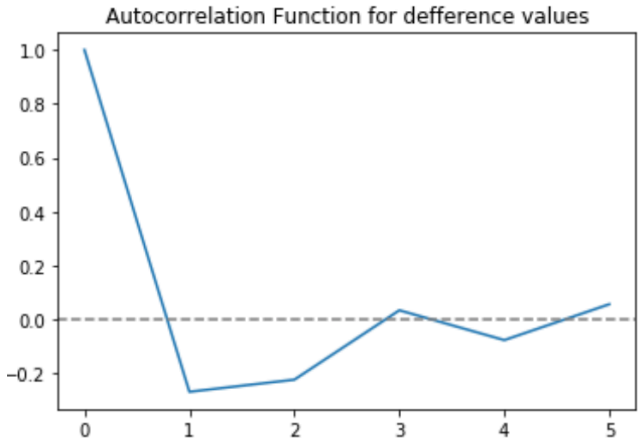
As a reference, a picture of the code is above. We call the methods *pacf()* from *statsmodels.tsa.stattools* and pass our difference values. Then we are able to plot the graph to acquire our root.

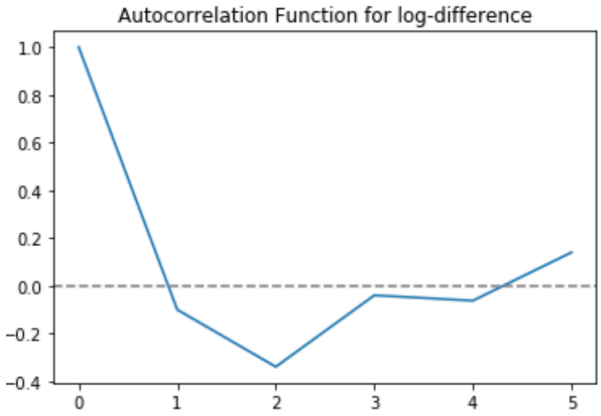
Here we see that the root is at approximately 0.8. But our ARIMA does not take in float values so we assign *p=1*.

Now for the log-difference model.

```
#Finding acf and pacf to get q and p value to get ARIMA model to work
from statsmodels.tsa.stattools import acf, pacf

acf = acf(df["diff_values"], nlags=5)
pacf = pacf(df["diff_values"], nlags=5, method='ols')


plt.plot(acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.title('Autocorrelation Function')
plt.show()

plt.plot(pacf)
plt.title('Partial Autocorrelation Function')
plt.axhline(y=0,linestyle='--',color='gray')
plt.show()
plt.tight_layout()
```

To no surprise, they are almost the name *p = 1*





G. *Using ACF to find q*

To find *q,* our value for MA model, we use the autocorrelation function. It gives the correlation between

y_i and y_{i+1}

MA is the error of the lagged forecast. ACF gives us values of how many MA terms are required to remove any autocorrelation.

```
from statsmodels.tsa.arima_model import ARIMA

df_diff_values_indexed = df.set_index(['Month'])

AR_model = ARIMA(index_avg, order=(1,1,1))
model_fit = AR_model.fit()

plt.plot(indexedDataset["diff_values"])
plt.plot(model_fit.fittedvalues, color='red')
plt.show()
```

First let us look at the difference methods.

The code method we use is *acf()* on our *diff_values.* Then we plot the graph to get our root.

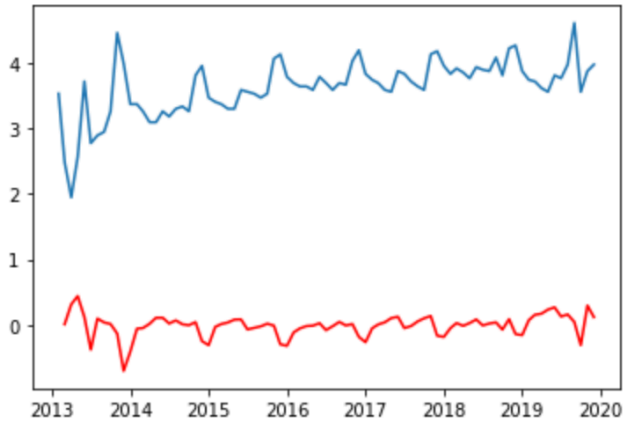
If we look at our root, it is approximately 0.8 again but ARIMA does not take float values so *q = 1*.

Now lets take a look at log-difference and find our root.

Not surprisingly, the values are the same for both methods q = *1*.

```
#geting the ARIMA model for Log difference

from statsmodels.tsa.arima_model import ARIMA


df_log['Month'] = pd.to_datetime(df['Month'])
log_diff_indexed = df_log.set_index(['Month'])
print(df_log_indexed)

AR_model = ARIMA(log_diff_indexed, order=(1,1,1))
model_fit = AR_model.fit(disp=-1)

plt.plot(df_log_indexed)
plt.plot(model_fit.fittedvalues, color='red')
plt.show()
```

H. *Modelling the ARIMA*



Now that we have found both our values of q and p for moth of our MA and AR parameters we can train the ARIMA model.

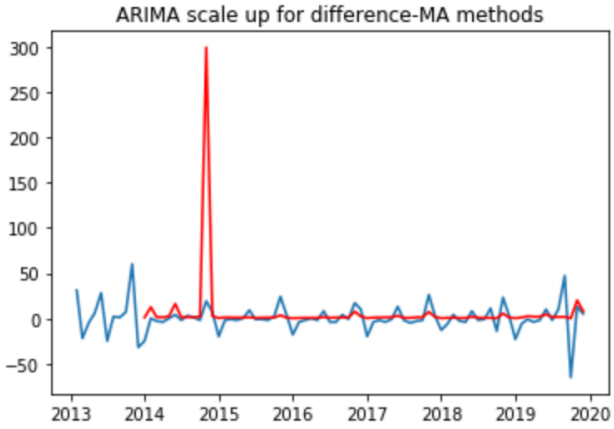
First lets take a look at the difference method.

Above is the picture of code used. The *ARIMA()* method is called for *index_avg* which is the date-parsed data frame for different and moving average.

The idea of implementing difference and then moving average and then ARIMA which already contains a moving average is a test of how much outcomes can differ or what the

```
# getting arima model to scale with the original data
# MOST IS NOT MY CODE

prediction_ARIMA_log = pd.Series(df["PlayStation 4: (United States)"].iloc[0], index=df.index)
prediction_ARIMA_log.add(prediction_ARIMA_cumsum, fill_value=0)

prediction_ARIMA = np.exp(prediction_ARIMA_series)

plt.plot(indexedDataset['diff_values'])
plt.plot(prediction_ARIMA, color='red')
```

predictions might be. Here is the graph:

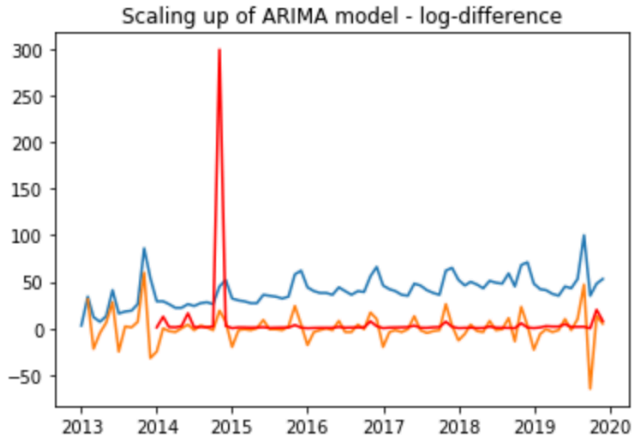


The red line is the ARIMA trained and the blue line is the data frame. The reason it looks flipped is because we took the difference. But this is corrected later on.

For the log-difference method we have

Code above is for log- difference data. Where *ARIMA()* is called on *log_diff_indexed* which is the date-parsed log-difference data set with values of *q=1, d=1, p=1.*

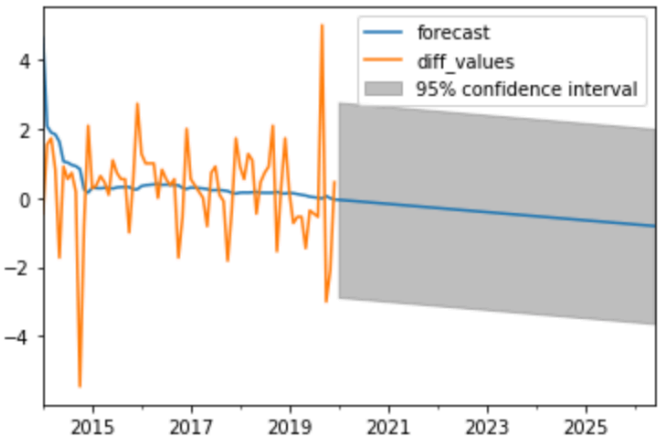
```
#NOT MY CODE
prediction_ARIMA_log = pd.Series(df["PlayStation 4: (United States)"
prediction_ARIMA_log.add(prediction_ARIMA_cumsum, fill_value=0)

prediction_ARIMA = np.exp(prediction_ARIMA_series)

print(prediction_ARIMA)
plt.plot(indexedDataset)
plt.plot(prediction_ARIMA, color='red')
```



This graph seems unusual primarily because the the blue line is the log dataset and the red line is the ARIMA model trying using the log-difference data set. It will correct itself later on.

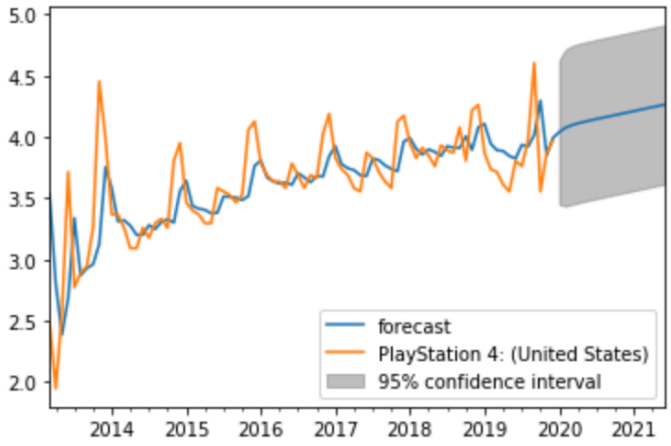
I. *Scaling Back*

Both methods used ways to make data stationary. After training the ARIMA model in a stationary environment, now



we need to scale it back up to the original time series data frame.

With the first methods we differenced it and then found the



moving average.

We use this piece of code to scale it back up. The *cumsum0* method gives the cumulative sum of the values to scale back up.

That is graph we get for scaling it. Red is the ARIMA. There is a huge anomaly in the middle but the other half, al thought difficult to see, seems to be a good fit.

For the second method we performed a log transformation to normalize the data then use differences to make it stationary.

We first used *cumsum()* to get the values before the difference and then perform a *np.exp()* on the ARIMA to scale up from the log.

Red is the ARIMA and orange is just the log dataframe.

J. *Prediction s*

Now that we have trained a model for both methods and scaled it back to the original data frame we are able to see the predictions.

For the first method our graphs looks like:

The prediction suggests that the interest on the playstation interest level will be going down.

For the second method:

It suggests that the interest levels will be going up. Both graphs show a different forecast. In the conclusion, I suggest why one is better than the other.

IV. Conclusion

The idea of this project was to show the interest of PS5 through PS4. Since the data received was relative to the highest interest rate, we can say that the PS5 interest rate will be relative to the previous console and so predict the interest rate for PS4 will be inherited by PS5.

Using both methods to make data stationary, the second method of log-difference gives a more reasonable and trust worthy forecast of the time series. As in the first methods we took the moving average and then applied ARIMA which already includes a moving average function it may have caused a bias change in the values.

The log-difference was a clean method which normalized the data and differenced it to make it stationary then trained the model to get a positive forecast.

The log-difference ARIMA model concludes with PS5 continuing to grow in interest through inheriting the PS4 interest and building more.

References

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first ...”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

1. “Time Series Analysis. James D. Hamilton Princeton University Press.” *Wilmott* 2003, no. 6 (2003): 92–92. https://doi.org/10.1002/wilm.42820030622.
2. Gagniuc, Paul A. (2017). Markov Chains: From Theory to Implementation and Experimentation. USA, NJ: John Wiley & Sons. pp. 1–256. ISBN 978-1-119-38755-8.
3. Farlie, D. J., and P. Whittle. “Prediction and Regulation by Linear Least-Square Methods.” *Or* 15, no. 4 (1964): 410. https://doi.org/10.2307/3007129.
4. “Forecasting: Principles and Practice.” 6.2 Moving averages. Accessed December 17, 2019. https://otexts.com/fpp2/moving-averages.html.

5. Singh, Aishwarya. "A Gentle Introduction to Handling a Non-Stationary Time Series in Python." Analytics Vidhya, September 6, 2019. https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/.
6. Sangarshanan. "Sangarshanan/Time-Series-Forecasting." GitHub. Accessed December 17, 2019. https://github.com/Sangarshanan/Time-Series-Forecasting/blob/master/ARIMA.ipynb
7. Tan, Swee Chuan, and Jess Pei San Lau. "Time Series Clustering: A Superior Alternative for Market Basket Analysis." SpringerLink. Springer, Singapore, January 1, 1970. https://link.springer.com/chapter/10.1007/978-981-4585-18-7_28.
8. "24 Best Ecommerce & Retail Datasets for Machine Learning." Lionbridge AI, September 20, 2019. https://lionbridge.ai/datasets/24-best-ecommerce-retail-datasets-for-machine-learning/.
9. "1.2 Sample ACF and Properties of AR(1) Model: STAT 510." PennState: Statistics Online Courses. Accessed December 17, 2019. https://newonlinecourses.science.psu.edu/stat510/lesson/1/1.2.