Problem 3 (15 credits)

HW3

Maya Carnie, Xianging He, Olivia Liang, Lauren Moore, Ilana Novakoski, William Wu; carni015, he000273, linag625, moor1985, novak560, wuxx1066

February 24, 2020

```r
suppressPackageStartupMessages({
  library(TSA)
  library(ggplot2)
  library(dplyr)
  library(forecast)
  library(tseries) #Only for the ADF test for testing stationarity
})
```

```
## Warning: package 'TSA' was built under R version 3.5.3

## Warning: package 'ggplot2' was built under R version 3.5.3

## Warning: package 'dplyr' was built under R version 3.5.3

## Warning: package 'forecast' was built under R version 3.5.3

## Warning: package 'tseries' was built under R version 3.5.3
```

Time Series Data of Your Choice

Background

This homework problem will allow you to apply the learned time series analysis and forecasting skills to your own or favorite dataset. This dataset could be your own data (from your interested hobby groups, sports or video game records, previous jobs, past school works, etc). Notice that, we DON'T require data disclosure so please feel free to use your own data if you would like us to help you understand the result. Or if you don't have any time series dataset, please feel free to get one from the Internet. Any topics are welcome!

Hint: If you have trouble finding a good dataset, sources of public time series data include `kaggle.com` where many of the class examples came from, and `Yahoo Finance` which provides rich information about historical prices of nearly every US stock. Or if you are a sport fun or a video game fun, I believe that similar data collections are also available online.

So, please feel free to explore! I would suggest using data from sources other than `Kaggle` and `Yahoo Finance` to avoid similarity.

Importance

Notice that, the questions I provided in this problem are mostly real-world tasks that we encountered in many data scientists' daily job, including those working in financial sectors (such as hedge fund companies). So while finishing the questions as required assignments, please make sure to take a few minutes to understand why these questions are raised, and what's the standard procedures to address them.

The credit will be given based on whether you do everything in standard procedures, as opposed to the results such as whether the forecasting accuracy is good.

General Requirements

However, I do have some very general and mild requirements in order for the analysis to be valid.

1. Please make sure the time series contains at least $T = 500$ time points. The final credits will be prorated if T is less than 500 (`floor` to the nearest hundred, for example, 499 will become 400, and hence $400/500 = 80\%$ credits will be given).
2. Please make sure the data are REAL data, not simulated ones. Given there're plenty of available datasets online, I can't find a reason to simulate data. Only 50% credits will be given if we find out the data are simulated.
3. Also make sure the data are non-trivial (having sufficient data variation and possibly a trend). For example, it is trivial to analyze a series of 500 zeros, denoting something like "the number of spacecrafts I owned in the past 1.5 years". 0 credits will be given if the data are regarded as trivial.
4. If two groups happen to use the same dataset (or one dataset being the subset of another), I reserve the right to place the two homework under scrutiny.
5. Please do not use any datasets (or their subsets) used in the lectures or previous homework. Otherwise, 0 credits will be given to this problem.

Question 1 (1 credit)

Please briefly describe the background of your dataset as I did for the Boston Crime Data in Homework 1 Problem 3, and its source (link) if you are using public data.

```
# This is a publicly-available data set from the U.S. Census Bureau (accessed through Kaggle) of the U.
# population over time.
# Link: https://www.kaggle.com/census/population-time-series-data
```

Question 2 (1 credit)

Please plot your data and provide the sample size. Use the first 80% of the data as training, and the last 20% as testing.

```
#Please provide your code here
library(tseries)
raw_pop <-read.csv("POP.csv")
pop<-raw_pop[,0:2]
head(pop)
```

```
##   realtime_start  value
## 1     2019-12-06 156309
## 2     2019-12-06 156527
## 3     2019-12-06 156731
## 4     2019-12-06 156943
## 5     2019-12-06 157140
## 6     2019-12-06 157343
```

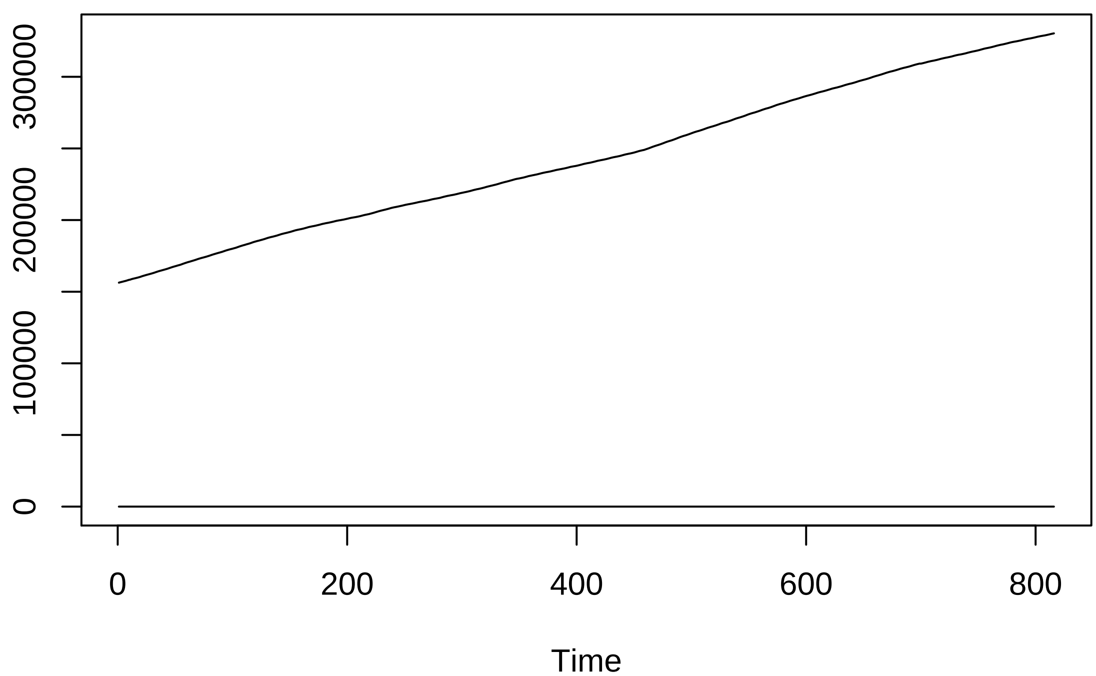
```r
nrow(pop) #816
```

```
## [1] 816
```

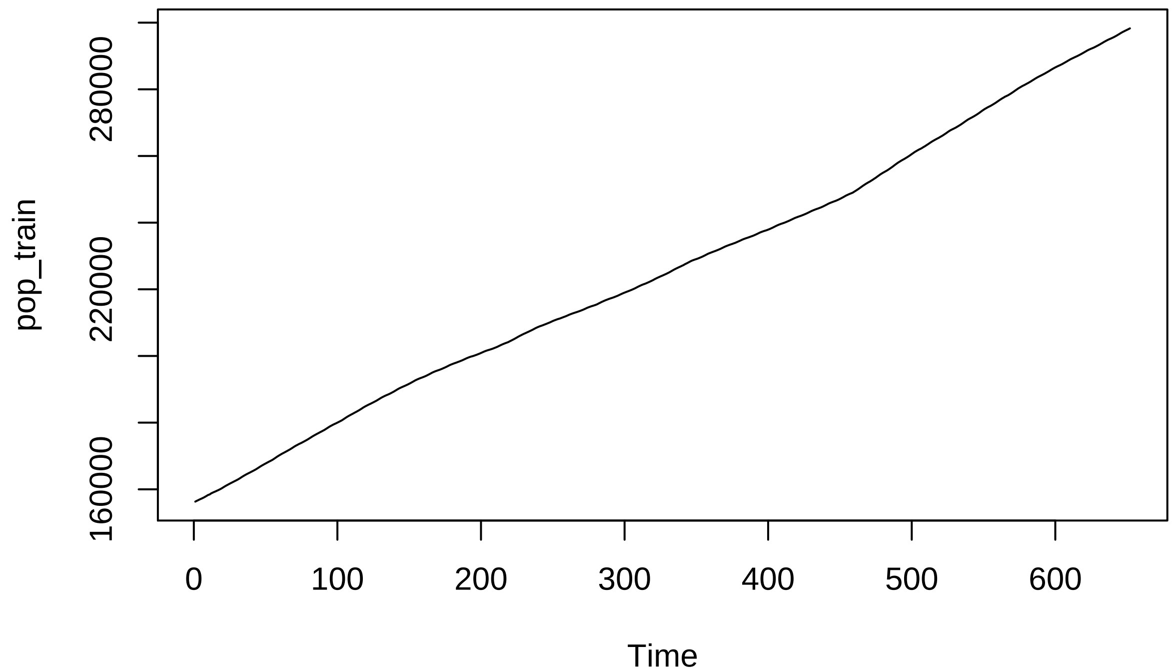
```r
train_len <-.8*816
train_len #652.8
```

```
## [1] 652.8
```

```r
T = ts(pop)
ts.plot(T)
```



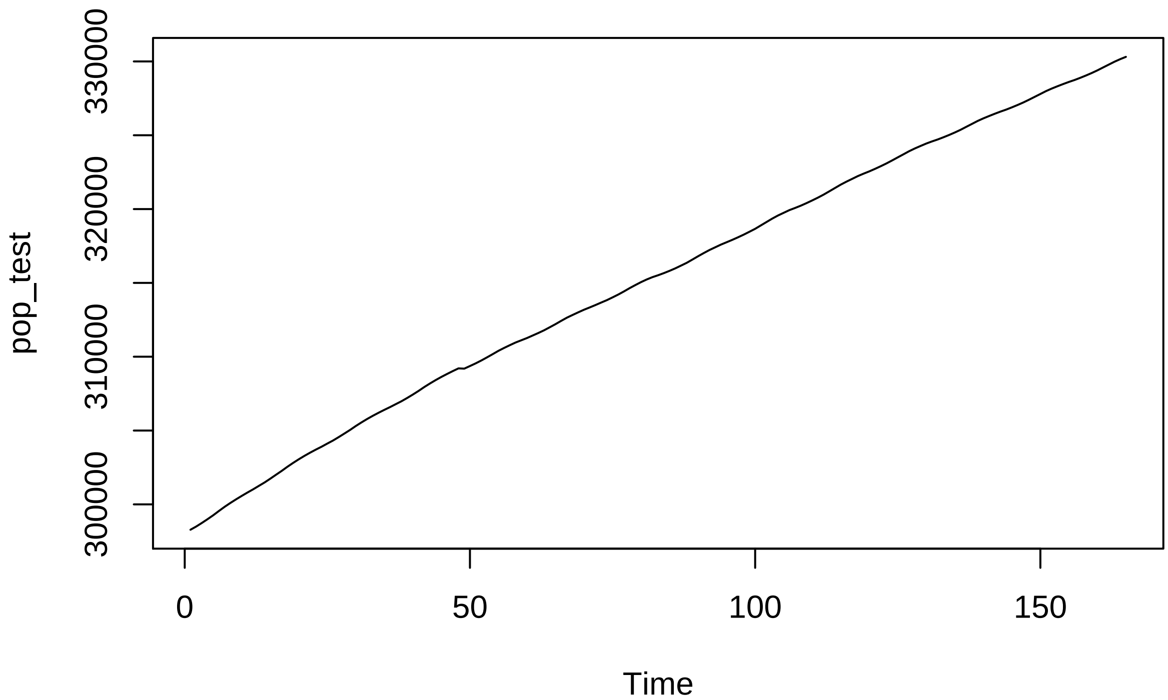

```r
#Training Set
pop_train=T[0:652,2]
ts.plot(pop_train)
```




```r
head(pop_train)
```

```
## [1] 156309 156527 156731 156943 157140 157343
```

```r
#Testing set
pop_test=T[652:816,2]
ts.plot(pop_test)
```



Question 3 (2 credits)

On the TRAINING set:

Please (make transformations if necessary, and) use the `ADF test` to check for stationarity. Remove trend if necessary, and check the residuals for spurious regression (proof of random walk)

Check ACF, PACF, and EACF for the order of the ARMA model (after differencing, if it has a random walk). Use AIC or BIC to select a final model from your candidate models. Report the orders.

```
#Please provide your code here
  library(TSA)
  library(ggplot2)
  library(dplyr)
  library(forecast)
  library(tseries)

arima0<-auto.arima(pop_train)

#ARIMA(5,2,1)

#Coefficients:
#intercept
        1
```

```
## [1] 1
```

```r
#sigma^2 estimated as 0:  log likelihood=Inf
#AIC=-Inf   AICc=-Inf   BIC=-Inf

#Coefficients:
#         ar1     ar2      ar3      ar4      ar5      ma1
#      0.4609  0.1869  -0.0664  -0.1684  -0.2862  -0.7999
#s.e.  0.0415  0.0411   0.0419   0.0411   0.0394   0.0225#

#sigma^2 estimated as 393.6:  log likelihood=-2793.1
#AIC=5600.21   AICc=5600.38   BIC=5631.55

adf.test(pop_train)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  pop_train
## Dickey-Fuller = -1.1475, Lag order = 8, p-value = 0.9149
## alternative hypothesis: stationary
```

```r
#Dickey-Fuller = -1.1475, Lag order = 8, p-value = 0.9149
#Not stationary

diff_pop <- c(0,0,diff(diff(pop_train)))
auto.arima(diff_pop)
```

```
## Series: diff_pop
## ARIMA(5,0,1) with zero mean
##
## Coefficients:
##          ar1     ar2      ar3      ar4      ar5      ma1
##       0.4611  0.1871  -0.0666  -0.1685  -0.2852  -0.7999
## s.e.  0.0415  0.0411   0.0418   0.0410   0.0393   0.0225
##
## sigma^2 estimated as 317.3:  log likelihood=-2800.93
## AIC=5615.85   AICc=5616.02   BIC=5647.21
```

```r
#ARIMA(5,0,1)
#      ar1     ar2      ar3      ar4      ar5      ma1
#      0.4611  0.1871  -0.0666  -0.1685  -0.2852  -0.7999
#s.e.  0.0415  0.0411   0.0418   0.0410   0.0393   0.0225
adf.test((diff_pop))
```

```
## Warning in adf.test((diff_pop)): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  (diff_pop)
## Dickey-Fuller = -20.434, Lag order = 8, p-value = 0.01
## alternative hypothesis: stationary
```

```
#Dickey-Fuller = -20.434, Lag order = 8, p-value = 0.01
#alternative hypothesis: stationary
#this appears to be stationary


#Final Model: ARIMA(5,2,1)(5,0,1)[2]
```

Question 4 (2 credits)

Fit your final model, write down the model (You may write down only the non-seasonal part, if you model contains seasonality).

Report the significance of the model coefficients.

Hints:

- Check Homework 2 - Problem 3 - Question 1(b) and 1(c) for how to write a model and how to define significance.

Answer:

$$Y_t = .46 \cdot (Y_{t-1}) + .187 \cdot (Y_{t-2}) - 0.66 \cdot (Y_{t-3}) - .1685 \cdot (Y_{t-4}) - .285 \cdot (Y_{t-5}) - .79 \cdot (e_{t-1})$$

```
#arima_fit
```

Question 5 (3 credits)

Forecast on the testing set. Provide RMSE.

Plot the fitted value, as well as 80% and 95% prediction intervals, superimposed on the raw data.

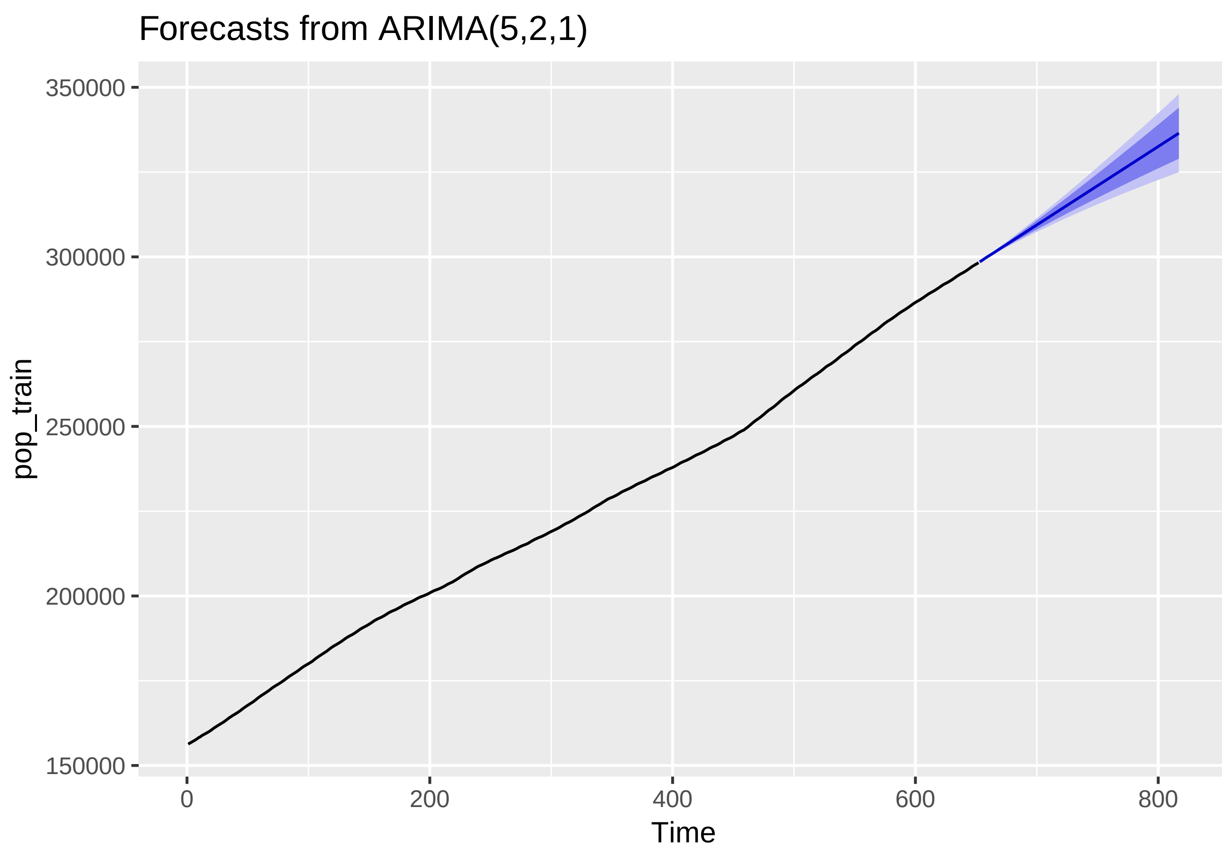
Explain whether your selected model fit the data well.

Hint:

- Please check the code of Lecture 7, where similar things are done for the US Consumption data, CO2 data and Bitcoin data
- If you made transformations on your training data, please use the same transformation on your testing data as well

```
diff_pop_test <- c(0,0,diff(diff(pop_test)))

arima0_forecast=forecast(arima0,h=length(diff_pop_test))
a_forecast=arima0_forecast$mean
autoplot(arima0_forecast)
```




```
  labs(title="U.S. Population Over Time", y="Population")
```

```
## $y
## [1] "Population"
##
## $title
## [1] "U.S. Population Over Time"
##
## attr(,"class")
## [1] "labels"
```

```
library(Metrics)
```

```
## Warning: package 'Metrics' was built under R version 3.5.3
```

```
##
## Attaching package: 'Metrics'
```

```
## The following object is masked from 'package:forecast':
##
##     accuracy
```

```r
rmse_arima=rmse(arima0_forecast$mean, diff_pop_test)
rmse_arima
```

```
## [1] 317703.5
```

```r
#yes, the mean and variation look reasonable
```

Question 6 (6 credits)

Please do the same forecasting task in Question 5, with XGBoost and LSTM.

Report the RMSE from each method.

For each method, plot the fitted value, superimposed on the raw data (prediction intervals are not required).

Comments on the performance of XGBoost and LSTM compared with ARIMA, in terms of both accuracy and computational speed. Which one is better for your data?

Hint:

1. Please feel free to use Python or other software to run XGBoost or LSTM if your code has been ready. But please paste all code as comments in the area below for reproducibility reason.
2. Some of my experiences are: ARIMA has the advantages of being fast, and being able to provide prediction intervals as a statistical model. But XGBoost and LSTM may provide better forecasting accuracy.
3. For LSTM, it's OK if you only have time to try a couple of layers with a few neurons.

```r
#Please provide your code, explanation and figures here, regardless of what software you use

# The LSTM and XGBoost are on the following pages
# RMSE for this model was 317703.5
# RMSE for LSTM was 1905.51
# RMSE for XGBoost was 19569.43
# Based on this and the performance charts, it appears that the best model for prediction
# in terms of accuracy is the LSTM model. The ARIMA seems to have performed the worst.
```

In [4]:
```
import pandas as pd
import numpy as np
```

In [74]:
```
pop_data = pd.read_csv("POP.csv", )
pop_data.head()
```

Out[74]:

	realtime_start	value	date	realtime_end
0	2019-12-06	156309.0	1952-01-01	2019-12-06
1	2019-12-06	156527.0	1952-02-01	2019-12-06
2	2019-12-06	156731.0	1952-03-01	2019-12-06
3	2019-12-06	156943.0	1952-04-01	2019-12-06
4	2019-12-06	157140.0	1952-05-01	2019-12-06

In [88]:
```
# Length of the dataframe is 816, 80/20 split is at 653 observations
df = pop_data[["date", "value"]]
df['date'] = pd.to_datetime(df['date'])
```

```
C:\Users\billp\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

In [95]:
```
df['dayofweek'] = df['date'].dt.dayofweek
df['quarter'] = df['date'].dt.quarter
df['month'] = df['date'].dt.month
df['year'] = df['date'].dt.year
df['dayofyear'] = df['date'].dt.dayofyear
df['dayofmonth'] = df['date'].dt.day
df['weekofyear'] = df['date'].dt.weekofyear
df = df.drop(columns = "date", axis = 1)
df
```

```
C:\Users\billp\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

Out[95]:

	value	dayofweek	quarter	month	year	dayofyear	dayofmonth	weekofyear
0	156309.000	1	1	1	1952	1	1	1
1	156527.000	4	1	2	1952	32	1	5
2	156731.000	5	1	3	1952	61	1	9
3	156943.000	1	2	4	1952	92	1	14
4	157140.000	3	2	5	1952	122	1	18
5	157343.000	6	2	6	1952	153	1	22
6	157553.000	1	3	7	1952	183	1	27
7	157798.000	4	3	8	1952	214	1	31
8	158053.000	0	3	9	1952	245	1	36
9	158306.000	2	4	10	1952	275	1	40
10	158451.000	5	4	11	1952	306	1	44
11	158757.000	0	4	12	1952	336	1	49
12	158973.000	3	1	1	1953	1	1	1
13	159170.000	6	1	2	1953	32	1	5
14	159349.000	6	1	3	1953	60	1	9
15	159556.000	2	2	4	1953	91	1	14
16	159745.000	4	2	5	1953	121	1	18
17	159956.000	0	2	6	1953	152	1	23
18	160184.000	2	3	7	1953	182	1	27
19	160449.000	5	3	8	1953	213	1	31
20	160718.000	1	3	9	1953	244	1	36
21	160978.000	3	4	10	1953	274	1	40
22	161223.000	6	4	11	1953	305	1	44
23	161453.000	1	4	12	1953	335	1	49
24	161690.000	4	1	1	1954	1	1	53
25	161912.000	0	1	2	1954	32	1	5
26	162124.000	0	1	3	1954	60	1	9
27	162350.000	3	2	4	1954	91	1	13
28	162564.000	5	2	5	1954	121	1	17
29	162790.000	1	2	6	1954	152	1	22
...	...	...	...	...	...	...	...	...
786	325367.612	5	3	7	2017	182	1	26
787	325567.716	1	3	8	2017	213	1	31
788	325766.019	4	3	9	2017	244	1	35
789	325965.952	6	4	10	2017	274	1	39

	value	dayofweek	quarter	month	year	dayofyear	dayofmonth	weekofyear
790	326142.633	2	4	11	2017	305	1	44
791	326301.399	4	4	12	2017	335	1	48
792	326454.123	0	1	1	2018	1	1	1
793	326600.823	3	1	2	2018	32	1	5
794	326736.690	3	1	3	2018	60	1	9
795	326887.866	6	2	4	2018	91	1	13
796	327048.704	1	2	5	2018	121	1	18
797	327219.140	4	2	6	2018	152	1	22
798	327403.909	6	3	7	2018	182	1	26
799	327600.250	2	3	8	2018	213	1	31
800	327794.788	5	3	9	2018	244	1	35
801	327990.950	0	4	10	2018	274	1	40
802	328163.864	3	4	11	2018	305	1	44
803	328318.861	5	4	12	2018	335	1	48
804	328467.812	1	1	1	2019	1	1	1
805	328610.744	4	1	2	2019	32	1	5
806	328742.843	4	1	3	2019	60	1	9
807	328890.250	0	2	4	2019	91	1	14
808	329047.319	2	2	5	2019	121	1	18
809	329213.989	5	2	6	2019	152	1	22
810	329394.993	0	3	7	2019	182	1	27
811	329591.333	3	3	8	2019	213	1	31
812	329785.872	6	3	9	2019	244	1	35
813	329982.035	1	4	10	2019	274	1	40
814	330154.949	4	4	11	2019	305	1	44
815	330309.946	6	4	12	2019	335	1	48

816 rows × 8 columns

In [96]:
```
train = df[0:653]
test = df[653:816]
test
```

Out[96]:

	value	dayofweek	quarter	month	year	dayofyear	dayofmonth	weekofyear
653	298739.000	3	2	6	2006	152	1	22
654	298996.000	5	3	7	2006	182	1	26
655	299263.000	1	3	8	2006	213	1	31
656	299554.000	4	3	9	2006	244	1	35
657	299835.000	6	4	10	2006	274	1	39
658	300094.000	2	4	11	2006	305	1	44
659	300340.000	4	4	12	2006	335	1	48
660	300574.000	0	1	1	2007	1	1	1
661	300802.000	3	1	2	2007	32	1	5
662	301021.000	3	1	3	2007	60	1	9
663	301254.000	6	2	4	2007	91	1	13
664	301483.000	1	2	5	2007	121	1	18
665	301739.000	4	2	6	2007	152	1	22
666	302004.000	6	3	7	2007	182	1	26
667	302267.000	2	3	8	2007	213	1	31
668	302546.000	5	3	9	2007	244	1	35
669	302807.000	0	4	10	2007	274	1	40
670	303054.000	3	4	11	2007	305	1	44
671	303287.000	5	4	12	2007	335	1	48
672	303506.000	1	1	1	2008	1	1	1
673	303711.000	4	1	2	2008	32	1	5
674	303907.000	5	1	3	2008	61	1	9
675	304117.000	1	2	4	2008	92	1	14
676	304323.000	3	2	5	2008	122	1	18
677	304556.000	6	2	6	2008	153	1	22
678	304798.000	1	3	7	2008	183	1	27
679	305045.000	4	3	8	2008	214	1	31
680	305309.000	0	3	9	2008	245	1	36
681	305554.000	2	4	10	2008	275	1	40
682	305786.000	5	4	11	2008	306	1	44
...	...	...	...	...	...	...	...	...
786	325367.612	5	3	7	2017	182	1	26
787	325567.716	1	3	8	2017	213	1	31
788	325766.019	4	3	9	2017	244	1	35
789	325965.952	6	4	10	2017	274	1	39

	value	dayofweek	quarter	month	year	dayofyear	dayofmonth	weekofyear
790	326142.633	2	4	11	2017	305	1	44
791	326301.399	4	4	12	2017	335	1	48
792	326454.123	0	1	1	2018	1	1	1
793	326600.823	3	1	2	2018	32	1	5
794	326736.690	3	1	3	2018	60	1	9
795	326887.866	6	2	4	2018	91	1	13
796	327048.704	1	2	5	2018	121	1	18
797	327219.140	4	2	6	2018	152	1	22
798	327403.909	6	3	7	2018	182	1	26
799	327600.250	2	3	8	2018	213	1	31
800	327794.788	5	3	9	2018	244	1	35
801	327990.950	0	4	10	2018	274	1	40
802	328163.864	3	4	11	2018	305	1	44
803	328318.861	5	4	12	2018	335	1	48
804	328467.812	1	1	1	2019	1	1	1
805	328610.744	4	1	2	2019	32	1	5
806	328742.843	4	1	3	2019	60	1	9
807	328890.250	0	2	4	2019	91	1	14
808	329047.319	2	2	5	2019	121	1	18
809	329213.989	5	2	6	2019	152	1	22
810	329394.993	0	3	7	2019	182	1	27
811	329591.333	3	3	8	2019	213	1	31
812	329785.872	6	3	9	2019	244	1	35
813	329982.035	1	4	10	2019	274	1	40
814	330154.949	4	4	11	2019	305	1	44
815	330309.946	6	4	12	2019	335	1	48

163 rows × 8 columns

In [161]:
```
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

xgbst=xgb.XGBRegressor()

param_test = {
    'learning_rate':[0.2,0.15,0.1,0.05,0.01],
    'n_estimators':range(5,200,10),
     'max_depth':range(3,20,2),
    'min_child_weight':range(1,6,2),
    'gamma':[i/10.0 for i in range(0,5)],
    'subsample':[i/10.0 for i in range(6,10)],
    'colsample_bytree':[i/10.0 for i in range(6,10)],
    'reg_alpha':[1e-5, 1e-2, 0.1, 1, 100]
}
```

In [181]:
```
model_xgb_hp = RandomizedSearchCV(xgbst, param_test, cv=3, n_jobs= -1)
model_xgb_hp.fit(train[["dayofweek","quarter","month","year","dayofyear","dayo
fmonth","weekofyear"]].values,train["value"].values)
model_xgb_hp.best_params_
```

```
C:\Users\billp\Anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:841: DeprecationWarning: The default of the `iid` parameter will change fro
m True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

Out[181]:
```
{'subsample': 0.7,
 'reg_alpha': 1e-05,
 'n_estimators': 155,
 'min_child_weight': 1,
 'max_depth': 17,
 'learning_rate': 0.2,
 'gamma': 0.4,
 'colsample_bytree': 0.8}
```

In [182]:
```
preds_xg_hp = model_xgb_hp.predict(test[["dayofweek","quarter","month","year",
"dayofyear","dayofmonth","weekofyear"]].values)
preds_xg_hp
```

Out[182]:
```
array([298315.2 , 298233.62, 298284.78, 298303.88, 298469.12, 298612.72,
       298688.47, 297202.56, 297783.75, 298052.53, 298270.34, 298250.97,
       298284.56, 298330.9 , 298290.97, 298384.8 , 298573.66, 298747.84,
       298819.84, 296981.1 , 297599.34, 297644.7 , 297920.25, 297982.22,
       298099.16, 298178.16, 297734.4 , 298137.72, 298218.3 , 298261.12,
       298304.06, 297152.47, 297765.3 , 297935.38, 298076.75, 298113.9 ,
       298338.75, 298224.62, 298170.56, 298365.28, 298518.4 , 298579.1 ,
       298768.12, 297058.6 , 297439.7 , 297684.47, 297818.2 , 297798.5 ,
       298164.94, 298095.34, 298100.4 , 298194.72, 298262.44, 298499.66,
       298590.47, 297478.47, 297725.97, 297908.7 , 298053.28, 298197.53,
       298224.4 , 298151.9 , 298160.34, 298290.75, 298417.9 , 298636.66,
       298711.94, 297646.7 , 297854.97, 297762.78, 297902.06, 298017.38,
       298024.22, 298034.38, 297879.4 , 298108.56, 298377.7 , 298355.88,
       298478.34, 296981.1 , 297599.34, 297883.03, 298048.72, 298145.56,
       298232.34, 298229.75, 298190.4 , 298334.34, 298501.72, 298547.62,
       298670.06, 297049.3 , 297629.47, 297838.38, 297974.06, 298088.62,
       298237.34, 298250.47, 298201.53, 298421.78, 298489.6 , 298583.4 ,
       298761.62, 297152.47, 297765.3 , 297935.38, 298076.75, 298113.9 ,
       298338.75, 298224.62, 298170.56, 298365.28, 298518.4 , 298579.1 ,
       298768.12, 297058.6 , 297439.7 , 297648.66, 297919.03, 297873.25,
       297965.4 , 297886.62, 297750.38, 298051.1 , 298223.84, 298379.06,
       298427.53, 297646.7 , 297854.97, 298059.62, 298281.88, 298494.62,
       298315.2 , 298233.62, 298284.78, 298303.88, 298469.12, 298612.72,
       298688.47, 297202.56, 297783.75, 298052.53, 298270.34, 298250.97,
       298284.56, 298330.9 , 298290.97, 298384.8 , 298573.66, 298747.84,
       298819.84, 296981.1 , 297599.34, 297883.03, 298048.72, 298145.56,
       298232.34, 298229.75, 298190.4 , 298334.34, 298501.72, 298547.62,
       298670.06], dtype=float32)
```

RMSE of XGBoost Model

In [183]:
```
np.sqrt(((preds_xg_hp - test['value'].values) ** 2).mean())
```

Out[183]:
```
19569.430906119174
```

Graph

In [184]:
```
import matplotlib.pyplot as plt
```

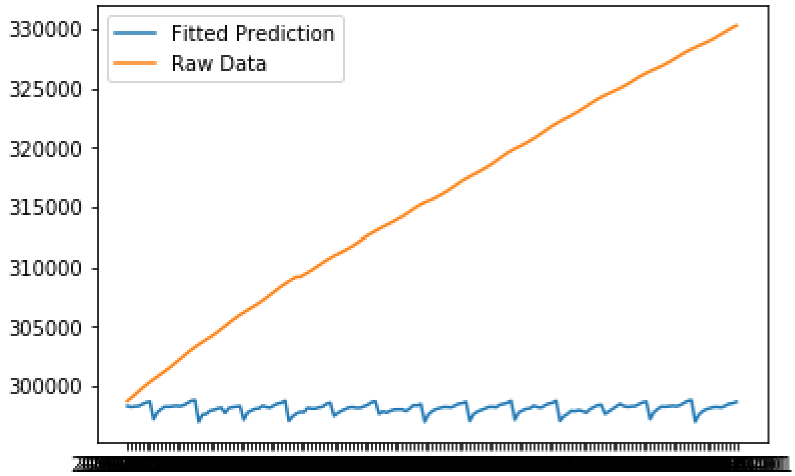
In [185]:
```
df2 = pop_data
x = df2['date'][653:816]
x = np.asarray(x)
```

In [186]:
```
y1 = preds_xg_hp
y2 = df2['value'][653:816].values
```

In [187]:
```
df3 = pd.DataFrame({'Time': x, 'Fitted Prediction': y1, 'Raw Data':y2})
```

In [188]:
```
plt.plot('Time','Fitted Prediction', data = df3)
plt.plot('Time','Raw Data', data = df3)
plt.legend()
```

Out[188]: <matplotlib.legend.Legend at 0x2457296c860>



LSTM for POP ran

February 24, 2020

[1]:
```python
import numpy
import matplotlib.pyplot as plt
import pandas
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

```
Using TensorFlow backend.
```

[2]:
```python
# load the dataset
dataframe = pandas.read_csv('POP.csv', usecols=[1], engine='python')
dataset = dataframe.values
dataset = dataset.astype('float32')
```

[3]:
```python
dataset
```

[3]:
```
array([[156309.  ],
       [156527.  ],
       [156731.  ],
       [156943.  ],
       [157140.  ],
       [157343.  ],
       [157553.  ],
       [157798.  ],
       [158053.  ],
       [158306.  ],
       [158451.  ],
       [158757.  ],
       [158973.  ],
       [159170.  ],
       [159349.  ],
       [159556.  ],
       [159745.  ],
       [159956.  ],
       [160184.  ],
```

```
[160449.  ],
[160718.  ],
[160978.  ],
[161223.  ],
[161453.  ],
[161690.  ],
[161912.  ],
[162124.  ],
[162350.  ],
[162564.  ],
[162790.  ],
[163026.  ],
[163290.  ],
[163570.  ],
[163847.  ],
[164107.  ],
[164349.  ],
[164588.  ],
[164809.  ],
[165018.  ],
[165251.  ],
[165463.  ],
[165695.  ],
[165931.  ],
[166192.  ],
[166473.  ],
[166755.  ],
[167023.  ],
[167270.  ],
[167513.  ],
[167746.  ],
[167977.  ],
[168221.  ],
[168436.  ],
[168659.  ],
[168903.  ],
[169191.  ],
[169488.  ],
[169780.  ],
[170063.  ],
[170315.  ],
[170571.  ],
[170806.  ],
[171029.  ],
[171271.  ],
[171501.  ],
[171741.  ],
```

```
[171984.  ],
[172257.  ],
[172538.  ],
[172816.  ],
[173070.  ],
[173298.  ],
[173533.  ],
[173746.  ],
[173945.  ],
[174176.  ],
[174397.  ],
[174639.  ],
[174882.  ],
[175143.  ],
[175413.  ],
[175697.  ],
[175966.  ],
[176207.  ],
[176447.  ],
[176685.  ],
[176905.  ],
[177146.  ],
[177365.  ],
[177591.  ],
[177830.  ],
[178101.  ],
[178376.  ],
[178657.  ],
[178921.  ],
[179153.  ],
[179386.  ],
[179597.  ],
[179788.  ],
[180007.  ],
[180222.  ],
[180444.  ],
[180671.  ],
[180945.  ],
[181238.  ],
[181528.  ],
[181796.  ],
[182042.  ],
[182287.  ],
[182520.  ],
[182742.  ],
[182992.  ],
[183217.  ],
```

```
[183452.  ],
[183691.  ],
[183958.  ],
[184243.  ],
[184524.  ],
[184783.  ],
[185016.  ],
[185242.  ],
[185452.  ],
[185650.  ],
[185874.  ],
[186087.  ],
[186314.  ],
[186538.  ],
[186790.  ],
[187058.  ],
[187323.  ],
[187574.  ],
[187796.  ],
[188013.  ],
[188213.  ],
[188387.  ],
[188580.  ],
[188790.  ],
[189018.  ],
[189242.  ],
[189496.  ],
[189761.  ],
[190028.  ],
[190265.  ],
[190472.  ],
[190668.  ],
[190858.  ],
[191047.  ],
[191245.  ],
[191447.  ],
[191666.  ],
[191889.  ],
[192131.  ],
[192376.  ],
[192631.  ],
[192847.  ],
[193039.  ],
[193223.  ],
[193393.  ],
[193540.  ],
[193709.  ],
```

```
[193888.  ],
[194087.  ],
[194303.  ],
[194528.  ],
[194761.  ],
[194997.  ],
[195195.  ],
[195372.  ],
[195539.  ],
[195688.  ],
[195831.  ],
[195999.  ],
[196178.  ],
[196372.  ],
[196560.  ],
[196762.  ],
[196984.  ],
[197207.  ],
[197398.  ],
[197572.  ],
[197736.  ],
[197892.  ],
[198037.  ],
[198206.  ],
[198363.  ],
[198537.  ],
[198712.  ],
[198911.  ],
[199113.  ],
[199311.  ],
[199498.  ],
[199657.  ],
[199808.  ],
[199920.  ],
[200056.  ],
[200208.  ],
[200361.  ],
[200536.  ],
[200706.  ],
[200898.  ],
[201095.  ],
[201290.  ],
[201466.  ],
[201621.  ],
[201760.  ],
[201881.  ],
[202023.  ],
```

```
[202161.  ],
[202331.  ],
[202507.  ],
[202677.  ],
[202877.  ],
[203090.  ],
[203302.  ],
[203500.  ],
[203675.  ],
[203849.  ],
[204008.  ],
[204156.  ],
[204401.  ],
[204607.  ],
[204830.  ],
[205052.  ],
[205295.  ],
[205540.  ],
[205788.  ],
[206024.  ],
[206238.  ],
[206466.  ],
[206668.  ],
[206855.  ],
[207065.  ],
[207260.  ],
[207462.  ],
[207661.  ],
[207881.  ],
[208114.  ],
[208345.  ],
[208555.  ],
[208740.  ],
[208917.  ],
[209061.  ],
[209212.  ],
[209386.  ],
[209545.  ],
[209725.  ],
[209896.  ],
[210075.  ],
[210278.  ],
[210479.  ],
[210656.  ],
[210821.  ],
[210985.  ],
[211120.  ],
```

```
[211254.  ],
[211420.  ],
[211577.  ],
[211746.  ],
[211909.  ],
[212092.  ],
[212289.  ],
[212475.  ],
[212634.  ],
[212785.  ],
[212932.  ],
[213074.  ],
[213211.  ],
[213361.  ],
[213513.  ],
[213686.  ],
[213854.  ],
[214042.  ],
[214246.  ],
[214451.  ],
[214625.  ],
[214782.  ],
[214931.  ],
[215065.  ],
[215198.  ],
[215353.  ],
[215523.  ],
[215768.  ],
[215973.  ],
[216195.  ],
[216393.  ],
[216587.  ],
[216771.  ],
[216931.  ],
[217095.  ],
[217249.  ],
[217381.  ],
[217528.  ],
[217685.  ],
[217861.  ],
[218035.  ],
[218233.  ],
[218440.  ],
[218644.  ],
[218834.  ],
[219006.  ],
[219179.  ],
```

```
[219344.  ],
[219504.  ],
[219684.  ],
[219859.  ],
[220046.  ],
[220239.  ],
[220458.  ],
[220688.  ],
[220904.  ],
[221109.  ],
[221303.  ],
[221477.  ],
[221629.  ],
[221792.  ],
[221991.  ],
[222176.  ],
[222379.  ],
[222585.  ],
[222805.  ],
[223053.  ],
[223271.  ],
[223477.  ],
[223670.  ],
[223865.  ],
[224053.  ],
[224235.  ],
[224438.  ],
[224632.  ],
[224843.  ],
[225055.  ],
[225295.  ],
[225547.  ],
[225801.  ],
[226027.  ],
[226243.  ],
[226451.  ],
[226656.  ],
[226849.  ],
[227061.  ],
[227251.  ],
[227522.  ],
[227726.  ],
[227953.  ],
[228186.  ],
[228417.  ],
[228612.  ],
[228779.  ],
```

```
[228937.  ],
[229071.  ],
[229224.  ],
[229403.  ],
[229575.  ],
[229761.  ],
[229966.  ],
[230187.  ],
[230412.  ],
[230641.  ],
[230822.  ],
[230989.  ],
[231157.  ],
[231313.  ],
[231470.  ],
[231645.  ],
[231809.  ],
[231992.  ],
[232188.  ],
[232392.  ],
[232599.  ],
[232816.  ],
[232993.  ],
[233160.  ],
[233322.  ],
[233473.  ],
[233613.  ],
[233781.  ],
[233922.  ],
[234118.  ],
[234307.  ],
[234501.  ],
[234701.  ],
[234907.  ],
[235078.  ],
[235235.  ],
[235385.  ],
[235527.  ],
[235675.  ],
[235839.  ],
[235993.  ],
[236160.  ],
[236348.  ],
[236549.  ],
[236760.  ],
[236976.  ],
[237159.  ],
```

```
[237316.  ],
[237468.  ],
[237602.  ],
[237732.  ],
[237900.  ],
[238074.  ],
[238270.  ],
[238466.  ],
[238679.  ],
[238898.  ],
[239113.  ],
[239307.  ],
[239477.  ],
[239638.  ],
[239788.  ],
[239928.  ],
[240094.  ],
[240271.  ],
[240459.  ],
[240651.  ],
[240854.  ],
[241068.  ],
[241274.  ],
[241467.  ],
[241620.  ],
[241784.  ],
[241930.  ],
[242079.  ],
[242252.  ],
[242423.  ],
[242608.  ],
[242804.  ],
[243012.  ],
[243223.  ],
[243446.  ],
[243639.  ],
[243809.  ],
[243981.  ],
[244131.  ],
[244279.  ],
[244445.  ],
[244610.  ],
[244806.  ],
[245021.  ],
[245240.  ],
[245464.  ],
[245693.  ],
```

```
[245884.  ],
[246056.  ],
[246224.  ],
[246378.  ],
[246530.  ],
[246721.  ],
[246906.  ],
[247114.  ],
[247342.  ],
[247573.  ],
[247816.  ],
[248067.  ],
[248281.  ],
[248479.  ],
[248659.  ],
[248827.  ],
[249012.  ],
[249306.  ],
[249565.  ],
[249849.  ],
[250132.  ],
[250439.  ],
[250751.  ],
[251057.  ],
[251346.  ],
[251626.  ],
[251889.  ],
[252135.  ],
[252372.  ],
[252643.  ],
[252913.  ],
[253207.  ],
[253493.  ],
[253807.  ],
[254126.  ],
[254435.  ],
[254718.  ],
[254964.  ],
[255214.  ],
[255448.  ],
[255703.  ],
[255992.  ],
[256285.  ],
[256589.  ],
[256894.  ],
[257232.  ],
[257548.  ],
```

```
[257861.  ],
[258147.  ],
[258413.  ],
[258679.  ],
[258919.  ],
[259152.  ],
[259414.  ],
[259680.  ],
[259963.  ],
[260255.  ],
[260566.  ],
[260867.  ],
[261163.  ],
[261425.  ],
[261674.  ],
[261919.  ],
[262123.  ],
[262352.  ],
[262631.  ],
[262877.  ],
[263152.  ],
[263436.  ],
[263724.  ],
[264017.  ],
[264301.  ],
[264559.  ],
[264804.  ],
[265044.  ],
[265270.  ],
[265495.  ],
[265755.  ],
[265998.  ],
[266270.  ],
[266557.  ],
[266843.  ],
[267152.  ],
[267456.  ],
[267715.  ],
[267943.  ],
[268151.  ],
[268364.  ],
[268595.  ],
[268853.  ],
[269108.  ],
[269386.  ],
[269667.  ],
[269976.  ],
```

```
[270284.  ],
[270581.  ],
[270878.  ],
[271125.  ],
[271360.  ],
[271585.  ],
[271821.  ],
[272083.  ],
[272342.  ],
[272622.  ],
[272912.  ],
[273237.  ],
[273553.  ],
[273852.  ],
[274126.  ],
[274372.  ],
[274626.  ],
[274838.  ],
[275047.  ],
[275304.  ],
[275564.  ],
[275836.  ],
[276115.  ],
[276418.  ],
[276714.  ],
[277003.  ],
[277277.  ],
[277526.  ],
[277790.  ],
[277992.  ],
[278198.  ],
[278451.  ],
[278717.  ],
[279001.  ],
[279295.  ],
[279602.  ],
[279903.  ],
[280203.  ],
[280471.  ],
[280716.  ],
[280976.  ],
[281190.  ],
[281409.  ],
[281653.  ],
[281877.  ],
[282126.  ],
[282385.  ],
```

```
[282653.  ],
[282932.  ],
[283201.  ],
[283453.  ],
[283696.  ],
[283920.  ],
[284137.  ],
[284350.  ],
[284581.  ],
[284810.  ],
[285062.  ],
[285309.  ],
[285570.  ],
[285843.  ],
[286098.  ],
[286341.  ],
[286570.  ],
[286788.  ],
[286994.  ],
[287190.  ],
[287397.  ],
[287623.  ],
[287864.  ],
[288105.  ],
[288360.  ],
[288618.  ],
[288870.  ],
[289106.  ],
[289313.  ],
[289518.  ],
[289714.  ],
[289911.  ],
[290125.  ],
[290346.  ],
[290584.  ],
[290820.  ],
[291072.  ],
[291321.  ],
[291574.  ],
[291807.  ],
[292008.  ],
[292192.  ],
[292368.  ],
[292561.  ],
[292779.  ],
[292997.  ],
[293223.  ],
```

```
[293463.   ],
[293719.   ],
[293971.   ],
[294230.   ],
[294466.   ],
[294694.   ],
[294914.   ],
[295105.   ],
[295287.   ],
[295490.   ],
[295704.   ],
[295936.   ],
[296186.   ],
[296440.   ],
[296707.   ],
[296972.   ],
[297207.   ],
[297431.   ],
[297647.   ],
[297854.   ],
[298060.   ],
[298281.   ],
[298496.   ],
[298739.   ],
[298996.   ],
[299263.   ],
[299554.   ],
[299835.   ],
[300094.   ],
[300340.   ],
[300574.   ],
[300802.   ],
[301021.   ],
[301254.   ],
[301483.   ],
[301739.   ],
[302004.   ],
[302267.   ],
[302546.   ],
[302807.   ],
[303054.   ],
[303287.   ],
[303506.   ],
[303711.   ],
[303907.   ],
[304117.   ],
[304323.   ],
```

```
[304556.  ],
[304798.  ],
[305045.  ],
[305309.  ],
[305554.  ],
[305786.  ],
[306004.  ],
[306208.  ],
[306402.  ],
[306588.  ],
[306787.  ],
[306984.  ],
[307206.  ],
[307439.  ],
[307685.  ],
[307946.  ],
[308189.  ],
[308418.  ],
[308633.  ],
[308833.  ],
[309027.  ],
[309212.  ],
[309191.22],
[309369.06],
[309548.5 ],
[309745.7 ],
[309957.78],
[310176.47],
[310399.97],
[310595.75],
[310781.72],
[310960.75],
[311113.38],
[311265.4 ],
[311436.25],
[311607.1 ],
[311791.22],
[311997.06],
[312205.38],
[312429.12],
[312644.16],
[312829.53],
[313009.72],
[313183.2 ],
[313338.97],
[313499.38],
[313667.12],
```

```
[313830.53],
[314017.6 ],
[314210.78],
[314422.34],
[314646.75],
[314853.97],
[315053.88],
[315232.75],
[315389.6 ],
[315520.16],
[315662.22],
[315817.84],
[315983.66],
[316171.03],
[316358.78],
[316580.3 ],
[316806.12],
[317022.28],
[317228.03],
[317411.56],
[317593.94],
[317753.88],
[317917.2 ],
[318089.22],
[318269.5 ],
[318464.16],
[318662.38],
[318893.78],
[319125.28],
[319353.72],
[319564.22],
[319746.16],
[319928.66],
[320074.5 ],
[320230.78],
[320402.28],
[320583.97],
[320773.56],
[320978.22],
[321202.44],
[321427.62],
[321653.03],
[321856.25],
[322043.06],
[322232.88],
[322398.1 ],
[322551.47],
```

```
[322721.22],
[322900.  ],
[323088.47],
[323291.03],
[323501.38],
[323709.84],
[323919.72],
[324106.53],
[324274.9 ],
[324438.2 ],
[324581.5 ],
[324713.97],
[324861.75],
[325019.2 ],
[325186.25],
[325367.62],
[325567.72],
[325766.03],
[325965.94],
[326142.62],
[326301.4 ],
[326454.12],
[326600.8 ],
[326736.7 ],
[326887.88],
[327048.72],
[327219.12],
[327403.9 ],
[327600.25],
[327794.78],
[327990.94],
[328163.88],
[328318.88],
[328467.8 ],
[328610.75],
[328742.84],
[328890.25],
[329047.3 ],
[329214.  ],
[329395.  ],
[329591.34],
[329785.88],
[329982.03],
[330154.94],
[330309.94]], dtype=float32)
```

[4]:
```
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

[5]:
```
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
print(len(train), len(test))
```

```
546 270
```

[6]:
```
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
        dataX, dataY = [], []
        for i in range(len(dataset)-look_back-1):
                a = dataset[i:(i+look_back), 0]
                dataX.append(a)
                dataY.append(dataset[i + look_back, 0])
        return numpy.array(dataX), numpy.array(dataY)
```

[7]:
```
# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
```

[8]:
```
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

[9]:
```
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)

# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
```

```
print('Test Score: %.2f RMSE' % (testScore))
```

```
Epoch 1/100
 - 1s - loss: 0.0483
Epoch 2/100
 - 1s - loss: 0.0137
Epoch 3/100
 - 1s - loss: 0.0049
Epoch 4/100
 - 1s - loss: 6.9705e-04
Epoch 5/100
 - 1s - loss: 6.8948e-05
Epoch 6/100
 - 1s - loss: 4.3064e-05
Epoch 7/100
 - 1s - loss: 3.7353e-05
Epoch 8/100
 - 1s - loss: 3.2209e-05
Epoch 9/100
 - 1s - loss: 2.5088e-05
Epoch 10/100
 - 1s - loss: 1.9909e-05
Epoch 11/100
 - 1s - loss: 1.5311e-05
Epoch 12/100
 - 1s - loss: 1.2591e-05
Epoch 13/100
 - 1s - loss: 1.1689e-05
Epoch 14/100
 - 1s - loss: 1.0876e-05
Epoch 15/100
 - 1s - loss: 1.1536e-05
Epoch 16/100
 - 1s - loss: 1.1269e-05
Epoch 17/100
 - 1s - loss: 1.1429e-05
Epoch 18/100
 - 1s - loss: 1.0387e-05
Epoch 19/100
 - 1s - loss: 1.1543e-05
Epoch 20/100
 - 1s - loss: 1.0639e-05
Epoch 21/100
 - 1s - loss: 1.0073e-05
Epoch 22/100
 - 1s - loss: 1.1216e-05
Epoch 23/100
```

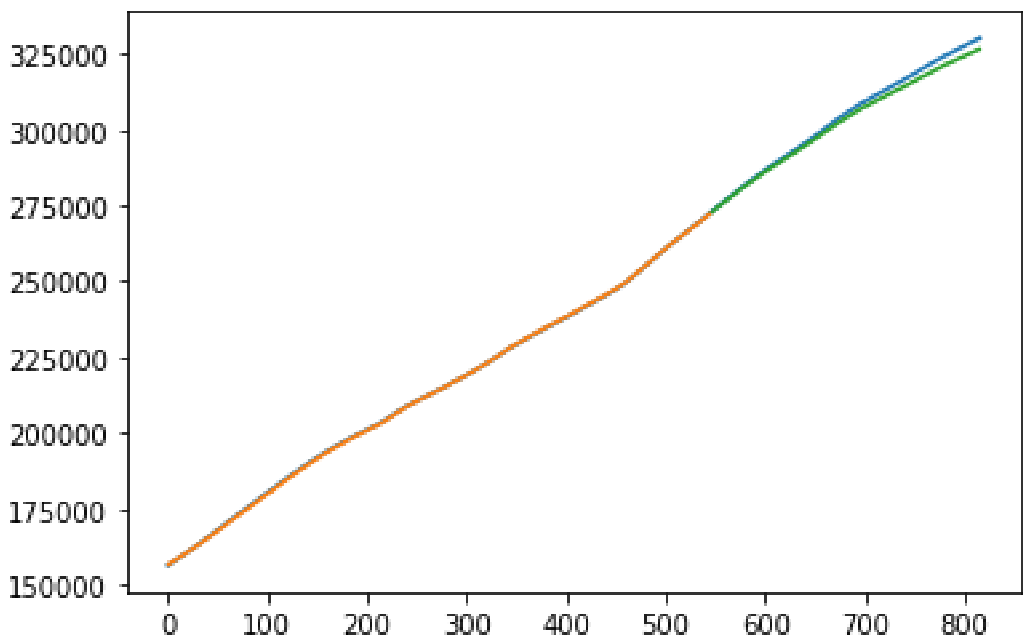
```
 - 1s - loss: 1.0232e-05
Epoch 24/100
 - 1s - loss: 1.3688e-05
Epoch 25/100
 - 1s - loss: 1.0160e-05
Epoch 26/100
 - 1s - loss: 9.1884e-06
Epoch 27/100
 - 1s - loss: 1.0009e-05
Epoch 28/100
 - 1s - loss: 1.0557e-05
Epoch 29/100
 - 1s - loss: 9.2110e-06
Epoch 30/100
 - 1s - loss: 8.6738e-06
Epoch 31/100
 - 1s - loss: 8.2968e-06
Epoch 32/100
 - 1s - loss: 9.4484e-06
Epoch 33/100
 - 1s - loss: 7.9738e-06
Epoch 34/100
 - 1s - loss: 9.8056e-06
Epoch 35/100
 - 1s - loss: 7.7512e-06
Epoch 36/100
 - 1s - loss: 8.0216e-06
Epoch 37/100
 - 1s - loss: 7.7554e-06
Epoch 38/100
 - 1s - loss: 8.7296e-06
Epoch 39/100
 - 1s - loss: 8.0023e-06
Epoch 40/100
 - 1s - loss: 7.5740e-06
Epoch 41/100
 - 1s - loss: 7.4975e-06
Epoch 42/100
 - 1s - loss: 7.1567e-06
Epoch 43/100
 - 1s - loss: 6.6152e-06
Epoch 44/100
 - 1s - loss: 6.8033e-06
Epoch 45/100
 - 1s - loss: 6.6685e-06
Epoch 46/100
 - 1s - loss: 6.2749e-06
Epoch 47/100
```

```
 - 1s - loss: 6.7397e-06
Epoch 48/100
 - 1s - loss: 6.9733e-06
Epoch 49/100
 - 1s - loss: 6.1789e-06
Epoch 50/100
 - 1s - loss: 5.6934e-06
Epoch 51/100
 - 1s - loss: 6.1403e-06
Epoch 52/100
 - 1s - loss: 5.8646e-06
Epoch 53/100
 - 1s - loss: 6.1443e-06
Epoch 54/100
 - 1s - loss: 5.8479e-06
Epoch 55/100
 - 1s - loss: 6.6950e-06
Epoch 56/100
 - 1s - loss: 5.3113e-06
Epoch 57/100
 - 1s - loss: 5.8066e-06
Epoch 58/100
 - 1s - loss: 5.2509e-06
Epoch 59/100
 - 1s - loss: 5.7224e-06
Epoch 60/100
 - 1s - loss: 5.3780e-06
Epoch 61/100
 - 1s - loss: 5.3430e-06
Epoch 62/100
 - 1s - loss: 5.1974e-06
Epoch 63/100
 - 1s - loss: 4.4067e-06
Epoch 64/100
 - 1s - loss: 4.8491e-06
Epoch 65/100
 - 1s - loss: 4.5305e-06
Epoch 66/100
 - 1s - loss: 4.2556e-06
Epoch 67/100
 - 1s - loss: 4.5702e-06
Epoch 68/100
 - 1s - loss: 4.3459e-06
Epoch 69/100
 - 1s - loss: 4.4365e-06
Epoch 70/100
 - 1s - loss: 4.0770e-06
Epoch 71/100
```

```
 - 1s - loss: 4.5513e-06
Epoch 72/100
 - 1s - loss: 3.8120e-06
Epoch 73/100
 - 1s - loss: 3.9103e-06
Epoch 74/100
 - 1s - loss: 4.1822e-06
Epoch 75/100
 - 1s - loss: 3.3623e-06
Epoch 76/100
 - 1s - loss: 4.1677e-06
Epoch 77/100
 - 1s - loss: 4.6122e-06
Epoch 78/100
 - 1s - loss: 3.5549e-06
Epoch 79/100
 - 1s - loss: 3.9210e-06
Epoch 80/100
 - 1s - loss: 3.0691e-06
Epoch 81/100
 - 1s - loss: 4.3342e-06
Epoch 82/100
 - 1s - loss: 3.3880e-06
Epoch 83/100
 - 1s - loss: 3.4242e-06
Epoch 84/100
 - 1s - loss: 3.0796e-06
Epoch 85/100
 - 1s - loss: 2.4279e-06
Epoch 86/100
 - 1s - loss: 3.3761e-06
Epoch 87/100
 - 1s - loss: 3.1242e-06
Epoch 88/100
 - 1s - loss: 3.1220e-06
Epoch 89/100
 - 1s - loss: 2.6207e-06
Epoch 90/100
 - 1s - loss: 2.8311e-06
Epoch 91/100
 - 1s - loss: 2.3217e-06
Epoch 92/100
 - 1s - loss: 2.6956e-06
Epoch 93/100
 - 1s - loss: 2.5967e-06
Epoch 94/100
 - 1s - loss: 2.3287e-06
Epoch 95/100
```

```
 - 1s - loss: 2.2899e-06
Epoch 96/100
 - 1s - loss: 2.6767e-06
Epoch 97/100
 - 1s - loss: 2.7100e-06
Epoch 98/100
 - 1s - loss: 2.3878e-06
Epoch 99/100
 - 1s - loss: 1.6431e-06
Epoch 100/100
 - 1s - loss: 1.6926e-06
Train Score: 208.27 RMSE
Test Score: 1905.51 RMSE
```

[10]:
```
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] =␣
 ↪testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



[ ]:

```
```

[ ]:

```
```

[0]:

```
```