Spring 2020

Home Sales as a Time Series Model

Noah R. Hellenthal

The University of Akron, nrh38@zips.uakron.edu

Time Series Analysis on Monthly Home Sales

Noah Hellenthal

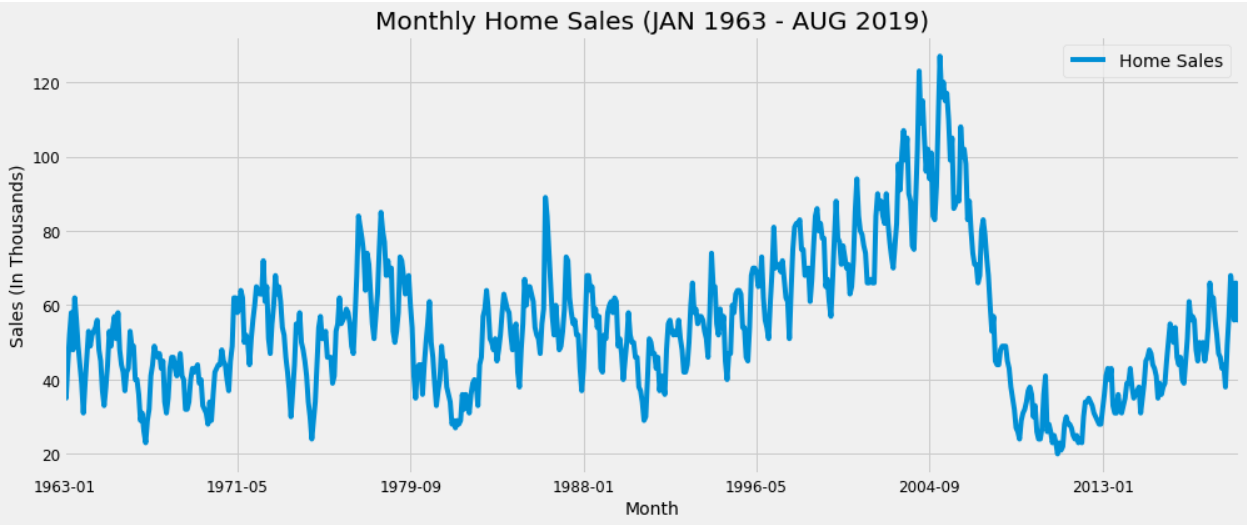
Introduction

When the Great Recession began in 2008, the prices of homes decreased greatly after enjoying around a decade of continuously rising prices. Between the years 2007 and 2008, median home sales fell about $25,000 and median home price fell around $20,000 (CNN Money). At the same time, foreclosures were skyrocketing and suddenly the number of houses on the market were overwhelming. Due to the collapse of subprime mortgages causing radical increases in monthly payments for homes, the quick turn in home prices and sales led to the numerous consequences of the Great Recession (Investopedia). Given the COVID-19 pandemic, we may be in the middle of our next "Great Recession" today. Once we use the history of home sales in order to craft a model of best fit, maybe we will see a downward trend in home sales like we did in 2008. With this model, we should be able to forecast a few years into the future and be able to get a rough estimation of monthly home sales.

Using data from the U.S. Census Bureau, I was able to obtain monthly home sales from January 1963 to August 2019. This data is formatted as a time series dataset where there are two variables: "Date" which is our Month/Year information for each instance and "Sales" which contains the monthly home sales across the USA in thousands. The "Date" variable is our independent index variable while "Sales" is the dependent response variable for the time series model. As an additional note, "Sales" specifies only the number of homes sold and does not include the number of homes put onto the market and not sold in the same month. Since the data is coming from the Census Bureau, I enjoyed the benefit of having no missing months or sales values within my dataset.

Throughout this report, I will be referencing results I obtained by using Python. With Python, I am able to conclude if my data has a strong seasonal component, confirm constant

variance throughout the dataset, ensure stationarity, fit ARIMA or sARIMA time series models (whichever is appropriate), and forecast for future monthly sales in the housing market. First, I will begin with plotting the totality of my dataset.

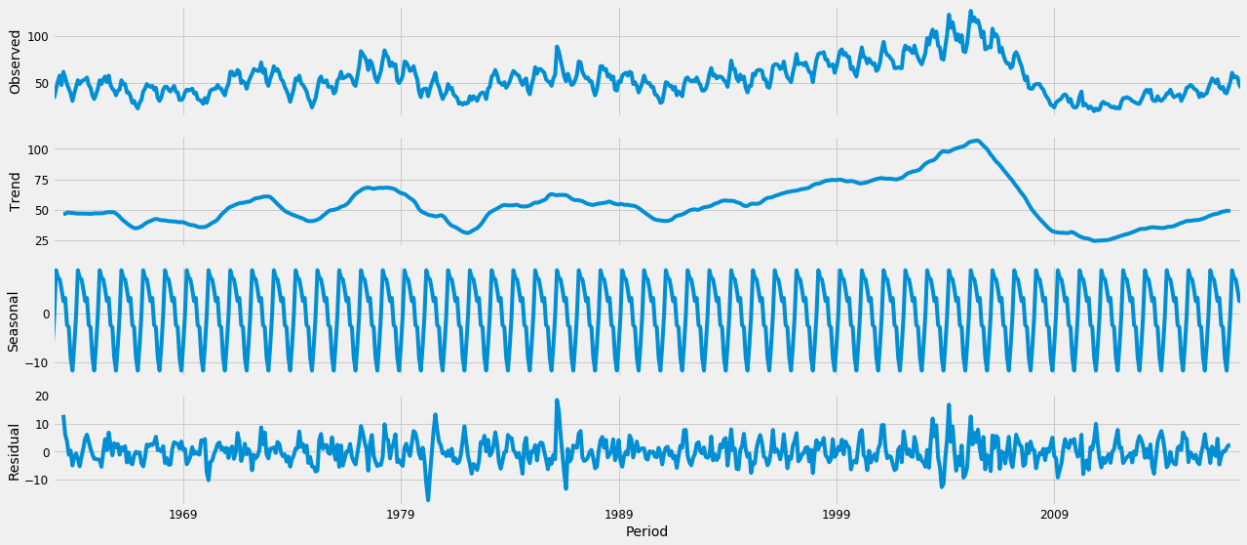


Just by looking at the plotted dataset, we can already notice trends in home sales including the steady incline and quick decline with the 2008 Recession. In addition, there seems to be some sense of seasonality in the plot. Patterns of increases and decreases are present as each year in the plot looks like it has the same basic shape. If seasonality is a major factor in our data, the sARIMA modeling approach will be preferred over conventional ARIMA modeling. To set the data up for modeling, it must be split into training and testing samples. Therefore, I split the last two years of the dataset off (September 2017 to August 2019) to create the testing sample and left the remainder to build the model.

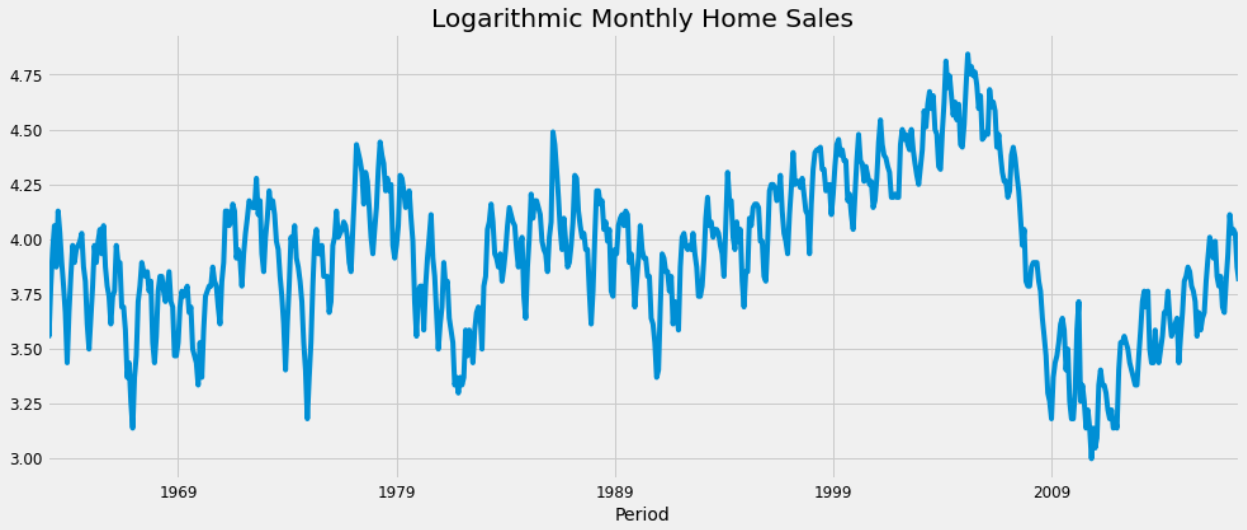
Seasonal Decomposition

Before any model building, I created a plot of decomposition in Python. This plot is a great visual that shows the breakdown of aspects like trend, seasonality, and error within my main dataset before I begin any analysis or model building. The plot below includes each of

these aspects as well as a copy of the plot of observed values. As an additional note, each of these plots are made from the training sample.



Immediately, it is easier to identify the periods where home sales increased and decreased due to the "Trend" line. With the seasonality and error removed, the trend line yields a smoother image of the dataset. Once again, the Great Recession is obvious. The "Seasonality" line reinforces the previous inclination that sARIMA modeling is more appropriate for this dataset. Each year seems to be giving the same shape where each month appears to plot the same value. Lastly, I can look at the "Residual" (error) line in order to determine if my data has an issue with nonconstant variance. This needs to be addressed before any model building so a more accurate model is obtained. To fix this issue, a logarithm could be applied to the dependent home sales variable. The residual plot does not lead me to believe that my variance is nonconstant. Despite some changes in variance due to recession, the variance stays pretty uniform throughout the dataset. Even though, I am sure that I should not apply a logarithm to my data, I will anyways just to get a plot of how the logarithm affects my data. Doing this gives the graph of the overall dataset below:



The plot of the logarithmic data gives a similar looking graph to the main dataset. However, now all of the data extremely pinched into a range of 3 to 4.75. This compression will eventually make my forecasts inaccurate because it will be difficult for the model built on this logarithmic data to make estimations for home sales in the thousands (when the transformation is removed). For now, I will continue with model building with non-logarithmic data and return to this step later if the diagnosis plots of the final model indicate that additional transformations are needed.

Stationarity

Applying the logarithm earlier was also an attempt to make the dataset stationary on variance. However, I determined that the data was already stationary on variance and concluded that the logarithm was not needed. With the concept of stationarity, we want a constant variance as well as a constant mean throughout the dataset. Looking back at the main plot, if the data can be marked as stationary on mean, then we would see a somewhat flat overall trend. Of course, the trend has its increases and decreases throughout including the dramatic drop in 2008, but the data looks like it may stay somewhat consistent and hover around a constant mean of 50.

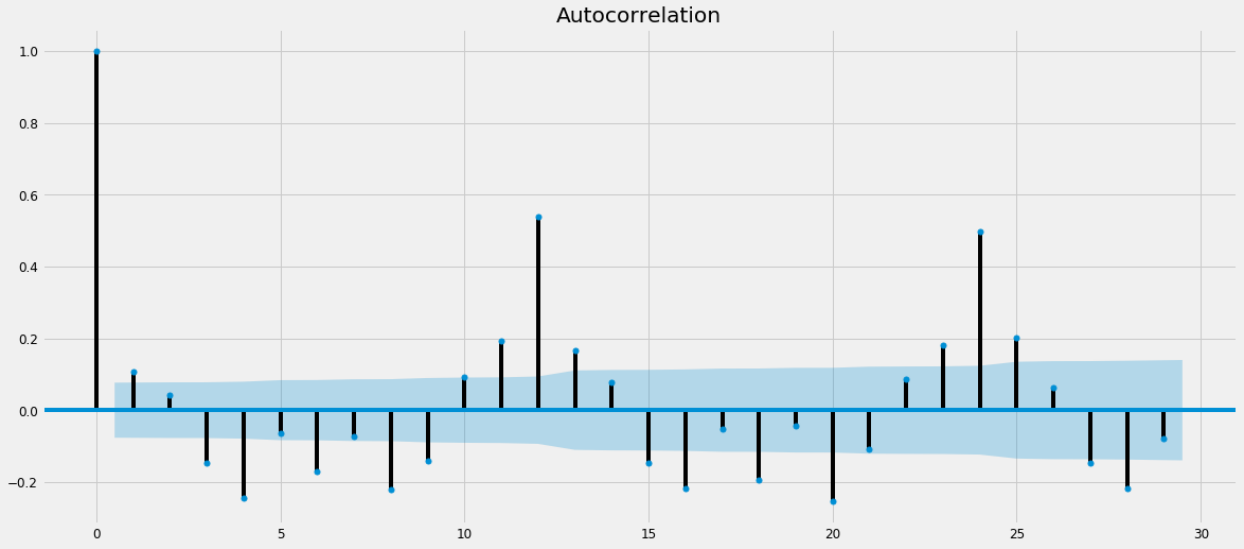
However, inference can only get us so far. Earlier, I had indicated that I would be taking the sARIMA modeling approach. These sARIMA models are compiled as sARIMA(p, d, q)(P, D, Q, m). Therefore, if I believe that my data is not already stationary on mean, then I will have to adjust the "d" parameter by increasing it one step at a time. Increasing this parameter "differences" the data and attempts to remove any trend that my exist within the main plot. There are statistical tests out there such as the ADF, KPSS, and PP tests that estimate the values of these parameter. However, I received rather inconclusive results from performing these types of tests in Python. The three tests suggested the value of "d" to be either zero or one.

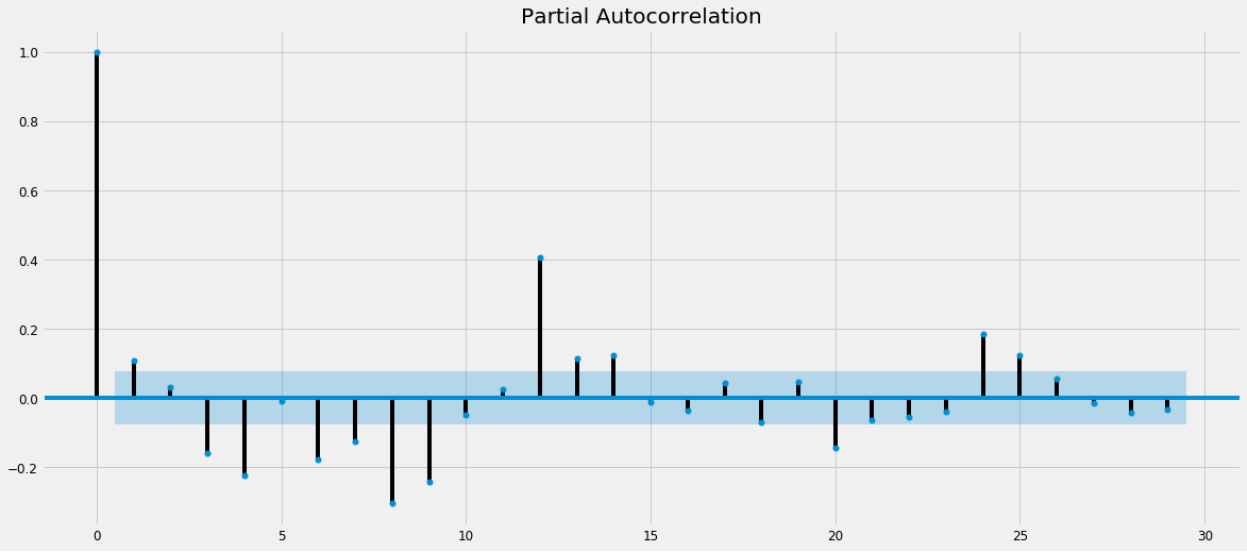
Another differencing parameter within the sARIMA model is "D." While D doesn't aim to make the dataset stationary on mean or variance, it will difference the data in order to remove seasonality if it is not equal to zero. While increasing "d" by one will difference the data by one, increasing "D" will difference the data by "m" which is included in the sARIMA model. I will be describing how to find this "m" parameter in the next section. Due to the inconclusive outcomes of this section, I am planning to craft four types of models. These types include d = 0 and D = 0, d = 1 and D = 0, d = 0 and D = 1, and lastly d = 1 and D = 1. The naming of the model variables will be specified in the "Model Building" section.

Seasonality Parameter

Beyond worrying about differencing, sARIMA models contain a seasonality parameter denoted by "m" which indicates the number of periods within a single season. Here, a single season would be a year. Since our data is monthly, I believe m will equal 12. Plotting the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) can give us two more visuals to confirm that m equals 12. These plots consist of bars called "lags" as well as a

shaded confidence interval. Below are the ACF and PACF of the differenced dataset to show significant lags:





We always ignore lag zero in these graphs, so the next largest identifiable lag is on twelve. Further down the X-axis we see another lag spike at twenty-four in both graphs. Before concluding that m should be 12, I also noticed how the lags seem large at 3, 4, 6, 8, and 9 in each plot. These spikes also make sense because there may be some quarterly or trimester seasonality in the data as well. However, I will conclude that m should be 12 because the lags that represent

the multiples of three and four are quickly muted the lag span of thirteen to twenty-four. Because these quarterly and trimester influences decrease significantly overtime while large spikes at multiples of twelve remain, I will not consider other values of m during my model fitting.

Model Building

In Python, I worked with two time series model building functions in order to craft my final model. First, I used "auto_arima" which builds multiple possible models for my training dataset and returns the best one it made. After using this function, I used a plain "ARIMA" function which allows me to specify every parameter in an sARIMA(p, d, q)x(P, D, Q, m) model. With the result of the auto_arima, I adjusted the parameters with this second function to see if I could build an even better model than auto_arima which uses stepwise procedures to obtain its result. While I adjusted the parameters, I would never increase them more than two steps because I would risk overfitting my data. If I allowed myself to overfit the data, I may get better values for model comparison criteria including AIC and BIC. However, my model would become more complex and may not be able to make accurate predictions when the testing dataset is applied.

When using auto_arima, you must specify the values of d, D, and m. Due to the iteration through d and D values I mentioned in the "Stationarity" section, I named my models, "Fit##" where the first "#" would be replaced with the value assigned for d and the second for D. To start model building, I created Fit00 with the auto_arima function where d and D both equal zero.

Fit00

Statespace Model Results

Dep. Variable:	y	No. Observations:	656
Model:	SARIMAX(2, 0, 2)x(2, 0, 1, 12)	Log Likelihood	-1919.166
Date:	Thu, 26 Mar 2020	AIC	3856.332
Time:	13:44:07	BIC	3896.707
Sample:	0	HQIC	3871.986
	- 656		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0035	0.006	0.553	0.580	-0.009	0.016
ar.L1	1.5281	0.327	4.680	0.000	0.888	2.168
ar.L2	-0.5381	0.317	-1.695	0.090	-1.160	0.084
ma.L1	-0.7430	0.335	-2.220	0.026	-1.399	-0.087
ma.L2	0.0424	0.095	0.446	0.656	-0.144	0.229
ar.S.L12	1.1718	0.048	24.431	0.000	1.078	1.266
ar.S.L24	-0.1784	0.046	-3.877	0.000	-0.269	-0.088
ma.S.L12	-0.9122	0.032	-28.908	0.000	-0.974	-0.850
sigma2	20.0185	1.008	19.868	0.000	18.044	21.993

Ljung-Box (Q):	46.84	Jarque-Bera (JB):	19.41
Prob(Q):	0.21	Prob(JB):	0.00
Heteroskedasticity (H):	1.12	Skew:	0.09
Prob(H) (two-sided):	0.41	Kurtosis:	3.82

While this summary window, we are able to see that an sARIMA(2, 0, 2)x(2, 0, 1, 12) model was built. However, even though this was the best model that auto_arima built with d and D both equal to zero, it may not be the best overall model. Looking at the table, the p-values (denoted P>|z|) for each parameter must be less than an alpha value of 0.05. Knowing this, it is easy to identify that AR(p=2) and MA(q=2) are insignificant. With this, I ran a few adjustments of this model to see if I could get anything better with the help of Python's ARIMA function. I ended up finding that an sARIMA(1, 0, 1)x(2, 0, 1, 12) model had all significant parameters. This adjustment is named "Fit00b" and the summary window of this model is posted below.

Fit00b – Adjustment made to auto_arima result

Statespace Model Results

Dep. Variable:	y	No. Observations:	656
Model:	SARIMAX(1, 0, 1)x(2, 0, 1, 12)	Log Likelihood	-1921.751
Date:	Thu, 26 Mar 2020	AIC	3857.503
Time:	12:26:11	BIC	3888.906
Sample:	0	HQIC	3869.678
	- 656		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0059	0.010	0.605	0.545	-0.013	0.025
ar.L1	0.9681	0.010	98.339	0.000	0.949	0.987
ma.L1	-0.1821	0.038	-4.797	0.000	-0.257	-0.108
ar.S.L12	1.1838	0.046	25.985	0.000	1.095	1.273
ar.S.L24	-0.1875	0.045	-4.197	0.000	-0.275	-0.100
ma.S.L12	-0.9222	0.025	-36.897	0.000	-0.971	-0.873
sigma2	19.7554	0.975	20.259	0.000	17.844	21.667

Ljung-Box (Q):	51.32	Jarque-Bera (JB):	15.48
Prob(Q):	0.11	Prob(JB):	0.00
Heteroskedasticity (H):	1.15	Skew:	0.10
Prob(H) (two-sided):	0.30	Kurtosis:	3.73

While the parameter situation improved in this model shown above, the model comparison criteria of AIC and BIC do not lead me to believe that this is a much better model. The AIC and BIC values of the model of best fit should be the lowest compared to all models being compared. Between these two models, the BIC decreases but the AIC does not. Instead of creating additional models where d = 0 and D = 0, I changed d to equal one and ran another auto_arima.

Fit10

Statespace Model Results

Dep. Variable:	y	No. Observations:	656
Model:	SARIMAX(1, 1, 1)x(2, 0, 1, 12)	Log Likelihood	-1917.322
Date:	Thu, 26 Mar 2020	AIC	3848.643
Time:	12:27:25	BIC	3880.036
Sample:	0	HQIC	3860.815
	- 656		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	-1.635e-05	0.001	-0.018	0.985	-0.002	0.002
ar.L1	0.5202	0.100	5.223	0.000	0.325	0.715
ma.L1	-0.7157	0.082	-8.746	0.000	-0.876	-0.555
ar.S.L12	1.1904	0.045	26.635	0.000	1.103	1.278
ar.S.L24	-0.1931	0.044	-4.398	0.000	-0.279	-0.107
ma.S.L12	-0.9344	0.025	-37.582	0.000	-0.983	-0.886
sigma2	19.7616	0.974	20.283	0.000	17.852	21.671

Ljung-Box (Q):	44.78	Jarque-Bera (JB):	13.55
Prob(Q):	0.28	Prob(JB):	0.00
Heteroskedasticity (H):	1.14	Skew:	-0.01
Prob(H) (two-sided):	0.32	Kurtosis:	3.70

By changing the differencing parameter, I(d), to equal one, I am working with the assumption that the data was not already stationary on mean. In addition, this model has a lower AIC and BIC that the earlier fit where d equaled zero. Once again, every parameter is significant in the new sARIMA(1, 1, 1)x(2, 0, 1, 12) model. I did build several other models off of this build with the ARIMA function, but each parameter increase or decrease I made led to insignificant parameters and a higher AIC and BIC.

Also, I made two other fits of Fit01 and Fit11. For both cases, I made the seasonal differencing parameter, D, equal to one while d could be zero or one. My model for Fit01 looks to be an improvement over Fit10 above.

Fit01

Statespace Model Results

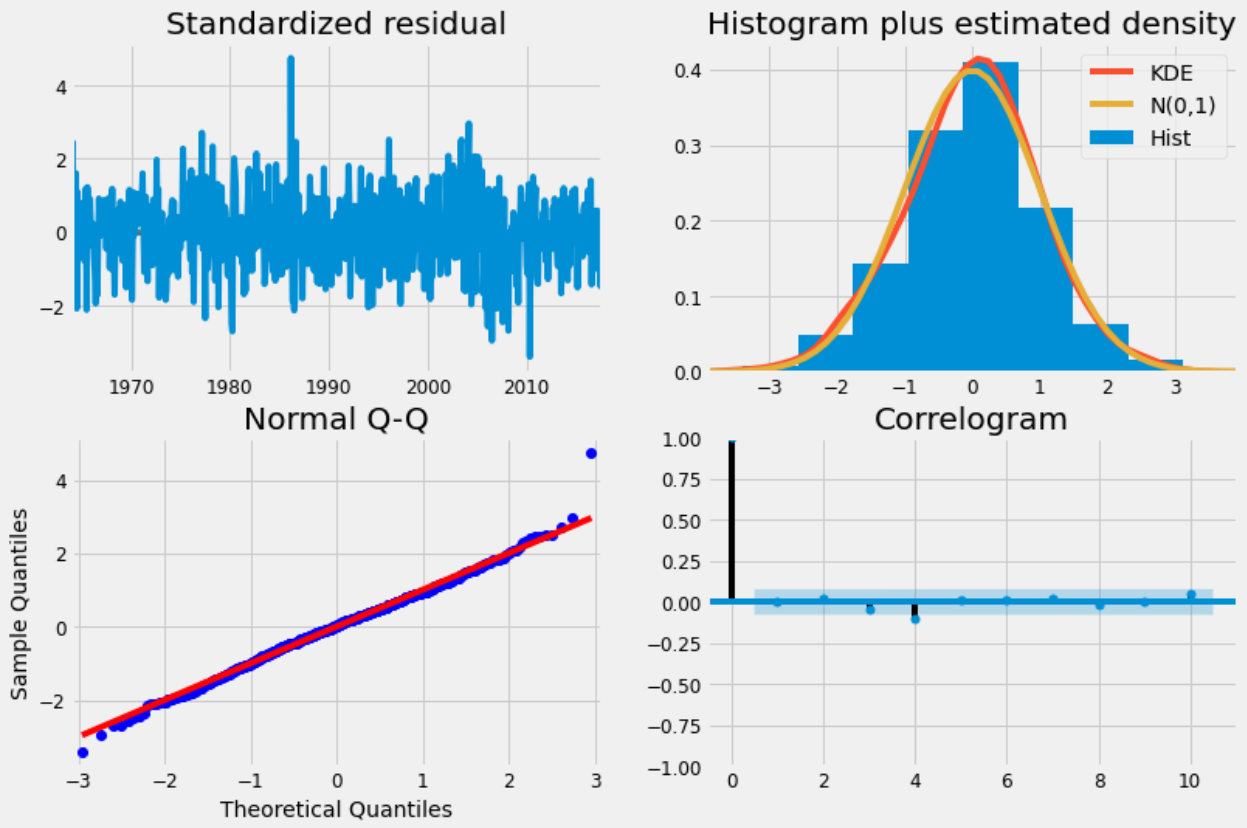
Dep. Variable:	y	No. Observations:	656
Model:	SARIMAX(1, 0, 2)x(1, 1, 1, 12)	Log Likelihood	-1884.083
Date:	Thu, 26 Mar 2020	AIC	3782.165
Time:	12:27:53	BIC	3813.439
Sample:	0	HQIC	3794.301
	- 656		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0014	0.010	0.134	0.894	-0.019	0.021
ar.L1	0.9780	0.009	107.342	0.000	0.960	0.996
ma.L1	-0.2046	0.039	-5.212	0.000	-0.282	-0.128
ma.L2	-0.0823	0.041	-2.024	0.043	-0.162	-0.003
ar.S.L12	0.2085	0.043	4.817	0.000	0.124	0.293
ma.S.L12	-0.9450	0.019	-49.487	0.000	-0.982	-0.908
sigma2	19.6591	0.959	20.494	0.000	17.779	21.539

Ljung-Box (Q):	49.34	Jarque-Bera (JB):	14.68
Prob(Q):	0.15	Prob(JB):	0.00
Heteroskedasticity (H):	1.17	Skew:	0.03
Prob(H) (two-sided):	0.24	Kurtosis:	3.74

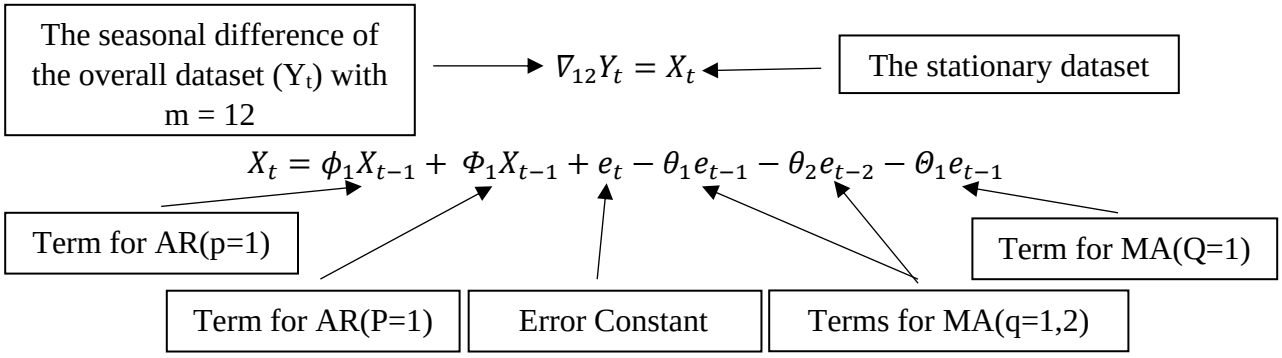
By changing D to equal 1 instead of d, I was able to build a model where the AIC and BIC saw a serious drop. My AIC here is 3782.165 where it was 3848.643 in Fit01. Fit11 models were also created; however, I saw no serious improvement in AIC and BIC and I worried that this “double differencing” was too extreme for my dataset and would lead to overfitting. Due to this extensive model building and changing parameters, I will conclude that Fit01 is the best possible model I can create off of my training dataset for Home Sales.

After building Fit01 for the training dataset, it is useful to plot certain diagnostic plots including a “Standardized Residual Plot” and a “Q-Q Plot” in order to see if the model fits certain assumptions made during model building. In Python, I was able to plot the residual diagnostics below:



The Standardized Residual plot in the top left of the image above pairs errors with time. Here, we are supposed to see random residual placements around the zero mark on the Y-axis. The plot above does seem to show this image; therefore, we can conclude that no trends exist within our errors. The histogram and the Q-Q plots are similar to each other in the way that they both show that our data does indeed have normal errors. The histogram shows this by having the shape of a Normal Distribution while the Q-Q plot shows it by having the large majority of its blue points line up along the diagonal red line. Lastly, the Correlogram shows us that the errors are uncorrelated because every lag (besides lag zero which is defaulted at one) stays within the bounds of the shaded confidence interval. Since each of the model building assumptions are met here, I can conclude that my Fit01 model is useful and I can now attempt forecasting.

Fit01 is indeed sARIMA(1, 0, 2)(1, 1, 1, 12), but this doesn't include the coefficients that is specific for forecasting Home Sales. Using the coefficients provided in the model summary for Fit01, I can build the following model:

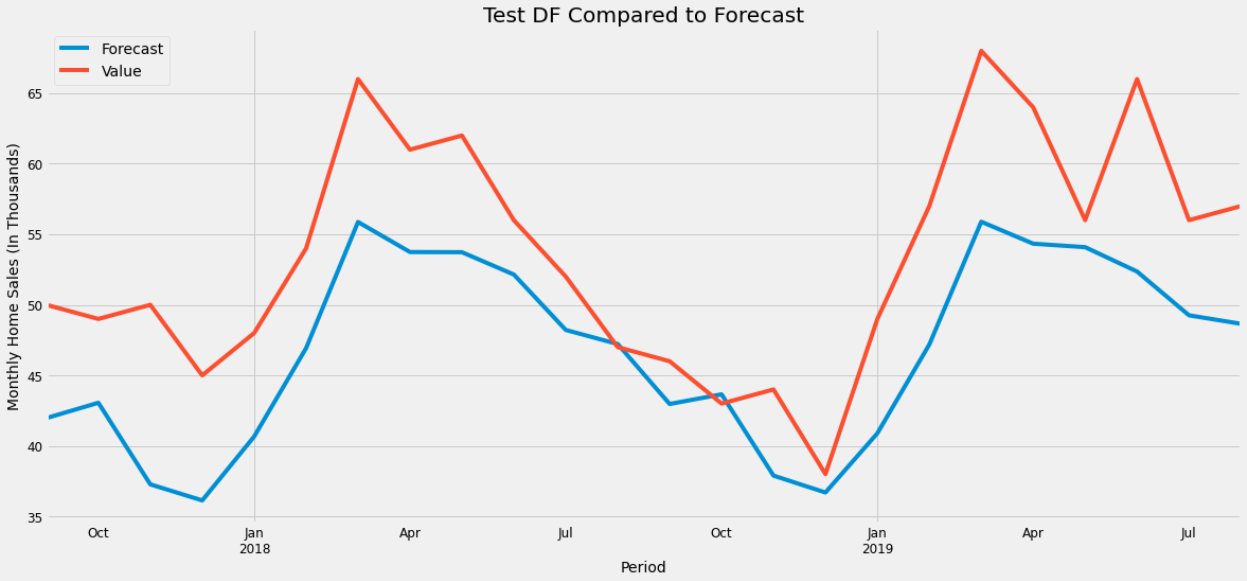


In the equations above, the coefficients are:

ϕ_1	Φ_1	θ_1	θ_2	Θ_1
0.9781	0.2082	-0.2047	-0.0823	-0.9447

Forecasting

Before forecasting for the future, I will forecast the two years that span my testing dataset. Once this is done, I will be able to compare how well my model performed by graphing the output of the forecast with the testing data.



In the plot above, the red line represents the testing dataset while the blue line represents what my sARIMA model is predicting from September 2017 to August 2019. While the forecast does not follow in line with the actual testing dataset with great accuracy, it does follow the seasonal trends pretty well. For both the forecast and the testing dataset, the lowest home sales are typically during November and December while the highest sales tend to be in March. The forecast seems to shift between being inaccurate to pretty close to the actual data. The two tables below give the actual values of the testing dataset versus the model results.

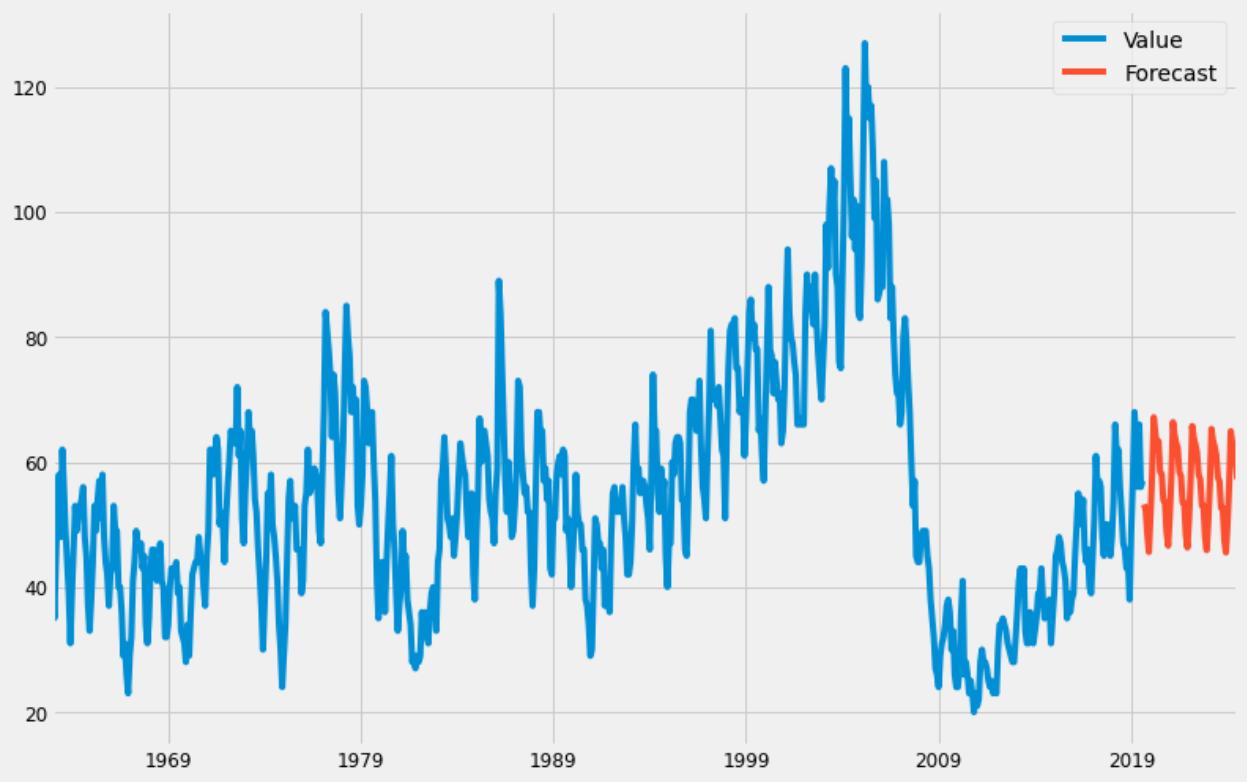
```
In [29]: testSales
Out[29]:
            Value
Period
2017-09-01     50
2017-10-01     49
2017-11-01     50
2017-12-01     45
2018-01-01     48
2018-02-01     54
2018-03-01     66
2018-04-01     61
2018-05-01     62
2018-06-01     56
2018-07-01     52
2018-08-01     47
2018-09-01     46
2018-10-01     43
2018-11-01     44
2018-12-01     38
2019-01-01     49
2019-02-01     57
2019-03-01     68
2019-04-01     64
2019-05-01     56
2019-06-01     66
2019-07-01     56
2019-08-01     57
```

```
In [28]: forecast_df
Out[28]:
2017-09-01    41.971769
2017-10-01    43.056831
2017-11-01    37.275390
2017-12-01    36.135526
2018-01-01    40.666520
2018-02-01    46.927665
2018-03-01    55.872158
2018-04-01    53.742408
2018-05-01    53.730099
2018-06-01    52.147059
2018-07-01    48.218062
2018-08-01    47.222087
2018-09-01    42.971302
2018-10-01    43.655155
2018-11-01    37.892741
2018-12-01    36.697855
2019-01-01    40.897645
2019-02-01    47.188384
2019-03-01    55.888866
2019-04-01    54.333034
2019-05-01    54.086757
2019-06-01    52.359636
2019-07-01    49.256093
2019-08-01    48.655683
```

The two tables above show the actual testing data that I obtained through the U.S. Census Bureau (left) and the forecasting dataset that was generated with the sARIMA model (right). The biggest difference looks to be in November 2017 where the actual data is over 12,000 sales higher that what the model predicted. However, towards the end of 2018, the model predicts more accurately. In October 2018, the model predicted only about 655 more sales than what the testing dataset shows. Perhaps, the model would have been more accurate in this forecast if the U.S. Census Bureau had not rounded their data.

Now that I have compared how the model performed alongside the testing data, it will be interesting to see how well the model will predict for the next five years in the future (September

2019 to August 2024 for this dataset). In Python, I was able to create the graph below which shows this five-year forecast:



In the graph above, the blue line represents the entire dataset (training and testing combined) while the red line is the five-year forecast. While the prediction doesn't reflect any of the wild jumps or drops that occurred in the past, it does show a slight downward trend. The seasonality is pretty consistent throughout each year, but the differences between the higher performing month of March and the lower performing month of December are predicted to decrease over time.

Conclusion

After testing for stationarity and seasonality, I was able to build an sARIMA model for Monthly Home Sales from data provided by the U.S. Census Bureau. I ran tests for stationarity

on variance, mean, and seasonality and I determined the appropriate seasonal factor for my model. With Python, I was able to build several different sARIMA models to test different parameters and compare AIC and BIC. In the end, I was able to create an sARIMA(1, 0, 2)x(1, 1, 1, 12) model which fulfilled all of the general assumptions a Statistician must make when building a Time Series model. While the forecasting of the last two years in the dataset did not adequately fit the actual data in the testing dataset, I still came pretty close at points and followed the same seasonality that the testing dataset exhibited. The Python code that I used to complete this analysis will be included below in an appendix. The model can only improve overtime as more data of Monthly Home Sales is made public. When predicting for future data, the model did in fact predict a slight downward trend in home sales. It will be interesting to compare this prediction to actual home sales over the next five years to determine if the market will indeed decline especially in light of the general economic downturn we have seen from COVID-19 affecting our current markets and lives.

References

6 October 2019. “Business and Industry.” *United States Census Bureau.* Retrieved October 6, 2019 from https://www.census.gov/econ/currentdata/dbsearch?program=RESSALES&startYear=1963&endYear=2019&categories=SOLD&dataType=TOTAL&geoLevel=US&notAdjusted=1&submit=GET+DATA&releaseScheduleId=

Boykin, Ryan. 25 October 2019. “The Great Recession’s Impact on the Housing Market.” *Investopedia.* Retrieved March 2, 2020 from https://www.investopedia.com/investing/great-recessions-impact-housing-market/

Christie, Les. 12 February 2009. “Home prices in record plunge.” *CNN Money.* Retrieved March 2, 2020 from https://money.cnn.com/2009/02/12/real_estate/Latest_median_prices/

Appendix

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 11 16:42:38 2020

@author: Noah Hellenthal
"""

import os #Used to change directory
import numpy as np #Needed for pmdarima
import pandas as pd #Needed for reading dataframe and for pmdarima

import pmdarima as pm #Needed for time series preliminaries and auto_arima
from pmdarima.arima import ARIMA #fits specific ARIMA models

import statsmodels.api as sm #Used for Data Decomposition
from pandas.tseries.offsets import DateOffset #Used to add 24 more months for forecasting
from sklearn.model_selection import train_test_split #Train/Test Split of Time Series Data

#General figure settings
import matplotlib
import matplotlib.pyplot as plt
#These global guidelines aren't required for anything, but they help with some visuals
plt.style.use("fivethirtyeight")
matplotlib.rcParams["axes.labelsize"] = 14
matplotlib.rcParams["xtick.labelsize"] = 12
matplotlib.rcParams["ytick.labelsize"] = 12
#These two lines make the seasonal decompose plot larger
```

```
from pylab import rcParams
rcParams["figure.figsize"] = 18,8

#Change Directory to open Home Sales Data
os.getcwd()
os.chdir("E:\\Honors Project\\New Home Sales")

#Load Dataset
df_plot = pd.read_excel("Home Data.xlsx") #Keep this one to reference plot
df = pd.read_excel("Home Data.xlsx") #Use this one through the changes
df.head()
df.index

#Plot Dataset (Use DataFrame)
df_plot.columns = ["Month", "Home Sales"]
df_plot = df_plot.set_index("Month")
ax = df_plot.plot(figsize=(15,6), title="Monthly Home Sales (JAN 1963 - AUG 2019)") #Run these 2 lines together
ax.set_ylabel("Sales (In Thousands)")


#Make Usable for Split and Decomposition
df.reset_index(inplace=True)
df["Period"] = pd.to_datetime(df["Period"])
df=df.set_index("Period")

#Make Train-Test Split
trainSales, testSales = train_test_split(df, test_size = 24, shuffle=False) #Test set is last 2 years
trainSales.drop(["index"], axis=1, inplace=True)
```

```
trainSales
testSales.drop(["index"], axis=1, inplace=True)
testSales

trainSales.plot(figsize=(15,6))
testSales.plot(figsize=(15,6))



#Decomposition Plot
decomp_add = sm.tsa.seasonal_decompose(trainSales, model="additive")
decomp_add.plot()



#General ACF and PACF
pm.utils.plot_acf(trainSales, alpha=0.05)
pm.utils.plot_pacf(trainSales, alpha=0.05)

#Stationary on Variance NOT REQUIRED
trainSales["Value"]
df_log = np.log(trainSales["Value"])
df_log.plot(figsize=(15,6), title = "Logarithmic Monthly Home Sales")

#This made the jumps a little more symmetric so we will keep it
decomp_log = sm.tsa.seasonal_decompose(trainSales, model="additive")
decomp_log.plot()

#In order to achieve stationary on mean, we could subtract the rolling average
#However, ARIMA does this for us through the d parameter
```

```
#Stationarizing (Differencing) the data

pm.arima.ndiffs(trainSales, alpha=0.05, test="adf", max_d=2) #Second and fourth parameters are defaults

pm.arima.ndiffs(trainSales, test="pp")

pm.arima.ndiffs(trainSales, test="kpss") #Default

#1, 0, 1 for d


#Seasonal Difference of the data

pm.arima.nsdiffs(trainSales, m=12, test="ocsb")

pm.arima.nsdiffs(trainSales, m=12, test="ch")

#0, 0 for D


#Differencing on d

df_stat_d = pm.utils.diff(trainSales, differences=1) #1 is the default


#ACF and PACF to predict seasonality (m)

pm.utils.plot_acf(df_stat_d, alpha=0.05)

pm.utils.plot_pacf(df_stat_d, alpha=0.05)

#Large spikes at lag 12 so there is likely a monthly seasonal trend


#ACF and PACF of D = 1

df_stat_D = pm.utils.diff(trainSales, differences=12) #D = 1

pm.utils.plot_acf(df_stat_D, alpha=0.05)

pm.utils.plot_pacf(df_stat_D, alpha=0.05)
```

```
#------------------Model Building----------------------------------------------

#Run a Auto-Arima with d=0, D=0

Fit00 = pm.auto_arima(trainSales, d=0, D=0, stepwise=True, error_action="ignore",
suppress_warnings=True, seasonal=True, m=12) #With d=0, D=0

Fit00.summary()

#sARIMA(2,0,2)(2,0,1,12) AIC = 3856.332 AR(p=2), MA(q=2) insignificant

#WARNING: This function may provide different results on different runs

#Now use the ARIMA function to tweek this

Fit00b = ARIMA(order=(1,0,1), seasonal_order=(2,0,1,12)).fit(trainSales)

Fit00b.summary() #AIC = 3857.503

#ACF and PACF of residuals

resid = Fit00b.resid()

pm.utils.plot_acf(resid, alpha=0.05)

pm.utils.plot_pacf(resid, alpha=0.05) #Both good

Fit10 = pm.auto_arima(trainSales, d=1, D=0, stepwise=True, error_action="ignore",
suppress_warnings=True, seasonal=True, m=12) #With d=1, D=0

Fit10.summary() #sARIMA(1,1,1)(2,0,1,12) AIC = 3848.643 All good

Fit10b = ARIMA(order=(1,1,1), seasonal_order=(2,0,1,12)).fit(trainSales)

Fit10b.summary() #Copy of above Summary good

#ACF and PACF of residuals

resid = Fit10b.resid()

pm.utils.plot_acf(resid, alpha=0.05)

pm.utils.plot_pacf(resid, alpha=0.05) #Both good (better than above)
```

```
Fit01 = pm.auto_arima(trainSales, d=0, D=1, stepwise=True, error_action="ignore",
suppress_warnings=True, seasonal=True, m=12) #With d=0, D=1

Fit01.summary()#sARIMA(1,0,4)(1,1,1,12) AIC = 3778.046 MA(q=2,3) insignificant

Fit01b = ARIMA(order=(1,0,2), seasonal_order=(1,1,1,12)).fit(trainSales)

Fit01b.summary() #AIC = 3782.165 Summary good


#ACF and PACF of residuals

resid = Fit01b.resid()

pm.utils.plot_acf(resid, alpha=0.05)

pm.utils.plot_pacf(resid, alpha=0.05) #Both good (ish?)


Fit11 = pm.auto_arima(trainSales, d=1, D=1, stepwise=True, error_action="ignore",
suppress_warnings=True, seasonal=True, m=12) #With d=1, D=1

Fit11.summary()#sARIMA(1,1,1)(1,1,1,12) AIC = 3777.107

Fit11b = ARIMA(order=(1,1,1), seasonal_order=(1,1,1,12)).fit(trainSales)

Fit11b.summary() #AIC = 3777.107 Summary good


#ACF and PACF of residuals

resid = Fit11b.resid()

pm.utils.plot_acf(resid, alpha=0.05)

pm.utils.plot_pacf(resid, alpha=0.05) #Both Good


#--------------------------------------------------------------------------------


#Refit ARIMA model with different package for forcasting

final = sm.tsa.statespace.SARIMAX(trainSales["Value"], order=(1,0,2),
seasonal_order=(1,1,1,12))

final_fit = final.fit()

print(final_fit.summary())
```

```
#Four in One plot
final_fit.plot_diagnostics(figsize=(12,8))

#Forecasting
future = [trainSales.index[-1]+ DateOffset(months=x)for x in range(0,25)]
future_df = pd.DataFrame(index=future[1:], columns=trainSales.columns)
future_df

future_total_df=pd.concat([trainSales, future_df])
future_total_df.tail(30)
future_total_df["Forecast"] = final_fit.predict(start=len(trainSales), end=(len(trainSales)+24), dynamic=True) #Exponential need since log was used
future_total_df[["Value", "Forecast"]].plot(figsize=(12,8))

#Get all forecasted values
forecast_df = future_total_df.Forecast.tail(24)
forecast_df

#Run below lines together
ax = forecast_df.plot()
testSales.plot(ax=ax)
plt.legend()
plt.ylabel("Monthly Home Sales (In Thousands)")
plt.title("Test DF Compared to Forecast");

#Fit final model to full dataset
total = sm.tsa.statespace.SARIMAX(df["Value"], order=(1,0,2), seasonal_order=(1,1,1,12))
total_fit = total.fit()
```

```
total_fit.summary()



#Forecasting Next Five Years (Future)
NextFive = [df.index[-1]+ DateOffset(months=x)for x in range(0,60)]
NextFive_df = pd.DataFrame(index=NextFive[1:], columns=df.columns)
NextFive_df

NextFive_total_df=pd.concat([df, NextFive_df])
NextFive_total_df.tail(70)
NextFive_total_df["Forecast"] = total_fit.predict(start=len(df), end=(len(df)+60), dynamic=True)
NextFive_total_df[["Value", "Forecast"]].plot(figsize=(12,8))
```