Objective: Predicting housing prices

The main objective in this notebook is to show proof of concept in accurately predicting housing prices using timeseries modeling for a real estate investment firm.

```
#importing packages and data
import pandas as pd
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import math
from math import sqrt

%matplotlib inline

import statsmodels.api as sm
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_predict
from statsmodels.tsa.arima_process import arma_generate_sample

from sklearn.metrics import mean_squared_error


df = pd.read_csv('data/zillow_data.csv.zip')
```

Data Understanding and Limitations

The data used in this project is from Zillow's housing data and is the median house price of zipcodes across the U.S.

General Overview:

- Over 14k unique zipcodes, this goes down to under 200 after we decided on which zip codes to use for our modeling
- Time range is from 1996 to April, 2018
- Prices are the median house price in the zipcode
- Data is not current and does not include anything after April 2018
- Data is very general, and we generalize even more after deciding on which zipcodes to use
- Our models are all built on the mean of the median of all the zipcode in our metros that we selected

```
#here we are filtering out our zipcodes to just look at the top 500 in
sizerank
df['Top500'] = df['SizeRank'].apply(lambda x: True if x <= 500 else
False)

df = df.loc[df.Top500, :]

#creating growth rate metric and sorting to see top zips
df['GrowthRate'] = df['2018-04'] / df['1996-04'] - 1
df.sort_values('GrowthRate', ascending = False)[:5]
```

```
     RegionID  RegionName         City State       Metro  \
117     62022       11211     New York    NY    New York
475     62027       11216     New York    NY    New York
191     60639        7302  Jersey City    NJ    New York
106     62026       11215     New York    NY    New York
258     66125       20001   Washington    DC  Washington

               CountyName  SizeRank   1996-04   1996-05   1996-06  ...
\
117                 Kings       118  133200.0  132900.0  132500.0  ...

475                 Kings       476  146100.0  146600.0  147200.0  ...

191                Hudson       192  137200.0  137800.0  138500.0  ...

106                 Kings       107  225700.0  227500.0  229400.0  ...

258  District of Columbia       259   92000.0   92600.0   93200.0  ...


     2017-09  2017-10  2017-11  2017-12  2018-01  2018-02  2018-03
2018-04  \
117  1424700  1435300  1440500  1463100  1496100  1531100  1581900
1623700
475  1553100  1567700  1559700  1545700  1540200  1553600  1578400
1598700
191  1411000  1435900  1446300  1447800  1454900  1453900  1439500
1427300
106  2244400  2266100  2275800  2287100  2288900  2265300  2244900
2243900
258   771200   773300   777600   780500   781600   785500   791400
793300

     Top500  GrowthRate
117    True   11.189940
475    True    9.942505
191    True    9.403061
106    True    8.941958
```

```
258    True    7.622826

[5 rows x 274 columns]
```

Metro Selection

So below we have the metros with the most zipcodes in our Top 500 SizeRank

```
#checking which metros have the most top500 sizerank zipcodes
#we are not using houston and atlanta as they have lower growth rates
than then rest
df.value_counts('Metro')[:10]
```

```
Metro
New York                          46
Los Angeles-Long Beach-Anaheim    32
Chicago                           27
Atlanta                           23
Miami-Fort Lauderdale             20
Houston                           20
Dallas-Fort Worth                 19
Washington                        14
San Francisco                     14
Charlotte                         11
dtype: int64
```

```
#showing that almost half of zip codes in our top 500 are represented
in the top 10 metros
df.value_counts('Metro')[:10].sum()
```

```
226
```

```
#making dfs for our Metros and checking mean growthrate
#going to use NYC, Chi, LA, MIA, and DFW highest growth rates with
atleast 20 zipcodes
df_nyc = df[df['Metro'] == 'New York']
df_chi = df[df['Metro'] == 'Chicago']
df_la = df[df['Metro'] == 'Los Angeles-Long Beach-Anaheim']
df_dfw = df[df['Metro'] == 'Dallas-Fort Worth']
df_mia = df[df['Metro'] == 'Miami-Fort Lauderdale']
```

Growth Rate

So we are going to select our metros that have the best combination of growth rate and zipcodes in the top 500. The reason we are not using Houston and Atlanta is they both have lower growth rates than DFW and have comprable zipcode counts.

The bottom 3 metros of D.C. San Francisco, and Charlotte all had better growth rates than DFW and CHI, but we felt they did not have enough zipcodes in the top 500.

```python
#showing growth rate in descending order
print('NYC:')
print(df_nyc['GrowthRate'].mean())
print('LA:')
print(df_la['GrowthRate'].mean())
print('MIA:')
print(df_mia['GrowthRate'].mean())
print('CHI:')
print(df_chi['GrowthRate'].mean())
print('DFW:')
print(df_dfw['GrowthRate'].mean())
```

```
NYC:
3.7712542878046262
LA:
3.2231950761389916
MIA:
1.9554887689343101
CHI:
1.3872262487454181
DFW:
0.9493054726931407
```

Functions

Below we have some functions we are going to use

- The first one is for converting our dfs to datetime
- the second is for melting our zipcodes creating a mean of the median of the zipcodes in our dfs
- the third one is a stationarity/dickey-fuller test check before we begin modeling

```python
# function for datetimes
def get_datetimes(df):
    """
    Takes a dataframe:
    returns only those column names that can be converted into
datetime objects
    as datetime objects.
    NOTE number of returned columns may not match total number of
columns in passed dataframe
    """


    return pd.to_datetime(df.columns.values[7:272], format='%Y-%m')



#funtion for melting dataframes
```

```
def melt_data(df):
    """
    Takes the zillow_data dataset in wide form or a subset of the
zillow_dataset.
    Returns a long-form datetime dataframe
    with the datetime column names as the index and the values as the
'values' column.

    If more than one row is passes in the wide-form dataset, the
values column
    will be the mean of the values from the datetime columns in all of
the rows.
    """

    melted = pd.melt(df, id_vars=['RegionName', 'RegionID',
'SizeRank', 'City', 'State', 'Metro', 'CountyName', 'Top500',
'GrowthRate'], var_name='time')
    melted['time'] = pd.to_datetime(melted['time'],
infer_datetime_format=True)
    melted = melted.dropna(subset=['value'])
    return melted.groupby('time').aggregate({'value':'mean'})



#function for checking stationarity
def stationarity_check(df):


    # Calculate rolling statistics
    roll_mean = df.rolling(window=8, center=False).mean()
    roll_std = df.rolling(window=8, center=False).std()

    # Perform the Dickey Fuller Test
    dftest = adfuller(df['value'])

    # Plot rolling statistics:
    fig = plt.figure(figsize=(12,6))
    plt.plot(df, color='blue',label='Original')
    plt.plot(roll_mean, color='red', label='Rolling Mean')
    plt.plot(roll_std, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    # Print Dickey-Fuller test results
    print('Results of Dickey-Fuller Test: \n')

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

```
Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)

    return None
```

Converting to datetime

```
#datetime in dataframes
get_datetimes(df_nyc)
get_datetimes(df_la)
get_datetimes(df_mia)
get_datetimes(df_dfw)
get_datetimes(df_chi)
```

```
DatetimeIndex(['1996-04-01', '1996-05-01', '1996-06-01', '1996-07-01',
               '1996-08-01', '1996-09-01', '1996-10-01', '1996-11-01',
               '1996-12-01', '1997-01-01',
               ...
               '2017-07-01', '2017-08-01', '2017-09-01', '2017-10-01',
               '2017-11-01', '2017-12-01', '2018-01-01', '2018-02-01',
               '2018-03-01', '2018-04-01'],
              dtype='datetime64[ns]', length=265, freq=None)
```

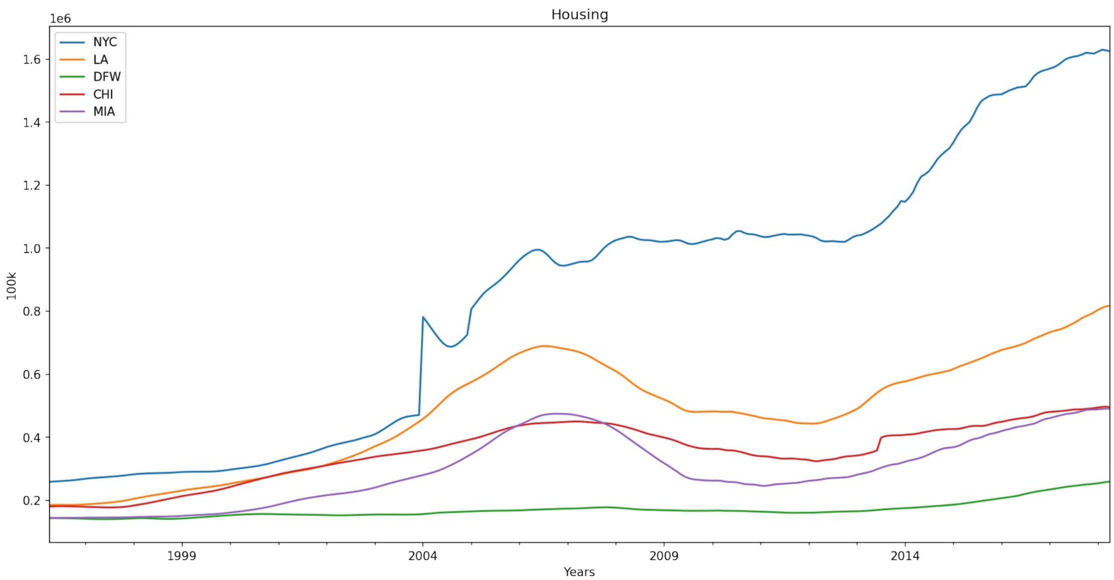
Melting our dataframes

```
#melting dataframes
df_nyc = melt_data(df_nyc)
df_la = melt_data(df_la)
df_dfw = melt_data(df_dfw)
df_mia = melt_data(df_mia)
df_chi = melt_data(df_chi)
```

Showing the price growth of our dataframes

```
# Here is a plot showing all our Metros mean prices from 1996 to 2018
plt.figure(figsize=(16, 8), dpi=150)


df_nyc['value'].plot(label='NYC')
df_la['value'].plot(label='LA')
df_dfw['value'].plot(label='DFW')
df_chi['value'].plot(label='CHI')
df_mia['value'].plot(label='MIA')
# adding title to the plot
plt.title('Housing')

# adding Label to the x-axis
```

```
plt.xlabel('Years')
plt.ylabel('100k')


# adding legend to the curve
plt.legend();
```

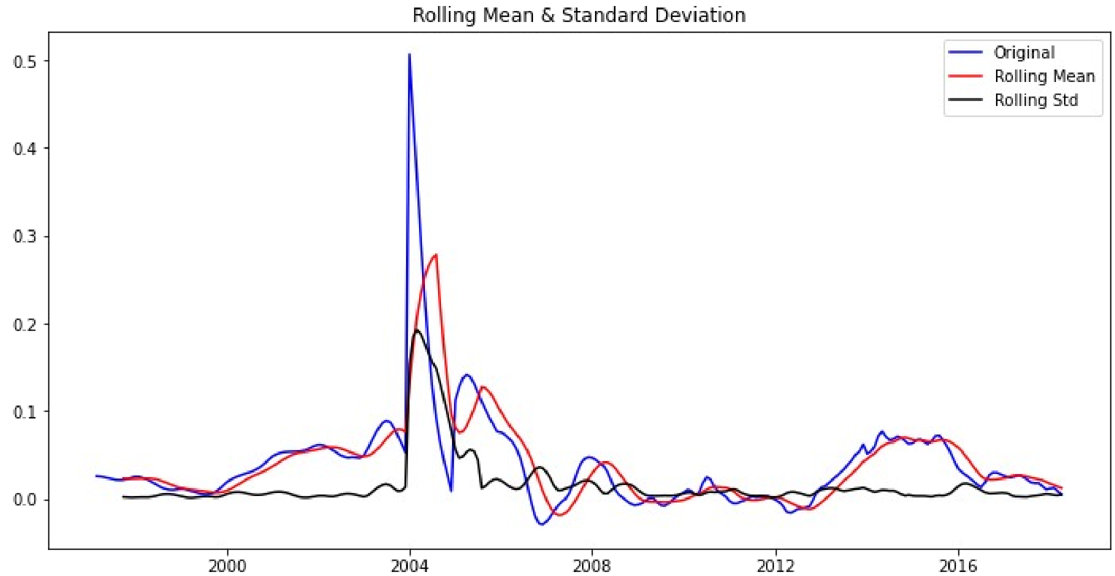


NYC ARIMA MODEL

So here is our NYC Model stationarity check, as you can see our pvalue is well below our .05 threshold so we will move forward to modeling this one.

A secondary metric here is our test statistic which we want below the critical values, this one is below all the critical values which is exactly what we want to see.

```
# Making data stationary
roll_mean_nyc = np.log(df_nyc).rolling(window=12).mean()
data_minus_roll_mean_nyc = np.log(df_nyc) - roll_mean_nyc
data_minus_roll_mean_nyc.dropna(inplace=True)
stationarity_check(data_minus_roll_mean_nyc)
```




```
Results of Dickey-Fuller Test:

Test Statistic                  -4.358815
p-value                          0.000351
#Lags Used                       0.000000
Number of Observations Used    253.000000
Critical Value (1%)             -3.456464
Critical Value (5%)             -2.873033
Critical Value (10%)            -2.572895
dtype: float64
```

NYC ARIMA

Below we have our train test split, our arima model's p,d,q orders, as well as our predictions for visualizations farther down.

We mainly focused on keeping our models AR and MA coef more than .2 away from zero and the pvalues under .05, as you can see this one meets those limitations.

Some of our secondary focuses were on Heteroskedasticity, Skew, and Kurtosis.

```
#Model DF, train/test, storing model pred for visuals, and model
summary
model_nyc = data_minus_roll_mean_nyc
train = model_nyc[:-13]
test = model_nyc[-13:]

arima_nyc = ARIMA(train, order=(3,3,1))
arima_nyc_fit = arima_nyc.fit()
pred_nyc = arima_nyc_fit.predict(start="2017-04-01", end="2018-04-01")
pred = arima_nyc_fit.get_prediction(start="2015-04-01", end="2018-04-
```

```
01", dynamic=False)
pred_ci = pred.conf_int()
print(arima_nyc_fit.summary())
```

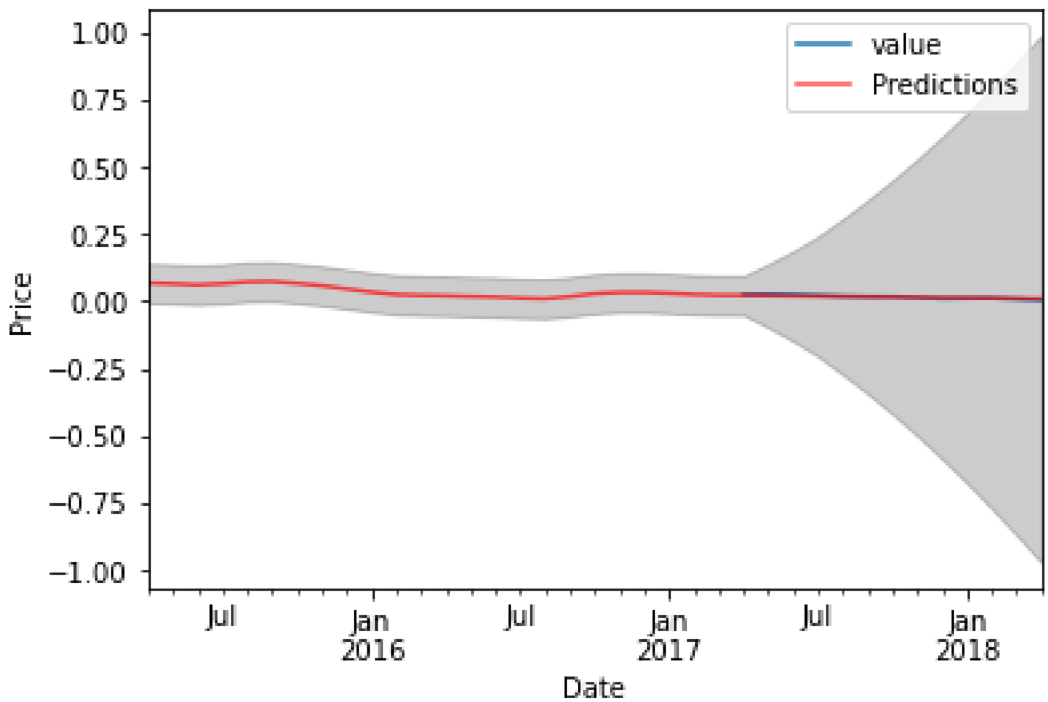
```
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
tsa\base\tsa_model.py:524: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
tsa\base\tsa_model.py:524: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
tsa\base\tsa_model.py:524: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  warnings.warn('No frequency information was'

                               SARIMAX Results

======================================================================
========
Dep. Variable:                  value   No. Observations:
241
Model:                 ARIMA(3, 3, 1)   Log Likelihood
440.799
Date:                Mon, 31 Oct 2022   AIC
-871.598
Time:                        17:50:38   BIC
-854.236
Sample:                    03-01-1997   HQIC
-864.601
                         - 03-01-2017

Covariance Type:                  opg

======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
ar.L1         -0.7241      0.021    -34.466      0.000      -0.765
-0.683
ar.L2         -0.4675      0.029    -16.269      0.000      -0.524
-0.411
ar.L3         -0.2337      0.025     -9.233      0.000      -0.283
-0.184
ma.L1         -0.9984      0.300     -3.331      0.001      -1.586
-0.411
sigma2         0.0014      0.000      3.569      0.000       0.001
```

```
0.002
===========================================================================
=============
Ljung-Box (L1) (Q):                   0.55   Jarque-Bera (JB):
101130.92
Prob(Q):                              0.46   Prob(JB):
0.00
Heteroskedasticity (H):               3.32   Skew:
6.73
Prob(H) (two-sided):                  0.00   Kurtosis:
103.09
===========================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
```

```
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
base\model.py:566: ConvergenceWarning: Maximum Likelihood optimization
failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

Visualization showing predications, confidence interval, and actual values

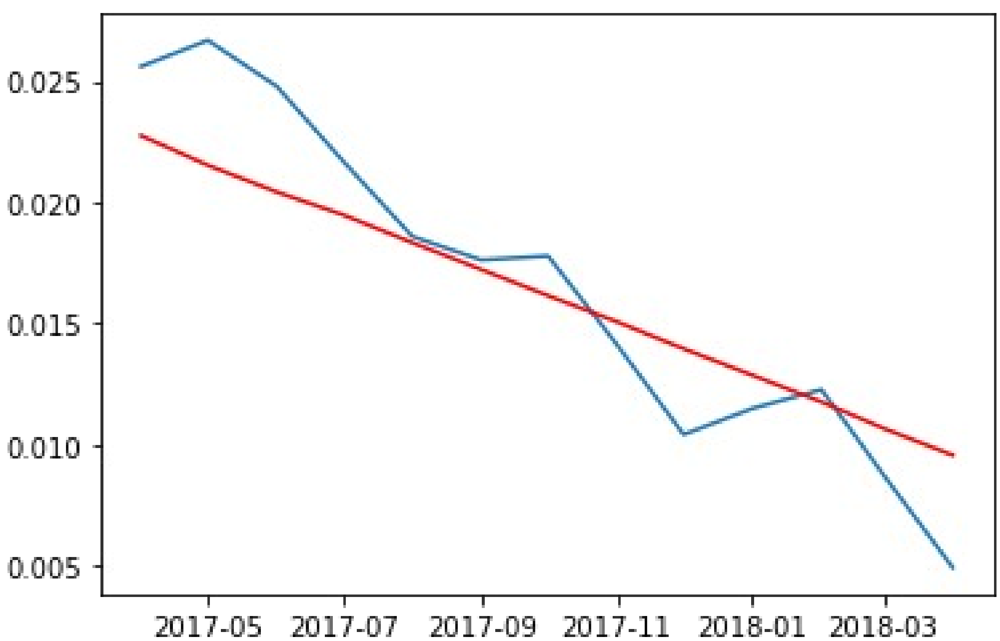
```
#Predictions, CI, and actual values visual
ax = test.plot(label='Actual values')
pred.predicted_mean.plot(ax=ax, label='Predictions', alpha=.7,
color='red')

ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)

ax.set_xlabel('Date')
ax.set_ylabel('Price')
plt.legend()

plt.show()
```



Zoomed in predictions and actual values

```
#zoomed in pred and value
plt.plot(test)
plt.plot(pred_nyc, color= 'red')
```

```
[<matplotlib.lines.Line2D at 0x23fb6f612e0>]
```



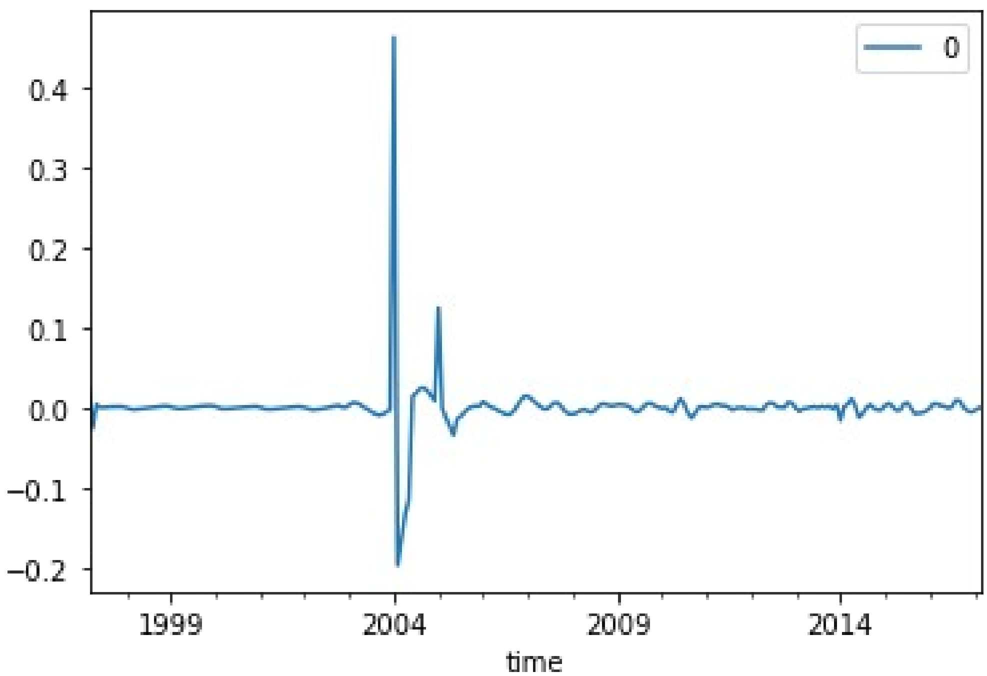
MSE, RMSE, and Residual check

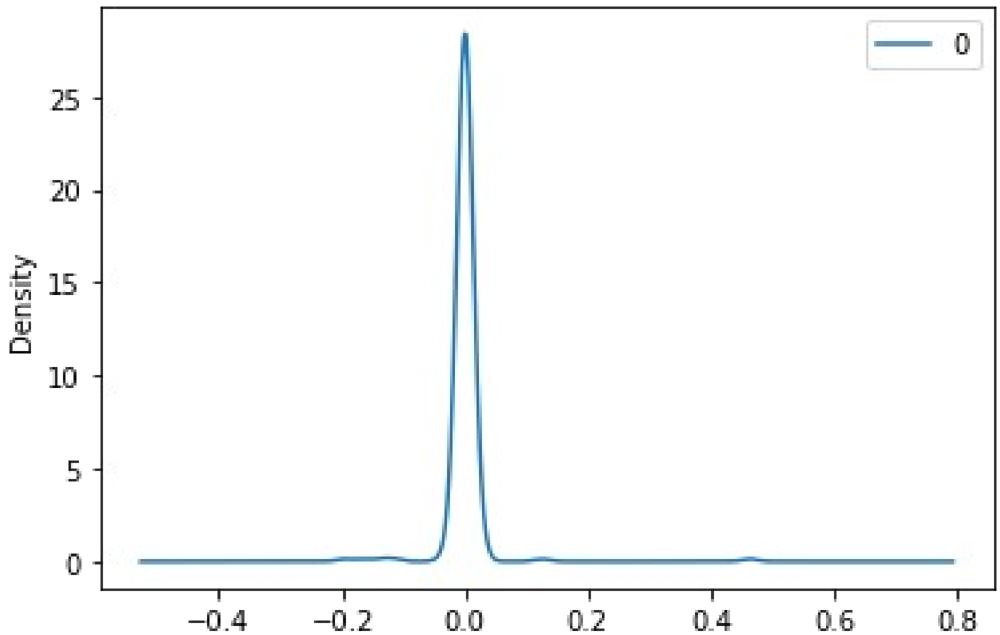
All of these are similar metrics and once again we are trying to get them as close to zero as possible

```
#MSE and RMSE check
expected = test
predictions = pred_nyc
mse = mean_squared_error(expected, predictions)
rmse = sqrt(mse)
print('MSE: %f' % mse)
print('RMSE: %f' % rmse)
```

```
MSE: 0.000008
RMSE: 0.002805
```

```
#residual check
residuals = pd.DataFrame(arima_nyc_fit.resid)
residuals.plot()
plt.show()

residuals.plot(kind='kde')
plt.show()

print(residuals.describe())
```



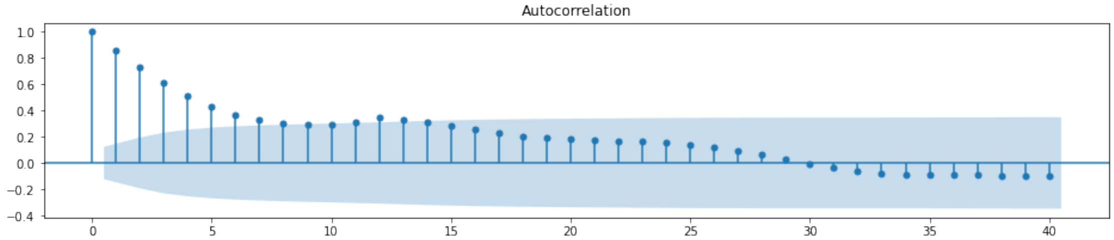


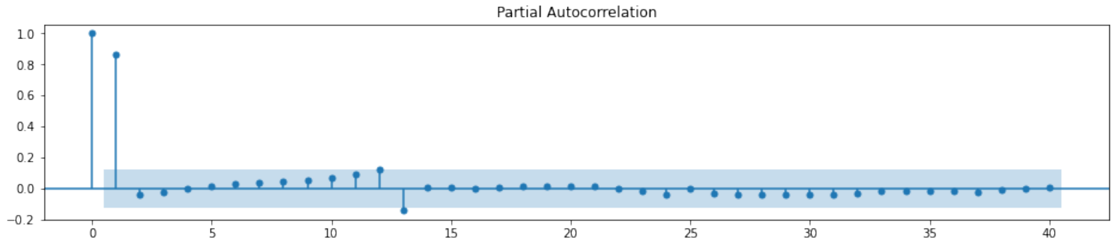

```
                0
count  241.000000
mean    -0.000015
std      0.037494
min     -0.197097
25%     -0.002194
50%      0.000232
75%      0.002258
max      0.462894
```

ACF and PACF

These ACF and PACF are for helping us tune our models p and q, while we might have not used the exact lags on this chart, it provided a starting point to tune our models and work from.

```
#ACF and PACF for model tuning
fig, ax = plt.subplots(figsize=(16,3))
plot_acf(model_nyc, ax=ax, lags=40);

fig, ax = plt.subplots(figsize=(16,3))
plot_pacf(model_nyc, ax=ax, lags=40);
```



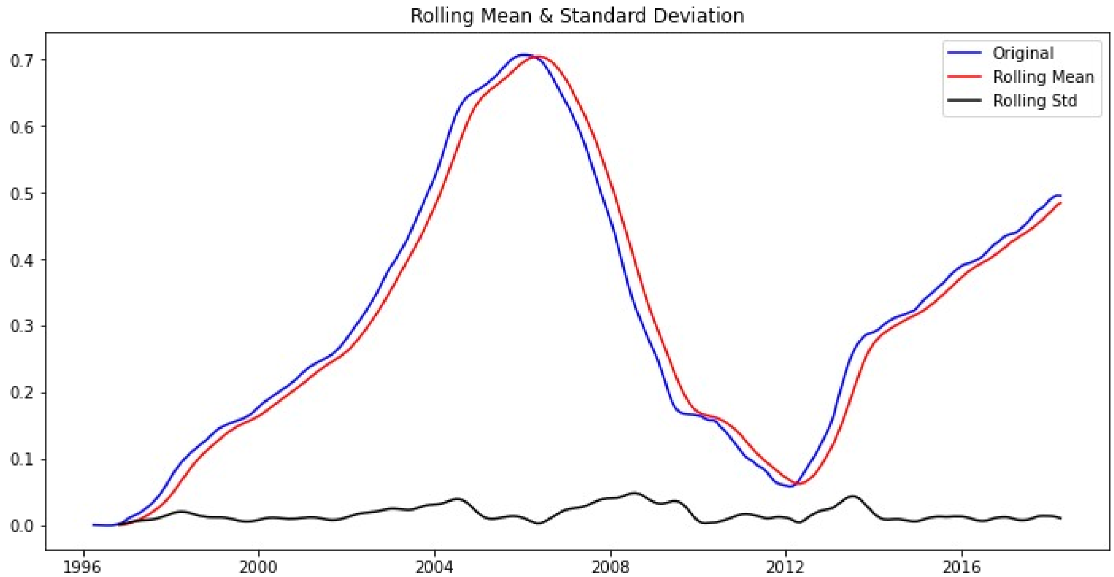


LA Model

So here is our LA Model stationarity check, as you can see our pvalue is below our .05 threshold.

A secondary metric here is our test statistic which we want below the critical values, while this one is not below the Critical Value of 1% it is below the other two so we will move forward.

```
#Model stationarity
exp_roll_mean_la = np.log(df_la).ewm(alpha = .005).mean()
data_minus_exp_roll_mean_la = np.log(df_la) - exp_roll_mean_la
stationarity_check(data_minus_exp_roll_mean_la)
```



```
Results of Dickey-Fuller Test:

Test Statistic                   -2.863459
p-value                           0.049765
#Lags Used                       15.000000
Number of Observations Used     249.000000
Critical Value (1%)              -3.456888
Critical Value (5%)              -2.873219
Critical Value (10%)             -2.572994
dtype: float64
```

LA ARIMA

Below we have our train test split, our arima model's p,d,q orders, as well as our predictions for visualizations farther down.

We mainly focused on keeping our models AR and MA coef more than .2 away from zero and the pvalues under .05, as you can see this one meets those limitations.

Some of our secondary focuses were on Heteroskedasticity, Skew, and Kurtosis.

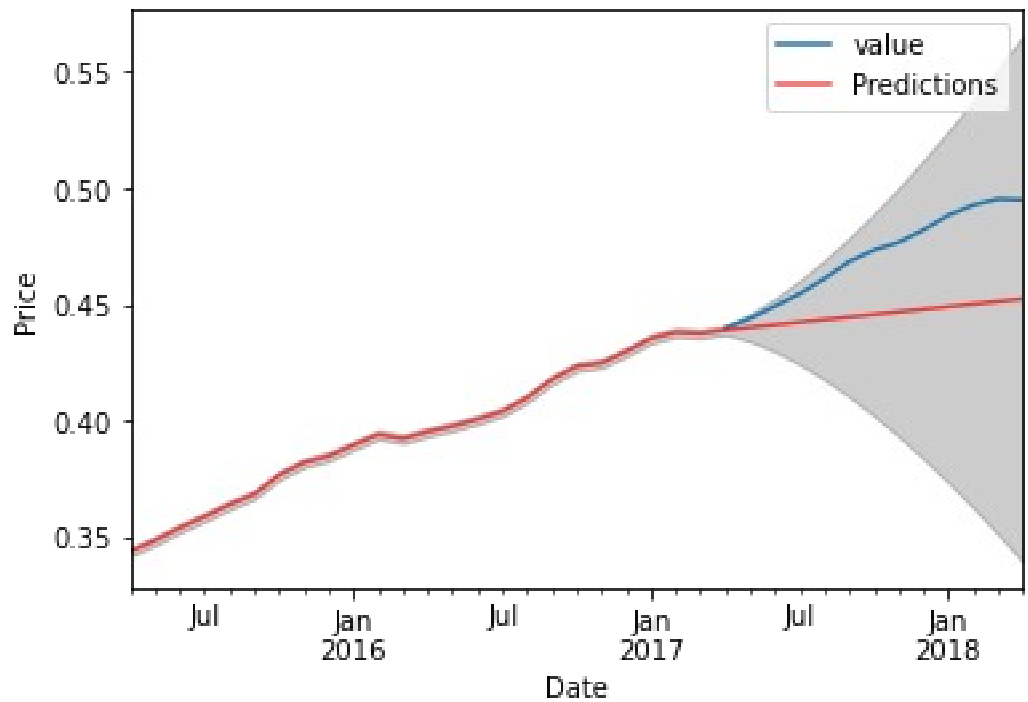
```
#Model DF, train/test, storing model pred for visuals, and model
summary
model_la = data_minus_exp_roll_mean_la
train = model_la[:-13]
test = model_la[-13:]

arima_la = ARIMA(train, order=(1,2,1))
arima_la_fit = arima_la.fit()
pred_la = arima_la_fit.predict(start="2017-04-01", end="2018-04-01")
pred = arima_la_fit.get_prediction(start="2015-04-01", end="2018-04-
01", dynamic=False)
pred_ci = pred.conf_int()
print(arima_la_fit.summary())
```

```
                               SARIMAX Results

======================================================================
========
Dep. Variable:                  value   No. Observations:
252
Model:                 ARIMA(1, 2, 1)   Log Likelihood
1359.797
Date:                Mon, 31 Oct 2022   AIC                          -
2713.595
Time:                        17:50:39   BIC                          -
2703.030
Sample:                    04-01-1996   HQIC                         -
2709.343
```

```
                         - 03-01-2017
Covariance Type:                  opg
======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
ar.L1          0.2927      0.065      4.534      0.000       0.166
0.419
ma.L1          0.4811      0.056      8.610      0.000       0.372
0.591
sigma2        1.1e-06   7.01e-08     15.697      0.000    9.63e-07
1.24e-06
======================================================================
=============
Ljung-Box (L1) (Q):                  0.00   Jarque-Bera (JB):
77.13
Prob(Q):                             0.96   Prob(JB):
0.00
Heteroskedasticity (H):              7.28   Skew:
0.41
Prob(H) (two-sided):                 0.00   Kurtosis:
5.60
======================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).

C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
tsa\base\tsa_model.py:524: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
tsa\base\tsa_model.py:524: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
tsa\base\tsa_model.py:524: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  warnings.warn('No frequency information was'
```

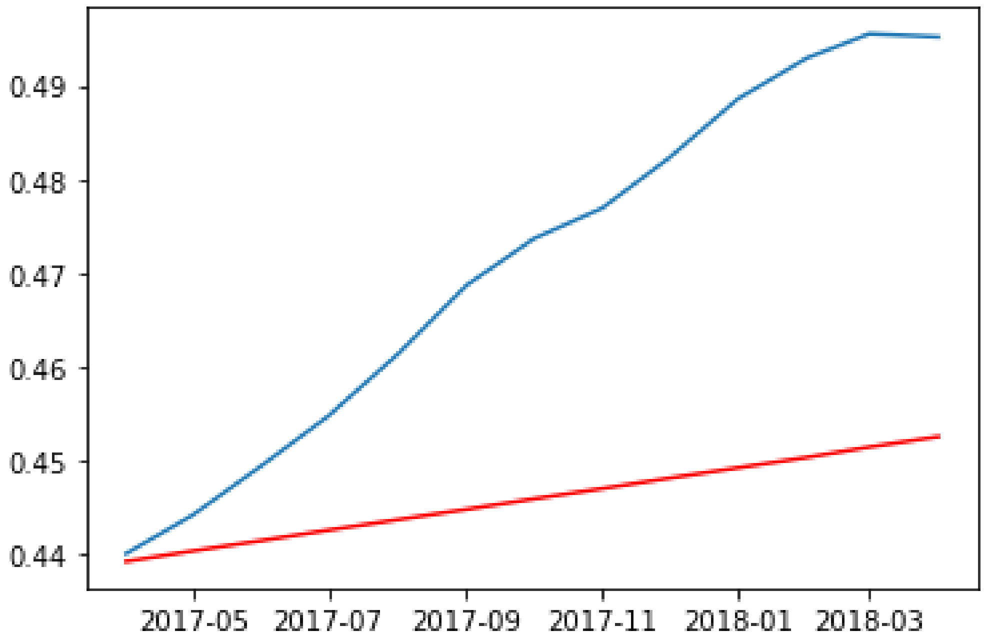
Visualization showing predications, confidence interval, and actual values

```
#Prediction, true values, and CI graph
ax = test.plot(label='Actual values')
pred.predicted_mean.plot(ax=ax, label='Predictions', alpha=.7,
color='red')

ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)

ax.set_xlabel('Date')
ax.set_ylabel('Price')
plt.legend()

plt.show()
```



Zoomed in predictions and actual values

```
#zoomed in predictions/values
plt.plot(test)
plt.plot(pred_la, color = 'red');
```



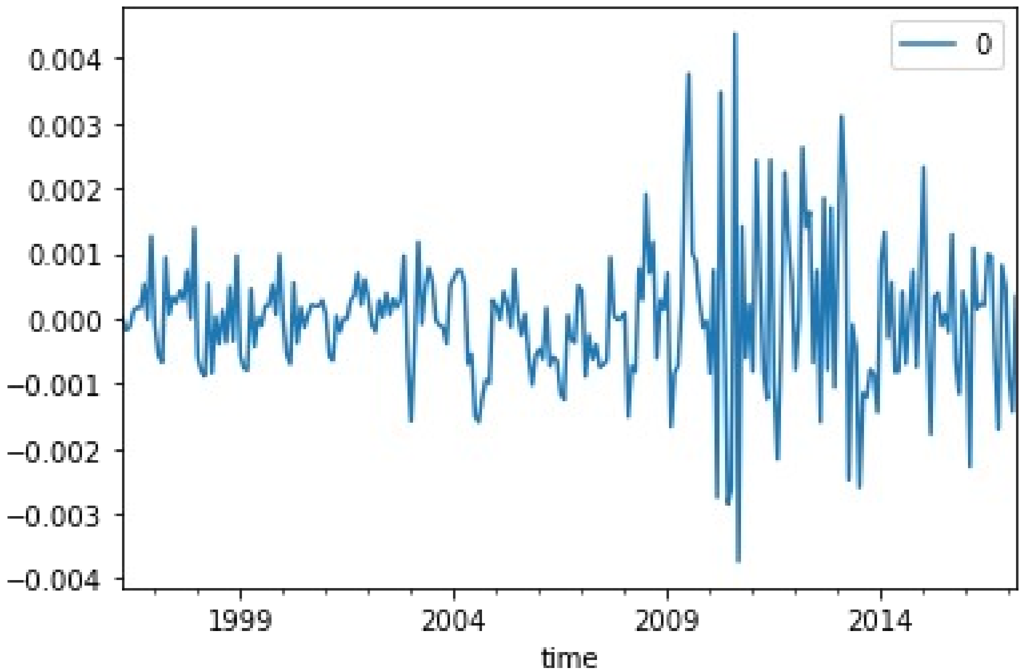
MSE, RMSE, and Residual check

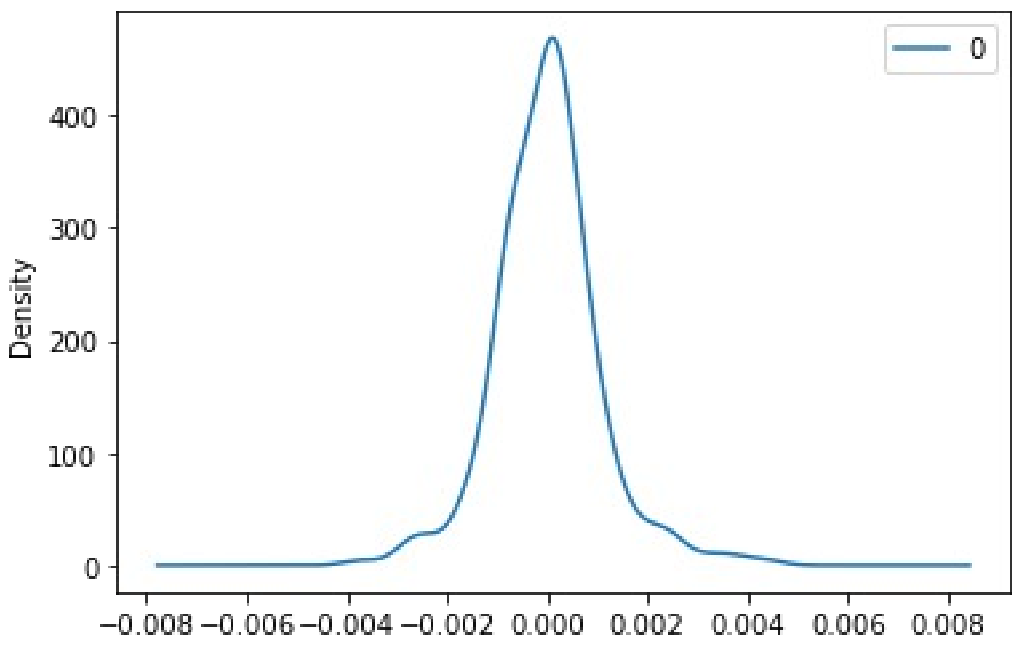
All of these are similar metrics and once again we are trying to get them as close to zero as possible

```
#MSE and RMSE check
expected = test
predictions = pred_la
mse = mean_squared_error(expected, predictions)
rmse = sqrt(mse)
print('MSE: %f' % mse)
print('RMSE: %f' % rmse)
```

```
MSE: 0.000851
RMSE: 0.029163
```

```
#checking residuals
residuals = pd.DataFrame(arima_la_fit.resid)
residuals.plot()
plt.show()
# density plot of residuals
residuals.plot(kind='kde')
plt.show()
# summary stats of residuals
print(residuals.describe())
```



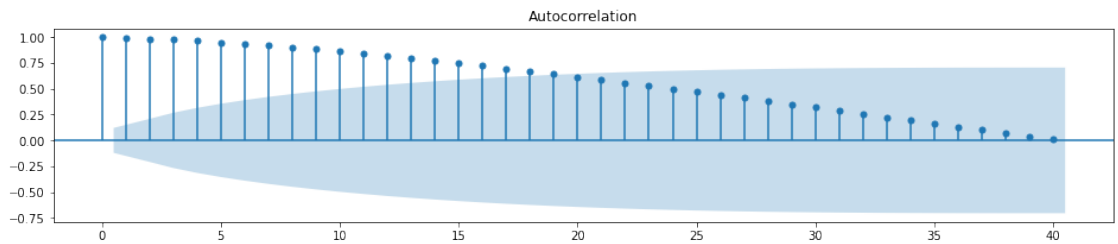


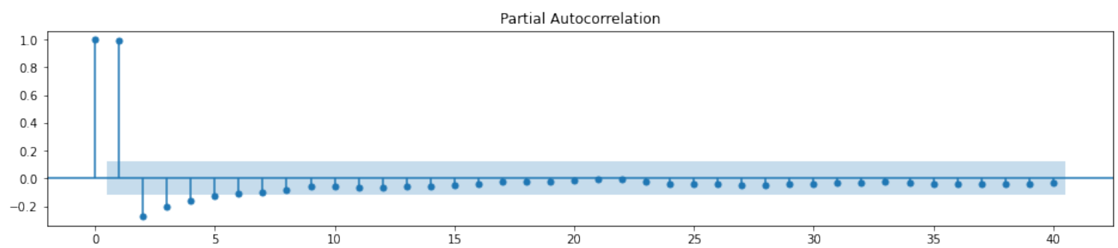

```
                0
count  252.000000
mean     0.000002
std      0.001048
min     -0.003731
25%     -0.000647
50%      0.000026
75%      0.000491
max      0.004375
```

ACF and PACF

These ACF and PACF are for helping us tune our models p and q, while we might have not used the exact lags on this chart, it provided a starting point to tune our models and work from.

```
#ACF and PACF for model tuning
fig, ax = plt.subplots(figsize=(16,3))
plot_acf(model_la, ax=ax, lags=40);

fig, ax = plt.subplots(figsize=(16,3))
plot_pacf(model_la, ax=ax, lags=40);
```



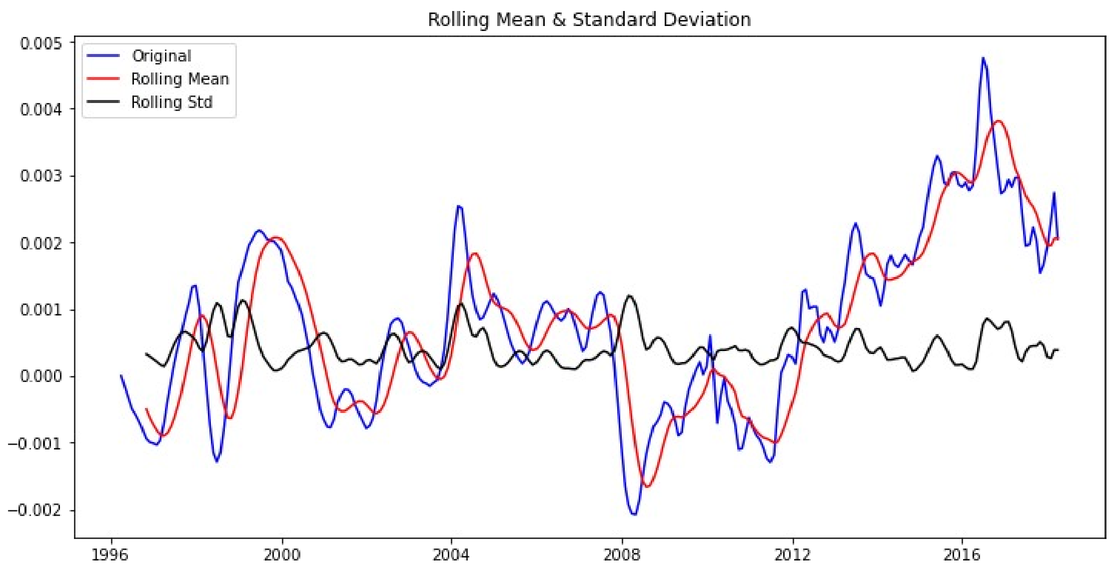


DFW model

So here is our DFW Model stationarity check, as you can see our pvalue is right at our .05 threshold.

A secondary metric here is our test statistic which we want below the critical values, while this is not below our 1% and 10% Critical Value, it is near enough to move forward with.

```
#Stationarity
exp_roll_mean_dfw = np.log(df_dfw).ewm(halflife = .5).mean()

data_minus_exp_roll_mean_dfw = np.log(df_dfw) - exp_roll_mean_dfw
stationarity_check(data_minus_exp_roll_mean_dfw)
```




```
Results of Dickey-Fuller Test:

Test Statistic                  -2.829496
p-value                          0.054182
#Lags Used                       5.000000
Number of Observations Used    259.000000
Critical Value (1%)             -3.455853
Critical Value (5%)             -2.872765
Critical Value (10%)            -2.572752
dtype: float64
```

DFW ARIMA

Below we have our train test split, our arima model's p,d,q orders, as well as our predictions for visualizations farther down.

We mainly focused on keeping our models AR and MA coef more than .2 away from zero and the pvalues under .05. The only place were it breaks those thresholds is at the MA 3 which still has a pvalue sub .05, so we will keep it in.

Some of our secondary focuses were on Heteroskedasticity, Skew, and Kurtosis.

```
#Model DF, train/test, storing model pred for visuals, and model
summary
model_dfw = data_minus_exp_roll_mean_dfw
train = model_dfw.iloc[:-13]
test = model_dfw.iloc[-13:]

arima_dfw = ARIMA(train, order=(1,1,3))
arima_dfw_fit = arima_dfw.fit()
pred_dfw = arima_dfw_fit.predict(start="2017-04-01", end="2018-04-01")
pred = arima_dfw_fit.get_prediction(start="2015-04-01", end="2018-04-
```

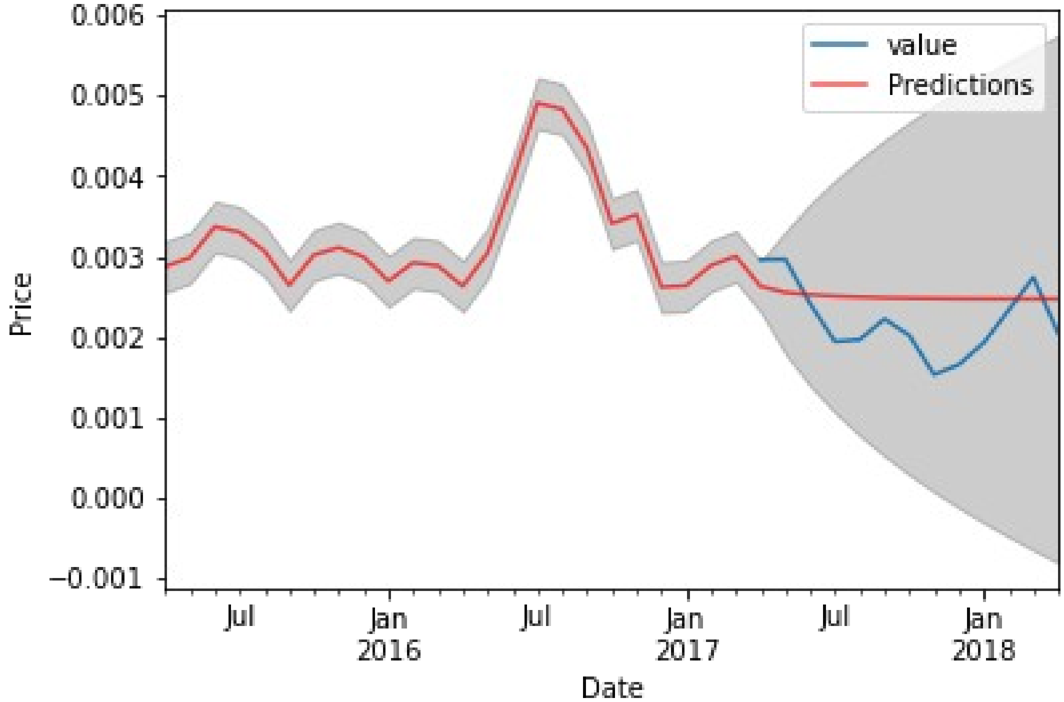
```
01", dynamic=False)
pred_ci = pred.conf_int()
print(arima_dfw_fit.summary())
```

```
                               SARIMAX Results                                
==============================================================================
========
Dep. Variable:                  value   No. Observations:                     
252
Model:                 ARIMA(1, 1, 3)   Log Likelihood                        
1832.230
Date:                Mon, 31 Oct 2022   AIC                                  -
3654.461
Time:                        17:50:40   BIC                                  -
3636.833
Sample:                    04-01-1996   HQIC                                 -
3647.367
                         - 03-01-2017                                         

Covariance Type:                  opg                                         

==============================================================================
========
                 coef    std err          z      P>|z|      [0.025      
0.975]
------------------------------------------------------------------------------
--------
ar.L1          0.6634      0.034     19.649      0.000       0.597       
0.730
ma.L1          0.5334      0.020     26.632      0.000       0.494       
0.573
ma.L2         -0.3308      0.021    -15.388      0.000      -0.373      
-0.289
ma.L3         -0.1332      0.027     -4.941      0.000      -0.186      
-0.080
sigma2      2.592e-08   1.57e-09     16.518      0.000    2.28e-08     
2.9e-08
==============================================================================
=============
Ljung-Box (L1) (Q):                   0.08   Jarque-Bera (JB):             
132.23
Prob(Q):                              0.77   Prob(JB):                     
0.00
Heteroskedasticity (H):               3.23   Skew:                         
-0.37
Prob(H) (two-sided):                  0.00   Kurtosis:                     
6.48
==============================================================================
=============
```

```
Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
[2] Covariance matrix is singular or near-singular, with condition
number 3.04e+16. Standard errors may be unstable.

C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
tsa\base\tsa_model.py:524: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
tsa\base\tsa_model.py:524: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
tsa\base\tsa_model.py:524: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  warnings.warn('No frequency information was'
```

Visualization showing predications, confidence interval, and actual values

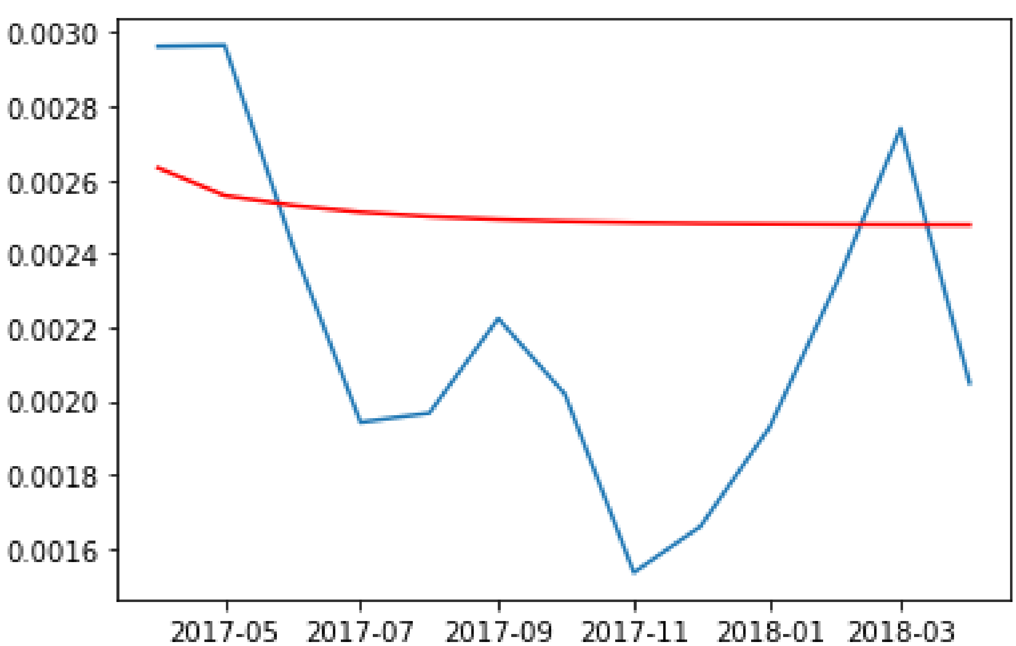
```
#Predictions, True values, and CI graph
ax = test.plot(label='Actual values')
pred.predicted_mean.plot(ax=ax, label='Predictions', alpha=.7,
color='red')

ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)

ax.set_xlabel('Date')
ax.set_ylabel('Price')
plt.legend()

plt.show()
```



Zoomed in predictions and actual values

```
#zoomed in of pred and values
plt.plot(test)
plt.plot(pred_dfw, color = 'red')
```

```
[<matplotlib.lines.Line2D at 0x23fb6f911f0>]
```



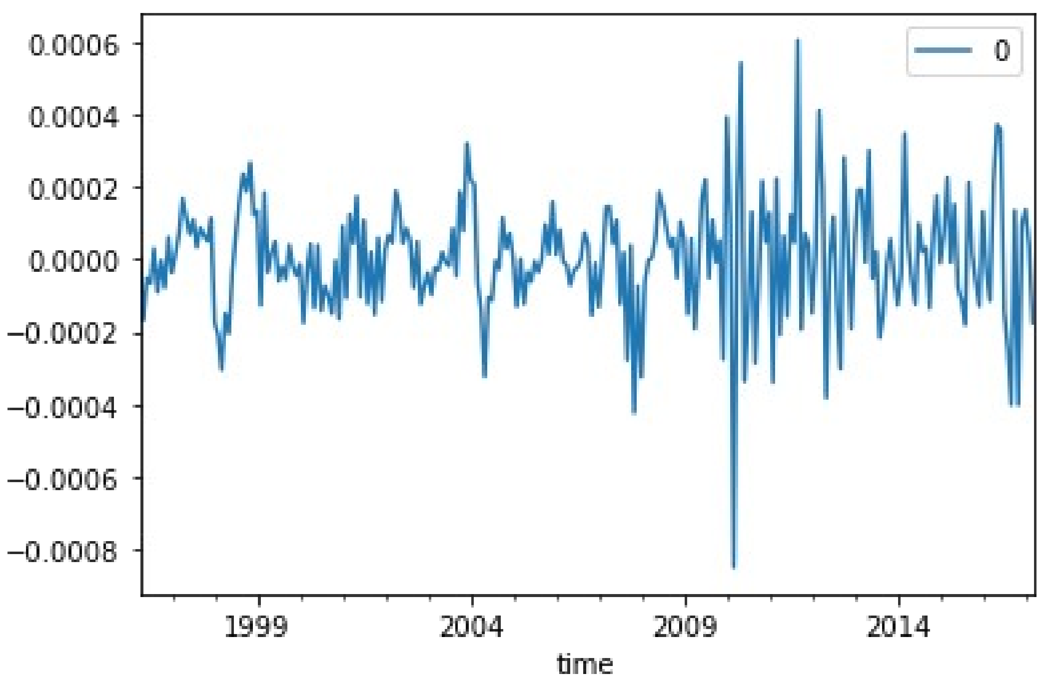
MSE, RMSE, and Residual check

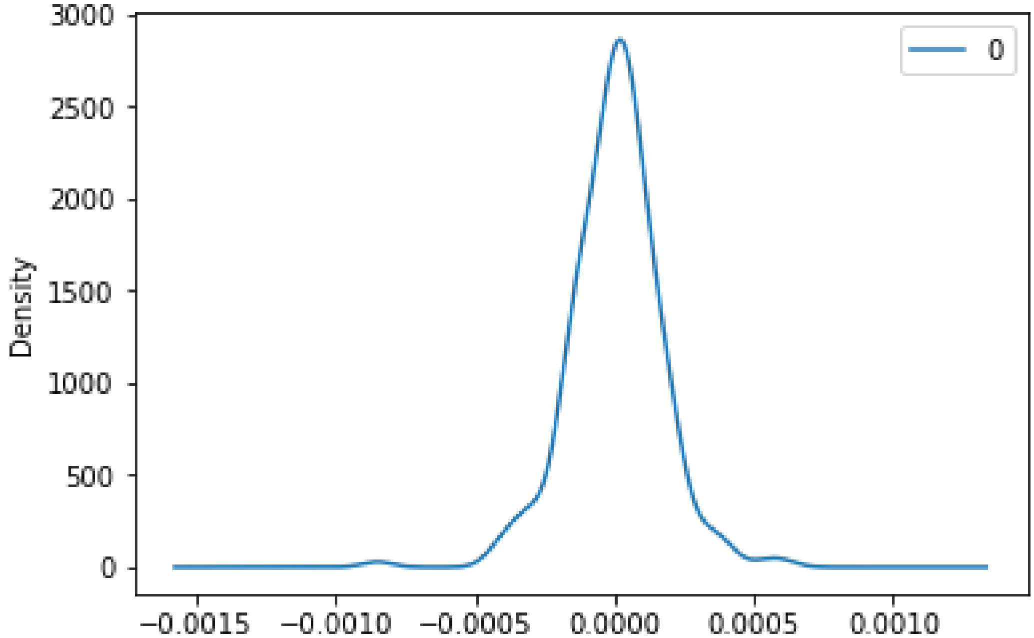
All of these are similar metrics and once again we are trying to get them as close to zero as possible

```
#MSE and RMSE check
expected = test
predictions = pred_dfw
mse = mean_squared_error(expected, predictions)
rmse = sqrt(mse)
print('MSE: %f' % mse)
print('RMSE: %f' % rmse)
```

```
MSE: 0.000000
RMSE: 0.000507
```

```
#Residuals check
residuals = pd.DataFrame(arima_dfw_fit.resid)
residuals.plot()
plt.show()
# density plot of residuals
residuals.plot(kind='kde')
plt.show()
# summary stats of residuals
print(residuals.describe())
```



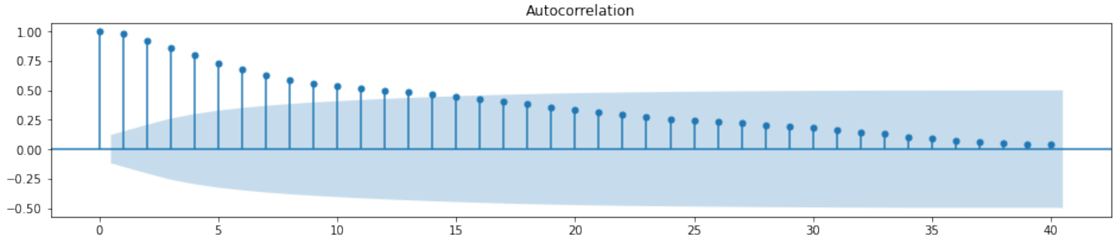


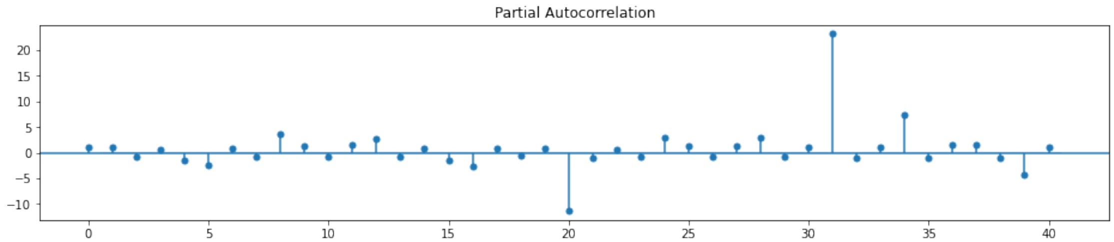

```
                0
count  252.000000
mean     0.000003
std      0.000163
min     -0.000851
25%     -0.000081
50%      0.000005
75%      0.000089
max      0.000605
```

ACF and PACF

```
#PACF and ACF for model tuning
fig, ax = plt.subplots(figsize=(16,3))
plot_acf(model_dfw, ax=ax, lags=40);

fig, ax = plt.subplots(figsize=(16,3))
plot_pacf(model_dfw, ax=ax, lags=40);
```

```
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
regression\linear_model.py:1434: RuntimeWarning: invalid value
encountered in sqrt
  return rho, np.sqrt(sigmasq)
```



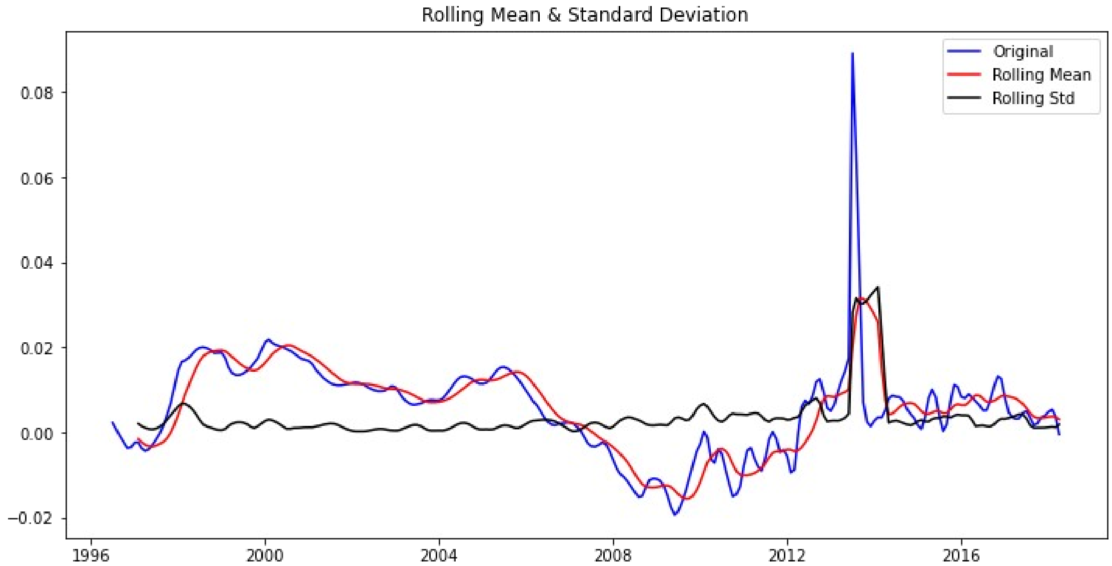


Chicago Model

So here is our Chicago Model stationarity check, as you can see our pvalue is right at our .05 threshold.

A secondary metric here is our test statistic which we want below the critical values, while this is not below our 1% it is quite close and below the other critical values, so we will move forward.

```
#Stationarity check
roll_mean_chi = np.log(df_chi).rolling(window=4).mean()
data_minus_roll_mean_chi = np.log(df_chi) - roll_mean_chi

# Print the first 10 rows
data_minus_roll_mean_chi.head(10)
data_minus_roll_mean_chi.dropna(inplace=True)

stationarity_check(data_minus_roll_mean_chi)
```




```
Results of Dickey-Fuller Test:

Test Statistic                  -3.281159
p-value                          0.015742
#Lags Used                       3.000000
Number of Observations Used    258.000000
Critical Value (1%)             -3.455953
Critical Value (5%)             -2.872809
Critical Value (10%)            -2.572775
dtype: float64
```

Chicago ARIMA

Below we have our train test split, our arima's p,d,q orders, as well as our predictions for visualizations farther down.

We mainly focused on keeping our models AR and MA coef more than .2 away from zero and the pvalues under .05. The place's where it breaks those thresholds is at the MA 1 which has a pvalue of .12 and AR 4 which has a pvalue of .07, we are keeping it in because the other MA and AR's are sub .05

Some of our secondary focuses were on Heteroskedasticity, Skew, and Kurtosis.

```
#Model DF, train/test, storing model pred for visuals, and model summary
model_chi = data_minus_roll_mean_chi
train = model_chi[:-13]
test = model_chi[-13:]

arima_chi = ARIMA(train, order=(4,2,3))
arima_chi_fit = arima_chi.fit()
```

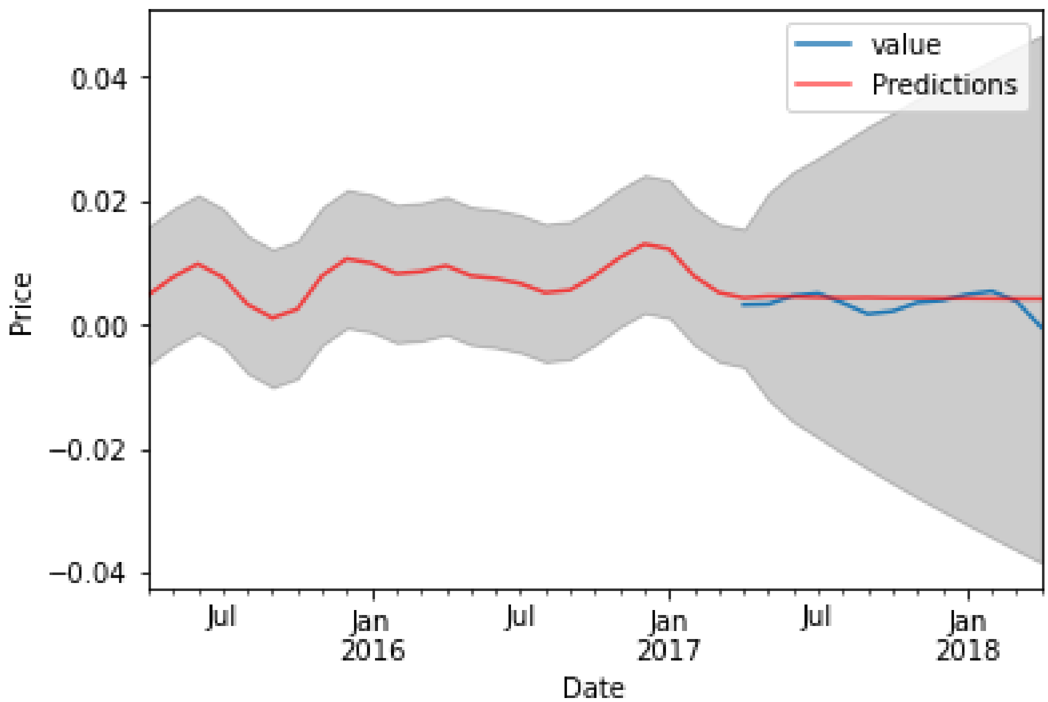
```
pred_chi = arima_chi_fit.predict(start="2017-04-01", end="2018-04-01")
pred = arima_chi_fit.get_prediction(start="2015-04-01", end="2018-04-
01", dynamic=False)
pred_ci = pred.conf_int()

print(arima_chi_fit.summary())
```

```
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
tsa\base\tsa_model.py:524: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
tsa\base\tsa_model.py:524: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
tsa\base\tsa_model.py:524: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  warnings.warn('No frequency information was'

                               SARIMAX Results

======================================================================
========
Dep. Variable:                  value   No. Observations:
249
Model:                 ARIMA(4, 2, 3)   Log Likelihood
925.073
Date:                Mon, 31 Oct 2022   AIC                          -
1834.146
Time:                        17:50:41   BIC                          -
1806.071
Sample:                    07-01-1996   HQIC                         -
1822.843
                         - 03-01-2017

Covariance Type:                  opg

======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
ar.L1         -1.4069      0.356     -3.949      0.000      -2.105
-0.709
ar.L2         -0.7447      0.264     -2.824      0.005      -1.262
-0.228
ar.L3         -0.2405      0.121     -1.992      0.046      -0.477
-0.004
```

```
ar.L4         -0.1948      0.109     -1.782      0.075      -0.409
0.020
ma.L1          0.5298      0.356      1.489      0.136      -0.167
1.227
ma.L2         -0.7148      0.295     -2.422      0.015      -1.293
-0.136
ma.L3         -0.7425      0.314     -2.368      0.018      -1.357
-0.128
sigma2      3.221e-05   4.49e-07     71.783      0.000    3.13e-05
3.31e-05
======================================================================
=============
Ljung-Box (L1) (Q):                  0.38   Jarque-Bera (JB):
113354.27
Prob(Q):                             0.54   Prob(JB):
0.00
Heteroskedasticity (H):             70.82   Skew:
6.87
Prob(H) (two-sided):                 0.00   Kurtosis:
107.05
======================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).

C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
base\model.py:566: ConvergenceWarning: Maximum Likelihood optimization
failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

Visualization showing predications, confidence interval, and actual values

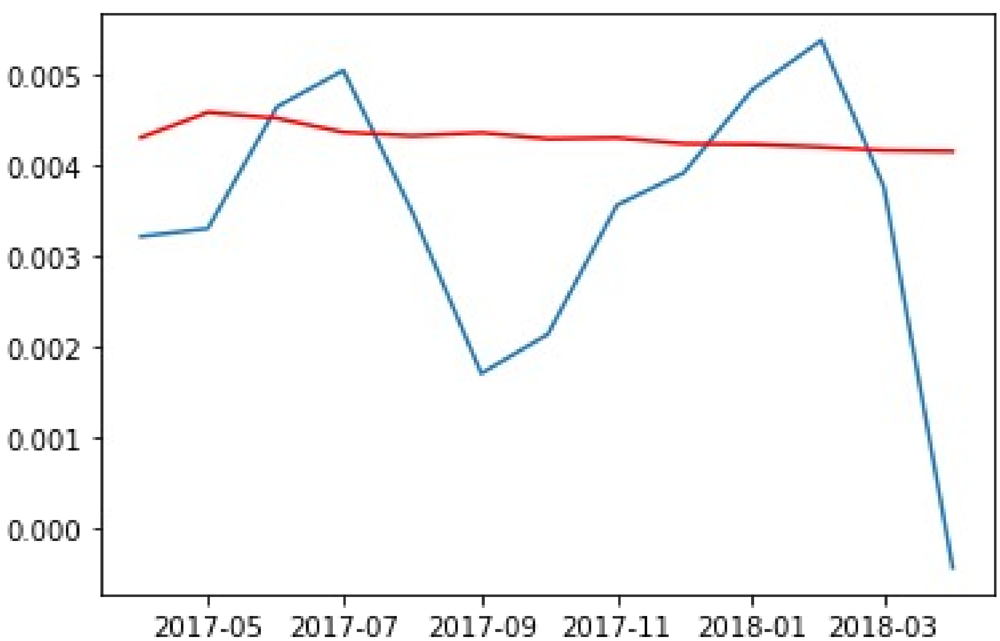
```
#Prediction, true values, and CI graph
ax = test.plot(label='Actual values')
pred.predicted_mean.plot(ax=ax, label='Predictions', alpha=.7,
color='red')

ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)

ax.set_xlabel('Date')
ax.set_ylabel('Price')
plt.legend()

plt.show()
```



Zoomed in predictions and values

```
#zoomed in pred and values
plt.plot(test)
plt.plot(pred_chi, color = 'red')
```

```
[<matplotlib.lines.Line2D at 0x23fb7b31f40>]
```

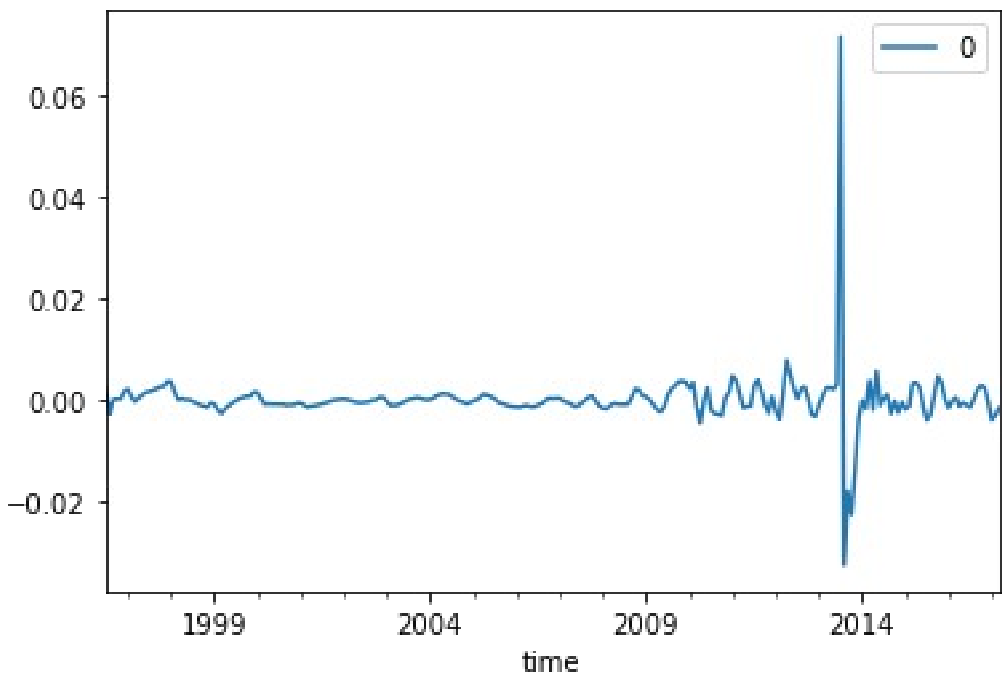


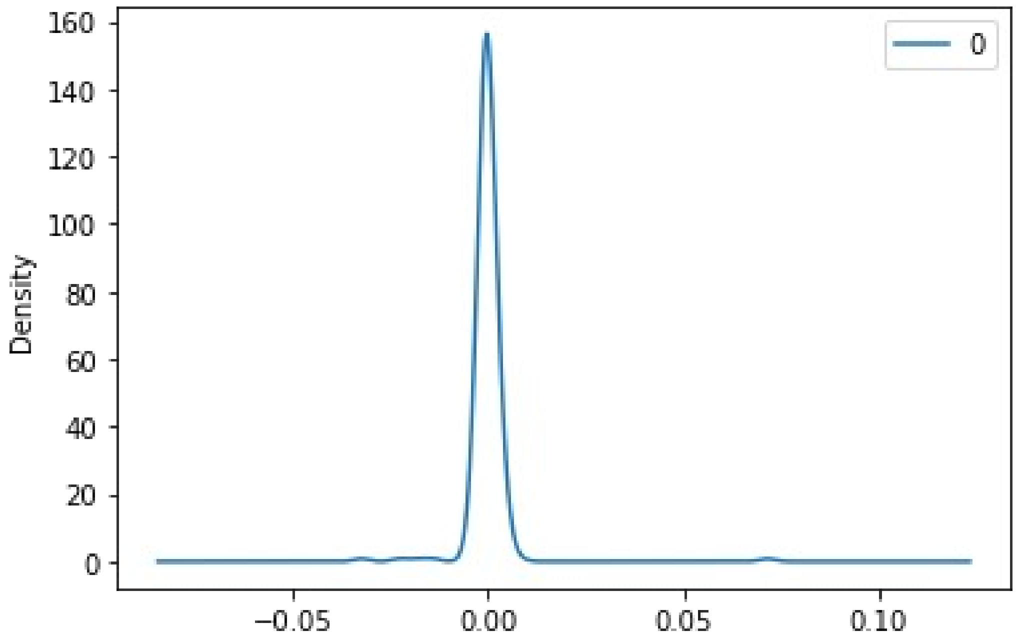
MSE, RMSE, and residuals check

```
#MSE and RMSE check
expected = test
predictions = pred_chi
mse = mean_squared_error(expected, predictions)
rmse = sqrt(mse)
print('MSE: %f' % mse)
print('RMSE: %f' % rmse)
```

```
MSE: 0.000003
RMSE: 0.001743
```

```
#residuals check
residuals = pd.DataFrame(arima_chi_fit.resid)
residuals.plot()
plt.show()
# density plot of residuals
residuals.plot(kind='kde')
plt.show()
# summary stats of residuals
print(residuals.describe())
```

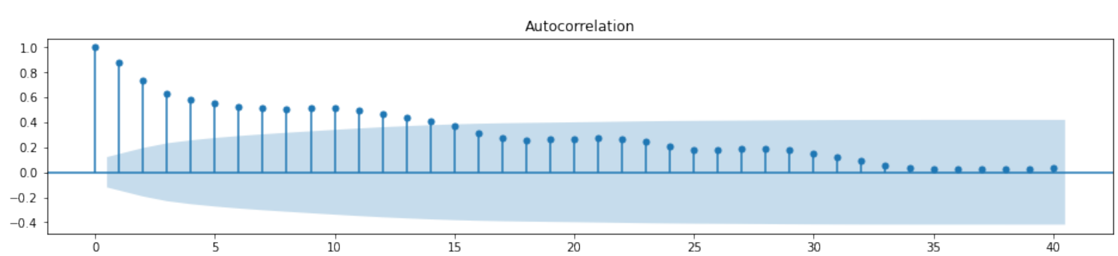


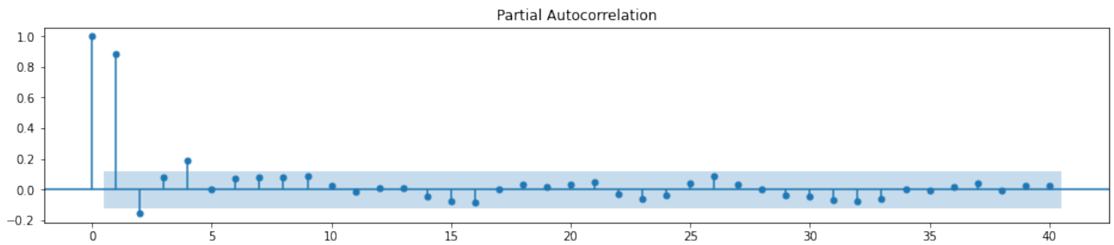



```
                0
count  249.000000
mean    -0.000008
std      0.005672
min     -0.032479
25%     -0.001083
50%     -0.000161
75%      0.000802
max      0.071362
```

PACF and ACF

```
#PACF and ACF graph for model tuning
fig, ax = plt.subplots(figsize=(16,3))
plot_acf(model_chi, ax=ax, lags=40);

fig, ax = plt.subplots(figsize=(16,3))
plot_pacf(model_chi, ax=ax, lags=40);
```



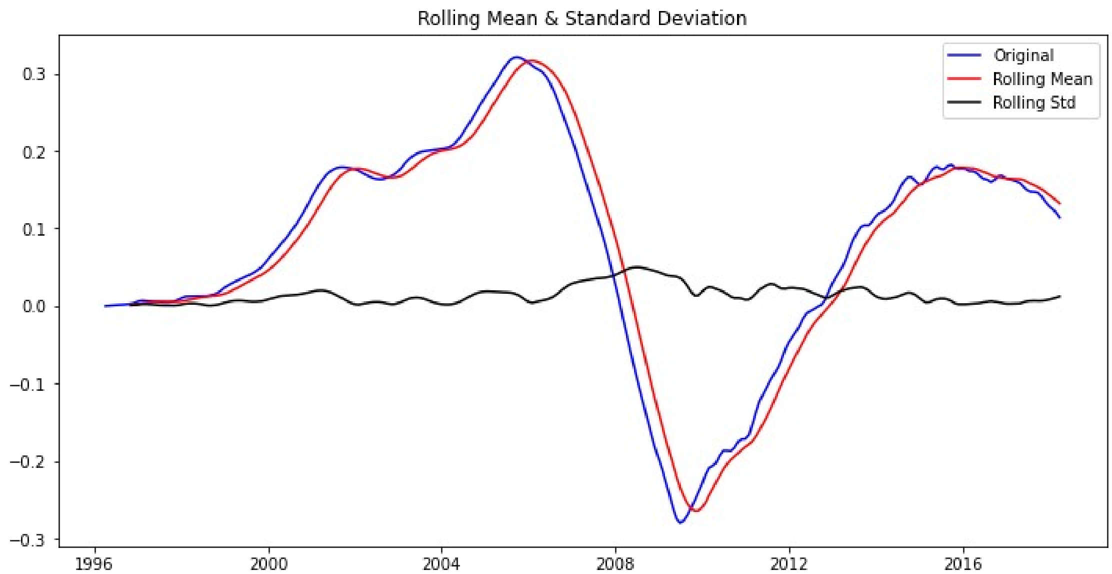


Miami Model

So here is our Miami stationarity check, as you can see our pvalue is right at our .05 threshold.

A secondary metric here is our test statistic which we want below the critical values, while this is not below our 1% it is quite close and below the other critical values, so we will move forward.

```
#Stationarity
exp_roll_mean_mia = np.log(df_mia).ewm(alpha=0.05).mean()
data_minus_exp_roll_mean_mia = np.log(df_mia) - exp_roll_mean_mia
stationarity_check(data_minus_exp_roll_mean_mia)
```




```
Results of Dickey-Fuller Test:

Test Statistic                  -3.062570
p-value                          0.029450
#Lags Used                      15.000000
Number of Observations Used    249.000000
Critical Value (1%)             -3.456888
Critical Value (5%)             -2.873219
```

```
Critical Value (10%)           -2.572994
dtype: float64
```

Miami ARIMA

Below we have our train test split, our arima's p,d,q orders, as well as our predictions for visualizations farther down.

We mainly focused on keeping our models AR and MA coef more than .2 away from zero and the pvalues under .05. This one keeps all those within our acceptable range.

Some of our secondary focuses were on Heteroskedasticity, Skew, and Kurtosis.

```
#Model DF, train/test, storing model pred for visuals, and model
summary
model_mia = data_minus_exp_roll_mean_mia
train = model_mia[:-13]
test = model_mia[-13:]

arima_mia = ARIMA(train, order=(2,0,1))
arima_mia_fit = arima_mia.fit()
pred_mia = arima_mia_fit.predict(start="2017-04-01", end="2018-04-01")
pred = arima_mia_fit.get_prediction(start="2015-04-01", end="2018-04-
01", dynamic=False)
pred_ci = pred.conf_int()
print(arima_mia_fit.summary())
```

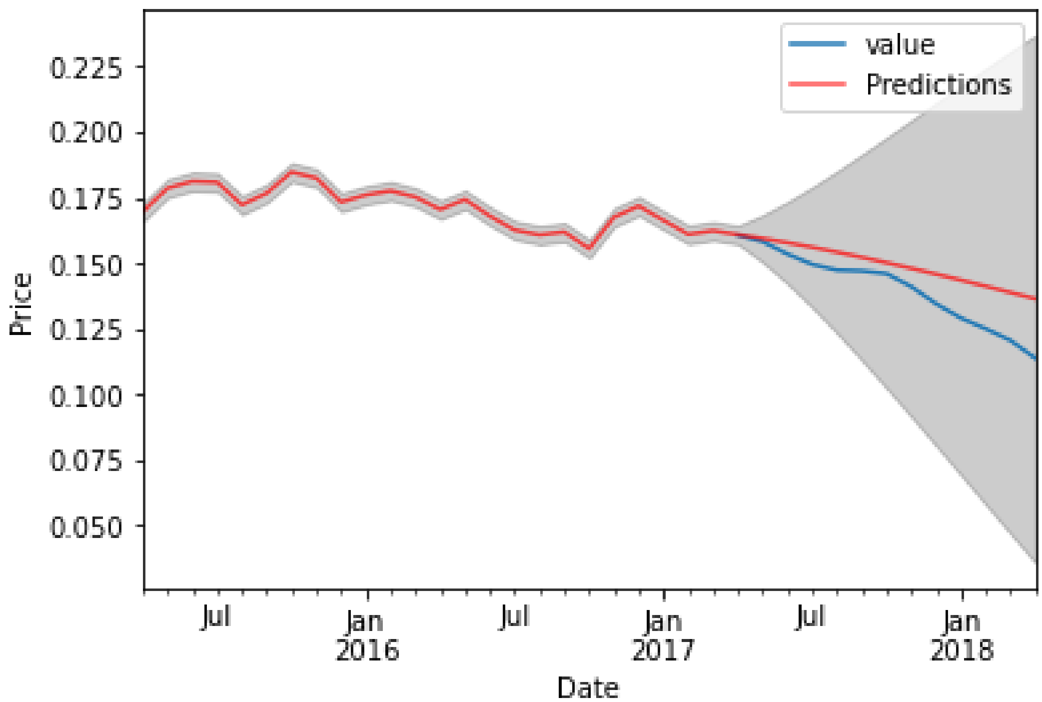
```
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
tsa\base\tsa_model.py:524: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
tsa\base\tsa_model.py:524: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\
tsa\base\tsa_model.py:524: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  warnings.warn('No frequency information was'

                               SARIMAX Results

======================================================================
========
Dep. Variable:                value   No. Observations:
252
Model:                 ARIMA(2, 0, 1)   Log Likelihood
1228.862
Date:                Mon, 31 Oct 2022   AIC                          -
2447.723
```

```
Time:                        17:50:43   BIC                          -
2430.076
Sample:                    04-01-1996   HQIC                         -
2440.622
                         - 03-01-2017

Covariance Type:                  opg

======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
const          0.0748      0.062      1.203      0.229      -0.047
0.196
ar.L1          1.9313      0.023     83.590      0.000       1.886
1.977
ar.L2         -0.9342      0.023    -40.453      0.000      -0.979
-0.889
ma.L1          0.4219      0.047      9.055      0.000       0.331
0.513
sigma2      3.251e-06   2.38e-07     13.669      0.000    2.78e-06
3.72e-06
======================================================================
=============
Ljung-Box (L1) (Q):                  14.73   Jarque-Bera (JB):
52.87
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):              22.95   Skew:
0.33
Prob(H) (two-sided):                  0.00   Kurtosis:
5.15
======================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
```

Visualization showing predications, confidence interval, and actual values

```
#Predictions, values, and CI graph
ax = test.plot(label='Actual values')
pred.predicted_mean.plot(ax=ax, label='Predictions', alpha=.7,
color='red')
```

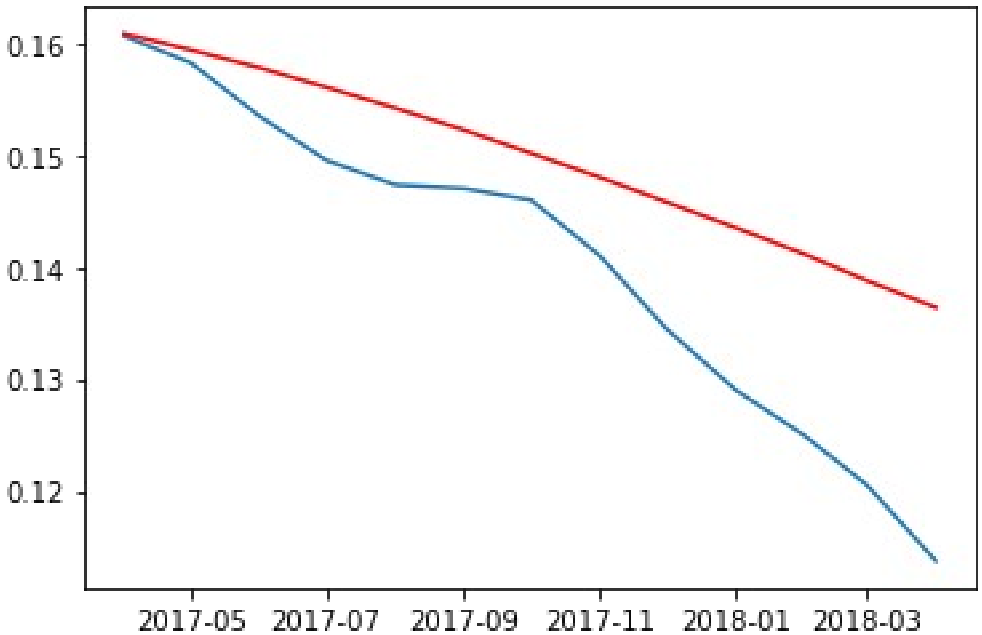
```
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)

ax.set_xlabel('Date')
ax.set_ylabel('Price')
plt.legend()

plt.show()
```



Zoomed in predictions and values

```
#zoomed in pred and values
plt.plot(test)
plt.plot(pred_mia, color = 'red')
```

```
[<matplotlib.lines.Line2D at 0x23fb7b870d0>]
```

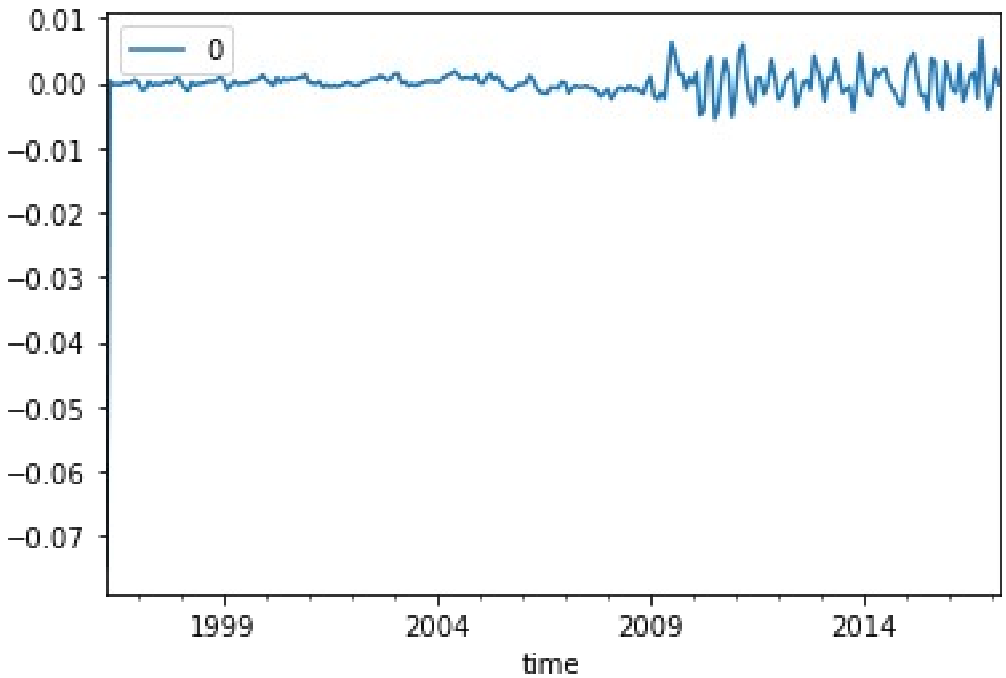


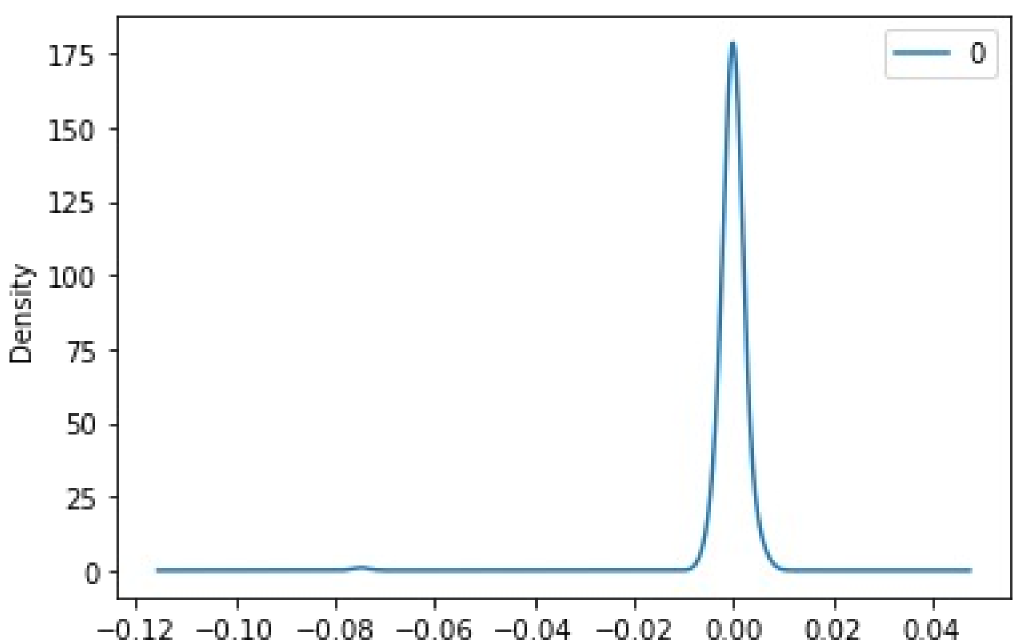
MSE, RMSE, and residual check

```
#MSE and RMSE check
expected = test
predictions = pred_mia
mse = mean_squared_error(expected, predictions)
rmse = sqrt(mse)
print('MSE: %f' % mse)
print('RMSE: %f' % rmse)
```

```
MSE: 0.000127
RMSE: 0.011270
```

```
#residuals check
residuals = pd.DataFrame(arima_mia_fit.resid)
residuals.plot()
plt.show()
# density plot of residuals
residuals.plot(kind='kde')
plt.show()
# summary stats of residuals
print(residuals.describe())
```





	0
count	252.000000
mean	-0.000270
std	0.005045
min	-0.074753
25%	-0.000792
50%	0.000018
75%	0.000839
max	0.006774

PACF and ACF

```
#PACF and ACF graph for model tuning
fig, ax = plt.subplots(figsize=(16,3))
plot_acf(model_mia, ax=ax, lags=40);

fig, ax = plt.subplots(figsize=(16,3))
plot_pacf(model_mia, ax=ax, lags=40);
```

```
C:\Users\Clay\anaconda3\envs\learn-env\lib\site-packages\statsmodels\regression\linear_model.py:1434: RuntimeWarning: invalid value encountered in sqrt
  return rho, np.sqrt(sigmasq)
```