05-ARMA-and-ARIMA

November 2, 2021

1 ARMA(p,q) and ARIMA(p,d,q)

2 Autoregressive Moving Averages

This section covers Autoregressive Moving Averages (ARMA) and Autoregressive Integrated Moving Averages (ARIMA).

Recall that an AR(1) model follows the formula

$$y_t = c + \phi_1 y_{t-1} + \varepsilon_t$$

while an MA(1) model follows the formula

$$y_t = \mu + \theta_1 \varepsilon_{t-1} + \varepsilon_t$$

where c is a constant, μ is the expectation of y_t (often assumed to be zero), ϕ_1 (phi-sub-one) is the AR lag coefficient, θ_1 (theta-sub-one) is the MA lag coefficient, and ε (epsilon) is white noise.

An ARMA(1,1) model therefore follows

$$y_t = c + \phi_1 y_{t-1} + \theta_1 \varepsilon_{t-1} + \varepsilon_t$$

ARMA models can be used on stationary datasets.

For non-stationary datasets with a trend component, ARIMA models apply a differencing coefficient as well.

Related Functions:

arima_model.ARMA(endog, order[, exog, …]) Autoregressive Moving Average ARMA(p,q) model arima_model.ARMAResults(model, params[, …]) Class to hold results from fitting an ARMA model arima_model.ARIMA(endog, order[, exog, …]) Autoregressive Integrated Moving Average ARIMA(p,d,q) model arima_model.ARIMAResults(model, params[, …]) Class to hold results from fitting an ARIMA model
kalmanf.kalmanfilter.KalmanFilter Kalman Filter code intended for use with the ARMA model

For Further Reading:

Wikipedia Autoregressive–moving-average model Forecasting: Principles and Practice Non-seasonal ARIMA models

[45]:
```
import pandas as pd
import numpy as np
```

```
%matplotlib inline

# Load specific forecasting tools
from statsmodels.tsa.arima_model import ARMA,ARMAResults,ARIMA,ARIMAResults
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf # for determining␣
 ↪(p,q) orders
from pmdarima import auto_arima # for determining ARIMA orders

# Ignore harmless warnings
import warnings
warnings.filterwarnings("ignore")
```

2.1 Automate the augmented Dickey-Fuller Test

Since we'll be using it a lot to determine if an incoming time series is stationary, let's write a function that performs the augmented Dickey-Fuller Test.

[46]:
```
from statsmodels.tsa.stattools import adfuller

def adf_test(series,title=''):
    """
    Pass in a time series and an optional title, returns an ADF report
    """
    print(f'Augmented Dickey-Fuller Test: {title}')
    result = adfuller(series.dropna(),autolag='AIC') # .dropna() handles␣
 ↪differenced data

    labels = ['ADF test statistic','p-value','# lags used','# observations']
    out = pd.Series(result[0:4],index=labels)

    for key,val in result[4].items():
        out[f'critical value ({key})']=val

    print(out.to_string())          # .to_string() removes the line "dtype:␣
 ↪float64"

    if result[1] <= 0.05:
        print("Strong evidence against the null hypothesis")
        print("Reject the null hypothesis")
        print("Data has no unit root and is stationary")
    else:
        print("Weak evidence against the null hypothesis")
        print("Fail to reject the null hypothesis")
        print("Data has a unit root and is non-stationary")
```

[47]:
```

```

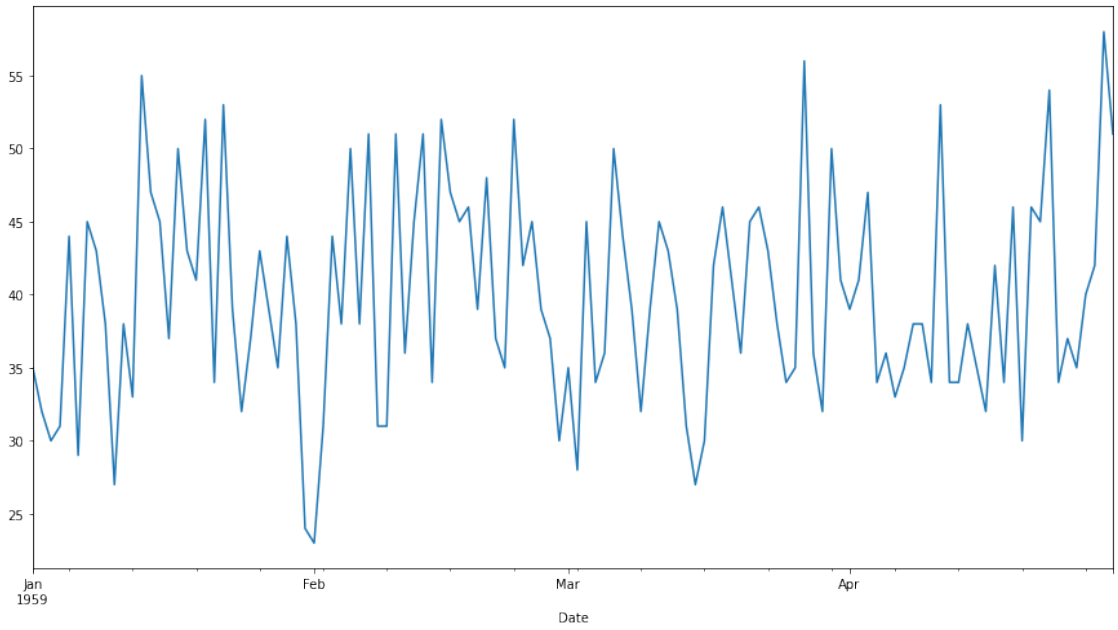
```
# Load datasets
df1 = pd.read_csv('../Data/DailyTotalFemaleBirths.
 ↪csv',index_col='Date',parse_dates=True)
df1.index.freq = 'D'
df1 = df1[:120]  # we only want the first four months

df2 = pd.read_csv('../Data/TradeInventories.
 ↪csv',index_col='Date',parse_dates=True)
df2.index.freq='MS'
```

2.2 Autoregressive Moving Average - ARMA(p,q)

In this first section we'll look at a stationary dataset, determine (p,q) orders, and run a forecasting ARMA model fit to the data. In practice it's rare to find stationary data with no trend or seasonal component, but the first four months of the Daily Total Female Births dataset should work for our purposes. ### Plot the source data

[48]:
```
df1["Births"].plot(figsize=(15,8))
```

[48]: <AxesSubplot:xlabel='Date'>



2.2.1 Run the augmented Dickey-Fuller Test to confirm stationarity

[49]:
```
'''
OK, so let's go ahead and run the augmented Dicky Fuller test to confirm␣
↪stationary.

notice the P value is absolutely tiny.

So there's strong evidence against the null hypothesis.

We reject the null hypothesis and we say the data has no root and is stationary

'''
adf_test(df1["Births"])
```

```
Augmented Dickey-Fuller Test:
ADF test statistic     -9.855384e+00
p-value                 4.373545e-17
# lags used             0.000000e+00
# observations          1.190000e+02
critical value (1%)    -3.486535e+00
critical value (5%)    -2.886151e+00
critical value (10%)   -2.579896e+00
Strong evidence against the null hypothesis
Reject the null hypothesis
Data has no unit root and is stationary
```

2.2.2 Determine the (p,q) ARMA Orders using pmdarima.auto_arima

This tool should give just p and q value recommendations for this dataset.

[50]:
```
auto_arima(df1['Births'],seasonal=False).summary()
```

[50]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==========================================================================================
Dep. Variable:                           y   No. Observations:                  120
Model:                           SARIMAX   Log Likelihood                -409.745
Date:                Tue, 02 Nov 2021   AIC                            823.489
Time:                        16:13:35   BIC                            829.064
Sample:                             0   HQIC                           825.753
                                  - 120
Covariance Type:                    opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     39.7833      0.687     57.896      0.000      38.437      41.130
```

```
sigma2         54.1197      8.319      6.506      0.000      37.815      70.424
==============================================================================
===
Ljung-Box (L1) (Q):                   0.85   Jarque-Bera (JB):
2.69
Prob(Q):                              0.36   Prob(JB):
0.26
Heteroskedasticity (H):               0.80   Skew:
0.26
Prob(H) (two-sided):                  0.48   Kurtosis:
2.48
==============================================================================
===

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

2.2.3 Split the data into train/test sets

As a general rule you should set the length of your test set equal to your intended forecast size. For this dataset we'll attempt a 1-month forecast.

[51]:
```
'''
And as I mentioned, a general rule is you should set the length of your test␣
 ↪set equal to your intended

forecast size.


So in this case, we're going to attempt to do a 1 month forecast.

So essentially, 25% of our data is going to be used for the test set since␣
 ↪we're forecasting

1 month into the future roughly.

Total 4 month data i.e 120 days
'''
train=df1.iloc[:90].astype('float32')
test=df1.iloc[90:].astype('float32')
```

2.2.4 Fit an ARMA(p,q) Model

If you want you can run help(ARMA) to learn what incoming arguments are available/expected, and what's being returned.

[52]:
```
'''
since this is just an ARMA model, we need the grab order of the Auto Regression
as well as the order of the moving average component i.e 0.0.

'''
model=ARMA(train["Births"],order=(0,0))
results=model.fit()
```

[53]:
```
results.summary()
```

[53]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                              ARMA Model Results
==============================================================================
Dep. Variable:                 Births   No. Observations:                   90
Model:                     ARMA(0, 0)   Log Likelihood                -308.379
Method:                           css   S.D. of innovations              7.445
Date:                Tue, 02 Nov 2021   AIC                            620.759
Time:                        16:13:35   BIC                            625.759
Sample:                    01-01-1959   HQIC                           622.775
                         - 03-31-1959
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         39.7667      0.785     50.675      0.000      38.229      41.305
==============================================================================
"""
```

2.2.5 Obtain a month's worth of predicted values

[54]:
```
'''
Now we'll need to do is actually obtain a month's worth of predictive values.

So we simply need to start and end dates or start and locations.

predict into the future where we have our start and our end.

And I'm going to rename this so that the predictions actually have a name so we␣
 ↪can plot it out

And notice here that for our predictions, they're all kind of hovering around␣
 ↪this mean.

'''
start=len(train)
end=len(train)+len(test)-1
```

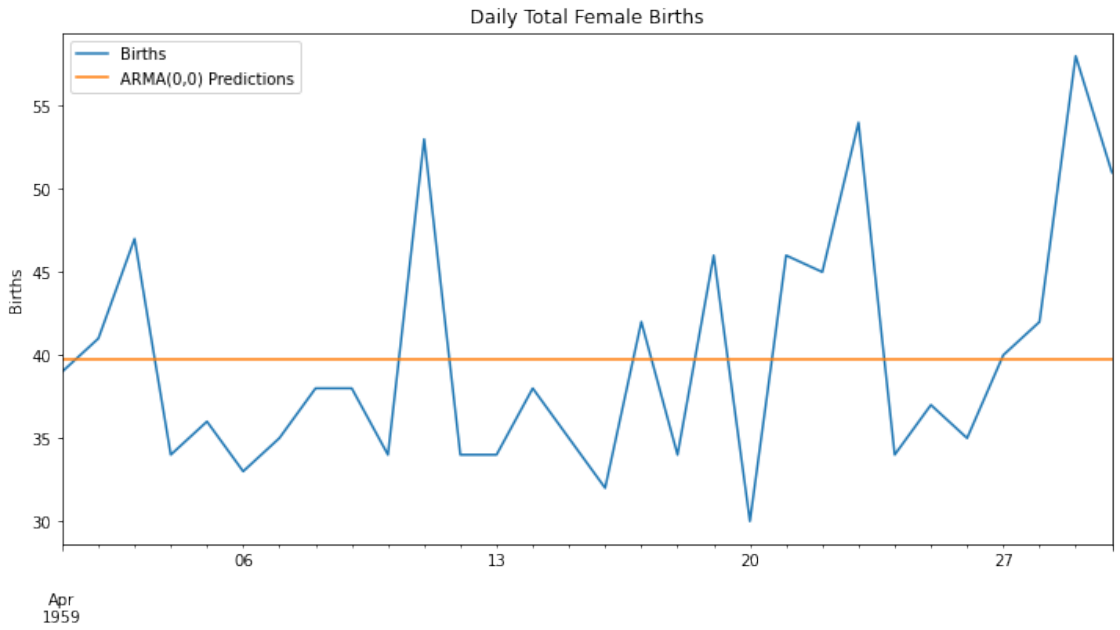
```
predictions=results.predict(start,end).rename("ARMA(0,0) Predictions")
predictions
```

[54]:
```
1959-04-01    39.766667
1959-04-02    39.766667
1959-04-03    39.766667
1959-04-04    39.766667
1959-04-05    39.766667
1959-04-06    39.766667
1959-04-07    39.766667
1959-04-08    39.766667
1959-04-09    39.766667
1959-04-10    39.766667
1959-04-11    39.766667
1959-04-12    39.766667
1959-04-13    39.766667
1959-04-14    39.766667
1959-04-15    39.766667
1959-04-16    39.766667
1959-04-17    39.766667
1959-04-18    39.766667
1959-04-19    39.766667
1959-04-20    39.766667
1959-04-21    39.766667
1959-04-22    39.766667
1959-04-23    39.766667
1959-04-24    39.766667
1959-04-25    39.766667
1959-04-26    39.766667
1959-04-27    39.766667
1959-04-28    39.766667
1959-04-29    39.766667
1959-04-30    39.766667
Freq: D, Name: ARMA(0,0) Predictions, dtype: float64
```

2.2.6 Plot predictions against known values

[55]:
```
'''
So now when you run them both, we're right now comparing our predictive values␣
 ↪to the true test values.

And keep in mind, we're just using an ARMA model.

So there's an auto regressive component and a moving average component.

And essentially what we're reading here is that since our starting data set␣
 ↪really exhibited no trend
```

```
or no seasonal component. And we confirm that with the augmented Dicky Fuller␣
↪test,

this prediction actually makes sense, its essentially just showing you its␣
↪forecasted average value.

'''
title = 'Daily Total Female Births'
ylabel='Births'
xlabel='' # we don't really need a label here

ax = test['Births'].plot(legend=True,figsize=(12,6),title=title)
predictions.plot(legend=True)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel);
```



Since our starting dataset exhibited no trend or seasonal component, this prediction makes sense. In the next section we'll take additional steps to evaluate the performance of our predictions, and forecast into the future.

[56]:
```
test.mean()
```

[56]:
```
Births    39.833332
dtype: float32
```

[57]:
```
'''
And recall that our model had never even seen the test set.

But if you take a look at our predictions and grab the mean off of that, it's␣
 ↪extremely close. It's 39.76.

So essentially our model, while not being able to predict the noise that␣
 ↪happened during these dates,

it was able to correctly predict extremely, close to the actual mean average␣
 ↪for that next month data,

which is pretty impressive given how simple this model is.

we discussed the AMA model and we can see here those results essentially␣
 ↪showing kind of

a simple mean around the average value.

And the prediction of that mean was actually quite accurate.
'''
predictions.mean()
```

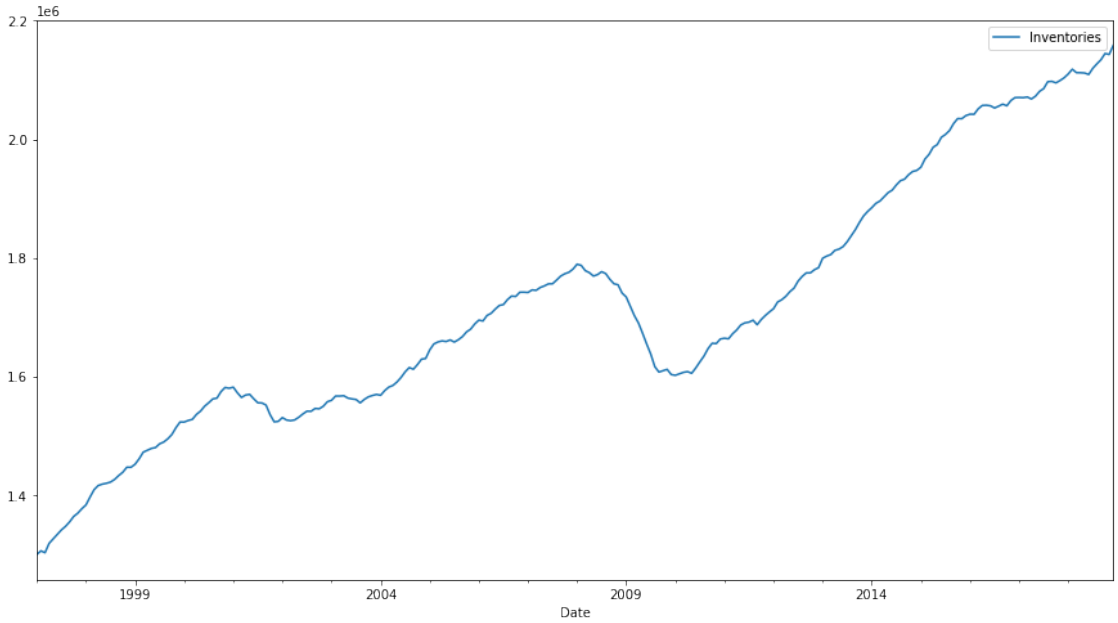
[57]:
```
39.76666666666666
```

2.3 Autoregressive Integrated Moving Average - ARIMA(p,d,q)

The steps are the same as for ARMA(p,q), except that we'll apply a differencing component to make the dataset stationary. First let's take a look at the Real Manufacturing and Trade Inventories dataset. ### Plot the Source Data

[58]:
```
df2.plot(figsize=(15,8))
```

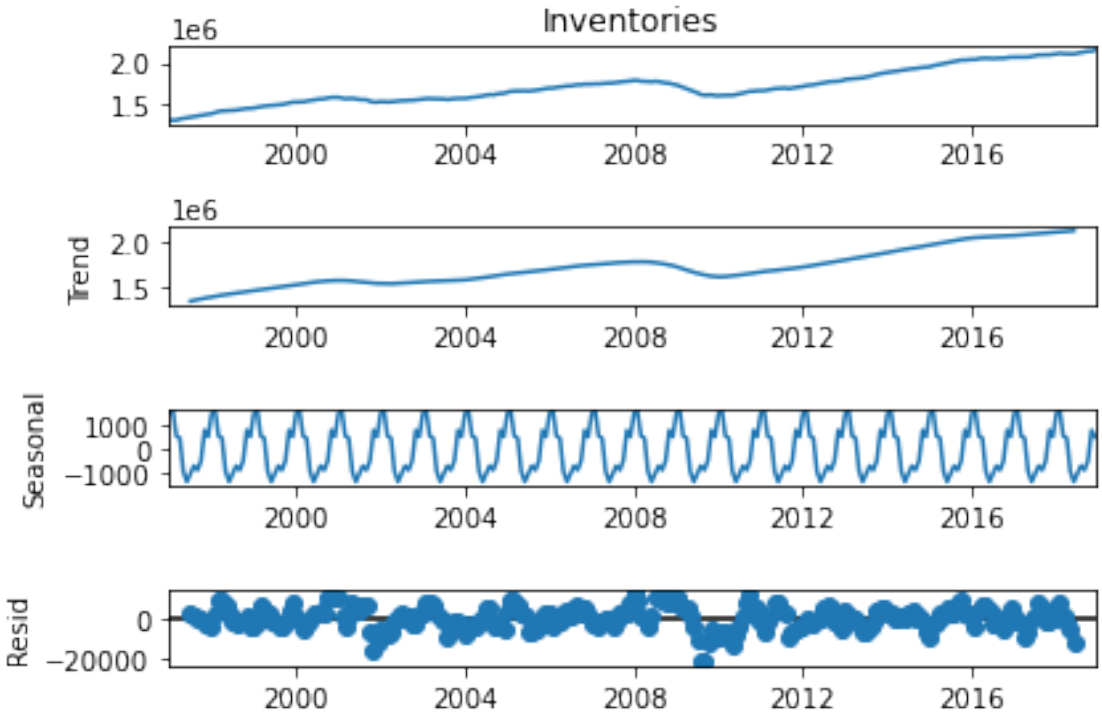
[58]:
```
<AxesSubplot:xlabel='Date'>
```



[59]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
```

[60]:
```
'''
OK, so let's go ahead and run an ETS decomposition, let's kind of pretend that␣
 ↪we didn't know if it

was seasonally adjusted or not.

And we'll notice that there is a tiny seasonal component to this.

And I'm going to call seasonal decompose on the entire dataset, so that's just␣
 ↪the inventories column.

Of the two, and I'll use an additive model, because it doesn't seem to be␣
 ↪growing exponentially,


you may notice that we actually do have the seasonal component.

The main thing to notice there, however, is the actual values and range, the␣
 ↪seasonal component.

So it looks like the seasonal component goes from  -1000 to positive 1000.

And the actual trend and observed values are magnitudes higher.
```

```
So you can imagine that seasonal effects in the plus or minus 1000 range won't␣
↪have enough of an effect

to actually use a full seasonal model.

So even if it's there, we're going to ignore it because it's just orders of␣
↪magnitude less than the

actual trend and observed values.

'''
result=seasonal_decompose(df2["Inventories"],model="add")
result.plot();
```



[61]:
```
'''
So let's go ahead and use PMDARIAM to determine the dream orders.

Hey, we think you should use an ARIMA(0,1,0).

And we think the data should be different one time.

'''
auto_arima(df2["Inventories"],seasonal=False).summary()
```

[61]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  264
Model:               SARIMAX(0, 1, 0)   Log Likelihood               -2672.018
Date:                Tue, 02 Nov 2021   AIC                           5348.037
Time:                        16:13:37   BIC                           5355.181
Sample:                             0   HQIC                          5350.908
                                  - 264
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept   3258.3802    470.991      6.918      0.000    2335.255    4181.506
sigma2       3.91e+07   2.95e+06     13.250      0.000    3.33e+07    4.49e+07
===================================================================================
Ljung-Box (L1) (Q):                  82.61   Jarque-Bera (JB):
100.74
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):               0.86   Skew:
-1.15
Prob(H) (two-sided):                  0.48   Kurtosis:
4.98
===================================================================================

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

This suggests that we should fit an ARIMA(0,1,0) model to best forecast future values of the series. Before we train the model, let's look at augmented Dickey-Fuller Test, and the ACF/PACF plots to see if they agree. These steps are optional, and we would likely skip them in practice.

[62]:
```
adf_test(df2["Inventories"])
```

```
Augmented Dickey-Fuller Test:
ADF test statistic       -0.087684
p-value                   0.950652
# lags used               5.000000
# observations          258.000000
critical value (1%)      -3.455953
critical value (5%)      -2.872809
critical value (10%)     -2.572775
```

```
Weak evidence against the null hypothesis
Fail to reject the null hypothesis
Data has a unit root and is non-stationary
```

[63]:
```
'''
OK, so let's go ahead and run the Augmented Thicky Fuller test on that first␣
 ↪difference.


So when we actually difference that by one augmented Dickey Fuller test also␣
 ↪confirmed that after a

different sync by one, the data has no unit and is stationary.

'''
from  statsmodels.tsa.statespace.tools import diff
df2["diff_1"]=diff(df2["Inventories"],k_diff=1)
```

[64]:
```
'''
And basically our Dicky Fuller test is agreeing with what Arima suggested, that␣
 ↪in order to use

the AR and MA components, we have to make sure that the data is stationary.

And in order to make sure that data stationary, we should use and order 1 or␣
 ↪difference in term of

1 here.

So when we actually difference that by one augmented Dickey Fuller test also␣
 ↪confirmed that after a

different sync by one, the data has no unit and is stationary.

'''
adf_test(df2["diff_1"])
```

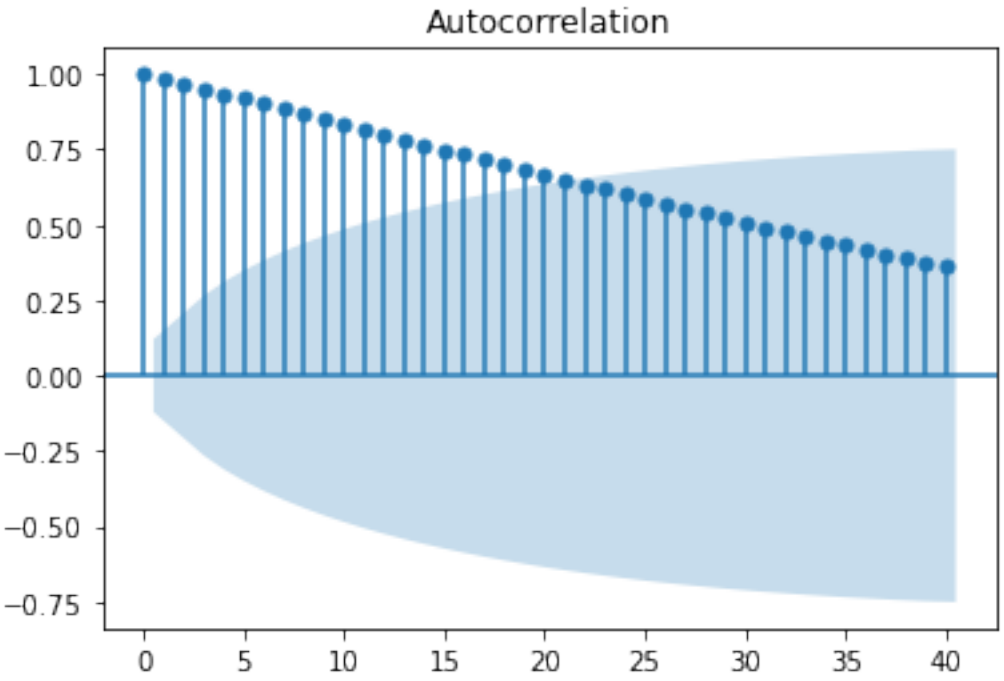
```
Augmented Dickey-Fuller Test:
ADF test statistic       -3.412249
p-value                   0.010548
# lags used               4.000000
# observations          258.000000
critical value (1%)      -3.455953
critical value (5%)      -2.872809
critical value (10%)     -2.572775
Strong evidence against the null hypothesis
Reject the null hypothesis
```

```
Data has no unit root and is stationary
```

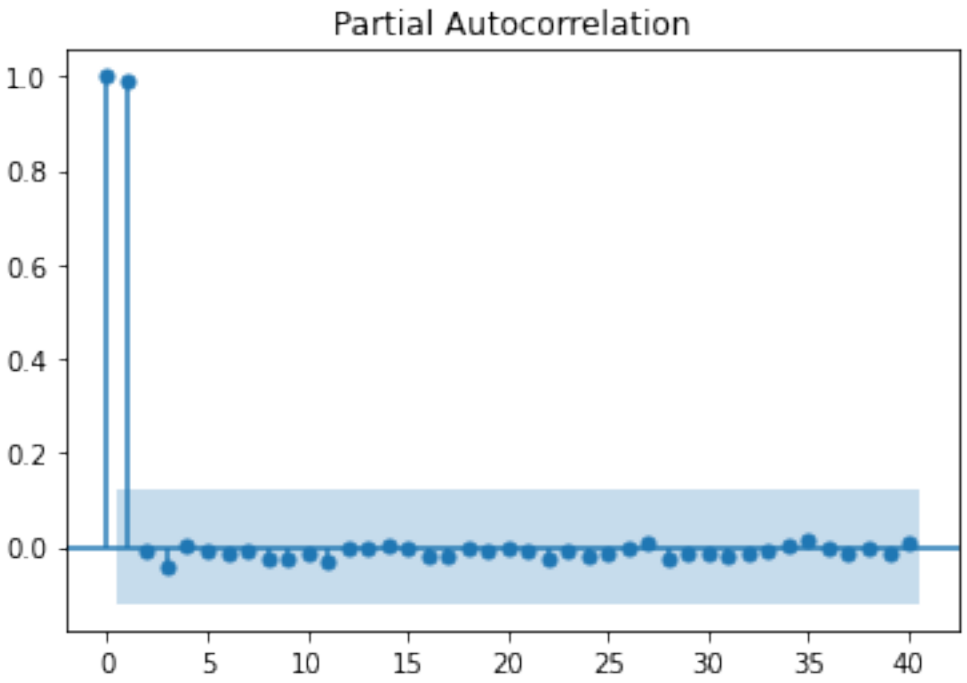
This confirms that we reached stationarity after the first difference. ### Run the ACF and PACF plots A PACF Plot can reveal recommended AR(p) orders, and an ACF Plot can do the same for MA(q) orders. Alternatively, we can compare the stepwise Akaike Information Criterion (AIC) values across a set of different (p,q) combinations to choose the best combination.

[65]:
```
'''
Let's go ahead and run the ACF and PACF plots in order to see how he would␣
 ↪actually grab the same P

and Q Order values.That's P for the AR component and Q for the MA  component.


All right, so now comes the complex procedure of if you are trying to do this␣
 ↪in a classical manner,

how would you actually choose your P and Q values from these two plots?

So recall that with a full Arima model, we have three values to choose from.

The P value, the D and the Q value.

The D is the easiest to choose.

The way we do that is we basically keep defensing until the data is stationary␣
 ↪and we saw that with

running our augmented Dicky Fuller test.

After one differcing , the data has no unit route and is stationary.


If you don't actually have auto arima, you just keep running this augmented␣
 ↪Deqi fuller test for different,

different values until you actually see that the data is stationary.


The much harder ones is trying to decide which component is more significant,␣
 ↪this auto regressive

component or this moving average component.\


Rule 6: If the PACF of the differenced series displays a sharp cutoff and/or␣
 ↪the lag-1 autocorrelation is positive
```

```
--i.e., if the series appears slightly "underdifferenced"--then consider adding␣
 ↪an AR term to the model.
The lag at which the PACF cuts off is the indicated number of AR terms.

And then we have to decide what is our M.A term going to be?

And the way we decide my term using these two plots is actually going to lead␣
 ↪to a slightly different

result than the auto arima.

If the PACF displays a sharp cutoff while the ACF decays more slowly (i.e., has␣
 ↪significant spikes at higher lags),
we say that the stationarized series displays an "AR signature," meaning that␣
 ↪the autocorrelation pattern
can be explained more easily by adding AR terms than by adding MA terms.

So this is actually starting to imply that maybe we just may equal to zero␣
 ↪because of this behavior,

or maybe one if we want to see a little bit of an m a signature.

'''
plot_acf(df2["Inventories"],lags=40);
```



[66]:
```
'''
So if we take a look at this notebook, we have partial autocorrelation.

And we have Lagat zero, then Lagat one, and then the cutoff happens, so this␣
 ↪basically implies because

the partial autocorrelation plot has that steep cut off at leg one, then the R␣
 ↪component should be one.

Now, that doesn't mean that the R component of lag one happens to be the best␣
 ↪choice.

You have to run a bunch of different models to do that, which is what Auto␣
 ↪Arima does.

'''
plot_pacf(df2["Inventories"],lags=40);
```



This tells us that the AR component should be more important than MA. From the Duke University Statistical Forecasting site: > If the PACF displays a sharp cutoff while the ACF decays more slowly (i.e., has significant spikes at higher lags), we say that the stationarized series displays an "AR signature," meaning that the autocorrelation pattern can be explained more easily by adding AR terms than by adding MA terms.

Let's take a look at pmdarima.auto_arima done stepwise to see if having p and q terms the same

still makes sense:

[67]:
```
stepwise_fit = auto_arima(df2['Inventories'], start_p=0, start_q=0,
                          max_p=2, max_q=2, m=12,
                          seasonal=False,
                          d=None, trace=True,
                          error_action='ignore',   # we don't want to know if␣
 ↪an order does not work
                          suppress_warnings=True,  # we don't want convergence␣
 ↪warnings
                          stepwise=True)           # set to stepwise

stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=5348.037, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=5399.843, Time=0.03 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=5350.241, Time=0.02 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=5409.217, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=5378.835, Time=0.09 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0] intercept
Total fit time: 0.167 seconds
```

[67]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  264
Model:               SARIMAX(0, 1, 0)   Log Likelihood               -2672.018
Date:                Tue, 02 Nov 2021   AIC                           5348.037
Time:                        16:13:37   BIC                           5355.181
Sample:                             0   HQIC                          5350.908
                                  - 264
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept   3258.3802    470.991      6.918      0.000    2335.255    4181.506
sigma2       3.91e+07   2.95e+06     13.250      0.000    3.33e+07    4.49e+07
===================================================================================
===
Ljung-Box (L1) (Q):                  82.61   Jarque-Bera (JB):
100.74
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):               0.86   Skew:
-1.15
```

```
Prob(H) (two-sided):                  0.48   Kurtosis:
4.98
================================================================================
===

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

[68]:
```
'''
Looks like it's 264 rows.

So I'm going to grab just that last year for testing.
'''
len(df2)
```

[68]:
```
264
```

[69]:
```
train = df2.iloc[:252].astype('float32')
test=df2.iloc[252:].astype('float32')
```

[70]:
```
model=ARIMA(train["Inventories"],order=(1,1,1))
results=model.fit()
results.summary()
```

[70]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                             ARIMA Model Results
==============================================================================
Dep. Variable:          D.Inventories   No. Observations:                  251
Model:                 ARIMA(1, 1, 1)   Log Likelihood               -2486.395
Method:                       css-mle   S.D. of innovations           4845.028
Date:                Tue, 02 Nov 2021   AIC                           4980.790
Time:                        16:13:37   BIC                           4994.892
Sample:                    02-01-1997   HQIC                          4986.465
                         - 12-01-2017
==============================================================================
=======
                        coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-------
const                3197.5697   1344.869      2.378      0.017     561.674
5833.465
ar.L1.D.Inventories     0.9026      0.039     23.010      0.000       0.826
0.979
```

```
ma.L1.D.Inventories     -0.5581       0.079     -7.048       0.000       -0.713
-0.403
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.1080           +0.0000j            1.1080            0.0000
MA.1            1.7918           +0.0000j            1.7918            0.0000
-----------------------------------------------------------------------------
"""
```

[71]:
```
start=len(train)
end=len(train)+len(test)-1

start,end
```

[71]:
```
(252, 263)
```

[72]:
```
'''
And you notice that there's essentially two string codes you can pass in␣
 ↪linnear or levels now linear

is going to be a linear prediction in terms of the differenced endogenous␣
 ↪variables.

So what does that mean?

Recall that we're now running in a arima model with a differenced term.

Now, the question is, do we want these values returns to come off of that␣
 ↪Differenced Time series

or do we want them to reflect the original data?

And typically, you're more interested in the original data, not this kind of␣
 ↪differenced version of it.

So if we again take a look here at the string codes, the default is linear,␣
 ↪which will return back

to different data.It would essentially return back the predictions difference␣
 ↪by one


And we want the real predictions.
```

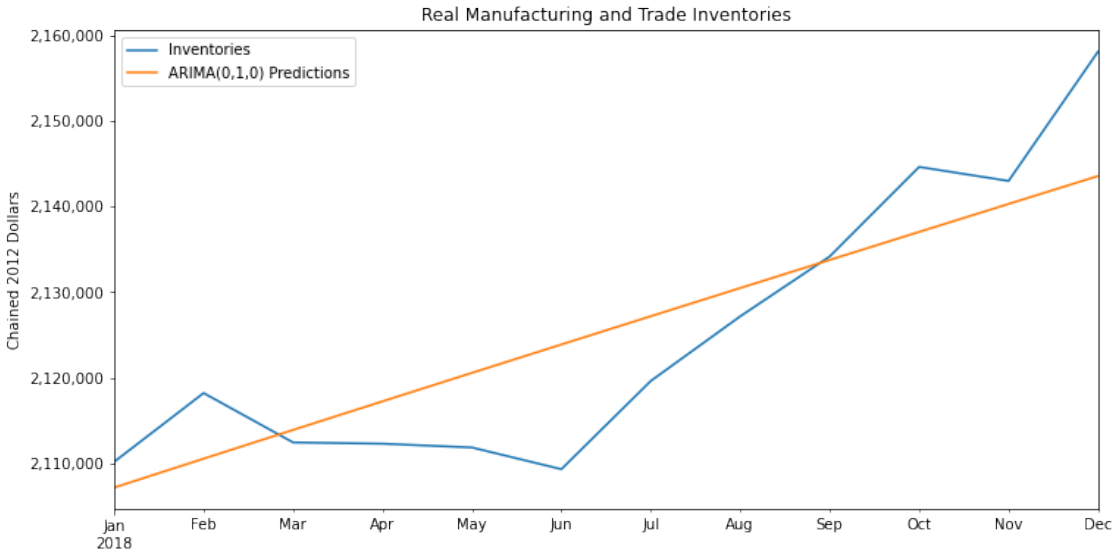
```
So we'll go ahead and specify levels, which is to predict the levels of the␣
 ↪original endogenous variables.

'''
predictions = results.predict(start,end,typ='levels').rename('ARIMA(0,1,0)␣
 ↪Predictions')
```

Passing dynamic=False means that forecasts at each point are generated using the full history up to that point (all lagged values).

Passing typ='levels' predicts the levels of the original endogenous variables. If we'd used the default typ='linear' we would have seen linear predictions in terms of the differenced endogenous variables.

For more information on these arguments visit https://www.statsmodels.org/stable/generated/statsmodels.tsa.arim

[74]:
```
# HERE'S A TRICK TO ADD COMMAS TO Y-AXIS TICK VALUES
import matplotlib.ticker as ticker
formatter = ticker.StrMethodFormatter('{x:,.0f}')
# Plot predictions against known values
title = 'Real Manufacturing and Trade Inventories'
ylabel='Chained 2012 Dollars'
xlabel='' # we don't really need a label here

ax = test['Inventories'].plot(legend=True,figsize=(12,6),title=title)
predictions.plot(legend=True)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
ax.yaxis.set_major_formatter(formatter);
```



[75]:
```
# Compare predictions to expected values
for i in range(len(predictions)):
    print(f"predicted={predictions[i]:<11.10},␣
 ↪expected={test['Inventories'][i]}")
```

```
predicted=2107148.334, expected=2110158.0
predicted=2110526.202, expected=2118199.0
predicted=2113886.501, expected=2112427.0
predicted=2117230.943, expected=2112276.0
predicted=2120561.073, expected=2111835.0
predicted=2123878.286, expected=2109298.0
predicted=2127183.84 , expected=2119618.0
predicted=2130478.872, expected=2127170.0
predicted=2133764.407, expected=2134172.0
predicted=2137041.37 , expected=2144639.0
predicted=2140310.597, expected=2143001.0
predicted=2143572.841, expected=2158115.0
```

[76]:
```
from sklearn.metrics import mean_squared_error

error = mean_squared_error(test['Inventories'], predictions)
print(f'ARIMA(1,1,1) MSE Error: {error:11.10}')
```

```
ARIMA(1,1,1) MSE Error:  60677827.4
```

[77]:
```
from statsmodels.tools.eval_measures import rmse

error = rmse(test['Inventories'], predictions)
print(f'ARIMA(1,1,1) RMSE Error: {error:11.10}')
```

```
ARIMA(1,1,1) RMSE Error: 7789.597384
```

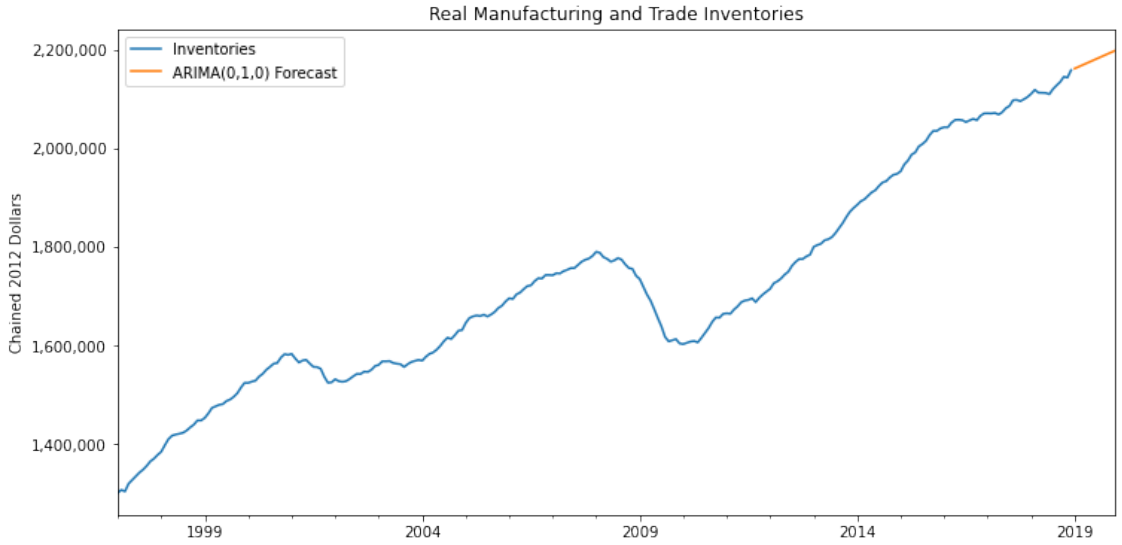
[81]:
```
'''
Test mean and predictions mean are really close
'''
test["Inventories"].mean(), predictions.mean()
```

[81]:
```
(2125075.75, 2125465.272049339)
```

2.3.1 Retrain the model on the full data, and forecast the future

[79]:
```
model = ARIMA(df2['Inventories'].astype('float32'),order=(0,1,0))
results = model.fit()
fcast = results.predict(start=len(df2),end=len(df2)+11,typ='levels').
 ↪rename('ARIMA(0,1,0) Forecast')
```

[80]:
```
# Plot predictions against known values
title = 'Real Manufacturing and Trade Inventories'
ylabel='Chained 2012 Dollars'
xlabel='' # we don't really need a label here

ax = df2['Inventories'].plot(legend=True,figsize=(12,6),title=title)
fcast.plot(legend=True)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
ax.yaxis.set_major_formatter(formatter);
```



[ ]: