05-ARMA-and-ARIMA

October 19, 2022

For more information, visit us at www.pieriandata.com

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

2.1 Perform standard imports and load datasets

[1]:
```
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")

# 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 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.

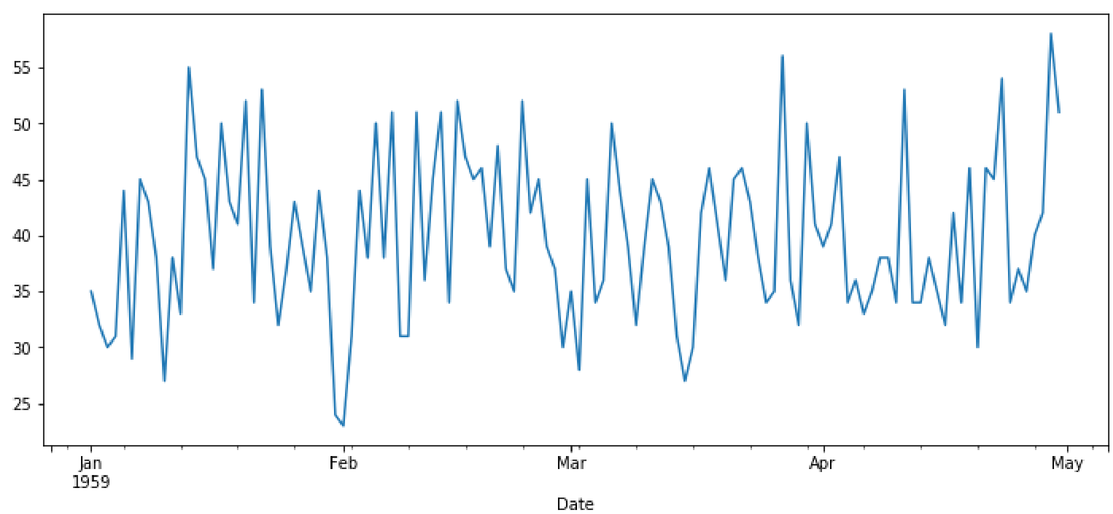
[2]:
```
from statsmodels.tsa.stattools import adfuller

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")
```

2.3 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

[3]:
```
df1['Births'].plot(figsize=(12,5));
```



2.3.1 Run the augmented Dickey-Fuller Test to confirm stationarity

[4]:
```
adf_test(df1['Births'])
```

```
Augmented Dickey-Fuller Test: None
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
```

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

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

[5]:
```
auto_arima(df1['Births'],seasonal=False).summary()
```

[5]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  120
Model:                     ARMA(2, 2)   Log Likelihood                -405.370
Method:                       css-mle   S.D. of innovations              6.991
Date:                Sat, 23 Mar 2019   AIC                            822.741
Time:                        12:02:45   BIC                            839.466
Sample:                             0   HQIC                           829.533

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         39.8162      0.108    368.841      0.000      39.605      40.028
ar.L1.y        1.8568      0.081     22.933      0.000       1.698       2.016
ar.L2.y       -0.8814      0.073    -12.030      0.000      -1.025      -0.738
ma.L1.y       -1.8634      0.109    -17.126      0.000      -2.077      -1.650
ma.L2.y        0.8634      0.108      8.020      0.000       0.652       1.074
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0533           -0.1582j            1.0652           -0.0237
AR.2            1.0533           +0.1582j            1.0652            0.0237
```

```
MA.1             1.0000           +0.0000j           1.0000           0.0000
MA.2             1.1583           +0.0000j           1.1583           0.0000
-----------------------------------------------------------------------------
"""
```

2.3.3 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.

[6]:
```python
# Set one month for testing
train = df1.iloc[:90]
test = df1.iloc[90:]
```

2.3.4 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.

[7]:
```python
model = ARMA(train['Births'],order=(2,2))
results = model.fit()
results.summary()
```

[7]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                              ARMA Model Results
==============================================================================
Dep. Variable:                 Births   No. Observations:                   90
Model:                     ARMA(2, 2)   Log Likelihood                -307.905
Method:                       css-mle   S.D. of innovations              7.405
Date:                Sat, 23 Mar 2019   AIC                            627.809
Time:                        12:08:30   BIC                            642.808
Sample:                    01-01-1959   HQIC                           633.858
                         - 03-31-1959
================================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
const           39.7549      0.912     43.607      0.000      37.968      41.542
ar.L1.Births    -0.1850      1.087     -0.170      0.865      -2.315       1.945
ar.L2.Births     0.4352      0.644      0.675      0.501      -0.828       1.698
ma.L1.Births     0.2777      1.097      0.253      0.801      -1.872       2.427
ma.L2.Births    -0.3999      0.679     -0.589      0.557      -1.730       0.930
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
```

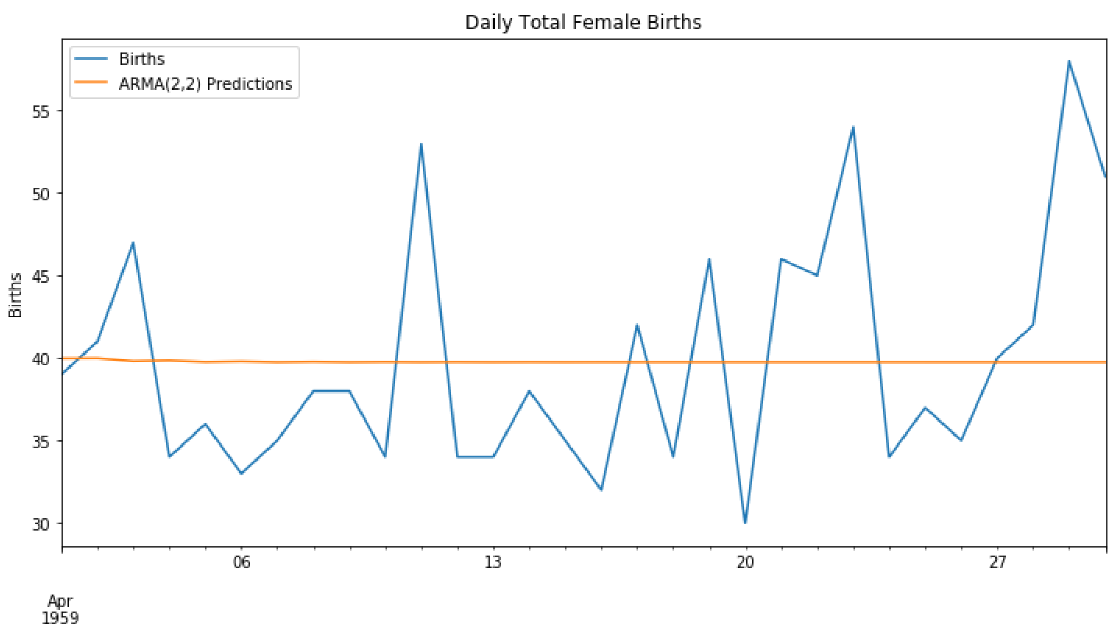
```
==============================================================================
AR.1           -1.3181        +0.0000j          1.3181          0.5000
AR.2            1.7434        +0.0000j          1.7434          0.0000
MA.1           -1.2718        +0.0000j          1.2718          0.5000
MA.2            1.9662        +0.0000j          1.9662          0.0000
------------------------------------------------------------------------------
"""
```

2.3.5 Obtain a month's worth of predicted values

[10]:
```
start=len(train)
end=len(train)+len(test)-1
predictions = results.predict(start=start, end=end).rename('ARMA(2,2)␣
↪Predictions')
```

2.3.6 Plot predictions against known values

[11]:
```
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);
```

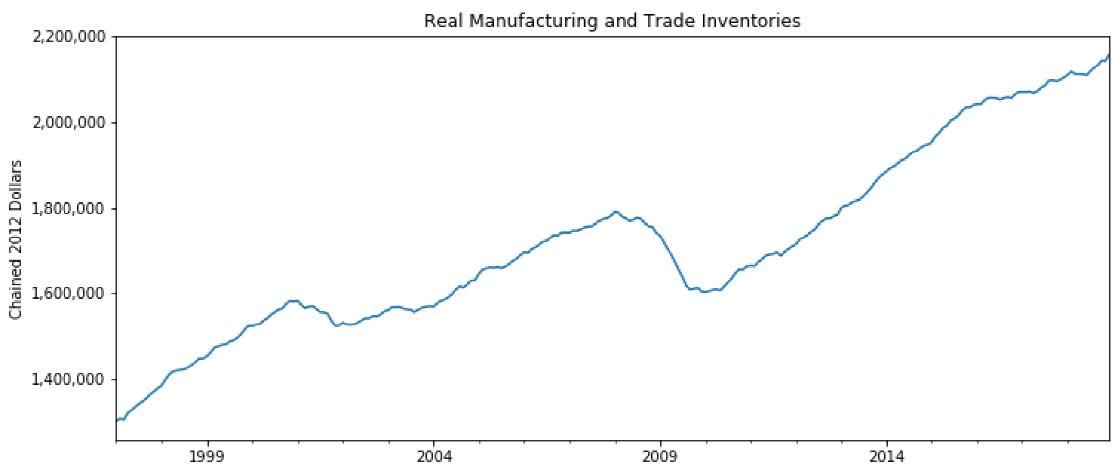


Since our starting dataset exhibited no trend or seasonal component, this prediction makes sense. In the next section we'll take additional steps to evaluate the performance of our predictions, and forecast into the future.

2.4 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

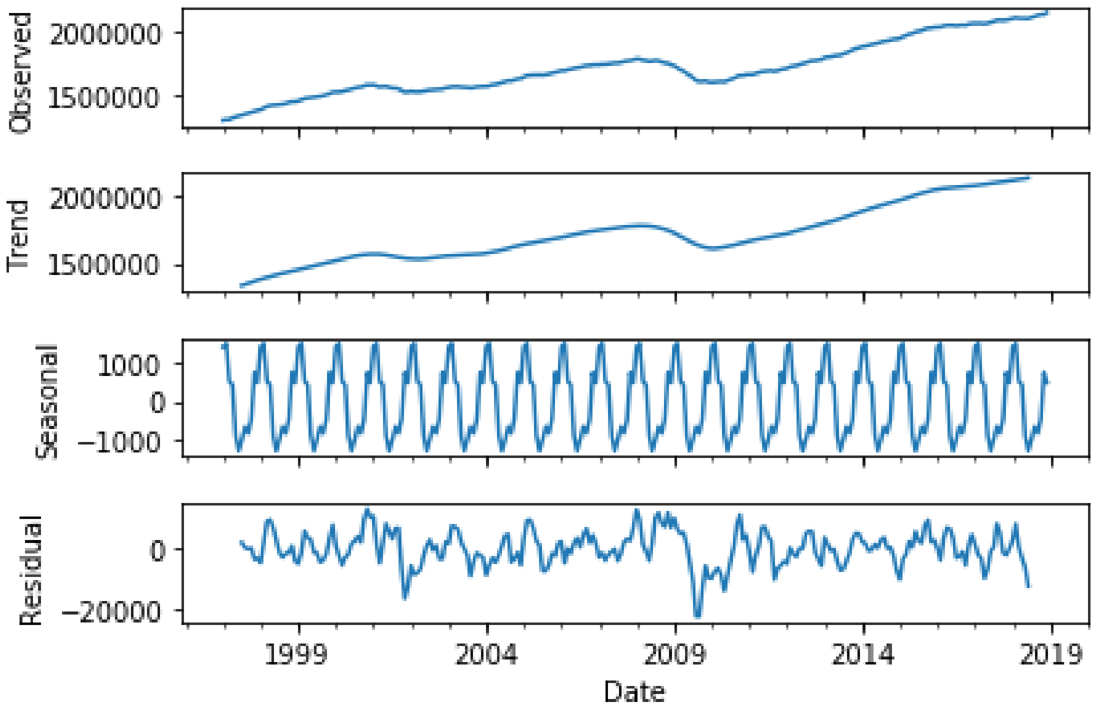
[12]:
```
# HERE'S A TRICK TO ADD COMMAS TO Y-AXIS TICK VALUES
import matplotlib.ticker as ticker
formatter = ticker.StrMethodFormatter('{x:,.0f}')

title = 'Real Manufacturing and Trade Inventories'
ylabel='Chained 2012 Dollars'
xlabel='' # we don't really need a label here

ax = df2['Inventories'].plot(figsize=(12,5),title=title)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
ax.yaxis.set_major_formatter(formatter);
```



2.4.1 Run an ETS Decomposition (optional)

We probably won't learn a lot from it, but it never hurts to run an ETS Decomposition plot.

[13]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(df2['Inventories'], model='additive')  #␣
↪model='add' also works
result.plot();
```



Here we see that the seasonal component does not contribute significantly to the behavior of the series. ### Use pmdarima.auto_arima to determine ARIMA Orders

[14]:
```
auto_arima(df2['Inventories'],seasonal=False).summary()
```

[14]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                             ARIMA Model Results
==============================================================================
Dep. Variable:                    D.y   No. Observations:                  263
Model:                 ARIMA(1, 1, 1)   Log Likelihood               -2610.252
Method:                       css-mle   S.D. of innovations           4938.258
Date:                Sat, 23 Mar 2019   AIC                           5228.505
Time:                        12:18:53   BIC                           5242.794
Sample:                             1   HQIC                          5234.247

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
```

```
-------------------------------------------------------------------------------
const       3472.9857   1313.669      2.644      0.009    898.241    6047.731
ar.L1.D.y      0.9037      0.039     23.414      0.000      0.828       0.979
ma.L1.D.y     -0.5732      0.076     -7.545      0.000     -0.722      -0.424
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.1065           +0.0000j            1.1065            0.0000
MA.1            1.7446           +0.0000j            1.7446            0.0000
-----------------------------------------------------------------------------
"""
```

This suggests that we should fit an ARIMA(1,1,1) 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.

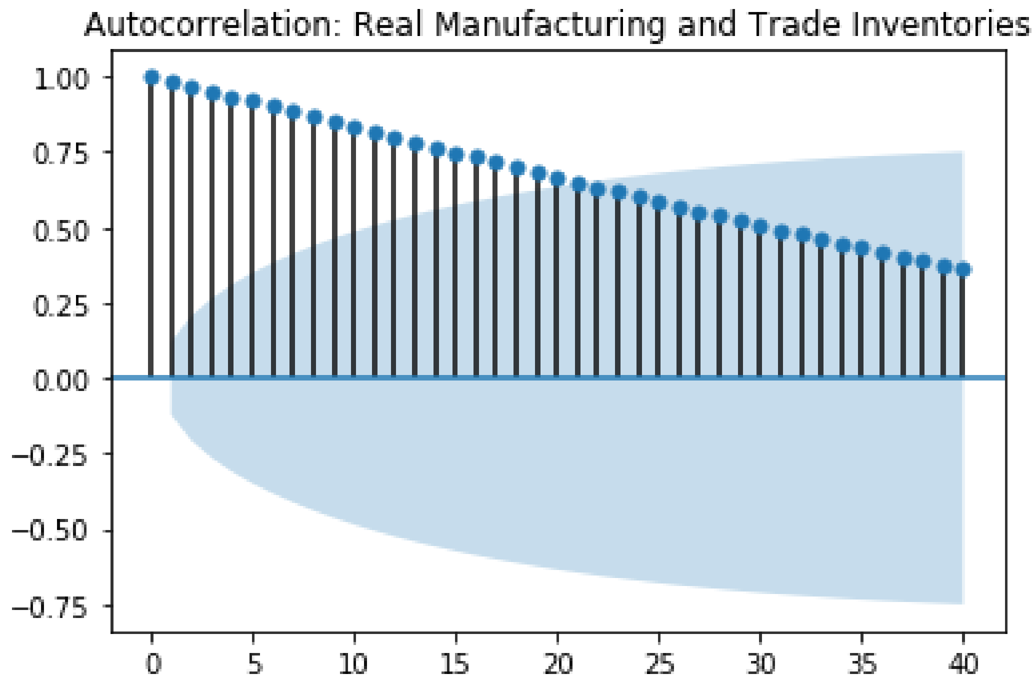
2.4.2 Run the augmented Dickey-Fuller Test on the First Difference

[15]:
```python
from statsmodels.tsa.statespace.tools import diff
df2['d1'] = diff(df2['Inventories'],k_diff=1)

# Equivalent to:
# df1['d1'] = df1['Inventories'] - df1['Inventories'].shift(1)

adf_test(df2['d1'],'Real Manufacturing and Trade Inventories')
```

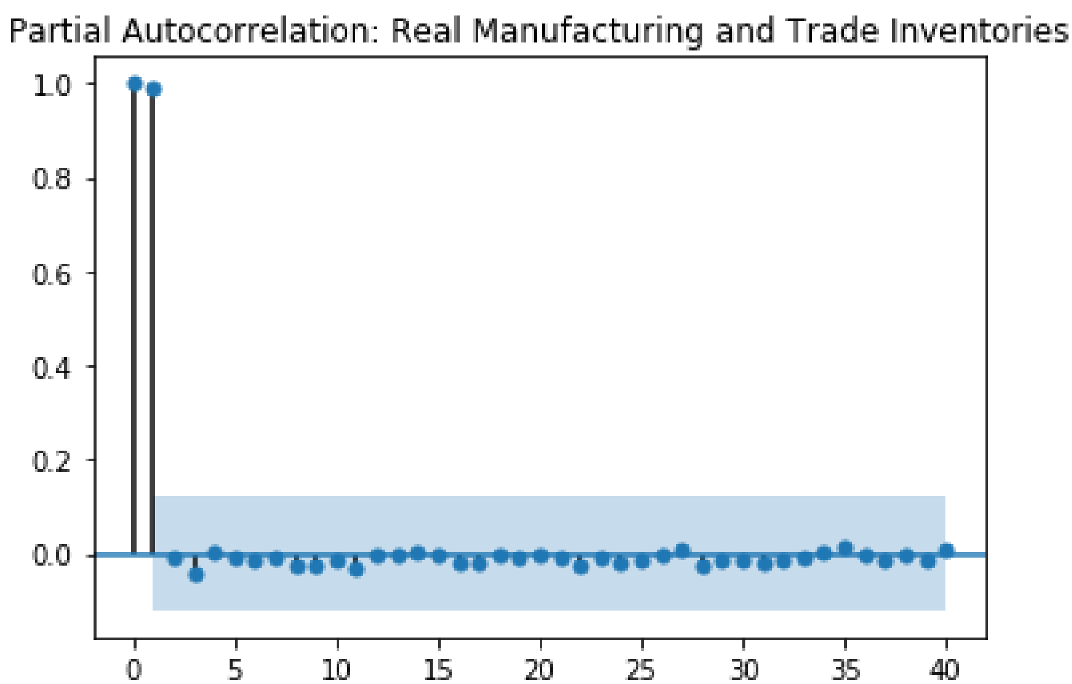
```
Augmented Dickey-Fuller Test: Real Manufacturing and Trade Inventories
ADF test statistic       -3.412249
p-value                   0.010548
# lags used               4.000000
# observations          258.000000
critical value (1%)      -3.455953
critical value (5%)      -2.872809
critical value (10%)     -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.

[16]:
```python
title = 'Autocorrelation: Real Manufacturing and Trade Inventories'
lags = 40
plot_acf(df2['Inventories'],title=title,lags=lags);
```



[17]:
```
title = 'Partial Autocorrelation: Real Manufacturing and Trade Inventories'
lags = 40
plot_pacf(df2['Inventories'],title=title,lags=lags);
```



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:

[18]:
```
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()
```

```
Fit ARIMA: order=(0, 1, 0); AIC=5348.037, BIC=5355.181, Fit time=0.004 seconds
Fit ARIMA: order=(1, 1, 0); AIC=5250.883, BIC=5261.599, Fit time=0.050 seconds
Fit ARIMA: order=(0, 1, 1); AIC=5283.095, BIC=5293.811, Fit time=0.020 seconds
Fit ARIMA: order=(2, 1, 0); AIC=5240.553, BIC=5254.842, Fit time=0.100 seconds
Fit ARIMA: order=(2, 1, 1); AIC=5229.528, BIC=5247.389, Fit time=0.104 seconds
Fit ARIMA: order=(1, 1, 1); AIC=5228.505, BIC=5242.794, Fit time=0.109 seconds
Fit ARIMA: order=(1, 1, 2); AIC=5229.289, BIC=5247.150, Fit time=0.157 seconds
Fit ARIMA: order=(2, 1, 2); AIC=nan, BIC=nan, Fit time=nan seconds
Total fit time: 0.563 seconds
```

[18]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                             ARIMA Model Results
==============================================================================
Dep. Variable:                    D.y   No. Observations:                  263
Model:                 ARIMA(1, 1, 1)   Log Likelihood               -2610.252
Method:                       css-mle   S.D. of innovations           4938.258
Date:                Sat, 23 Mar 2019   AIC                           5228.505
Time:                        12:19:52   BIC                           5242.794
Sample:                             1   HQIC                          5234.247

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       3472.9857   1313.669      2.644      0.009     898.241    6047.731
ar.L1.D.y      0.9037      0.039     23.414      0.000       0.828       0.979
```

```
ma.L1.D.y      -0.5732       0.076      -7.545      0.000      -0.722      -0.424
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.1065           +0.0000j            1.1065            0.0000
MA.1            1.7446           +0.0000j            1.7446            0.0000
-----------------------------------------------------------------------------
"""
```

Looks good from here! Now let's train & test the ARIMA(1,1,1) model, evaluate it, then produce a forecast of future values. ### Split the data into train/test sets

[19]:
```python
len(df2)
```

[19]:
```
264
```

[20]:
```python
# Set one year for testing
train = df2.iloc[:252]
test = df2.iloc[252:]
```

2.4.3 Fit an ARIMA(1,1,1) Model

[21]:
```python
model = ARIMA(train['Inventories'],order=(1,1,1))
results = model.fit()
results.summary()
```

[21]:
```
<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:                Sat, 23 Mar 2019   AIC                           4980.790
Time:                        12:20:09   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   1344.871      2.378      0.018     561.671
5833.468
```

```
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
-----------------------------------------------------------------------------
"""
```

[22]:

```
# Obtain predicted values
start=len(train)
end=len(train)+len(test)-1
predictions = results.predict(start=start, end=end, dynamic=False,␣
 ↪typ='levels').rename('ARIMA(1,1,1) 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.arin

[23]:

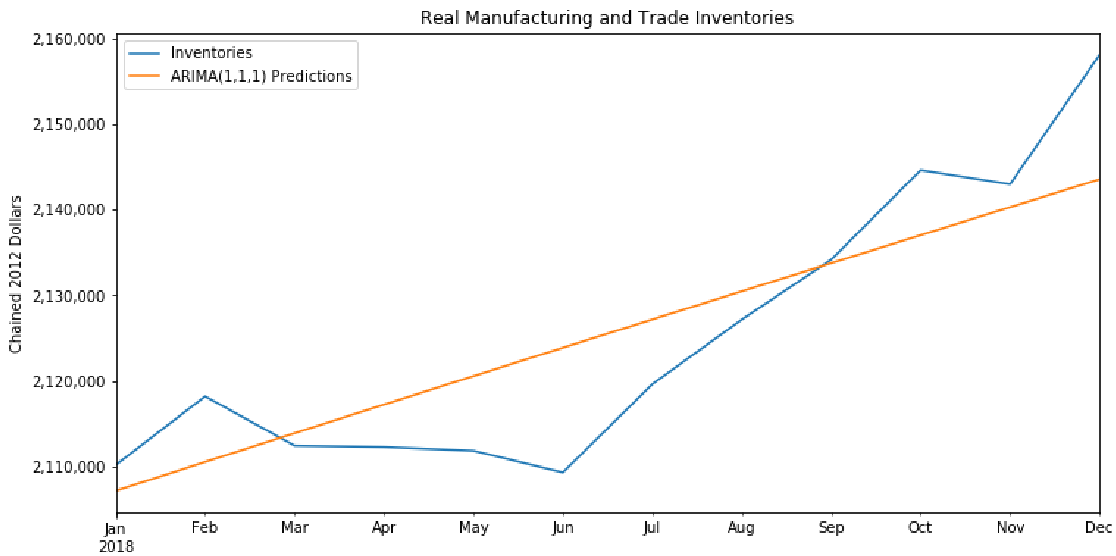
```
# Compare predictions to expected values
for i in range(len(predictions)):
    print(f"predicted={predictions[i]:<11.10},␣
 ↪expected={test['Inventories'][i]}")
```

```
predicted=2107148.333, expected=2110158
predicted=2110526.201, expected=2118199
predicted=2113886.499, expected=2112427
predicted=2117230.941, expected=2112276
predicted=2120561.071, expected=2111835
predicted=2123878.284, expected=2109298
predicted=2127183.839, expected=2119618
predicted=2130478.871, expected=2127170
predicted=2133764.405, expected=2134172
predicted=2137041.369, expected=2144639
predicted=2140310.596, expected=2143001
predicted=2143572.84 , expected=2158115
```

[24]:

```
# 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);
```



2.4.4 Evaluate the Model

[25]:
```
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: 60677824.72
```

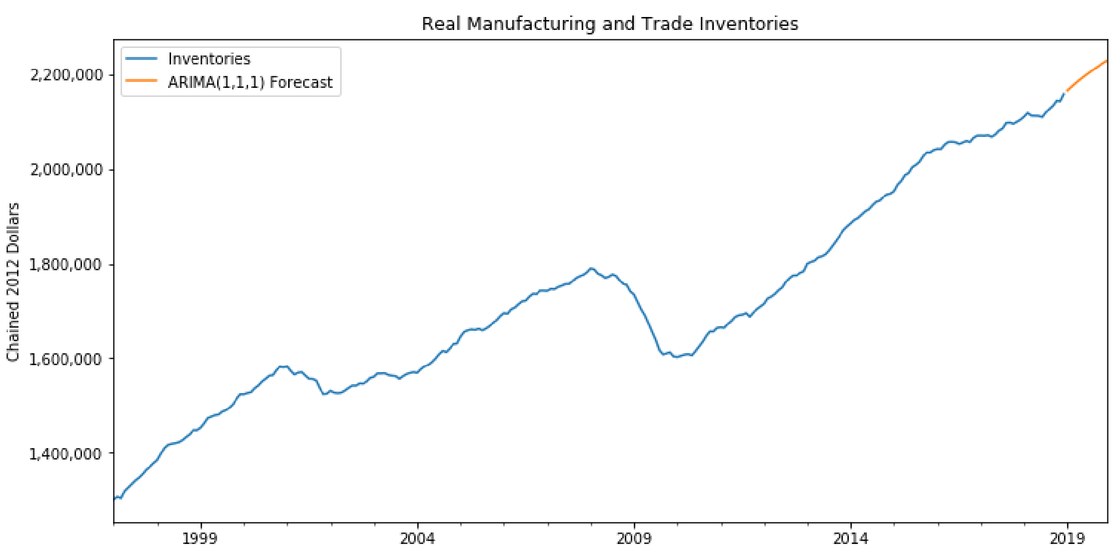
[26]:
```
from statsmodels.tools.eval_measures import rmse

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

```
ARIMA(1,1,1) RMSE Error: 7789.597212
```

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

```
[29]: model = ARIMA(df2['Inventories'],order=(1,1,1))
      results = model.fit()
      fcast = results.predict(len(df2),len(df2)+11,typ='levels').rename('ARIMA(1,1,1)␣
       ↪Forecast')
```

```
[30]: # 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);
```



2.5 Great job!