09-Vector-AutoRegressive-Moving-Average-VARMA

October 19, 2022

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

1 VARMA(p,q)

1.1 Vector Autoregressive Moving Average

This lesson picks up where VAR(p) left off.

Recall that the system of equations for a 2-dimensional VAR(1) model is:

$y_{1,t} = c_1 + \phi_{11,1}y_{1,t-1} + \phi_{12,1}y_{2,t-1} + \varepsilon_{1,t} \qquad y_{2,t} = c_2 + \phi_{21,1}y_{1,t-1} + \phi_{22,1}y_{2,t-1} + \varepsilon_{2,t}$

where the coefficient $\phi_{ii,l}$ captures the influence of the lth lag of variable y_i on itself, the coefficient $\phi_{ij,l}$ captures the influence of the lth lag of variable y_j on y_i. Most importantly, $\varepsilon_{1,t}$ and $\varepsilon_{2,t}$ are white noise processes that may be correlated.

In a VARMA(p,q) model we give the error terms ε_t a moving average representation of order q.

1.1.1 Formulation

We've seen that an autoregressive moving average ARMA(p,q) model is described by the following:

$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$

A K-dimensional VARMA model of order (p, q) considers each variable y_K in the system.

For example, the system of equations for a 2-dimensional VARMA(1,1) model is:

$y_{1,t} = c_1 + \phi_{11,1}y_{1,t-1} + \phi_{12,1}y_{2,t-1} + \theta_{11,1}\varepsilon_{1,t-1} + \theta_{12,1}\varepsilon_{2,t-1} + \varepsilon_{1,t} \qquad y_{2,t} = c_2 + \phi_{21,1}y_{1,t-1} + \phi_{22,1}y_{2,t-1} + \theta_{21,1}\varepsilon_{1,t-1} + \theta_{22,1}\varepsilon_{2,t-1} + \varepsilon_{2,t}$

where the coefficient $\theta_{ii,l}$ captures the influence of the lth lag of error ε_i on itself, the coefficient $\theta_{ij,l}$ captures the influence of the lth lag of error ε_j on ε_i, and $\varepsilon_{1,t}$ and $\varepsilon_{2,t}$ are residual white noise.

The general steps involved in building a VARMA model are: * Examine the data * Visualize the data * Test for stationarity * If necessary, transform the data to make it stationary * Select the appropriate p and q orders * Instantiate the model and fit it to a training set * If necessary, invert the earlier transformation * Evaluate model predictions against a known test set * Forecast the future

Related Functions:

varmax.VARMAX(endog[, exog, order, trend, …]) Vector Autoregressive Moving Average with eXogenous regressors model varmax.VARMAXResults(model, params[, …]) Class to hold results from fitting an VARMAX model

For Further Reading:

Statsmodels Tutorial: Time Series Analysis by State Space Methods Statsmodels Example: VARMAX models

1.1.2 Perform standard imports and load dataset

For this analysis we'll reuse our money and spending datasets. We'll look at the M2 Money Stock which is a measure of U.S. personal assets, and U.S. personal spending. Both datasets are in billions of dollars, monthly, seasonally adjusted. They span the 21 years from January 1995 to December 2015 (252 records). Sources: https://fred.stlouisfed.org/series/M2SL https://fred.stlouisfed.org/series/PCE

[1]:
```
import numpy as np
import pandas as pd
%matplotlib inline

# Load specific forecasting tools
from statsmodels.tsa.statespace.varmax import VARMAX, VARMAXResults
from statsmodels.tsa.stattools import adfuller
from pmdarima import auto_arima
from statsmodels.tools.eval_measures import rmse

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

# Load datasets
df = pd.read_csv('../Data/M2SLMoneyStock.csv',index_col=0, parse_dates=True)
df.index.freq = 'MS'

sp = pd.read_csv('../Data/PCEPersonalSpending.csv',index_col=0,␣
 ↪parse_dates=True)
sp.index.freq = 'MS'
```

1.1.3 Inspect the data

[2]:
```
df = df.join(sp)
df.head()
```

[2]:

	Money	Spending
Date		
1995-01-01	3492.4	4851.2
1995-02-01	3489.9	4850.8
1995-03-01	3491.1	4885.4
1995-04-01	3499.2	4890.2
1995-05-01	3524.2	4933.1

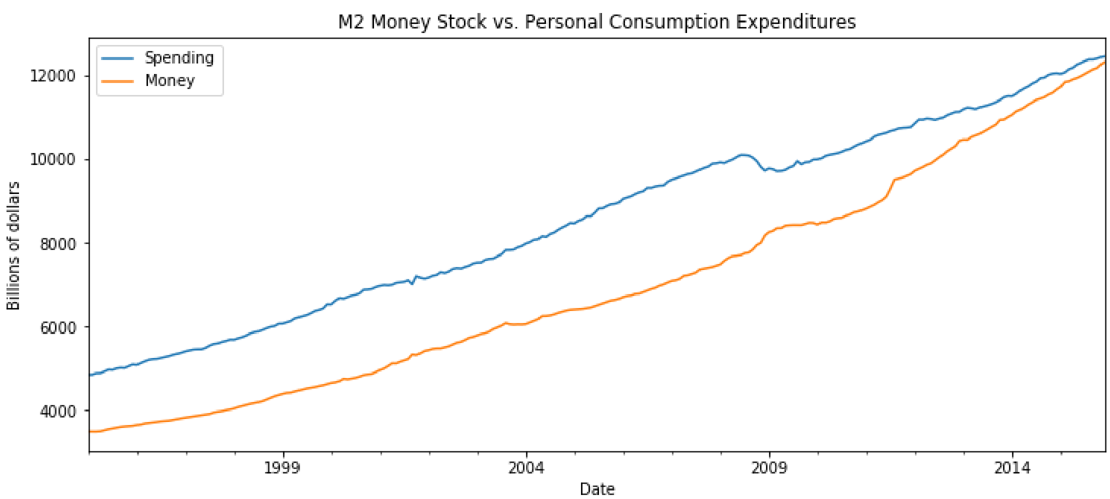
[3]:
```
df = df.dropna()
df.shape
```

[3]: (252, 2)

1.1.4 Plot the source data

[4]:
```
title = 'M2 Money Stock vs. Personal Consumption Expenditures'
ylabel='Billions of dollars'
xlabel=''

ax = df['Spending'].plot(figsize=(12,5),title=title,legend=True)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
df['Money'].plot(legend=True);
```



1.2 Test for stationarity, perform any necessary transformations

In the previous section we applied the augmented Dickey-Fuller test and found that a second-order difference achieved stationarity. In this section we'll perform the auto_arima prediction to identify optimal p and q orders.

[ ]:
```
# INCLUDED HERE IF YOU CHOOSE TO USE IT
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")
```

NOTE: When performing the auto_arima function we're likely to see a ConvergenceWarning: Maximum Likelihood optimization failed to converge. This is not unusual in models which have to estimate a large number of parameters. However, we can override the maximum iterations default of 50, and pass an arbitrarily large number with maxiter=1000. We'll see this come up again when we fit our model.

[5]:
```
auto_arima(df['Money'],maxiter=1000)
```

[5]:
```
ARIMA(callback=None, disp=0, maxiter=1000, method=None, order=(1, 2, 2),
   out_of_sample_size=0, scoring='mse', scoring_args={},
   seasonal_order=(0, 0, 0, 1), solver='lbfgs', start_params=None,
   suppress_warnings=False, transparams=True, trend=None,
   with_intercept=True)
```

[6]:
```
auto_arima(df['Spending'],maxiter=1000)
```

[6]:
```
ARIMA(callback=None, disp=0, maxiter=1000, method=None, order=(1, 1, 2),
   out_of_sample_size=0, scoring='mse', scoring_args={},
   seasonal_order=(0, 0, 0, 1), solver='lbfgs', start_params=None,
   suppress_warnings=False, transparams=True, trend=None,
   with_intercept=True)
```

It looks like a VARMA(1,2) model is recommended. Note that the d term (2 for Money, 1 for Spending) is about to be addressed by transforming the data to make it stationary. As before we'll apply a second order difference.

[7]:
```
df_transformed = df.diff().diff()
df_transformed = df_transformed.dropna()
df_transformed.head()
```

[7]:

Date	Money	Spending
1995-03-01	3.7	35.0
1995-04-01	6.9	-29.8
1995-05-01	16.9	38.1
1995-06-01	-0.3	1.5
1995-07-01	-6.2	-51.7

[8]:
```
len(df_transformed)
```

[8]: 250

1.3 Train/test/split

It is useful to define a number of observations variable for our test set. For this analysis, let's use 12 months.

[9]:
```
nobs=12
train, test = df_transformed[0:-nobs], df_transformed[-nobs:]
```

[10]:
```
print(train.shape)
print(test.shape)
```

```
(238, 2)
(12, 2)
```

1.4 Fit the VARMA(1,2) Model

This may take awhile.

[11]:
```
model = VARMAX(train, order=(1,2), trend='c')
results = model.fit(maxiter=1000, disp=False)
```

```
results.summary()
```

[11]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                              Statespace Model Results
================================================================================
=
Dep. Variable:     ['Money', 'Spending']   No. Observations:
238
Model:                        VARMA(1,2)   Log Likelihood
-2286.286
                             + intercept   AIC
4606.571
Date:                   Wed, 03 Apr 2019   BIC
4665.600
Time:                           08:25:17   HQIC
4630.361
Sample:                       03-01-1995
                            - 12-01-2014
Covariance Type:                     opg
================================================================================
===
Ljung-Box (Q):                68.42, 28.14   Jarque-Bera (JB):       547.62,
120.94
Prob(Q):                        0.00, 0.92   Prob(JB):                   0.00,
0.00
Heteroskedasticity (H):         5.61, 2.91   Skew:                      1.33,
-0.34
Prob(H) (two-sided):            0.00, 0.00   Kurtosis:                   9.94,
6.42
                            Results for equation Money
================================================================================
==
                     coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
--
const              0.2618      0.954      0.274      0.784      -1.608
2.131
L1.Money          -1.0465      4.176     -0.251      0.802      -9.232
7.139
L1.Spending        2.2414      7.952      0.282      0.778     -13.344
17.827
L1.e(Money)        0.2846      4.429      0.064      0.949      -8.397
8.966
L1.e(Spending)    -2.3643      7.962     -0.297      0.767     -17.969
13.241
```

```
L2.e(Money)       -1.3093      4.982     -0.263      0.793     -11.074
8.456
L2.e(Spending)     2.0885      7.033      0.297      0.766     -11.696
15.873
                          Results for equation Spending
================================================================================
==
                     coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
--
const              0.0641      0.212      0.303      0.762      -0.351
0.479
L1.Money          -0.2625      2.293     -0.114      0.909      -4.756
4.231
L1.Spending        0.6693      4.221      0.159      0.874      -7.604
8.943
L1.e(Money)        0.3785      2.367      0.160      0.873      -4.260
5.017
L1.e(Spending)    -1.6238      4.196     -0.387      0.699      -9.848
6.601
L2.e(Money)       -0.3753      2.535     -0.148      0.882      -5.344
4.593
L2.e(Spending)     0.6601      3.689      0.179      0.858      -6.570
7.890
                                  Error covariance matrix
================================================================================
===========
                              coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
-----------
sqrt.var.Money             25.6292     19.879      1.289      0.197     -13.334
64.592
sqrt.cov.Money.Spending   -10.1411      4.730     -2.144      0.032     -19.411
-0.871
sqrt.var.Spending          33.8594      1.934     17.509      0.000      30.069
37.650
================================================================================
===========

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

1.5 Predict the next 12 values

Unlike the VAR model we used in the previous section, the VARMAX .forecast() function won't require that we pass in a number of previous observations, and it will provide an extended DateTime index.

[12]:
```
df_forecast = results.forecast(12)
df_forecast
```

[12]:
```
                Money   Spending
2015-01-01 -11.501568  36.789494
2015-02-01 -10.883687  -4.696517
2015-03-01   1.124754  -0.222204
2015-04-01  -1.413346  -0.379833
2015-05-01   0.889492   0.180924
2015-06-01  -0.263555  -0.048266
2015-07-01   0.429407   0.101016
2015-08-01   0.038821   0.019025
2015-09-01   0.263797   0.066679
2015-10-01   0.135170   0.039517
2015-11-01   0.208897   0.055102
2015-12-01   0.166674   0.046180
```

1.6 Invert the Transformation

Remember that the forecasted values represent second-order differences. To compare them to the original data we have to roll back each difference. To roll back a first-order difference we take the most recent value on the training side of the original series, and add it to a cumulative sum of forecasted values. When working with second-order differences we first must perform this operation on the most recent first-order difference.

Here we'll use the nobs variable we defined during the train/test/split step.

[13]:
```
# Add the most recent first difference from the training side of the original␣
↪dataset to the forecast cumulative sum
df_forecast['Money1d'] = (df['Money'].iloc[-nobs-1]-df['Money'].iloc[-nobs-2])␣
↪+ df_forecast['Money'].cumsum()

# Now build the forecast values from the first difference set
df_forecast['MoneyForecast'] = df['Money'].iloc[-nobs-1] + df_forecast['Money'].
↪cumsum()
```

[14]:
```
# Add the most recent first difference from the training side of the original␣
↪dataset to the forecast cumulative sum
df_forecast['Spending1d'] = (df['Spending'].iloc[-nobs-1]-df['Spending'].
↪iloc[-nobs-2]) + df_forecast['Spending'].cumsum()
```

```python
# Now build the forecast values from the first difference set
df_forecast['SpendingForecast'] = df['Spending'].iloc[-nobs-1] +␣
↪df_forecast['Spending'].cumsum()
```

[15]:
```python
df_forecast
```

[15]:
```
                Money   Spending    Money1d  MoneyForecast  Spending1d  \
2015-01-01 -11.501568  36.789494  67.098432   11658.598432   47.389494
2015-02-01 -10.883687  -4.696517  56.214745   11647.714745   42.692977
2015-03-01   1.124754  -0.222204  57.339499   11648.839499   42.470773
2015-04-01  -1.413346  -0.379833  55.926154   11647.426154   42.090940
2015-05-01   0.889492   0.180924  56.815646   11648.315646   42.271865
2015-06-01  -0.263555  -0.048266  56.552091   11648.052091   42.223598
2015-07-01   0.429407   0.101016  56.981498   11648.481498   42.324614
2015-08-01   0.038821   0.019025  57.020319   11648.520319   42.343640
2015-09-01   0.263797   0.066679  57.284116   11648.784116   42.410319
2015-10-01   0.135170   0.039517  57.419286   11648.919286   42.449836
2015-11-01   0.208897   0.055102  57.628183   11649.128183   42.504939
2015-12-01   0.166674   0.046180  57.794858   11649.294858   42.551119

            SpendingForecast
2015-01-01      12098.789494
2015-02-01      12094.092977
2015-03-01      12093.870773
2015-04-01      12093.490940
2015-05-01      12093.671865
2015-06-01      12093.623598
2015-07-01      12093.724614
2015-08-01      12093.743640
2015-09-01      12093.810319
2015-10-01      12093.849836
2015-11-01      12093.904939
2015-12-01      12093.951119
```

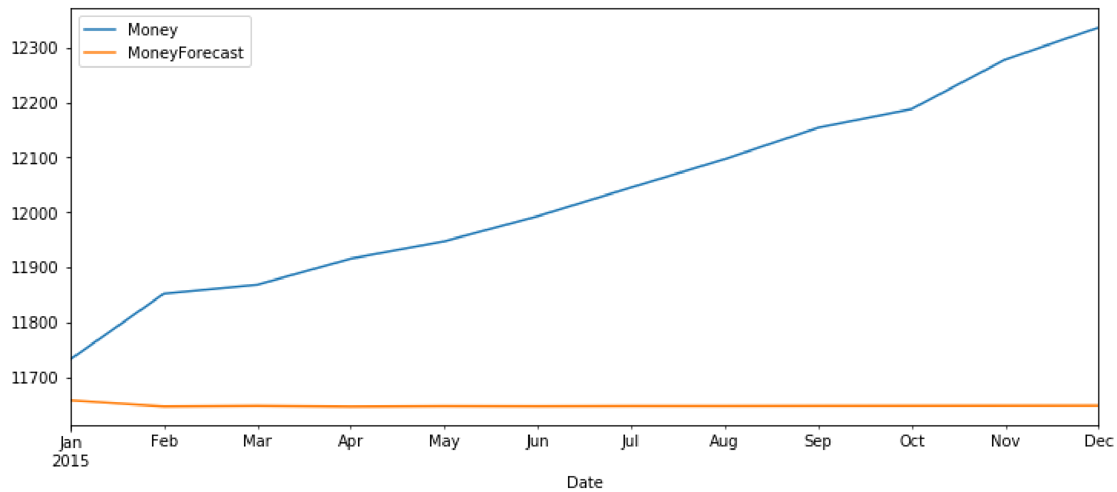
[16]:
```python
pd.concat([df.iloc[-12:
↪],df_forecast[['MoneyForecast','SpendingForecast']]],axis=1)
```

[16]:
```
              Money  Spending  MoneyForecast  SpendingForecast
Date
2015-01-01  11733.2   12046.0   11658.598432      12098.789494
2015-02-01  11852.4   12082.4   11647.714745      12094.092977
2015-03-01  11868.8   12158.3   11648.839499      12093.870773
2015-04-01  11916.1   12193.8   11647.426154      12093.490940
2015-05-01  11947.6   12268.1   11648.315646      12093.671865
2015-06-01  11993.1   12308.3   11648.052091      12093.623598
2015-07-01  12045.3   12355.4   11648.481498      12093.724614
2015-08-01  12096.8   12394.0   11648.520319      12093.743640
```

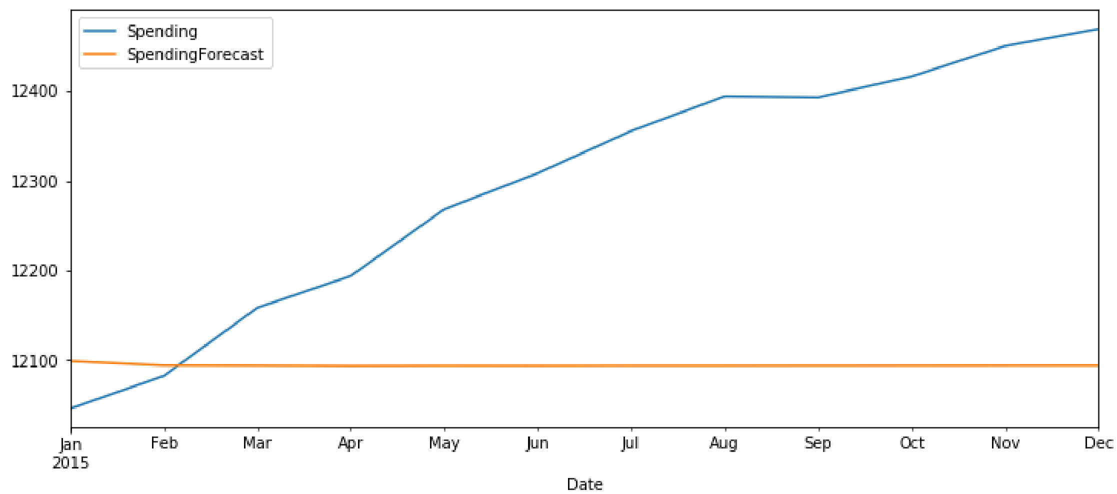
```
2015-09-01  12153.8   12392.8   11648.784116      12093.810319
2015-10-01  12187.7   12416.1   11648.919286      12093.849836
2015-11-01  12277.4   12450.1   11649.128183      12093.904939
2015-12-01  12335.9   12469.1   11649.294858      12093.951119
```

1.7 Plot the results

[17]:
```
df['Money'][-nobs:].plot(figsize=(12,5),legend=True).
↪autoscale(axis='x',tight=True)
df_forecast['MoneyForecast'].plot(legend=True);
```



[18]:
```
df['Spending'][-nobs:].plot(figsize=(12,5),legend=True).
↪autoscale(axis='x',tight=True)
df_forecast['SpendingForecast'].plot(legend=True);
```



1.7.1 Evaluate the model

$RMSE = \sqrt{\frac{1}{L}\sum_{l=1}^{L}(y_{T+l} - \hat{y}_{T+l})^2}$ where T is the last observation period and l is the lag.

[19]:
```
RMSE1 = rmse(df['Money'][-nobs:], df_forecast['MoneyForecast'])
print(f'Money VAR(5) RMSE: {RMSE1:.3f}')
```

```
Money VAR(5) RMSE: 422.942
```

[20]:
```
RMSE2 = rmse(df['Spending'][-nobs:], df_forecast['SpendingForecast'])
print(f'Spending VAR(5) RMSE: {RMSE2:.3f}')
```

```
Spending VAR(5) RMSE: 243.777
```

Clearly these results are less accurate than our earlier VAR(5) model. Still, this tells us something! ## Let's compare these results to individual ARMA(1,2) models

[21]:
```
from statsmodels.tsa.arima_model import ARMA,ARMAResults
```

1.7.2 Money

[22]:
```
model = ARMA(train['Money'],order=(1,2))
results = model.fit()
results.summary()
```

[22]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                              ARMA Model Results
==============================================================================
Dep. Variable:                  Money   No. Observations:                  238
Model:                     ARMA(1, 2)   Log Likelihood               -1117.710
Method:                       css-mle   S.D. of innovations             26.214
Date:                Wed, 03 Apr 2019   AIC                           2245.421
Time:                        08:25:54   BIC                           2262.782
Sample:                    03-01-1995   HQIC                          2252.418
                         - 12-01-2014
===============================================================================
                  coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------
const           0.1814      0.029      6.302      0.000       0.125       0.238
ar.L1.Money    -0.3569      0.293     -1.218      0.225      -0.931       0.218
ma.L1.Money    -0.4087      0.260     -1.573      0.117      -0.918       0.101
```

```
ma.L2.Money    -0.5912      0.259     -2.278      0.024      -1.100      -0.083
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -2.8022           +0.0000j            2.8022            0.5000
MA.1            1.0000           +0.0000j            1.0000            0.0000
MA.2           -1.6913           +0.0000j            1.6913            0.5000
-----------------------------------------------------------------------------
"""
```

[23]:

```
start=len(train)
end=len(train)+len(test)-1
z1 = results.predict(start=start, end=end).rename('Money')
z1 = pd.DataFrame(z1)
```

[24]:

```
z1
```

[24]:

```
                Money
2015-01-01 -14.498910
2015-02-01 -10.947218
2015-03-01   4.152839
2015-04-01  -1.235882
2015-05-01   0.687178
2015-06-01   0.000900
2015-07-01   0.245811
2015-08-01   0.158410
2015-09-01   0.189600
2015-10-01   0.178470
2015-11-01   0.182442
2015-12-01   0.181024
```

1.7.3 Invert the Transformation, Evaluate the Forecast

[25]:

```
# Add the most recent first difference from the training set to the forecast␣
↪cumulative sum
z1['Money1d'] = (df['Money'].iloc[-nobs-1]-df['Money'].iloc[-nobs-2]) +␣
↪z1['Money'].cumsum()

# Now build the forecast values from the first difference set
z1['MoneyForecast'] = df['Money'].iloc[-nobs-1] + z1['Money1d'].cumsum()
```

[26]:

```
z1
```

```
[26]:                  Money    Money1d  MoneyForecast
2015-01-01 -14.498910  64.101090   11734.201090
2015-02-01 -10.947218  53.153872   11787.354962
2015-03-01   4.152839  57.306711   11844.661673
2015-04-01  -1.235882  56.070829   11900.732502
2015-05-01   0.687178  56.758007   11957.490509
2015-06-01   0.000900  56.758908   12014.249417
2015-07-01   0.245811  57.004718   12071.254135
2015-08-01   0.158410  57.163128   12128.417263
2015-09-01   0.189600  57.352729   12185.769991
2015-10-01   0.178470  57.531198   12243.301190
2015-11-01   0.182442  57.713640   12301.014830
2015-12-01   0.181024  57.894664   12358.909494
```

```
[27]: RMSE3 = rmse(df['Money'][-nobs:], z1['MoneyForecast'])

print(f'Money VARMA(1,2) RMSE: {RMSE1:.3f}')
print(f'Money  ARMA(1,2) RMSE: {RMSE3:.3f}')
```

```
Money VARMA(1,2) RMSE: 422.942
Money  ARMA(1,2) RMSE: 32.236
```

1.8 Personal Spending

```
[28]: model = ARMA(train['Spending'],order=(1,2))
results = model.fit()
results.summary()
```

```
[28]: <class 'statsmodels.iolib.summary.Summary'>
"""
                              ARMA Model Results
==============================================================================
Dep. Variable:               Spending   No. Observations:                  238
Model:                     ARMA(1, 2)   Log Likelihood               -1182.411
Method:                       css-mle   S.D. of innovations             34.661
Date:                Wed, 03 Apr 2019   AIC                           2374.823
Time:                        08:26:04   BIC                           2392.184
Sample:                    03-01-1995   HQIC                          2381.820
                         - 12-01-2014
================================================================================
==
                     coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
--
const              0.0856      0.245      0.350      0.727      -0.394
```

```
0.565
ar.L1.Spending    -0.3403      0.511     -0.666      0.506      -1.342
0.661
ma.L1.Spending    -0.6451      0.521     -1.237      0.217      -1.667
0.377
ma.L2.Spending    -0.2139      0.485     -0.441      0.660      -1.165
0.737
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -2.9388           +0.0000j            2.9388            0.5000
MA.1            1.1281           +0.0000j            1.1281            0.0000
MA.2           -4.1438           +0.0000j            4.1438            0.5000
-----------------------------------------------------------------------------
"""
```

[29]:

```
start=len(train)
end=len(train)+len(test)-1
z2 = results.predict(start=start, end=end).rename('Spending')
z2 = pd.DataFrame(z2)
z2
```

[29]:

	Spending
2015-01-01	33.555831
2015-02-01	-3.338262
2015-03-01	1.250702
2015-04-01	-0.310832
2015-05-01	0.220527
2015-06-01	0.039716
2015-07-01	0.101243
2015-08-01	0.080306
2015-09-01	0.087431
2015-10-01	0.085006
2015-11-01	0.085831
2015-12-01	0.085551

1.8.1 Invert the Transformation, Evaluate the Forecast

[30]:

```
# Add the most recent first difference from the training set to the forecast␣
↪cumulative sum
z2['Spending1d'] = (df['Spending'].iloc[-nobs-1]-df['Spending'].iloc[-nobs-2])␣
↪+ z2['Spending'].cumsum()

# Now build the forecast values from the first difference set
```

```python
z2['SpendingForecast'] = df['Spending'].iloc[-nobs-1] + z2['Spending1d'].
↪cumsum()
```

[31]:
```python
z2
```

[31]:

	Spending	Spending1d	SpendingForecast
2015-01-01	33.555831	44.155831	12106.155831
2015-02-01	-3.338262	40.817569	12146.973400
2015-03-01	1.250702	42.068270	12189.041670
2015-04-01	-0.310832	41.757439	12230.799108
2015-05-01	0.220527	41.977966	12272.777074
2015-06-01	0.039716	42.017682	12314.794756
2015-07-01	0.101243	42.118925	12356.913681
2015-08-01	0.080306	42.199231	12399.112912
2015-09-01	0.087431	42.286662	12441.399574
2015-10-01	0.085006	42.371668	12483.771242
2015-11-01	0.085831	42.457500	12526.228742
2015-12-01	0.085551	42.543050	12568.771792

[32]:
```python
RMSE4 = rmse(df['Spending'][-nobs:], z2['SpendingForecast'])

print(f'Spending VARMA(1,2) RMSE: {RMSE2:.3f}')
print(f'Spending  ARMA(1,2) RMSE: {RMSE4:.3f}')
```

```
Spending VARMA(1,2) RMSE: 243.777
Spending  ARMA(1,2) RMSE: 52.334
```

CONCLUSION: It looks like the VARMA(1,2) model did a relatively poor job compared to simpler alternatives. This tells us that there is little or no interdepence between Money Stock and Personal Consumption Expenditures, at least for the timespan we investigated. This is helpful! By fitting a model and getting poor results we know more about the data than we did before.