06-SARIMA

October 19, 2022

Copyright Pierian Data

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

1 SARIMA(p,d,q)(P,D,Q)m

2 Seasonal Autoregressive Integrated Moving Averages

We have finally reached one of the most fascinating aspects of time series analysis: seasonality.

Where ARIMA accepts the parameters (p, d, q), SARIMA accepts an additional set of parameters $(P, D, Q)m$ that specifically describe the seasonal components of the model. Here P, D and Q represent the seasonal regression, differencing and moving average coefficients, and m represents the number of data points (rows) in each seasonal cycle.

NOTE: The statsmodels implementation of SARIMA is called SARIMAX. The "X" added to the name means that the function also supports exogenous regressor variables. We'll cover these in the next section.

Related Functions:

sarimax.SARIMAX(endog[, exog, order, …]) sarimax.SARIMAXResults(model, params, …[, …]) Class to hold results from fitting a SARIMAX model.

For Further Reading:

Statsmodels Tutorial: Time Series Analysis by State Space Methods

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.statespace.sarimax import SARIMAX
```

```python
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf # for determining␣
↪(p,q) orders
from statsmodels.tsa.seasonal import seasonal_decompose      # for ETS Plots
from pmdarima import auto_arima                              # for determining␣
↪ARIMA orders

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

# Load dataset
df = pd.read_csv('../Data/co2_mm_mlo.csv')
```

2.1.1 Inspect the data, create a DatetimeIndex

[2]:
```python
df.head()
```

[2]:

	year	month	decimal_date	average	interpolated
0	1958	3	1958.208	315.71	315.71
1	1958	4	1958.292	317.45	317.45
2	1958	5	1958.375	317.50	317.50
3	1958	6	1958.458	NaN	317.10
4	1958	7	1958.542	315.86	315.86

We need to combine two integer columns (year and month) into a DatetimeIndex. We can do this by passing a dictionary into pandas.to_datetime() with year, month and day values. For more information visit https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_datetime.html

[3]:
```python
# Add a "date" datetime column
df['date']=pd.to_datetime(dict(year=df['year'], month=df['month'], day=1))
```

[4]:
```python
# Set "date" to be the index
df.set_index('date',inplace=True)
df.index.freq = 'MS'
df.head()
```

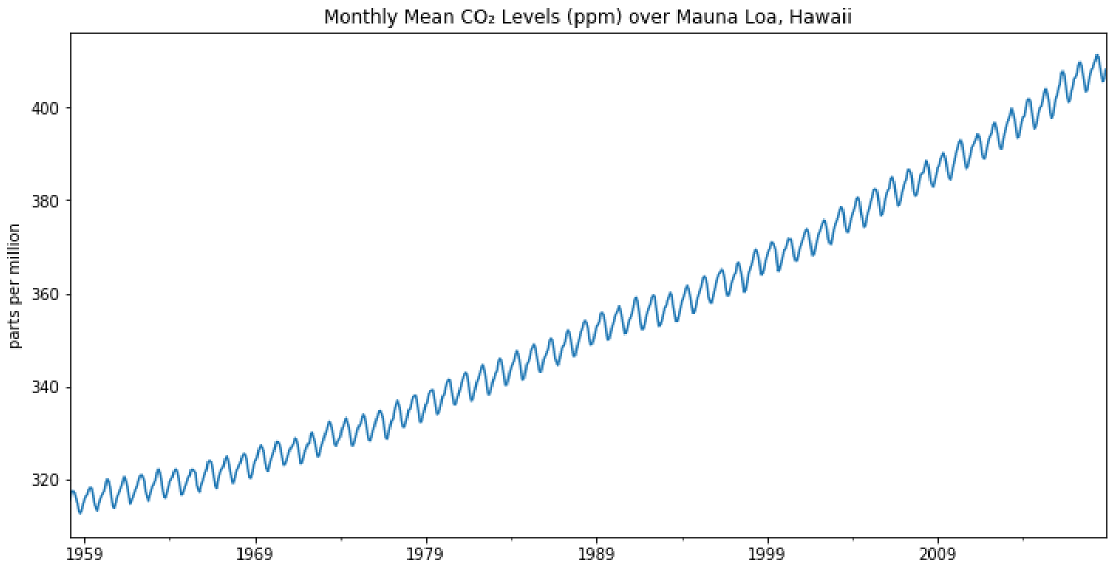
[4]:

	year	month	decimal_date	average	interpolated
date					
1958-03-01	1958	3	1958.208	315.71	315.71
1958-04-01	1958	4	1958.292	317.45	317.45
1958-05-01	1958	5	1958.375	317.50	317.50
1958-06-01	1958	6	1958.458	NaN	317.10
1958-07-01	1958	7	1958.542	315.86	315.86

2.1.2 Plot the source data

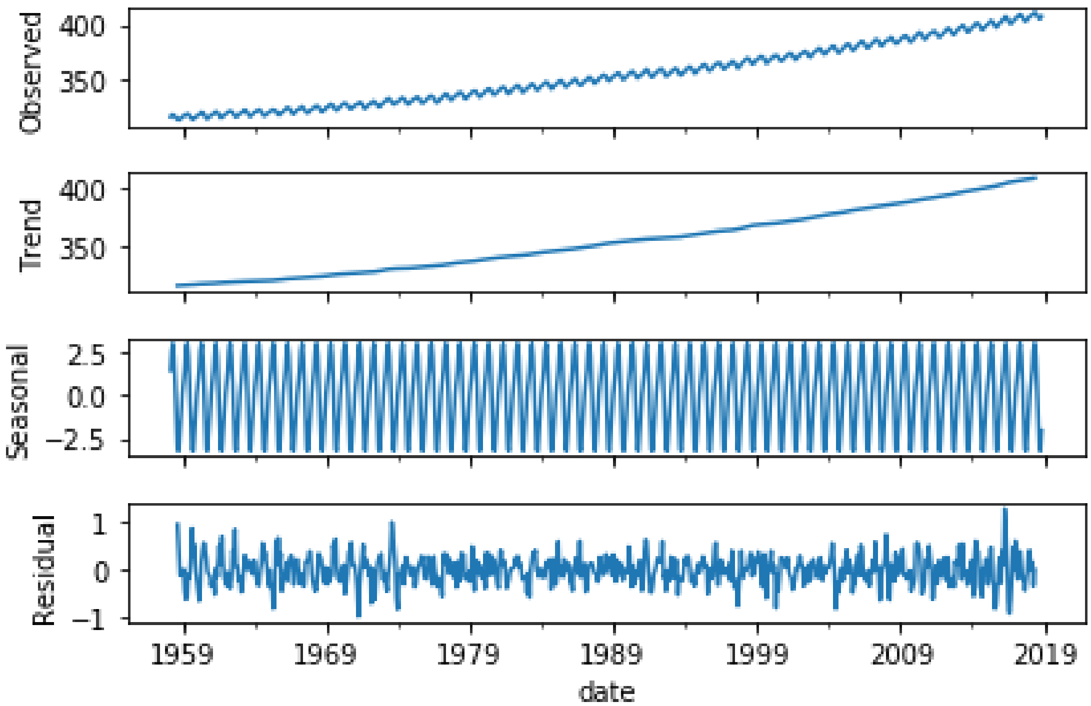
[5]:
```
title = 'Monthly Mean CO  Levels (ppm) over Mauna Loa, Hawaii'
ylabel='parts per million'
xlabel='' # we don't really need a label here

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



2.1.3 Run an ETS Decomposition

[6]:
```
result = seasonal_decompose(df['interpolated'], model='add')
result.plot();
```



Although small in scale compared to the overall values, there is a definite annual seasonality.

2.1.4 Run pmdarima.auto_arima to obtain recommended orders

This may take awhile as there are a lot more combinations to evaluate.

[7]:
```
# For SARIMA Orders we set seasonal=True and pass in an m value
auto_arima(df['interpolated'],seasonal=True,m=12).summary()
```

[7]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                 Statespace Model Results
================================================================================
==========
Dep. Variable:                                  y   No. Observations:
729
Model:             SARIMAX(0, 1, 3)x(1, 0, 1, 12)   Log Likelihood
-203.092
Date:                            Wed, 03 Apr 2019   AIC
420.183
Time:                                    17:49:04   BIC
452.315
Sample:                                         0   HQIC
```

```
432.582
                                             - 729
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0009      0.001      1.449      0.147      -0.000       0.002
ma.L1         -0.3577      0.037     -9.728      0.000      -0.430      -0.286
ma.L2         -0.0310      0.038     -0.813      0.416      -0.106       0.044
ma.L3         -0.0865      0.037     -2.349      0.019      -0.159      -0.014
ar.S.L12       0.9994      0.000   2999.488      0.000       0.999       1.000
ma.S.L12      -0.8695      0.021    -42.160      0.000      -0.910      -0.829
sigma2         0.0958      0.005     20.352      0.000       0.087       0.105
================================================================================
===
Ljung-Box (Q):                       45.20   Jarque-Bera (JB):
4.09
Prob(Q):                              0.26   Prob(JB):
0.13
Heteroskedasticity (H):               1.11   Skew:
0.01
Prob(H) (two-sided):                  0.40   Kurtosis:
3.37
================================================================================
===

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

Excellent! This provides an ARIMA Order of (0,1,3) combined with a seasonal order of (1,0,1,12) Now let's train & test the SARIMA(0,1,3)(1,0,1,12) model, evaluate it, then produce a forecast of future values. ### Split the data into train/test sets

[8]:
```python
len(df)
```

[8]: 729

[9]:
```python
# Set one year for testing
train = df.iloc[:717]
test = df.iloc[717:]
```

2.1.5 Fit a SARIMA(0,1,3)(1,0,1,12) Model

[10]:
```
model = SARIMAX(train['interpolated'],order=(0,1,3),seasonal_order=(1,0,1,12))
results = model.fit()
results.summary()
```

[10]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                  Statespace Model Results
================================================================================
==========
Dep. Variable:                        interpolated   No. Observations:
717
Model:               SARIMAX(0, 1, 3)x(1, 0, 1, 12)   Log Likelihood
-201.201
Date:                             Wed, 03 Apr 2019   AIC
414.402
Time:                                     17:49:12   BIC
441.845
Sample:                                 03-01-1958   HQIC
424.999
                                      - 11-01-2017
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.3543      0.035    -10.200      0.000      -0.422      -0.286
ma.L2         -0.0244      0.038     -0.648      0.517      -0.098       0.050
ma.L3         -0.0866      0.032     -2.686      0.007      -0.150      -0.023
ar.S.L12       0.9997      0.000   3239.477      0.000       0.999       1.000
ma.S.L12      -0.8680      0.022    -38.921      0.000      -0.912      -0.824
sigma2         0.0949      0.005     20.315      0.000       0.086       0.104
================================================================================
===
Ljung-Box (Q):                       43.96   Jarque-Bera (JB):
4.45
Prob(Q):                              0.31   Prob(JB):
0.11
Heteroskedasticity (H):               1.15   Skew:
0.02
Prob(H) (two-sided):                  0.27   Kurtosis:
3.38
================================================================================
===

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

```
step).
"""
```

[11]:
```
# Obtain predicted values
start=len(train)
end=len(train)+len(test)-1
predictions = results.predict(start=start, end=end, dynamic=False,␣
↪typ='levels').rename('SARIMA(0,1,3)(1,0,1,12) 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.

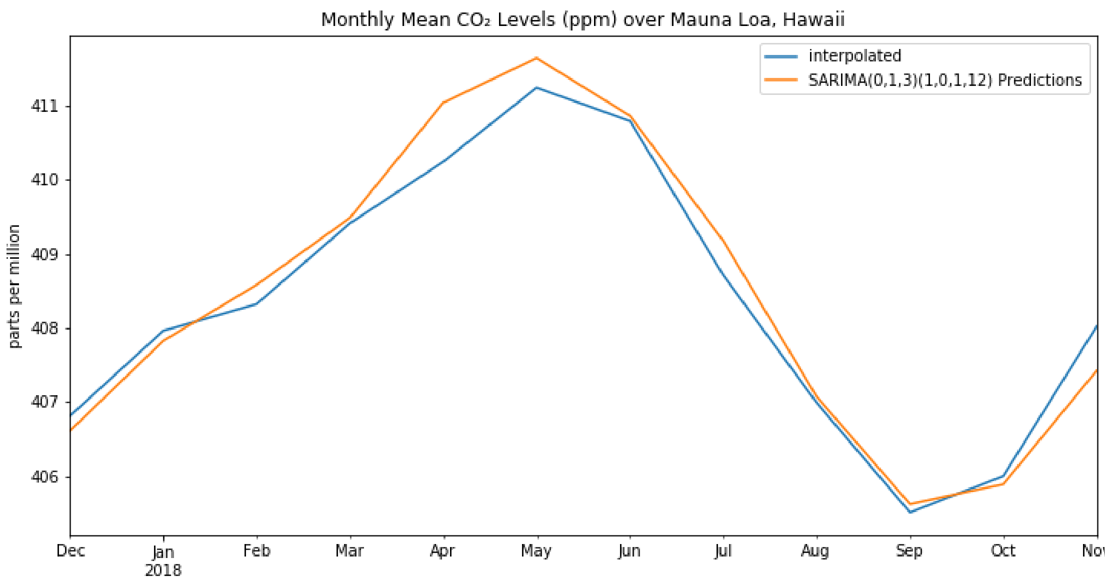
For more information on these arguments visit https://www.statsmodels.org/stable/generated/statsmodels.tsa.arin

[12]:
```
# Compare predictions to expected values
for i in range(len(predictions)):
    print(f"predicted={predictions[i]:<11.10},␣
↪expected={test['interpolated'][i]}")
```

```
predicted=406.6094505, expected=406.81
predicted=407.8239343, expected=407.96
predicted=408.578265 , expected=408.32
predicted=409.4831633, expected=409.41
predicted=411.036772 , expected=410.24
predicted=411.6394325, expected=411.24
predicted=410.8612658, expected=410.79
predicted=409.1726191, expected=408.71
predicted=407.0722543, expected=406.99
predicted=405.6214644, expected=405.51
predicted=405.8902221, expected=406.0
predicted=407.4224706, expected=408.02
```

[13]:
```
# Plot predictions against known values
title = 'Monthly Mean CO  Levels (ppm) over Mauna Loa, Hawaii'
ylabel='parts per million'
xlabel=''

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



2.1.6 Evaluate the Model

[14]:
```
from sklearn.metrics import mean_squared_error

error = mean_squared_error(test['interpolated'], predictions)
print(f'SARIMA(0,1,3)(1,0,1,12) MSE Error: {error:11.10}')
```

```
SARIMA(0,1,3)(1,0,1,12) MSE Error: 0.1277131368
```

[15]:
```
from statsmodels.tools.eval_measures import rmse

error = rmse(test['interpolated'], predictions)
print(f'SARIMA(0,1,3)(1,0,1,12) RMSE Error: {error:11.10}')
```

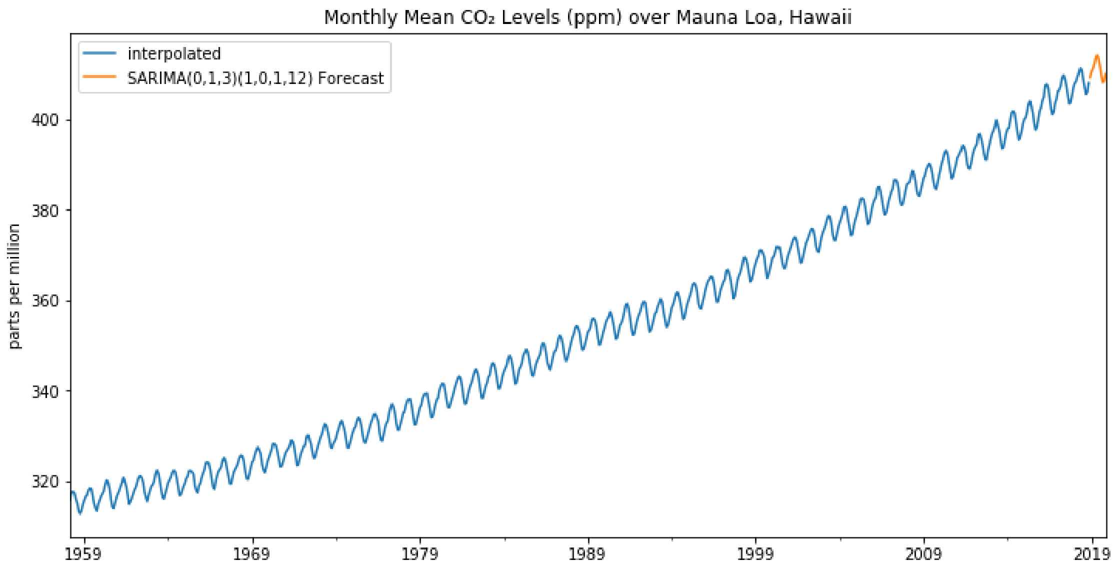
```
SARIMA(0,1,3)(1,0,1,12) RMSE Error: 0.357369748
```

These are outstanding results! ### Retrain the model on the full data, and forecast the future

[16]:
```
model = SARIMAX(df['interpolated'],order=(0,1,3),seasonal_order=(1,0,1,12))
results = model.fit()
fcast = results.predict(len(df),len(df)+11,typ='levels').
 ↪rename('SARIMA(0,1,3)(1,0,1,12) Forecast')
```

[17]:
```
# Plot predictions against known values
title = 'Monthly Mean CO  Levels (ppm) over Mauna Loa, Hawaii'
ylabel='parts per million'
xlabel=''
```

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



2.2 Great job!