07-Exogenous-Variables-SARIMAX

October 19, 2022

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

1 SARIMAX

1.1 Seasonal AutoRegressive Integrated Moving Average with eXogenous regressors

So far the models we've looked at consider past values of a dataset and past errors to determine future trends, seasonality and forecasted values. We look now to models that encompass these non-seasonal (p,d,q) and seasonal (P,D,Q,m) factors, but introduce the idea that external factors (environmental, economic, etc.) can also influence a time series, and be used in forecasting.

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 Statsmodels Example: SARIMAX

1.2 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

from statsmodels.graphics.tsaplots import plot_acf,plot_pacf # for determining␣
 ↪(p,q) orders
```

```python
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/RestaurantVisitors.
 ↪csv',index_col='date',parse_dates=True)
df.index.freq = 'D'
```

1.2.1 Inspect the data

For this section we've built a Restaurant Visitors dataset that was inspired by a recent Kaggle competition. The data considers daily visitors to four restaurants located in the United States, subject to American holidays. For the exogenous variable we'll see how holidays affect patronage. The dataset contains 478 days of restaurant data, plus an additional 39 days of holiday data for forecasting purposes.

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

[2]:
```
              weekday  holiday    holiday_name  rest1  rest2  rest3  rest4  \
date
2016-01-01     Friday        1  New Year's Day   65.0   25.0   67.0  139.0
2016-01-02   Saturday        0              na   24.0   39.0   43.0   85.0
2016-01-03     Sunday        0              na   24.0   31.0   66.0   81.0
2016-01-04     Monday        0              na   23.0   18.0   32.0   32.0
2016-01-05    Tuesday        0              na    2.0   15.0   38.0   43.0

            total
date
2016-01-01  296.0
2016-01-02  191.0
2016-01-03  202.0
2016-01-04  105.0
2016-01-05   98.0
```

Notice that even though the restaurant visitor columns contain integer data, they appear as floats. This is because the bottom of the dataframe has 39 rows of NaN data to accommodate the extra holiday data we'll use for forecasting, and pandas won't allow NaN's as integers. We could leave it like this, but since we have to drop NaN values anyway, let's also convert the columns to dtype int64.

[3]:
```python
df.tail()
```

[3]:

```
              weekday  holiday  holiday_name  rest1  rest2  rest3  rest4  \
date
2017-05-27   Saturday        0            na    NaN    NaN    NaN    NaN
2017-05-28     Sunday        0            na    NaN    NaN    NaN    NaN
2017-05-29     Monday        1  Memorial Day    NaN    NaN    NaN    NaN
2017-05-30    Tuesday        0            na    NaN    NaN    NaN    NaN
2017-05-31  Wednesday        0            na    NaN    NaN    NaN    NaN

            total
date
2017-05-27    NaN
2017-05-28    NaN
2017-05-29    NaN
2017-05-30    NaN
2017-05-31    NaN
```

[4]:

```
df1 = df.dropna()
df1.tail()
```

[4]:

```
              weekday  holiday holiday_name  rest1  rest2  rest3  rest4  total
date
2017-04-18    Tuesday        0           na   30.0   30.0   13.0   18.0   91.0
2017-04-19  Wednesday        0           na   20.0   11.0   30.0   18.0   79.0
2017-04-20   Thursday        0           na   22.0    3.0   19.0   46.0   90.0
2017-04-21     Friday        0           na   38.0   53.0   36.0   38.0  165.0
2017-04-22   Saturday        0           na   97.0   20.0   50.0   59.0  226.0
```

[5]:

```
# Change the dtype of selected columns
cols = ['rest1','rest2','rest3','rest4','total']
for col in cols:
    df1[col] = df1[col].astype(int)
df1.head()
```

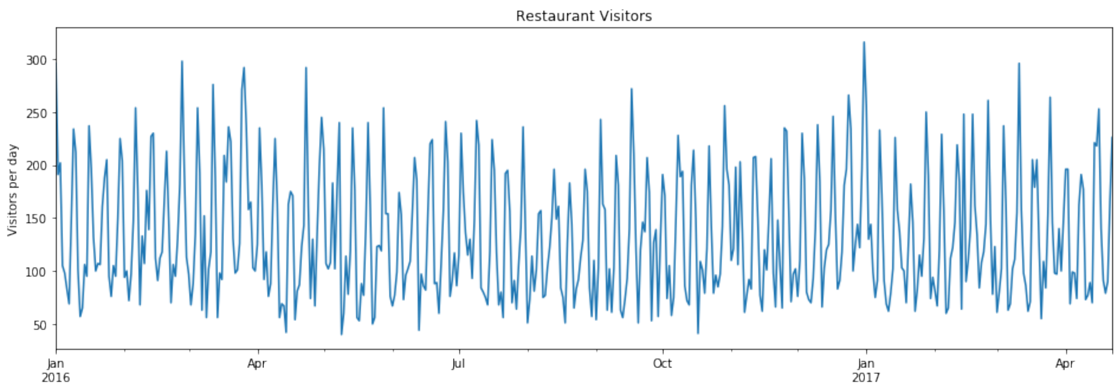
[5]:

```
             weekday  holiday    holiday_name  rest1  rest2  rest3  rest4  \
date
2016-01-01    Friday        1  New Year's Day     65     25     67    139
2016-01-02  Saturday        0              na     24     39     43     85
2016-01-03    Sunday        0              na     24     31     66     81
2016-01-04    Monday        0              na     23     18     32     32
2016-01-05   Tuesday        0              na      2     15     38     43

            total
date
2016-01-01    296
2016-01-02    191
2016-01-03    202
2016-01-04    105
```

2016-01-05   98

1.2.2 Plot the source data

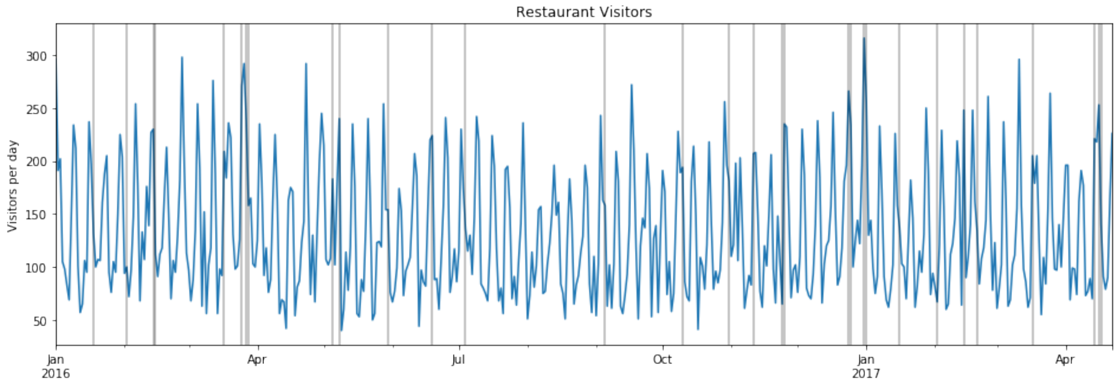
[6]:
```
title='Restaurant Visitors'
ylabel='Visitors per day'
xlabel='' # we don't really need a label here

ax = df1['total'].plot(figsize=(16,5),title=title)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel);
```



1.3 Look at holidays

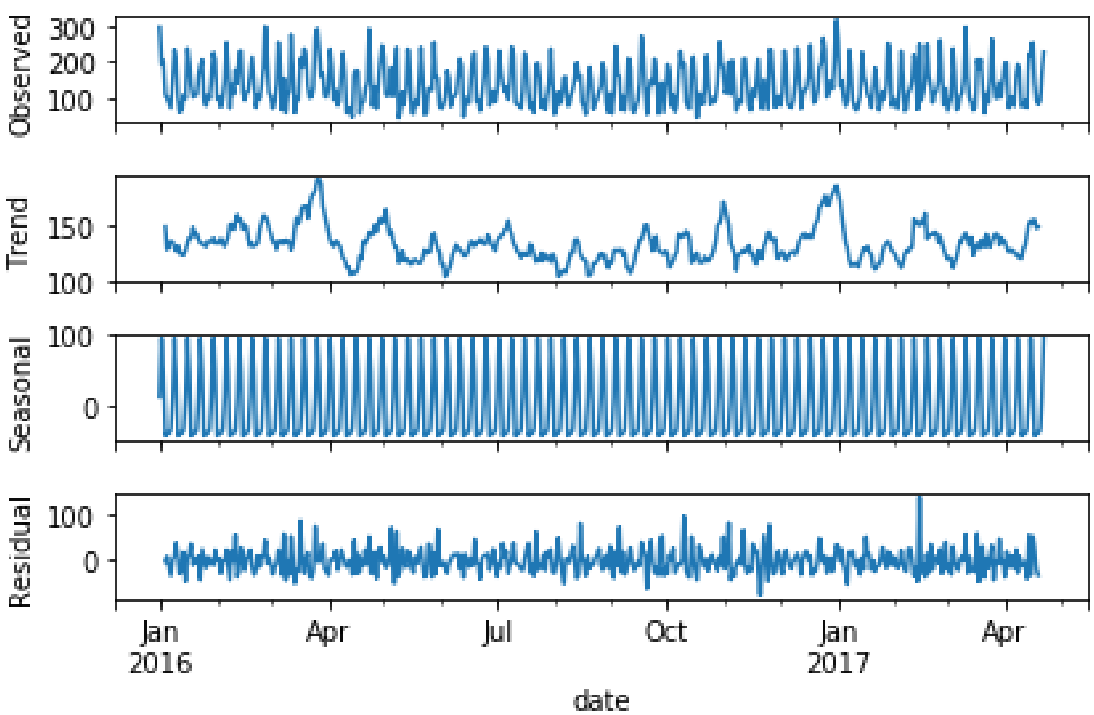
Rather than prepare a separate plot, we can use matplotlib to shade holidays behind our restaurant data.

[7]:
```
title='Restaurant Visitors'
ylabel='Visitors per day'
xlabel='' # we don't really need a label here

ax = df1['total'].plot(figsize=(16,5),title=title)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
for x in df1.query('holiday==1').index:       # for days where holiday == 1
    ax.axvline(x=x, color='k', alpha = 0.3);  # add a semi-transparent grey line
```



1.3.1 Run an ETS Decomposition

[8]:
```python
result = seasonal_decompose(df1['total'])
result.plot();
```



1.4 Test for stationarity

[9]:
```
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")
```

[10]:
```
adf_test(df1['total'])
```

```
Augmented Dickey-Fuller Test:
ADF test statistic       -5.592497
p-value                   0.000001
# lags used              18.000000
# observations          459.000000
critical value (1%)      -3.444677
critical value (5%)      -2.867857
critical value (10%)     -2.570135
Strong evidence against the null hypothesis
Reject the null hypothesis
Data has no unit root and is stationary
```

1.4.1 Run pmdarima.auto_arima to obtain recommended orders

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

[11]:
```
# For SARIMA Orders we set seasonal=True and pass in an m value
auto_arima(df1['total'],seasonal=True,m=7).summary()
```

[11]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                 Statespace Model Results
================================================================================
=========
Dep. Variable:                                y   No. Observations:
478
Model:             SARIMAX(1, 0, 0)x(2, 0, 0, 7)   Log Likelihood
-2417.721
Date:                          Wed, 03 Apr 2019   AIC
4845.442
Time:                                  17:52:06   BIC
4866.290
Sample:                                       0   HQIC
4853.638
                                          - 478
Covariance Type:                            opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     20.5662      4.363      4.714      0.000      12.015      29.118
ar.L1          0.1897      0.045      4.221      0.000       0.102       0.278
ar.S.L7        0.4258      0.037     11.606      0.000       0.354       0.498
ar.S.L14       0.3873      0.036     10.734      0.000       0.317       0.458
sigma2      1427.3967     86.679     16.468      0.000    1257.510    1597.283
================================================================================
===
Ljung-Box (Q):                       77.20   Jarque-Bera (JB):
27.47
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):               0.81   Skew:
0.47
Prob(H) (two-sided):                  0.18   Kurtosis:
3.71
================================================================================
===

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

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

Excellent! This provides an ARIMA Order of (1,0,0) and a seasonal order of (2,0,0,7) Now let's train & test the SARIMA model, evaluate it, then compare the result to a model that uses an exogenous variable. ### Split the data into train/test sets We'll assign 42 days (6 weeks) to the test set so that it includes several holidays.

[12]:
```
len(df1)
```

[12]:
```
478
```

[13]:
```
# Set four weeks for testing
train = df1.iloc[:436]
test = df1.iloc[436:]
```

1.4.2 Fit a SARIMA(1,0,0)(2,0,0,7) Model

NOTE: To avoid a ValueError: non-invertible starting MA parameters found we're going to set enforce_invertibility to False.

[14]:
```
model =␣
 ↪SARIMAX(train['total'],order=(1,0,0),seasonal_order=(2,0,0,7),enforce_invertibility=False)
results = model.fit()
results.summary()
```

[14]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                 Statespace Model Results
================================================================================
=========
Dep. Variable:                              total   No. Observations:
436
Model:             SARIMAX(1, 0, 0)x(2, 0, 0, 7)   Log Likelihood
-2224.701
Date:                            Wed, 03 Apr 2019   AIC
4457.403
Time:                                    17:52:35   BIC
4473.713
Sample:                                01-01-2016   HQIC
4463.840
                                     - 03-11-2017
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
ar.L1          0.2212      0.047      4.711      0.000       0.129       0.313
ar.S.L7        0.5063      0.036     14.187      0.000       0.436       0.576
ar.S.L14       0.4574      0.037     12.379      0.000       0.385       0.530
sigma2      1520.2899     82.277     18.478      0.000    1359.029    1681.550
================================================================================
===
Ljung-Box (Q):                       83.96   Jarque-Bera (JB):
29.23
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):               0.86   Skew:
0.34
Prob(H) (two-sided):                  0.37   Kurtosis:
4.07
================================================================================
===

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

[15]:

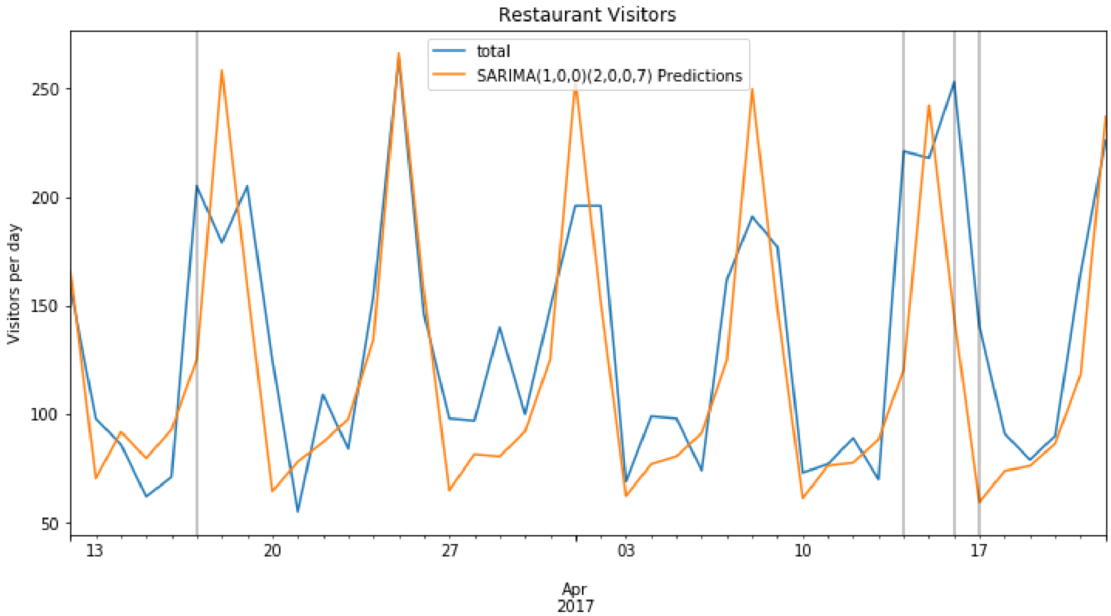
```
# Obtain predicted values
start=len(train)
end=len(train)+len(test)-1
predictions = results.predict(start=start, end=end, dynamic=False).
 ↪rename('SARIMA(1,0,0)(2,0,0,7) Predictions')
```

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

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

[16]:

```
# Plot predictions against known values
title='Restaurant Visitors'
ylabel='Visitors per day'
xlabel=''

ax = test['total'].plot(legend=True,figsize=(12,6),title=title)
predictions.plot(legend=True)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
for x in test.query('holiday==1').index:
    ax.axvline(x=x, color='k', alpha = 0.3);
```



1.4.3 Evaluate the Model

[17]:
```
from statsmodels.tools.eval_measures import mse,rmse

error1 = mse(test['total'], predictions)
error2 = rmse(test['total'], predictions)

print(f'SARIMA(1,0,0)(2,0,0,7) MSE Error: {error1:11.10}')
print(f'SARIMA(1,0,0)(2,0,0,7) RMSE Error: {error2:11.10}')
```

```
SARIMA(1,0,0)(2,0,0,7) MSE Error: 1702.647958
SARIMA(1,0,0)(2,0,0,7) RMSE Error: 41.26315497
```

1.5 Now add the exog variable

[18]:
```
model =␣
 ↪SARIMAX(train['total'],exog=train['holiday'],order=(1,0,0),seasonal_order=(2,0,0,7),enforce
results = model.fit()
results.summary()
```

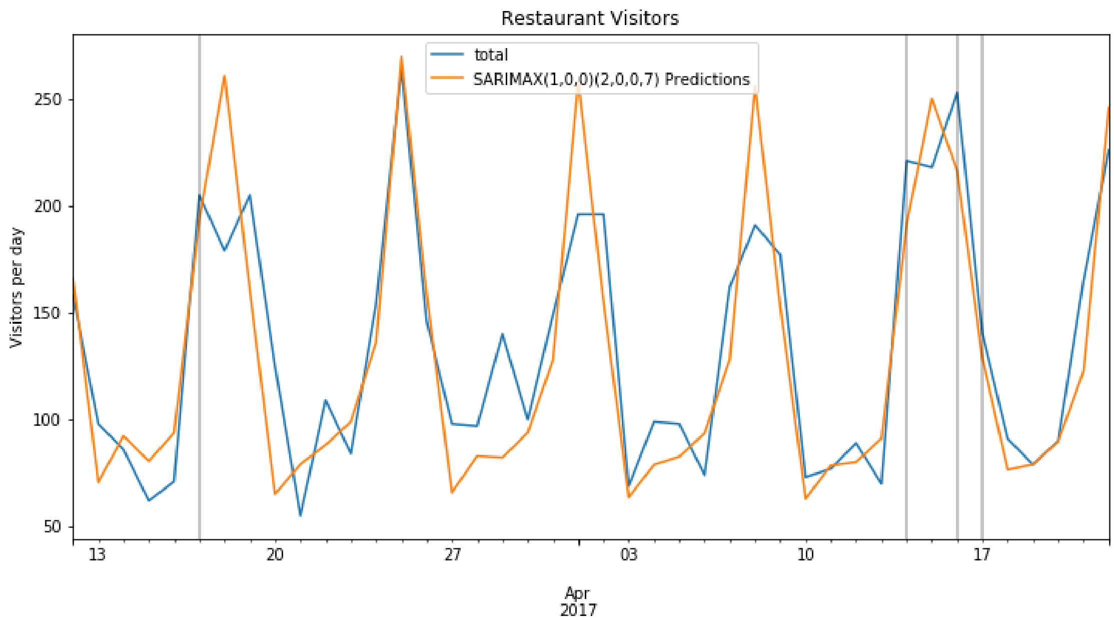
[18]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                 Statespace Model Results
================================================================================
```

```
=========
Dep. Variable:                            total   No. Observations:
436
Model:             SARIMAX(1, 0, 0)x(2, 0, 0, 7)   Log Likelihood
-2158.482
Date:                          Wed, 03 Apr 2019   AIC
4326.963
Time:                                  17:53:20   BIC
4347.352
Sample:                              01-01-2016   HQIC
4335.010
                                   - 03-11-2017
Covariance Type:                            opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
holiday       66.8897      4.241     15.774      0.000      58.578      75.201
ar.L1          0.2145      0.049      4.375      0.000       0.118       0.311
ar.S.L7        0.5147      0.042     12.312      0.000       0.433       0.597
ar.S.L14       0.4575      0.042     10.997      0.000       0.376       0.539
sigma2      1117.3977     73.302     15.244      0.000     973.729    1261.066
================================================================================
===
Ljung-Box (Q):                      100.96   Jarque-Bera (JB):
1.24
Prob(Q):                              0.00   Prob(JB):
0.54
Heteroskedasticity (H):               0.91   Skew:
0.11
Prob(H) (two-sided):                  0.58   Kurtosis:
3.14
================================================================================
===

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

[19]:

```python
# Obtain predicted values
start=len(train)
end=len(train)+len(test)-1
exog_forecast = test[['holiday']]  # requires two brackets to yield a shape of␣
↪(35,1)
predictions = results.predict(start=start, end=end, exog=exog_forecast).
↪rename('SARIMAX(1,0,0)(2,0,0,7) Predictions')
```

[20]:
```
# Plot predictions against known values
title='Restaurant Visitors'
ylabel='Visitors per day'
xlabel=''

ax = test['total'].plot(legend=True,figsize=(12,6),title=title)
predictions.plot(legend=True)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
for x in test.query('holiday==1').index:
    ax.axvline(x=x, color='k', alpha = 0.3);
```



We can see that the exogenous variable (holidays) had a positive impact on the forecast by raising predicted values at 3/17, 4/14, 4/16 and 4/17! Let's compare evaluations: ### Evaluate the Model

[22]:
```
# Print values from SARIMA above
print(f'SARIMA(1,0,0)(2,0,0,7) MSE Error: {error1:11.10}')
print(f'SARIMA(1,0,0)(2,0,0,7) RMSE Error: {error2:11.10}')
print()

error1x = mse(test['total'], predictions)
error2x = rmse(test['total'], predictions)

# Print new SARIMAX values
print(f'SARIMAX(1,0,0)(2,0,0,7) MSE Error: {error1x:11.10}')
print(f'SARIMA(1,0,0)(2,0,0,7) RMSE Error: {error2x:11.10}')
```

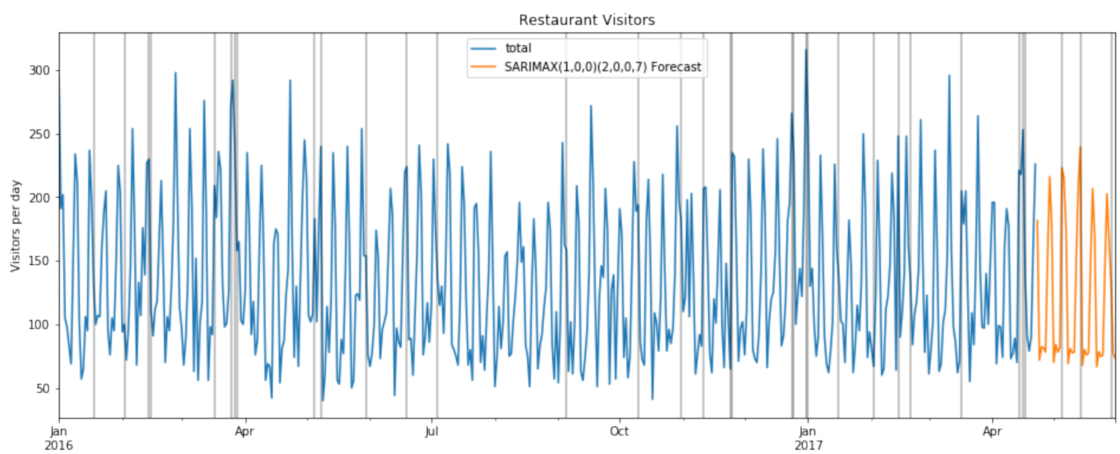
```
SARIMA(1,0,0)(2,0,0,7) MSE Error: 1702.647958
SARIMA(1,0,0)(2,0,0,7) RMSE Error: 41.26315497

SARIMAX(1,0,0)(2,0,0,7) MSE Error: 950.6656518
SARIMAX(1,0,0)(2,0,0,7) RMSE Error: 30.83286642
```

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

We're going to forecast 39 days into the future, and use the additional holiday data

[23]:
```
model =␣
↪SARIMAX(df1['total'],exog=df1['holiday'],order=(1,0,0),seasonal_order=(2,0,0,7),enforce_inv
results = model.fit()
exog_forecast = df[478:][['holiday']]
fcast = results.predict(len(df1),len(df1)+38,exog=exog_forecast).
↪rename('SARIMAX(1,0,0)(2,0,0,7) Forecast')
```

[24]:
```
# Plot the forecast alongside historical values
title='Restaurant Visitors'
ylabel='Visitors per day'
xlabel=''

ax = df1['total'].plot(legend=True,figsize=(16,6),title=title)
fcast.plot(legend=True)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
for x in df.query('holiday==1').index:
    ax.axvline(x=x, color='k', alpha = 0.3);
```



1.6 Great job!