How to forecast sales with Python using SARIMA model

A step-by-step guide of statistic and python to time series forecasting

Raphael Bubolz Larrosa

Jul 15 · 6 min read ★

Have you ever imagined predicting the future? Well, we are not there yet, but forecasting models (with a level of uncertainty) give us an excellent orientation to plan our business more assertively when we look to the future. In this post we will demonstrate an approach for forecasting time series of sales in the automotive industry using the SARIMA model.

Explaining the model

SARIMA is used for non-stationary series, that is, where the data do not fluctuate around the same mean, variance and co-variance. This model can identify trend and seasonality, which makes it so important. The SARIMA consists of other forecasting models:

AR: Auto regressive model (can be a simple, multiple or non-linear regression)

MA: Moving averages model. The moving average models can use weighting factors, where the observations are weighted by a trim factor (for the oldest data in the series) and with a higher weight for the most recent observations.

The composition of AR and MA together do not carry the **ARMA** model, but this model is used only for stationary series (mean, variance constant over time).

If the series has a tendency, it will be necessary to use the **ARIMA** model.
ARIMA is used for non-stationary series. In this model, a differentiation step I (d) is used to eliminate non-stationarity.
The integrated element "I" for differentiation allows the method to support time series with trend. But still this model does not identify seasonality.

Finally, we arrive at the **SARIMA** model, which has a seasonal correlation and can identify the seasonality of the time series. Now we can move for the codes!

Data Processing

We are going to use a data set of the automotive sales that can be downloaded from here.

```
import warnings
import itertools
import numpy as np
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
plt.style.use('fivethirtyeight')
import pandas as pd
import statsmodels.api as sm
import matplotlib
matplotlib.rcParams['axes.labelsize'] = 14
matplotlib.rcParams['xtick.labelsize'] = 12
matplotlib.rcParams['ytick.labelsize'] = 12
matplotlib.rcParams['text.color'] = 'G'
```

```
df = pd.read_excel("Retail2.xlsx")
```

This step is only to import the libraries, such as numpy, pandas, matplotlib and statsmodels, that is the library that contain the SARIMA model and other statistics features. This part of the code is used to setup the charts of matplotlib.

The original dataset and code are a little bit complex, but to make everything easier the file available to download has just a date and sale columns to avoid the data preprocessing.

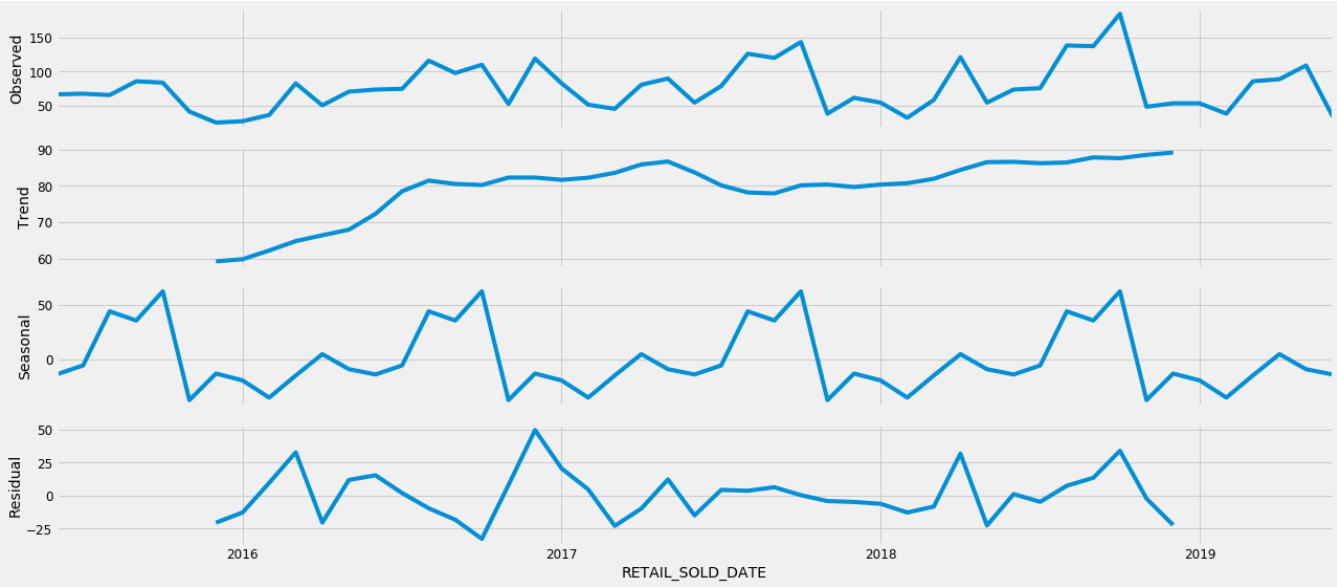
```
y = df.set_index(['Date'])
y.head(5)
```

The set_index command is setting the column date as an Index and the head is printing the first 5 rows of the data set.

```
y.plot(figsize=(19, 4))
plt.show()
```

Analyzing the chart, we can observe that the time-series has seasonality pattern. October has a peak of sales, at least for the last 3 years. There is an upward trend over the years as well.

```
from pylab import rcParams
rcParams['figure.figsize'] = 18, 8
decomposition = sm.tsa.seasonal_decompose(y, model='additive')
fig = decomposition.plot()
plt.show()
```



Using the “sm.tsa.seasonal_decompose” command from the pylab library we can decompose the time-series into three distinct components: trend, seasonality, and noise.

SARIMA to time series forecasting

Let’s use SARIMA. The models notation is `SARIMA(p, d, q).(P,D,Q)m`. These three parameters account for seasonality, trend, and noise in data

```
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in
list(itertools.product(p, d, q))]
print('Examples of parameter for SARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

Examples of parameter for SARIMA…
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)

```
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
```

```
            mod =
sm.tsa.statespace.SARIMAX(y,order=param,seasonal_order=param_seasona
l,enforce_stationarity=False,enforce_invertibility=False)
            results = mod.fit()
            print('ARIMA{}x{}12 - AIC:
{}'.format(param,param_seasonal,results.aic))
        except:
            continue
```

ARIMA(0, 0, 0)x(0, 0, 1, 12)12 — AIC:410.521537786262
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 — AIC:363.03322198787765
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 — AIC:348.13731052896617
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 — AIC:237.2069523573001
ARIMA(0, 0, 1)x(0, 1, 1, 12)12 — AIC:1005.9793011993983
ARIMA(0, 0, 1)x(1, 0, 0, 12)12 — AIC:364.8331172721984
ARIMA(0, 0, 1)x(1, 0, 1, 12)12 — AIC:341.5095766512678
ARIMA(0, 0, 1)x(1, 1, 0, 12)12 — AIC:239.13525566698246
ARIMA(0, 0, 1)x(1, 1, 1, 12)12 — AIC:223.43134436185036

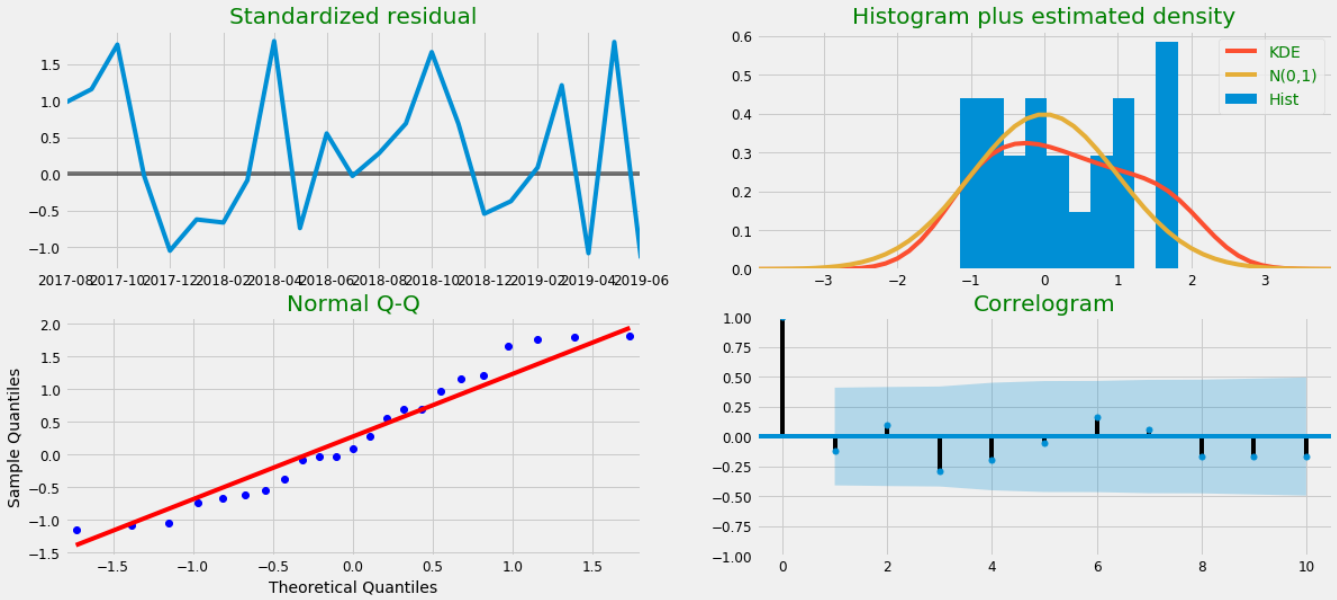
According Peterson, T. (2014) the **AIC** (Akaike information criterion) is an estimator of the relative quality of **statistical** models for a given set of data. Given a collection of models for the data, **AIC** estimates the quality of each model, relative to each of the other models. The low `AIC` value the better. Our output suggests that `SARIMAX(0, 0, 1)x(1, 1, 1, 12)` with `AIC` value of 223.43 is the best combination, so we should consider this to be optimal option.

```
mod = sm.tsa.statespace.SARIMAX(y,
                                order=(0, 0, 1),
                                seasonal_order=(1, 1, 1, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results = mod.fit()
print(results.summary().tables[1])
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.2084      0.282     -0.739      0.460      -0.761       0.344
ar.S.L12      -0.3157      0.184     -1.720      0.085      -0.675       0.044
ma.S.L12       0.3337      0.439      0.760      0.447      -0.527       1.194
sigma2       650.7902    264.392      2.461      0.014     132.592    1168.989
==============================================================================
```

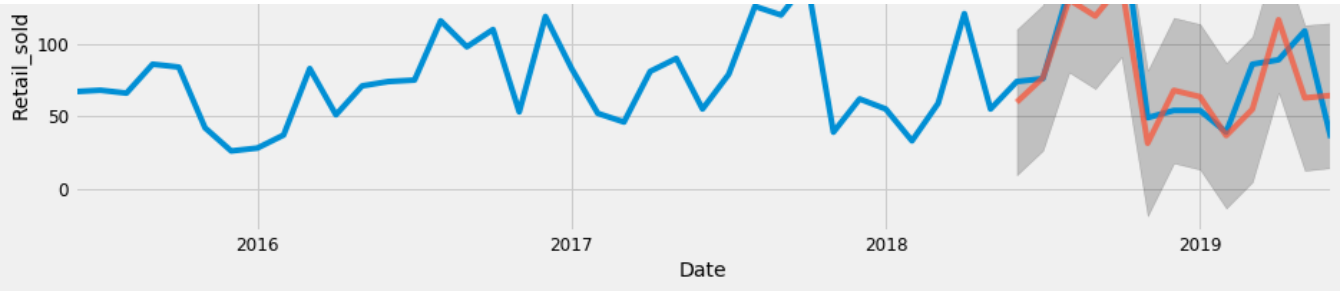
in the “mod = sm.tsa.statespace.SARIMAX” command we need to set up the chosen combination.

```
results.plot_diagnostics(figsize=(18, 8))
plt.show()
```



With the diagnostic above we can visualize important information as the distribution and the Auto correlation function ACF (correlogram). Values upward the “0” has some correlation over the time series data. Values near to “1” demonstrates strongest correlation.

```
pred = results.get_prediction(start=pd.to_datetime('2018-06-01'),
dynamic=False)
pred_ci = pred.conf_int()
ax = y['2015':].plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast',
alpha=.7, figsize=(14, 4))
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)
ax.set_xlabel('Date')
ax.set_ylabel('Retail_sold')
plt.legend()
plt.show()
```



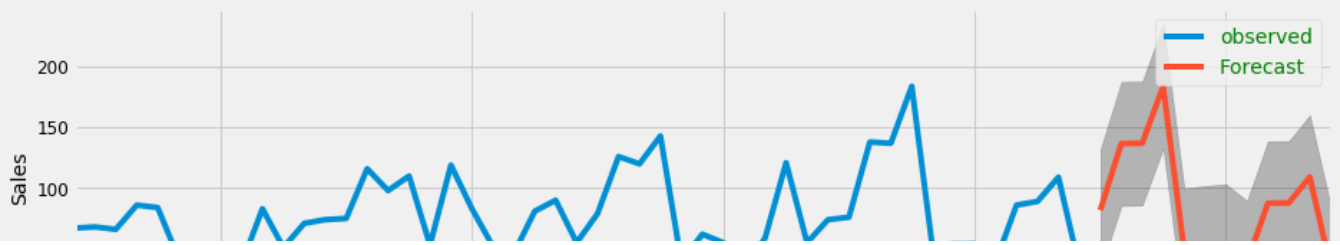
This step consists in comparing the true values with the forecast predictions. Our forecasts fit with the true values very well. The command “pred = results.get_prediction(start=pd.to_datetime(‘2018–06–01’)” determines the period which you would forecast in comparing wiht the true data.

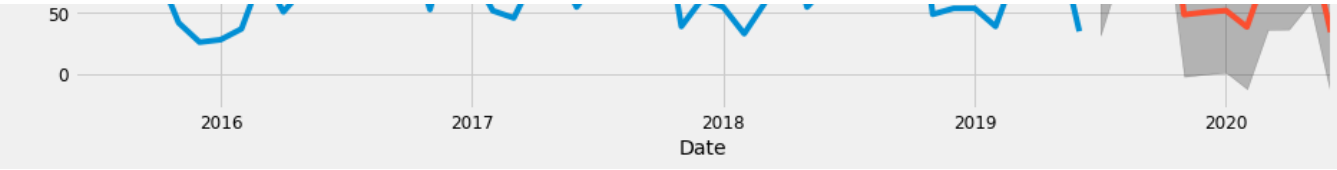
```
y_forecasted = pred.predicted_mean
y_truth = y['2018-06-01':]
mse = ((y_forecasted - y_truth) ** 2).mean()
print('The Mean Squared Error is {}'.format(round(mse, 2)))
print('The Root Mean Squared Error is {}'.format(round(np.sqrt(mse),
2)))
```

The Mean Squared Error is 595.97
The Root Mean Squared Error is 24.41

Obs: In both MSE and RMSE, values closer to zero are better. They are a measure of accuracy.

```
pred_uc = results.get_forecast(steps=12)
pred_ci = pred_uc.conf_int()
ax = y.plot(label='observed', figsize=(14, 4))
pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.25)
ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.legend()
plt.show()
```





Here we forecast the sales for the next 12 months. This parameter can me modified in the line “pred_uc = results.get_forecast(**steps=12**)” of the code.

```
y_forecasted = pred.predicted_mean
y_forecasted.head(12)
```

```
2018-06-01     59.727702
2018-07-01     76.652537
2018-08-01    130.655063
2018-09-01    119.265115
2018-10-01    141.234876
2018-11-01     31.356323
2018-12-01     67.898651
2019-01-01     63.491013
2019-02-01     36.668665
2019-03-01     54.852340
2019-04-01    116.709311
2019-05-01     62.698820
2019-06-01     64.399689
Freq: MS, dtype: float64
```

This step indicate the predicted values of the test we have ran before.

```
y_truth.head(12)
```

```
RETAIL_SOLD_DATE
2018-06-01     74
2018-07-01     76
2018-08-01    138
2018-09-01    137
2018-10-01    184
2018-11-01     49
2018-12-01     54
2019-01-01     54
2019-02-01     39
2019-03-01     86
2019-04-01     89
2019-05-01    109
2019-06-01     35
Freq: MS, Name: FACTORY_CODE, dtype: int64
```

This step indicate the truth values of the data set. We can compare the two series above to measure the model accuracy.

```
pred_ci.head(24)
```

	lower FACTORY_CODE	upper FACTORY_CODE
2019-07-01	31.846160	131.908064
2019-08-01	85.576140	187.785388
2019-09-01	85.876846	188.086066
2019-10-01	132.847593	235.056813
2019-11-01	-2.378853	99.830367
2019-12-01	-0.379817	101.829404
2020-01-01	1.034268	103.243488
2020-02-01	-12.576802	89.632418
2020-03-01	36.492424	138.701644
2020-04-01	36.709164	138.918385
2020-05-01	58.062528	160.269517
2020-06-01	-16.749453	85.404818

In the table above we can visualize the lower and upper values which the model indicate as boundaries for the forecasting.

```
forecast = pred_uc.predicted_mean
forecast.head(12)
```

```
2019-07-01     81.877112
2019-08-01    136.680764
2019-09-01    136.981456
2019-10-01    183.952203
2019-11-01     48.725757
2019-12-01     50.724793
2020-01-01     52.138878
2020-02-01     38.527808
2020-03-01     87.597034
2020-04-01     87.813775
2020-05-01    109.166022
2020-06-01     34.327682
Freq: MS, dtype: float64
```

Finally here we have the sales forecasting for the next 12 months!

We can notice that the sales increasing over time. January and February are the worst months, such as the last years.

This is just a experimentation with a statistical model. I really recommend try recurrent neural networks to forecast, my tip is just use much more data. But this is certainly theme for another article.

Reference:

A Guide to Time Series Forecasting with ARIMA in Python 3

Data Science Statistical Analysis Timeseries Python Programming