Pollen SARIMAX TimeSeries Forecasting

May 28, 2019

In [1]:
```
!pip install --user xlrd
```

```
Requirement already satisfied: xlrd in /home/omar/.local/lib/python3.5/site-packages (1.2.0)
You are using pip version 19.0.3, however version 19.1.1 is available.You should consider upgrad
```

In [2]:
```
!ls
```

```
ARIMA TimeSeries Forecasting.ipynb    Historical Pollution Index
ARIMA TimeSeries Forecasting.pdf      Historical Symptom Logs
ARIMA_timeseries_prediction.ipynb     __MACOSX
cleaned_data                              pollen_history.csv
eda and time series prediction.ipynb  symptom_cause_2019051616-03_001.csv
Historical Pollen Index                   weatherStored_2019051615-59_001.csv
```

In [2]:
```
!ls cleaned_data/
```

```
Pollen_full_data_Eve.xlsx     Symptom Cause Eve.xlsx
Pollution full data Eve.xlsx  Symptom log full data Eve.xlsx
~$Symptom Cause Eve.xlsx
```

In [3]:
```
import pandas as pd
from datetime import datetime

import plotly
plotly.offline.init_notebook_mode()

import plotly.plotly as py
import plotly.graph_objs as go

from statsmodels.tsa.stattools import adfuller
from numpy import log

import itertools
import numpy as np
import matplotlib.pyplot as plt
```

```
import warnings
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'] = 'k'
```

```
In [22]: data = pd.read_excel('cleaned_data/Pollen_full_data_Eve.xlsx')
```

```
In [23]: list(data)
```

```
Out[23]: ['city',
          'date_time',
          'date_time.1',
          'pollenGlobalIndex',
          'grassPollenIndex',
          'treePollenIndex',
          'weedPollenIndex',
          'city_id']
```

```
In [24]: data.head()
```

```
Out[24]:      city           date_time         date_time.1  pollenGlobalIndex  \
         0  Austin 2017-09-06 14:00:00 2017-09-06 14:00:00                  1
         1  Austin 2017-09-06 15:00:00 2017-09-06 15:00:00                  9
         2  Austin 2017-09-06 16:00:00 2017-09-06 16:00:00                  9
         3  Austin 2017-09-06 17:00:00 2017-09-06 17:00:00                  9
         4  Austin 2017-09-06 18:00:00 2017-09-06 18:00:00                  9

            grassPollenIndex  treePollenIndex  weedPollenIndex  city_id
         0                 1                1                1      125
         1                 1                9                3      125
         2                 1                9                3      125
         3                 1                9                3      125
         4                 1                9                3      125
```

```
In [25]: data = data[['date_time', 'pollenGlobalIndex']]
```

```
In [26]: data.head()
```

```
Out[26]:             date_time  pollenGlobalIndex
         0 2017-09-06 14:00:00                  1
         1 2017-09-06 15:00:00                  9
```

```
2 2017-09-06 16:00:00                  9
3 2017-09-06 17:00:00                  9
4 2017-09-06 18:00:00                  9
```

In [27]:
```
data.index[0]
```

Out[27]:
```
0
```

In [28]:
```
data.dropna()
```

Out[28]:

	date_time	pollenGlobalIndex
0	2017-09-06 14:00:00	1
1	2017-09-06 15:00:00	9
2	2017-09-06 16:00:00	9
3	2017-09-06 17:00:00	9
4	2017-09-06 18:00:00	9
5	2017-09-06 19:00:00	9
6	2017-09-06 20:00:00	9
7	2017-09-06 21:00:00	9
8	2017-09-06 22:00:00	9
9	2017-09-06 23:00:00	9
10	2017-09-07 00:00:00	1
11	2017-09-07 01:00:00	1
12	2017-09-07 02:00:00	1
13	2017-09-07 03:00:00	1
14	2017-09-07 04:00:00	1
15	2017-09-07 05:00:00	1
16	2017-09-07 06:00:00	1
17	2017-09-07 07:00:00	1
18	2017-09-07 08:00:00	1
19	2017-09-07 09:00:00	1
20	2017-09-07 10:00:00	1
21	2017-09-07 11:00:00	1
22	2017-09-07 13:00:00	1
23	2017-09-07 14:00:00	1
24	2017-09-07 15:00:00	9
25	2017-09-07 16:00:00	9
26	2017-09-07 17:00:00	9
27	2017-09-07 18:00:00	9
28	2017-09-07 19:00:00	9
29	2017-09-07 20:00:00	9
...	...	...
33205	2017-10-26 12:00:00	1
33206	2017-11-06 16:00:00	1
33207	2017-11-07 12:00:00	1
33208	2017-11-07 13:00:00	1
33209	2017-11-07 14:00:00	1
33210	2017-11-16 15:00:00	1
33211	2017-11-16 16:00:00	1

```
33212 2017-11-16 17:00:00                  1
33213 2017-11-16 18:00:00                  3
33214 2017-11-16 19:00:00                  3
33215 2017-11-16 20:00:00                  3
33216 2017-11-16 21:00:00                  3
33217 2017-11-16 22:00:00                  3
33218 2017-11-16 23:00:00                  3
33219 2017-11-17 00:00:00                  1
33220 2017-11-17 01:00:00                  1
33221 2017-11-17 02:00:00                  1
33222 2017-11-17 03:00:00                  1
33223 2017-11-17 04:00:00                  1
33224 2017-11-17 05:00:00                  1
33225 2017-11-17 06:00:00                  1
33226 2017-11-17 07:00:00                  1
33227 2017-11-17 08:00:00                  1
33228 2017-11-17 09:00:00                  1
33229 2017-11-17 10:00:00                  1
33230 2017-11-17 11:00:00                  1
33231 2017-11-17 13:00:00                  1
33232 2017-11-17 14:00:00                  1
33233 2017-11-17 15:00:00                  1
33234 2017-11-17 16:00:00                  1

[33235 rows x 2 columns]
```

print(data.index.min() + " " + data.index.max())
a = data['date_time'].astype(str).str.split(" ", n = 1, expand = True)
data['date'] = a[0]

```
In [29]: data.head()
```

```
Out[29]:             date_time  pollenGlobalIndex
         0 2017-09-06 14:00:00                  1
         1 2017-09-06 15:00:00                  9
         2 2017-09-06 16:00:00                  9
         3 2017-09-06 17:00:00                  9
         4 2017-09-06 18:00:00                  9
```

```
In [30]: data = data.groupby('date_time')['pollenGlobalIndex'].sum().reset_index()
```

```
In [31]: data.head(20)
```

```
Out[31]:              date_time  pollenGlobalIndex
         0  2017-09-06 14:00:00                  1
         1  2017-09-06 15:00:00                  9
         2  2017-09-06 16:00:00                  9
         3  2017-09-06 17:00:00                  9
         4  2017-09-06 18:00:00                  9
```

```
5  2017-09-06 19:00:00                  9
6  2017-09-06 20:00:00                  9
7  2017-09-06 21:00:00                  9
8  2017-09-06 22:00:00                  9
9  2017-09-06 23:00:00                  9
10 2017-09-07 00:00:00                  1
11 2017-09-07 01:00:00                  1
12 2017-09-07 02:00:00                  1
13 2017-09-07 03:00:00                  1
14 2017-09-07 04:00:00                  1
15 2017-09-07 05:00:00                  1
16 2017-09-07 06:00:00                  1
17 2017-09-07 07:00:00                  1
18 2017-09-07 08:00:00                  1
19 2017-09-07 09:00:00                  1
```

```
In [54]: data['2017-09-30'].max()
```

```
Out[54]: pollenGlobalIndex    6
         dtype: int64
```

```
In [32]: type(data)
```

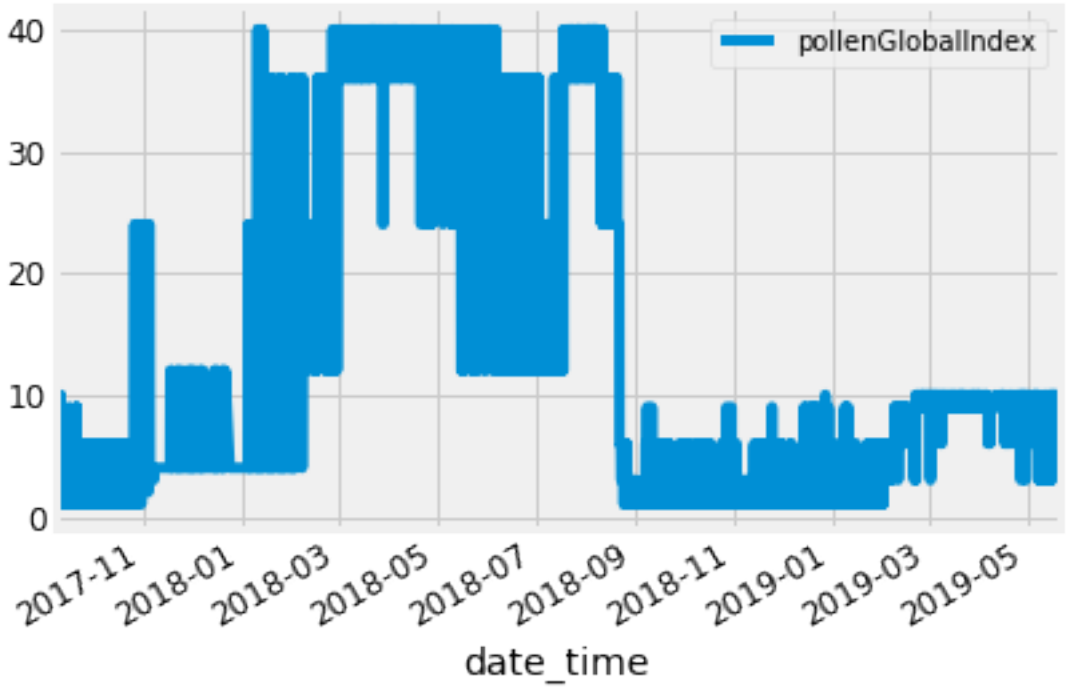
```
Out[32]: pandas.core.frame.DataFrame
```

```
In [34]: data.set_index('date_time', inplace=True)
```

```
In [35]: data.index = pd.to_datetime(data.index)
```

```
In [36]: data.plot()
```

```
Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x7fdf76b3d780>
```



We will use a "grid search" to iteratively explore different combinations of parameters. For each combination of parameters, we fit a new seasonal ARIMA model with the SARIMAX() function from the statsmodels module and assess its overall quality. Once we have explored the entire landscape of parameters, our optimal set of parameters will be the one that yields the best performance for our criteria of interest. Let's begin by generating the various combination of parameters that we wish to assess:

```
In [37]: # Define the p, d and q parameters to take any value between 0 and 2
         p = d = q = range(0, 2)

         # Generate all different combinations of p, q and q triplets
         pdq = list(itertools.product(p, d, q))

         # Generate all different combinations of seasonal p, q and q triplets
         seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]

         print('Examples of parameter combinations for Seasonal ARIMA...')
         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 combinations for Seasonal ARIMA...
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)
```

```
In [38]: for param in pdq:
             for param_seasonal in seasonal_pdq:
                 try:
                     mod = sm.tsa.statespace.SARIMAX(y,
                                                     order=param,
                                                     seasonal_order=param_seasonal,
                                                     enforce_stationarity=False,
                                                     enforce_invertibility=False)

                     results = mod.fit()

                     print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
                 except:
                     continue
```

```
In [39]: y = data
```

The code below iterates through combinations of parameters and uses the SARIMAX function from statsmodels to fit the corresponding Seasonal ARIMA model. Here, the order argument

specifies the (p, d, q) parameters, while the seasonal_order argument specifies the (P, D, Q, S) seasonal component of the Seasonal ARIMA model. After fitting each SARIMAX()model, the code prints out its respective AIC score.

The lesser the AIC score, the better model architecture it is.

Akaike's Information Criterion (AIC): Formally, AIC is defined as 2log+2 where is the maximized log likelihood and is the number of parameters in the model. For the normal regression problem, AIC is an estimate of the Kullback-Leibler discrepancy between a true model and a candidate model.

```
In [40]: for param in pdq:
             for param_seasonal in seasonal_pdq:

                 mod = sm.tsa.statespace.SARIMAX(y,
                                                 order=param,
                                                 seasonal_order=param_seasonal,
                                                 enforce_stationarity=False,
                                                 enforce_invertibility=False)

                 results = mod.fit()

                 print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
```

```
ARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:123602.00864334067
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:114622.03314673914
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:95419.60884102684
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:85345.55725251965
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:94953.96954013838
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:85418.95011318663
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:81182.90817991982
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:81149.13847756415
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:107567.27862059412
ARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:100377.75661785596
ARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:86971.64326628382
ARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:77209.82188750691
ARIMA(0, 0, 1)x(1, 0, 0, 12)12 - AIC:86300.5613795847
ARIMA(0, 0, 1)x(1, 0, 1, 12)12 - AIC:77273.2192864093
ARIMA(0, 0, 1)x(1, 1, 0, 12)12 - AIC:75419.534501847
ARIMA(0, 0, 1)x(1, 1, 1, 12)12 - AIC:75203.00302273597
ARIMA(0, 1, 0)x(0, 0, 0, 12)12 - AIC:72417.05309915464
ARIMA(0, 1, 0)x(0, 0, 1, 12)12 - AIC:72362.28252755731
ARIMA(0, 1, 0)x(0, 1, 0, 12)12 - AIC:82132.71940713383
ARIMA(0, 1, 0)x(0, 1, 1, 12)12 - AIC:71843.04995072834
ARIMA(0, 1, 0)x(1, 0, 0, 12)12 - AIC:72366.17151806387
ARIMA(0, 1, 0)x(1, 0, 1, 12)12 - AIC:71584.38623412396
ARIMA(0, 1, 0)x(1, 1, 0, 12)12 - AIC:73420.69690211516
ARIMA(0, 1, 0)x(1, 1, 1, 12)12 - AIC:71488.85151084632
ARIMA(0, 1, 1)x(0, 0, 0, 12)12 - AIC:72180.46252277913
ARIMA(0, 1, 1)x(0, 0, 1, 12)12 - AIC:72125.294908597
```

```
ARIMA(0, 1, 1)x(0, 1, 0, 12)12 - AIC:81932.89296497568
ARIMA(0, 1, 1)x(0, 1, 1, 12)12 - AIC:71525.54129203035
ARIMA(0, 1, 1)x(1, 0, 0, 12)12 - AIC:72133.03533271703
ARIMA(0, 1, 1)x(1, 0, 1, 12)12 - AIC:71206.88798758533
ARIMA(0, 1, 1)x(1, 1, 0, 12)12 - AIC:72443.26157349144
ARIMA(0, 1, 1)x(1, 1, 1, 12)12 - AIC:71013.80915688911
ARIMA(1, 0, 0)x(0, 0, 0, 12)12 - AIC:72339.78404424048
ARIMA(1, 0, 0)x(0, 0, 1, 12)12 - AIC:72279.45593001845
ARIMA(1, 0, 0)x(0, 1, 0, 12)12 - AIC:80726.77087056923
ARIMA(1, 0, 0)x(0, 1, 1, 12)12 - AIC:70602.79736999792
ARIMA(1, 0, 0)x(1, 0, 0, 12)12 - AIC:72279.44841211641
ARIMA(1, 0, 0)x(1, 0, 1, 12)12 - AIC:70640.51246568309
ARIMA(1, 0, 0)x(1, 1, 0, 12)12 - AIC:71333.48384831286
ARIMA(1, 0, 0)x(1, 1, 1, 12)12 - AIC:70040.10943366261
ARIMA(1, 0, 1)x(0, 0, 0, 12)12 - AIC:72119.8182905429
ARIMA(1, 0, 1)x(0, 0, 1, 12)12 - AIC:72065.41285250867
ARIMA(1, 0, 1)x(0, 1, 0, 12)12 - AIC:80714.63765844211
ARIMA(1, 0, 1)x(0, 1, 1, 12)12 - AIC:70544.61722840657
ARIMA(1, 0, 1)x(1, 0, 0, 12)12 - AIC:72069.39983270015
ARIMA(1, 0, 1)x(1, 0, 1, 12)12 - AIC:70575.26802358094
ARIMA(1, 0, 1)x(1, 1, 0, 12)12 - AIC:71138.82240758542
ARIMA(1, 0, 1)x(1, 1, 1, 12)12 - AIC:69934.62489770666
ARIMA(1, 1, 0)x(0, 0, 0, 12)12 - AIC:72196.89537326491
ARIMA(1, 1, 0)x(0, 0, 1, 12)12 - AIC:72141.72850432101
ARIMA(1, 1, 0)x(0, 1, 0, 12)12 - AIC:81943.36363310218
ARIMA(1, 1, 0)x(0, 1, 1, 12)12 - AIC:71553.50040652996
ARIMA(1, 1, 0)x(1, 0, 0, 12)12 - AIC:72141.18743041545
ARIMA(1, 1, 0)x(1, 0, 1, 12)12 - AIC:71249.1914685459
ARIMA(1, 1, 0)x(1, 1, 0, 12)12 - AIC:72560.2523271135
ARIMA(1, 1, 0)x(1, 1, 1, 12)12 - AIC:71065.17361112984
ARIMA(1, 1, 1)x(0, 0, 0, 12)12 - AIC:71071.76877203808
ARIMA(1, 1, 1)x(0, 0, 1, 12)12 - AIC:71010.33786447193
ARIMA(1, 1, 1)x(0, 1, 0, 12)12 - AIC:81934.89159430447
ARIMA(1, 1, 1)x(0, 1, 1, 12)12 - AIC:70486.73875518187
ARIMA(1, 1, 1)x(1, 0, 0, 12)12 - AIC:71011.5488070289
ARIMA(1, 1, 1)x(1, 0, 1, 12)12 - AIC:69958.61593729301
ARIMA(1, 1, 1)x(1, 1, 0, 12)12 - AIC:71339.9725521626
ARIMA(1, 1, 1)x(1, 1, 1, 12)12 - AIC:70026.97799177296
```

```
In [41]: mod = sm.tsa.statespace.SARIMAX(y,
                                         order=(1, 1, 1),
                                         seasonal_order=(1, 1, 1, 12),
                                         enforce_stationarity=False,
                                         enforce_invertibility=False)

         results = mod.fit()
```