T02 - SARIMA - 2

March 19, 2021

1 SARIMA

- Energy Consumption
- https://www.kaggle.com/robikscube/hourly-energy-consumption
- DAYTON_hourly.csv

1.1 Setting up

[ ]:

```
# Check package version
from packaging import version
import statsmodels

if version.parse(statsmodels.__version__) < version.parse('0.12.1'):
  !pip install statsmodels==0.12.1
```

[ ]:

```
#Perform Dickey-Fuller test:
from statsmodels.tsa.stattools import adfuller
def adf_test(timeseries):
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags␣
 ↪Used','Number of Observations Used'])
    for key,value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)
# kpss_test
from statsmodels.tsa.stattools import kpss
def kpss_test(timeseries):
    print ('Results of KPSS Test:')
    kpsstest = kpss(timeseries, regression='c')
    kpss_output = pd.Series(kpsstest[0:3], index=['Test␣
 ↪Statistic','p-value','Lags Used'])
    for key,value in kpsstest[3].items():
      kpss_output['Critical Value (%s)'%key] = value
    print (kpss_output)
```

2 Data preparation

2.1 Load data

[ ]:

```
import pandas as pd
#df_all = pd.read_csv('DAYTON_hourly.csv')
df_all = pd.read_csv('https://github.com/nnnpooh/energy-class/blob/main/
↪T2%20-%20ARIMA/DAYTON_hourly.csv?raw=true')
df_all.head()
```

[ ]:

```
df_all['Datetime'] = pd.to_datetime(df_all['Datetime'])
df = df_all.set_index('Datetime')
df.head()
```

[ ]:

```
df.describe()
```

[ ]:

```
df.info()
```

[ ]:

```
import matplotlib.pyplot as plt
df.plot(figsize=(10, 3))
plt.show()
```

2.2 Resample data

[ ]:

```
max_sample = 365
#df_avg = df['DAYTON_MW'].resample('MS').mean()
df_avg = df['DAYTON_MW'].resample('D').mean()
if max_sample > 0:
    df_avg = df_avg.iloc[-max_sample-1:-1]
display(df_avg.describe())

#Convert result to DataFrame
df_avg = pd.DataFrame(df_avg)
```

[ ]:

```
#Check for NaN values
#df_avg.info()
#df_avg[df_avg.isna().any(axis=1)]
```

[ ]:

```
df_avg.plot(figsize=(10, 3))
plt.show()
```

[ ]:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(df_avg, lags=30)
plot_pacf(df_avg, lags=30)
plt.show()
```

[ ]:

```
import statsmodels.api as sm
decomposition = sm.tsa.seasonal_decompose(df_avg, model='additive')
```

```
fig = decomposition.plot()
fig.set_size_inches(11,8)
```

[ ]:

```
adf_test(df_avg)
kpss_test(df_avg)
```

2.3 Test for stationariy and seasonality

[ ]:

```
df_diff = df_avg.diff(1).diff(7).dropna()
plot_acf(df_diff, lags=30)
plot_pacf(df_diff, lags=30)
fig, ax = plt.subplots(figsize=(10, 3))
df_diff.plot(ax=ax)
plt.show()
adf_test(df_diff)
kpss_test(df_diff)
```

3 Model selection

[ ]:

```
import itertools
p = [0,1,2]
d = [1]
q = [0,1,2]
P = [0,1]
D = [1]
Q = [0,1]
lag = [7]
params = list(itertools.product(p, d, q, P, D, Q, lag))
print(f"Number of models to test: {len(params)}")
```

[ ]:

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
import numpy as np
df_results = pd.DataFrame()
for param in params:
    pdq = param[0:3]
    PDQL = param[3:7]
    try:
        mod = SARIMAX(df_avg, order=pdq, seasonal_order=PDQL)
        results = mod.fit(method = 'powell',start_params=np.random.random(7))
        data = {'param': pdq, 'param_seasonal': PDQL, 'AIC':results.aic }
        df_results = df_results.append(data, ignore_index=True)
    except:
        continue
df_results = df_results.sort_values(by='AIC',ascending=True)
```

[ ]:

```
df_results
```

4 Model training

```
[ ]: rank = 1
     pdq = df_results.iloc[rank-1,1]
     PDQL = df_results.iloc[rank-1,2]

     print(f"Using␣
      ↪({pdq[0]},{pdq[1]},{pdq[2]})({PDQL[0]},{PDQL[1]},{PDQL[2]},{PDQL[3]})")

     mod = SARIMAX(df_avg, order=pdq, seasonal_order=PDQL)
     results = mod.fit(method = 'powell', start_params=np.random.random(7))
```

5 Model evaluation

```
[ ]: fig = results.plot_diagnostics(figsize=(10, 6))
     fig.tight_layout()
```

```
[ ]: pred = results.get_prediction(start=df_avg.index[1], end=df_avg.index[-1],␣
      ↪dynamic=False)
     comb = pd.concat([df_avg, pred.predicted_mean], axis=1).dropna()
     comb['error'] = comb.iloc[:,0] - comb.iloc[:,1]
     comb['percentage'] = comb['error']/comb.iloc[:,0]*100

     MAE = comb['error'].abs().mean()
     RMSE = np.sqrt((comb['error']**2).mean())
     MAPE = comb['percentage'].abs().mean()

     print(f"Mean absolute error: {MAE:6.3f}")
     print(f"Root mean squared error: {RMSE:6.3f}")
     print(f"Mean absolute percentage error: {MAPE:6.3f}")
```

6 Plotting and forecasting

```
[ ]: num_forecast = 10

     start_dt = df_avg.index[10]
     end_dt_data = df_avg.index[-1]
     if num_forecast > 0:
         end_dt = end_dt_data + num_forecast * end_dt_data.freq
     else:
         end_dt = end_dt_data

     print(start_dt)
     print(end_dt)
```

[ ]:

```
pred = results.get_prediction(start=pd.to_datetime(start_dt), end=pd.
 ↪to_datetime(end_dt), dynamic=False)
pred_ci = pred.conf_int()
ax = df_avg.plot(label='Observed')
pred.predicted_mean.plot(ax=ax, label='Prediction', alpha=.8, figsize=(14, 6))
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('Y')
ax.set_xlim(start_dt,end_dt)
yp_max = pred.predicted_mean.max()
yp_min = pred.predicted_mean.min()
yp_mean = pred.predicted_mean.mean()
ax.set_ylim(yp_min-0.1*yp_mean,yp_max+0.1*yp_mean)
plt.legend()
plt.show()
```

[ ]:

```
if num_forecast > 0:
    display(pred.predicted_mean.loc[end_dt_data:end_dt])
```

[ ]: