Scenario1

April 22, 2019

```
In [1]: import csv
        import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        from statsmodels.tsa.stattools import acf
        from statsmodels.graphics.tsaplots import plot_acf
        import itertools
        from statsmodels.tsa.statespace.sarimax import SARIMAX
        import statsmodels.api as sm
        import warnings
```

```
In [89]: #define function for ADF test
         from statsmodels.tsa.stattools import adfuller
         def adf_test(timeseries):
             #Perform Dickey-Fuller test:
             print ('Results of Dickey-Fuller Test:')
             dftest = adfuller(timeseries, autolag='AIC')
             dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used',
             for key,value in dftest[4].items():
                dfoutput['Critical Value (%s)'%key] = value
             print (dfoutput)
```

0.0.1 To run this code, please make sure 'hyd_post.txt' is in the same directionary.

```
In [90]: data = np.genfromtxt(fname='hyd_post.txt',delimiter=",",skip_header=True,dtype=np.floa
         df_data=pd.DataFrame(data=data,columns=['Monthly Resolution'])
```

```
In [91]: fig=plt.figure(figsize=(12,5))
         ax = fig.add_subplot(111)
         ax.set_title('Monthly resolution of the level of a body of water')
         ax.text(0.5,-0.15, "Fig. 1.1", size=12, ha="center", transform=ax.transAxes,weight='bo
         plt.plot(data)
         plt.show();
```

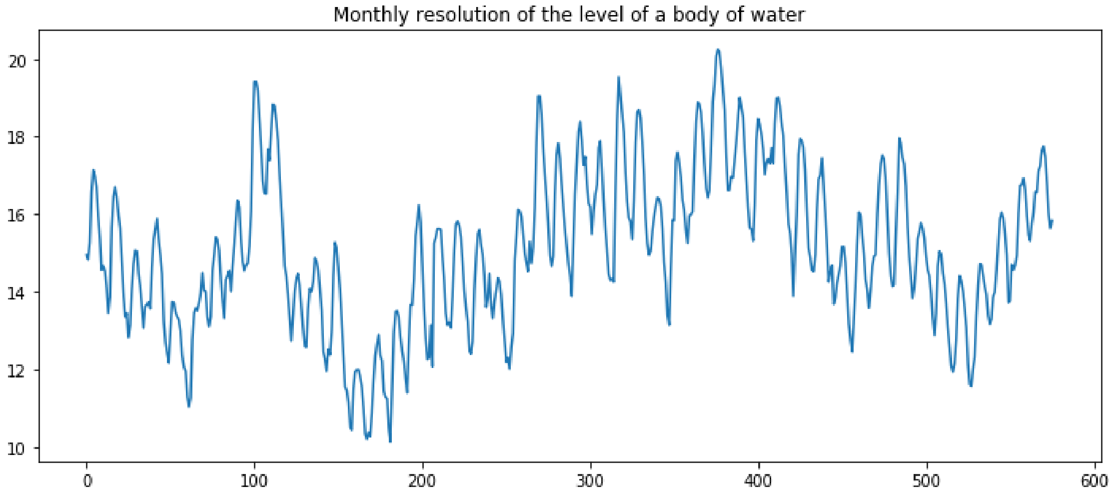

Fig. 1.1

```
In [94]: adf_test(data)
```

```
Results of Dickey-Fuller Test:
Test Statistic                  -2.667544
p-value                          0.079825
#Lags Used                      18.000000
Number of Observations Used    557.000000
Critical Value (1%)             -3.442145
Critical Value (5%)             -2.866743
Critical Value (10%)            -2.569541
dtype: float64
```

```
In [72]: fig = plt.figure(figsize=(12,5))
         ax = fig.add_subplot(111)
         ax.text(0.5,-0.15, "Fig. 1.2", size=12, ha="center", transform=ax.transAxes,weight='bo
         plot_acf(data,lags=np.arange(0,48),ax=ax)
```

Out[72]:

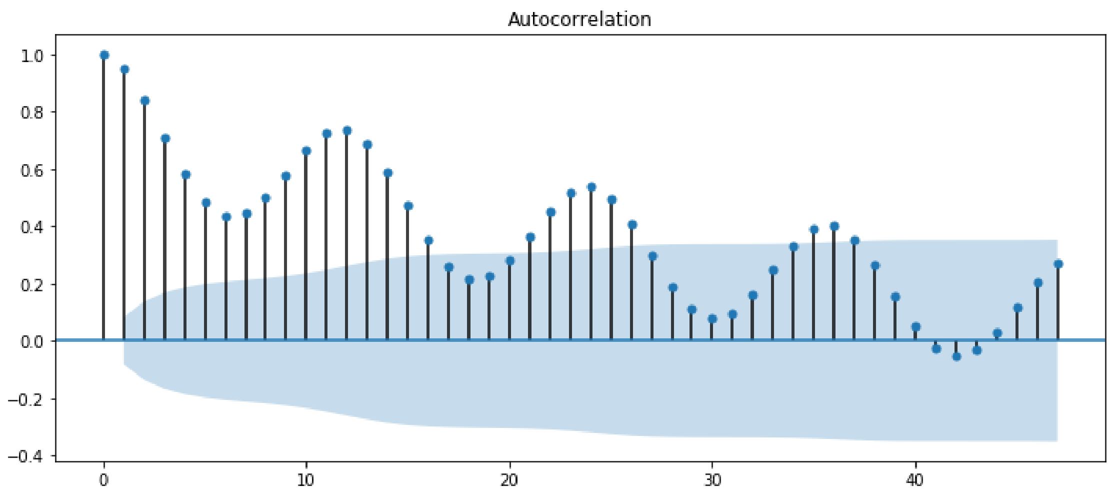

Fig. 1.2

Observed seasonal effect of lag 12

```
In [25]: # Define the p, d and q parameters to take any value between 0 and 3
         p = d = q = range(0, 3)

         # 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, 0, 2, 12)
SARIMAX: (0, 0, 2) x (0, 1, 0, 12)
SARIMAX: (0, 0, 2) x (0, 1, 1, 12)
```

Choose the best parameter by the lowest AIC/BIC

```
In [28]: #Using grid search, we can identify the set of parameters that produces the best fitt
         #to our time series data. We can proceed to analyze this particular model in more dep
         warnings.filterwarnings("ignore") # specify to ignore warning messages
         AIC=10000
         BIC=10000
```

```
        order_aic=...
        seasonal_order_aic=...
        order_bic=...
        seasonal_order_bic=...
        for param in pdq:
            for param_seasonal in seasonal_pdq:
                try:
                    mod = sm.tsa.statespace.SARIMAX(data,
                                                    order=param,
                                                    seasonal_order=param_seasonal,
                                                    enforce_stationarity=False,
                                                    enforce_invertibility=False)

                    results = mod.fit()
                    a=results.aic
                    if a<=AIC:
                        AIC=a
                        order_aic=param
                        seasonal_order_aic=param_seasonal
                    b=results.bic
                    if b<=BIC:
                        BIC=b
                        order_bic=param
                        seasonal_order_bic=param_seasonal
                    # print('ARIMA{}x{} - AIC:{}, BIC:{}'.format(param, param_seasonal, resul
                except:
                    continue
```

```
In [29]: AIC,BIC
```

```
Out[29]: (596.3422624079133, 622.8894237518107)
```

```
In [30]: order_aic,seasonal_order_aic
```

```
Out[30]: ((1, 0, 2), (1, 1, 2, 12))
```

```
In [31]: order_bic,seasonal_order_bic
```

```
Out[31]: ((1, 0, 2), (0, 1, 2, 12))
```

The optimal parameters determined by AIC and BIC are different. Here we will use the one chosen by the lowest AIC by the model diagostic done below.

0.0.2 Using the optimal parameters calculated above

```
In [73]: mod = sm.tsa.statespace.SARIMAX(df_data,
                                         order=(1, 0, 2),
                                         seasonal_order=(1, 1, 2, 12),
                                         enforce_stationarity=False,
```

```
                                        enforce_invertibility=False)

results = mod.fit()
print(results.summary())
```

```
                                 Statespace Model Results
==========================================================================================
Dep. Variable:                  Monthly Resolution   No. Observations:                 576
Model:             SARIMAX(1, 0, 2)x(1, 1, 2, 12)   Log Likelihood                -291.171
Date:                             Sun, 21 Apr 2019   AIC                            596.342
Time:                                     23:53:03   BIC                            626.344
Sample:                                          0   HQIC                           608.079
                                             - 576
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.9550      0.017     57.066      0.000       0.922       0.988
ma.L1          0.2364      0.035      6.725      0.000       0.167       0.305
ma.L2          0.1352      0.042      3.205      0.001       0.053       0.218
ar.S.L12      -0.6150      0.287     -2.146      0.032      -1.177      -0.053
ma.S.L12      -0.4665      0.297     -1.571      0.116      -1.048       0.115
ma.S.L24      -0.6171      0.310     -1.988      0.047      -1.225      -0.009
sigma2         0.1512      0.010     15.901      0.000       0.133       0.170
===================================================================================
Ljung-Box (Q):                       51.15   Jarque-Bera (JB):               218.79
Prob(Q):                              0.11   Prob(JB):                         0.00
Heteroskedasticity (H):               0.67   Skew:                             0.79
Prob(H) (two-sided):                  0.01   Kurtosis:                         5.70
===================================================================================

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

```
In [88]: results.plot_diagnostics(figsize=(15, 12))
         plt.text(-2,-1.4, "Fig. 1.3", size=12, ha="center",weight='bold')
         plt.show()
```

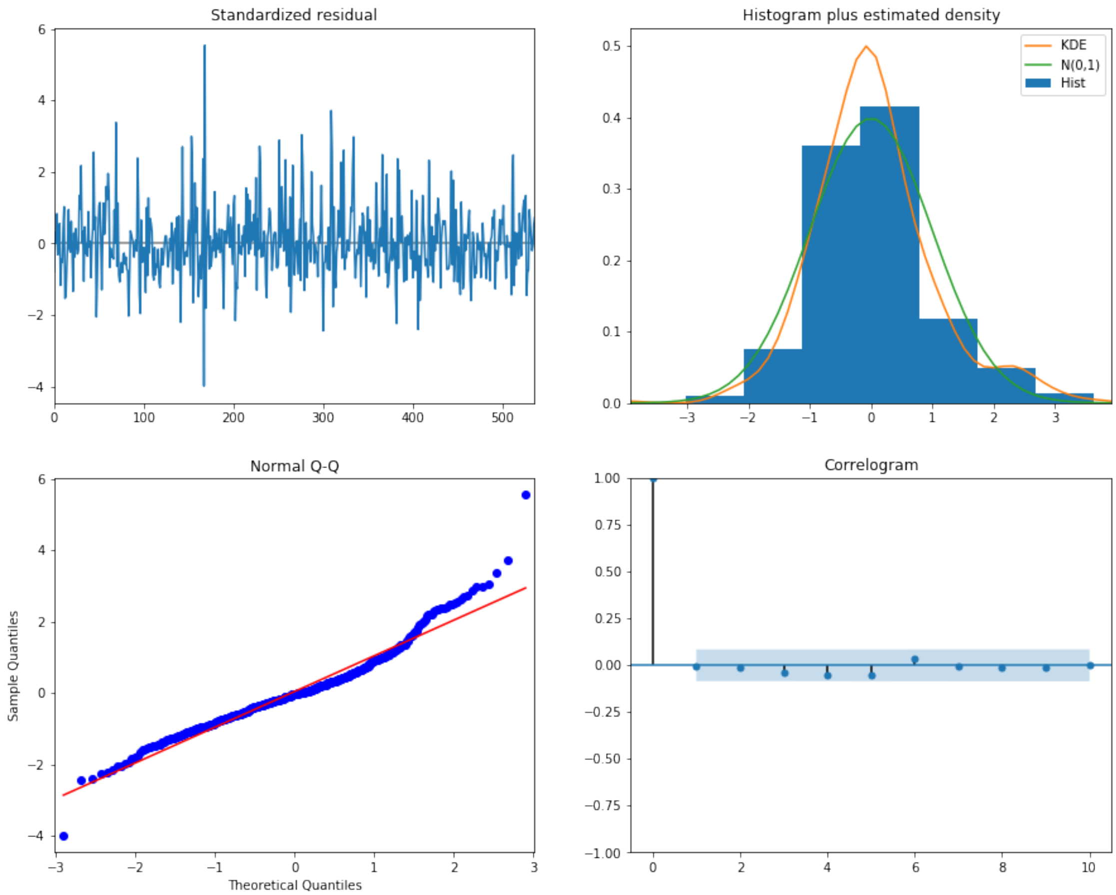

Fig. 1.3

0.0.3 Using the parameters got by auto.arima from R

In [11]:
```
mod_R = sm.tsa.statespace.SARIMAX(df_data,
                                  order=(2, 0, 0),
                                  seasonal_order=(1, 1, 0, 12),
                                  enforce_stationarity=False,
                                  enforce_invertibility=False)


results_by_R = mod_R.fit()
print(results_by_R.summary())
```

```
                           Statespace Model Results
==========================================================================================
Dep. Variable:                Monthly Resolution   No. Observations:                  576
Model:            SARIMAX(2, 0, 0)x(1, 1, 0, 12)   Log Likelihood                -380.958
Date:                           Sun, 21 Apr 2019   AIC                            769.916
Time:                                   22:42:24   BIC                            787.156
Sample:                                        0   HQIC                           776.653
```

```
                                          - 576
Covariance Type:                            opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.1561      0.032     36.583      0.000       1.094       1.218
ar.L2         -0.2190      0.033     -6.622      0.000      -0.284      -0.154
ar.S.L12      -0.5314      0.030    -17.970      0.000      -0.589      -0.473
sigma2         0.2340      0.011     21.161      0.000       0.212       0.256
===================================================================================
Ljung-Box (Q):                      128.99   Jarque-Bera (JB):                99.11
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.61   Skew:                             0.37
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.94
===================================================================================

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

Html_file= open("a.html","w") Html_file.write(a) Html_file.close() imgkit.from_file('a.html', 'out.jpg')

In [12]:
```
results_by_R.plot_diagnostics(figsize=(15, 12))
plt.text(-2,-1.4, "Fig. 1.4", size=12, ha="center",weight='bold')
plt.show()
```

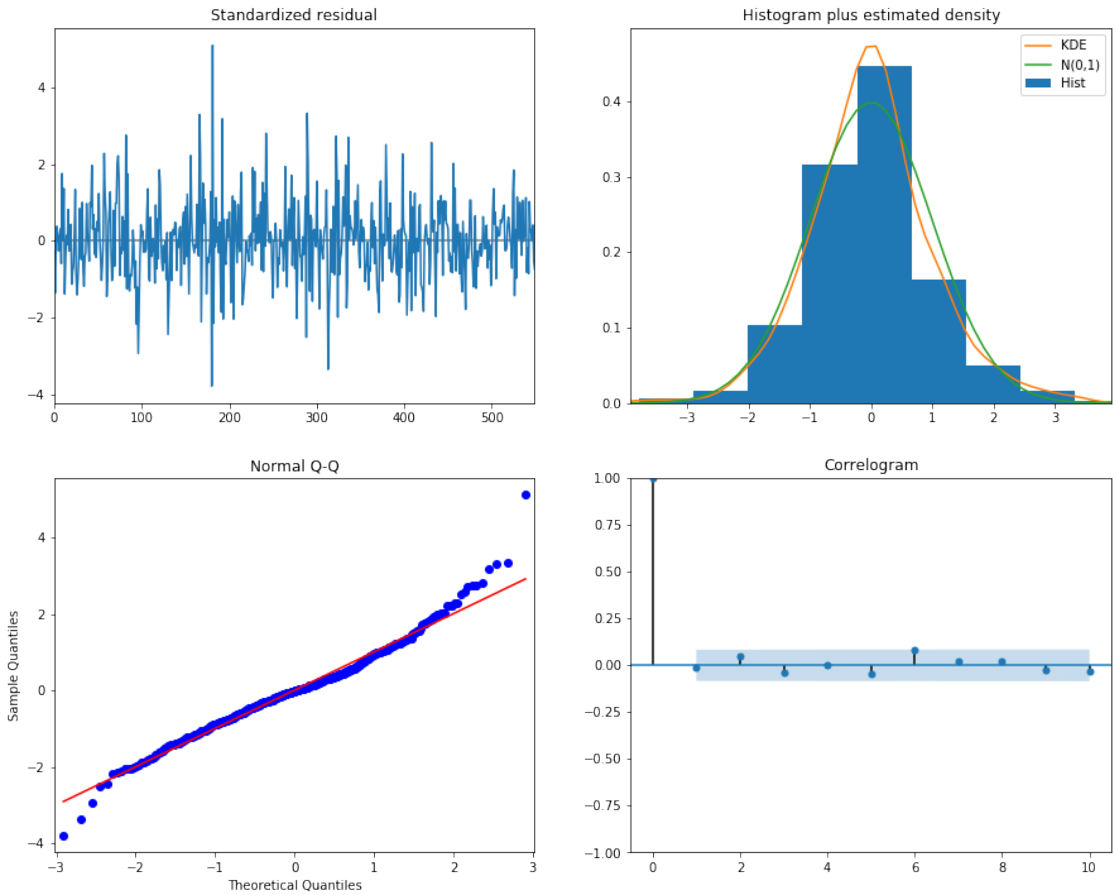

Fig. 1.4

I would prefer SARIMA(1, 0, 2)x(1, 1, 2, 12) by the Ljung-Box Test.

```
In [95]: # Get forecast 24 steps ahead in future
         pred_uc = results.get_forecast(steps=24)

         # Get 95% confidence intervals of forecasts
         pred_ci = pred_uc.conf_int(alpha=0.05)

In [96]: tocsv=pred_uc.predicted_mean.values
         np.savetxt('../Result in CSV/Li_Scenario1.csv', tocsv, delimiter=',')

In [97]: mean=pred_uc.predicted_mean.values
         var=pred_uc.var_pred_mean
         prediction_u=mean+1.96*var
         prediction_l=mean-1.96*var

In [98]: pred_pi = pred_uc.conf_int(alpha=0.05)
         pred_pi['lower Monthly Resolution']=prediction_l
         pred_pi['upper Monthly Resolution']=prediction_u
```

In [101]:
```
ax = df_data[500:].plot(label='observed', figsize=(12, 5))
pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_ci.index,
                pred_pi.iloc[:, 0],
                pred_pi.iloc[:, 1], color='y', alpha=.35,label='95% Prediction Inter
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='b', alpha=.50,label='95% Confidence Inter
ax.text(0.5,-0.1, "Fig. 1.5", size=12, ha="center", transform=ax.transAxes,weight='bo
plt.legend(['Observed','Forcast','95% Prediction Interval','95% Confidence Interval']
plt.show()
```

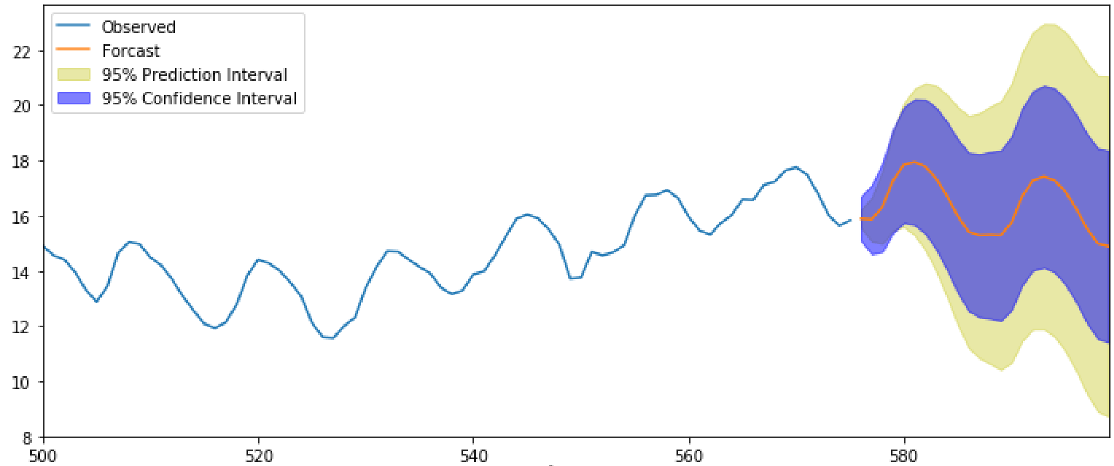

Fig. 1.5