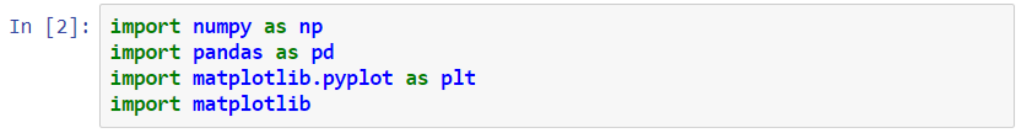
Mohamed Sadok Gastli — Waterloo ID #: 20853612
Mohita Chaudhary — Waterloo ID #: 20830560

SYDE 631 Assignment #2

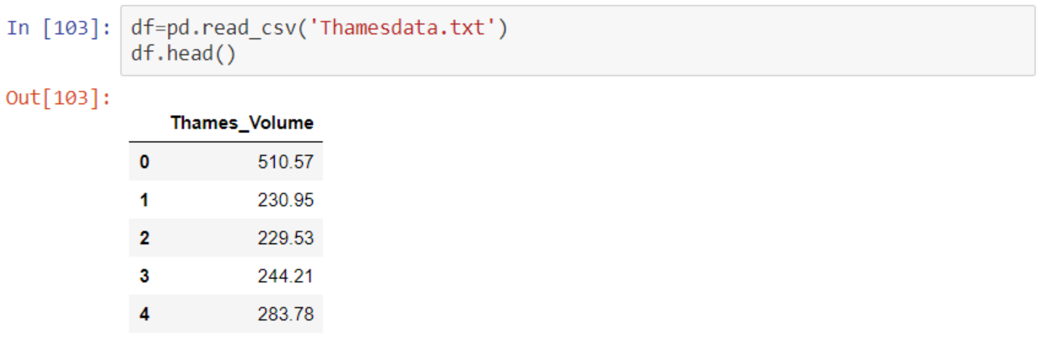
The data set used for this Assignment is time series data of the River Thames at Kingston, which keeps in account the volume of the River from year 1883 to the year 2018. This data has been downloaded from the National River Flow Archive using the following link: https://nrfa.ceh.ac.uk/data/station/peakflow/39001, under the peak flow data. The unit of the flow of the river is in cubic meter per second.

Exploratory Data Analysis

The first step to apply any model to the data is to perform the Exploratory Data Analysis on it:



```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import matplotlib
```

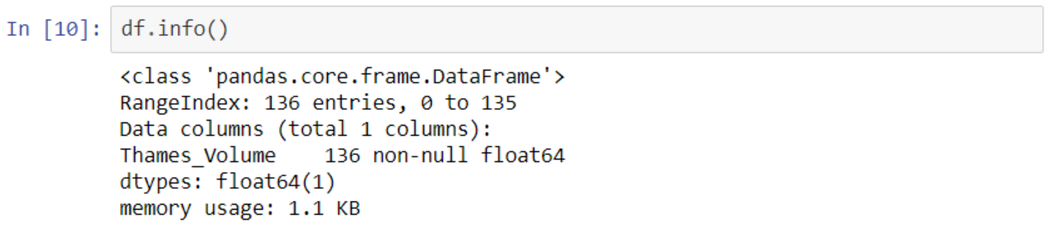


```
In [103]: df=pd.read_csv('Thamesdata.txt')
          df.head()
```

Out[103]:

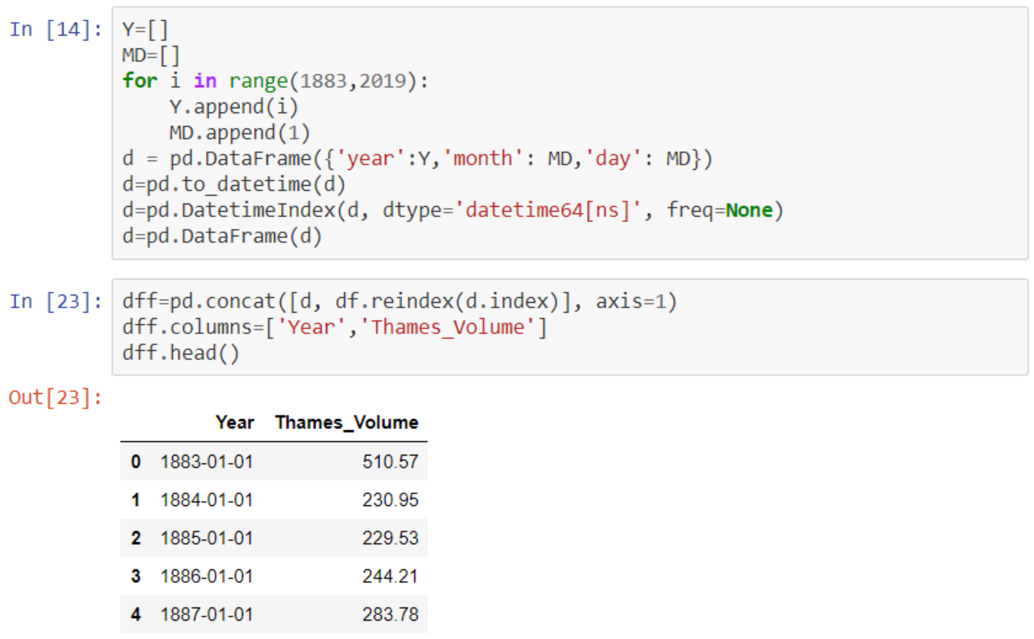
	Thames_Volume
0	510.57
1	230.95
2	229.53
3	244.21
4	283.78

Data information which depicts 136 entries and one column, the data type for the values is float64 type:



```
In [10]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 136 entries, 0 to 135
Data columns (total 1 columns):
Thames_Volume    136 non-null float64
dtypes: float64(1)
memory usage: 1.1 KB
```

We added the corresponding Year for the data, by adding a year column in our data. The date is in DatetimeIndex format:



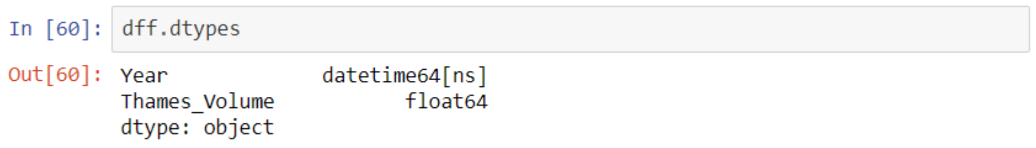
```
In [14]: Y=[]
         MD=[]
         for i in range(1883,2019):
             Y.append(i)
             MD.append(1)
         d = pd.DataFrame({'year':Y,'month': MD,'day': MD})
         d=pd.to_datetime(d)
         d=pd.DatetimeIndex(d, dtype='datetime64[ns]', freq=None)
         d=pd.DataFrame(d)
```

```
In [23]: dff=pd.concat([d, df.reindex(d.index)], axis=1)
         dff.columns=['Year','Thames_Volume']
         dff.head()
```

Out[23]:

	Year	Thames_Volume
0	1883-01-01	510.57
1	1884-01-01	230.95
2	1885-01-01	229.53
3	1886-01-01	244.21
4	1887-01-01	283.78

The data is finally converted into a dataframe with the year in yyyy/mm/dd format which has the data type datetime64[ns], and the corresponding flow measurements of datatype float64:



```
In [60]: dff.dtypes

Out[60]: Year                datetime64[ns]
         Thames_Volume              float64
         dtype: object
```

Solution for 5.6 and 5.11

5.6: Examine a graph of time series which is of direct interest to you. Describe general statistical properties of the series that you can detect in the graph.

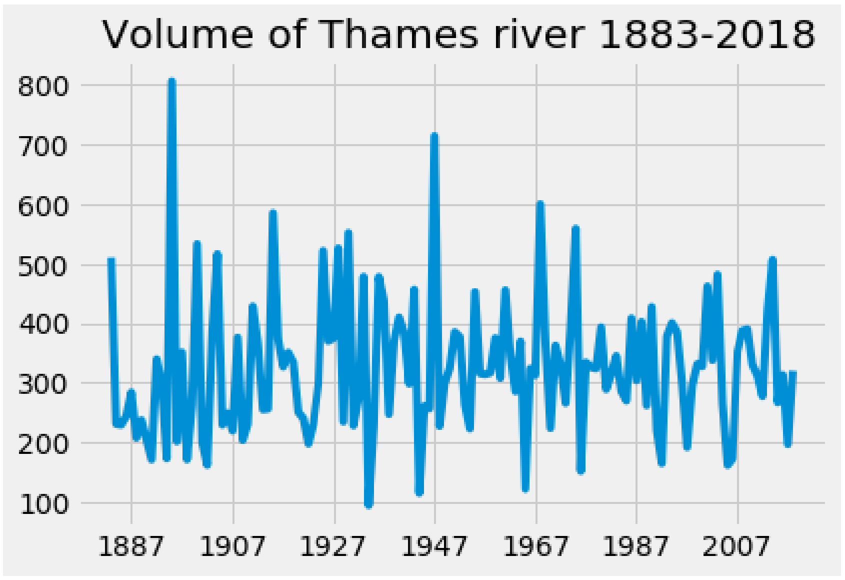
5.11: Select a non-seasonal Time Series that is of interest to you. Obtain each of the identification graphs in Sections 5.3.3 to 5.3.7 for the series. Based upon these identification results, what is the most appropriate type of ARMA or ARIMA model to fit to the dataset?

The General Statistics

We have obtained the following for the data:
(1) The general plot volume vs year
(2) The variance which comes out to be 13470.93.
(3) The mean which comes out to be 325.93 m^3/s.

The plot of the flow v/s year:

```
In [28]: Year = dff.Year.astype('O')
         fig, ax = plt.subplots()
         ax.plot(Year, dff.Thames_Volume)
         ax.set_title(' Volume of Thames river 1883-2018')

Out[28]: Text(0.5,1,' Volume of Thames river 1883-2018')
```



The variance of the data is 13470.93:

```
In [29]: dff.var()

Out[29]: Thames_Volume    13470.926548
         dtype: float64
```

The mean of the data is 325.93 m^3/s:

```
In [30]: dff.mean()

Out[30]: Thames_Volume    325.925919
         dtype: float64
```

(4) The Augmented Dickey Fuller Test (ADF), which is done to check for stationarity. The following are the results:

```
In [20]: from statsmodels.tsa.stattools import adfuller
         result = adfuller(dff['Thames_Volume'])
         print('ADF Statistic: %f' % result[0])
         print('p-value: %f' % result[1])
         print('Critical Values:')
         for key, value in result[4].items():
             print('\t%s: %.3f' % (key, value))
```

```
ADF Statistic: -8.115788
p-value: 0.000000
Critical Values:
        1%: -3.481
        5%: -2.884
        10%: -2.578
```

ADF value = -8.115788, p-value = 0.00000001

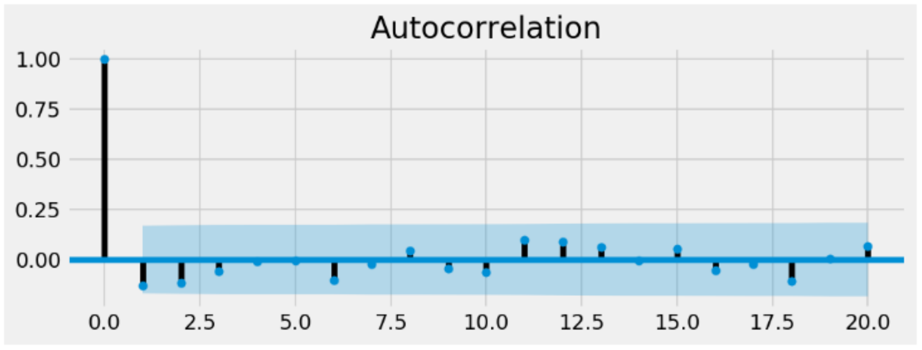
Alternative-hypothesis = Stationary, **Null-hypothesis** = Non-stationary

Since, the ADF value is negative thus we reject the null hypothesis. Moreover, the p-value is less than 0.05, which indicates the data is stationary. Thus, it can be concluded from the ADF and p-values that the given data is stationary.

(5) A plot for ACF
(6) A plot for PACF

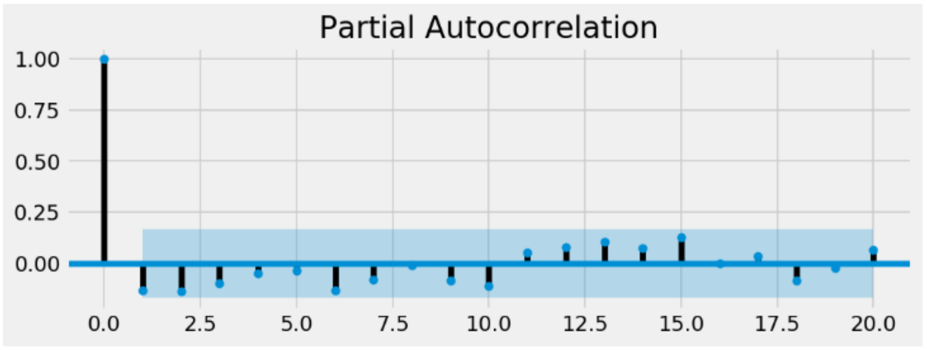
Plot for Autocorrelation Function (ACF):

```
In [126]: from statsmodels.graphics.tsaplots import plot_acf
          plot_acf(df,lags=20)
          plt.show()
```



Plot for Partial Autocorrelation Function (PACF):

```
In [94]: #from statsmodels.graphics.tsaplots import plot_pacf
         plot_pacf(df, lags=20)
         plt.show()
```

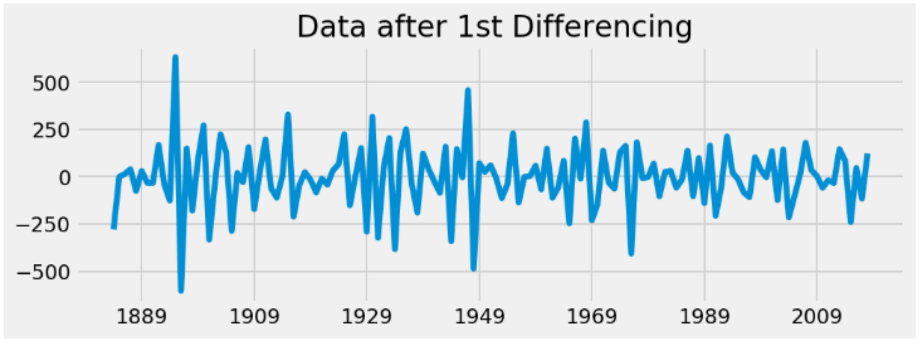


From the ACF and the PACF plots we can see that they are not attenuating and not cutting off as well, thus we cannot clearly determine which model we can apply, we further perform differencing to determine the new ACF and PACf plots. IACF and IPACF plots cannot be used since there is no seasonal component in the data and hence nothing can be extracted from these plots, since these plots are only for the seasonal data.

Performing differencing on the data in order to make it more stationary

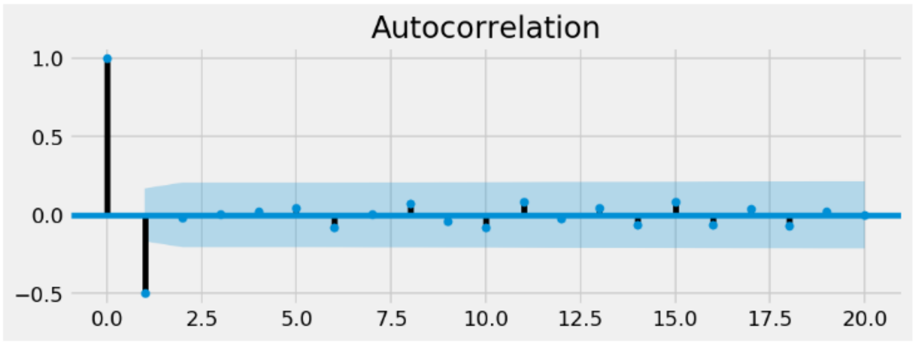
Also, to deduce some inferences from the ACF and PACF plot obtained for the differenced data:

```
In [131]: fig, ax = plt.subplots()
          ax.plot(Year, df.Thames_Volume.diff())
          ax.set_title(' Data after 1st Differencing')

Out[131]: Text(0.5,1,' Data after 1st Differencing')
```



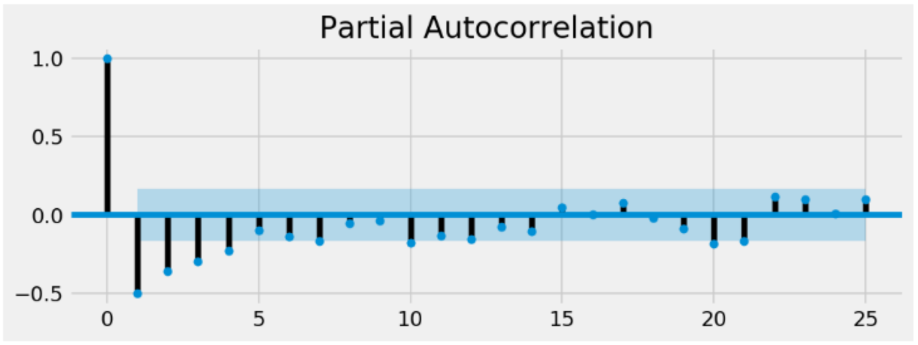
ACF plot after differencing

```
In [144]: plot_acf(df.Thames_Volume.diff().dropna(), lags=20)

          plt.show()
```



PACF plot after differencing

```
In [174]: plot_pacf(df.Thames_Volume.diff().dropna(), lags=25)

          plt.show()
```



After analysing the above ACF and PACF plots we do not get any conditions for AR, MA and ARMA since they are neither cutting off nor attenuating, thus we choose ARIMA model.

Solution for 6.11

Two approaches for employing the AIC in conjunction with the model construction are described in Section 6.3.3. Using an annual time series of your choice, employ these two procedures for determining the best overall ARMA or ARIMA model to fit to the series.

According to the Section 6.3.3, the first method is to calculate AIC value for all possible combinations of p,d,q as done below:

The ARIMA model takes in three variables namely p, d and q.

p is the periods to lag for. The p helps us to adjust the line that is being fitted to forecast the series.

d refers to the number of differencing transformations required by the time series to get stationary, since differencing makes the time series stationary.

q denotes the lag of the error component, here the error component is a part of the time series not explained by trend or seasonality.

Since, we know that the model with the least value of AIC (Akaike Information Criterion) is the best model therefore, we tested our ARIMA model with the different values of p, d and q in the function ARIMA(p,q,d), in order to obtain the least value of AIC (Akaike Information Criterion). We tried it for the following:

(1,2,3) --> AIC: 1668.527, Log Likelihood: -828.264, BIC:1685.914

(1,1,2) --> AIC: 1673.125, Log Likelihood: -831.562, BIC:1687.651

(1,1,3) --> AIC: 1674.854, Log Likelihood: -831.427, BIC:1692.286

(0,1,1) --> AIC: 1677.177, Log Likelihood: -835.589, BIC:1685.893

(0,0,1) --> AIC: 1680.758, Log Likelihood: -837.379, BIC:1689.496

(1,0,0) --> AIC: 1681.686, Log Likelihood: --837.843, BIC:1690.424

(0,0,0) --> AIC: 1682.075, Log Likelihood: -839.037, BIC:1687.900

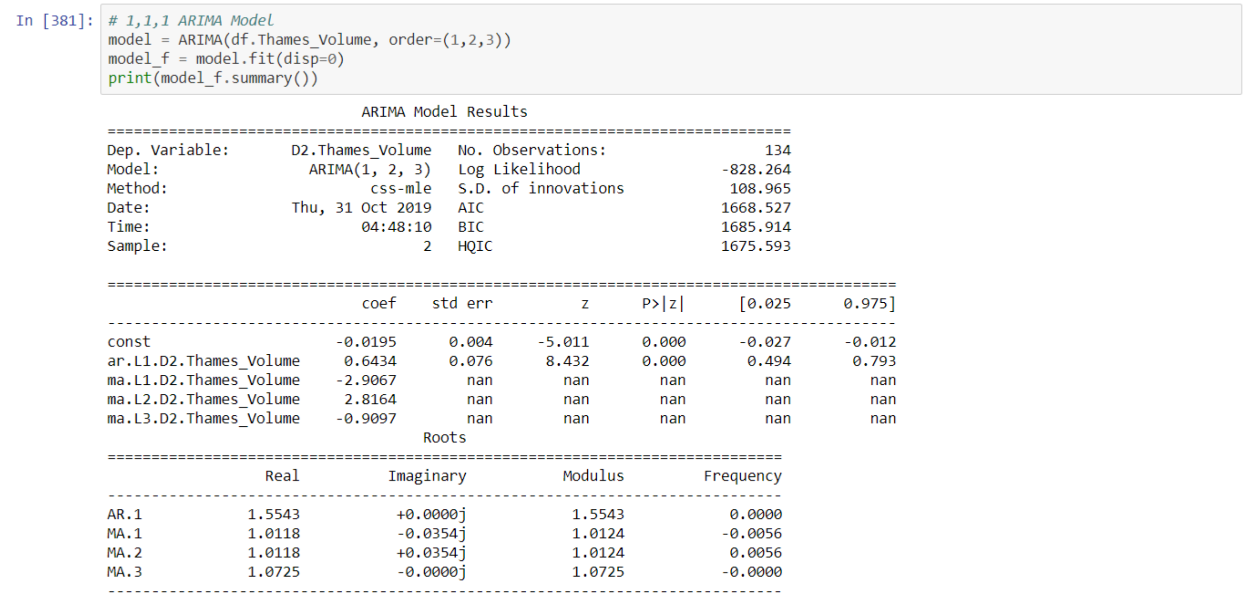
(1,1,0) --> AIC: 1742.486, Log Likelihood: -868.243, BIC:1751.202

(0,1,0) --> AIC: 1779.815, Log Likelihood: -887.908, BIC:1785.626

(1,0,1) gives a ValueError: The computed initial MA coefficients are not invertible You should induce invertibility, choose a different model order, or you can pass your own start_params.

Also, d>3 is not allowed for the model used.

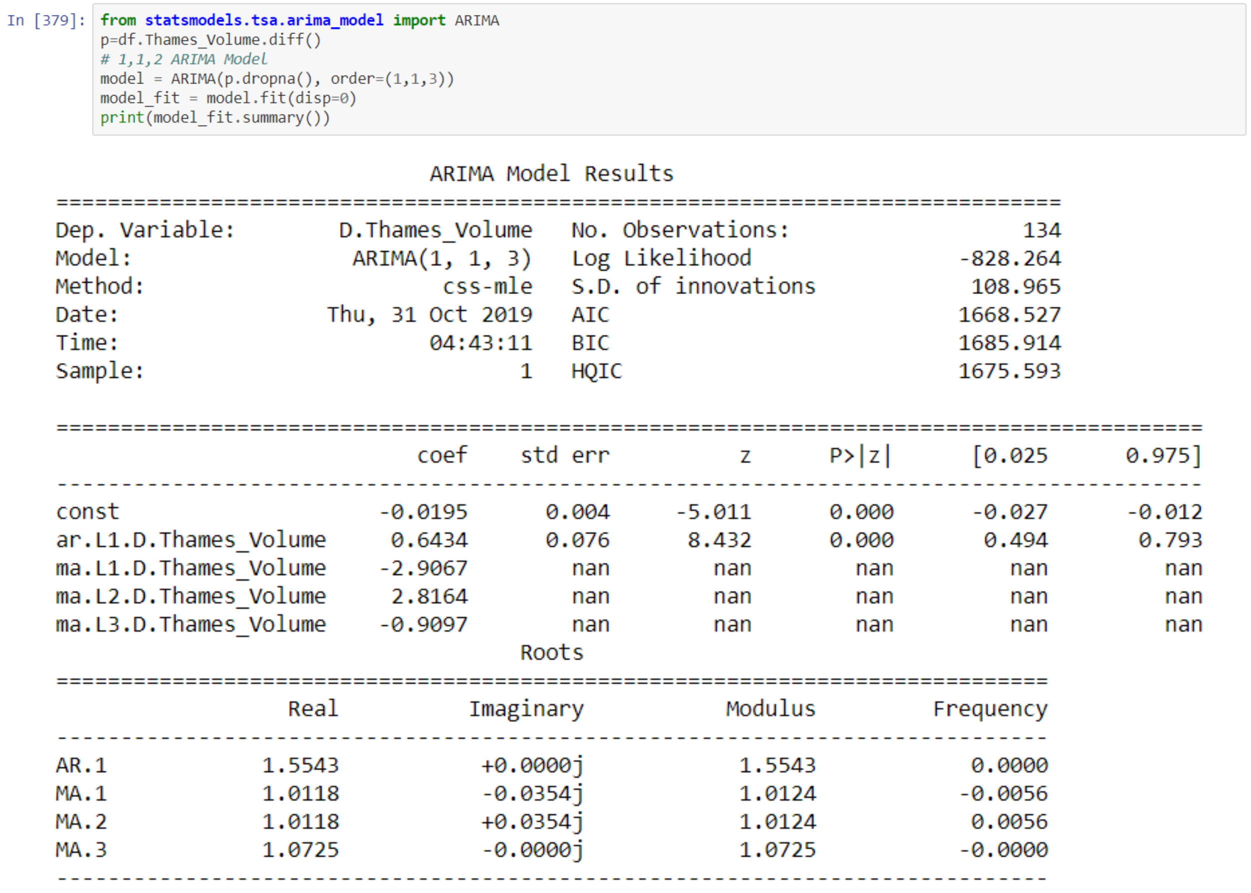
From the above results we deduce that the ARIMA model gives the least value of AIC for p=1, d=2 and r=3. Thus, the best ARIMA model for our data is ARIMA(1,2,3)



```
In [381]: # 1,1,1 ARIMA Model
          model = ARIMA(df.Thames_Volume, order=(1,2,3))
          model_f = model.fit(disp=0)
          print(model_f.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:       D2.Thames_Volume   No. Observations:                  134
Model:                 ARIMA(1, 2, 3)   Log Likelihood                -828.264
Method:                       css-mle   S.D. of innovations            108.965
Date:                Thu, 31 Oct 2019   AIC                           1668.527
Time:                        04:48:10   BIC                           1685.914
Sample:                             2   HQIC                          1675.593

==========================================================================================
                             coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
const                     -0.0195      0.004     -5.011      0.000      -0.027      -0.012
ar.L1.D2.Thames_Volume     0.6434      0.076      8.432      0.000       0.494       0.793
ma.L1.D2.Thames_Volume    -2.9067        nan        nan        nan         nan         nan
ma.L2.D2.Thames_Volume     2.8164        nan        nan        nan         nan         nan
ma.L3.D2.Thames_Volume    -0.9097        nan        nan        nan         nan         nan
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.5543           +0.0000j            1.5543            0.0000
MA.1            1.0118           -0.0354j            1.0124           -0.0056
MA.2            1.0118           +0.0354j            1.0124            0.0056
MA.3            1.0725           -0.0000j            1.0725           -0.0000
-----------------------------------------------------------------------------
```

If we apply ARIMA model on an already differenced data, then ARIMA(1,1,3) on differenced data gives similar result as ARIMA(1,2,3) on non-differenced data.

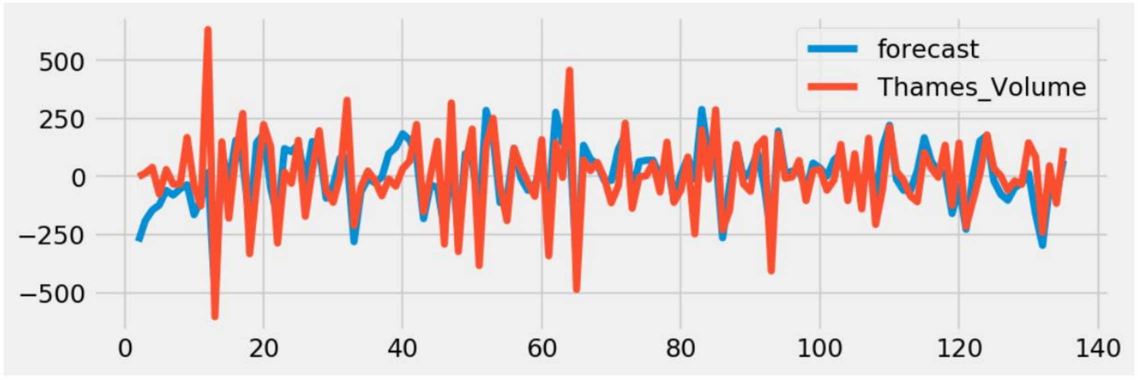


```
In [379]: from statsmodels.tsa.arima_model import ARIMA
          p=df.Thames_Volume.diff()
          # 1,1,2 ARIMA Model
          model = ARIMA(p.dropna(), order=(1,1,3))
          model_fit = model.fit(disp=0)
          print(model_fit.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:        D.Thames_Volume   No. Observations:                  134
Model:                 ARIMA(1, 1, 3)   Log Likelihood                -828.264
Method:                       css-mle   S.D. of innovations            108.965
Date:                Thu, 31 Oct 2019   AIC                           1668.527
Time:                        04:43:11   BIC                           1685.914
Sample:                             1   HQIC                          1675.593

=========================================================================================
                            coef    std err          z      P>|z|      [0.025      0.975]
-----------------------------------------------------------------------------------------
const                    -0.0195      0.004     -5.011      0.000      -0.027      -0.012
ar.L1.D.Thames_Volume     0.6434      0.076      8.432      0.000       0.494       0.793
ma.L1.D.Thames_Volume    -2.9067        nan        nan        nan         nan         nan
ma.L2.D.Thames_Volume     2.8164        nan        nan        nan         nan         nan
ma.L3.D.Thames_Volume    -0.9097        nan        nan        nan         nan         nan
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.5543           +0.0000j            1.5543            0.0000
MA.1            1.0118           -0.0354j            1.0124           -0.0056
MA.2            1.0118           +0.0354j            1.0124            0.0056
MA.3            1.0725           -0.0000j            1.0725           -0.0000
-----------------------------------------------------------------------------
```

The forecast using the ARIMA(1,2,3) model.

```
In [380]: model_fit.plot_predict(dynamic=False)
          plt.show()
```



According to Section 6.3.3, the second method is to obtain Maximum Likelihood Estimation of the box-cox transformation parameter(lambda) for the best model and fix lambda at this value.

We performed the Whiteness Test on the residuals as seen below and found them to be white, also we found them to be normally distributed and with constant variance, thus we do not need to perform Box Cox Transformation on our data.

Residuals

The residuals in a time series model are what is left over after fitting a model.

The residuals have the following properties:

1. The residuals are uncorrelated. If there are correlations between residuals, then there is information left in the residuals which should be used in computing forecasts.
2. They should have a mean 0.
3. They should have constant variance.
4. The residuals should be normally distributed.

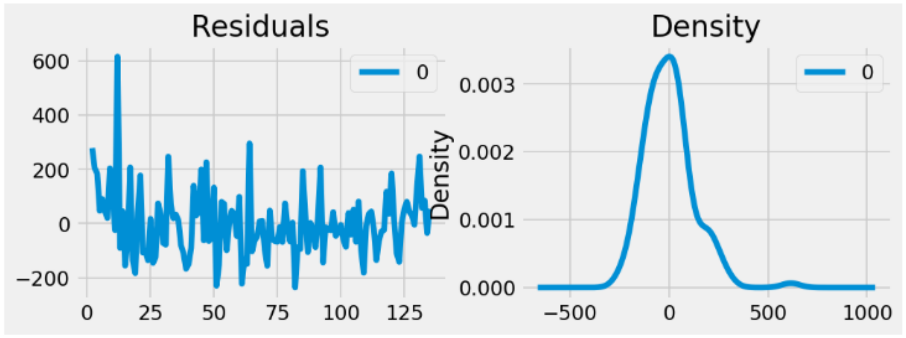
We find that the residuals which we obtained from the ARIMA(1,2,3), are normally distributed.

They have a mean of 1.03 which is close to 0. Also, from the ACF graph we deduce that they are uncorrelated as the values fall within the confidence boundaries. From the residual plot we deduce that they have a constant variance. Therefore, we do not need any transformation to achieve these properties and the model which have chosen seems correct.

Calculating the residuals and plotting the density of the residues

We have calculated the Residuals using the python function stated below (model.resid()). We also plotted the Kernel Density Estimation for the residuals (Gaussian Plot) and found them to be normally distributed.

```
In [382]: # Plot residual errors
          residuals = pd.DataFrame(model_fit.resid)
          fig, ax = plt.subplots(1,2)
          residuals.plot(title="Residuals", ax=ax[0])
          ## kde=kernel density estimation
          residuals.plot(kind='kde', title='Density', ax=ax[1])
          plt.show()
```



Mean for the residuals:

```
In [383]: residuals.mean()

Out[383]: 0    1.02821
          dtype: float64
```

Box-Ljung p-value -> Test for Independence (THE WHITENESS TEST) for ARIMA(1,2,3) on non-differenced data and ARIMA(1,1,3) on the differenced data.

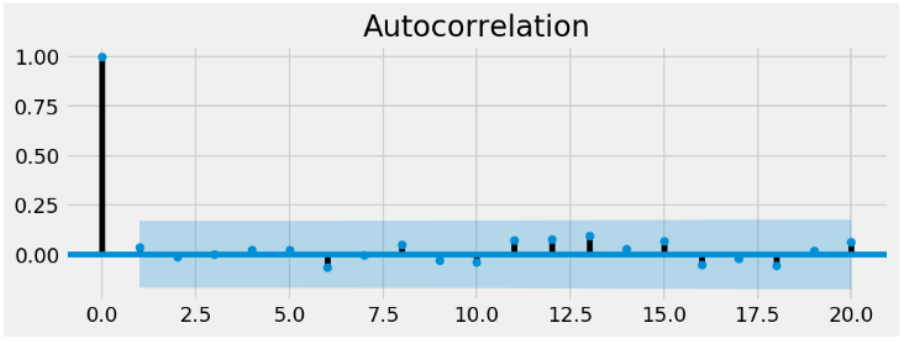
We performed the Box-Ljung test on the residuals. This test is to check whether the discrete white noise being a good fit to the residuals or not. The piece of code below returns two values, the first one is The Ljung-Box test statistic and the second one is the p value. The p value comes out to be 0.91970588 and since, the p-value is significantly larger than 0.05 which is strong evidence for discrete white noise being a good fit to the residuals. Hence, the ARIMA(1,1,2) model is a good fit.

```
In [280]: sm.stats.acorr_ljungbox(residuals, lags=[10])

Out[280]: (array([4.5402008]), array([0.91970588]))
```

The ACF plot for the Residuals

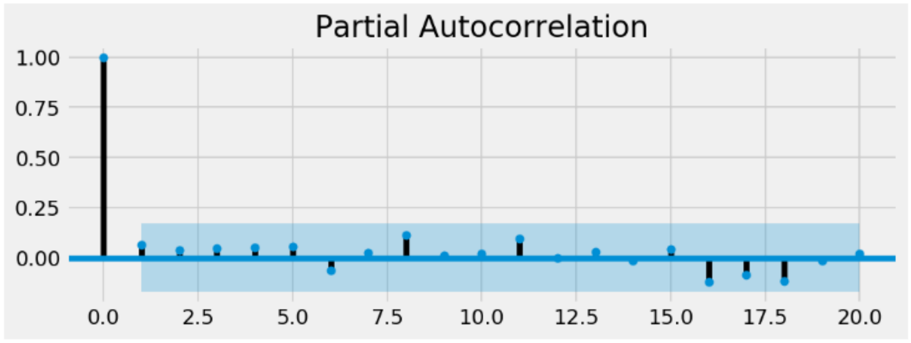
Parameter estimates fall within the confidence intervals and are close to the true parameter values of the simulated ARIMA series. Hence, the residuals looking like a realisation of discrete white noise, and they are not correlated.

```
In [208]: plot_acf(residuals.dropna(), lags=20)
          plt.show()
```



The PACF plot for the residuals:

```
In [282]: plot_pacf(residuals.dropna(), lags=20)
          plt.show()
```

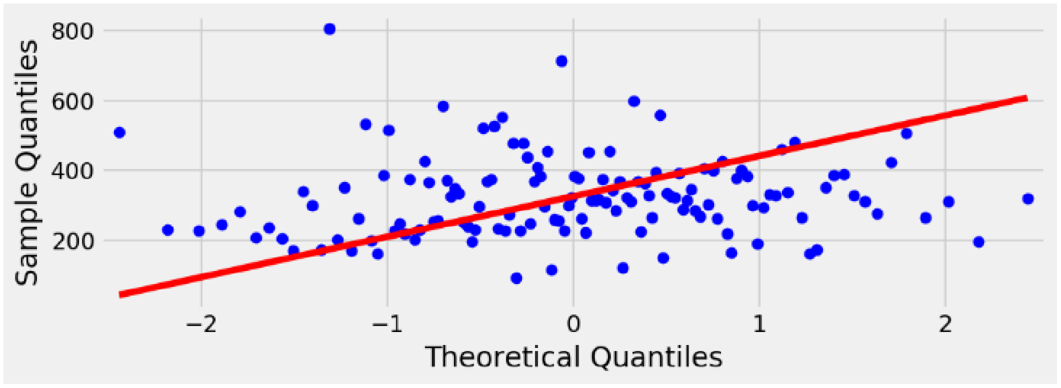


Solution for 7.8 (iii)

To check if the series is Gaussian using the normality plot.

The following plot shows that our data is not normally distributed originally (Before differencing):

```
In [124]: from statsmodels.graphics.gofplots import qqplot
          from matplotlib import pyplot
          # seed the random number generator

          # q-q plot
          qqplot(df, line='s')
          pyplot.show()
```



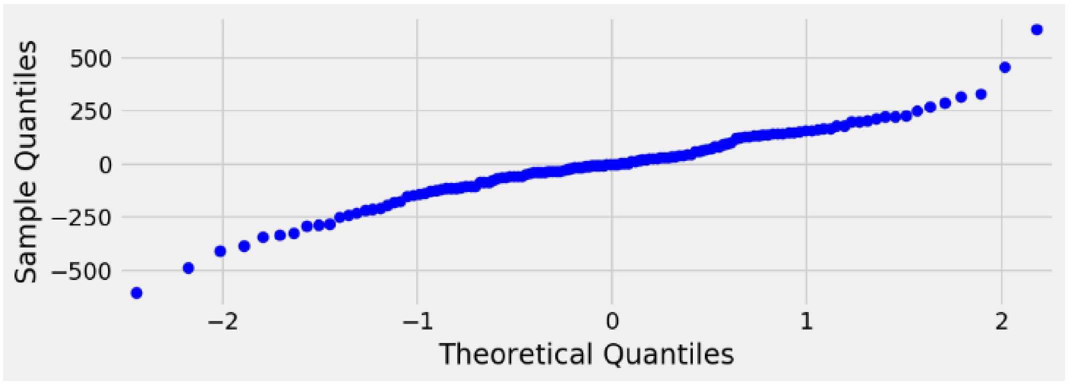
The normality test showing that the data is not normally distributed originally (Before differencing):

```
In [46]: from numpy.random import seed
         from numpy.random import randn
         from scipy.stats import shapiro


         # normality test
         stat, p = shapiro(df)
         print('Statistics=%.3f, p=%.3f' % (stat, p))
         # interpret
         alpha = 0.05
         if p > alpha:
             print('Sample looks Gaussian (fail to reject H0)')
         else:
             print('Sample does not look Gaussian (reject H0)')
```

```
Statistics=0.952, p=0.000
Sample does not look Gaussian (reject H0)
```

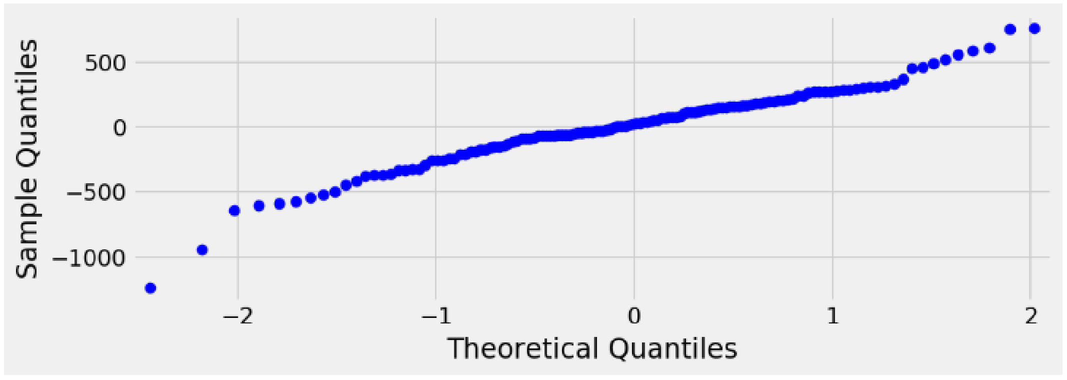
The normality plot showing that our data follows a normal distribution after differencing:

```
In [362]: p=df.Thames_Volume.diff()
          qqplot(p, line='s')
          pyplot.show()
```



The normality plot showing that our data follows a normal distribution after further differencing (d=2). The skewing at the edges of the line are due the limited number of samples, which is expected:

```
In [363]: p=p.to_frame()
          f1=p.Thames_Volume.diff()
          qqplot(f1, line='s')
          pyplot.show()
```



Solution for 7.1 and 7.12:

Applying ARMA model on the Differenced data (More stationary)

The following are the values of AIC and Log Likelihood for the various ARMA models we tried:

(1,0) --> AIC: 1742.486 Log Likelihood: -868.243 SE: 150.114

(1,1) --> AIC: 1676.854 Log Likelihood: -834.427 SE: 114.759

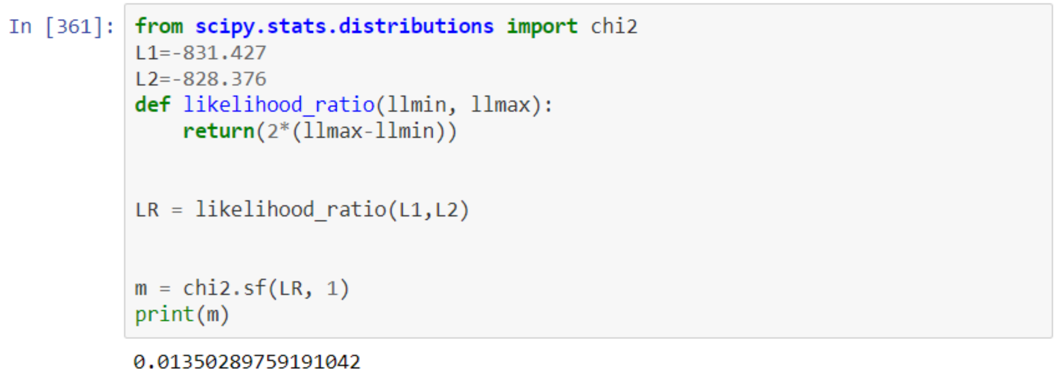
(1,2) --> AIC: 1673.125 Log Likelihood: -831.562 SE: 111.837

(1,3) --> AIC: 1674.854 Log Likelihood: -831.427 SE: 111.715

(1,4) --> AIC: 1670.752 Log Likelihood: -828.376 SE: 109.869

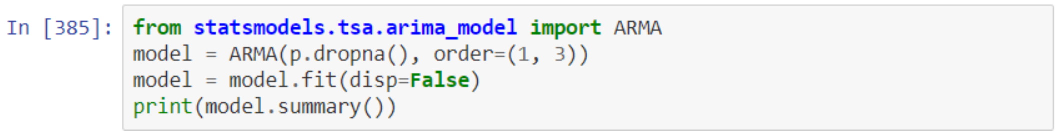
The value of SE's decreases as we overfit the model.

Let's consider two ARMA models with the least AIC values, ARMA(1,3) and ARMA(1,4), in order to perform the likelihood ratio test we consider the log likelihood values for these two models. The test value comes out to be 0.01350289759191042, thus we reject the alternative model (overfitted ARMA(1,4)) in favour of the simpler model ARMA(1,3):



```
In [361]: from scipy.stats.distributions import chi2
          L1=-831.427
          L2=-828.376
          def likelihood_ratio(llmin, llmax):
              return(2*(llmax-llmin))


          LR = likelihood_ratio(L1,L2)


          m = chi2.sf(LR, 1)
          print(m)

          0.01350289759191042
```

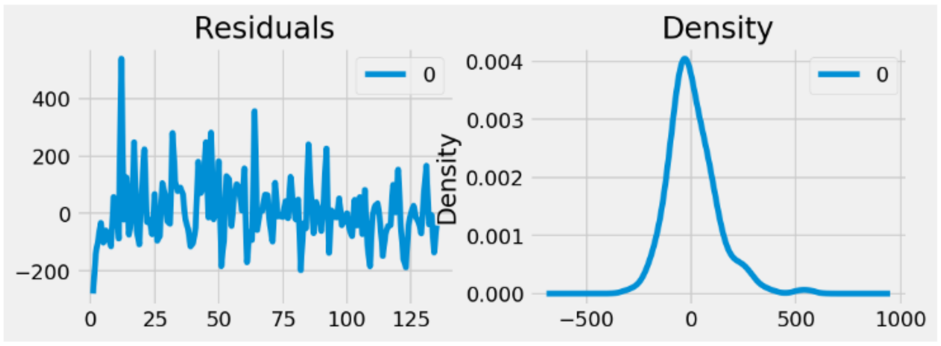
The data-frame 'p' used below is the differenced dataset, which stores the differenced values of the original dataset and hence is more stationary:



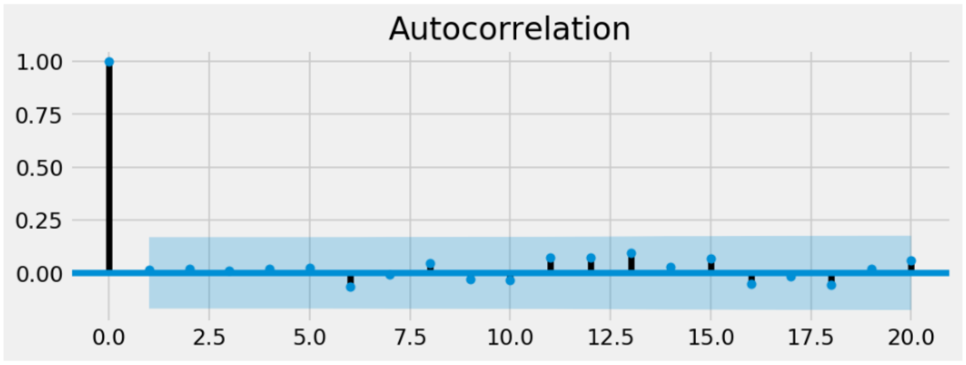
```
In [385]: from statsmodels.tsa.arima_model import ARMA
          model = ARMA(p.dropna(), order=(1, 3))
          model = model.fit(disp=False)
          print(model.summary())
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:          Thames_Volume   No. Observations:                  135
Model:                     ARMA(1, 3)   Log Likelihood                -831.427
Method:                       css-mle   S.D. of innovations            111.715
Date:                Thu, 31 Oct 2019   AIC                           1674.854
Time:                        04:52:11   BIC                           1692.286
Sample:                             0   HQIC                          1681.938

=======================================================================================
                          coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------
const                   0.1868      0.130      1.440      0.152      -0.067       0.441
ar.L1.Thames_Volume     0.4838      0.244      1.985      0.049       0.006       0.962
ma.L1.Thames_Volume    -1.6728      0.246     -6.788      0.000      -2.156      -1.190
ma.L2.Thames_Volume     0.6050      0.369      1.641      0.103      -0.118       1.328
ma.L3.Thames_Volume     0.0678      0.129      0.526      0.600      -0.185       0.320
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            2.0669           +0.0000j            2.0669            0.0000
MA.1            1.0000           +0.0000j            1.0000            0.0000
MA.2            1.3126           +0.0000j            1.3126            0.0000
MA.3          -11.2336           +0.0000j           11.2336            0.5000
-----------------------------------------------------------------------------
```

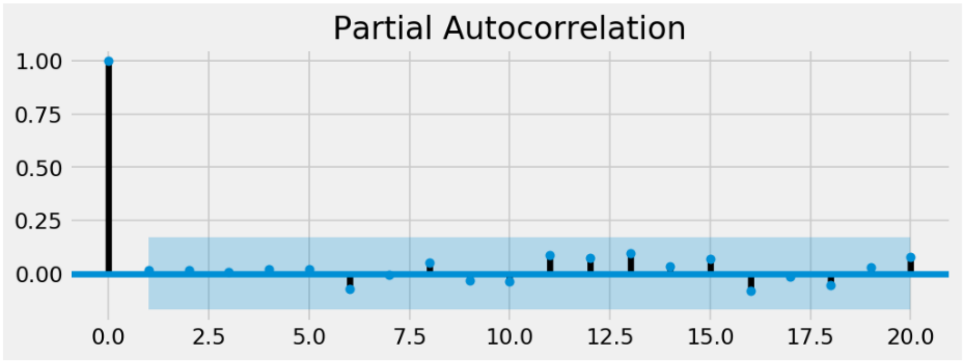
```
In [386]: # Plot residual errors
residuals = pd.DataFrame(model.resid)
fig, ax = plt.subplots(1,2)
residuals.plot(title="Residuals", ax=ax[0])
residuals.plot(kind='kde', title='Density', ax=ax[1])
plt.show()
```



```
In [387]: plot_acf(residuals.dropna(), lags=20)
          plt.show()
```



```
In [389]: plot_pacf(residuals.dropna(), lags=20)
          plt.show()
```



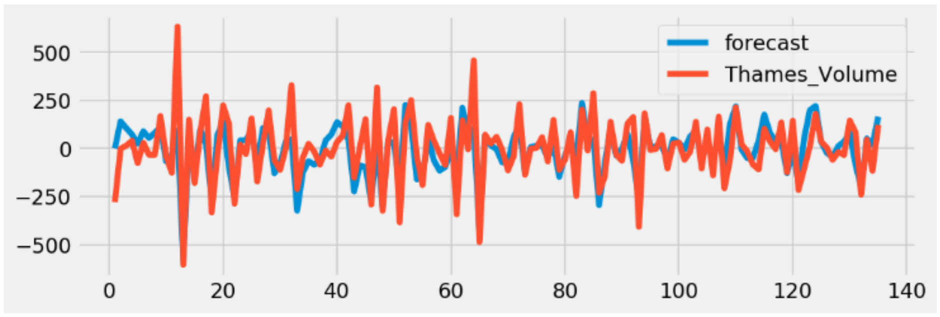
Mean of residuals:

```
In [388]: residuals.mean()
Out[388]: 0    7.501245
          dtype: float64
```

The WHITENESS TEST on residuals for ARMA(1,3) model

The piece of code below returns two values, the first one is The Ljung-Box test statistic and the second one is the p value. The p value comes out to be 0.999 and since the p-value is significantly larger than 0.05, it is strong evidence for discrete white noise being a good fit to the residuals. Hence, the ARMA(1,3) model is a good fit.

```
In [390]: sm.stats.acorr_ljungbox(residuals, lags=[10])
Out[390]: (array([1.4886198]), array([0.99897032]))

In [367]: model.plot_predict(dynamic=False)
          plt.show()
```



We find that the residuals which we obtained from the ARMA(1,3) on differenced data are normally distributed. Also, they have a mean which is close to 0. Moreover, from the ACF graph we deduce that they are uncorrelated as the values fall within the confidence boundaries. Finally, from the residual plot we deduce that they have a constant variance.

Therefore, we do not need any transformation to achieve these properties and the model which was chosen is deemed suitable.

Discussion of intervention in data and forecast

From the final plot, we can deduce that the forecasting is correctly representing the data to a reasonable extent, overlapping at most of the points. There are a few abrupt changes which may be due to man-induced interventions such as dam construction, change in water quality, etc.