MACHINE LEARNING
REPORT 6

The jupyter notebook with the code is at the end of the report

I. INTRODUCTION

i. BACKGROUND & MAIN OBJECTIVE

I will work with a time series that I will randomly generate using the timesyth library. The Time series will be generated with a gaussian process. The GaussianProcess() method samples the time series with a specified covariance function. I will use the '*Matern*' kernel. After the generation of the time series I will proceed to check if the series generated is stationary or not. I will apply smoothing techniques and then I will process to implement different SARIMA models to find the best model for forecasting future data.

II. THE DATA

i. DESCRIPTION OF THE DATASET

The data is going to be a gaussian generated time series. The frequency for the time is going to be in days.

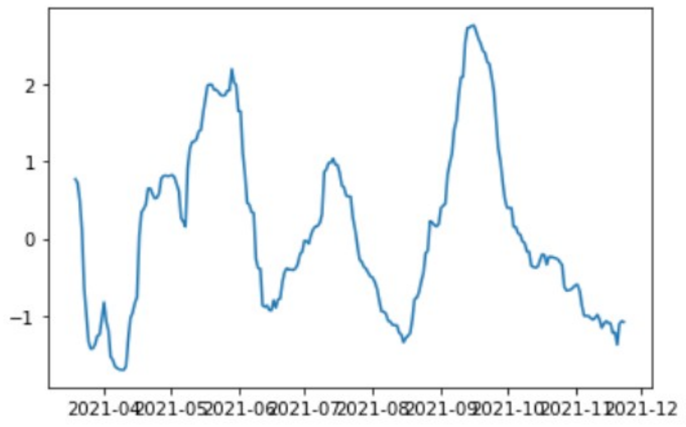


Figure 1. Gaussain Generated Time Serie

Observing the generated time series (*figure 1*) we see that both the mean and the variance should be somewhat constant. However, there might be also a bit of seasonality.

ii. EXPLORATORY ANALYSIS & FEATURES ENGINEERING

First I will check if the mean and the variance are constant throughout all the series by subdividing the series in a number of pieces and check the mean and variance on every chunk. I used the same process we used in the notebooks with the only difference that I defined a function to do it.

```
Chunk | Mean    | Variance
---------------------------
    1 | -1.0477 | 0.553035
    2 | 0.34249 | 0.334318
    3 | 1.647 | 0.1989
    4 | -0.24689 | 0.47596
    5 | 0.39059 | 0.212853
    6 | -0.6428 | 0.246298
    7 | 0.059812 | 0.770574
    8 | 1.8048 | 0.79818
    9 | -0.25943 | 0.0405714
   10 | -0.98886 | 0.0401443
```

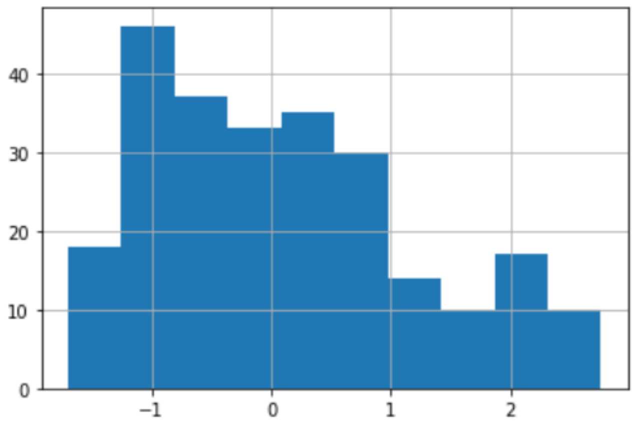


Figure 2, 3. Mean and Variance of every chunk of the original time serie before any transformation. And histogram of the distribution

We can see that the mean is not really constant and the variance has some spikes on chunks 3, 7 and 8. From the histogram the time series does not seem to have a normal distribution which tells us that the time series is probably non stationary. However, this numbers are not strongly indicating whatever the time series is sationary or not. Therefore I will implement the Augmented Dickey-Fuller (ADF) test to clear things up.

Augmented Dickey-Fuller on ORIGINAL TIME SERIES	
ADF	-2.82290
Pvalue	0.0550
UsedLag	4
nobs	245
icbest	-243.7025

Table 1. Augmented Dickey-Fuller test on the Original Time Seires

The ADF test on the time series shows that this series is non-stationary with a pvalue bigger then 0.05. However the ADF value is promising and, considering that the pvalue is actually pretty close to rejecting the null hypothesis I will try some simple transformations before going into decompositions, to see if I can get the time series stationary. I will apply a log, square-root and box-cox transformations.

Logarithmic

```
Chunk | Mean    | Variance
---------------------------
    1 | -0.29833 | 0.464716
    2 | 0.8076 | 0.104632
    3 | 1.285 | 0.019565
    4 | 0.49427 | 0.126007
    5 | 0.8522 | 0.0394434
    6 | 0.23952 | 0.132264
    7 | 0.61681 | 0.233685
    8 | 1.3042 | 0.0693046
    9 | 0.54727 | 0.0141927
   10 | -0.0077341 | 0.037097
```

Square Root

```
Chunk | Mean    | Variance
---------------------------
    1 | 0.91549 | 0.114198
    2 | 1.5155 | 0.0457643
    3 | 1.9057 | 0.0154406
    4 | 1.3014 | 0.0594563
    5 | 1.5388 | 0.0227496
    6 | 1.146 | 0.0437934
    7 | 1.3997 | 0.10056
    8 | 1.9356 | 0.058171
    9 | 1.317 | 0.00596618
   10 | 1.0008 | 0.00952857
```

Box-Cox

```
Chunk | Mean    | Variance
---------------------------
    1 | -0.19396 | 0.450292
    2 | 0.9799 | 0.162972
    3 | 1.6853 | 0.0489074
    4 | 0.57809 | 0.208266
    5 | 1.0261 | 0.0768003
    6 | 0.28059 | 0.164735
    7 | 0.75742 | 0.35737
    8 | 1.7338 | 0.181963
    9 | 0.61515 | 0.0214724
   10 | -0.00028702 | 0.0378279
```

Table 2. Log, Sqrt and Box-Cox transformations applied to the time series

Lets take a look at the distribution after the transformations.

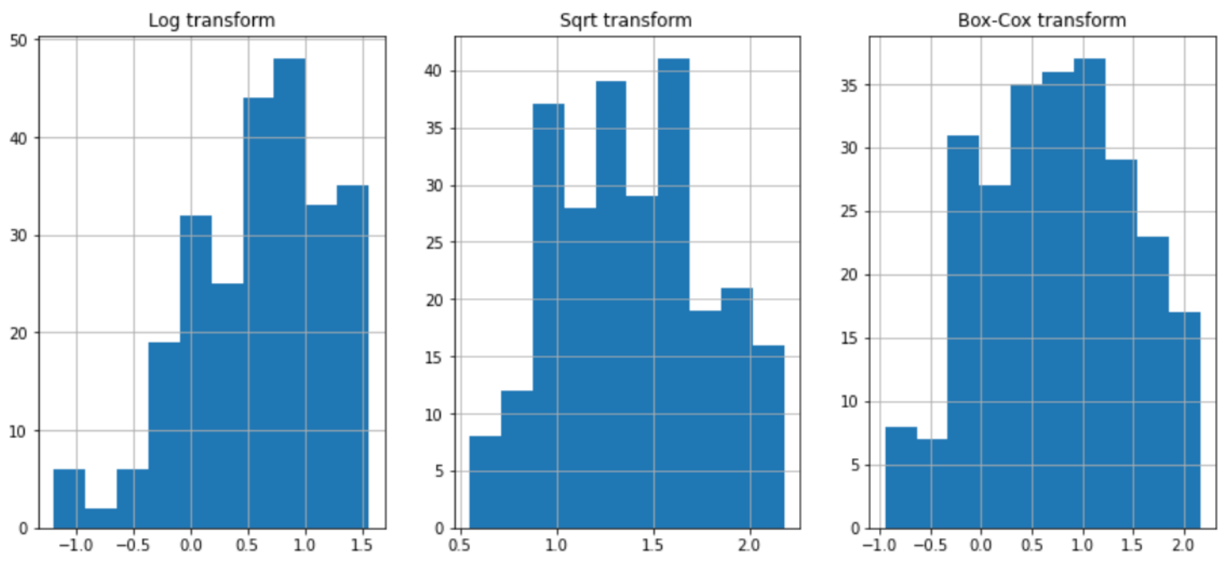


Figure 4. Histograms of the distribution of the time series after three different transformations. Logarithmic, Square root and Box-Cox

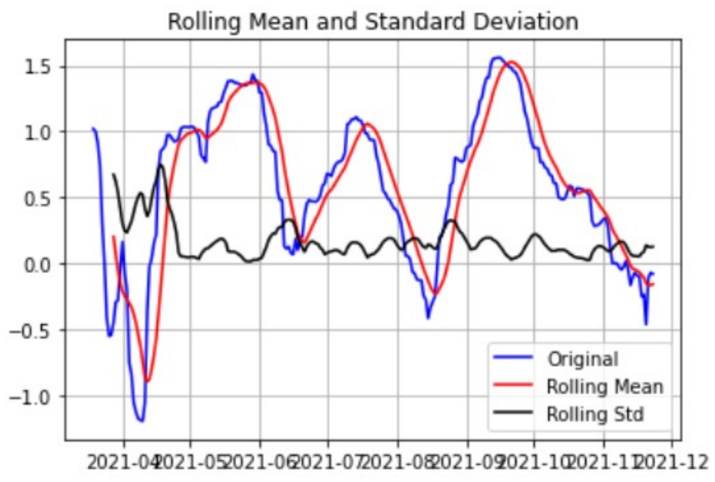
The data seems to be much more normally distributed after all three transformations. The transformations used are all generally used to get a constant variance but as a consequence they also usually reduce a little bit the fluctuations on the mean. To further understand the implication of these transformations on the times series I will proceed on implementing a Augmented Dickey-Fuller test there different transformed time series.

ADF TEST	Logarithmic	Square Root	Box-Cox
ADF	-2.9834	-2.9243	-2.9271
Pvalue	0.03648	0.04258	0.04423
UsedLag	5	5	5
nobs	244	244	244
icbest	-465.2128	-715.2553	-414.0024

Table 3. Results of the Augmented Dickey-Fuller test applied on the time series after the Log, Sqrt and Box-Cox transformations.

We can see from the ADF and pvalues that all three these transformation have made the series stationary. In fact the pvalue is lower the 0.05 in all cases. For further analysis and model implementation I will used the time series after logarithmic transformation since it is the one that clears the threshold with the biggest difference.

To be sure that this is a stationary series lest plot the log transformed time series with the rolling mean and standard deviations. If there is much change with time then I will mean that the series is not stationary after all.



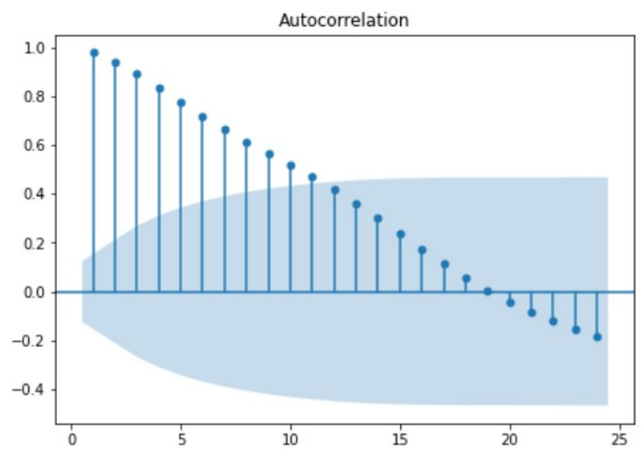
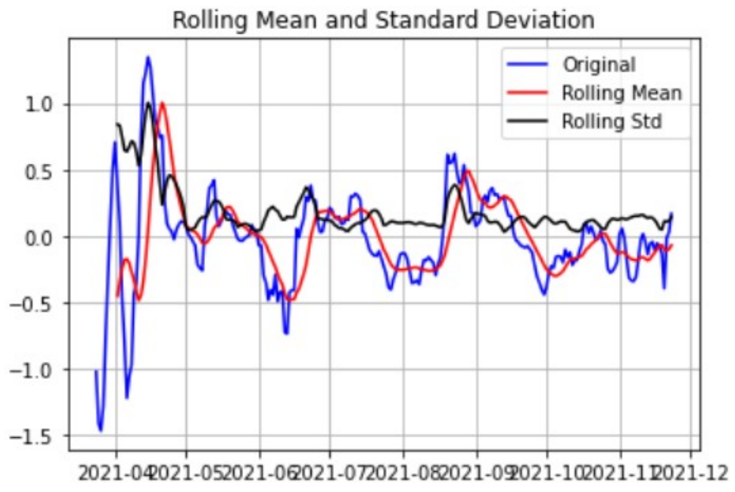


Figure 5. Rolling Means and Standard Deviation for the time series after the log transformation and the Autocorrelation function.

We clearly see that the rolling mean is not constant at all and that the autocorrelation function is decaying to zero very slowly. These are clear indicators that the time series is not stationary. What we can do in this case is to do a seasonal differencing with lag 5 and re-run the ADF test.

Augmented Dickey-Fuller on Log Transformed and seasonal differenced time series	
ADF	-4.8058
Pvalue	0.000053
UsedLag	12
nobs	232

Table 4. Results of the Augmented Dickey-Fuller test applied on the time series after the Log transformation and seasonal differencing



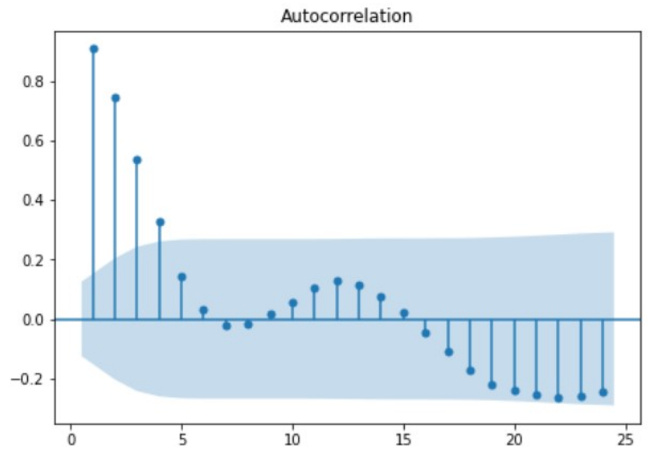


Figure 6. Rolling Means and Standard Deviation for the time series after the log transformation and seasonal differencing with lag 5. And the Autocorrelation function

After the seasonal differencing the time series looks much better and more stationary. The pvalue of the Augmented Dickey-Fuller test is very promising (0.00055). The Rolling mean and standard deviation are much more constant and the auto-correlation function decays to zero faster then before.

SMOOTHING MODELS

The gaussian process I used to generate the time series introduced in it white noise. Now I would like to see the forecast power of three different advanced smoothing techniques. The single, double and triple exponential smoothing. The time series generate has demonstrate a certain degree of trend and seasonality, therefore, single exponential smoothing should be the most under-preforming model since it can not pick up trend nor seasonality. The double exponential smoothing model should pick up the trend on the end of the training part but will fail in considering seasonality. Therefore, I'm expecting that the triple exponential smoothing will be the most suitable forecast models since it can pick up both trend and seasonality.

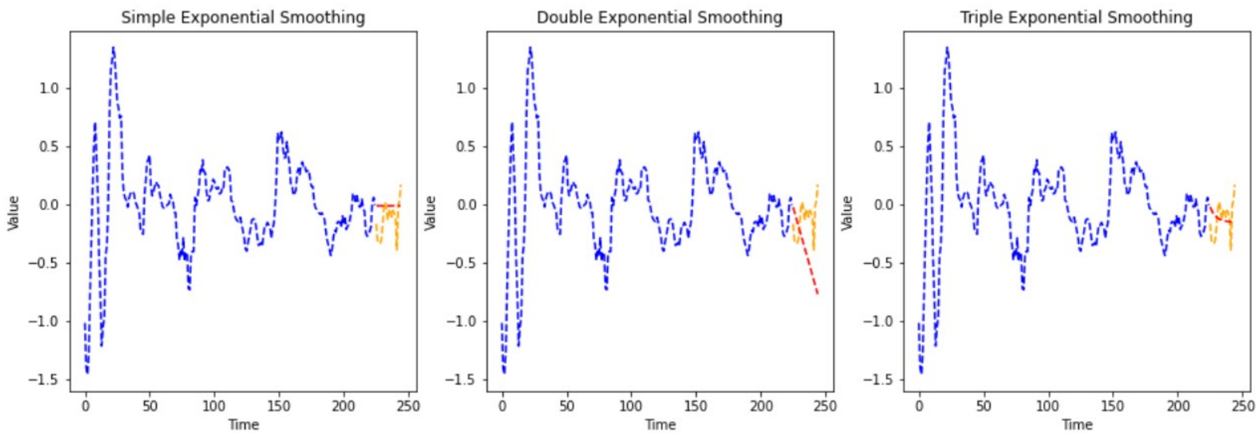


Figure 7. Plot of the time series after implementing, respectively, single, double and triple exponential smoothing. Blue: training_set, Yellow: validation, Red: forecast

As we can see the triple exponential smoothing is a far better model then the other two. The single exponential is giving a flat line and the double exponential smoothing is picking up the trend of the last part but fails to keep up with seasonality.

SARIMA MODELS

Now that we have a stationary series I will try to build the best model to forecast future values. For starters we must understand the orders of the Autoregressive and Moving Averages models. In this case we should take a look at the Autocorrelation and Partial-Autocorrelation functions.

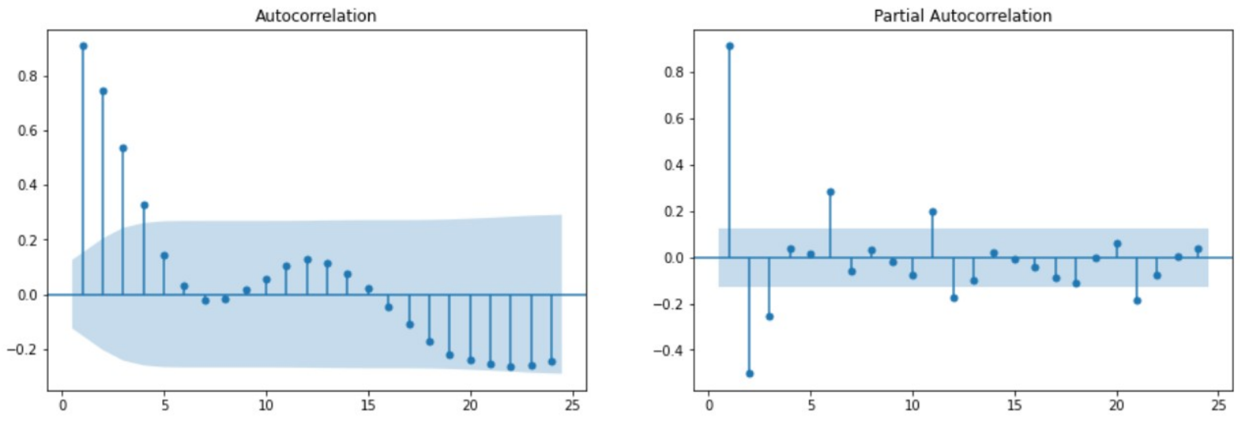


Figure 8. Plot of the autocorrelation and partial-autocorrelation functions

The autocorrelation function is decaying to zero and the partial-autocorralation indicates a order of 3 and a q=1. So maybe a AR(3) model. I will proceed in building different models starting from a very simple and crude model so to have a baseline.

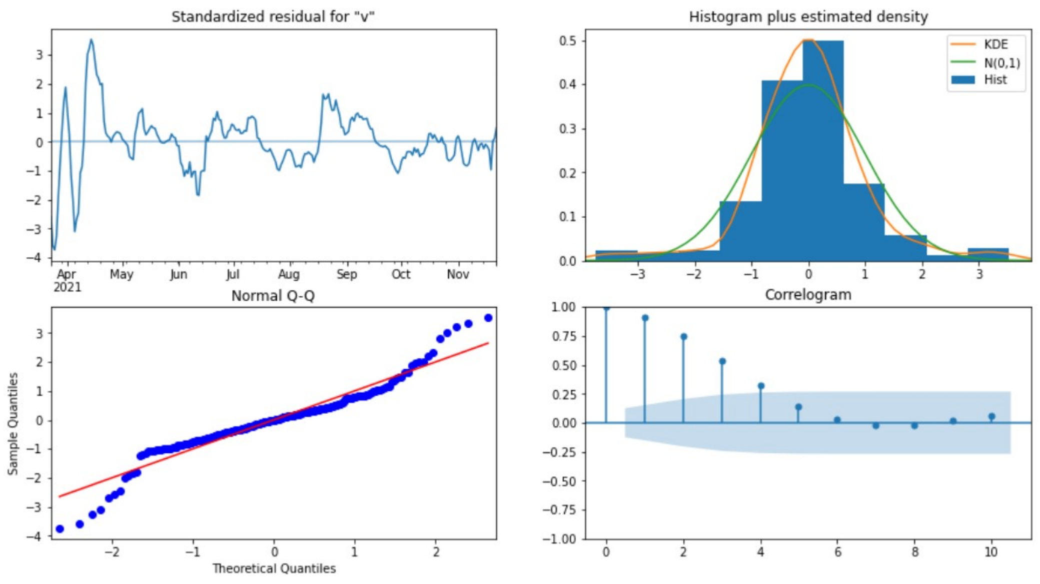
MODEL 1 - Baseline - (0,0,0)(0,1,0)[5]	
ORDER (0,0,0)	**SEASONAL ORDER (0,1,0) [5]**
	

Table 5. Plot diagnostic of the first baseline model

This first model is a very under-performing model. There is no AR nor MA component. As we can see from the diagnostic in the upper left corner the residual are far from random noise, the normality on the bottom right plot is not perfect and the Correlogram shows strong correlation after lag 0.

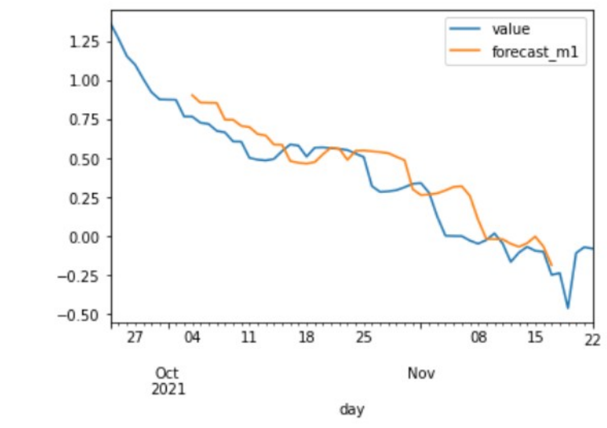
		Forecast - Model 1
AIC	233.574	
BIC	240.557	
Heteroskedasticity	0.12	
Mape	0.4009	

Table 6. Forecast plot of model 1 and SARIMAX results

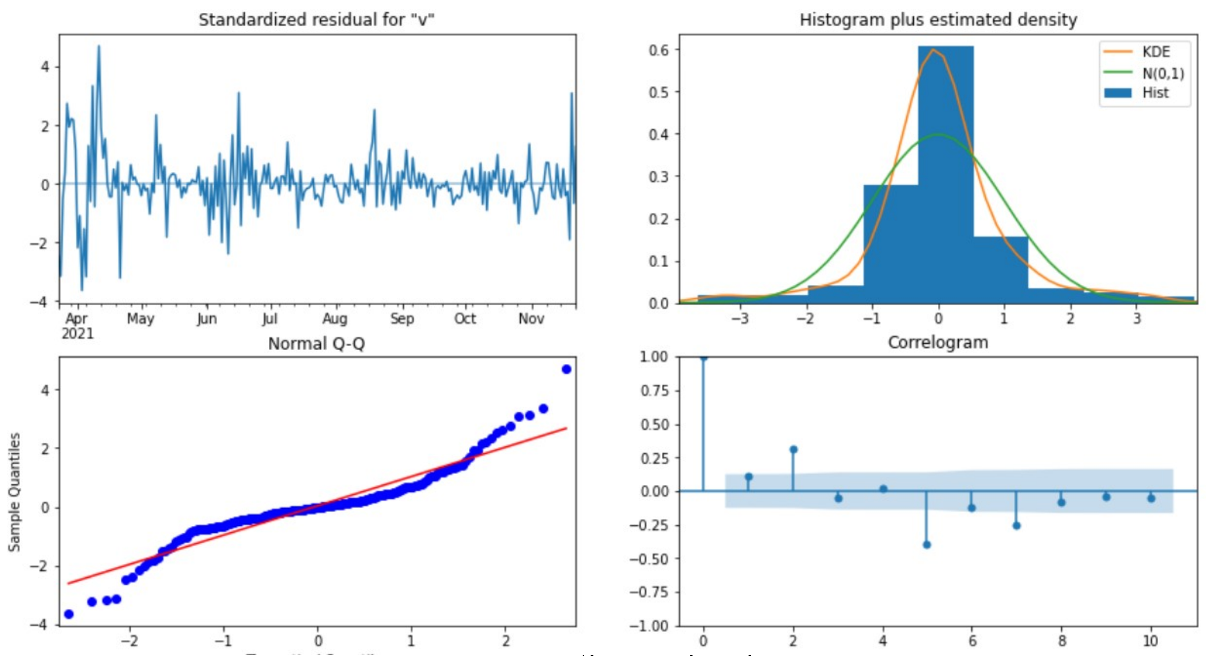
MODEL 2 - (1,0,1)(0,1,0)[5]	
ORDER (1,0,1)	**SEASONAL ORDER (0,1,0) [5]**
	

Table 7. Diagnostic Plot of the second model

This second model is a AR(1), MA(1) model with single seasonal differencing and a seasonality of 5. As we can see from the diagnostic this is a far better model the the baseline one. The plot of the standardized residual is much more like white noise and the correlogram shows the default correlation of 1 at lag 0 and then gets very close to zero for all the subsequent lags. There is still some problem with the distribution which is not great if we take a look at the bottom left plot. Even though it is not perfect it is a much better model the the first one.

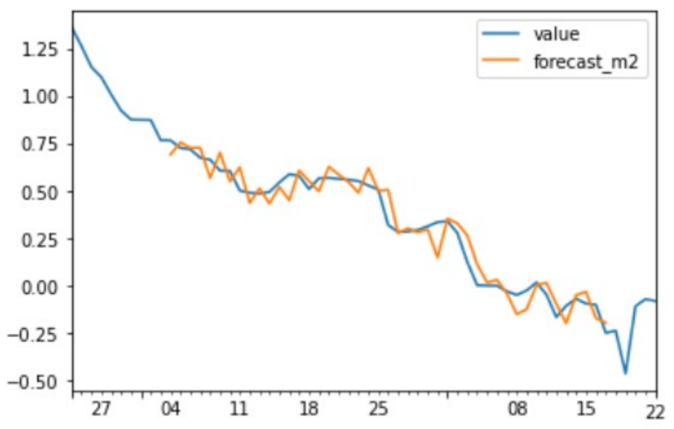
		Forecast – Model 2
AIC	-267.412	
BIC	-253.407	
Heteroskedasticity	0.21	
Mape	0.3099	

Table 8. Forecast plot of model 2 and SARIMAX results

As we can see this model fits the time series much better and has a massively lower AIC and BIC compared to the baseline model. Also the Mape is lower indicating that this is a better model.

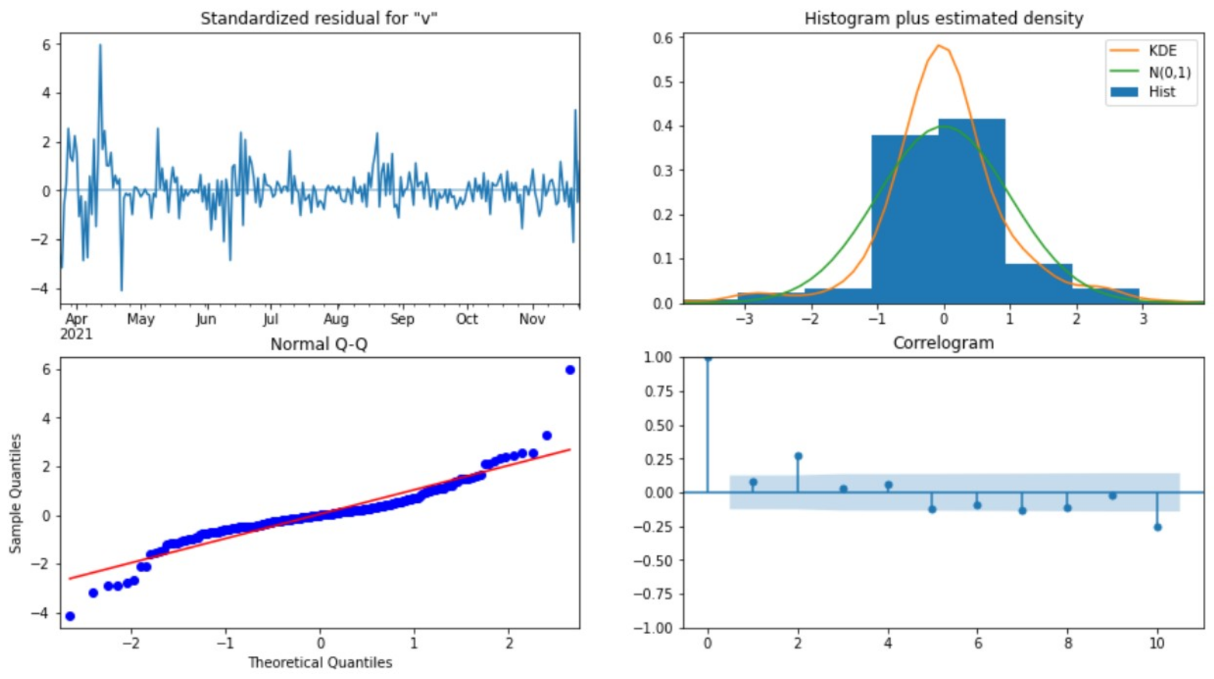
MODEL 3 - (1,0,1)(1,1,0)[5]	
ORDER (1,0,1)	**SEASONAL ORDER (1,1,0) [5]**
	

Table 9. Diagnostic Plot of the third model

This third model is pretty close to model2, with maybe a slightly better normalization.

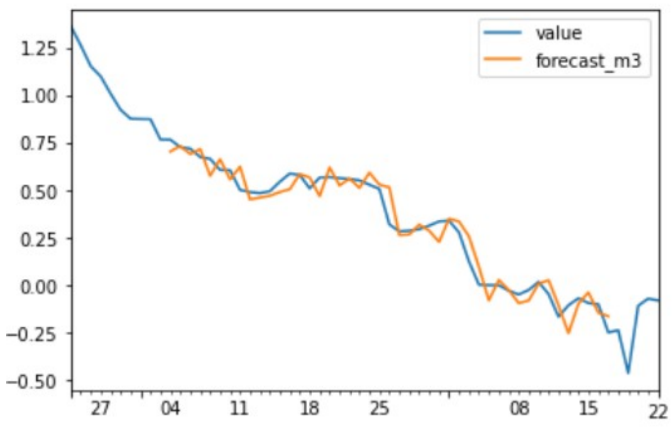
		Forecast – Model 3
AIC	-312.089	
BIC	-294.583	
Heteroskedasticity	0.23	
Mape	0.2842	

Table 10. Forecast plot of model 3 and SARIMAX results

All considered the model 3 is better the model2, it has a lower AIC, BIC and Mape. I wonder if I add a order 1 on the seasonal MA model what would happen.

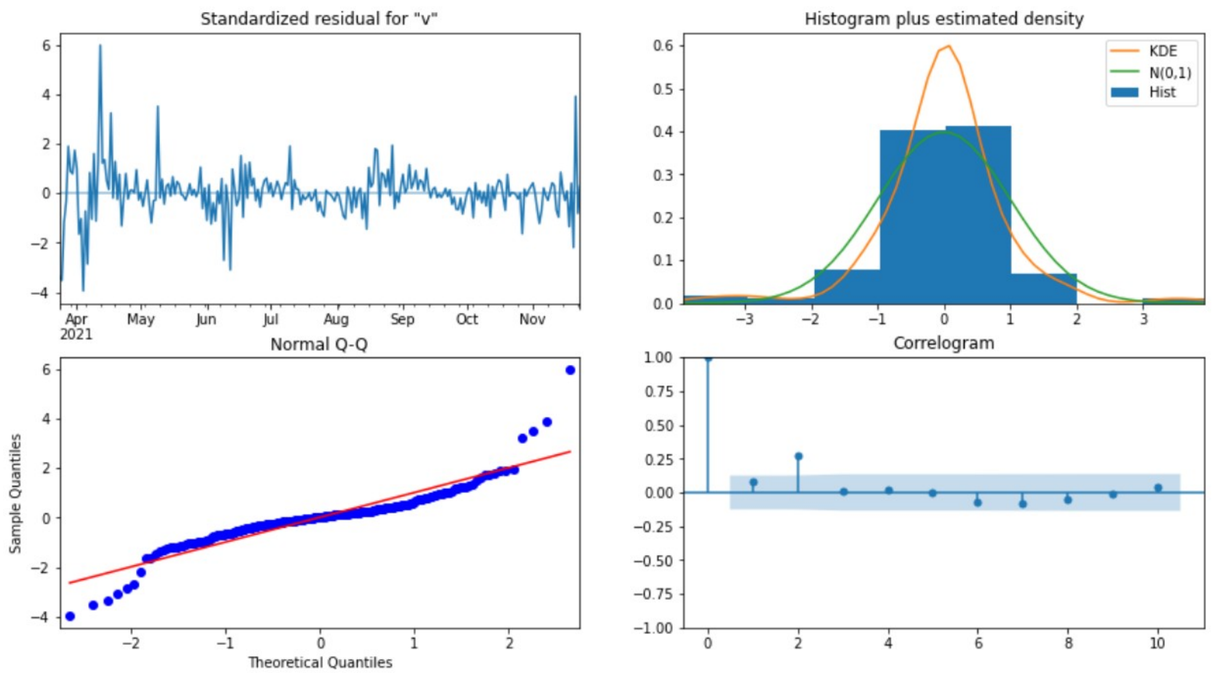
MODEL 4 - (1,0,1)(1,1,1)[5]	
ORDER (1,0,1)	**SEASONAL ORDER (1,1,1) [5]**
	

Table 11. Diagnostic Plot of the fourth model

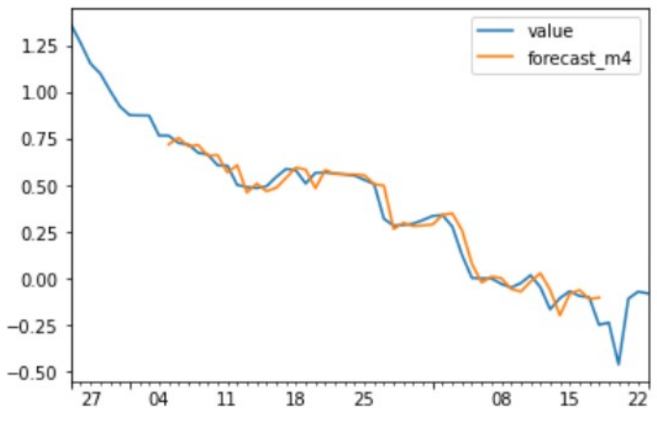
		Forecast – Model 4
AIC	-368.265	
BIC	-347.257	
Heteroskedasticity	0.25	
Mape	0.2972	

Table 12. Forecast plot of model 4 and SARIMAX results

This is interesting. The model 4 looks like a better forecast then all previous models, however it shows a slightly higher mape but lower AIC and BIC.

Now, in the start of the report I said that the best order for the AR model should be 3. I will test it with this following model.

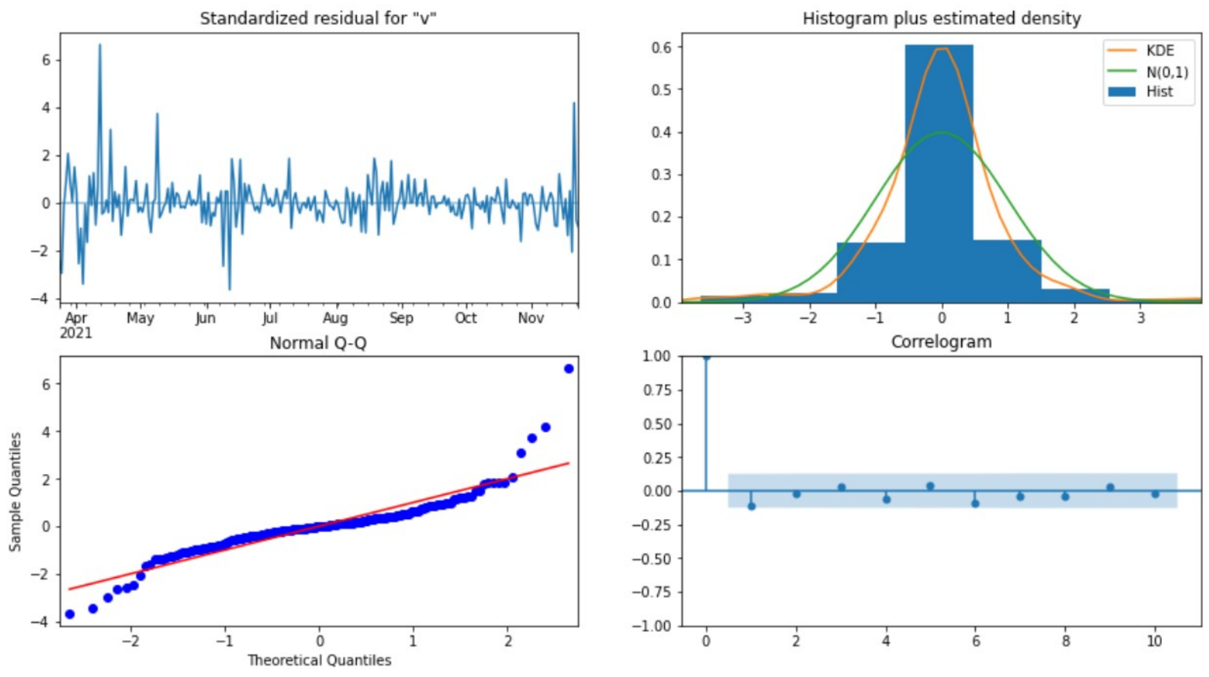
MODEL 5 - (3,0,1)(1,1,1)[5]	
ORDER (3,0,1)	**SEASONAL ORDER (1,1,1) [5]**
	

Table 13. Diagnostic Plot of the fifth model

Using a AR(3) feels like we get a better result. Especially considering the Correlogaram which for the first time shows all non significant correlations.

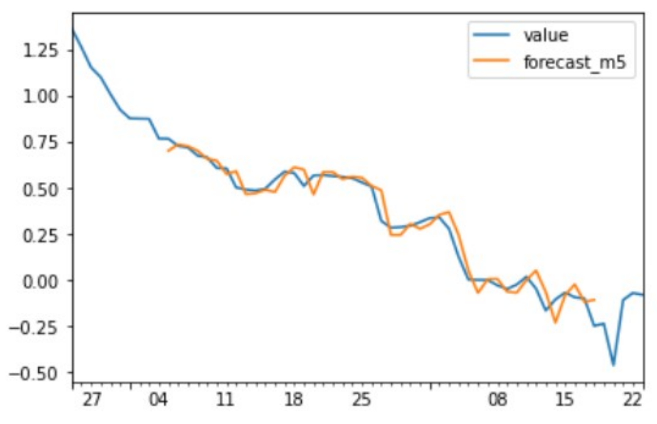
		Forecast – Model 5
AIC	-389.066	
BIC	-361.056	
Heteroskedasticity	0.29	
Mape	0.3235	

Table 14. Forecast plot of model 5 and SARIMAX results

The forecast does not seem much better, however the numbers shows that this is a better model with a lower AIC and BIC.

Lets see what order and seasonal order will the auto model find as the best one.

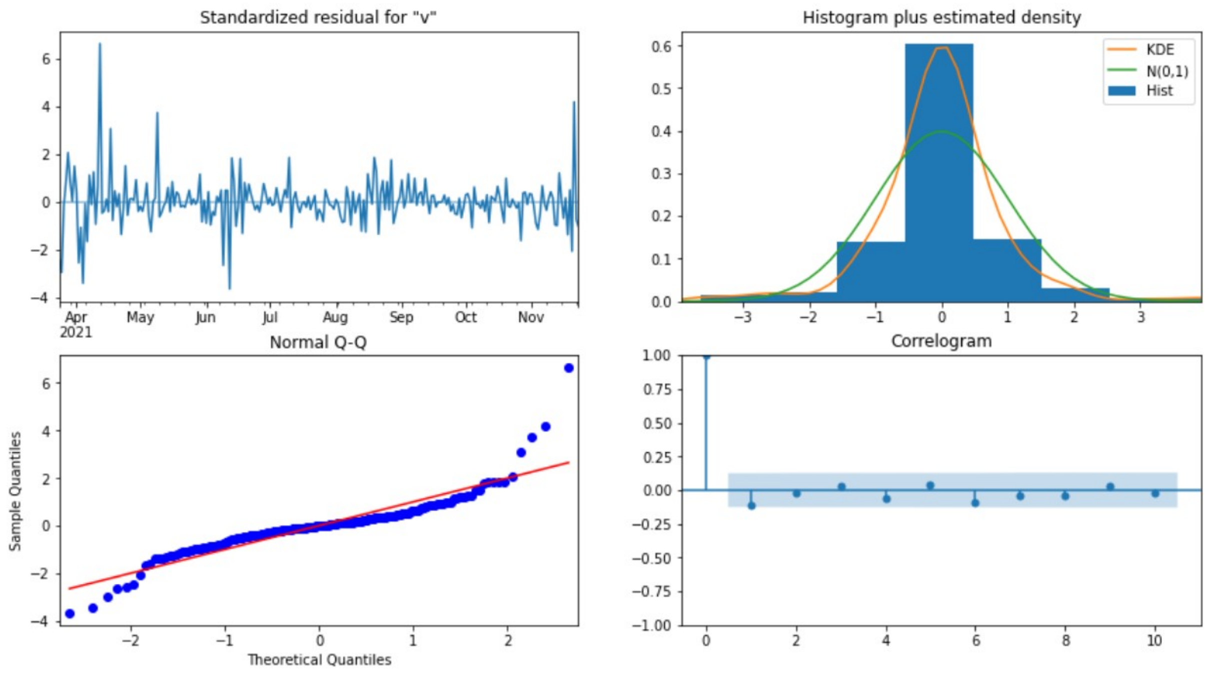
MODEL AUTO - (3,0,1)(1,1,2)[5]	
ORDER (3,0,1)	**SEASONAL ORDER (1,1,2) [5]**
	

Table 15. Diagnostic Plot of the auto model

The auto model finds order (3,0,1) and seasonal order (1,1,2)[5] as the best SARIMA model.

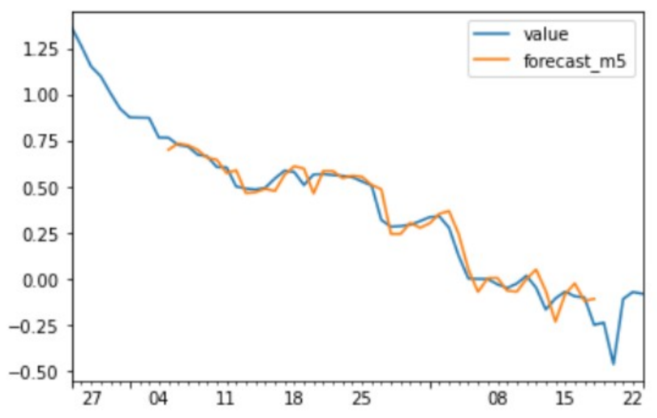
		Forecast – Auto Model
AIC	-389.979	
BIC	-358.468	
Heteroskedasticity	0.29	
Mape	0.3096	

Table 16. Forecast plot of the auto model and SARIMAX results

SUMMARY & KEY FINDINGS

In this project I generated a time series from scratch, using the timesynth library. The timeseries has been generated with the gaussian process. Initially this series was not stationary so I performed a few transformations. In particular the log transformation to get a more constant variance. To ensure that I were working with a stationary series I proceeded to implement the ADF test which, after the transformations, rejected the null indicating that the series was stationary. However, since the ADF test is not always right I decided to visualize in a plot the rolling mean, the rolling standard deviation and the autocorrelation function. At first, the rolling mean was highly dependant on time and the autocorrelation function was decaying to zero very slowly. Both these factors indicated that the series was not stationary after all, despite what the ADF test said. Therefore I proceeded in re-plotting the rolling mean, and standard deviation introducing seasonal differencing. This gave us a stationary series and a much better pvalue in the ADF test. Hence, our model should be comprehensive of seasonal differencing. Subsequently, I decided to test some smoothing models. I tried the advanced smoothing models such as single, double and triple exponential. As expected, the best model resulted to be the triple exponential smoothing. Because it is the only one that can pick-up both trend and seasonality. In the last part of the project I tried to find the best SARIMA model to forecast the time series. Looking at the autocorrelation and partial-autocorrelation plots I deducted that the best model should have p=3 and q=1 (and seasonal differencing from before). However, I started with a very simple model as to have a baseline. Afterwards I build other 5 models. All models are reported in *Table 17* below. We see that as we increase the orders and seasonal orders the model tend to get better. It is interesting to see that the heteroskedasticity is increasing as the model perform better. The results shows that despite the little increase in the mape value from model 3 to 5 the best model is the auto-model (3,0,1)(1,1,2)[5] which has the lowest AIC and BIC of all models maintaining a low mape value.

MODEL	AIC	BIC	Heteroskedasticity	Mape
Baseline - (0,0,0)(0,1,0)[5]	233.574	240.557	0.12	0.4009
Model 2 - (1,0,1)(0,1,0)[5]	-267.412	-253.407	0.21	0.3099
Model 3 - (1,0,1)(1,1,0)[5]	-312.089	-294.583	0.23	0.2842
Model 4 - (1,0,1)(1,1,1)[5]	-368.265	-347.257	0.25	0.2972
Model 5 - (3,0,1)(1,1,1)[5]	-389.066	-361.056	0.29	0.3235
Auto Model - (3,0,1)(1,1,2)[5]	-389.979	-358.468	0.29	0.3096

Table 17. Summary of all the built SARIMAX models

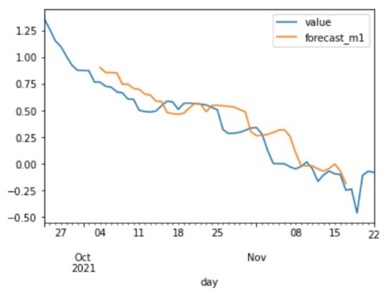
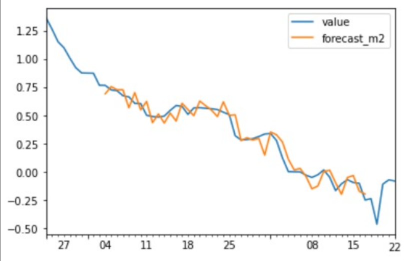
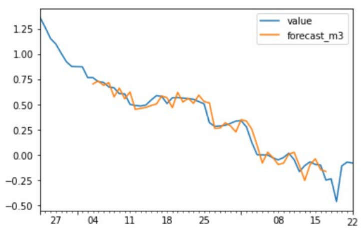
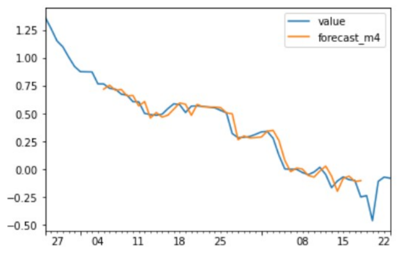
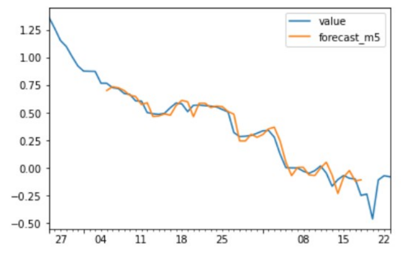
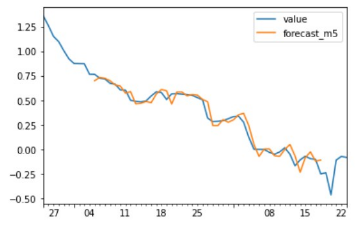
BaseLine Model	Model 2	Model 3
		
Model 4	**Model 5**	**Model 6**
		

Table 18. Summary of all the models forecasts

SUGGESTIONS

For this project I generated a synthetic time series which is probably not very interesting to work with. However, my aim was not to solve a real world problem but to experiment and play around with time series and hopefully learning something. Therefore I don't really have any other suggestions besides keeping on experimenting with different time series, apply different transformations, different models etc.

final_course6

March 19, 2021

1 Machine Learning Final Exercise, Course 6 - Lab-Diary

1.0.1 Background

I will work with a time series that has been generated by an algorithm. This means that this series does not refer to any real data. I will use this time series as an exercise to tests the concept discussed during the lectures.To generate the serie I will use the temesynth library. I will generate a time series which shows seasonality.

```
[51]: '''Importing the libraries'''

import pandas as pd
from datetime import datetime
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import timesynth as tsy
from datetime import timedelta
from dateutil.relativedelta import relativedelta
from IPython.display import display
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.api import SimpleExpSmoothing, Holt, ExponentialSmoothing
import statsmodels.api as sm
import statsmodels.tsa.stattools as ts
import pmdarima as pm
from scipy import stats
import warnings

%matplotlib inline
```

1.1 Helper functions

```
[52]: def chunks(residual, indices_or_sections):
          chunks = np.split(residual, indices_or_sections)
          print("{} | {:6} | {}".format("Chunk", "Mean", "Variance"))
```

```
    print("-" * 26)
    for i, chunk in enumerate(chunks, 1):
        print("{:5} | {:.5} | {:.6}".format(i, np.mean(chunk), np.var(chunk)))
```

[53]:
```
#Making use of the notebook function
def dftest(timeseries):
    dftest = ts.adfuller(timeseries,)
    dfoutput = pd.Series(dftest[0:4],
                         index=['Test Statistic','p-value','Lags␣
↪Used','Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)

    rolmean = timeseries.rolling(window=10).mean()
    rolstd = timeseries.rolling(window=10).std()

    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean and Standard Deviation')
    plt.grid()
    plt.show(block=False)
```

[54]:
```
def cross_validate(series,horizon,start,step_size,order =␣
↪(1,0,0),seasonal_order = (0,0,0,0),trend=None):
    '''
    Function to determine in and out of sample testing of arima model

    arguments
    ---------
    series (seris): time series input
    horizon (int): how far in advance forecast is needed
    start (int): starting location in series
    step_size (int): how often to recalculate forecast
    order (tuple): (p,d,q) order of the model
    seasonal_order (tuple): (P,D,Q,s) seasonal order of model

    Returns
    -------
    DataFrame: gives fcst and actuals with date of prediction
    '''
    fcst = []
    actual = []
    date = []
    for i in range(start,len(series)-horizon,step_size):
        model = sm.tsa.statespace.SARIMAX(series[:i+1], #only using data␣
↪through to and including start
                                order=order,
                                seasonal_order=seasonal_order,
                                trend=trend).fit()
        fcst.append(model.forecast(steps = horizon)[-1]) #forecasting horizon␣
↪steps into the future
        actual.append(series[i+horizon]) # comparing that to actual value at␣
↪that point
        date.append(series.index[i+horizon]) # saving date of that value
    return pd.DataFrame({'fcst':fcst,'actual':actual},index=date)
```

[55]:
```
def plots(data, lags=None):
    layout = (1, 3)
    raw  = plt.subplot2grid(layout, (0, 0))
    acf  = plt.subplot2grid(layout, (0, 1))
    pacf = plt.subplot2grid(layout, (0, 2))

    raw.plot(data)
    sm.tsa.graphics.plot_acf(data, lags=lags, ax=acf, zero=False)
    sm.tsa.graphics.plot_pacf(data, lags=lags, ax=pacf, zero = False)
    sns.despine()
    plt.tight_layout()
```

[56]:
```
def mape(df_cv):
    return abs(df_cv.actual - df_cv.fcst).sum() / df_cv.actual.sum()
```

1.2 Generating a Synthetic Time serie

Generating a time serie with a Gaussian process signal generation.

[57]:
```
'''
Initialize TimeSampler.
This a class that determines when samples will be taken from noise and signal.
'''
time_sampler = tsy.TimeSampler(stop_time=20)

'''
sample_irregular_time() is a function that samples irregularly spaced time␣
↪using:
    - num_points (number of points in time series)
    - keep_percentage, which is the percentage of the data points that will be␣
↪retained in the irregular serie
'''
```

```
irregular_time_samples = time_sampler.sample_irregular_time(num_points=500,␣
 ↪keep_percentage=50)
```

[58]:
```
'''
Initialize TimeSampler.
This a class that determines when samples will be taken from noise and signal.
'''
gp = tsy.signals.GaussianProcess(kernel='Matern', nu=3./2)

'''
Initialize the TimeSampler object with gp as the signal generator
'''
gp_series = tsy.TimeSeries(signal_generator=gp)

'''
Sample the time series
'''
samples = gp_series.sample(irregular_time_samples)[0]
```

[59]:
```
#plt.plot(irregular_time_samples, samples, marker='o', markersize=4)
#plt.xlabel('Time')
#plt.ylabel('Value')
#plt.title('Gaussian Process signal with Matern 3/2-kernel');
```

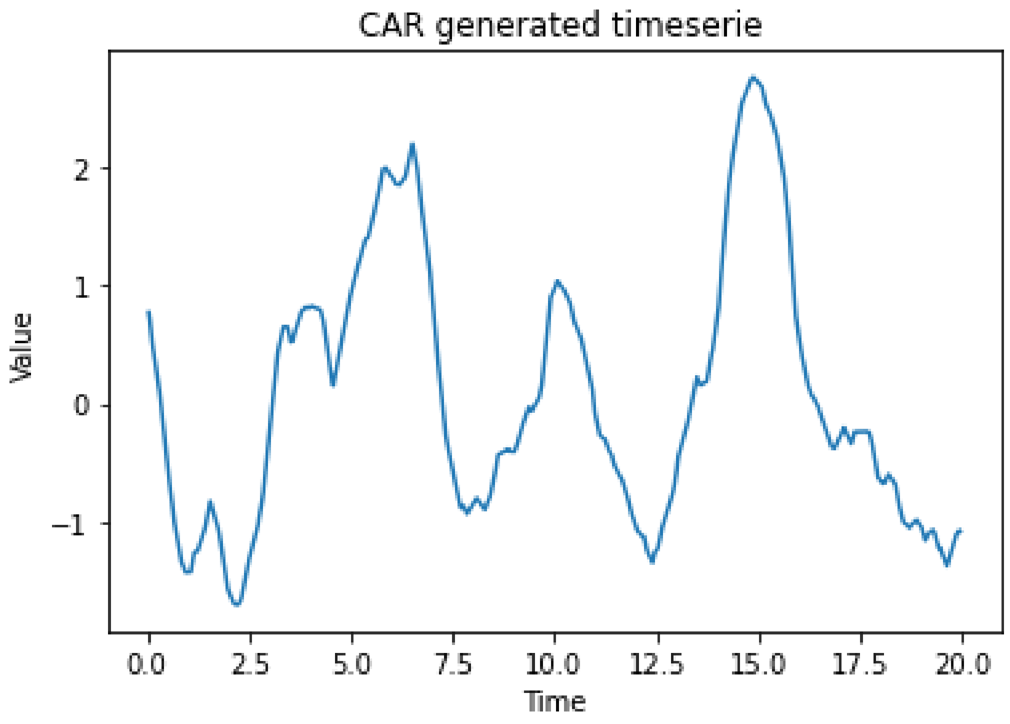
[60]:
```
'''
Saving the autogenerated time serie and commenting out to avoit overwriting the␣
 ↪files
'''
#samples.tofile('timeS_bak.dat')
#irregular_time_samples.tofile('irregular_time_samples_bak.dat')
```

[60]:
```
'\nSaving the autogenerated time serie and commenting out to avoit overwriting
the files\n'
```

[61]:
```
timeS_bak = np.fromfile('timeS_bak.dat', dtype=float)
irregular_time_samples_bak = np.fromfile('irregular_time_samples_bak.dat',␣
 ↪dtype=float)
```

[62]:
```
plt.plot(irregular_time_samples_bak, timeS_bak, markersize=4)
plt.xlabel('Time')
plt.ylabel('Value')
plt.title(' CAR generated timeserie')
```

[62]:
```
Text(0.5, 1.0, ' CAR generated timeserie')
```

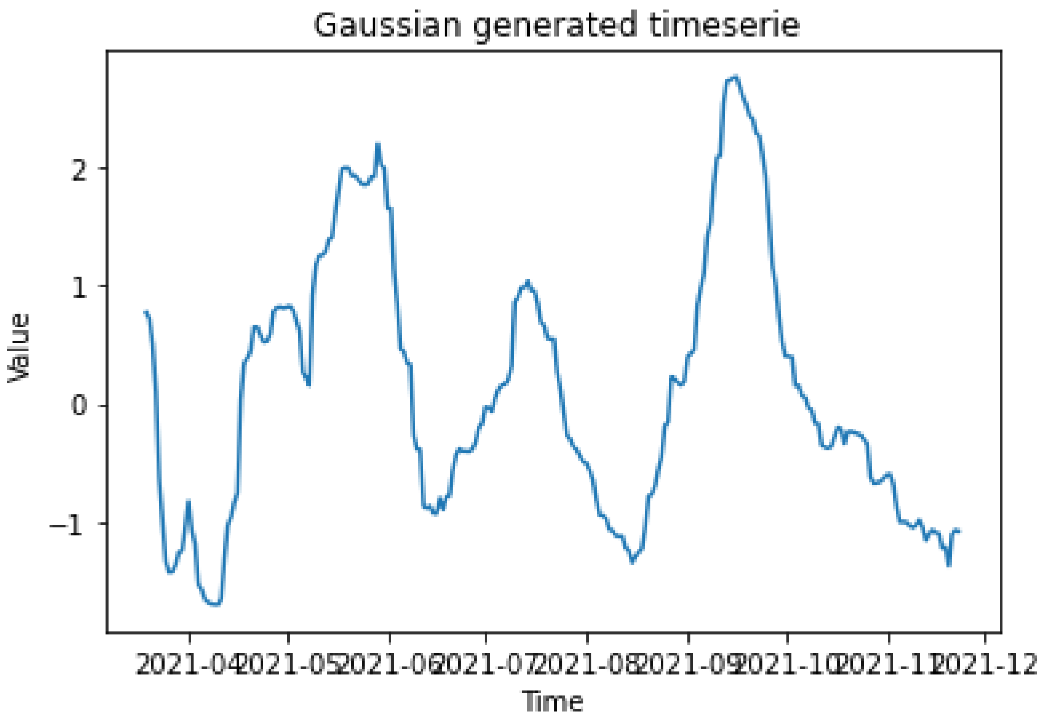


The time with integers is not really interesting. Lets modify it by creating a list of dates with frequency in Days and substitute it to the plain integers.

```
[63]: datelist = pd.date_range(datetime.today(), periods=250).tolist()
      datelist = np.array(datelist, dtype='datetime64[D]')
```

```
[64]: plt.plot(datelist, timeS_bak, markersize=4)
      plt.xlabel('Time')
      plt.ylabel('Value')
      plt.title(' Gaussian generated timeserie')
```

```
[64]: Text(0.5, 1.0, ' Gaussian generated timeserie')
```



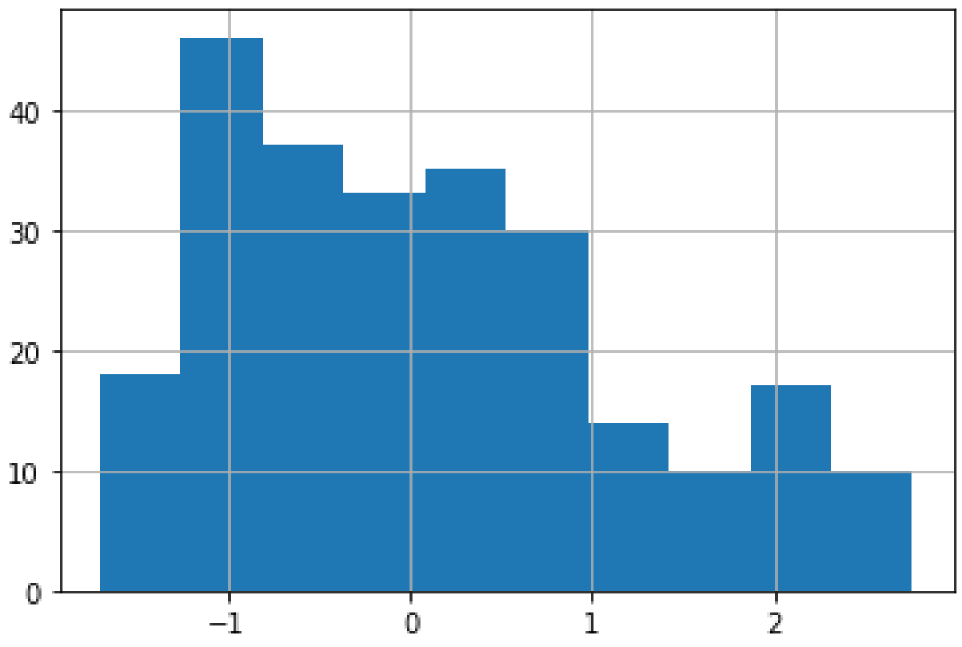
At first glance this seire looks like it has some autocorrelation but no seasonality going on. Lets start to analyse if this time serie is stationary or not.

```
[65]: chunks(timeS_bak, 10)
```

```
Chunk | Mean    | Variance
--------------------------
    1 | -1.0477 | 0.553035
    2 | 0.34249 | 0.334318
    3 | 1.647 | 0.1989
    4 | -0.24689 | 0.47596
    5 | 0.39059 | 0.212853
    6 | -0.6428 | 0.246298
    7 | 0.059812 | 0.770574
    8 | 1.8048 | 0.79818
    9 | -0.25943 | 0.0405714
   10 | -0.98886 | 0.0401443
```

```
[66]: pd.Series(timeS_bak).hist()
```

```
[66]: <AxesSubplot:>
```



[67]:
```
adf, pvalue, usedlag, nobs, critical_values, icbest = adfuller(timeS_bak)
print('adf:', adf, '| pvalue:', pvalue, '| usedlag:', usedlag, '| nobs:', nobs,␣
 ↪'| icbest:', icbest)
```

```
adf: -2.822904819182101 | pvalue: 0.05507573017923365 | usedlag: 4 | nobs: 245 |
icbest: -243.7024856913365
```

[68]:
```
print('critical_values:', critical_values)
```

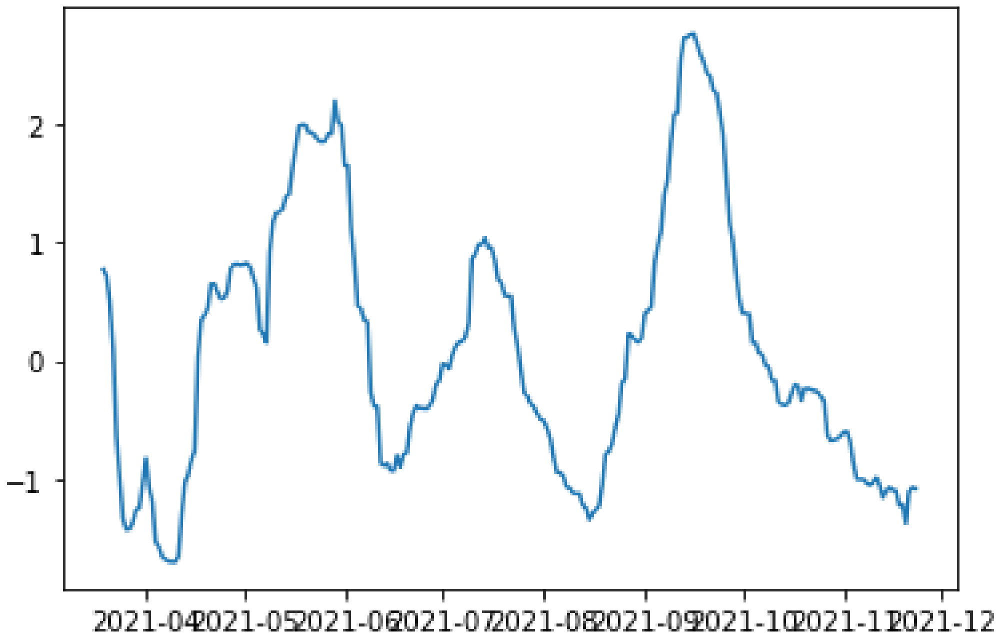
```
critical_values: {'1%': -3.4573260719088132, '5%': -2.873410402808354, '10%':
-2.573095980841316}
```

Alright, now I am pretty sure this time serie is non-stationary.

---

[69]:
```
plt.plot(datelist, timeS_bak, markersize=4)
```

[69]:
```
[<matplotlib.lines.Line2D at 0x7f0be1467278>]
```



The mean seems to be varying while the variance looks pretty constant.

Lets try some transformations to see if we get a better constant mean

1.3 Log Transformation

[70]:
```
timeS_bak.min()
```

[70]:
```
-1.697396954259025
```

[71]:
```
timeS_bak_plus = timeS_bak + 2 # Ensuring all values are positive
timeS_bak_log = np.log(timeS_bak_plus)
```

[72]:
```
chunks(timeS_bak_log, 10)
```

```
Chunk | Mean    | Variance
--------------------------
    1 | -0.29833 | 0.464716
    2 | 0.8076 | 0.104632
    3 | 1.285 | 0.019565
    4 | 0.49427 | 0.126007
    5 | 0.8522 | 0.0394434
    6 | 0.23952 | 0.132264
    7 | 0.61681 | 0.233685
    8 | 1.3042 | 0.0693046
    9 | 0.54727 | 0.0141927
   10 | -0.0077341 | 0.037097
```

1.4 Square Root Transformation

[73]:
```
timeS_bak_sqrt = np.sqrt(timeS_bak_plus)
```

[74]:
```
chunks(timeS_bak_sqrt, 10)
```

```
Chunk | Mean    | Variance
--------------------------
    1 | 0.91549 | 0.114198
    2 | 1.5155 | 0.0457643
    3 | 1.9057 | 0.0154406
    4 | 1.3014 | 0.0594563
    5 | 1.5388 | 0.0227496
    6 | 1.146 | 0.0437934
    7 | 1.3997 | 0.10056
    8 | 1.9356 | 0.058171
    9 | 1.317 | 0.00596618
   10 | 1.0008 | 0.00952857
```

Way better, this seems to have done a great job ! Out of curiosity lets try to use a box-cox tranformation

1.5 Box-Cox Transformation

[75]:
```
timeS_bak_box_cox = stats.boxcox(timeS_bak_plus)
```

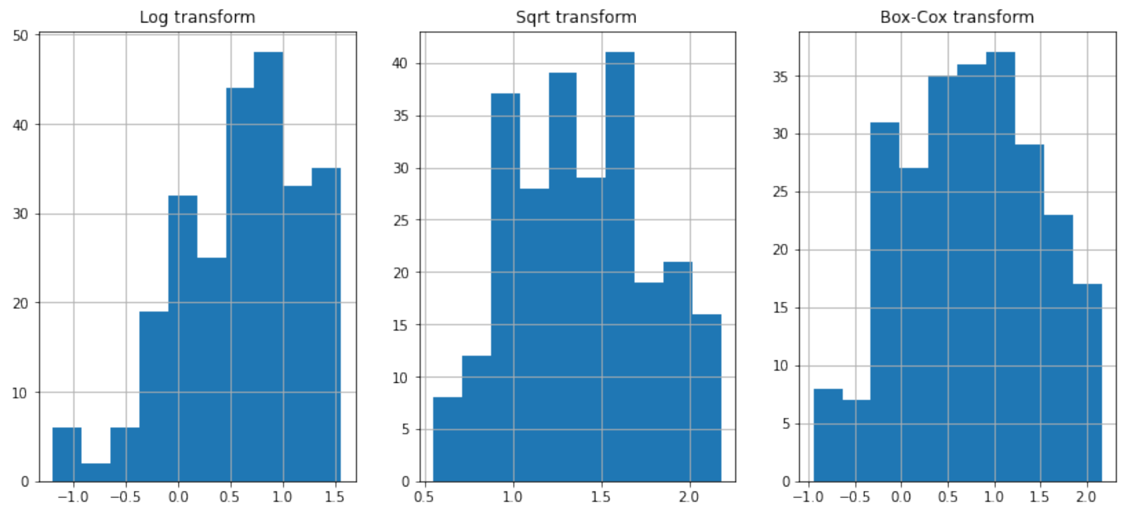
[76]:
```
chunks(timeS_bak_box_cox[0], 10)
```

```
Chunk | Mean    | Variance
--------------------------
    1 | -0.19396 | 0.450292
    2 | 0.9799 | 0.162972
    3 | 1.6853 | 0.0489074
    4 | 0.57809 | 0.208266
    5 | 1.0261 | 0.0768003
    6 | 0.28059 | 0.164735
    7 | 0.75742 | 0.35737
    8 | 1.7338 | 0.181963
    9 | 0.61515 | 0.0214724
   10 | -0.00028702 | 0.0378279
```

[77]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(14,6))
pd.Series(timeS_bak_log).hist(ax=ax1)
ax1.set(title='Log transform')
pd.Series(timeS_bak_sqrt).hist(ax=ax2)
```

```
ax2.set(title='Sqrt transform')
pd.Series(timeS_bak_box_cox[0]).hist(ax=ax3)
ax3.set(title='Box-Cox transform')
```

[77]: [Text(0.5, 1.0, 'Box-Cox transform')]



[78]:
```
adf, pvalue, usedlag, nobs, critical_values, icbest = adfuller(timeS_bak_log)
print('After Log transformation')
print('adf:', adf, '| pvalue:', pvalue, '| usedlag:', usedlag, '| nobs:', nobs,␣
 ↪'| icbest:', icbest)

print('--'*16)
print('After sqrt transformation')
adf, pvalue, usedlag, nobs, critical_values, icbest = adfuller(timeS_bak_sqrt)
print('adf:', adf, '| pvalue:', pvalue, '| usedlag:', usedlag, '| nobs:', nobs,␣
 ↪'| icbest:', icbest)

print('--'*16)
print('After box-cox tranformation')
adf, pvalue, usedlag, nobs, critical_values, icbest =␣
 ↪adfuller(timeS_bak_box_cox[0])
print('adf:', adf, '| pvalue:', pvalue, '| usedlag:', usedlag, '| nobs:', nobs,␣
 ↪'| icbest:', icbest)
```

```
After Log transformation
adf: -2.983357368101594 | pvalue: 0.03647811447157651 | usedlag: 5 | nobs: 244 |
icbest: -465.2127944358241
--------------------------------
After sqrt transformation
adf: -2.924392762507869 | pvalue: 0.04258409425359015 | usedlag: 5 | nobs: 244 |
icbest: -715.2553124114175
--------------------------------
After box-cox tranformation
adf: -2.9271647079974894 | pvalue: 0.042279078096742376 | usedlag: 5 | nobs: 244
| icbest: -414.0023661932315
```

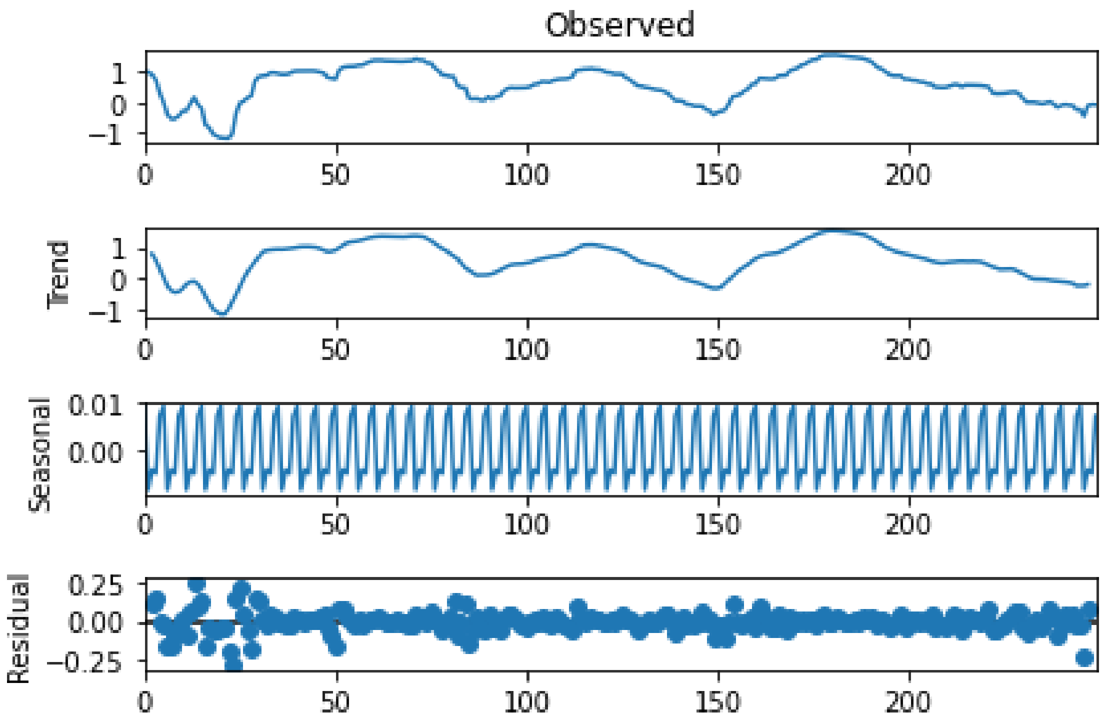
Stationary ? The adf is low and the pvalue is under 0.05 which by the means of the adf test means that this serie is now stationary. Lets take the log one

---

1.6 Plot the data after the log transformation - OLD

[84]:
```
#fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(14,5))
#ax1.plot(residual_log)
#ax2.plot(residual_sqrt)
#ax3.plot(residual_box_cox[0])
```

[85]:
```
seasonal_decompose(timeS_bak_log,period=5).plot();
```



---

[86]:
```
data = pd.DataFrame()
data['value'] = pd.Series(timeS_bak_log[:250])
#data['value'] = pd.Series(residual_log)
```

```
data['day'] = pd.Series(datelist[:250], index=None)
data=data.set_index('day')
data.head()
```

[86]:
```
                value
day
2021-03-19  1.018774
2021-03-20  0.999753
2021-03-21  0.913257
2021-03-22  0.739658
2021-03-23  0.306375
```

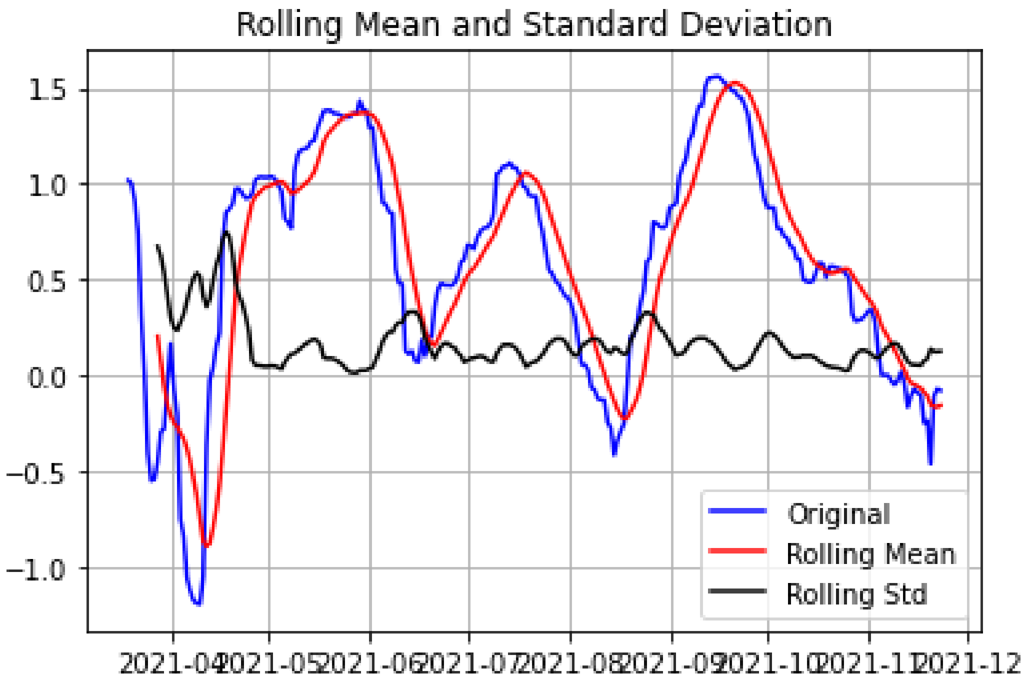
[87]:
```
data.tail()
```

[87]:
```
                value
day
2021-11-19 -0.237068
2021-11-20 -0.462210
2021-11-21 -0.108840
2021-11-22 -0.070945
2021-11-23 -0.080747
```

Now we have the stationary serie that we can feed to a model.

[90]:
```
dftest(data['value'].dropna())
```

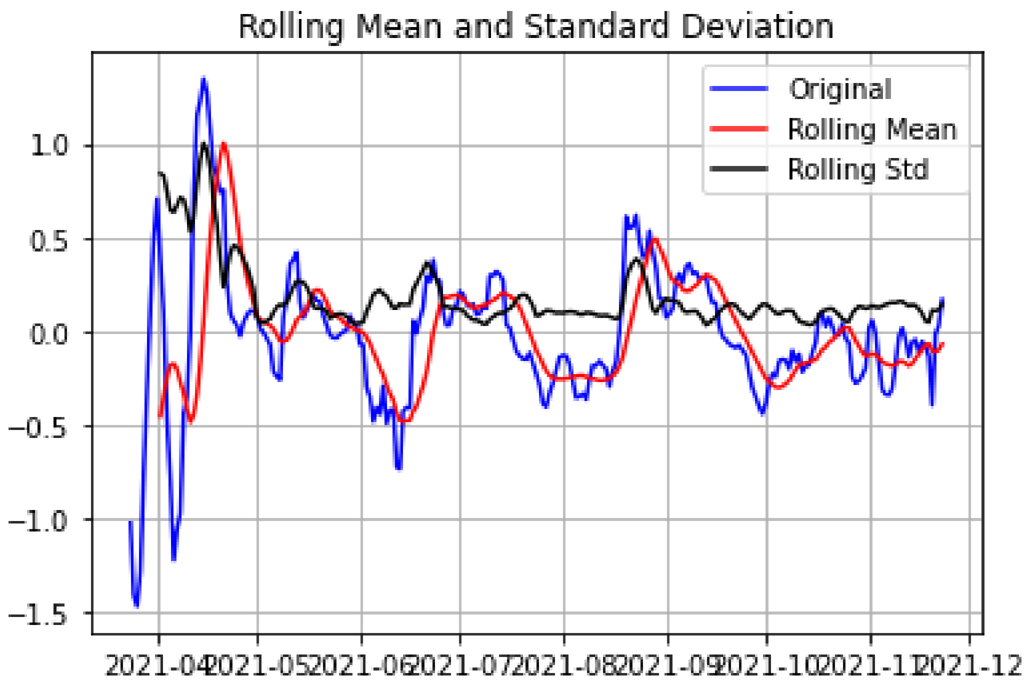
```
Test Statistic           -2.983357
p-value                   0.036478
Lags Used                 5.000000
Observations Used       244.000000
Critical Value (1%)      -3.457438
Critical Value (5%)      -2.873459
Critical Value (10%)     -2.573122
dtype: float64
```



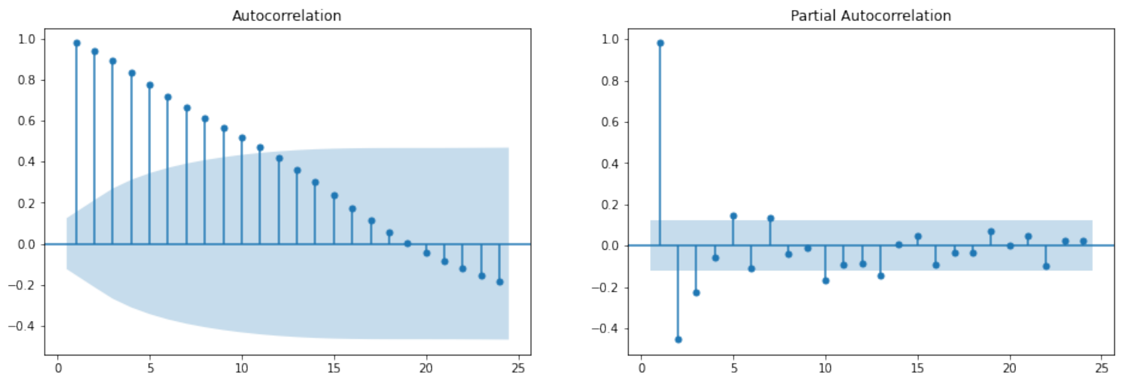
```
[88]: data['lag_5'] = data.value.shift(5)
      data['seasonal_diff'] = data.value - data['lag_5']
```

```
[89]: dftest(data['seasonal_diff'].dropna())
```

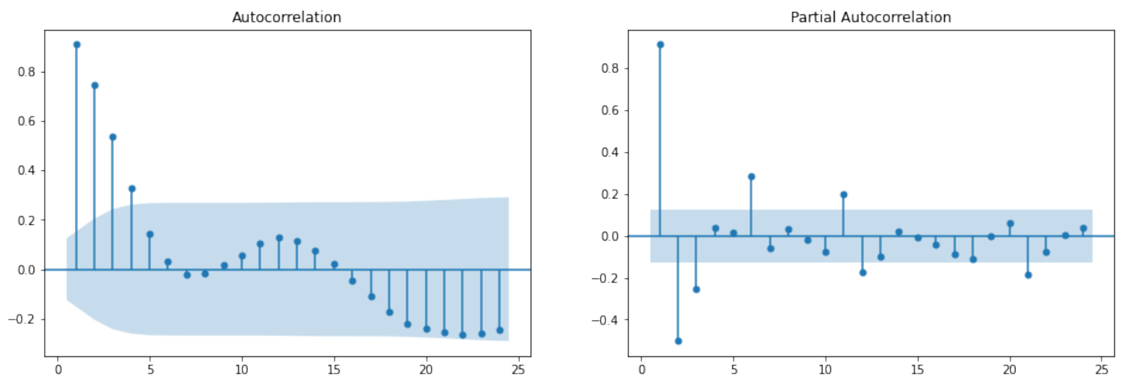
```
Test Statistic           -4.805765
p-value                   0.000053
Lags Used                12.000000
Observations Used       232.000000
Critical Value (1%)      -3.458855
Critical Value (5%)      -2.874080
Critical Value (10%)     -2.573453
dtype: float64
```



[131]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16,5))
sm.tsa.graphics.plot_acf(data.value,zero=False, ax=ax1)
sm.tsa.graphics.plot_pacf(data.value,zero=False, ax=ax2);
```



[132]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16,5))
sm.tsa.graphics.plot_acf(data['seasonal_diff'].dropna(),zero=False, ax=ax1)
sm.tsa.graphics.plot_pacf(data['seasonal_diff'].dropna(),zero=False, ax=ax2);
```



There is clearly a correlation from one effect to the next so I would guess this is a an ar(1) models or ma(1).

Lets try some smoothing

```
[91]: len(data['seasonal_diff'])
```

```
[91]: 250
```

```
[105]: seasonal_diff = data['seasonal_diff'].dropna()

       test_size = 20
       timeS_size = np.arange(len(seasonal_diff))

       train_set = seasonal_diff[:-test_size]
       test_set = seasonal_diff[-test_size:]

       print('residual size:', seasonal_diff.size, 'train set size:', train_set.size,␣
        ↪'test set size:', test_set.size)
```

```
residual size: 245 train set size: 225 test set size: 20
```

```
[107]: seasonal_diff.head()
```

```
[107]: day
       2021-03-24   -1.021163
       2021-03-25   -1.406501
       2021-03-26   -1.464623
       2021-03-27   -1.283497
       2021-03-28   -0.757425
       Name: seasonal_diff, dtype: float64
```

```
[108]: '''
```

```
Run the simple smoothing on the dataset and the double and triple exponential␣
↪smoothing.
'''
simple_smoothing = SimpleExpSmoothing(train_set).fit(optimized=True)
simple_prediction = simple_smoothing.forecast(len(test_set))
simple_mse = ((test_set-simple_prediction)**2).sum()/len(test_set)

double_smoothing = Holt(train_set).fit(optimized=True)
double_prediction = double_smoothing.forecast(len(test_set))
double_mse = ((test_set-double_prediction)**2).sum()/len(test_set)

triple_smoothing = ExponentialSmoothing(train_set,damped=True,
                              trend="additive",
                              seasonal=None,
                              seasonal_periods=None).fit(optimized=True)

triple_prediction = triple_smoothing.forecast(len(test_set))
triple_mse = ((test_set-triple_prediction)**2).sum()/len(test_set)
```

```
/home/Enry/.local/share/virtualenvs/jupyterlab-fEo2NtID/lib/python3.7/site-
packages/statsmodels/tsa/base/tsa_model.py:527: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  % freq, ValueWarning)
/home/Enry/.local/share/virtualenvs/jupyterlab-fEo2NtID/lib/python3.7/site-
packages/statsmodels/tsa/base/tsa_model.py:527: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  % freq, ValueWarning)
/home/Enry/.local/share/virtualenvs/jupyterlab-fEo2NtID/lib/python3.7/site-
packages/ipykernel_launcher.py:15: FutureWarning: the 'damped'' keyword is
deprecated, use 'damped_trend' instead
  from ipykernel import kernelapp as app
/home/Enry/.local/share/virtualenvs/jupyterlab-fEo2NtID/lib/python3.7/site-
packages/statsmodels/tsa/base/tsa_model.py:527: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  % freq, ValueWarning)
```

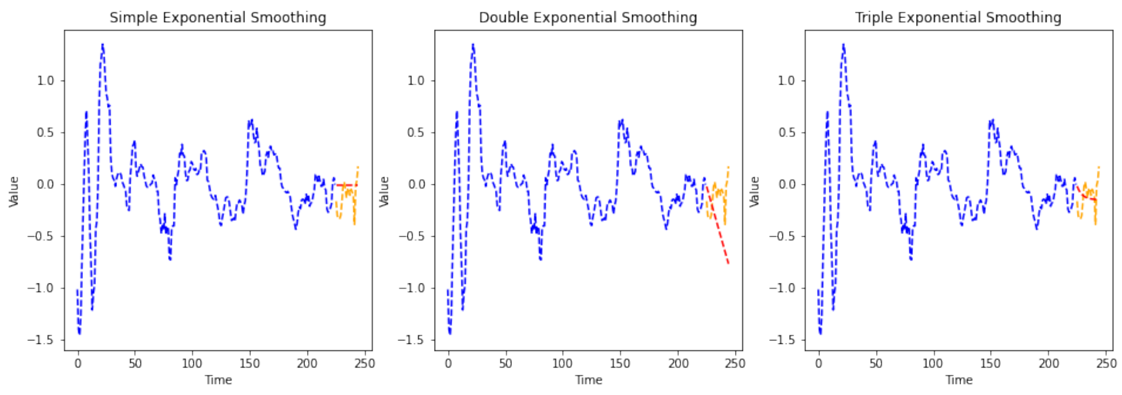
[109]:
```
print('MSE of: simple_smoothing:', simple_mse, 'double_smoothing:', double_mse,␣
↪'triple_smoothing:', triple_mse)
```

```
MSE of: simple_smoothing: 0.03417483130387546 double_smoothing:
0.18292165935477114 triple_smoothing: 0.027576083568331154
```

[110]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16,5))

#Plot the simple smoothing
ax1.plot(timeS_size[:-test_size], train_set, 'b--', label="train_set")
```

```
ax1.plot(timeS_size[-test_size:], test_set, color='orange', linestyle="--",␣
 ↪label="test_2")
ax1.plot(timeS_size[-test_size:], simple_prediction, 'r--', label="predictions")
ax1.set(xlabel='Time', ylabel='Value', title='Simple Exponential Smoothing')

#Plot the double smoothing
ax2.plot(timeS_size[:-test_size], train_set, 'b--', label="train_set")
ax2.plot(timeS_size[-test_size:], test_set, color='orange', linestyle="--",␣
 ↪label="test_2")
ax2.plot(timeS_size[-test_size:], double_prediction, 'r--', label="predictions")
ax2.set(xlabel='Time', ylabel='Value', title='Double Exponential Smoothing')

#Plot the triple smoothing
ax3.plot(timeS_size[:-test_size], train_set, 'b--', label="train_set")
ax3.plot(timeS_size[-test_size:], test_set, color='orange', linestyle="--",␣
 ↪label="test_2")
ax3.plot(timeS_size[-test_size:], triple_prediction, 'r--', label="predictions")
ax3.set(xlabel='Time', ylabel='Value', title='Triple Exponential Smoothing')
```

```
[110]: [Text(0.5, 0, 'Time'),
        Text(0, 0.5, 'Value'),
        Text(0.5, 1.0, 'Triple Exponential Smoothing')]
```



Using different advanced smoothing techniques does yield different results. The first 2 are not really capturing the behaviour of the time serie however the last one, the triple exponential smoothing is able to get a better forecast. The single exponential smoothing gives us a flat line as forecast.This is because the Single Exponential Smoothing cannot pickup trend or seasonality. The Double Exponential Smoothing can pickup on trend, which is exactly what we see here. This is a significant leap down but it is pretty over-reaching.Triple Exponential Smoothing pickups trend and seasonality. This is clear in the third plot to the right on the plot above. This approach makes the most sense for this data.

```
[72]: #decomp = seasonal_decompose(x=irregular_time_samples_bak, model='additive',␣
       ↪period=5)
      #trend = decomp.trend
      #seasonal = decomp.seasonal
      #residual = decomp.resid
```

```
[73]: #time = np.arange(100)
```

```
[74]: #fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16,5))

      #ax1.plot(time, trend)
      #ax1.set(xlabel='Time', ylabel='serie_trend', title='Trend')
      #ax2.plot(time, seasonal)
      #ax2.set(xlabel='Time', ylabel='serie_seasonal', title='Seasonality')
      #ax3.plot(time, residual)
      #ax3.set(xlabel='Time', ylabel='serie_residual', title='Residual')
```

period of a sinuoidal function shoudl be [period = 2pi/frequency]

```
[83]: #residual=residual[2:-2]
      #adf_residual, pvalue_residual, usedlag_residual, nobs_residual,␣
       ↪critical_values_residual, icbest_residual = adfuller(residual)
      #print('adf:', adf_residual, '| pvalue:', pvalue_residual, '| usedlag:',␣
       ↪usedlag_residual, '| nobs:', nobs_residual, '| icbest:', icbest_residual)
```

```
[84]: #pd.Series(residual).hist()
```

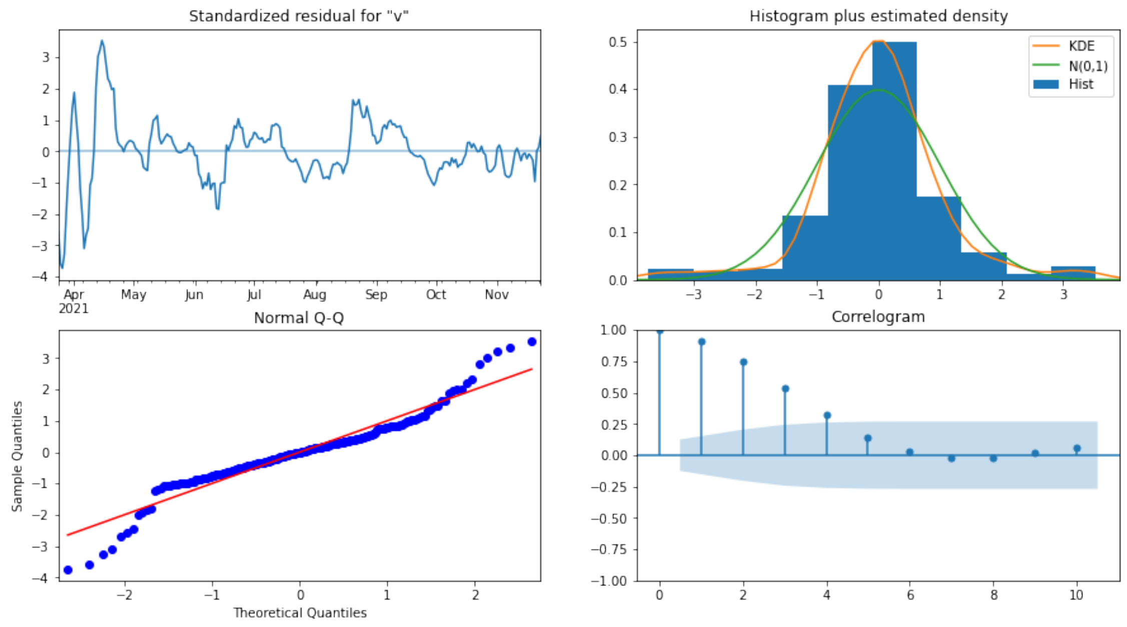
```
[85]: #len(residual)
```

1.7 First Model - Baseline

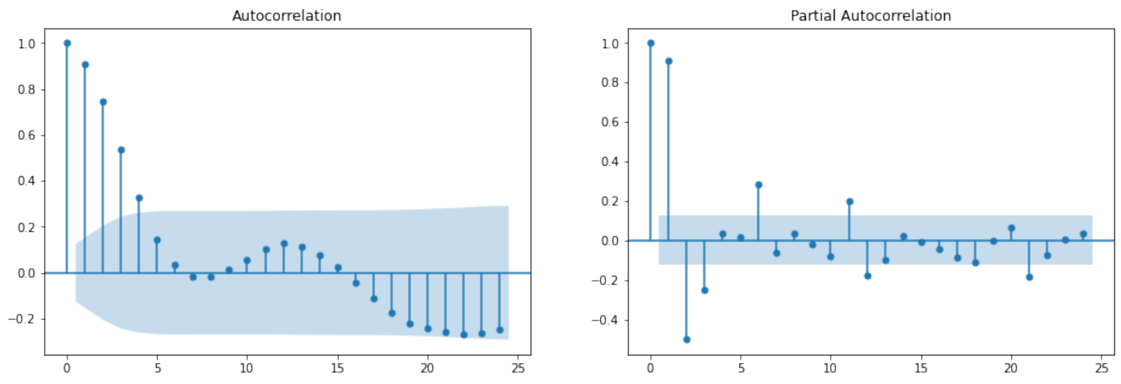
```
[111]: model1 = sm.tsa.statespace.SARIMAX(data.value,
                                          order=(0,0,0),
                                          seasonal_order=(0,1,0,5),
                                          trend='c').fit()
```

```
/home/Enry/.local/share/virtualenvs/jupyterlab-fEo2NtID/lib/python3.7/site-
packages/statsmodels/tsa/base/tsa_model.py:527: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  % freq, ValueWarning)
/home/Enry/.local/share/virtualenvs/jupyterlab-fEo2NtID/lib/python3.7/site-
packages/statsmodels/tsa/base/tsa_model.py:527: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  % freq, ValueWarning)
```

```
[112]: model1.plot_diagnostics(figsize = (15,8), lags=10);
```



```
[113]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16,5))
       sm.tsa.graphics.plot_acf(model1.resid[model1.loglikelihood_burn:], ax=ax1)
       sm.tsa.graphics.plot_pacf(model1.resid[model1.loglikelihood_burn:], ax=ax2);
```



```
[114]: model1.summary()
```

```
[114]: <class 'statsmodels.iolib.summary.Summary'>
       """
                                     SARIMAX Results
       ==============================================================================
       Dep. Variable:                  value   No. Observations:                  250
```

```
Model:             SARIMAX(0, 1, 0, 5)   Log Likelihood                -114.787
Date:                 Fri, 19 Mar 2021   AIC                            233.574
Time:                         15:19:38   BIC                            240.577
Sample:                     03-19-2021   HQIC                           236.394
                          - 11-23-2021
Covariance Type:                   opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.0202      0.025     -0.816      0.415      -0.069       0.028
sigma2         0.1494      0.008     17.649      0.000       0.133       0.166
================================================================================
===
Ljung-Box (L1) (Q):                 205.45   Jarque-Bera (JB):
97.29
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):               0.12   Skew:
-0.03
Prob(H) (two-sided):                  0.00   Kurtosis:
6.09
================================================================================
===

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

As we can see its fairly normal but the heteroskedacity is very high

[115]:
```python
warnings.filterwarnings("ignore")

series = data['value']
horizon = 3
start = int(len(data.value)*.75)
step_size = 1
order = (0,0,0)
seasonal_order = (0,1,0,5)

cross_model1 = cross_validate(series,horizon,start,step_size,
                    order = order,
                    seasonal_order = seasonal_order)
```

Define a Mape Array for collecting these values

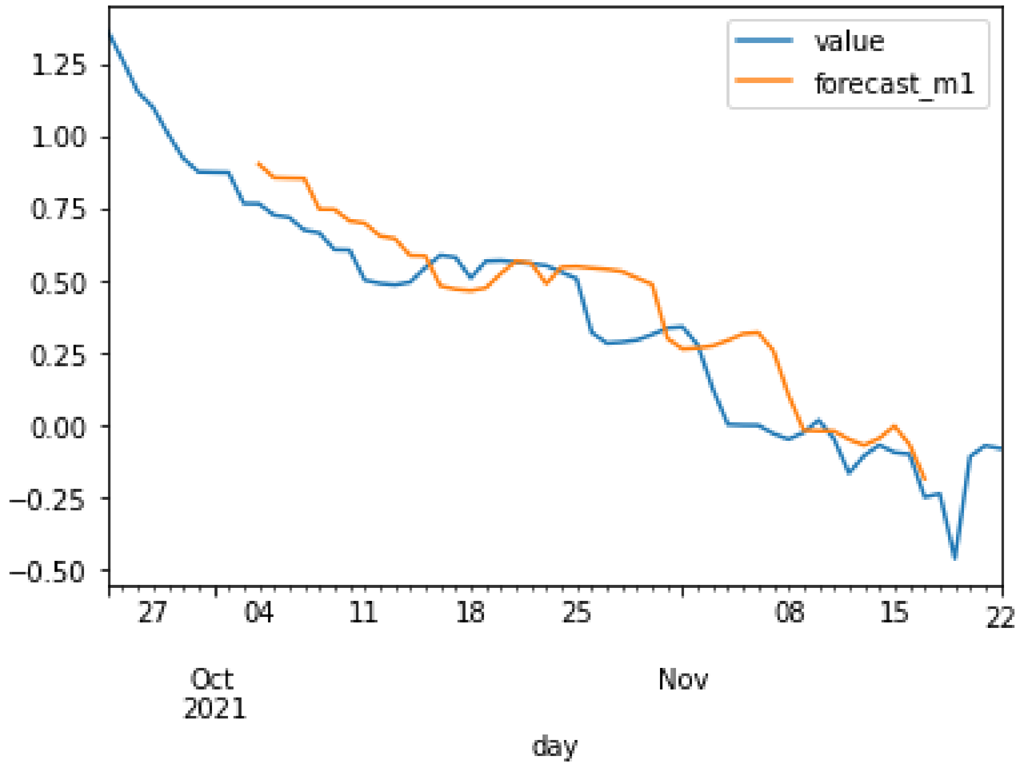
[116]:
```python
Mape_arr=[]
```

[117]:
```
Mape_arr.append(mape(cross_model1))
print('Mape:', mape(cross_model1))
```

```
Mape: 0.40093597504755946
```

1.7.1 Model 1 (baseline) Forecast

[148]:
```
data['forecast_m1'] = model1.predict(start = 200, end= 244)
data[190:][['value', 'forecast_m1']].plot();
```

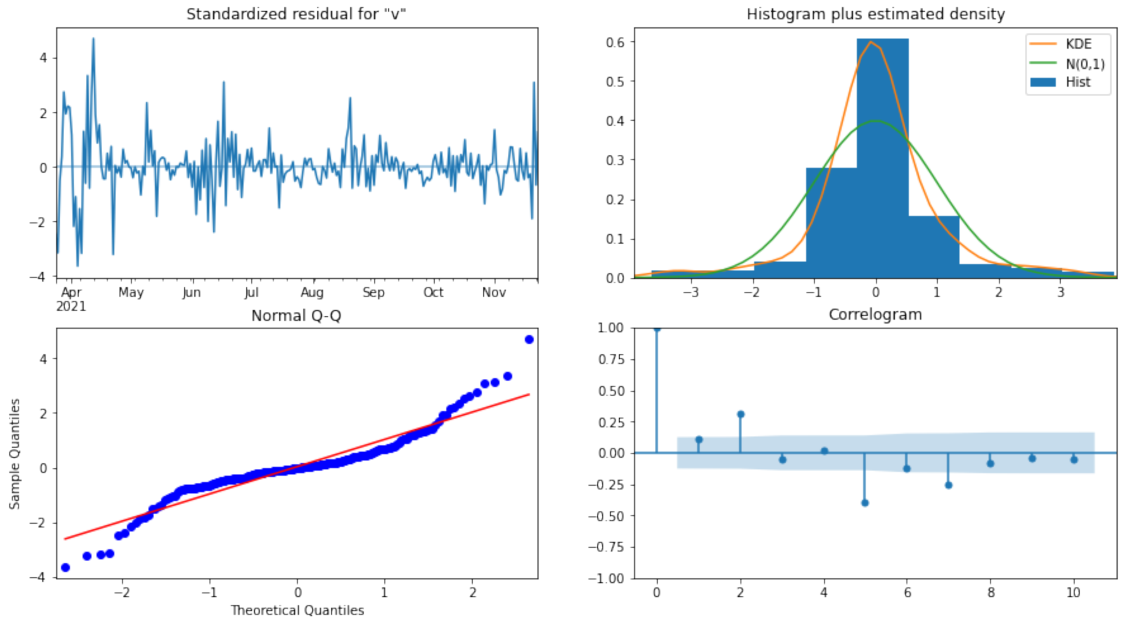


Apply a log transformation does not remove heteroschedacity

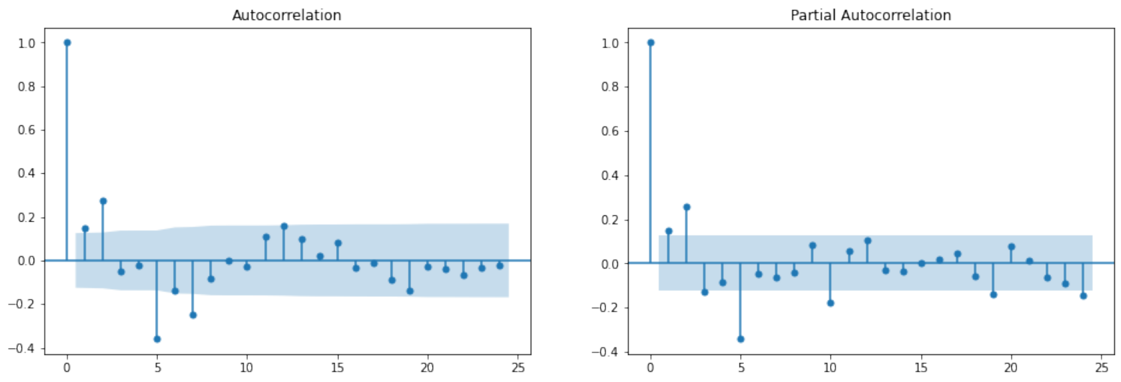
1.8 Second Model

[118]:
```
model2 = sm.tsa.statespace.SARIMAX(data.value,
                                   order=(1,0,1),
                                   seasonal_order=(0,1,0,5),
                                   trend='c').fit()
```

[119]:
```
model2.plot_diagnostics(figsize = (15,8), lags=10);
```



[120]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16,5))
sm.tsa.graphics.plot_acf(model2.resid[model2.loglikelihood_burn:], ax=ax1)
sm.tsa.graphics.plot_pacf(model2.resid[model2.loglikelihood_burn:], ax=ax2);
```



[121]:
```
model2.summary()
```

[121]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
================================================================================
==========
Dep. Variable:                                 value   No. Observations:
250
```

```
Model:              SARIMAX(1, 0, 1)x(0, 1, [], 5)   Log Likelihood
137.706
Date:                             Fri, 19 Mar 2021   AIC
-267.412
Time:                                     15:36:48   BIC
-253.407
Sample:                                 03-19-2021   HQIC
-261.773
                                      - 11-23-2021
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.0044      0.011     -0.383      0.702      -0.027       0.018
ar.L1          0.8941      0.024     36.545      0.000       0.846       0.942
ma.L1          0.3237      0.051      6.322      0.000       0.223       0.424
sigma2         0.0189      0.001     17.671      0.000       0.017       0.021
================================================================================
===
Ljung-Box (L1) (Q):                   3.15   Jarque-Bera (JB):
181.87
Prob(Q):                              0.08   Prob(JB):
0.00
Heteroskedasticity (H):               0.21   Skew:
0.37
Prob(H) (two-sided):                  0.00   Kurtosis:
7.16
================================================================================
===

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

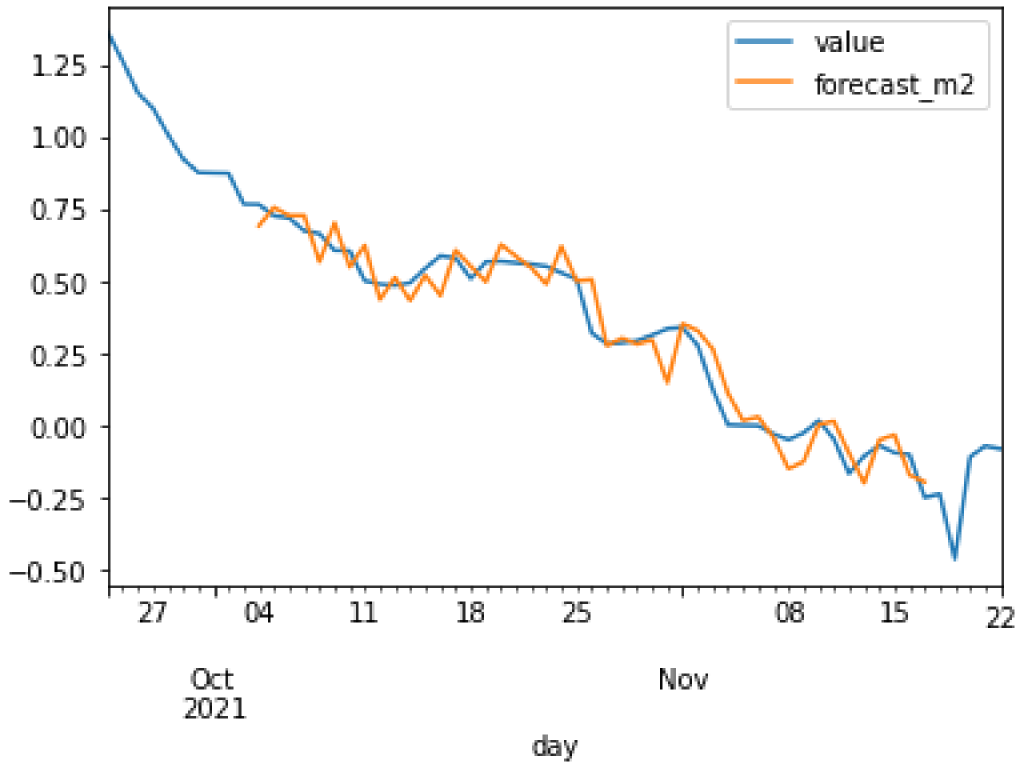
[122]:
```
warnings.filterwarnings("ignore")

series = data['value']
horizon = 3
start = int(len(data.value)*.75)
step_size = 1
order = (1,0,1)
seasonal_order = (0,1,0,5)

cross_model2 = cross_validate(series,horizon,start,step_size,
                    order = order,
                    seasonal_order = seasonal_order)
```

[123]:
```
Mape_arr.append(mape(cross_model2))
print('Mape:', mape(cross_model2))
```

```
Mape: 0.3099567003615063
```

1.8.1 Model 2 - Forecast

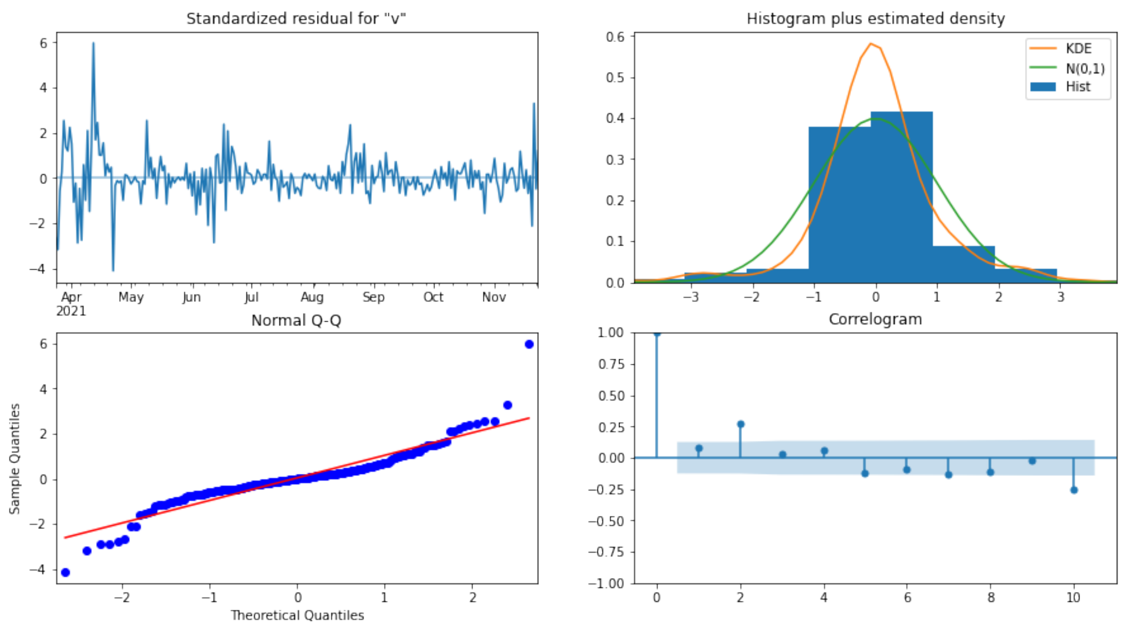
[155]:
```
data['forecast_m2'] = model2.predict(start = 200, end= 244)
data[190:][['value', 'forecast_m2']].plot();
```



1.9 Third Model

[124]:
```
model3 = sm.tsa.statespace.SARIMAX(data.value,
                                order=(1,0,1),
                                seasonal_order=(1,1,0,5),
                                trend='c').fit()
```

[125]:
```
model3.plot_diagnostics(figsize = (15,8));
```



[126]:
```
model3.summary()
```

[126]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
================================================================================
==========
Dep. Variable:                                value   No. Observations:
250
Model:             SARIMAX(1, 0, 1)x(1, 1, [], 5)   Log Likelihood
161.044
Date:                              Fri, 19 Mar 2021   AIC
-312.089
Time:                                      15:44:55   BIC
-294.583
Sample:                                  03-19-2021   HQIC
-305.039
                                       - 11-23-2021
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.0042      0.010     -0.426      0.670      -0.023       0.015
ar.L1          0.9377      0.016     57.153      0.000       0.906       0.970
ma.L1          0.3199      0.050      6.409      0.000       0.222       0.418
ar.S.L5       -0.4575      0.042    -10.975      0.000      -0.539      -0.376
sigma2         0.0155      0.001     19.706      0.000       0.014       0.017
```

```
================================================================================
===
Ljung-Box (L1) (Q):                   1.73   Jarque-Bera (JB):
489.33
Prob(Q):                              0.19   Prob(JB):
0.00
Heteroskedasticity (H):               0.23   Skew:
0.56
Prob(H) (two-sided):                  0.00   Kurtosis:
9.83
================================================================================
===

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

[127]:
```
warnings.filterwarnings("ignore")

series = data['value']
horizon = 3
start = int(len(data.value)*.75)
step_size = 1
order = (1,0,1)
seasonal_order = (1,1,0,5)

cross_model3 = cross_validate(series,horizon,start,step_size,
                    order = order,
                    seasonal_order = seasonal_order)
```

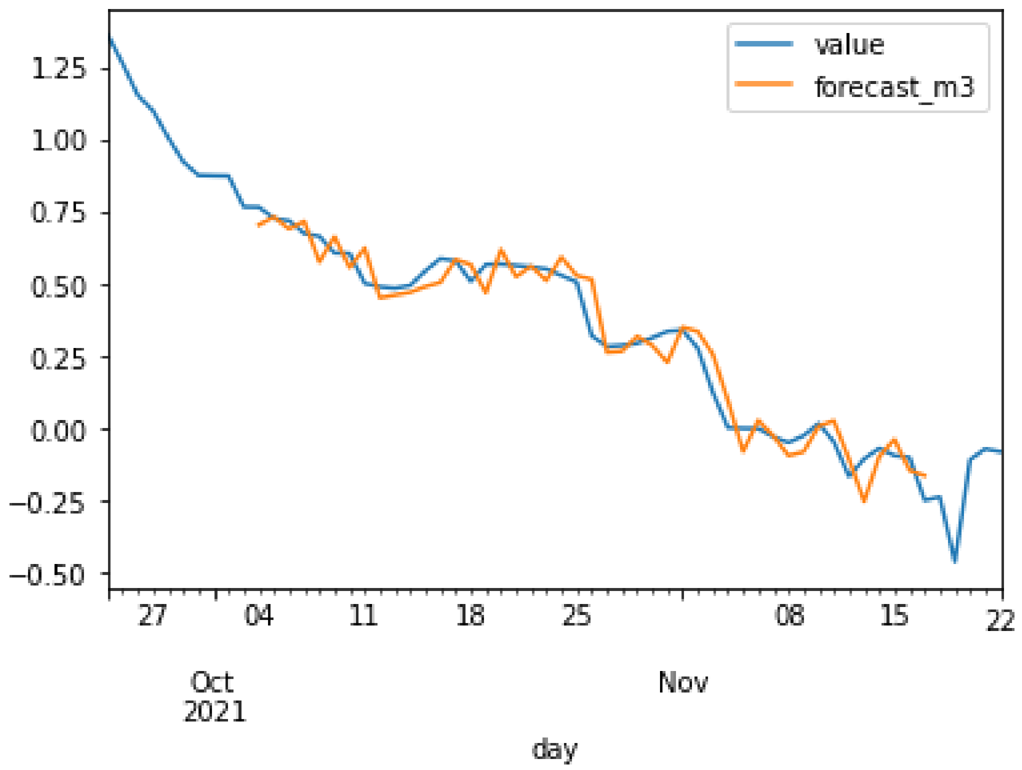
Lets try to apply a log transformation to the residual so to squash it a little bit and maybe get a more constant variance.

[128]:
```
Mape_arr.append(mape(cross_model3))
print('Mape:', mape(cross_model3))
```

```
Mape: 0.2842502070476099
```

1.9.1 Model 3 - Forecast

[170]:
```
data['forecast_m3'] = model3.predict(start = 200, end= 244)
data[190:][['value', 'forecast_m3']].plot();
```

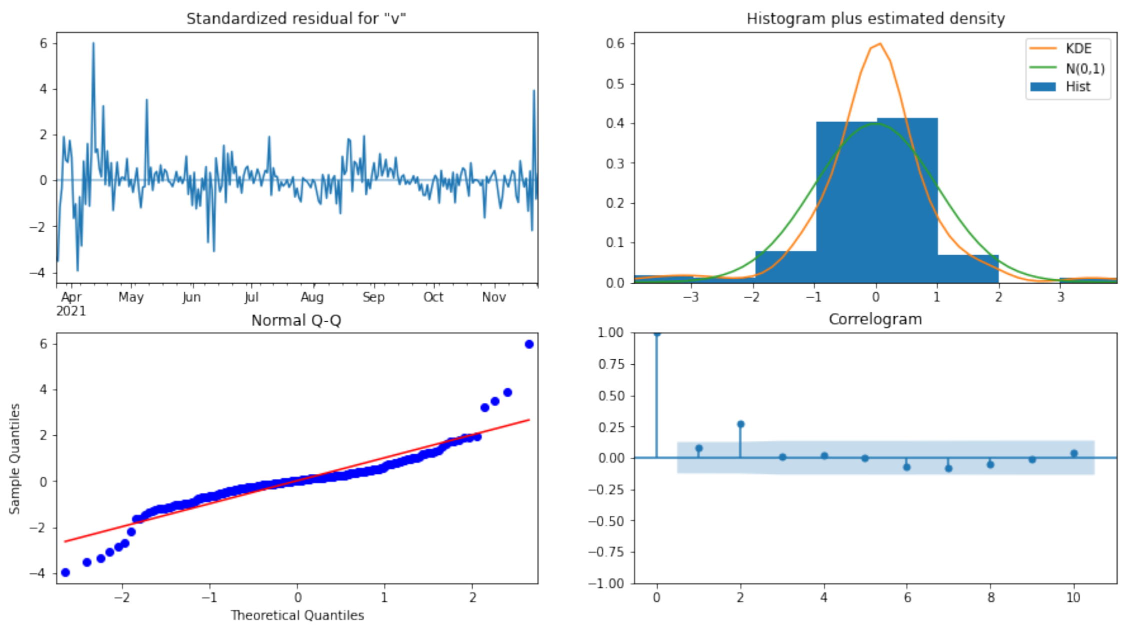


The auto model is definetly better. However I want to see if I can reduce the heteroschedacity a little bit more

1.10 Model 4

```
[129]: model4 = sm.tsa.statespace.SARIMAX(data.value,
                                         order=(1,0,1),
                                         seasonal_order=(1,1,1,5),
                                         trend='c').fit()
```

```
[134]: model4.plot_diagnostics(figsize = (15,8));
```



[130]:
```
model4.summary()
```

[130]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                      SARIMAX Results
================================================================================
=========
Dep. Variable:                              value   No. Observations:
250
Model:             SARIMAX(1, 0, 1)x(1, 1, 1, 5)   Log Likelihood
190.133
Date:                            Fri, 19 Mar 2021   AIC
-368.265
Time:                                    15:56:09   BIC
-347.257
Sample:                                03-19-2021   HQIC
-359.805
                                     - 11-23-2021
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.0002      0.001     -0.336      0.737      -0.002       0.001
ar.L1          0.9845      0.023     42.726      0.000       0.939       1.030
ma.L1          0.2979      0.052      5.680      0.000       0.195       0.401
ar.S.L5        0.1166      0.046      2.526      0.012       0.026       0.207
ma.S.L5       -0.9789      0.122     -8.040      0.000      -1.218      -0.740
```

```
sigma2          0.0117      0.001      9.327      0.000       0.009       0.014
===============================================================================
===
Ljung-Box (L1) (Q):                   1.41   Jarque-Bera (JB):
664.52
Prob(Q):                              0.23   Prob(JB):
0.00
Heteroskedasticity (H):               0.25   Skew:
0.58
Prob(H) (two-sided):                  0.00   Kurtosis:
10.98
===============================================================================
===

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

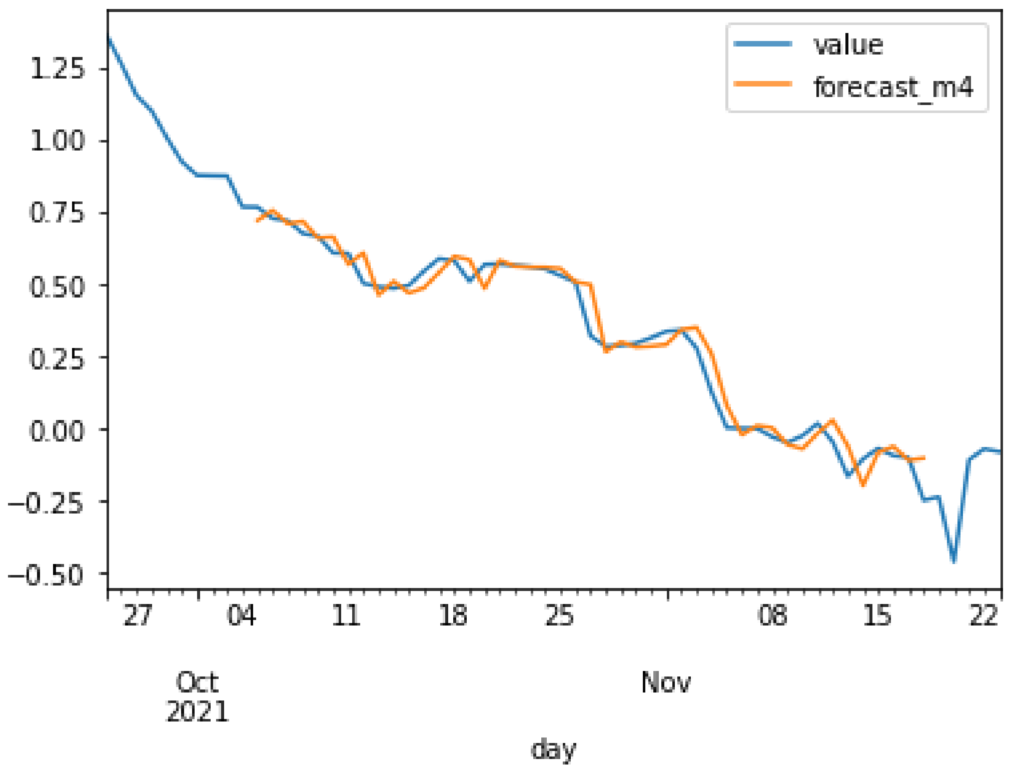
[131]:
```
series = data['value']
horizon = 3
start = int(len(data.value)*.75)
step_size = 1
order = (1,0,1)
seasonal_order = (1,1,1,5)

cross_model4 = cross_validate(series,horizon,start,step_size,
                    order = order,
                    seasonal_order = seasonal_order)
```

[132]:
```
Mape_arr.append(cross_model4)
mape(cross_model4)
```

[132]:
```
0.2972472629498805
```

1.10.1 Model 4 - Forecast

[133]:
```
data['forecast_m4'] = model4.predict(start = 200, end= 244)
data[190:][['value', 'forecast_m4']].plot();
```



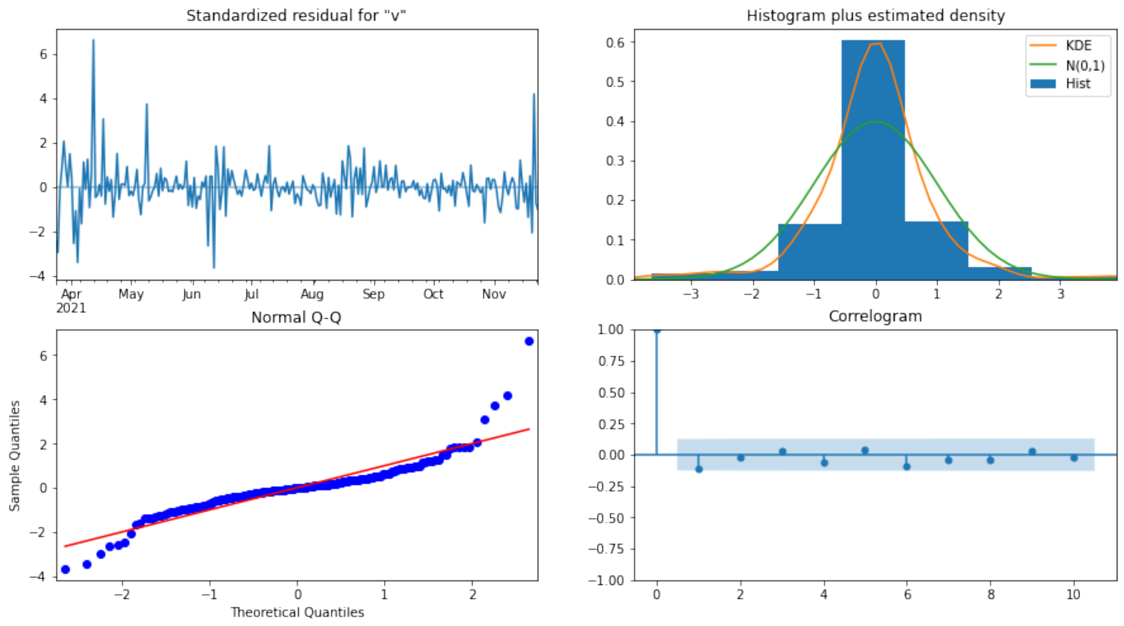
1.11 Model 5

[135]:
```
model5 = sm.tsa.statespace.SARIMAX(data.value,
                                   order=(3,0,1),
                                   seasonal_order=(1,1,1,5),
                                   trend='c').fit()
model5.summary()
```

[135]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
================================================================================
=========
Dep. Variable:                               value   No. Observations:
250
Model:             SARIMAX(3, 0, 1)x(1, 1, 1, 5)   Log Likelihood
202.533
Date:                             Fri, 19 Mar 2021   AIC
-389.066
Time:                                     16:01:52   BIC
-361.056
Sample:                                 03-19-2021   HQIC
```

```
-377.786
                                   - 11-23-2021
Covariance Type:                             opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.0003      0.001     -0.248      0.804      -0.002       0.002
ar.L1          0.6922      0.141      4.903      0.000       0.415       0.969
ar.L2          0.7758      0.168      4.632      0.000       0.447       1.104
ar.L3         -0.4975      0.063     -7.872      0.000      -0.621      -0.374
ma.L1          0.7362      0.156      4.733      0.000       0.431       1.041
ar.S.L5        0.1529      0.050      3.069      0.002       0.055       0.251
ma.S.L5       -0.9629      0.075    -12.884      0.000      -1.109      -0.816
sigma2         0.0106      0.001     12.427      0.000       0.009       0.012
================================================================================
===
Ljung-Box (L1) (Q):                   2.93   Jarque-Bera (JB):
1141.82
Prob(Q):                              0.09   Prob(JB):
0.00
Heteroskedasticity (H):               0.29   Skew:
1.23
Prob(H) (two-sided):                  0.00   Kurtosis:
13.29
================================================================================
===

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

```
[139]: model5.plot_diagnostics(figsize = (15,8));
```



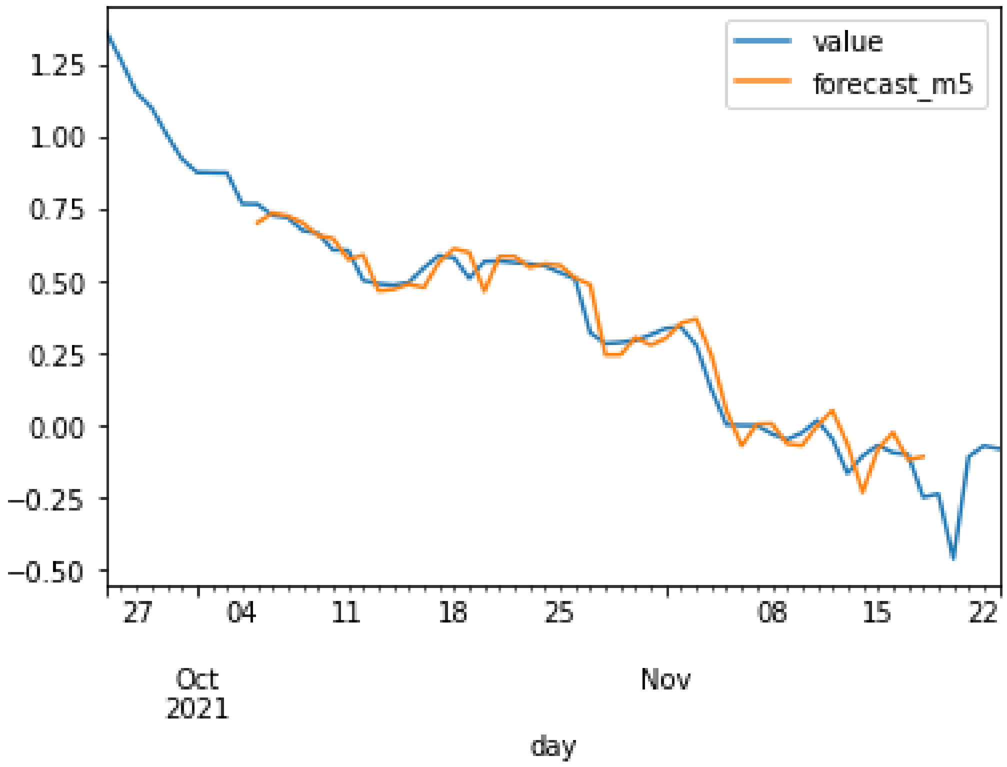
[136]:
```
series = data['value']
horizon = 3
start = int(len(data.value)*.75)
step_size = 1
order = (3,0,1)
seasonal_order = (1,1,1,5)

cross_model5 = cross_validate(series,horizon,start,step_size,
                    order = order,
                    seasonal_order = seasonal_order)
```

[137]:
```
Mape_arr.append(cross_model5)
mape(cross_model5)
```

[137]:
```
0.32357376854363096
```

1.11.1 Model 5 - Forecast

[138]:
```
data['forecast_m5'] = model5.predict(start = 200, end= 244)
data[190:][['value', 'forecast_m5']].plot();
```



1.12 Auto-Model

[140]:
```
model_auto = pm.auto_arima(data.value, start_p=0, start_q=0,
                           max_p=3, max_q=3, m=5,
                           start_P=0, seasonal=True,
                           d=0, D=1, trace=True,
                           error_action='ignore',
                           suppress_warnings=True,
                           stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(0,1,1)[5] intercept   : AIC=229.682, Time=0.22 sec
 ARIMA(0,0,0)(0,1,0)[5] intercept   : AIC=233.574, Time=0.03 sec
 ARIMA(1,0,0)(1,1,0)[5] intercept   : AIC=-278.627, Time=0.33 sec
 ARIMA(0,0,1)(0,1,1)[5] intercept   : AIC=6.519, Time=0.38 sec
 ARIMA(0,0,0)(0,1,0)[5]             : AIC=232.239, Time=0.02 sec
 ARIMA(1,0,0)(0,1,0)[5] intercept   : AIC=-231.431, Time=0.12 sec
 ARIMA(1,0,0)(2,1,0)[5] intercept   : AIC=-303.842, Time=1.07 sec
 ARIMA(1,0,0)(2,1,1)[5] intercept   : AIC=-322.005, Time=1.38 sec
 ARIMA(1,0,0)(1,1,1)[5] intercept   : AIC=inf, Time=0.69 sec
 ARIMA(1,0,0)(2,1,2)[5] intercept   : AIC=inf, Time=2.21 sec
 ARIMA(1,0,0)(1,1,2)[5] intercept   : AIC=inf, Time=1.11 sec
 ARIMA(0,0,0)(2,1,1)[5] intercept   : AIC=232.918, Time=0.64 sec
 ARIMA(2,0,0)(2,1,1)[5] intercept   : AIC=inf, Time=2.00 sec
 ARIMA(1,0,1)(2,1,1)[5] intercept   : AIC=inf, Time=1.69 sec
 ARIMA(0,0,1)(2,1,1)[5] intercept   : AIC=9.176, Time=0.79 sec
 ARIMA(2,0,1)(2,1,1)[5] intercept   : AIC=-379.249, Time=2.08 sec
 ARIMA(2,0,1)(1,1,1)[5] intercept   : AIC=inf, Time=1.10 sec
 ARIMA(2,0,1)(2,1,0)[5] intercept   : AIC=-366.422, Time=1.66 sec
 ARIMA(2,0,1)(2,1,2)[5] intercept   : AIC=inf, Time=2.07 sec
 ARIMA(2,0,1)(1,1,0)[5] intercept   : AIC=-344.073, Time=0.81 sec
 ARIMA(2,0,1)(1,1,2)[5] intercept   : AIC=inf, Time=1.93 sec
 ARIMA(3,0,1)(2,1,1)[5] intercept   : AIC=-389.604, Time=2.20 sec
 ARIMA(3,0,1)(1,1,1)[5] intercept   : AIC=inf, Time=1.49 sec
 ARIMA(3,0,1)(2,1,0)[5] intercept   : AIC=-367.803, Time=1.88 sec
 ARIMA(3,0,1)(2,1,2)[5] intercept   : AIC=inf, Time=2.36 sec
 ARIMA(3,0,1)(1,1,0)[5] intercept   : AIC=-344.455, Time=1.26 sec
 ARIMA(3,0,1)(1,1,2)[5] intercept   : AIC=-389.979, Time=2.08 sec
 ARIMA(3,0,1)(0,1,2)[5] intercept   : AIC=inf, Time=1.89 sec
 ARIMA(3,0,1)(0,1,1)[5] intercept   : AIC=inf, Time=1.31 sec
 ARIMA(3,0,0)(1,1,2)[5] intercept   : AIC=inf, Time=2.62 sec
 ARIMA(3,0,2)(1,1,2)[5] intercept   : AIC=inf, Time=2.41 sec
 ARIMA(2,0,0)(1,1,2)[5] intercept   : AIC=inf, Time=2.03 sec
 ARIMA(2,0,2)(1,1,2)[5] intercept   : AIC=inf, Time=2.80 sec
 ARIMA(3,0,1)(1,1,2)[5]             : AIC=inf, Time=1.79 sec

Best model:  ARIMA(3,0,1)(1,1,2)[5] intercept
Total fit time: 48.485 seconds
```

[141]:
```python
print('order: ',model_auto.order)
print('seasonal order: ',model_auto.seasonal_order)
```

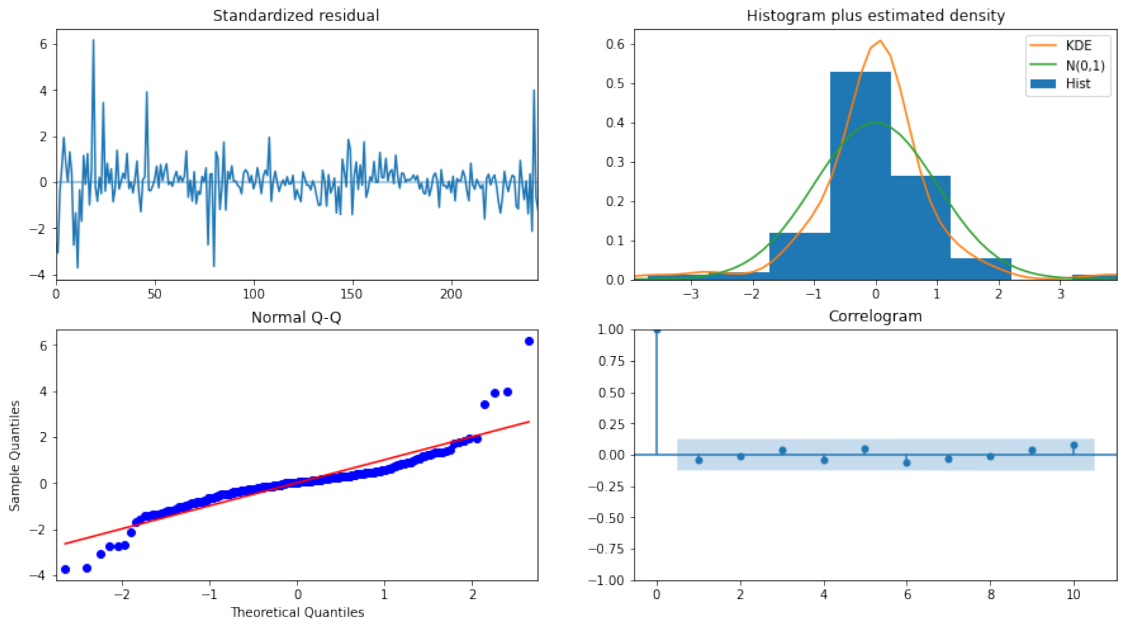
```
order:  (3, 0, 1)
seasonal order:  (1, 1, 2, 5)
```

[142]:
```python
model_auto5 = sm.tsa.statespace.SARIMAX(data.value,
                                order=(3,0,1),
                                seasonal_order=(1,1,2,5),
                                trend='c').fit()
model_auto5.summary()
```

[142]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                       SARIMAX Results
================================================================================
==============
Dep. Variable:                                  value   No. Observations:
250
Model:               SARIMAX(3, 0, 1)x(1, 1, [1, 2], 5)   Log Likelihood
```

```
203.989
Date:                            Fri, 19 Mar 2021   AIC
-389.979
Time:                                    16:14:58   BIC
-358.468
Sample:                                03-19-2021   HQIC
-377.289
                                     - 11-23-2021
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.0006      0.002     -0.263      0.792      -0.005       0.004
ar.L1          0.7241      0.207      3.494      0.000       0.318       1.130
ar.L2          0.6641      0.270      2.464      0.014       0.136       1.192
ar.L3         -0.4435      0.086     -5.184      0.000      -0.611      -0.276
ma.L1          0.6303      0.229      2.753      0.006       0.182       1.079
ar.S.L5       -0.4754      0.192     -2.474      0.013      -0.852      -0.099
ma.S.L5       -0.3196      0.212     -1.511      0.131      -0.734       0.095
ma.S.L10      -0.5838      0.175     -3.328      0.001      -0.928      -0.240
sigma2         0.0106      0.001     14.810      0.000       0.009       0.012
================================================================================
===
Ljung-Box (L1) (Q):                   0.36   Jarque-Bera (JB):
814.89
Prob(Q):                              0.55   Prob(JB):
0.00
Heteroskedasticity (H):               0.29   Skew:
0.87
Prob(H) (two-sided):                  0.00   Kurtosis:
11.76
================================================================================
===

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

[143]:
```
model_auto.plot_diagnostics(figsize = (15,8));
```



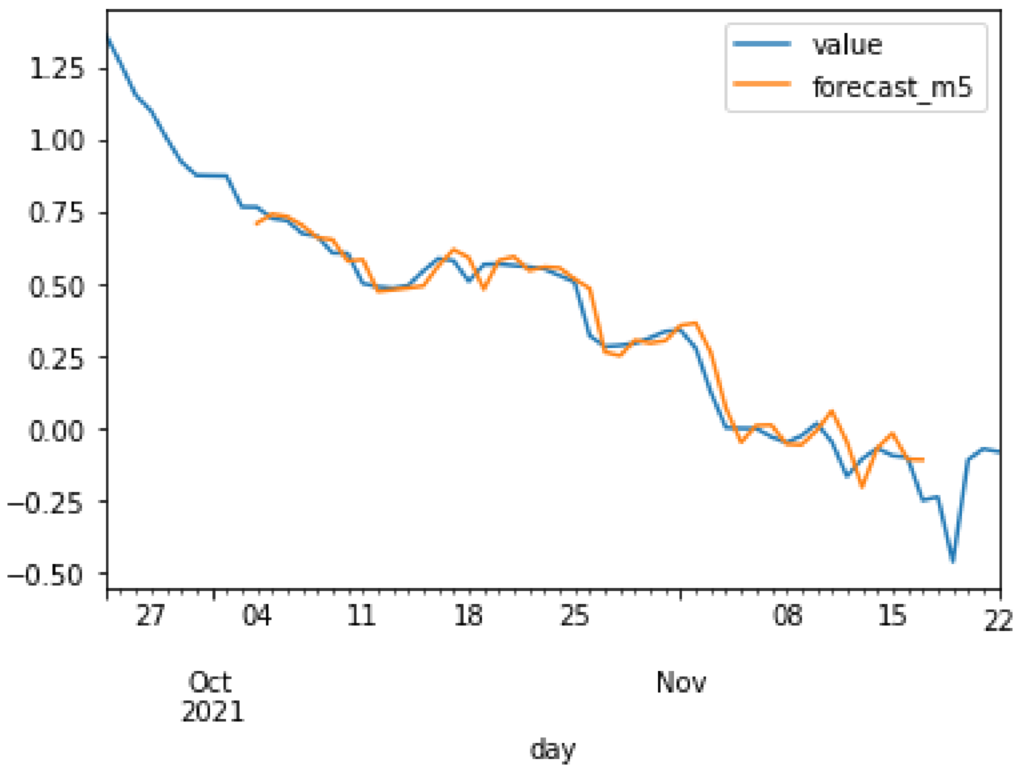
[183]:
```
series = data['value']
horizon = 3
start = int(len(data.value)*.75)
step_size = 1
order = (3,0,1)
seasonal_order = (1,1,2,5)

cross_model_auto5 = cross_validate(series,horizon,start,step_size,
                                   order = order,
                                   seasonal_order = seasonal_order)
```

[184]:
```
Mape_arr.append(cross_model_auto5)
mape(cross_model_auto5)
```

[184]:
```
0.3096862311869493
```

1.12.1 Model auto_5 Prediction

[185]:
```
data['forecast_m5'] = model_auto5.predict(start = 200, end= 244)
data[190:][['value', 'forecast_m5']].plot();
```


value
forecast_m5
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
27
04
11
18
25
08
15
22
Oct
2021
Nov
day