Time Series Analysis: Smart Water Analytics

Paulina Kossowska
September 2021

“One of the main objectives of this course is to help you gain hands-on experience in communicating insightful and impactful findings to stakeholders. In this project you will leverage the tools and techniques you learned throughout this course, including Time Series Analysis, Survival Analysis, or Deep Learning in an analytical task of your choosing. You can use any of these analysis in any data set that includes a time component. You are free to choose the technique that best help analyze a data set that you feel passionate about. Just as important as the analytical work, you will also prepare a report that communicates insights you found from your model development exercise.”

Understanding the Data

Aquifer_Petrignano.csv

I have downloaded a dataset from Kaggle

Acea Smart Water Analytics

The Acea Group is one of the leading Italian multiutility operators. Listed on the Italian Stock Exchange since 1999, the company manages and develops water and electricity networks and environmental services. Acea is the foremost Italian operator in the water services sector supplying 9 million inhabitants in Lazio, Tuscany, Umbria, Molise, Campania.

Goals of this analysis

The main goals for this analysis:

The goal for this analysis will be perform EDA, together with data pre-processing, features engineering and also perform time series forecasting.

1. **Data Pre-processing**
   - 1.1. Chronological Order and Equidistant Timestamps
   - 1.2. Missing Values
   - 1.3. Smoothing data/Resampling
   - 1.4. Stationarity
   - 1.5. Transformation

2. **Feature Engineering**
   - 2.1. Cyclical Features
   - 2.2. Time Series Decomposition
   - 2.3. Lag

3. **Exploratory Data Analysis**
   - 3.1. Seasonal Components of Features
   - 3.2. Correlations and Heatmap
   - 3.3. Autocorrelations Analysis

4. **Modeling**
   - 4.1. Prophet
   - 4.2. ARIMA
   - 4.3. Auto-ARIMA
   - 4.4. LSTM

Data Pre-processing

Chronological Order and Equidistant Timestamps

After examining some basic information about dataset I found out that my dataset contains 5223 rows and 8 columns. I used shape(), dtypes(), isnull() methods from pandas library to understand my data better. Then, I decided to rename columns to made them more clear for further analysis. I finished with data shown on below table.

	date	rainfall	depth_to_groundwater	temperature	drainage_volume	hydrometry
0	01/01/2009	0.0	-31.14	5.2	-24530.688	2.4
1	02/01/2009	0.0	-31.11	2.3	-28785.888	2.5
2	03/01/2009	0.0	-31.07	4.4	-25766.208	2.4
3	04/01/2009	0.0	-31.05	0.8	-27919.296	2.4
4	05/01/2009	0.0	-31.01	-1.9	-29854.656	2.3

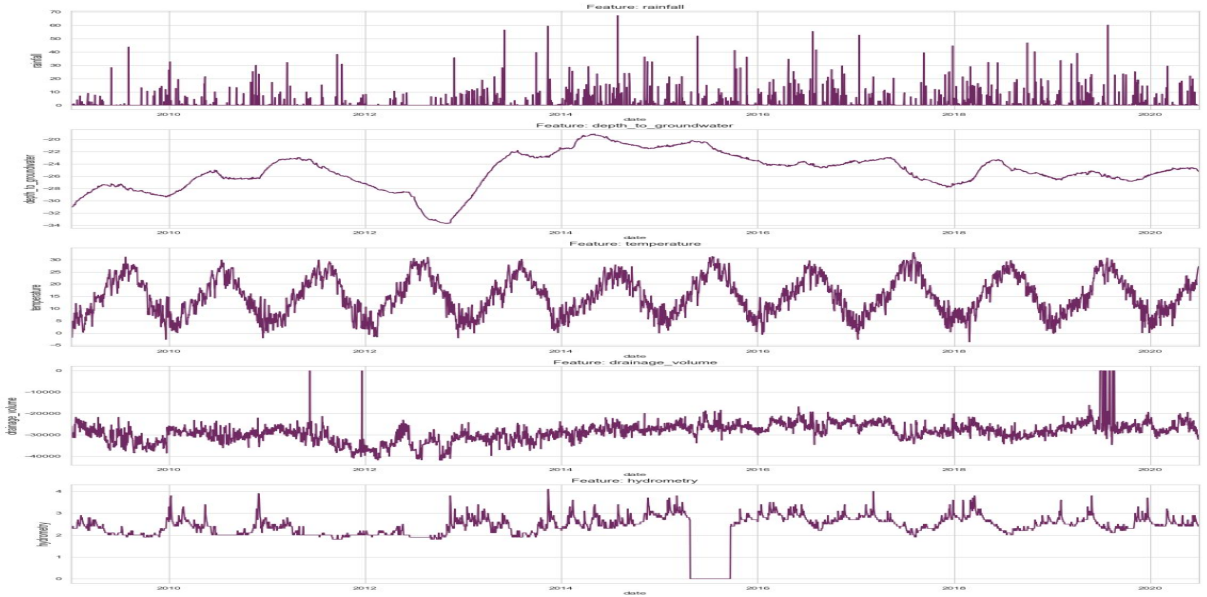
Chronological Order and Equidistant Timestamps

Columns description:

- **Rainfall** indicates the quantity of rain falling (mm)
- **Temperature** indicates the temperature (°C)
- **Volume** indicates the volume of water taken from the drinking water treatment plant (m 3)
- **Hydrometry** indicates the groundwater level (m)
- **Depth to Groundwater** indicates the groundwater level (m from the ground floor)

Chronological Order and Equidistant Timestamps

Each features is represented on the plot graph on the right. To understand the results better some closer look will be needed.



Chronological Order and Equidistant Timestamps

To finish in this subsection I checked if data are in chronological order and the timestamps should be equidistant in time series. The chronological order can be achieved by sorting the dataframe by the timestamps. Equidistant timestamps indicates constant time intervals. To check this, the difference between each timestamp can be taken.

The time interval is one day and the data is already in chronological order. Therefore, I do not have to do this additional data preparation step.

Missing Values

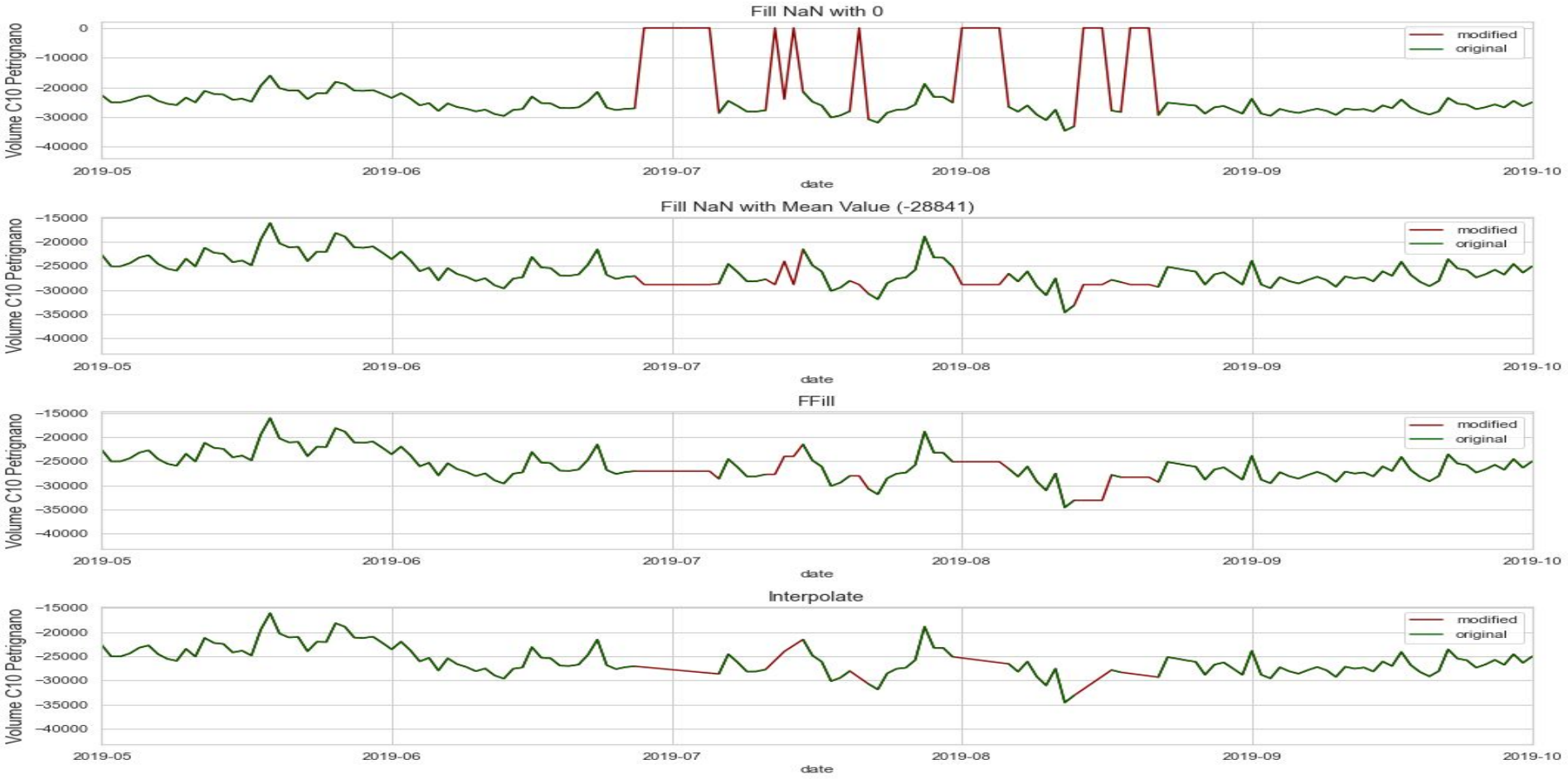
Only two columns **depth_to_groundwater** and **drainage_volume** has missing values.

On the next slide I will show the results of plotting different methods to handle with missing values. I used 4 options:

- Fill NaN with Outlier or Zero
- Fill NaN with Mean Value
- Fill NaN with Last Value with ffill()
- Fill NaN with Linearly Interpolated Value with interpolate()

Then, as results will show the best options was the interpolate, so I chosen this method to deal with missing values in my dataset.

Missing Values



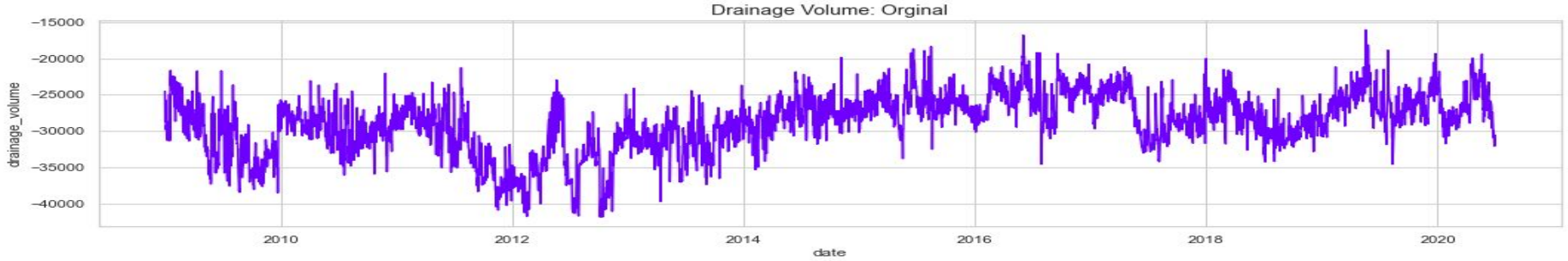
Smoothing data/Resampling

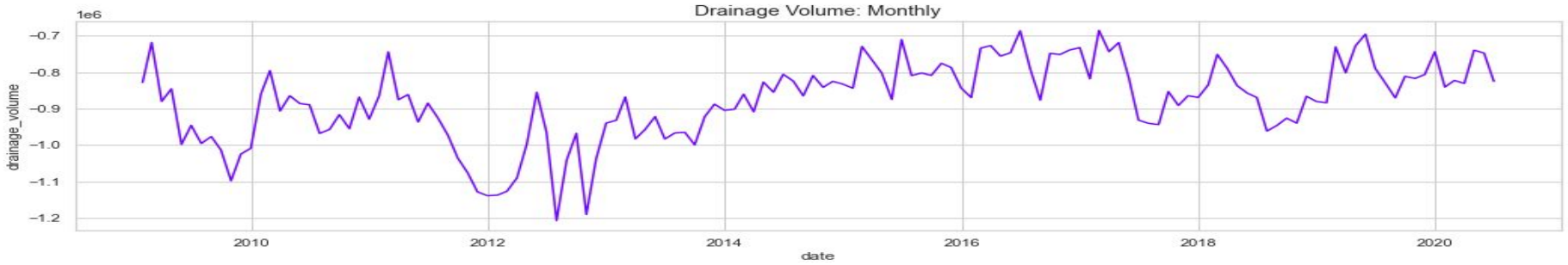
Resampling can provide additional information on the data. There are two main types of resampling:

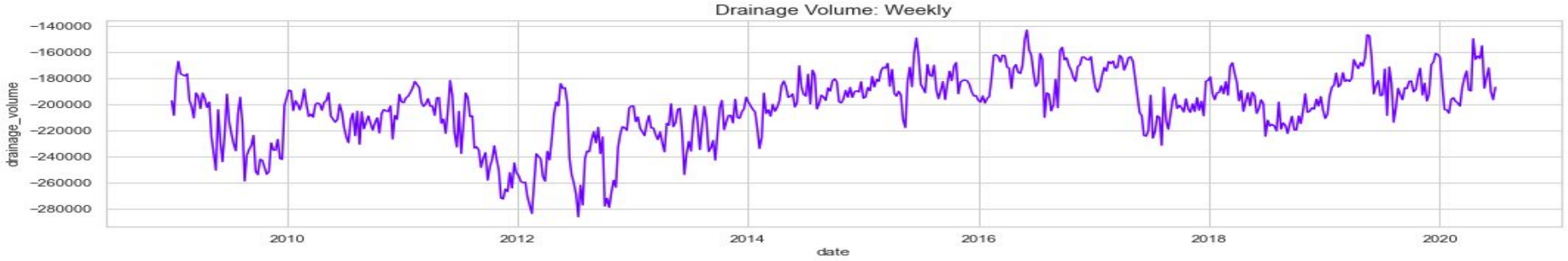
- upsampling is when the frequency of samples is increased
- downsampling is when the frequency of samples is decreased

In my analysis first I took **drainage_volume** column and downsampled it to monthly and weekly and visualize it. The results will be show on the next slide. Based on that graph I decided to downsampled my data to weekly because this smooth the data and help me in further analysis.

Smoothing data/Resampling







Stationarity

To check for stationarity I used different approaches:

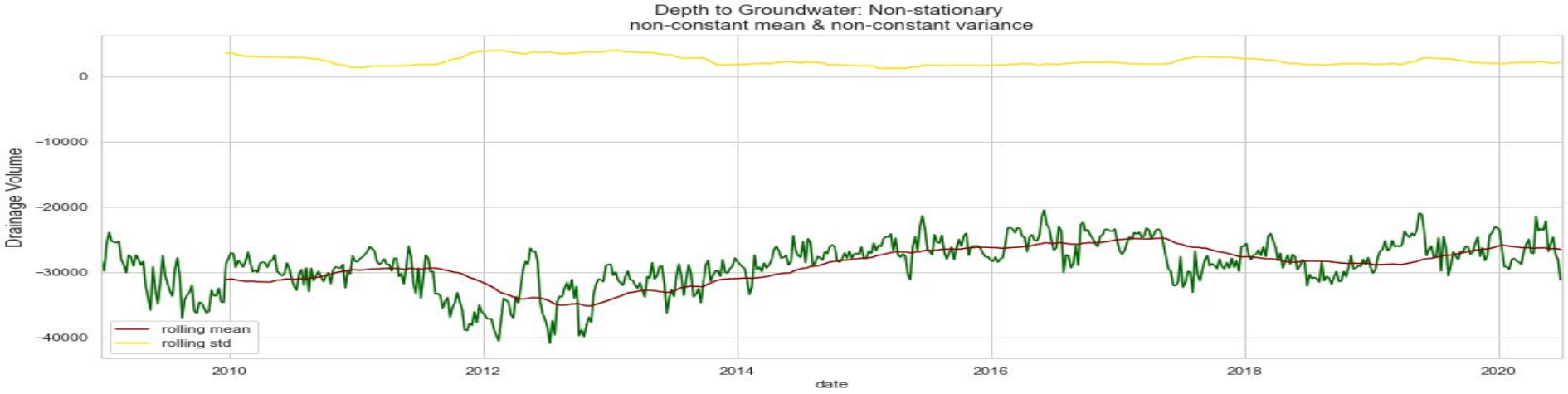
1. Basic statistics
2. Visually check for trends and seasonality
3. Augmented Dickey-Fuller test

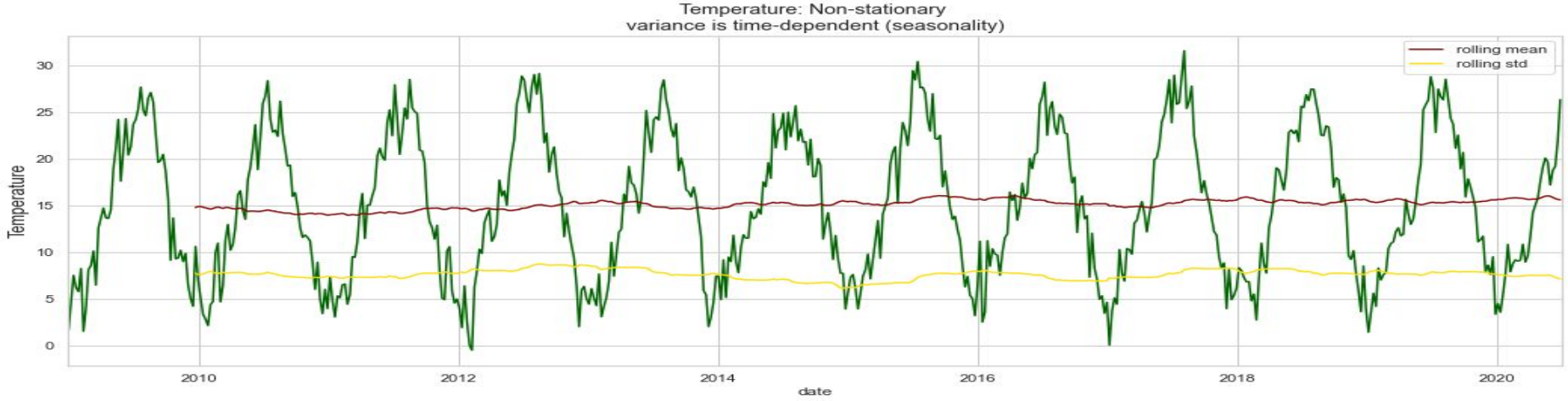
First, I created a function **check_chunks** which split column into 10 different chunks and helped me checked for mean and variance for that column. Secondly, I plotted rolling mean and standard deviation together with histograms to visually check for stationarity. Finally, I used statistical procedure **adfuller** from **statsmodels.tsa.stattools** to suss out whether a time series is stationary or not according to assumptions:

- null hypothesis: the series is not stationary
- alternative hypothesis: the series is stationary
- I chosen significance level equal to 0.05

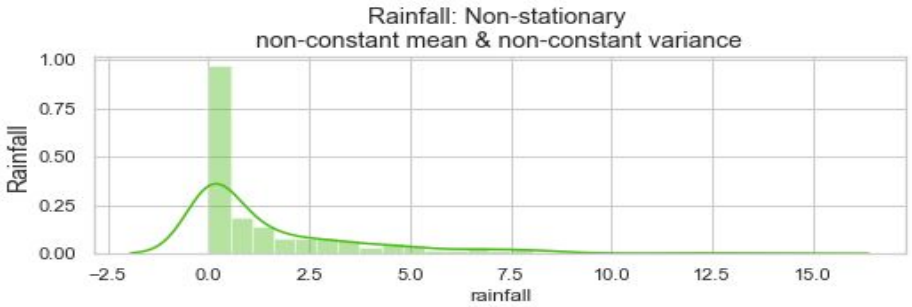
The results of this subsection are represented the next couple of slides.

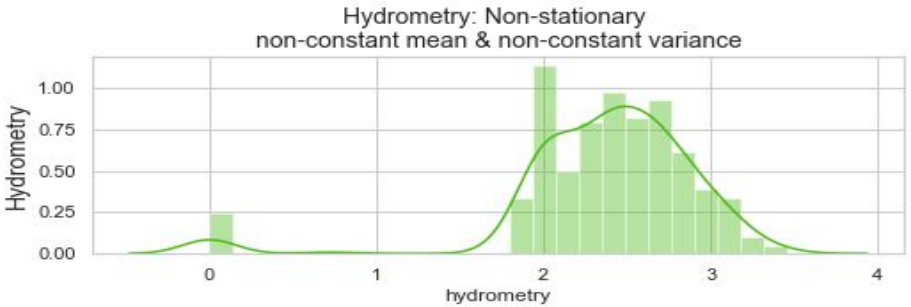
Stationarity

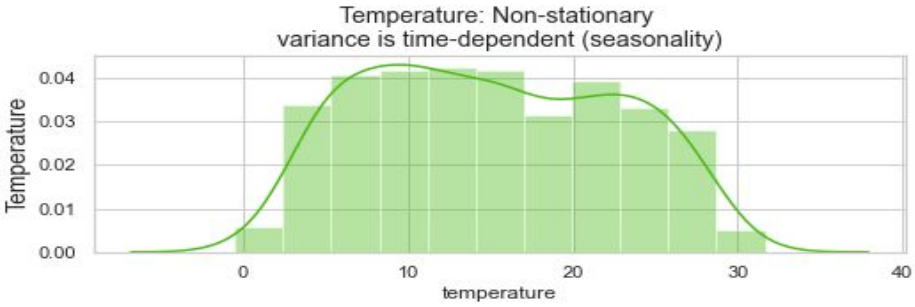


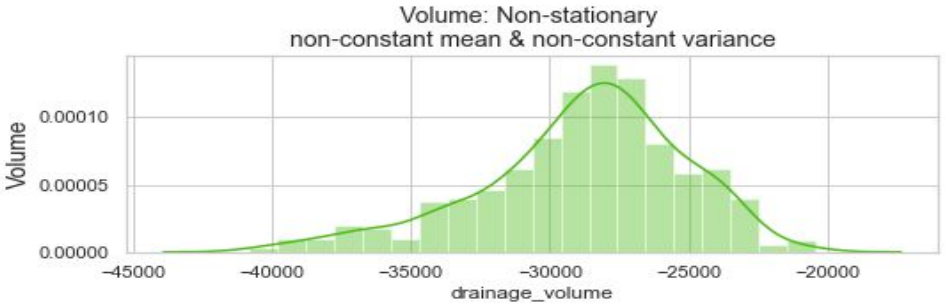


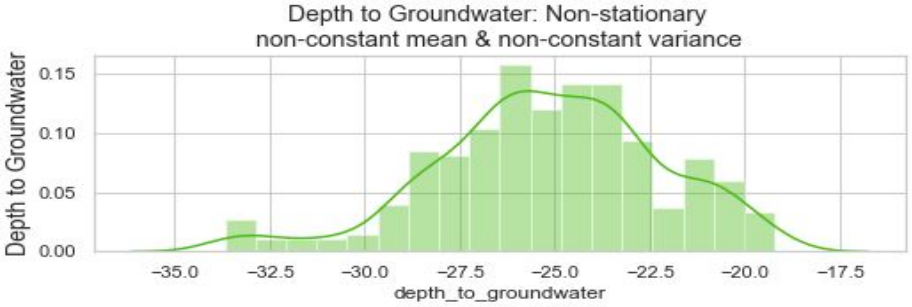
Stationarity



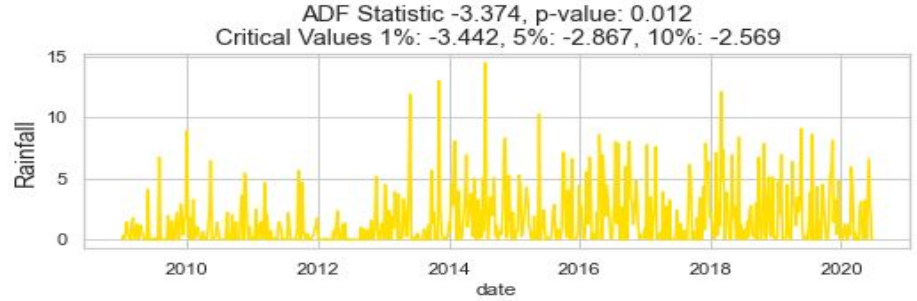


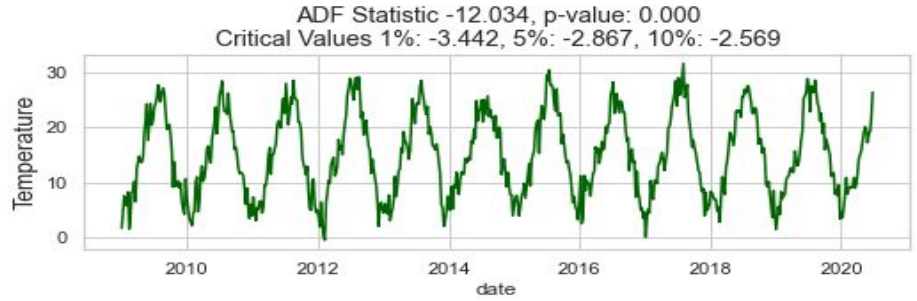


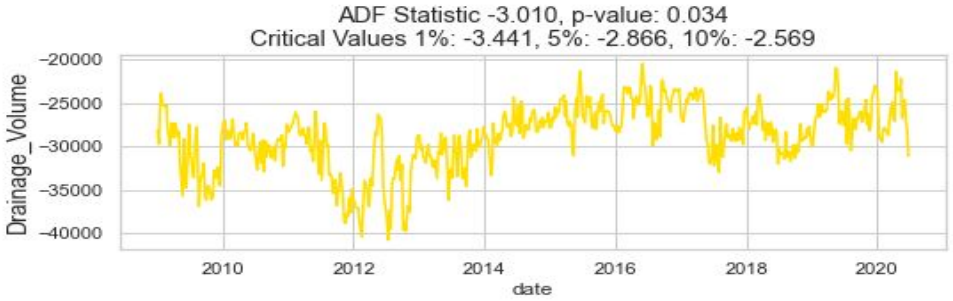


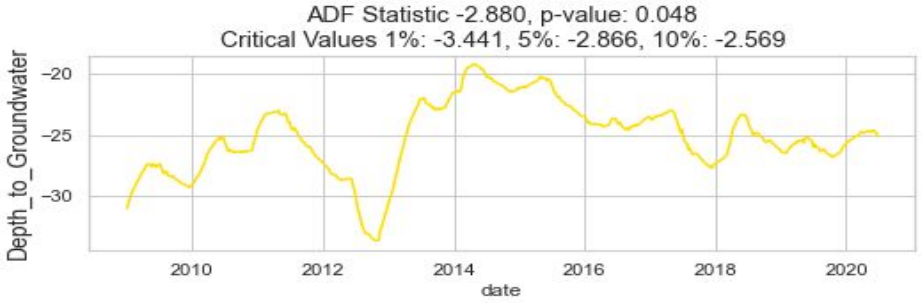


Stationarity







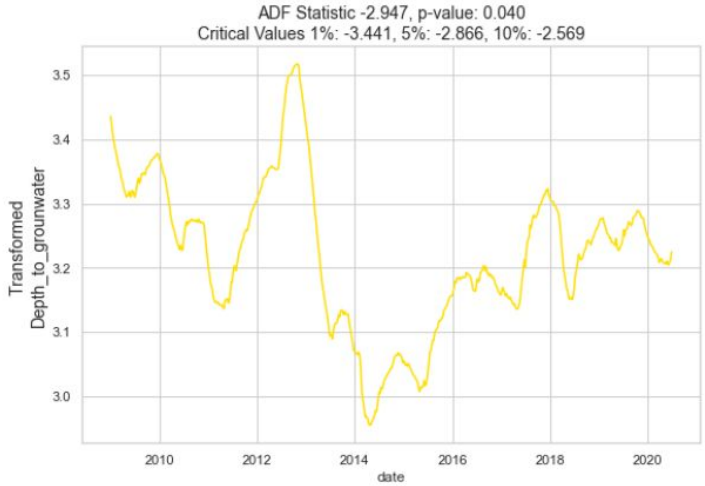


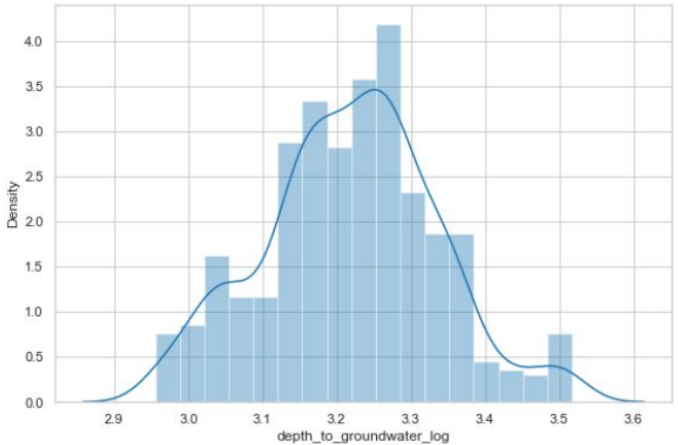
Transformation

The two most common methods to transform series into stationary ones are:

- ❖ transformation: e.g. log or square root to stabilize non-constant variance
- ❖ differencing: subtract the current value from the previous one

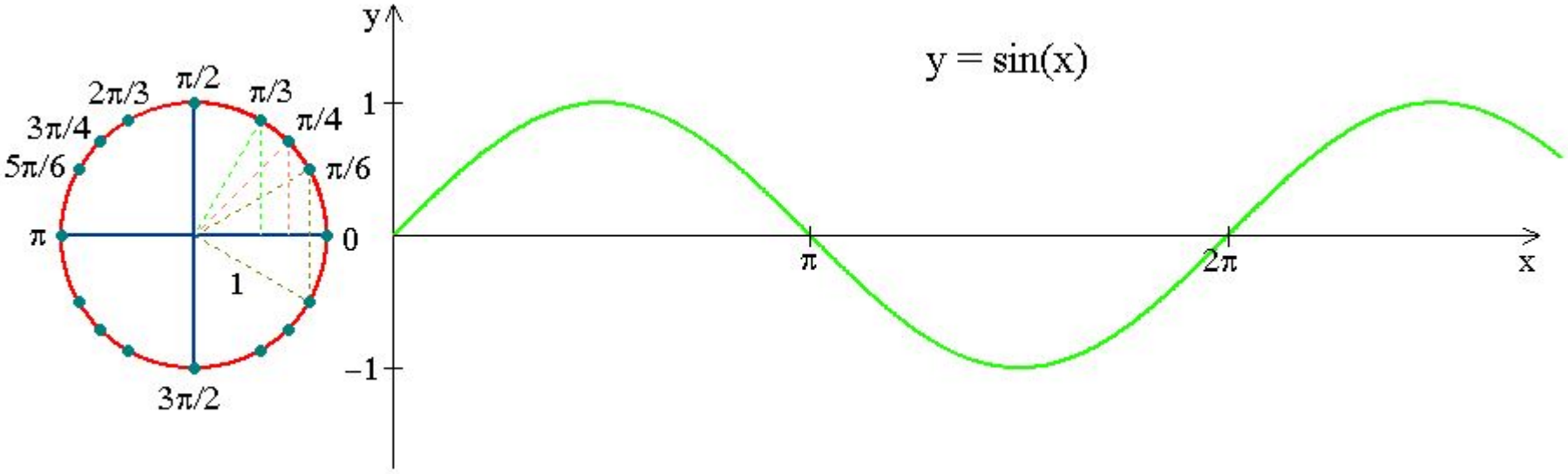
I used **np.log** of absolute values together with **first order differencing**. Then, I performed adfuller test on this log variables and visualize it.





Feature Engineering

Cyclical Features



As dataset can have hidden patterns which will not be revealed by the regular features, I am talking here about cyclical patterns such as hours of the day, days of the week, months, seasons, etc.

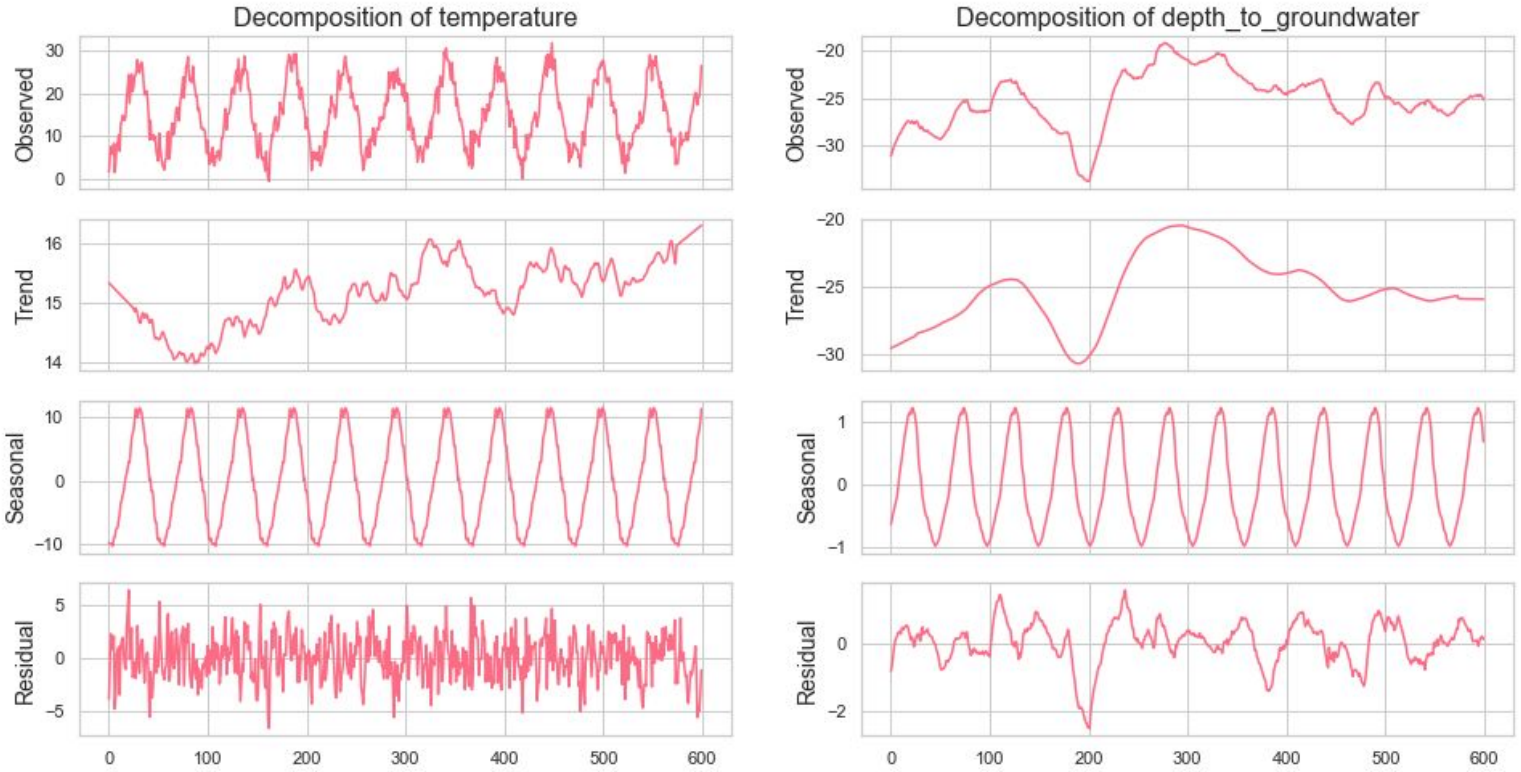
Time Series Decomposition

Time series decomposition involves thinking of a series as a combination of level, trend, seasonality, and noise components.

These components are defined as follows:

- **Level:** The average value in the series
- **Trend:** The increasing or decreasing value in the series
- **Seasonality:** The repeating short-term cycle in the series
- **Noise:** The random variation in the series

Time Series Decomposition



Lag

I want to calculate each variable with a shift() (lag) to compare the correlation with the other variables.

```
weeks_in_month = 4

for column in core_columns:
    df[f'{column}_seasonal_shift_b_2m'] = df[f'{column}_seasonal'].shift(-2 * weeks_in_month)
    df[f'{column}_seasonal_shift_b_1m'] = df[f'{column}_seasonal'].shift(-1 * weeks_in_month)
    df[f'{column}_seasonal_shift_1m'] = df[f'{column}_seasonal'].shift(1 * weeks_in_month)
    df[f'{column}_seasonal_shift_2m'] = df[f'{column}_seasonal'].shift(2 * weeks_in_month)
    df[f'{column}_seasonal_shift_3m'] = df[f'{column}_seasonal'].shift(3 * weeks_in_month)
```

Exploratory Data Analysis

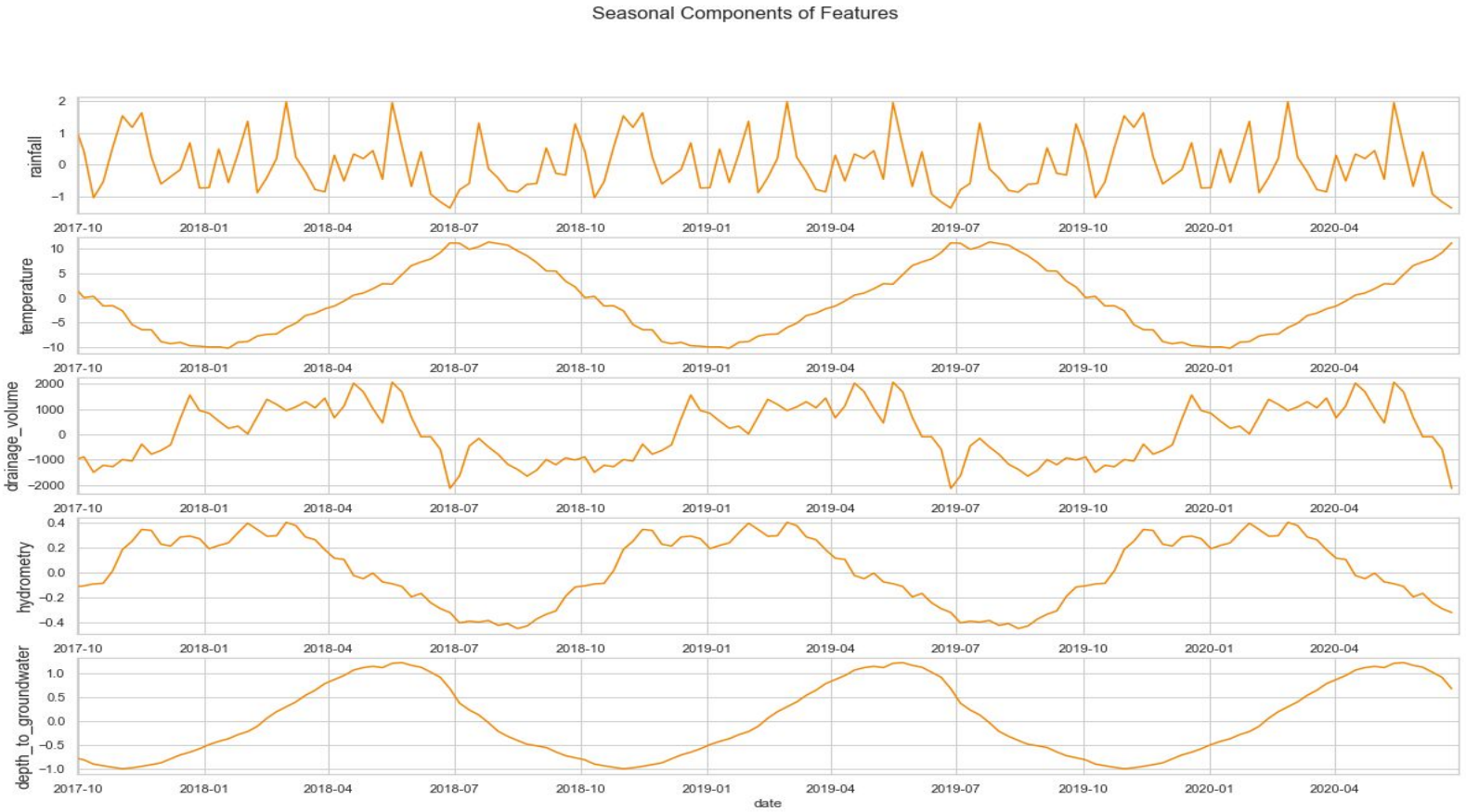
Seasonal Components of Features

In this subsection I want to deep deeper into my dataset and visualize in more details how variables looks like. Earlier in my notebook I created seasonal versions of each features and now I would like to plot this results and present them in the next slide.

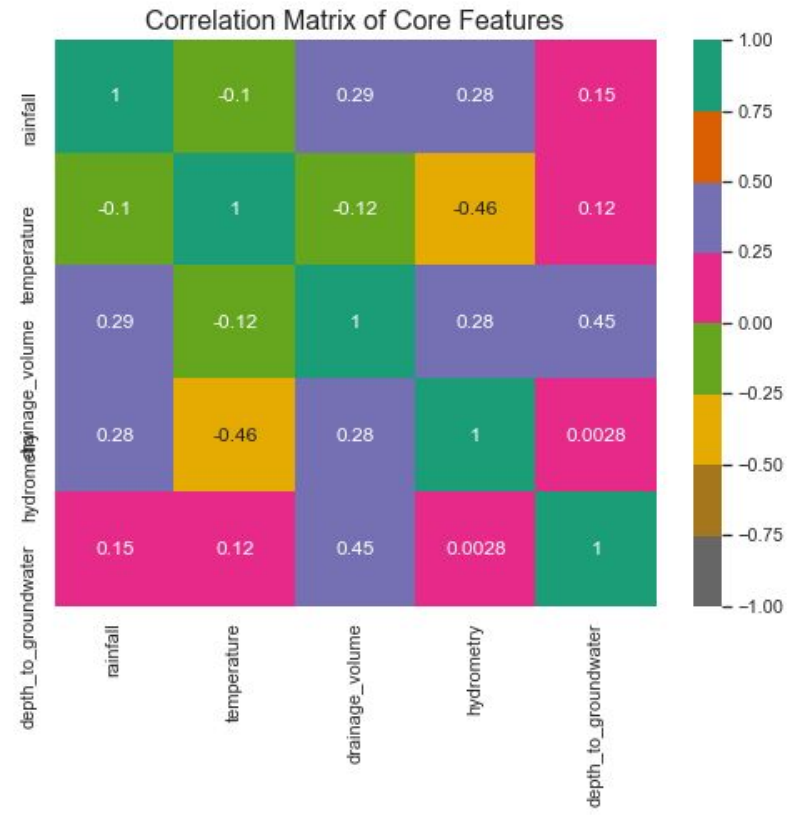
Based on the seasonal graph we can noticed:

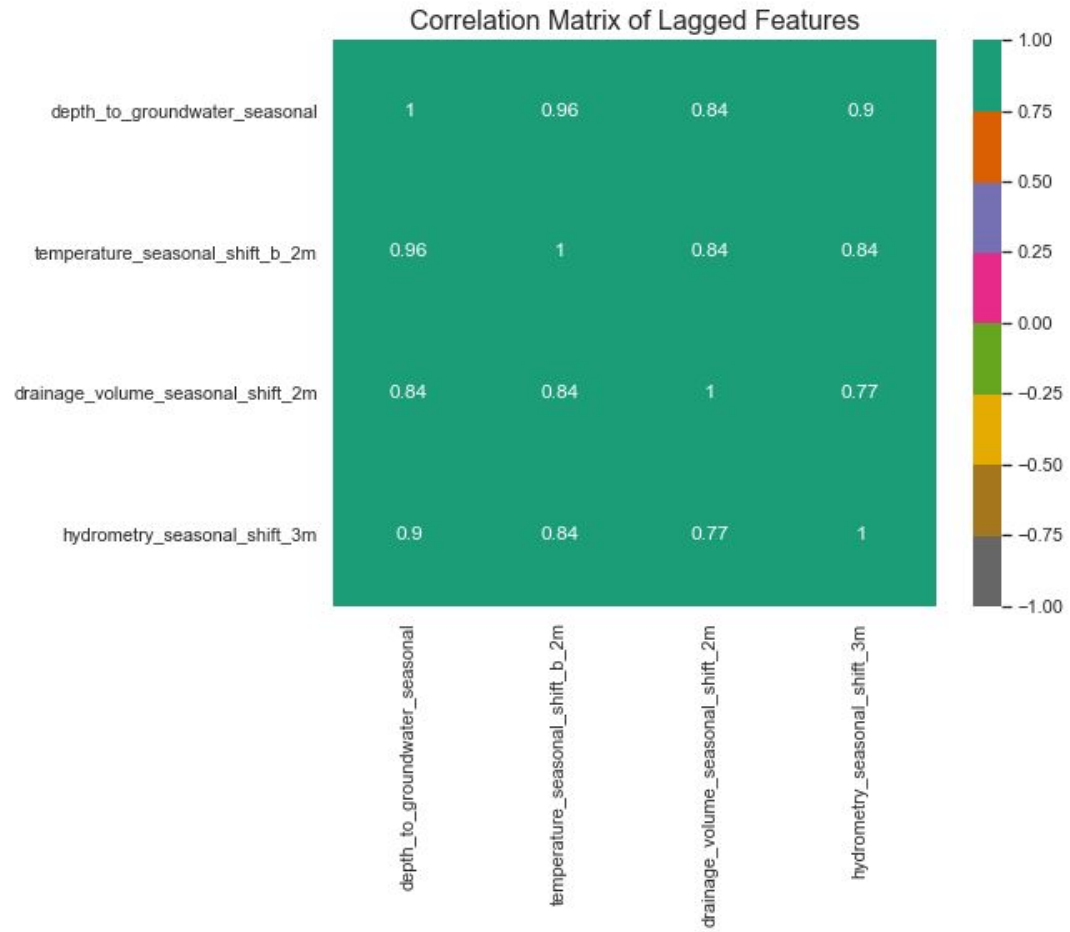
- **depth_to_groundwater:** reaches its maximum around May/June and its minimum around November
- **temperature:** reaches its maximum around August and its minimum around January
- **drainage_volume:** reaches its maximum around June and rapidly decrease in July
- **hydrometry:** reaches its maximum around February/March and its minimum around September

Seasonal Components of Features



Correlations and Heatmap





Correlations and Heatmap

On the heatmaps presented earlier I decided to plot two types of features: core features which are basic one and shifted features (which I created in previous section).

The results are represented by graph on above slide.

As we can see shifted features are highly correlated than original one.

The highest negative correlation occurs between temperature and hydrometry.

The highest positive correlation occurs between drainage_volume and depth_to_groundwater.

Autocorrelations Analysis

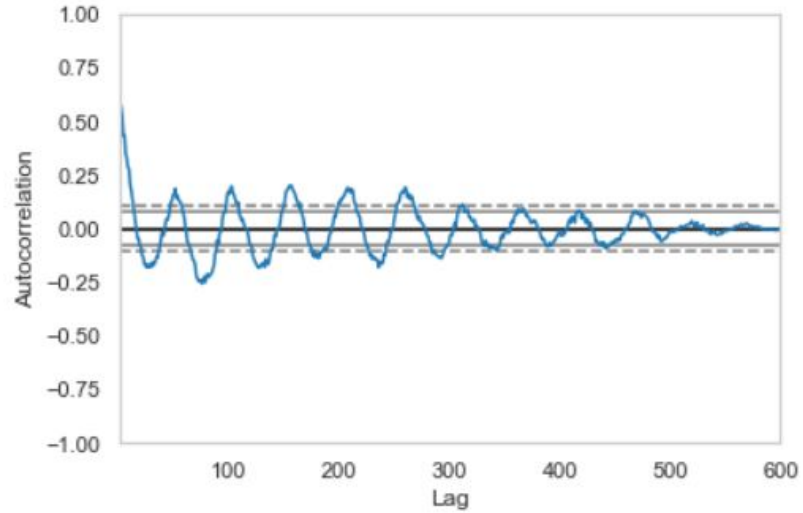
By looking at the autocorrelation function (ACF) and partial autocorrelation (PACF) plots of the differenced series, you can tentatively identify the numbers of AR and/or MA terms that are needed.

- **Autocorrelation Function (ACF):** its commonly use to detect dependence on prior observations. Its summarizes correlations between the variable and it past values. It help determine if correlations is statistical significant.
- **Partial Autocorrelation Function (PACF):** also summarizes dependence on past observations. However, its measures partial results - including all lags.

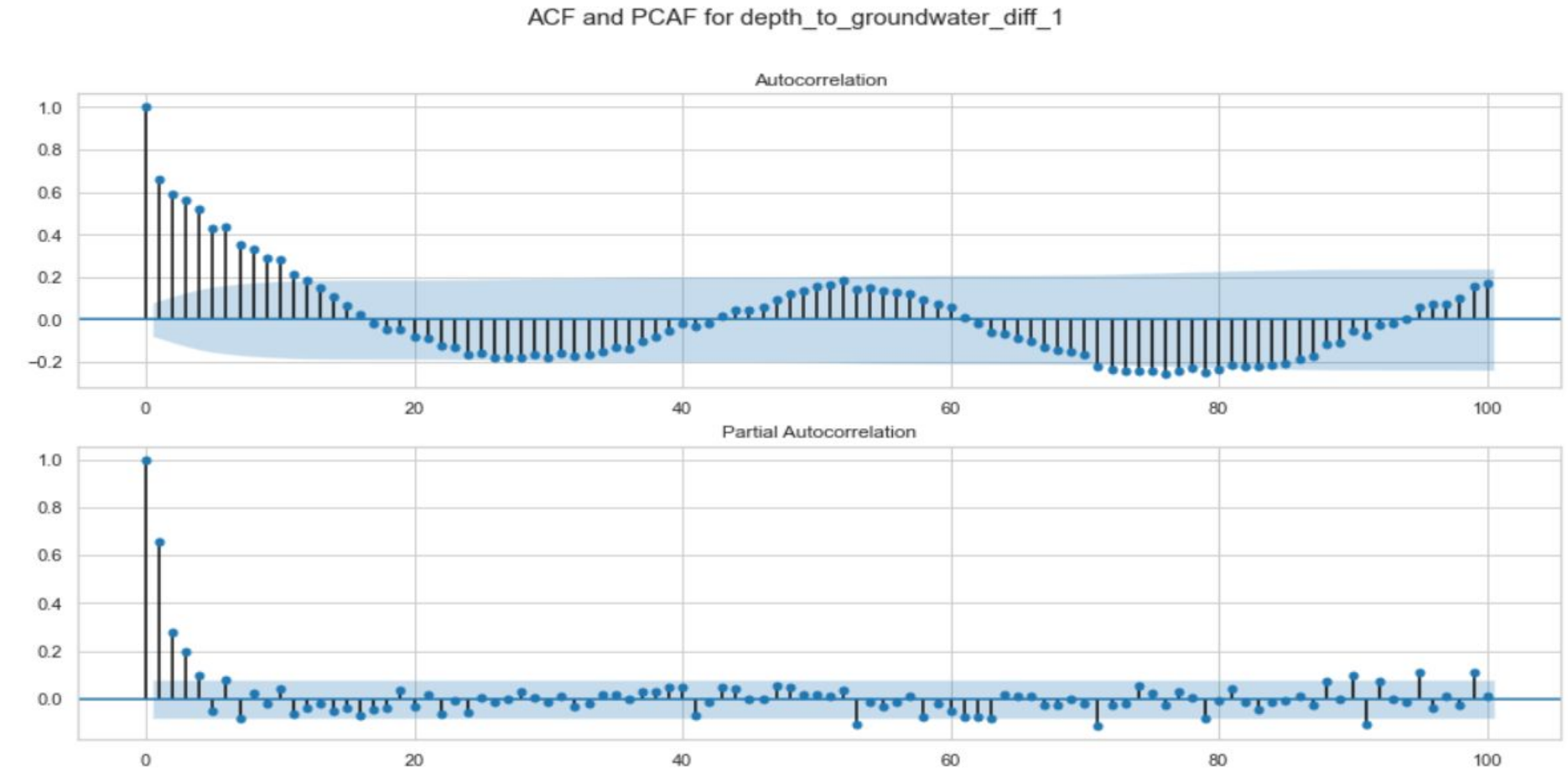
Autocorrelations Analysis

```
autocorrelation_plot(df['depth_to_groundwater_diff_1'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a7bba1f970>
```



Autocorrelations Analysis



Modeling

Prophet

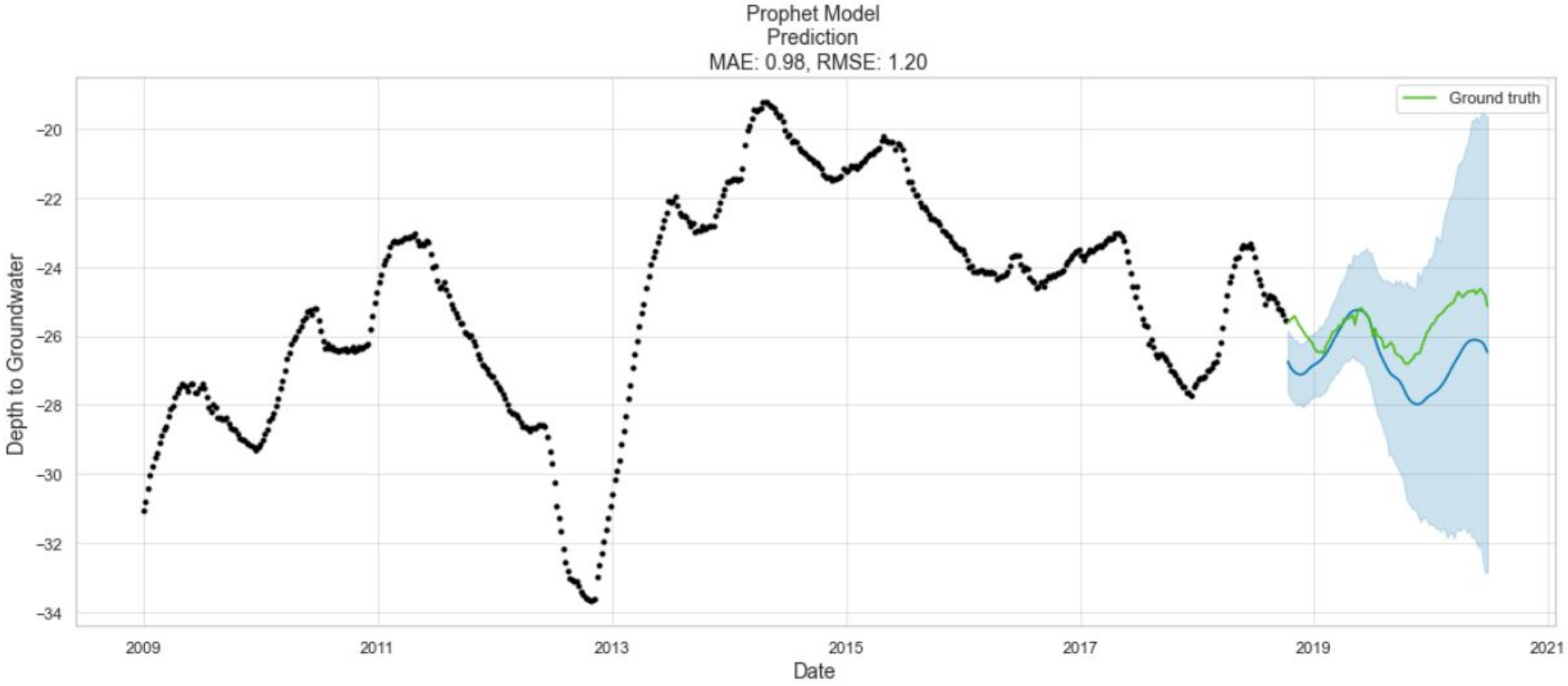
In this notebook I focused on univariate time series analysis.

- ➢ Only one variable is varying over time. For example, data collected from a sensor measuring the temperature of a room every second. Therefore, each second, you will only have a one-dimensional value, which is the temperature.
- ➢ I tried to predict **dept_to_groundwater** variable

First of all I split my dataset into train (0.85) and test (0.15) sets.

Then I fitted my model, calculated MAE and RMSE and plot the results.

Prophet

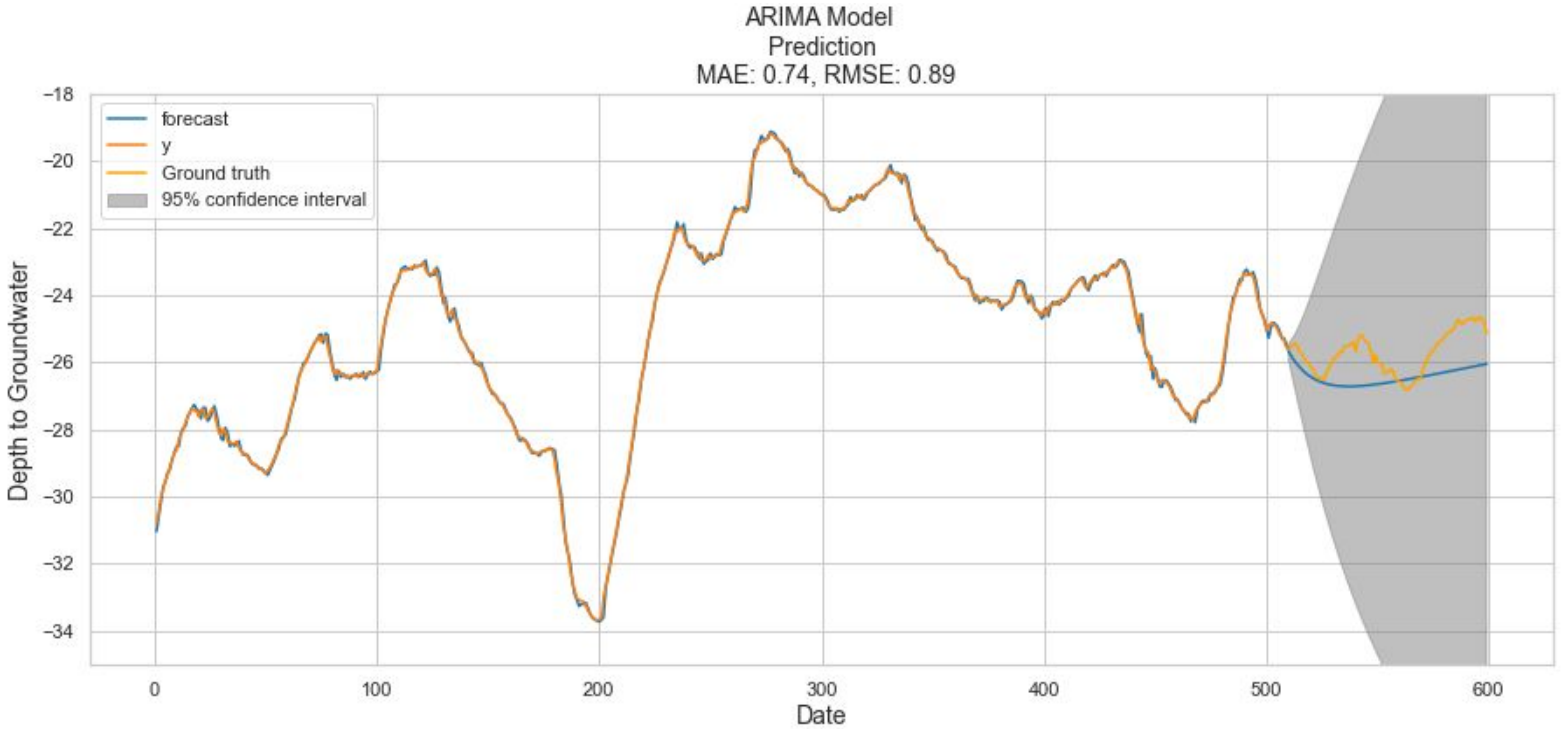

Prophet Model
Prediction
MAE: 0.98, RMSE: 1.20
Ground truth
Depth to Groundwater
−20
−22
−24
−26
−28
−30
−32
−34
2009
2011
2013
2015
2017
2019
2021
Date

ARIMA

Steps to analyze ARIMA:

- **Step 1** — Check stationarity: If a time series has a trend or seasonality component, it must be made stationary before we can use ARIMA to forecast. .
- **Step 2** — Difference: If the time series is not stationary, it needs to be stationarized through differencing. Take the first difference, then check for stationarity. Take as many differences as it takes. Make sure you check seasonal differencing as well.
- **Step 3** — Filter out a validation sample: This will be used to validate how accurate our model is. Use train test validation split to achieve this
- **Step 4** — Select AR and MA terms: Use the ACF and PACF to decide whether to include an AR term(s), MA term(s), or both.
- **Step 5** — Build the model: Build the model and set the number of periods to forecast to N (depends on your needs).
- **Step 6** — Validate model: Compare the predicted values to the actuals in the validation sample.

ARIMA


ARIMA Model
Prediction
MAE: 0.74, RMSE: 0.89
forecast
y
Ground truth
95% confidence interval
Depth to Groundwater
Date
−18
−20
−22
−24
−26
−28
−30
−32
−34
0
100
200
300
400
500
600

Auto-ARIMA

To find the best parameters for ARIMA model I decided to use **pmdarima** library. This is a statistical library designed to fill the void in Python time series analysis capabilities. I fitted the model and achieved the following results:

Auto-ARIMA validates that (1,1,1) is the best configuration for (p,d,q)

```
Best model:  ARIMA(1,1,1)(0,0,0)[0]
Total fit time: 6.154 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  510
Model:               SARIMAX(1, 1, 1)   Log Likelihood                319.497
Date:                Wed, 29 Sep 2021   AIC                         -632.995
Time:                        11:25:00   BIC                         -620.297
Sample:                             0   HQIC                        -628.016
                                  - 510
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.9195      0.021     43.765      0.000       0.878       0.961
ma.L1         -0.4885      0.037    -13.357      0.000      -0.560      -0.417
sigma2         0.0167      0.001     24.810      0.000       0.015       0.018
===================================================================================
Ljung-Box (Q):                       42.02   Jarque-Bera (JB):               185.01
Prob(Q):                              0.38   Prob(JB):                         0.00
Heteroskedasticity (H):               1.17   Skew:                             0.22
Prob(H) (two-sided):                  0.32   Kurtosis:                         5.92
===================================================================================

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

Auto-ARIMA

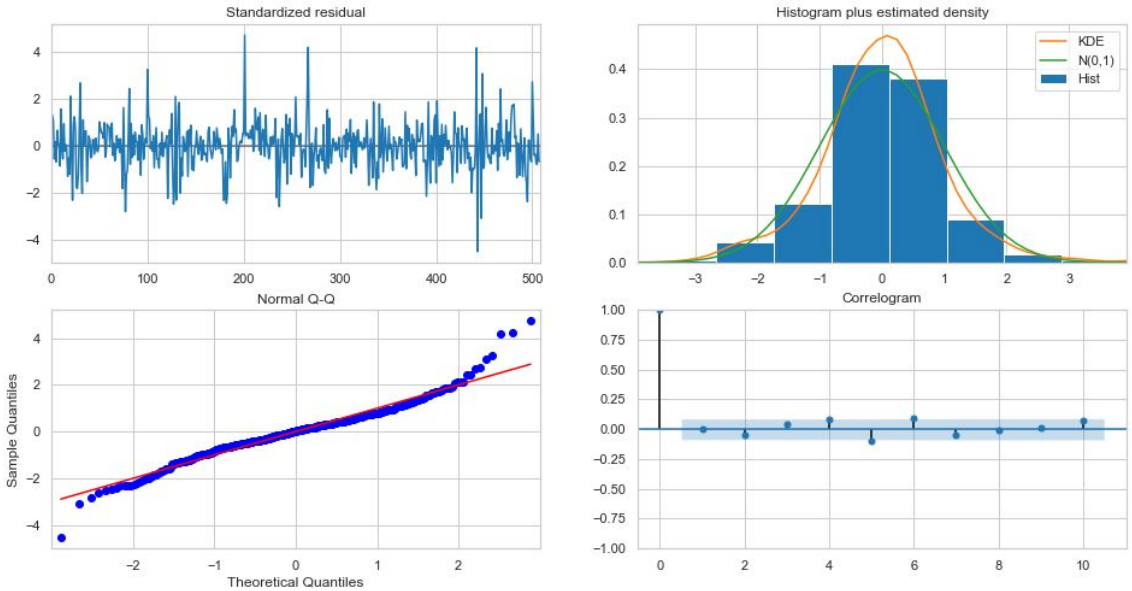
Then I used **plot_diagnostics** function and achieved the following results:

Top left: The residual errors seem to fluctuate around a mean of zero and have a uniform variance between (-4, 4).

Top Right: The density plot suggest normal distribution with mean zero.

Bottom left: The most part of the blue dots are over the red line, so it seems that the distribution in very low skewed (not skewed for me).

Bottom Right: The Correlogram, aka, ACF plot shows the residual errors are not autocorrelated.



LSTM

I used a multi-layered LSTM recurrent neural network to predict the last value of a sequence of values.

The following data pre-processing and feature engineering needed to be done before construct the LSTM model.

- ❏ Create the dataset, ensure all data is float
- ❏ Normalize the features
- ❏ Split into training and test sets
- ❏ Convert an array of values into a dataset matrix
- ❏ Reshape into X=t and Y=t+1
- ❏ Reshape input to be 3D (num_samples, num_timesteps, num_features)

LSTM

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 1, 128)            92672
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                49408
_________________________________________________________________
dense (Dense)                (None, 25)                1625
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 26
=================================================================
Total params: 143,731
Trainable params: 143,731
Non-trainable params: 0
_________________________________________________________________
```

LSTM


LSTM Model
Prediction
MAE: 0.46, RMSE: 0.52
Train Set
Prediction
Ground truth
Depth to Groundwater
−20
−22
−24
−26
−28
−30
−32
−34
2010
2012
2014
2016
2018
2020
Date

Key Findings

Conclusions

Based on information provided by time series analysis I can concluded that the best model used to predict feature **depth_to_groundwater** variable was:

★ LSTM

Limitations

The overall job was done. I performed time series analysis and tried to predict feature values of one chosen by me variable: **depth_to_groundwater**. I figured out that the best model I fitted was LSTM.

But still there are some limitations worthy to mentions:

- ★ once the dataset will be updated, the algorithms could be review
- ★ the goal was to perform some basic time series analysis and was done pretty good but still models could be further improved and parameters could be tuned to perform even better
- ★ I focused on univariate analysis but multivariate analysis is also available using Prophet ability to add regressors