CCT College Dublin

Assessment Cover Page

Module Title:	Machine Learning for Business
Assessment Title:	CA1_ML_HDip_Lvl8
Lecturer Name:	Muhammad Iqbal
Student Full Name:	Arthur Claudino Gomes de Assis
Student Number:	2023146
Assessment Due Date:	05/11/2023
Date of Submission:	10/11/2023

Declaration

By submitting this assessment, I confirm that I have read the CCT policy on Academic Misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source. I declare it to be my own work and that all material from third parties has been appropriately referenced. I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution.

1 Table of Contents

2 IMPORTING ALL THE NECESSARY LIBRARIES 3

3 BUSINESS UNDERSTANDING AND READING THE DATA 3

4 DATA CLEANING AND PREPATION 5

5 EXPLORATORY DATA ANALYSIS 7

5.1 ANALYSING SEASONALITIES 10
5.2 ANALYSING CHANGING ON VARIANCE 14

6 PREDICTING OUR DATA THROUGH DIFFERENT METHODS 15

6.1 SPLITTING THE DATA INTO TRAINING AND TESTING 15
6.2 BUILDING A BASELINE SOLUTION FOR THE FORECASTING PROBLEM 15
6.2.1 PREDICTING HISTORICAL MEAN 16
6.2.2 PREDICTING LAST YEAR MEAN 16
6.2.3 PREDICTING WITH NAÏVE SEASONAL FORECAST 16
6.2.4 VISUALIZING OUR BASELINES COMPARED WITH REAL DATA 16
6.2.5 CALCULATING THE MEAN ABSOLUTE PERCENTAGE ERROR (MAPE) 17
6.3 APPLYING HOLT-WINTER EXPONENTIAL SMOOTHING 17
6.4 PREDICTING THROUGH UNOBSERVED COMPONENT MODEL 19
6.5 PREDICTING THROUGH SARIMA 21

7 COMPARING METRICS ON THE METHODS 23

8 CONCLUSION 24

9 REFERENCES 24

10 TOTAL NUMBER OF WORDS 25

2 IMPORTING ALL THE NECESSARY LIBRARIES

In [150]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.stattools import adfuller
import pmdarima as pm
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
import statsmodels.api as sm
import statsmodels.tsa.api as smtsa
from statsmodels.tsa.holtwinters import ExponentialSmoothing as HWES
from statsmodels.tsa.holtwinters import SimpleExpSmoothing

%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

Figure 1 - All libraries imported

3 BUSINESS UNDERSTANDING AND READING THE DATA

The dataset that will be used in this project is available in the following link https://www.kaggle.com/datasets/nicholasjhana/energy-consumption-generation-prices-and-weather?rvi=1. The dataset contains 4 years of hourly electrical consumption, generation and pricing in Spain. The data contains some forecasts that stands for the predictions made by the source of the data itself.

Forecasting electrical demand is important for planning ahead how to allocate resources and to match region's needs. A demand higher the production could lead to several blackouts, whereas a production higher than the demand could lead to waste of energy and overloading in the transmission lines (Pawar, 2020). Furthermore, accurate demand forecast contribute to more efficient production and use of energy, what have a direct impact on climate change (Cerqueira, 2023).

Another importance on predicting demand of energy is planning ahead in regards to the sources of energy available to provide a region, especially those that depend on weather conditions, and might produce power irregularly. For instance, while in some moments the supply of certain sort of power, such as wind or solar, may overweight the demand of energy and some energy can be stored for later using, in other moments a higher demand or smaller production must be balanced by an extra flow from storages. The complexity of these systems leads to a fragile balance that must be carefully studied.

In this project electrical demand in Spain will be analysed through the dataset stated, exploring seasonalities on the data, and developing machine learning models typically used to solve time series problems. The models that will be explored will be Holt-Winters Exponential Smoothing, Unobserved Components Model (UCM), and SARIMA.

The main goal of this project is predicting electrical consume in a window of 2 months, what will stand for 1465 hours in the case of our dataset.

Predicting energy demand is a difficult task, once external factors might not be readily available for the modelling, like weather and economics factors that may influence the

consume of energy (Cerqueira, 2023). In our case, we will focus in auto regressive models, what means that we will try to predict future values based on past values of the data itself.

Let us start reading our data and creating dataframe to be worked on. To perform this it will be used the function pd.read_csv in Python.

After reading the dataset, the next step will be verifying the main properties of the dataset, such as shape, type of variables, and also glance at the first and last 5 rows. The next 3 images show pieces of the code with those operations.

```
In [8]:  1 #Printing the main informations of the dataset df1.
         2 df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35064 entries, 0 to 35063
Data columns (total 29 columns):
 #   Column                                       Non-Null Count  Dtype
---  ------                                       --------------  -----
 0   time                                         35064 non-null  object
 1   generation biomass                           35045 non-null  float64
 2   generation fossil brown coal/lignite         35046 non-null  float64
 3   generation fossil coal-derived gas           35046 non-null  float64
 4   generation fossil gas                        35046 non-null  float64
 5   generation fossil hard coal                  35046 non-null  float64
 6   generation fossil oil                        35045 non-null  float64
 7   generation fossil oil shale                  35046 non-null  float64
 8   generation fossil peat                       35046 non-null  float64
 9   generation geothermal                        35046 non-null  float64
 10  generation hydro pumped storage aggregated   0 non-null      float64
 11  generation hydro pumped storage consumption  35045 non-null  float64
 12  generation hydro run-of-river and poundage   35045 non-null  float64
 13  generation hydro water reservoir             35046 non-null  float64
 14  generation marine                            35045 non-null  float64
 15  generation nuclear                           35047 non-null  float64
 16  generation other                             35046 non-null  float64
 17  generation other renewable                   35046 non-null  float64
 18  generation solar                             35046 non-null  float64
 19  generation waste                             35045 non-null  float64
 20  generation wind offshore                     35046 non-null  float64
 21  generation wind onshore                      35046 non-null  float64
 22  forecast solar day ahead                     35064 non-null  float64
 23  forecast wind offshore eday ahead            0 non-null      float64
 24  forecast wind onshore day ahead              35064 non-null  float64
 25  total load forecast                          35064 non-null  float64
```

Figure 2 - Basic information about the dataset

```
In [9]:  1 #Printing the first 5 rows of the dataset df1.
         2 df1.head()
```

Out[9]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	...
0	2015-01-01 00:00:00+01:00	447.0	329.0	0.0	4844.0	4821.0	162.0	0.0	0.0	0.0	...
1	2015-01-01 01:00:00+01:00	449.0	328.0	0.0	5196.0	4755.0	158.0	0.0	0.0	0.0	...
2	2015-01-01 02:00:00+01:00	448.0	323.0	0.0	4857.0	4581.0	157.0	0.0	0.0	0.0	...
3	2015-01-01 03:00:00+01:00	438.0	254.0	0.0	4314.0	4131.0	160.0	0.0	0.0	0.0	...
4	2015-01-01 04:00:00+01:00	428.0	187.0	0.0	4130.0	3840.0	156.0	0.0	0.0	0.0	...

5 rows × 29 columns

```
In [10]:  1 #Printing the last 5 rows of the dataset df1.
          2 df1.tail()
```

Out[10]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	...
35059	2018-12-31 19:00:00+01:00	297.0	0.0	0.0	7634.0	2628.0	178.0	0.0	0.0	0.0	...
35060	2018-12-31 20:00:00+01:00	296.0	0.0	0.0	7241.0	2566.0	174.0	0.0	0.0	0.0	...
35061	2018-12-31 21:00:00+01:00	292.0	0.0	0.0	7025.0	2422.0	168.0	0.0	0.0	0.0	...
35062	2018-12-31 22:00:00+01:00	293.0	0.0	0.0	6562.0	2293.0	163.0	0.0	0.0	0.0	...
35063	2018-12-31 23:00:00+01:00	290.0	0.0	0.0	6926.0	2166.0	163.0	0.0	0.0	0.0	...

5 rows × 29 columns

Figure 3 - First and last 5 rows of the dataset.

As it is possible to be verified primarily, the dataset contains 35064 observation and 29 columns. Apart from the feature 'time', which is identified as an object, all the rest are identified as float, indicating that they are continuous variables. It is possible to identify also that 2 features are completely null, and there are some missing data among every feature, although they apparently do not sum up a big amount.

The time series on this dataset starts on January 1st, 2015 ad 00:00:00, and it finishes on December 31st, 2018, at 23:00. One important characteristic of the dataset is that the time series is given in UTC timing plus and offset, what will need to be treated before applying any machine learning model. It will be also important to convert the timeseries, which is read so far as an object, to a time series.

The variable that will be considered in this study is 'total load actual', that stands for demand of electricity in Spain.

Words counting: 590

4 DATA CLEANING AND PREPATION

As our dataset does not contain locations, we do not find necessary representing the timeseries with an UCT timing and an offset, because all the data in the dataset is referent to the same timestamp, therefore we will extract the offset from the timeseries, separating it from the timestamp the pandas function 'split'.

After splitting the dataset and expanding it, setting the offset in a separate column, we will verify the proportion of offsets found on the data. The bar graph below shows this information.

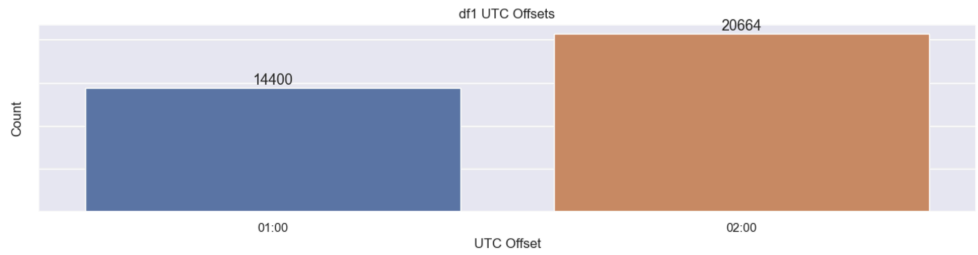

Figure 4 - Proportions of offsets in the dataset.

It is possible to verify we have a mix of Offset of 1 and 2 hours. It is desirable to carry on the analysis in the time of the time zone, rather than in the time of UCT, once trends and patterns within the dataset are likely related to time on the place where measurements were taken. Therefore, the offset will be summed up to the time stamp. Before performing this step, it is necessary to convert the timestamp, so far considered only as a string, into a time series. It is important to observe that the offset might change due to changing on the time of the time zone, for instance, due to for summer time. Therefore, once we are summing up the offset into the UCT time, we may get duplicated rows, and missing values that will need to be treated.

To perform the operation, UCT time was split once again to eliminate commas, hours were converted in integers, and this hours were summed back to the columns DateTime using the function stated below. This function has been found using chatGPT.

```python
#Applying Lambda function to sum up row by row off the dataset the DateTime column with the respective offset.
#Solution found using chatGPT.

df1['DateTime'] = df1.apply(lambda row: row['DateTime'] + pd.DateOffset(hours=row['UTC Offset']), axis=1)
```

After summing up the offset on DateTime, 8 duplicated rows were found. As the amount is very small compared to the size of the dataset, those observations were be dropped.

Once some rows were dropped, some timestamps were lost. It is necessary to assure that the timeseries is regular, or that it contains regular intervals. To assure this, the range of the dataset was considered and a new time series containing all the datapoints in the range of the dataset was created and inserted on the original dataset.

The next step performed was verifying our dataset to discard columns that might be useless in our analysis. To verify that, histograms were plot with the distribution of all the variables using the code df1.hist().

By the histogram, some variables were suspected to contain only ‘0’s, and they were tested using nunique(), what has proved that they did not contain any useful value to any analysis. The columns 'generation fossil coal-derived gas', 'generation fossil peat', 'generation geothermal', 'generation fossil oil shale', 'generation marine' and 'generation wind offshore' have only 0’s. Furthermore, 'generation hydro pumped storage aggregated' and 'forecast wind offshore eday ahead' contain only NaN. All these features were discarded together with the features 'UTC Offset', 'time' and 'Time', that have been created in the process of summing up the offset with the time series.

The next step performed was verifying and filling up null values contained on the dataset.

The amount of missing information found on the dataset is very small, something close to 25 in most of the columns missing data, and the biggest amount 36, on total load actual. Those numbers count for approximately 1% of all the observation on the data, so that the imputation method used will not highly influence the shape of the data itself. It has been chosen to impute the values only with the mean value of each column.

By the end of our cleaning and preparation steps, our dataset contains 35064 rows and 21 columns, in which 20 are continuous and 1 is the timestamp, as is can be seen on the figure below. Afterwards we can follow on to the exploratory data analysis.

```
In [189]: #Printing information of the dataset
          df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35064 entries, 0 to 35063
Data columns (total 21 columns):
 #   Column                                       Non-Null Count  Dtype
---  ------                                       --------------  -----
 0   index                                        35064 non-null  datetime64[ns]
 1   generation biomass                           35064 non-null  float64
 2   generation fossil brown coal/lignite         35064 non-null  float64
 3   generation fossil gas                        35064 non-null  float64
 4   generation fossil hard coal                  35064 non-null  float64
 5   generation fossil oil                        35064 non-null  float64
 6   generation hydro pumped storage consumption  35064 non-null  float64
 7   generation hydro run-of-river and poundage   35064 non-null  float64
 8   generation hydro water reservoir             35064 non-null  float64
 9   generation nuclear                           35064 non-null  float64
 10  generation other                             35064 non-null  float64
 11  generation other renewable                   35064 non-null  float64
 12  generation solar                             35064 non-null  float64
 13  generation waste                             35064 non-null  float64
 14  generation wind onshore                      35064 non-null  float64
 15  forecast solar day ahead                     35064 non-null  float64
 16  forecast wind onshore day ahead              35064 non-null  float64
 17  total load forecast                          35064 non-null  float64
 18  total load actual                            35064 non-null  float64
 19  price day ahead                              35064 non-null  float64
 20  price actual                                 35064 non-null  float64
dtypes: datetime64[ns](1), float64(20)
memory usage: 5.6 MB
```

Figure 5 - Basic information of the dataset after cleaning process.

Words counting: 544.

5 EXPLORATORY DATA ANALYSIS

The variable we are interested in this case is the total load actual, it represents the hourly demand of energy in Spain in MWh. Let us visualize the descriptive analysis of it.

```
In [192]:
1 ## Printing the descriptive analysis of the target variable.
2
3 df1['total load actual'].describe()
```

```
Out[192]: count    35064.000000
          mean     28698.996488
          std       4570.615019
          min      18041.000000
          25%      24812.000000
          50%      28894.000000
          75%      32186.250000
          max      41015.000000
          Name: total load actual, dtype: float64
```

Figure 6 - Descriptive analysis of electricity demand

The timeseries will be plotted now to be visualized.

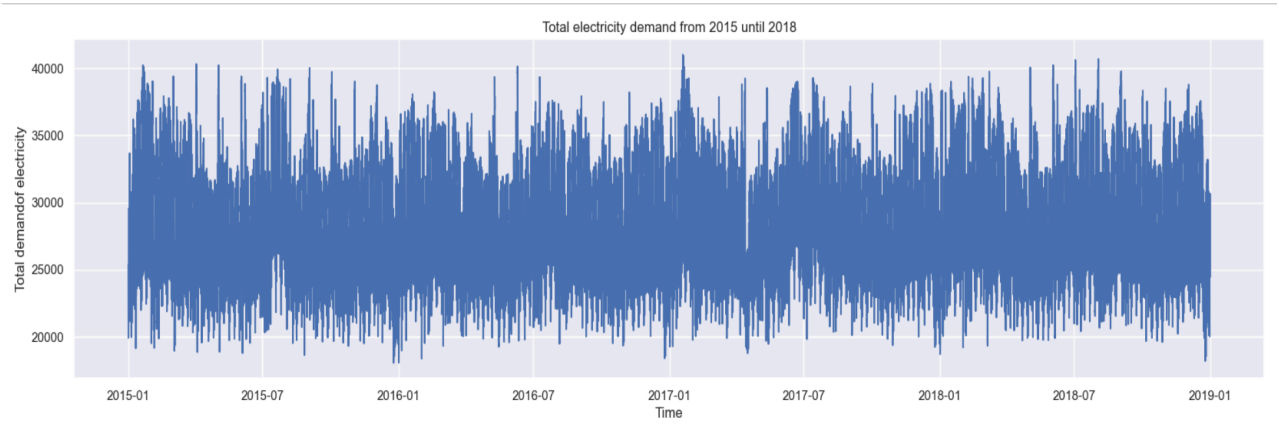

Figure 7 - Plot of the electricity demand from 2015 until 2018

As it is possible to see, the graph is very overloaded, making difficult to extract any information from it. Thus, it will be plotted the moving average based on a window of 14 days, that stands for 336 hours.

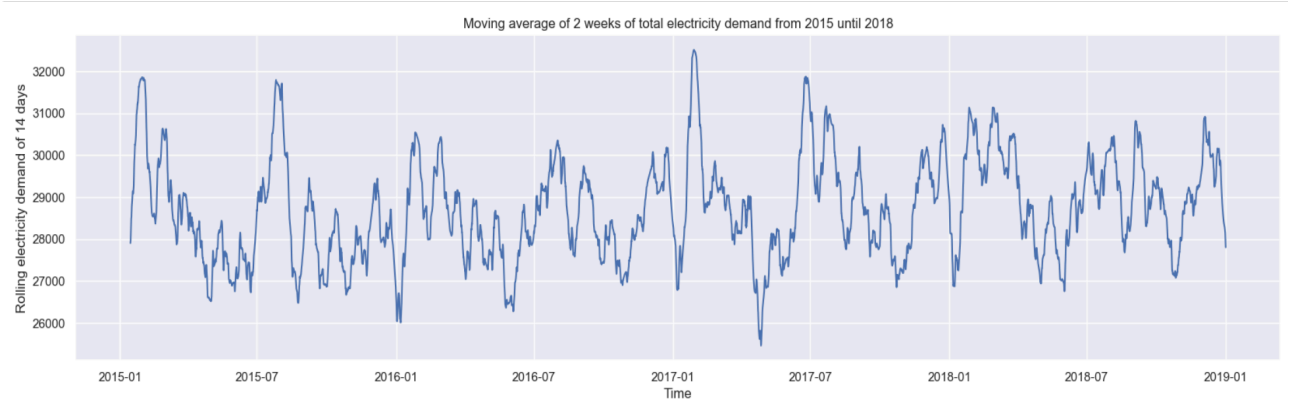

Figure 8 - Moving average of 15 days of the electricity demand.

It is possible to be verified primarily, that it seems that every year has 2 peaks of electricity consumption, one around January, and another one around July. The data will be plot in 3 different zooms, so that we can identify some patterns hidden within it. The descriptive analysis of the data shows that the total demand hold the following properties:
Min: 18041.00 MWh
Mean: 27697.95 MWh
Max: 41015.00 MWh

The average of electricity demand of Spain from 2015 until 2018 is 27697.95 MWh.

It is also possible to verify that apparently the data does not contain any strong trend, and by the graphs we can identify the complexity of the task. It is not possible to identify much of patterns beforehand on the graphs.

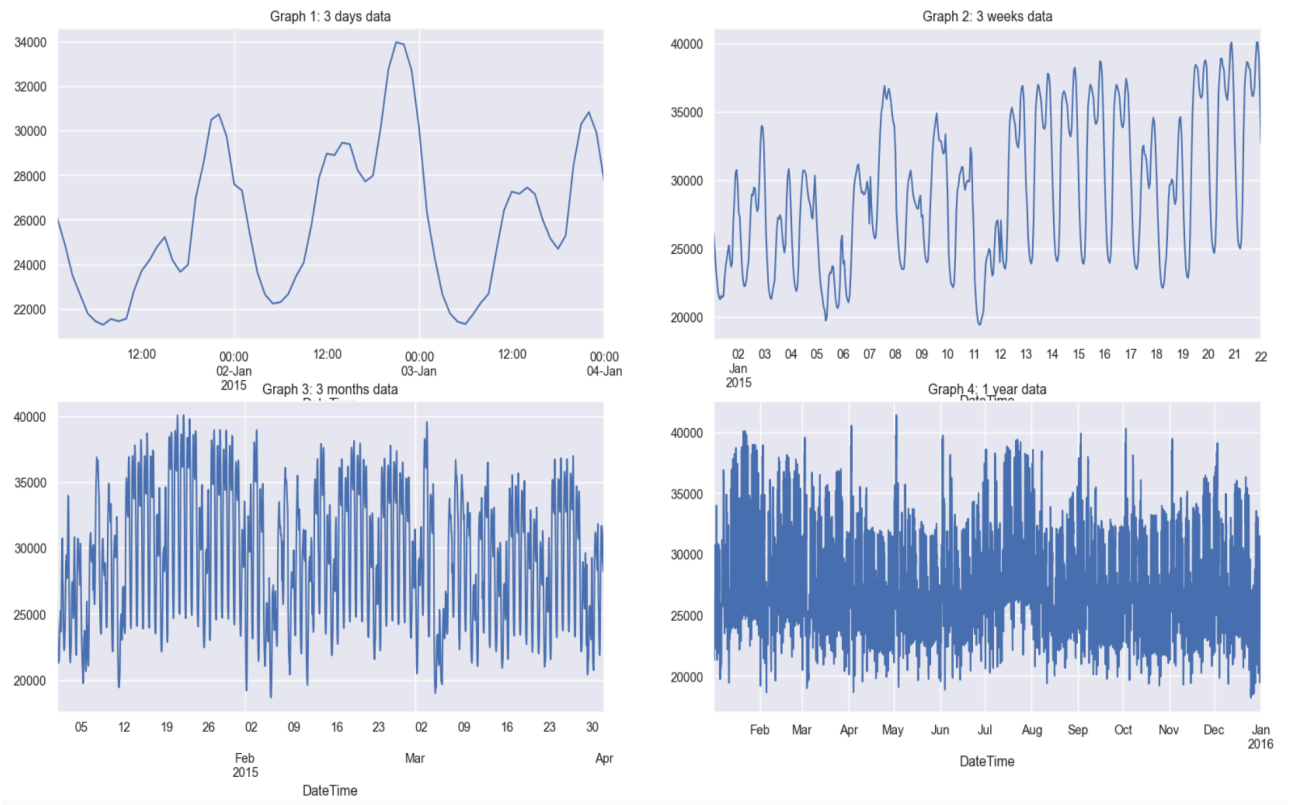

Figure 9 - Electricity demand seen in different zooms.

To make possible verifying the dataset more in depth through filters and to create better visualizations, the DateTime series was used to create new columns identifying year, month, day on the week, day on the month and hours, separated in different columns, this methodology was used by (Pawar, 2020). This makes possible to group the data in different ways.

After proceeding the task, data was averaged by hour, weekday, day on the month and month and plotted in interesting intervals, so that we can observe patterns.

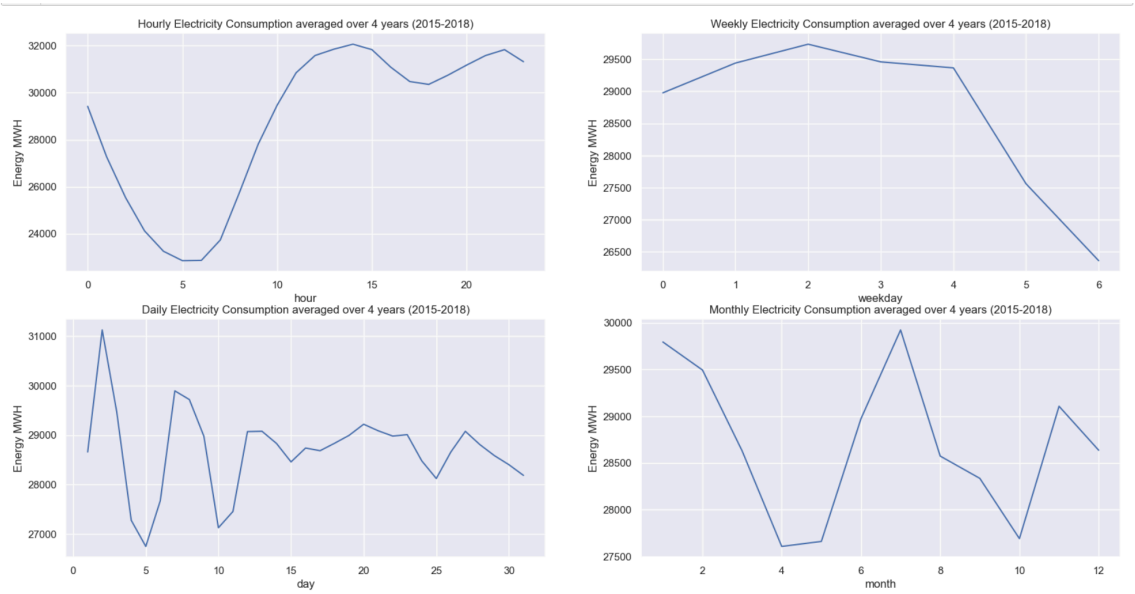

Figure 10 - Different averages on electricity demand in different windows of time.

It is possible to verify some patterns when looking to the data averaged by different periods of time. It seems that overall when looking to daily data, the demand of energy decreases from around 23 until 05, when it reaches a minimum, and then it increases reaching a local peak around 14, and another peak around 22.

Weekly data seems to be consumed somewhat regularly from day 0 until day 4, what represents from Monday until Friday, what are commonly working days, and thn it decreases through the weekend, reaching a minimum on day 6, which is Sunday. That indicates pattern within the data throughout weeks, due to differences between working days and non-working days

Daily data observed through a month window seems to behave randomly throughout the month. There is no clear pattern immediately possible to be identified.

The monthly data observed through a year window shows a peak on January, that decreases reaching a minimum in April. From April until May values are levelled, when they increase again reaching another peak in July. Values will decrease towards a minimum in October, and then increase again until January. Those periods match with the changing on the seasons, and consequently, probably with increasing and decreasing of temperature.

The next step will be verifying whether the patterns observed in the graphs above repeats throughout our dataset, what would turn them into seasonal patterns or cyclical patterns.

5.1 Analysing seasonalities

The first plot shows the demand of energy averaged by hour in each month of the year.

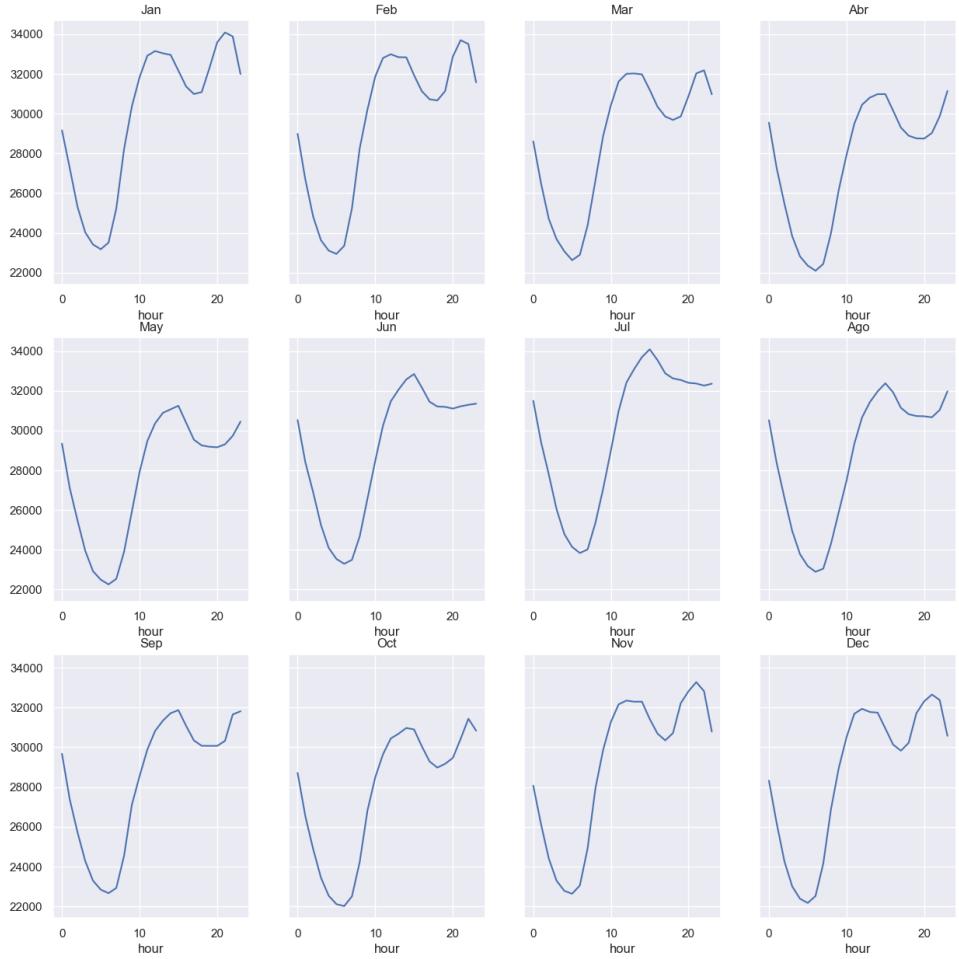

Figure 11 - Demand of electricity averaged by hour and split by month.

It is possible to see that exists an hourly correlation. When looking to the data monthly, it is clearly that a similar pattern repeats itself everyday throughout the months. It is possible to see that one more clear peak is present from May until September, usually the warmer season of the year, and also the season with longer daylight. It is also possible to notice that the consume is in a level slightly higher than the colder season, probably for a constant need for cooling facilities. On the other hand, from October until Abril, the line draws a more clear second peak of electricity demand after 8, probably for need of heating facilities, and also a higher use of electrical lights. The graphs show a daily seasonality, that will need to be removed onwards.

Let us verify now how a autocorrelation graph and partial autocorrelation show this pattern.

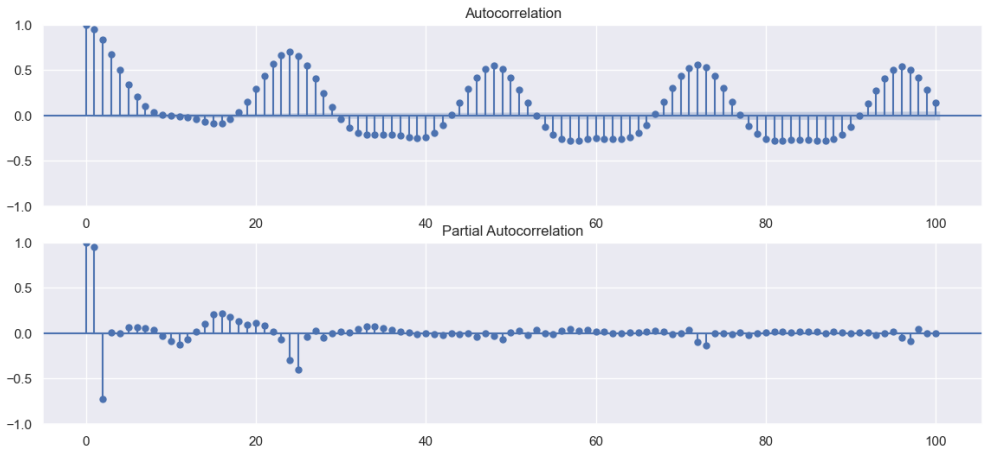

Figure 12 - ACF and PACF graphs of the data within 100 lags.

As expected, we can see that the Autocorrelation graph shows a strong correlation with past values in the lag 24, representing the hourly seasonality that has been discussed.

Let us verify now the weekly pattern seen before.

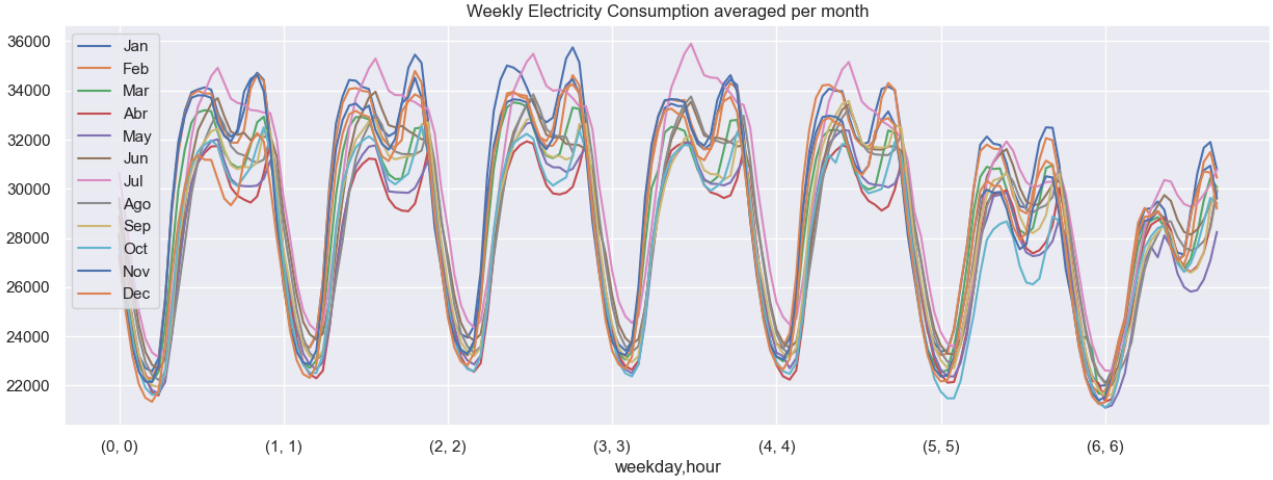

Figure 13 - Electricity demand averaged by hour and split by week day (Monday until Sunday)

As it is possible to verify from the graph above, it seems that there is also a seasonality within the days of the week, what means that at every 7 days, or something around 168 hours, the same pattern will repeat. To verify this, it will be plotted a resampled version of hour dataset, resampling it from hourly to daily series.

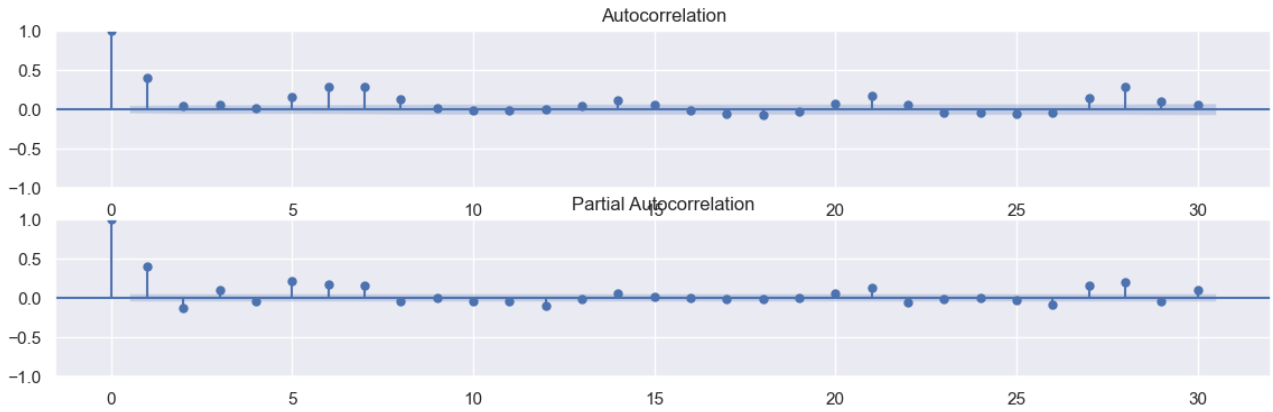

Figure 14 - ACF and PACF Graphs for dataset resampled from hours to days, within 30 lags.

As it is possible to verify, although the peaks perceived in this case are smoother than the ones we found when verifying daily data, they are still present, and they account for weekly seasonality on the data, which will also need to be removed when creating our machine learning model.

Now we will verify monthly cycles, to verify if any patterns repeats along months, or along each 30 days.

As we can verify from the graph below, we cannot verify any repetitive pattern within the days of the month. That probably happens because there is no clear reasons for patterns being created along days of the month, and they therefore seem pretty random.

The last step will be verifying seasonality along years.

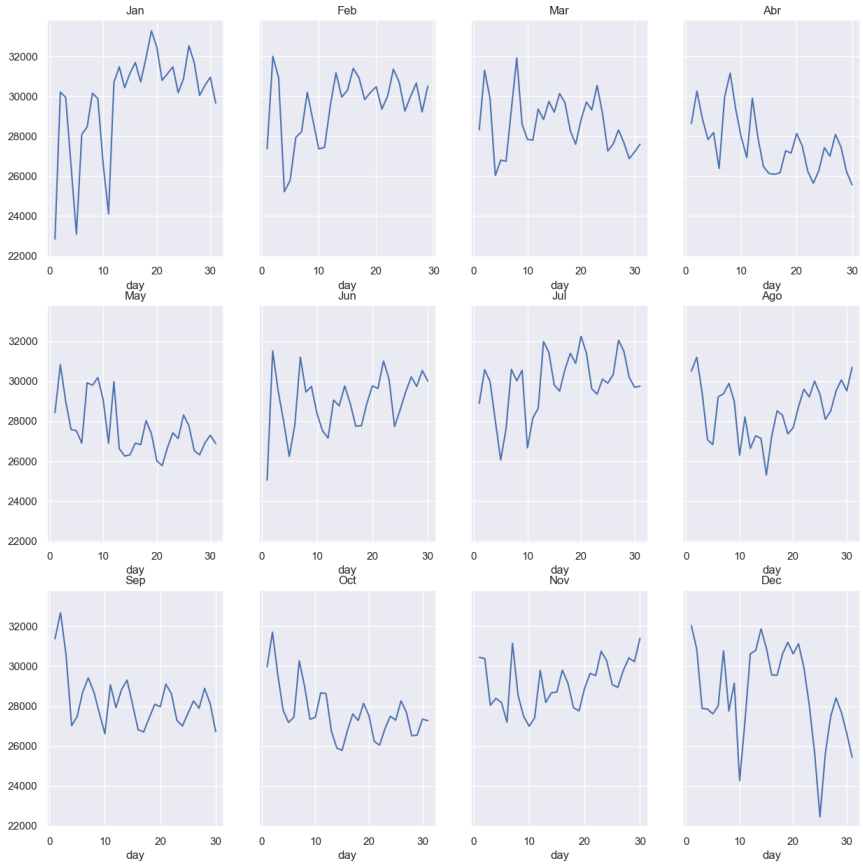

Figure 15 - Electrical demand averaged by day and split by months.

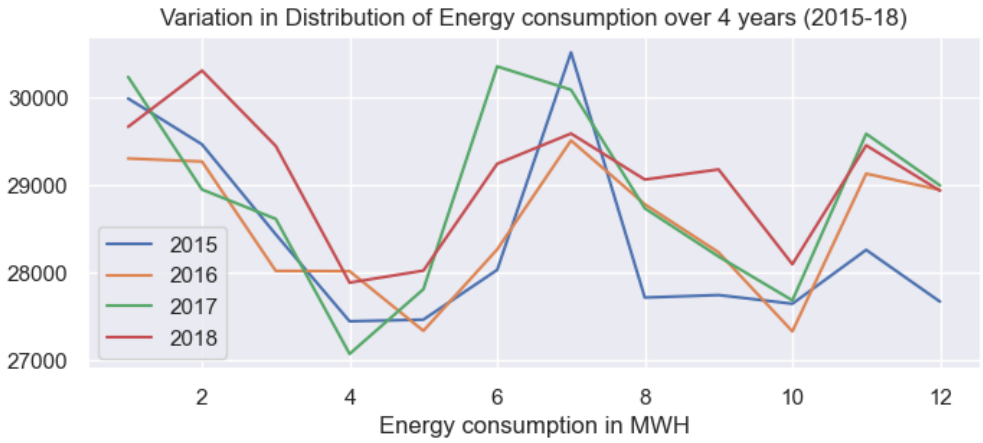

Figure 16 - Electricity demand along the year in 4 different years.

As we can verify and as we have mentioned before, energy reaches two peaks on the year, one around January, and one around July. However they have different intensities as it is possible to be verified through the distribution of the values. It is more likely that by the end of the year, the peak is slightly lower than by the middle of the year. Another important aspect to be observed, is that apparently some event has happened on 2018, by the middle of the year that reduced the consume of electricity in Spain. Nevertheless, it is possible to see through the distribution graph a high similarity, and therefore, it will be considered in this project the presence of an yearly seasonality, what in our case, represents a seasonality in periods of 8766 hours.

In conclusion, we will consider in this study, based on the exploration carried out so far, the presence of three seasonalities in the data:

1. Daily – 24 hours.
2. Weekly – 168 hours.
3. Yearly – 8766 hours.

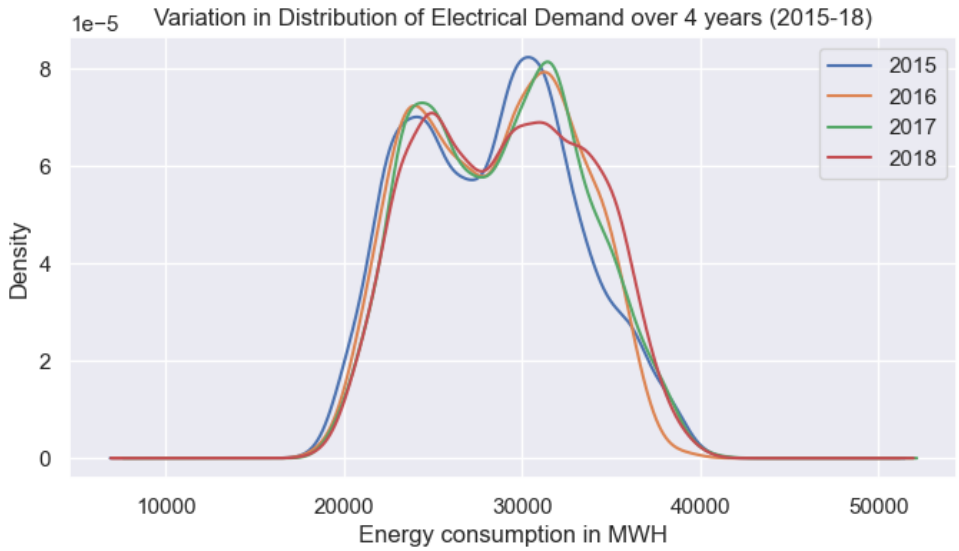

Figure 17 - Distribution of electricity demand along years in the dataset.

5.2 Analysing changing on variance

To verify changes in the variance 2 graphs will be plotted, the variance averaged in a rolling window of 24 months for the original data, and the variance averaged in a rolling window of 24 months for the moving average of 2 weeks of the dataset, visualized before.

It is possible to verify that although the variance on the dataset fluctuates, it does not increase or decrease over time. Therefore the variance will not be an issue in regards to stationarity.

Words counting: 1165

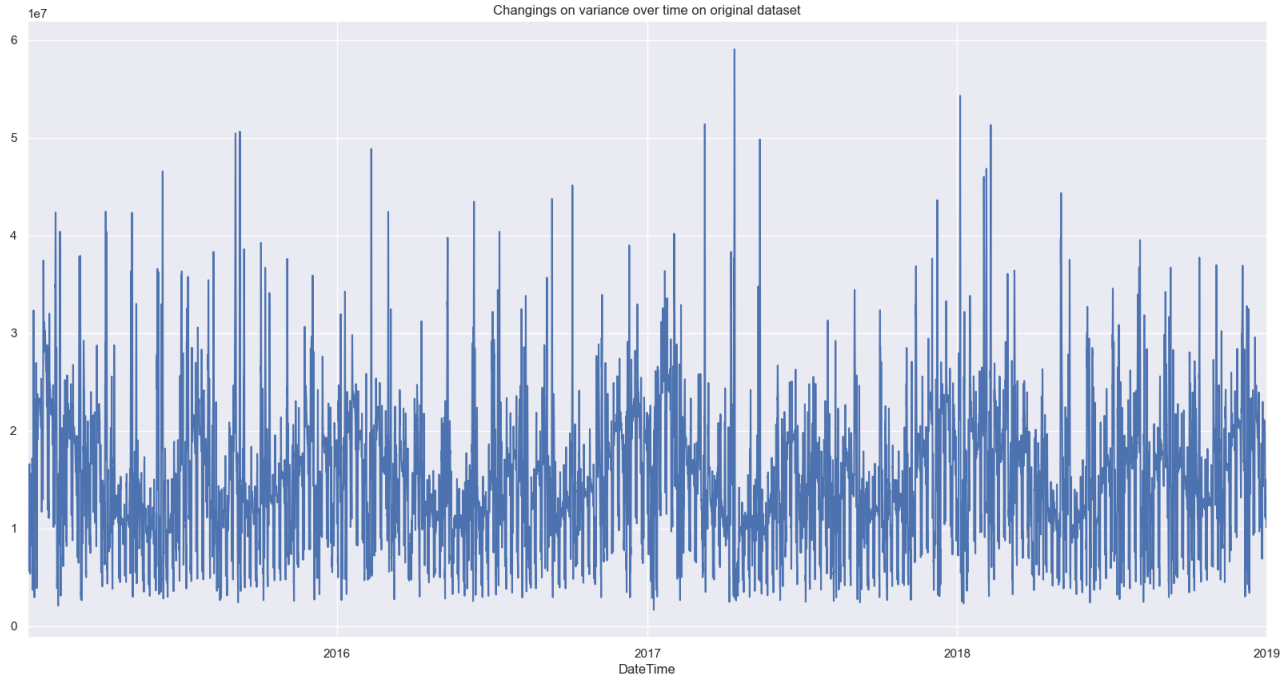

Figure 18 - Changing on variance along time.

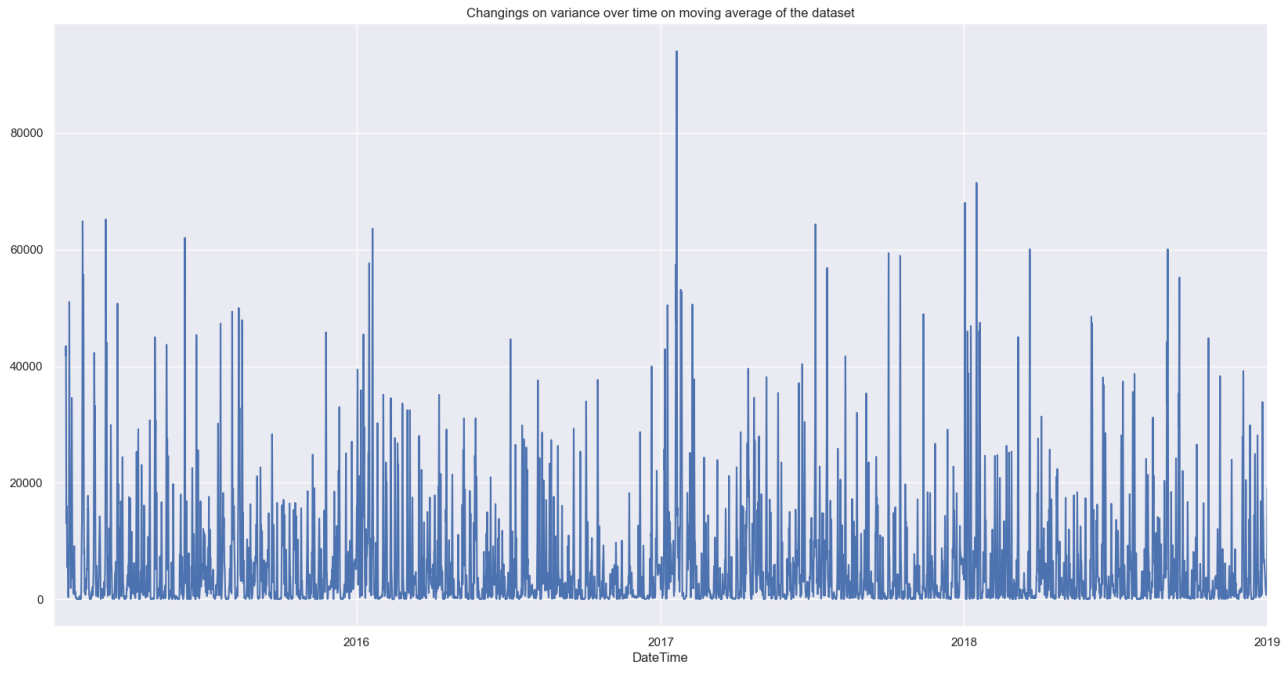

Figure 19 - Changing on variance along time observed with moving average of 2 weeks.

6 PREDICTING OUR DATA THROUGH DIFFERENT METHODS

As it was stated before, our main goal is predicting our data in a windows of 2 months, what is equivalent in hour dataset to a period of 1465 hours. To proceed this task, we will first split our dataset in training and testing, and we will create our training dataset with the same amount of observations that we want to predict, rather than choosing an arbitrary percentage. In the sequence, it will be created through trivial methods, baselines that will be used to be compared with the models built, and will be useful to analyse whether the models can outperform the trivial solutions.

The explanation of how 1465 hours were defined is below.

```
In [49]: 1 #Visualizing how many observations are present after 2018-11-01 00:00:00 to define our 2 months window.
         2
         3 df1[df1.index >= '2018-11-01 00:00:00'].shape
Out[49]: (1465, 27)
```

6.1 Splitting the data into training and testing

Now that our dataset is going to be split, only the variables that may help in our analysis will be kept. The training dataset contains all the observations until 1465 rows before the end of the dataset, whereas the training dataset contains the last 1465 of the dataset.

The first rows of each dataset are on the Figure below.

```
In [210]: 1 training.head()
```

Out[210]:

DateTime	total load actual	year	month	day	hour	weekday
2015-01-01 01:00:00	25385.0	2015	1	1	1	3
2015-01-01 02:00:00	24382.0	2015	1	1	2	3
2015-01-01 03:00:00	22734.0	2015	1	1	3	3
2015-01-01 04:00:00	21286.0	2015	1	1	4	3
2015-01-01 05:00:00	20264.0	2015	1	1	5	3

```
In [211]: 1 testing.head()
```

Out[211]:

DateTime	total load actual	year	month	day	hour	weekday
2018-11-01 00:00:00	28361.0	2018	11	1	0	3
2018-11-01 01:00:00	28830.0	2018	11	1	1	3
2018-11-01 02:00:00	26646.0	2018	11	1	2	3
2018-11-01 03:00:00	25233.0	2018	11	1	3	3
2018-11-01 04:00:00	24749.0	2018	11	1	4	3

Figure 20 - First and last 5 rows of our training and testing dataset.

6.2 Building a baseline solution for the forecasting problem

When trying to build a model to predict values on the future, it is important to build a baseline that composes trivial solutions of the problem. Some models are the historic mean of the variable, the mean of the last year of the variable, and repeating the same values for the

same period on the past as prediction. Those solutions will provide values to compare with the model show if the model can perform at least better than the trivial solutions.

6.2.1 Predicting historical mean

In [53]:
```
# Calculate the average of the training data

historical_mean = np.mean(training['total load actual'])

# Display the mean value of the training data
historical_mean
```

Out[53]: 28677.712992811088

It is possible to verify that from January 2015 until October 2018, Spain has been demanding a total of 28677.71 MWh of electrical energy per hour. This is already one possible forecast for the demand of the country, however it does not consider any seasonality on the demand of the country, such as it had been discussed before. We will store this result into our testing dataset to calculate its efficiency later.

6.2.2 Predicting last year mean

A window of 1 year is equivalent to 8766, considering that each year has 365.25 days.

In [55]:
```
#Predicting the data of the last year

last_year_mean = np.mean(training['total load actual'][-8759:])

last_year_mean
```

Out[55]: 29081.74368979636

6.2.3 Predicting with Naïve seasonal forecast

Naive seasonal takes exactly the same period we want to predict backwards, and replicates the values of the same time series to use as a prediction.

In [57]:
```
# Getting the values from the last cycle with the same range of our testing dataset.

testing.loc[:, 'Naive_seasonal'] = training['total load actual'][-1465:].values
```

6.2.4 Visualizing our baselines compared with real data

The figure below show graphically the baselines created overlapping with the real data.

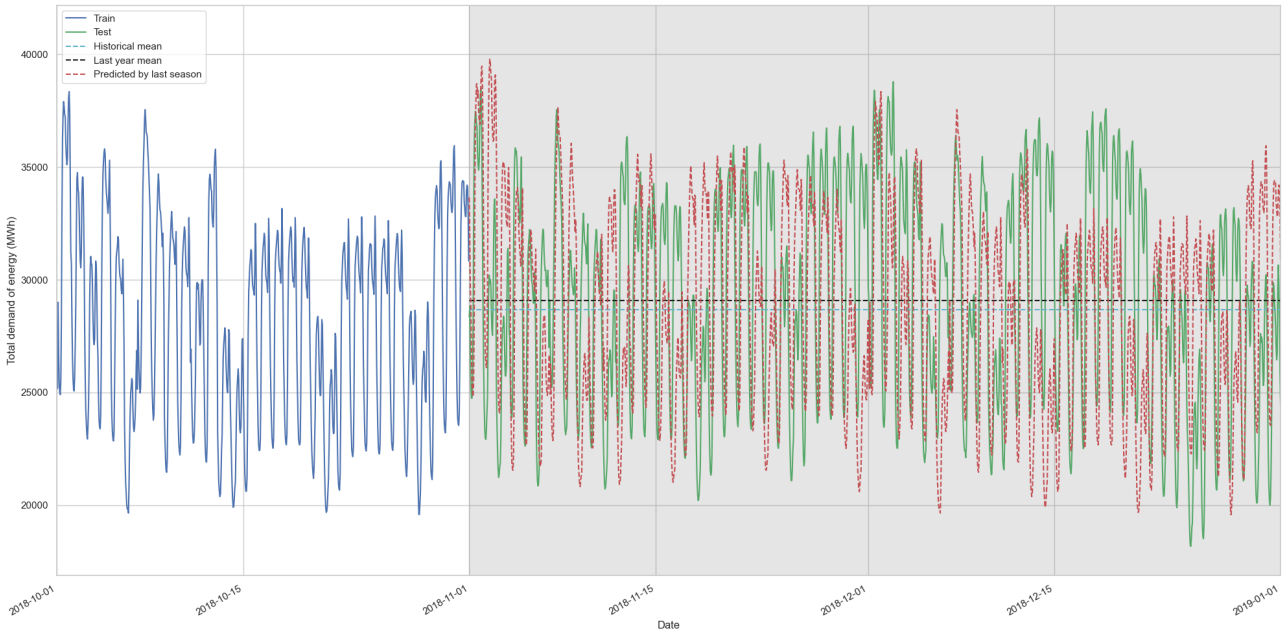

Figure 21 - Baselines created overlapping with original data.

6.2.5 Calculating the Mean Absolute Percentage Error (MAPE)

It will be used in this project the mean absolute percentage error (MAPE) as a metric for the baselines defined. To calculate the MAPE, we subtract our prediction from the actual value, and divide by the actual value, point by point, and afterwards calculate the mean of all the values calculated. To perform the method we will define a function.

The graph below shows the MAPE calculated for the three baselines drawn.

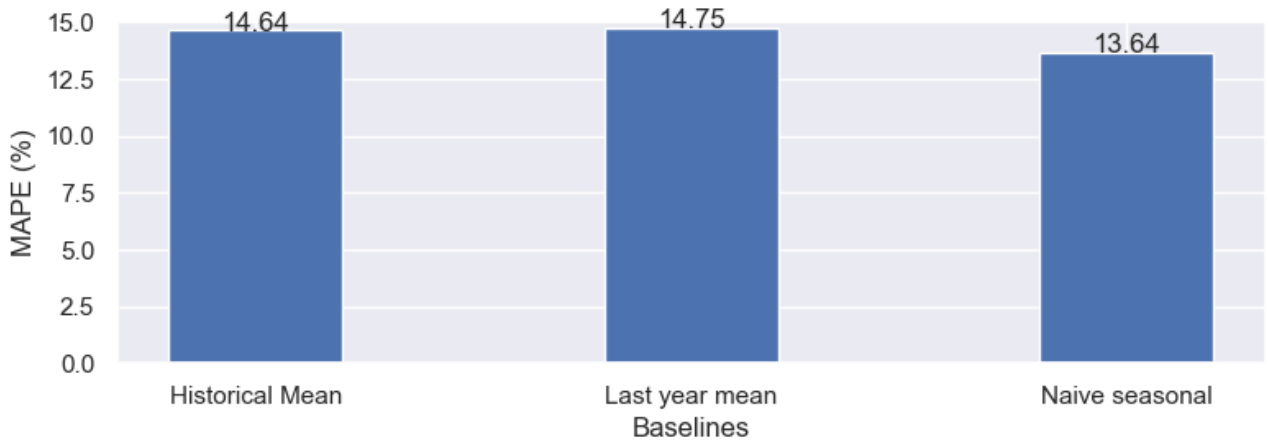

Figure 22 - Mean absolute error percentage of baselines created.

As we can see, our baselines are not very accurate, and they cannot be used as models to predict energy, once they present error over 10%, what is an enormous values given the important of energy production and distribution. As it follows, we will try to tackle the problem using some different models.

6.3 Applying Holt-Winter Exponential Smoothing

The first method that will be tried in our dataset is the Holt-Winter Exponential Smoothing. The method is able to predict recognize seasonality and to model the data in terms of level, trend, seasonality and noise. It is also able to be perform in different ways, in which the trend

is additive or multiplicative to the level, and seasonality is additive or multiplicative to the trend. The method is only able to be imputed with one seasonality though.

Although we have tried different combinations, it will be included only the best combination in this report. For such, it was considered m=24, accounting for the daily seasonality, the trend as an additive term to current level, and the seasonality as a multiplicative term to trend.

We achieve the following prediction and metrics using these parameters.

```
In [219]:
#Printing the metrics for the method

print('Mean Absolute Error: (MAE)', mean_absolute_error(testing['total load actual'], TripleHWSE_predictions))
print('Mean Squared Error: (MSE)', mean_squared_error(testing['total load actual'], TripleHWSE_predictions))
print('R Squared Score:', r2_score(testing['total load actual'], TripleHWSE_predictions))
```

```
Mean Absolute Error: (MAE) 2916.9272504687915
Mean Squared Error: (MSE) 13079513.549383383
R Squared Score: 0.41970481571559237
```

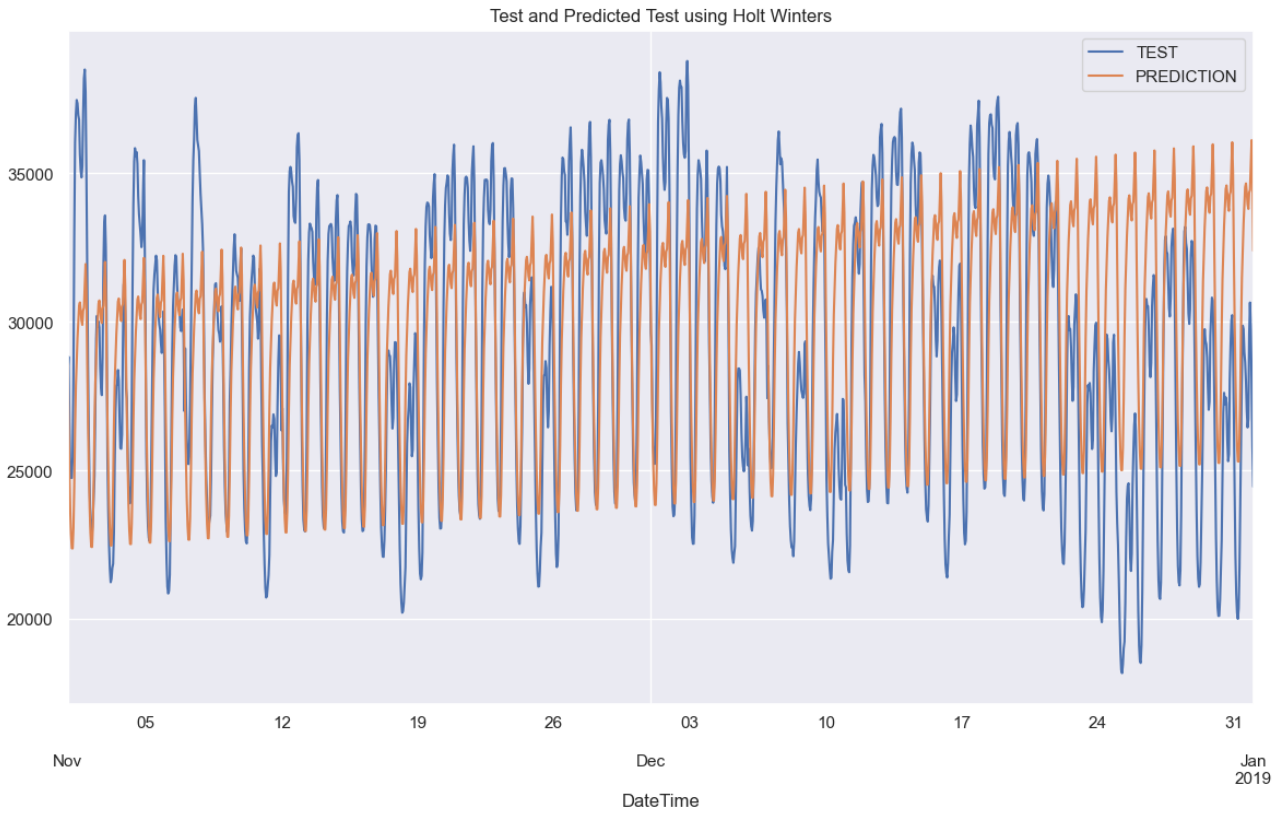

Figure 23 - Predictions generated with Holt-Winter Exponential Smoothing method.

It is interesting to observe that, although the model seems to predict the data to certain extent, it rather explodes increasing constantly along the interval selected, than predicts the data precisely. In the windows we are observing, the model might perform somewhat reasonably, but we cannot consider that the model could be generalized to longer forecasts.

When evaluating MAPE for the method we get the value 10.33%. The MAPE shows small improvement compared to our baselines, around 3% compared with Naive Seasonal.

However, the fact that the model explodes could lead to worst results compared with our baseline in case the window of prediction was longer.

6.4 Predicting through Unobserved Component Model

The method of Unobserved Components was performed by (TOTH, 2021) achieving consistent results in a dataset with similar properties of the one is being treated in this project, and therefore, we will use the same method to analyse how well it performs in our case. The model will be applied as it follows. All the parameters used to create the model were presented by (TOTH, 2021), however, the values chosen for each one of them were found by multiple attempts, observing and trying to increase the value of R squared.

In [120]:
```
#Unobserved Components model definition
model_UC1 = sm.tsa.UnobservedComponents(training['total load actual'],
                                        trend=True,
                                        level=False,
                                        irregular=False,
                                        stochastic_level = False,
                                        stochastic_trend = False,
                                        stochastic_freq_seasonal = [False, False, False],
                                        freq_seasonal=[{'period': 24, 'harmonics': 3},
                                                       {'period': 168, 'harmonics': 2},
                                                       {'period': 8766, 'harmonics': 1}])
#fitting model to train data
model_UC1res = model_UC1.fit()

#printing statsmodels summary for model
print(model_UC1res.summary())

```

Figure 24 - Parameters used to create the Unobserved Component model used to fit the data.

The method was applied according to the code above. The metric found with the methods, as well as the residual diagnosis are in the figures below.

In [126]:
```
print('Mean Absolute Error: (MAE)', mean_absolute_error(testing['total load actual'], forecast_UC1res))
print('Mean Square Error: (MSE)', mean_squared_error(testing['total load actual'], forecast_UC1res))
print('R Squared Score:', r2_score(testing['total load actual'], forecast_UC1res))
```

```
Mean Absolute Error: (MAE) 2324.0530623215113
Mean Square Error: (MSE) 9064405.040911071
R Squared Score: 0.5978420318321409
```

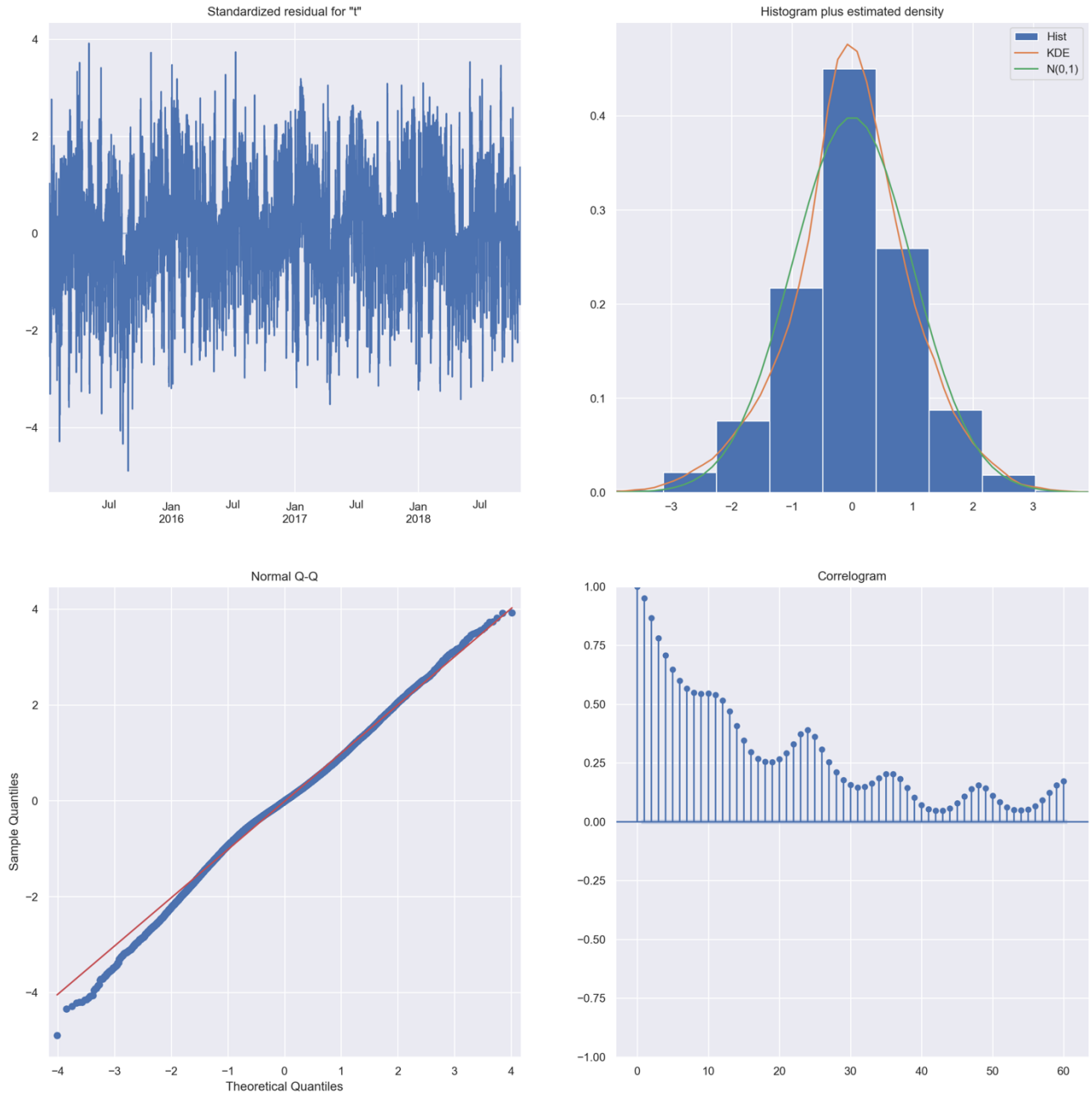

Figure 25 - Residual diagnosis of Unobserved Components model.

The value of MAPE obtained with the method was 8.09%.
As we can verify from our indicators and graphs, we could reach a much better result when using the method suggested. We could reach the highest value of R squared, reaching slightly less than 60%. MAPE has been reduced by another 2% compared to Holt-Winters, and around 5% compared with our best performing baseline. When we observe our results through its diagnosis, we can verify:

1- Apparently there is still a seasonality in our residuals, what is not ideal to the model, and might be responsible for our still far from ideal performance. However, that might be caused by external factors that we cannot predict, such as temperature and other weather variables.

2- Distribution of the residuals if fairly normal, what is a positive point of the model.

3- The Q-Q graph shows the blue spots distributed closely to the red line that represents an ideal normal distribution, deviating mainly on the beginning, what is another sign that the residuals are normally distributed.

4- The correlogram tends to 0, but it still shows a high correlation with lagged values on the residuals, what might again because by external factors.

Below it will be plotted the decomposition of the serie just created with the model, showing the trending, seasonal, and residual components.

The figure below shows the prediction overlapped with the real values on the dataset.

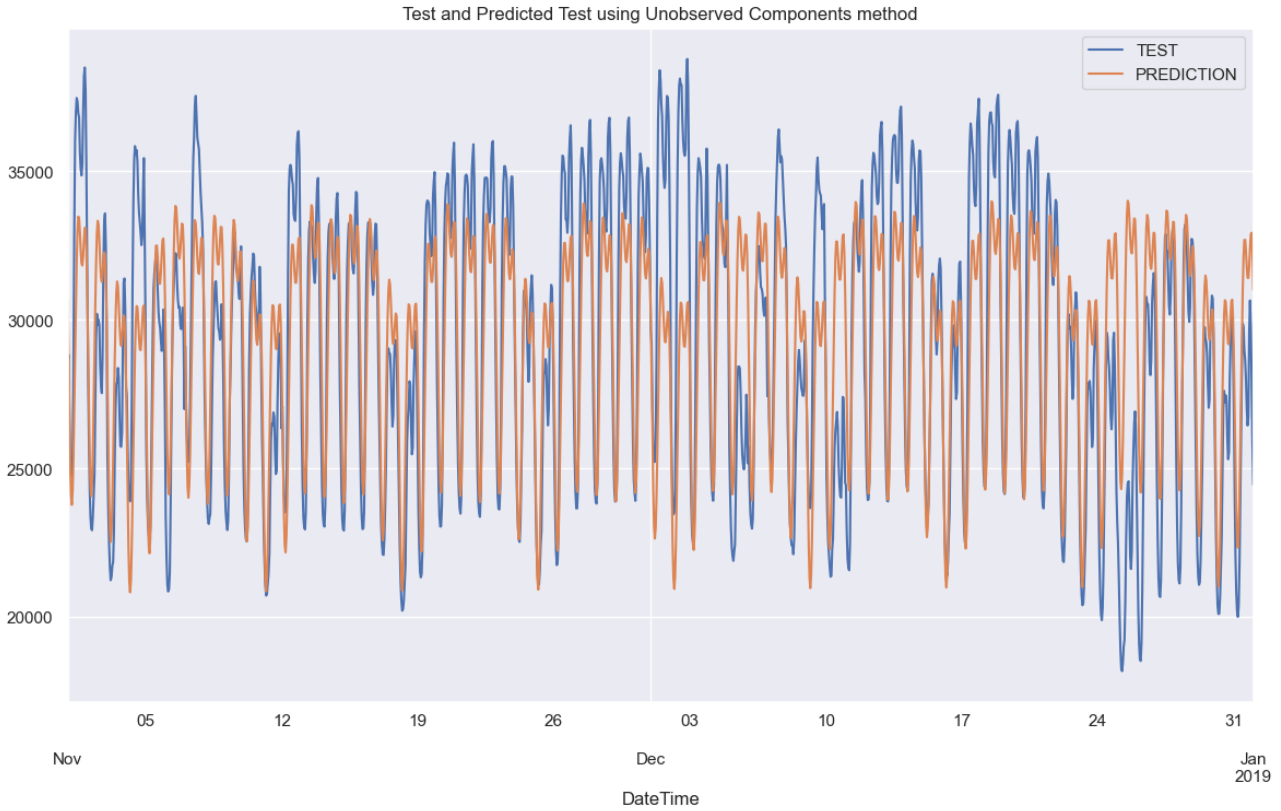

Figure 26 - Predictions generated with Unobserved Components Model.

It is possible to see that the model fairly predicts the data. The method decomposes the dataset in a trend, seasonal components, in our case three, and noise. It models these three components and sums them up to generate predictions. It is possible to see that the method predicts the overall behaviour of the electrical demand, however, unexpected cycles are still let out by the method.

6.5 Predicting through SARIMA

(Pawar, 2020) states that SARIMAX cannot handle multiple seasonalities, are more time consuming and require a lot of historical data. However, we will try to forecast electrical demand taking into consideration that the most important seasonality on the dataset is the daily one, and trying to fit the model only differencing on this seasonality. The method SARIMA is composed by two components of the lag order, two component for the degree of differencing, and two component for the moving average window, one for the seasonal component of our dataset and one for the non-seasonal component.

It will be applied first the Augmented Dickey Fuller test to verify the necessity of differencing our data. The test is a hypothesis test that takes the null hypothesis of non-stationarity of the data. Therefore, if p-value is higher than the threshold, that will be defined as 5%, we fail to reject the null hypothesis, and the data is non-stationary. If p-value is lower than the threshold, we reject the null hypothesis, hence the data in stationary.

In [134]:
```
# Display the outcomes of Dicky Fuller test
print(f'ADF Statistic: {ad_fuller_result[0]}')
print(f'p-value: {ad_fuller_result[1]}')
```

```
ADF Statistic: -21.701366778990288
p-value: 0.0
```

As we can verify, p-value < 0.05, therefore we reject the null hypothesis, and our data should be considered stationary.

As we have 6 parameters to discover on our SARIMA model, we will first verify the parameters to ARIMA model, and then in the sequence we will try to verify the remaining three to SARIMA.

In [90]:
```
aicVal=[]
for d in range(1,3):
    for ari in range(0, 3):
        for maj in range(0,3):
            try:
                arima_obj = ARIMA(df1['total load actual'].tolist(), order=(ari,d,maj))
                arima_obj_fit=arima_obj.fit()
                aicVal.append([ari, d, maj, arima_obj_fit.aic])
            except ValueError:
                pass
print(aicVal)
```

```
[[0, 1, 0, 609797.9939073379], [0, 1, 1, 593673.0885364002], [0, 1, 2, 589234.2532689325], [1, 1, 0, 588268.5227877
43], [1, 1, 1, 587735.112256179], [1, 1, 2, 587555.7633461403], [2, 1, 0, 587568.7698907919], [2, 1, 1, 583057.8444
057746], [2, 1, 2, 583057.0871907509], [0, 2, 0, 594280.9786252663], [0, 2, 1, 594208.3056387977], [0, 2, 2, 59390
0.0791996022], [1, 2, 0, 594220.2159219974], [1, 2, 1, 588158.2430051655], [1, 2, 2, 587621.9893895506], [2, 2, 0,
594006.577859355], [2, 2, 1, 587455.4855124742], [2, 2, 2, 582938.5727899147]]
```

Figure 27 - Loop to calculate parameters on ARIMA that minimizes AIC value.

We can verify that the best order for ARIMA in our dataset is 2,2,2. As it follow, it will be tried to fit a SARIMA model with differencing seasonality on lag 24.

In [112]:
```
print('Mean Absolute Error: (MAE)', mean_absolute_error(testing['total load actual'], forecast_values_SARIMA.pre
print('Mean Square Error: (MSE)', mean_squared_error(testing['total load actual'], forecast_values_SARIMA.predic
print('R Squared Score:', r2_score(testing['total load actual'], forecast_values_SARIMA.predicted_mean))
```

```
Mean Absolute Error: (MAE) 9030.174208343758
Mean Square Error: (MSE) 101904299.12653007
R Squared Score: -3.5211600429735395
```

Figure 28 - Metrics found from SARIMA Model.

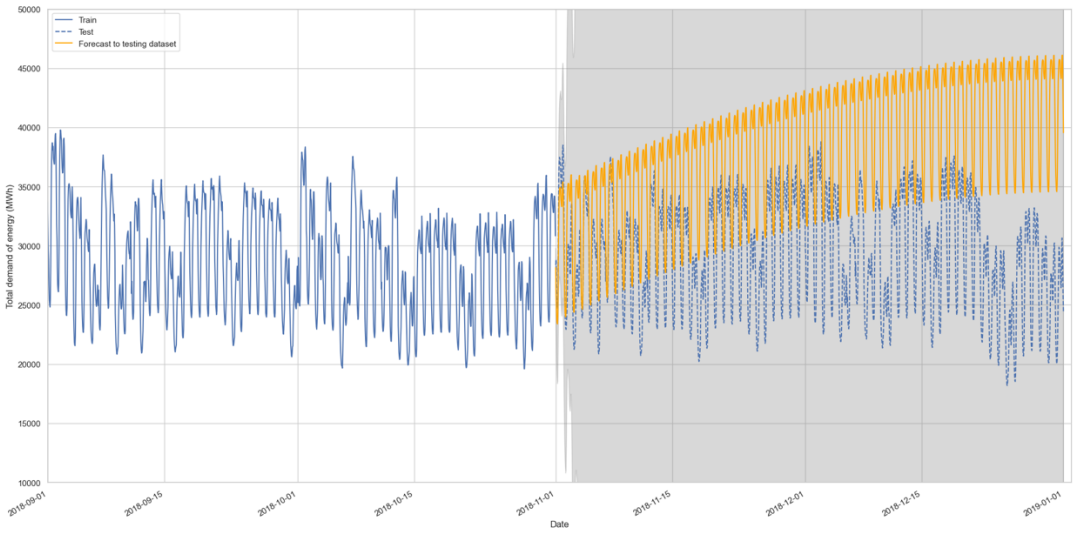

Figure 29 - Predictions generated with SARIMA model.

As we can observe from the Method SARIMA, it did not perform well as it was already expected. From the plot of diagnosis, we can verify that all the indicators highly deviate from the ideal. The histogram do not show a distribution even close to the idea, and que Q-Q chat highly deviates from the baseline. All the metrics used to analyse models show scores far away from ideal, and by comparing the MAPE, that has got value of 32.72%, is possible to verify that the model perform even much worse than our stablished baselines.

It has been presented by (Pawar, 2020) the use of SARIMAX, using a decomposed version of the dataset as external variables, an use of the method with a fair accuracy. However, the same study highlights the drawbacks of the method. In our project, it took quite a long time to perform only one run of the method on the dataset. Furthermore, references show that the method is by far outperformed by methods more specific to this application. Therefore, we will no longer explore it.

Words counting: 1601.

7 Comparing Metrics on the methods

After fitting the models, we will briefly expose some metrics that are important to our models in comparative ways. Our best baseline before applying methods to get predictions to the next 2 months was 13.64% far from the real values, whereas our best model, fitted using a method names Unobserved Components, was 8.08%.

We will plot bellow a graph containing all the results obtained of MAPE for each method.

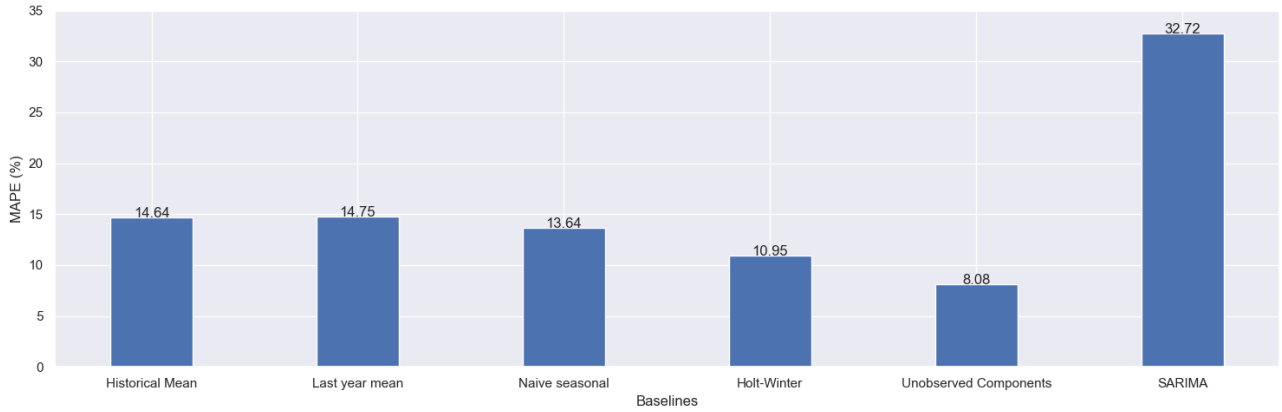

Figure 30 - MAPE of baselines and methods applied to predict the data for comparison.

The method of Unobserved Components shows a good result. Some advantages using the method during this project include the simplicity for application, low time for processing the whole method including the computational cost while running, and a very good result in a dataset with complex's seasonalities being treated in a simple way.

Words counting: 121

8 CONCLUSION

Forecasting electricity demand is a task that helps to prevent waste of energy and promotes a more smart use of electricity and the energy grid. At the same time, electricity forecast is an important output when planning projects to expansion of energy grids and prevents shortage of energy, which is a precious resource.

Although fine adjusts, and experimentation with other models are necessary to reach a reliable level in this study, it presents a starting point in a methodology that must be improved for better results.

The project explores the complexity of a time series with multiple seasonalities and sets a path that might be followed in the study of electricity forecast. Another variables must be included on the machine learning models so that they can perform better. Some studies explored on the references include information such as whether a day is a working day or not, the season of the year, external punctual events that may have influenced the data base on historic research, and also variations on temperature, and other weather variables.

The model that has performed the best in our study was the Unobserved Components Model, that gives us a fair predictions, although still in need of improvements.

Words counting: 201.

9 REFERENCES

1. Brownlee, J., 2020. Machine Learning Mastery. [Online] Available at: https://machinelearningmastery.com/decompose-time-series-data-trend-seasonality/ [Accessed 29 10 2021].

2. Cerqueira, V., 2023. Towards Data Science. [Online] Available at: https://towardsdatascience.com/time-series-for-climate-change-forecasting-energy-

demand-79f39c24c85e#:~:text=Power%20systems%20use%20forecasting%20models,is%20also%20valuable%20within%20households. [Accessed 28 10 2023].

3. Ismiguzel, I., 2020. Towards Data Science. [Online] Available at: https://towardsdatascience.com/hands-on-time-series-forecasting-with-python-d4cdcabf8aac [Accessed 07 11 2023].

4. Lazzeri, F., 2021. Data Science at Microsoft. [Online] Available at: https://medium.com/data-science-at-microsoft/automated-machine-learning-for-time-series-forecasting-657cfe7f8920 [Accessed 04 11 2023].

5. Manani, K., 2022. Toward Data Science. [Online] Available at: https://towardsdatascience.com/multi-seasonal-time-series-decomposition-using-mstl-in-python-136630e67530 [Accessed 05 11 2023].

6. Pawar, P., 2020. Towards Data Science. [Online] Available at: https://towardsdatascience.com/part-1-time-series-analysis-predicting-hourly-energy-consumption-of-san-diego-short-term-long-3a1dd1a589c9 [Accessed 27 10 2023].

7. TOTH, D. J., 2021. Analytics Vidhya. [Online] Available at: https://medium.com/analytics-vidhya/multi-seasonal-time-series-analysis-decomposition-and-forecasting-with-python-609409570007 [Accessed 05 11 2023].

10 Total number of words

4222.