Università degli Studi di Milano-Bicocca

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di Laurea Magistrale in Data Science

Automated multiscale time series forecasting comparison

Relatore: Prof. Matteo Maria Pelagatti
Co-Relatore: Prof Antonio Candelieri

Tesi di Laurea Magistrale di:
Stoffa Giacomo
Matricola 830159

Anno Accademico 2022-2023

Abstract

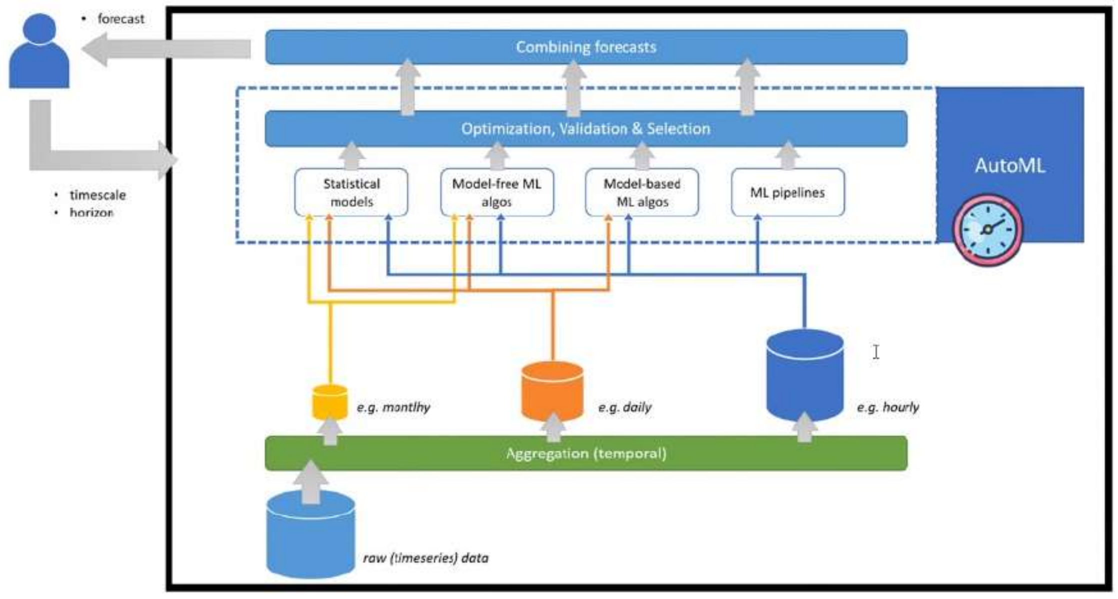
The following project was developed with the purpose of creating an automated forecasting system that allows multiscale forecasting with selectable aggregation. It is possible to aggregate a time series with different methods and automatically manage the data preprocessing phase before making the forecast,checking the parameters to optimize the transformations of the series.
The future forecast of a time series is carried out automatically using statistical tools such as ARIMA and UCM and machine learning models such as LSTM, GRU and KNN, with the possibility of calculating the forecast even at different time scales, comparing the results each other.
Predictions will be compared at different scales, analyzing their characteristics and evaluating the best forecast, with the possibility of also aggregating the predictions and comparing them with other results obtained.

Indice

1. Introduction

A multiscale time series means the possibility of considering a series from the point of view of different time frequencies, such as every ten minutes, every hour or days, months or every year when we talk about more numerous time series.
This aspect can certainly be considered for various uses, first of all making the reference dataset less numerous and, therefore, the possibility of creating and using models that require less computational time. The second consists in viewing the series from the point of view of a different perspective, considering different characteristics such as seasonality.
Forecasting is essential for supporting decisions at strategic, tactical, and operational levels. Accurate forecasts can assist companies and organizations in reducing costs, avoid risks, and exploit opportunities, thus finding application in a variety of settings.
Univariate forecasting is a challenging task that usually requires identifying patterns in time series data and selecting the most appropriate model for extrapolating them in time.
What we are trying to do in this case and which turns out to be one of the main purposes is to create an automated forecasting system, which does not require a specific study for each time series but which turns out to be able to find the best parameters for each series types and for any model, based on scale and time frequency user wants to use.
Forecasts constructed at all levels of aggregation are combined with the proposed framework to produce temporally reconciled, accurate and robust forecasts, aligned on

different planning horizons: from short-term planning to long-term strategic planning.
Temporal aggregation (TA) refers to transforming a time series from higher to lower frequencies (e.g. monthly to quarterly).
Temporal aggregation is an intuitively appealing approach that transforms a time series from higher frequencies (e.g. daily) to lower frequencies (e.g. monthly), strengthening or attenuating different characteristics of the time series. There are two different types of time aggregation: non-overlapping, where the time series is divided into lower frequency size bins; and superposition, where the values are replaced by a moving average of the length of the lower frequency.
Temporal aggregation is logically centered on the observation that the same time series (or some of their elements) may prove easier to forecast at a lower frequency than the initial level, often combined with the need for forecasts covering multiple periods (of the granularity initial).
Different characteristics of the time series can be masked or highlighted by intelligently selecting the correct level for the forecasting purpose.
Obviously it must also be considered that since it is an automated context that aims to find the best model and the best parameters automatically, it must be considered that the choice of the prediction algorithm also depends on the size of the dataset.



Initial scheme

In this case, in the initial structure of the scheme we can see how the type of computational memory influences decisions. Based on the type of aggregation you want to carry out, you need to evaluate the type of forecasting strategy to use.
Statistical models are more effective with a smaller dataset and therefore with low frequency, while machine learning models are equally effective if not more, with a larger dataset and therefore with high frequency information as in this case of hourly aggregation.
The general purpose of this project is the possibility of choosing the type of time scale with the relative type of aggregation, then going on to make a future forecast upon user request. The available scales types range from seconds to years.
ARIMA, UCM, LSTM, GRU and KNN models have been used with various techniques. As methods of comparison and accuracy it was chosen to use AIC and MAPE or MAE.

Akaike's information criterion referred to as AIC, is a method for evaluating and comparing statistical models based on the concept of entropy as a measure of information, by which it evaluates the amount of information lost when a given model is used to describe reality.
The rule is to prefer models with the lowest AIC.
The AIC is designed to find the model that explains the most variation in the data, while penalizing models that use an excessive number of parameters.
Once you've fit several regression models, you can compare the AIC value of each model. The model with the lowest AIC offers the best fit.
It provides a measure of the quality of the estimate of a statistical model by taking into account both the goodness of fit and the complexity of the model.

The MAPE is Mean Absolute Percent Error and S is a measure of relative error that uses absolute values to keep positive and negative errors from canceling each other out and uses relative errors to allow the user to compare forecast accuracy between time series models.

Formula

$$M = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

MAPE = Average of (abs ((Actual – Forecast)/(Actual)))

Meanwhile the mean absolute error (MAE) is a measure of errors between paired observations expressing the same phenomenon. MAE is calculated as the sum of absolute errors divided by the sample size:

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}.$$

Once the forecasts were made on different time scales, they were then compared to see and evaluate the actual forecast results on several different scales.
Comparing and aggregating smaller forecasts with larger time-scaled forecasts to find correlations between different forecasts.

In addition to the choice of dataset, the initial parameters that can be set are the following:

- Scale of the dataset
- type of aggregation
- number of predictions
- percentage of train and test size

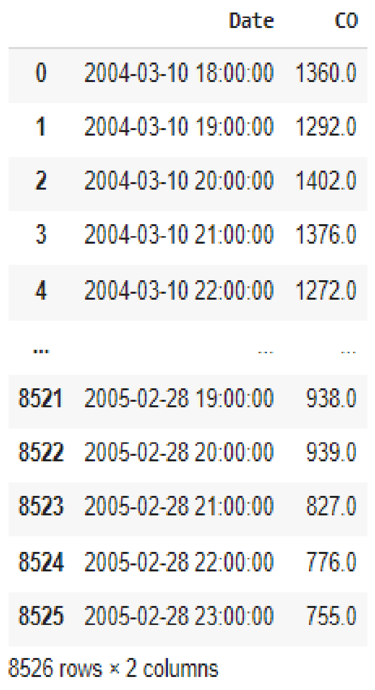
It is also possible to impose the type of transformation to be done if necessary. In cases where you already know if the series needs a certain type of transformation for stationarity and you don't want to entrust the calculation to automation, we will discuss this part in the next chapter. Obviously the dataset can only be scaled on temporal values greater than the original one.
Seconds→Minutes→Hours→Days→Weeks→Months→Years
with the following types of aggregations:

- mean
- median
- sum
- max
- min
- last

Once all the values necessary for the model to adapt to the data have been found and set, the model is created. The created models are automatically saved within the work folder, and the forecasts are saved in a specific folder as they will be used for the comparison of the forecasts at different time scales which is carried out in the next phase.

Multiscale time series

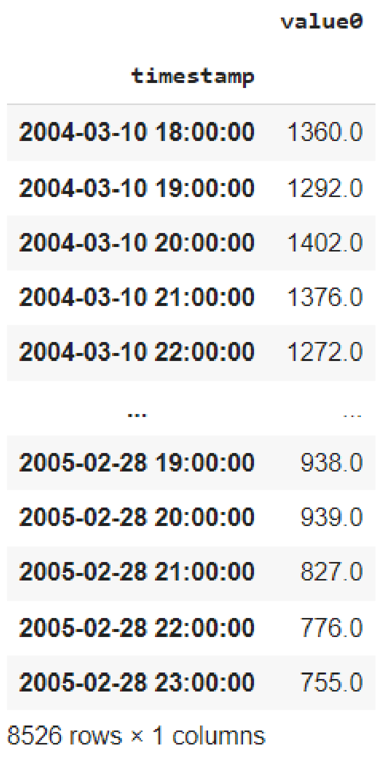
To test the algorithm different test datasets were used to improve the optimization process in order to find the best optimization parameters.
It is possible to use any type of time series file as the reading is automated by finding the column containing the date by itself and automatically creating the "value0" column containing the values of the series. Based on whichever type of file is used, the column relating to the date is automatically found and renamed "timestamp".
The date column is found and handled by changing the format to the following:
‘%Y-%m-%d %H:%M:%S'.
If there should be more other columns, many columns "value0" , "value1" , "value2" etc. will be created, in case of a multivariate development in the future.



	Date	CO
0	2004-03-10 18:00:00	1360.0
1	2004-03-10 19:00:00	1292.0
2	2004-03-10 20:00:00	1402.0
3	2004-03-10 21:00:00	1376.0
4	2004-03-10 22:00:00	1272.0
...	...	...
8521	2005-02-28 19:00:00	938.0
8522	2005-02-28 20:00:00	939.0
8523	2005-02-28 21:00:00	827.0
8524	2005-02-28 22:00:00	776.0
8525	2005-02-28 23:00:00	755.0

8526 rows × 2 columns

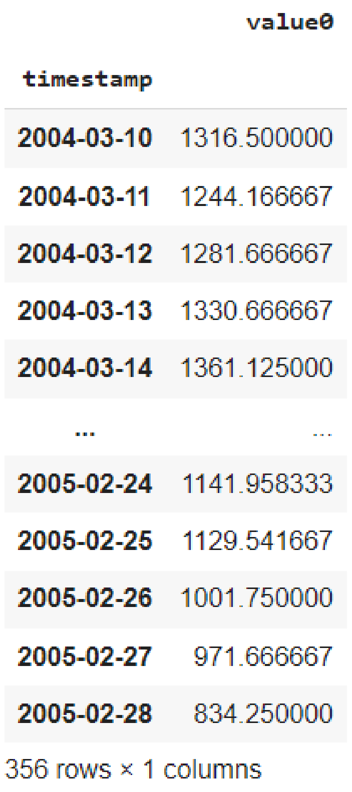
Example of imported file



	value0
timestamp	
2004-03-10 18:00:00	1360.0
2004-03-10 19:00:00	1292.0
2004-03-10 20:00:00	1402.0
2004-03-10 21:00:00	1376.0
2004-03-10 22:00:00	1272.0
...	...
2005-02-28 19:00:00	938.0
2005-02-28 20:00:00	939.0
2005-02-28 21:00:00	827.0
2005-02-28 22:00:00	776.0
2005-02-28 23:00:00	755.0

8526 rows × 1 columns

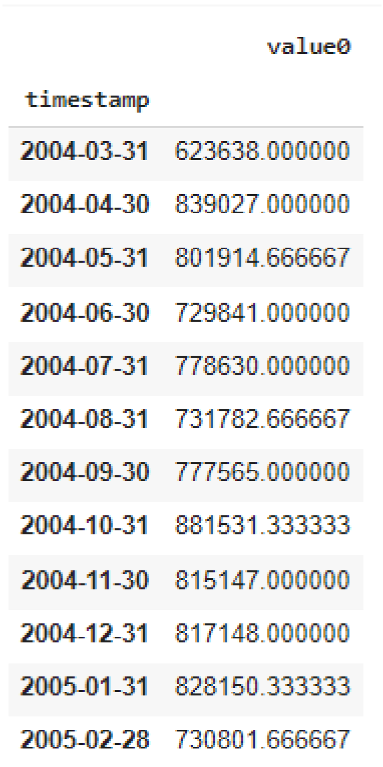
Example of preprocessed dataframe



	value0
timestamp	
2004-03-10	1316.500000
2004-03-11	1244.166667
2004-03-12	1281.666667
2004-03-13	1330.666667
2004-03-14	1361.125000
...	...
2005-02-24	1141.958333
2005-02-25	1129.541667
2005-02-26	1001.750000
2005-02-27	971.666667
2005-02-28	834.250000

356 rows × 1 columns

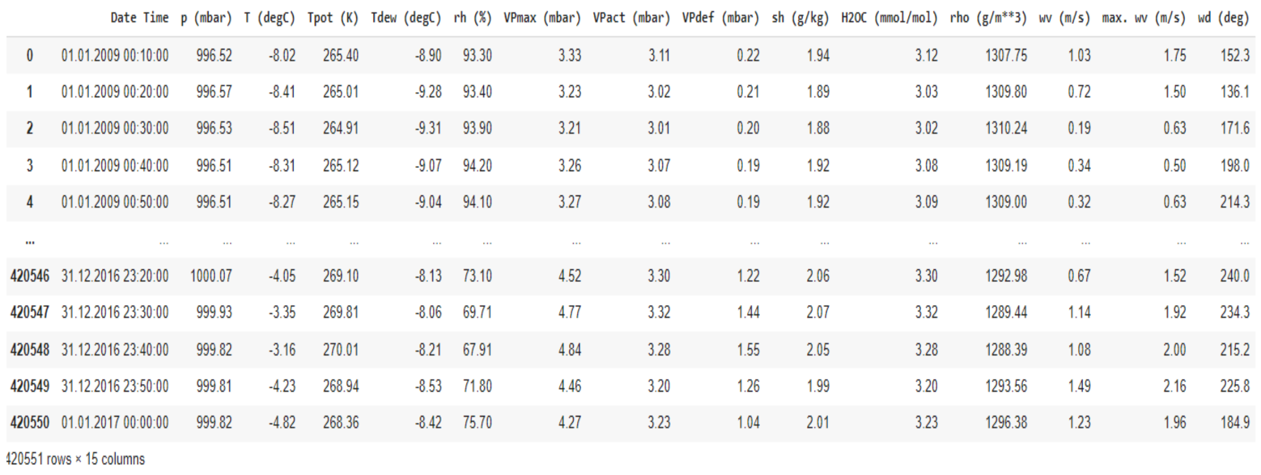
Example of dataframe scaled by days and mean from hours



	value0
timestamp	
2004-03-31	623638.000000
2004-04-30	839027.000000
2004-05-31	801914.666667
2004-06-30	729841.000000
2004-07-31	778630.000000
2004-08-31	731782.666667
2004-09-30	777565.000000
2004-10-31	881531.333333
2004-11-30	815147.000000
2004-12-31	817148.000000
2005-01-31	828150.333333
2005-02-28	730801.666667

Example of dataframe scaled by months and sum

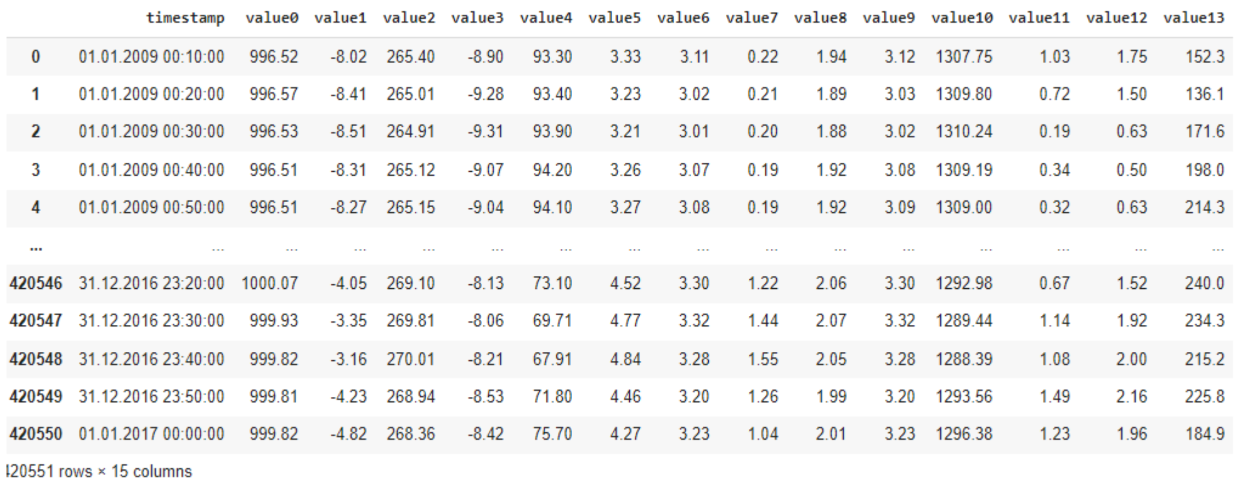
So we can see that it is possible to use data aggregation types such as mean and median for standard values for example electricity or consumption values, while types of aggregation such as max min or last associated with price values like market values relating to the price of a certain thing or service. While it sums for time series that have cumulative values that must be considered dependent on each other over time.



	Date Time	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)	max. wv (m/s)	wd (deg)
0	01.01.2009 00:10:00	996.52	-8.02	265.40	-8.90	93.30	3.33	3.11	0.22	1.94	3.12	1307.75	1.03	1.75	152.3
1	01.01.2009 00:20:00	996.57	-8.41	265.01	-9.28	93.40	3.23	3.02	0.21	1.89	3.03	1309.80	0.72	1.50	136.1
2	01.01.2009 00:30:00	996.53	-8.51	264.91	-9.31	93.90	3.21	3.01	0.20	1.88	3.02	1310.24	0.19	0.63	171.6
3	01.01.2009 00:40:00	996.51	-8.31	265.12	-9.07	94.20	3.26	3.07	0.19	1.92	3.08	1309.19	0.34	0.50	198.0
4	01.01.2009 00:50:00	996.51	-8.27	265.15	-9.04	94.10	3.27	3.08	0.19	1.92	3.09	1309.00	0.32	0.63	214.3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
420546	31.12.2016 23:20:00	1000.07	-4.05	269.10	-8.13	73.10	4.52	3.30	1.22	2.06	3.30	1292.98	0.67	1.52	240.0
420547	31.12.2016 23:30:00	999.93	-3.35	269.81	-8.06	69.71	4.77	3.32	1.44	2.07	3.32	1289.44	1.14	1.92	234.3
420548	31.12.2016 23:40:00	999.82	-3.16	270.01	-8.21	67.91	4.84	3.28	1.55	2.05	3.28	1288.39	1.08	2.00	215.2
420549	31.12.2016 23:50:00	999.81	-4.23	268.94	-8.53	71.80	4.46	3.20	1.26	1.99	3.20	1293.56	1.49	2.16	225.8
420550	01.01.2017 00:00:00	999.82	-4.82	268.36	-8.42	75.70	4.27	3.23	1.04	2.01	3.23	1296.38	1.23	1.96	184.9

420551 rows × 15 columns

Example of imported file



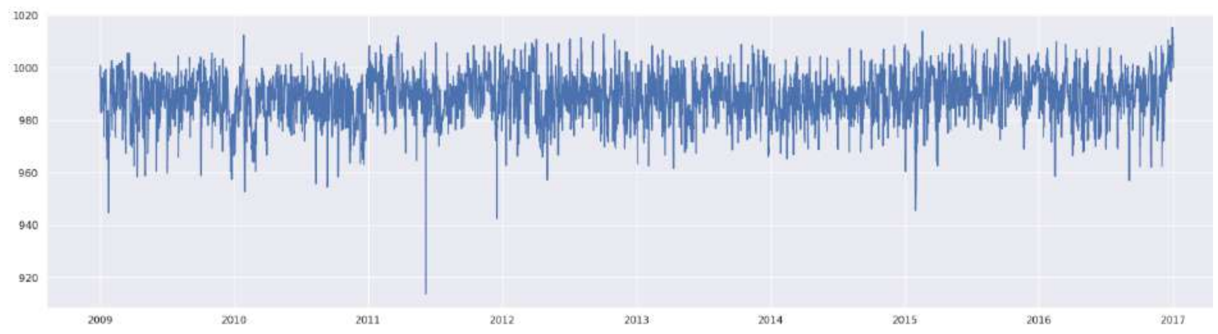
	timestamp	value0	value1	value2	value3	value4	value5	value6	value7	value8	value9	value10	value11	value12	value13
0	01.01.2009 00:10:00	996.52	-8.02	265.40	-8.90	93.30	3.33	3.11	0.22	1.94	3.12	1307.75	1.03	1.75	152.3
1	01.01.2009 00:20:00	996.57	-8.41	265.01	-9.28	93.40	3.23	3.02	0.21	1.89	3.03	1309.80	0.72	1.50	136.1
2	01.01.2009 00:30:00	996.53	-8.51	264.91	-9.31	93.90	3.21	3.01	0.20	1.88	3.02	1310.24	0.19	0.63	171.6
3	01.01.2009 00:40:00	996.51	-8.31	265.12	-9.07	94.20	3.26	3.07	0.19	1.92	3.08	1309.19	0.34	0.50	198.0
4	01.01.2009 00:50:00	996.51	-8.27	265.15	-9.04	94.10	3.27	3.08	0.19	1.92	3.09	1309.00	0.32	0.63	214.3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
420546	31.12.2016 23:20:00	1000.07	-4.05	269.10	-8.13	73.10	4.52	3.30	1.22	2.06	3.30	1292.98	0.67	1.52	240.0
420547	31.12.2016 23:30:00	999.93	-3.35	269.81	-8.06	69.71	4.77	3.32	1.44	2.07	3.32	1289.44	1.14	1.92	234.3
420548	31.12.2016 23:40:00	999.82	-3.16	270.01	-8.21	67.91	4.84	3.28	1.55	2.05	3.28	1288.39	1.08	2.00	215.2
420549	31.12.2016 23:50:00	999.81	-4.23	268.94	-8.53	71.80	4.46	3.20	1.26	1.99	3.20	1293.56	1.49	2.16	225.8
420550	01.01.2017 00:00:00	999.82	-4.82	268.36	-8.42	75.70	4.27	3.23	1.04	2.01	3.23	1296.38	1.23	1.96	184.9

420551 rows × 15 columns

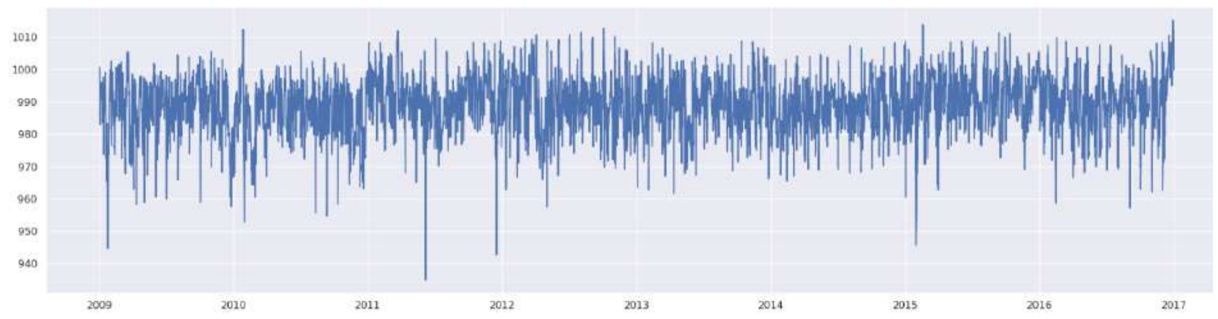
Example of first-step preprocessed dataframe

This is an example of how to import a multivariate and not a univariate series in which, in addition to creating the column relating to the timestamp, the columns are automatically renamed in reference to how many values in the series need to be considered in the case of a multivariate implementation.

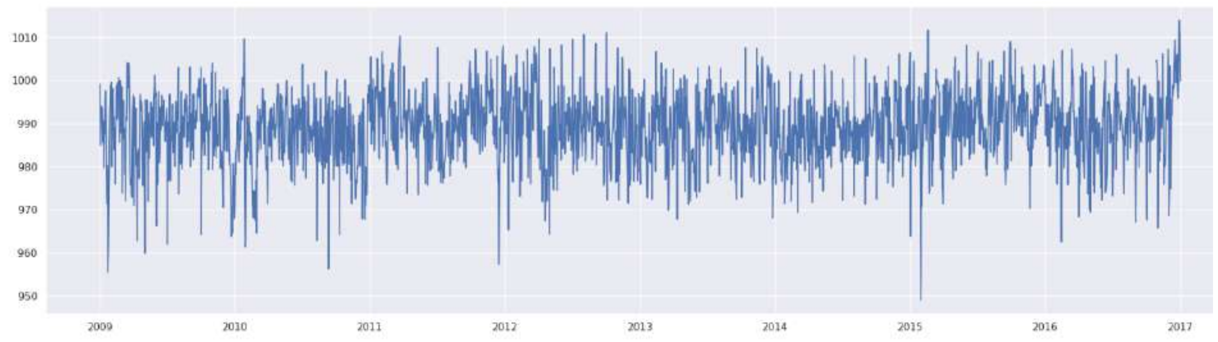
Now we can see the graphic trend of a time series to which data aggregations from "10min" to "years" have been applied.



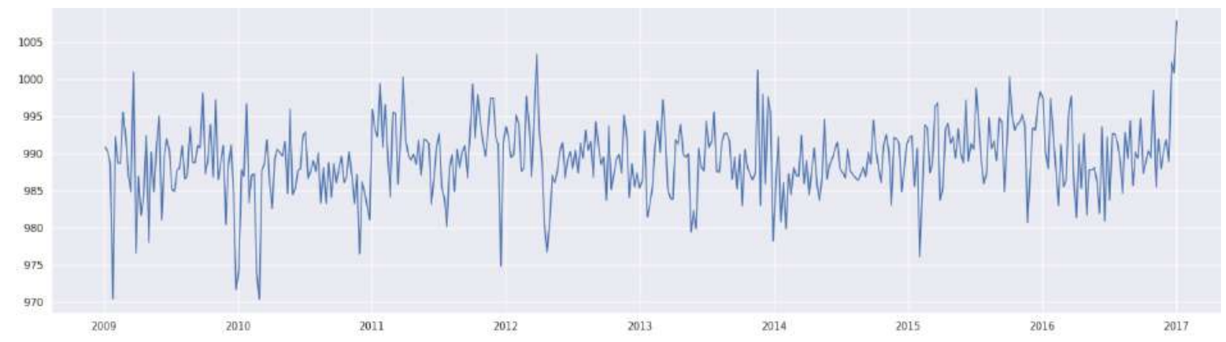
10 minutes timeseries

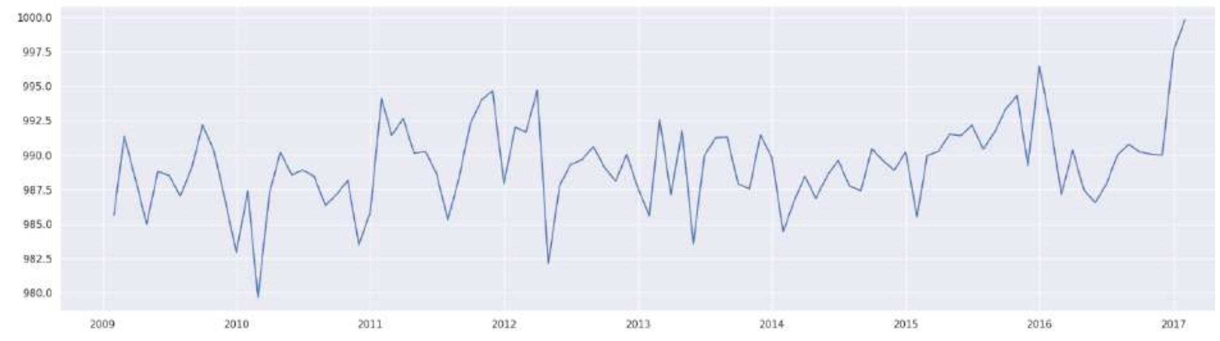


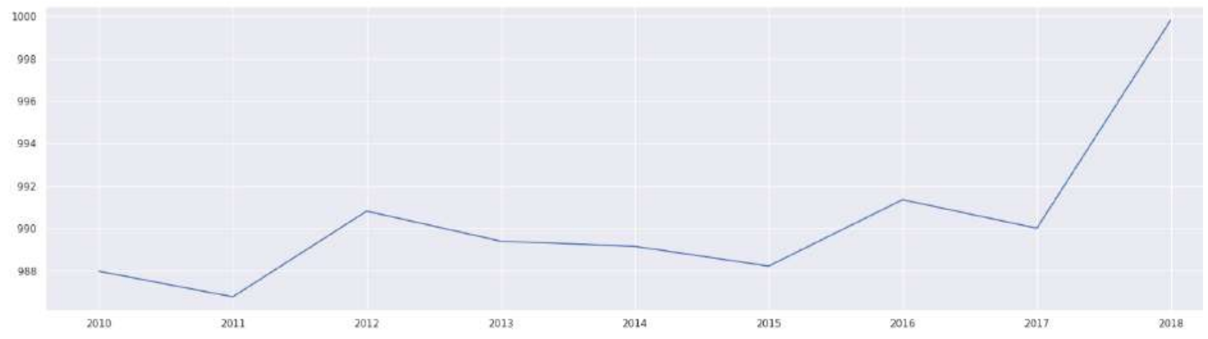
hours aggregation with mean



days aggregation with mean

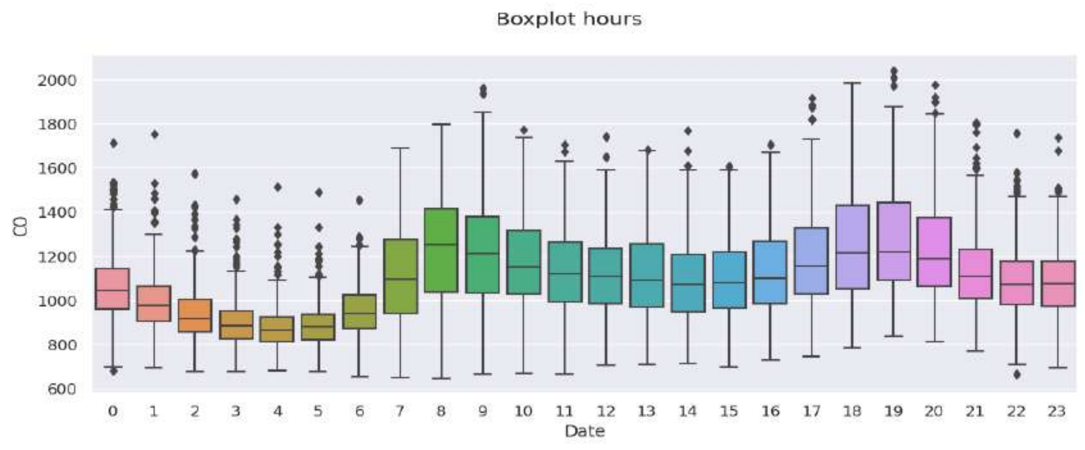

weeks aggregation with mean


months aggregation with mean

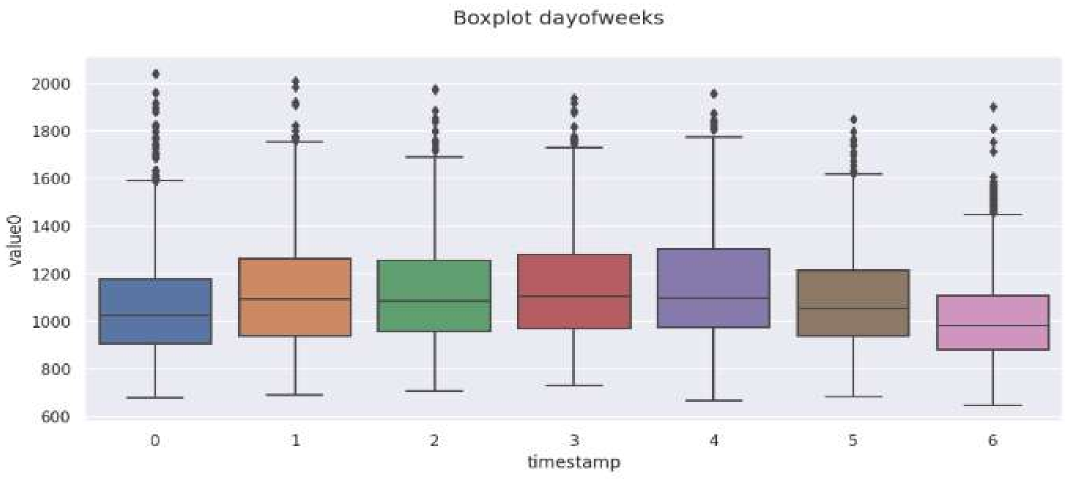

years aggregation with mean

Exploration and preprocessing

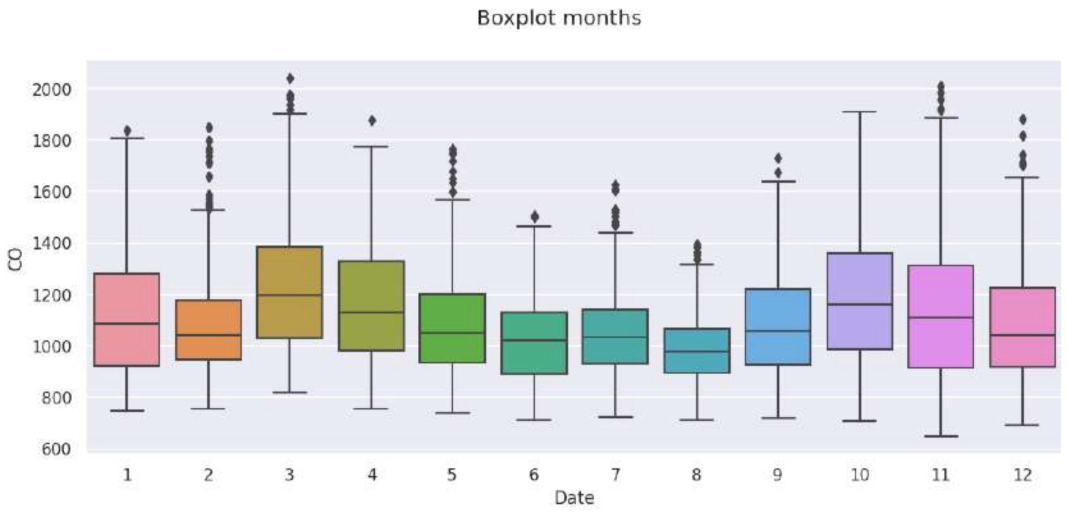
Once understood how the initial series was structured as a dataframe, it was necessary to understand how to be able to automatically manage some initial preprocessing phases.
Regardless of the type of aggregation and scale used, boxplots were used for the initial phase which automatically understood if there were clearly any months, days of the week or hours in which the values were significantly altered.
Often this procedure highlighted the fact that during the weekend on different time series that have been used for testing the values were significantly different, either positively or negatively.



Example Boxplot hours



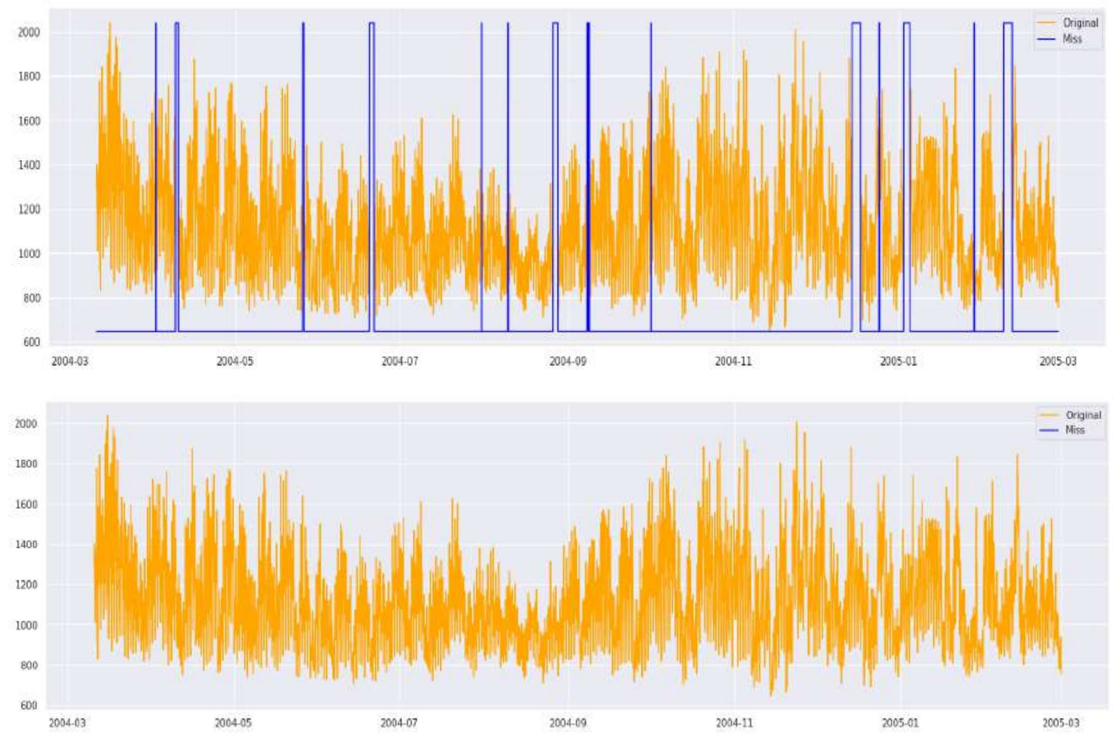
Example Boxplot dayofweeks



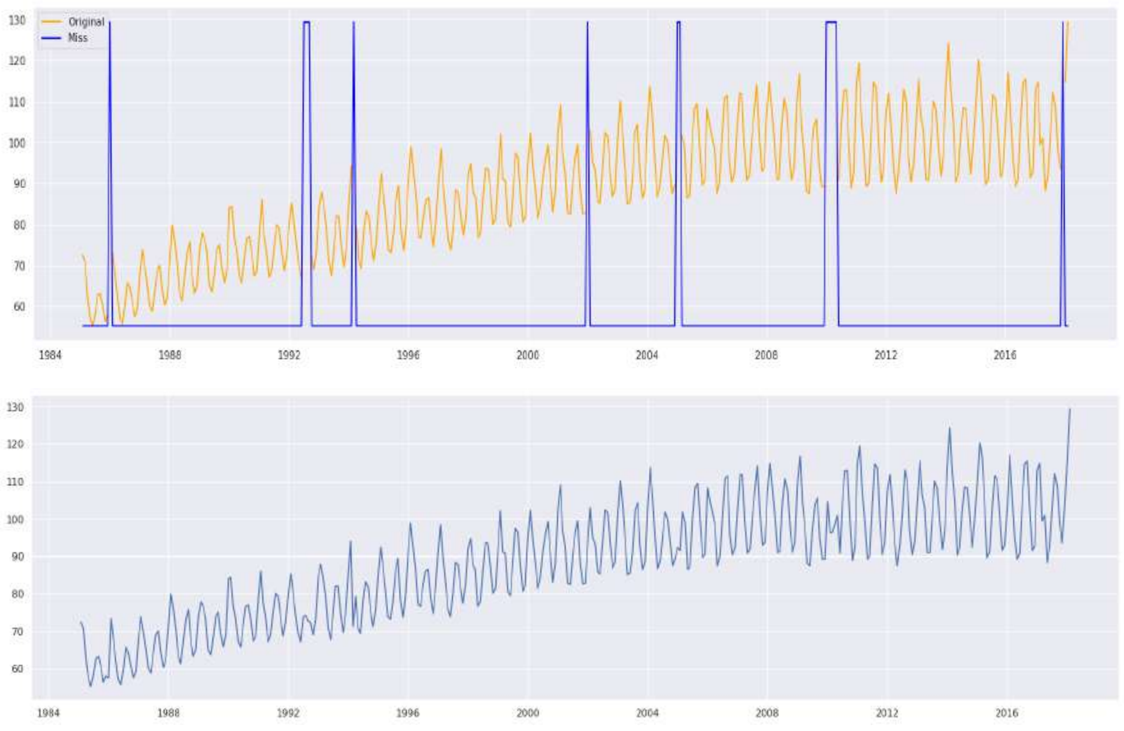
Example Boxplot months

A threshold is assigned to this process which allows to establish whether some months, days of the week or hours are to be considered as outliers in the next phase of when the forecasting models will be created.

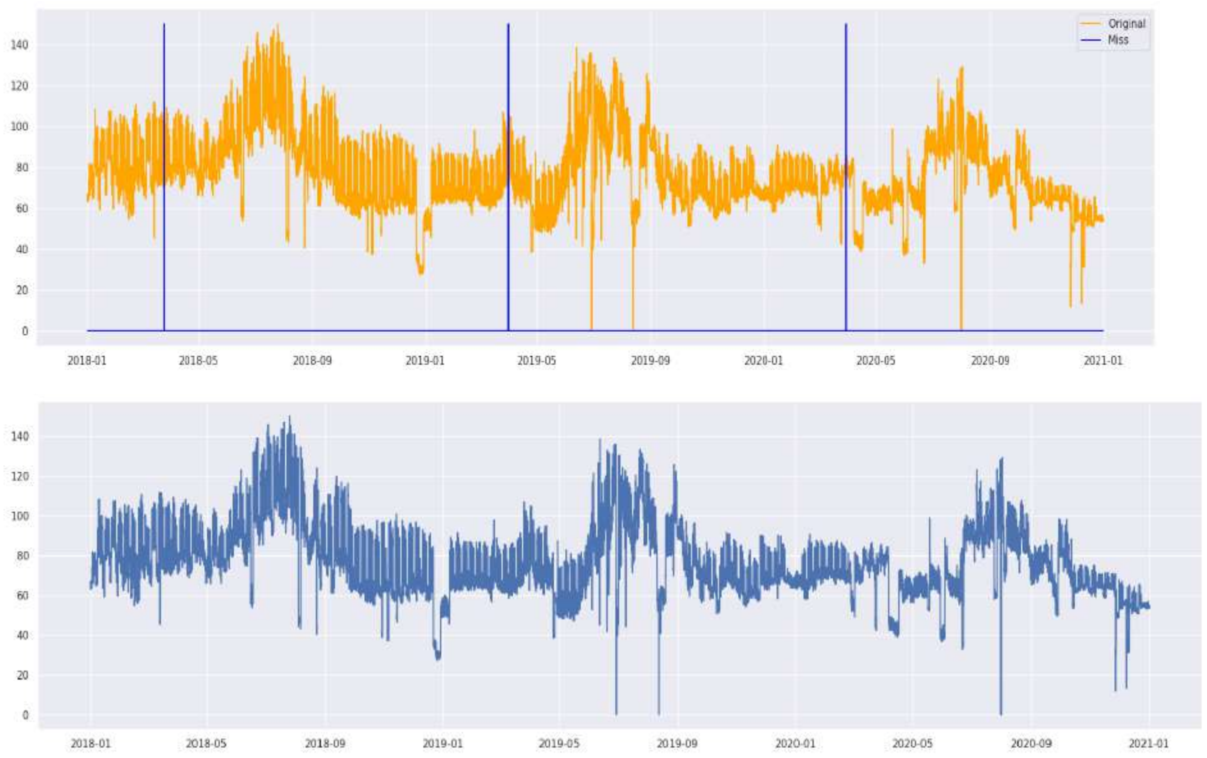
The initial phase of managing a time series must be optimized for whatever type of series is present, even with missing values. consequently a solution had to be found to replace the missing values automatically.



Example of interpolated time serie



Example of interpolated time serie



Example of interpolated time serie

That it was decided to interpolate by making an average between the 3 previous values with the same characteristics such as for example at the same time and same day of the week.

2. Stationarity in mean and variance

The stationarity of a time series is a fundamental concept in time series data analysis. It indicates that the statistical properties of the series, such as the mean and variance, remain constant over time. In other words, the data does not exhibit evident trends or seasonal cycles that significantly influence its characteristics over time.

A stationary time series is one whose properties do not depend on the time at which the series is observed. Thus, time series with trends, or with seasonality, are not stationary, the trend and seasonality will affect the value of the time series at different times. It is generally preferable to work with a stationary time series for forecasting, for data to be stationary, the statistical properties of a system do not change over time.

In a stationary series, the observed values are all independent of each other and the statistical properties of the series do not change with time. This means that past values can be used to predict the future, as the relationships between the various observations are constant and persistent over time.

In a non-stationary series, the observed values are affected by random trends and fluctuations and are not easily predictable. This makes it difficult to obtain accurate and reliable forecasts based on past values.

In summary, stationarity is crucial to ensure that analyses of time series data are accurate, reliable, and interpretable, thereby enabling a better understanding of the data and more precise forecasting of future developments.

To try to automatically understand what type of transformation to use, if logarithmic transformation or seasonal difference, it was decided to use several tests in a combined way, finding a solution based on how many tests were passed correctly.

The Augmented Dickey-Fuller (ADF) and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) statistical tests are essential tools in time series analysis to determine whether a time series is stationary or not.
The **Augmented Dickey-Fuller test** (ADF) is a statistical test used to determine whether or not a time series is stationary. It is particularly useful for identifying the presence of unit roots, which indicate the presence of non-stationary trends in the data series.
This is useful for detecting stationarity in the mean.

```
#ADFULLER
#HO serie non stazionaria
#H1: serie  stazionaria (debole) varianza costante nel tempo,
valori osservati sono tutti indipendenti tra loro e che le
proprietà statistiche della serie non cambiano nel tempo.
#se p<0.05 rifiuto ipotesi nulla--> valuto H1
```

The null hypothesis H0 of the Augmented Dickey-Fuller is that there is a unit root, with the alternative that there is no unit root. If the p value is above a critical size, then we cannot reject that there is a unit root.
The p-values are obtained through regression surface approximation from MacKinnon 1994.
If the p-value is close to significant, then the critical values should be used to judge whether to reject the null.
Example:

ADF: -1.2401890027744202 P-value: 0.19746110365456304

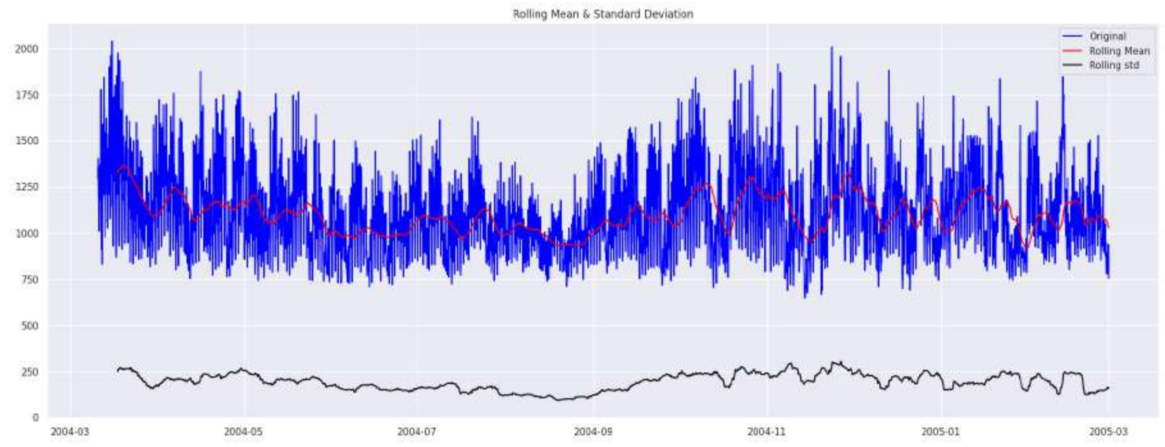
```
Num of lags: 37 Num of observations used for ADF regression and critical values
calculation: 8488 Critical values:      1% : -2.5660034575057895
        5% : -1.9410316913299555    10% : -1.6167887463941413
dati non stazionari
```

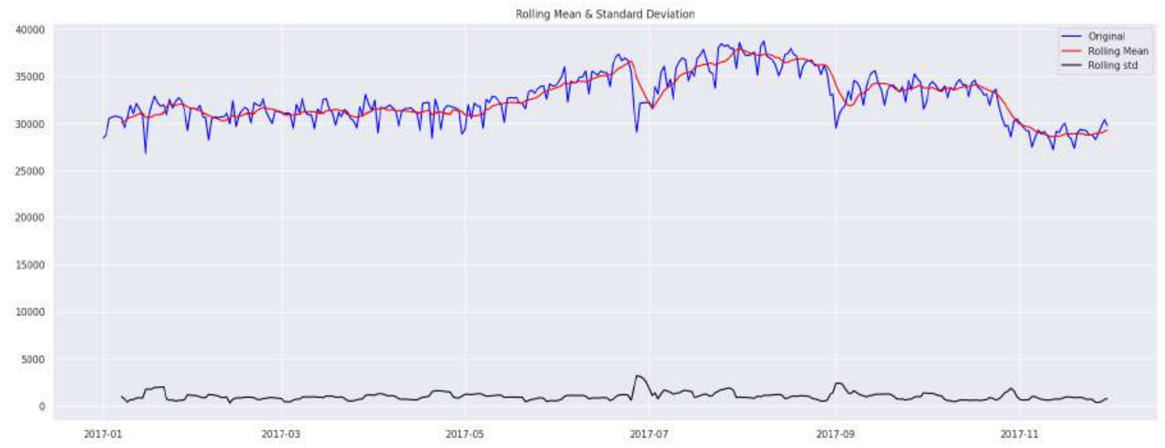
The **Kwiatkowski-Phillips-Schmidt-Shin** (KPSS) test was then performed whose null hypothesis is that a time series is stationary.
If the null hypothesis is rejected, it can be concluded that the time series is non-stationary and has a systematic trend or drift.
The KPSS test is based on the idea that a stationary series should have constant variance over time, while a non-stationary series may show trends or patterns that lead to increasing or decreasing variance over time.

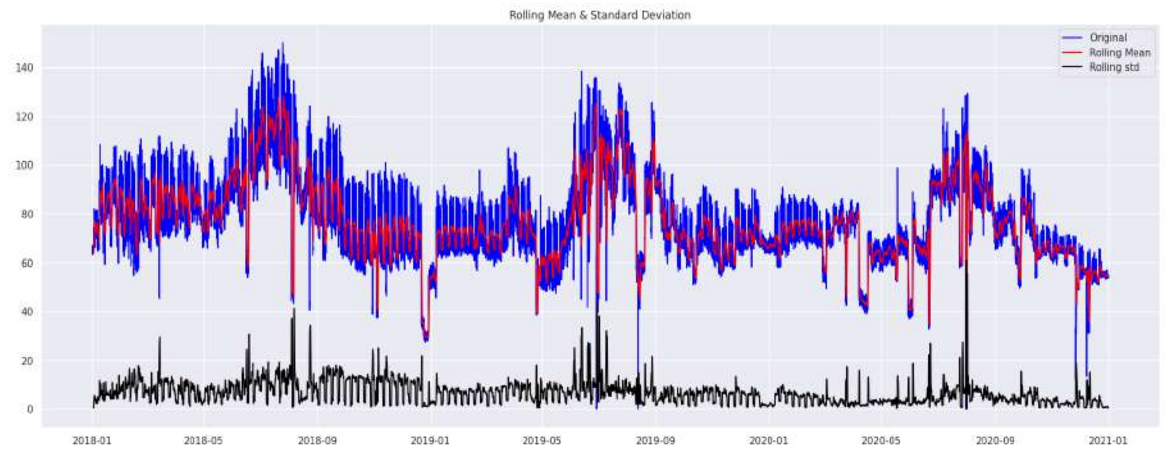
Example:
```
KPSS Statistic: 0.610241
p-value: 0.010000
prove contro H0, dati non stazionari
```

A key difference from ADF test is the null hypothesis of the KPSS test is that the series is stationary.
So practically, the interpretation of p-value is just the opposite to each other.
That is, if p-value is < significant level (say 0.05), then the series is non-stationary. Whereas in ADF test, it would mean the tested series is stationary.
Finally, the number of lags reported is the number of lags of the series that was actually used by the model equation of the kpss test. By default, the statsmodels kpss() uses the 'legacy' method. In legacy method, int(12 * (n / 100)**(1 / 4)) number of lags is included, where n is the length of the series.

In summary, while the ADF test focuses on mean stationarity, the KPSS test focuses on variance stationarity. Both are valuable tools in time series analysis for understanding how variables behave over time.



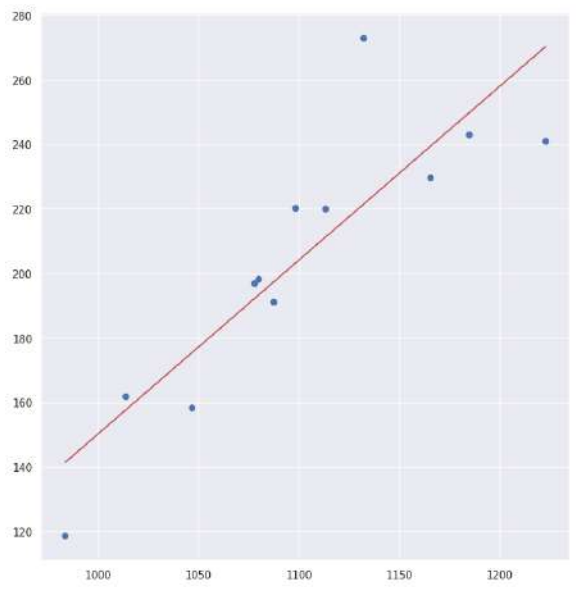


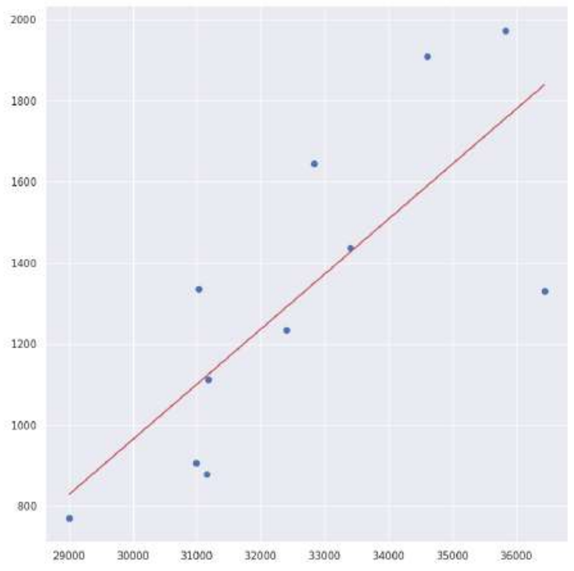


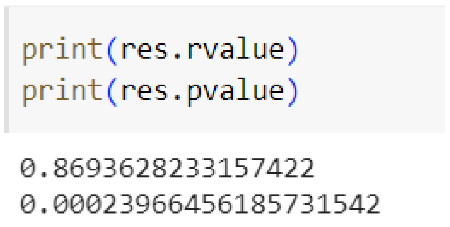
Examples of Rolling mean and standard deviation with different datasets

Even these graphs as an example, using one of the test dataframes, can be seen how the trend of the moving average and the standard deviation are too variable and not steady,

we can conclude that it is a non-stationary historical series that needs some operations before proceeding with the development of forecasting algorithms.
The following graph also allows you to analyze the linear correlation between the mean and the standard deviation.

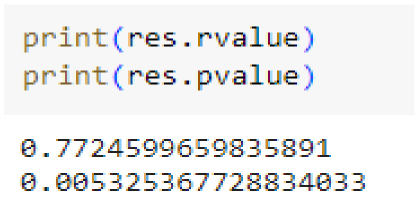






```
print(res.rvalue)
print(res.pvalue)
```

```
0.8693628233157422
0.00023966456185731542
```



```
print(res.rvalue)
print(res.pvalue)
```

```
0.7724599659835891
0.005325367728834033
```

examples of a significant positive correlation between means and standard deviation

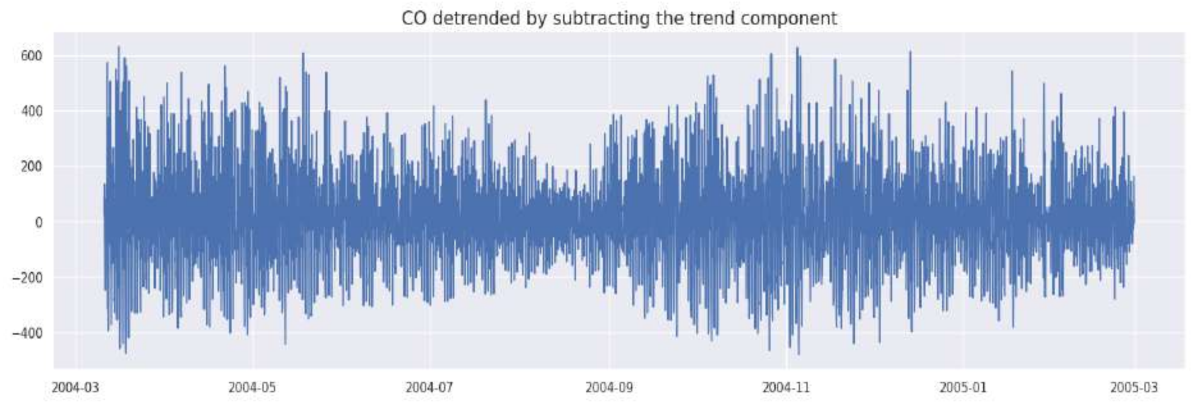
Time series data typically consist of several components, each contributing to the overall pattern and variability of the series. These components are:
The **trend** component that represents the long-term movement or direction in the data. It captures the underlying growth or decline over time and can be ascending, descending, or stable.
The **seasonality** refers to the repeating patterns or fluctuations that occur at regular intervals, often corresponding to seasons, months, days of the week, or other time periods.
Unlike seasonality, **cyclic patterns** are not fixed to specific calendar intervals. They represent periodic fluctuations that are not as precisely timed as seasonality but still occur over extended periods.
Irregularity or Noise: This component represents random variations or noise in the data that cannot be attributed to the trend, seasonality, or cyclic patterns. It often results from unpredictable events or measurement errors.

Understanding and decomposing these components is crucial in time series analysis, as it allows us to model and make forecasts based on the underlying patterns within the data. Various statistical techniques and models are implemented to extract and analyze these components effectively.

Trend component

The simplest way to detrend a time series is by subtracting the mean value of the data. This is called a constant model, and it assumes that the trend of the time series is a straight horizontal line.
As a first test I tried to eliminate the trend component using the available function of the statmodels library.
For convenience, for all these tests i decided to use the same dataset.
It's rare to find a time series where the trend is a simple horizontal line through time, but it's very common to find a linear trend.

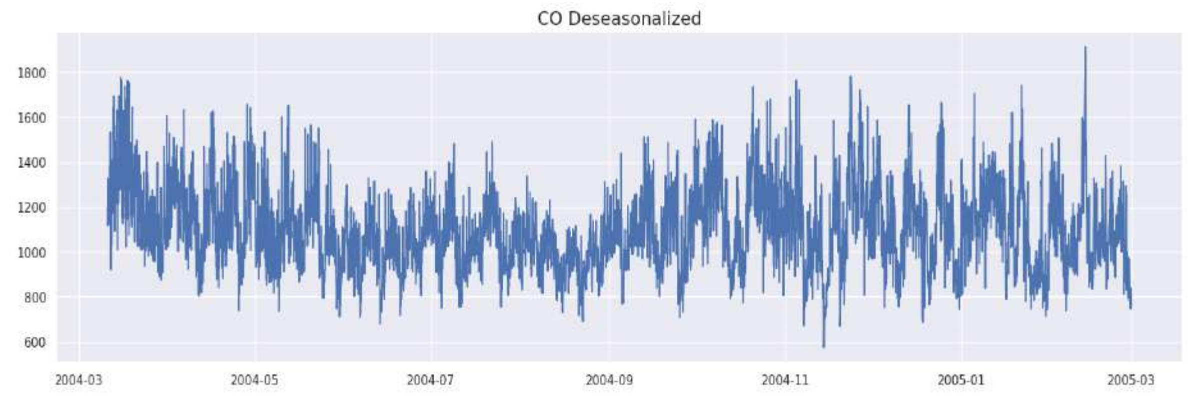
Value = Base Level + Trend + Seasonality + Error



Example of detrended time serie

Seasonal component

Seasonality in time series refers to a pattern that repeats itself at regular intervals over time.
This pattern could be daily, weekly, monthly, or even yearly, and it's often caused by factors such as weather patterns, holidays, and other recurring events.
In general, an additive model is more appropriate when the seasonal pattern in the data is consistent over time, regardless of whether the overall trend of the data is increasing or decreasing. On the other hand, a multiplicative model is more appropriate when the seasonal fluctuations are proportional to the trend of the time series.
Deseasonalizing a time series amounts to estimating the S contribution and removing it by dividing scale by S.



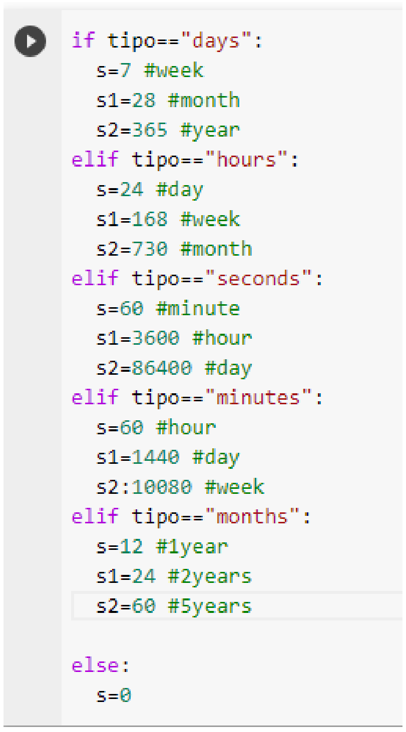
example of deseasonalized time serie

Seasonal difference

Seasonal differencing involves subtracting the value of a time series from the value at the same time in the previous season.
This can be useful in cases where the time series exhibits a seasonal pattern.
For example, if we have a time series of monthly sales data, we would subtract the value of the same month in the previous year from the current month.

If we have a time series of daily sales data, we could subtract the value of the same day in the previous week from the current day.

Considering that it is always an automated forecasting algorithm that aims to obtain a forecast independently and without knowing the timestamp of the dataframe, it was decided to automatically define 3 seasonality values s, s1 and s2 based on the type of time scale frequency of the dataframe , as to use the following values for subsequent operations automatically.



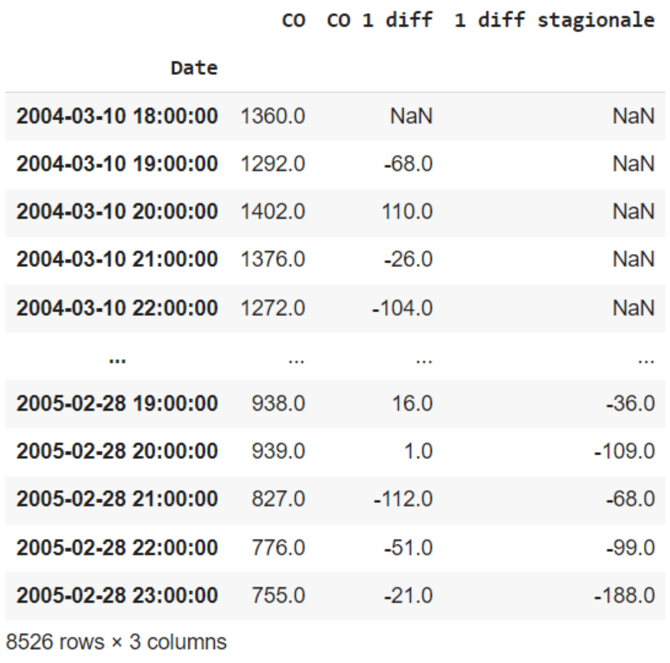
```
if tipo=="days":
  s=7 #week
  s1=28 #month
  s2=365 #year
elif tipo=="hours":
  s=24 #day
  s1=168 #week
  s2=730 #month
elif tipo=="seconds":
  s=60 #minute
  s1=3600 #hour
  s2=86400 #day
elif tipo=="minutes":
  s=60 #hour
  s1=1440 #day
  s2:10080 #week
elif tipo=="months":
  s=12 #1year
  s1=24 #2years
  s2=60 #5years

else:
  s=0
```

standard seasonal values

	minute	**hour**	**day**	**week**	**month**	**1year**	**2years**	**5years**
seconds	s=60	s1= 3600	s2= 86400					
minutes		s=60	s1= 1440	s2= 10080				
hours			s=24	s1=168	s2=730			
days				s=7	s1=28	s2=365		
months						s=12	s1=24	s2=60

Standard seasonal values

Then were carried out try going to subtract 1 difference and a seasonal difference.

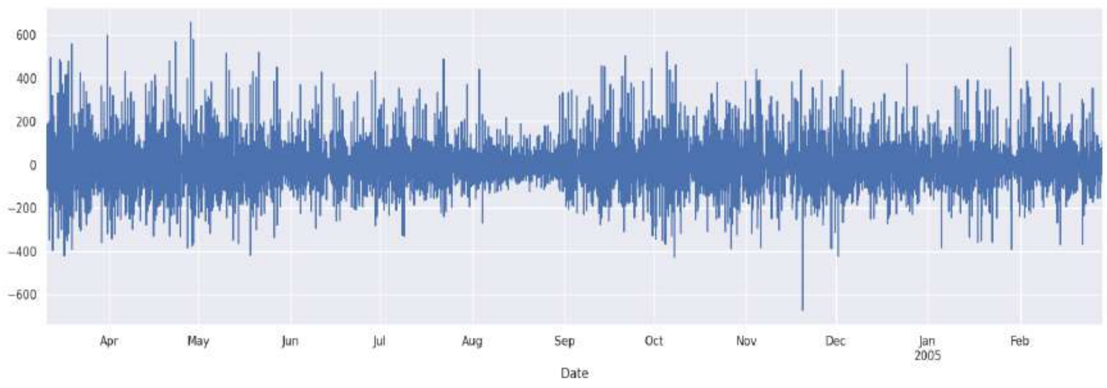


Date	CO	CO 1 diff	1 diff stagionale
2004-03-10 18:00:00	1360.0	NaN	NaN
2004-03-10 19:00:00	1292.0	-68.0	NaN
2004-03-10 20:00:00	1402.0	110.0	NaN
2004-03-10 21:00:00	1376.0	-26.0	NaN
2004-03-10 22:00:00	1272.0	-104.0	NaN
...	...	...	...
2005-02-28 19:00:00	938.0	16.0	-36.0
2005-02-28 20:00:00	939.0	1.0	-109.0
2005-02-28 21:00:00	827.0	-112.0	-68.0
2005-02-28 22:00:00	776.0	-51.0	-99.0
2005-02-28 23:00:00	755.0	-21.0	-188.0

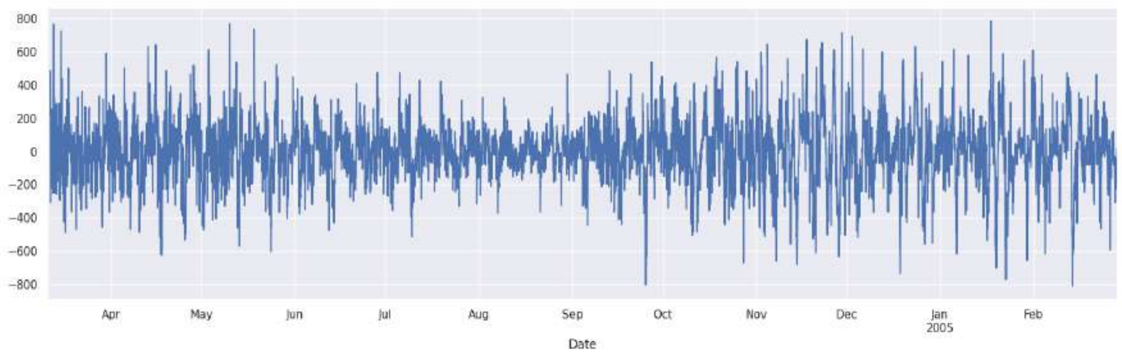
8526 rows × 3 columns

example of differentiated series

In this case, two differentiations were made from an initial historical series, one of one element and the other of seasonality s.
Thus obtaining the stationarity of the two new series, checking through the previously used tests of adfuller and kpss.



Example time serie 1 diff

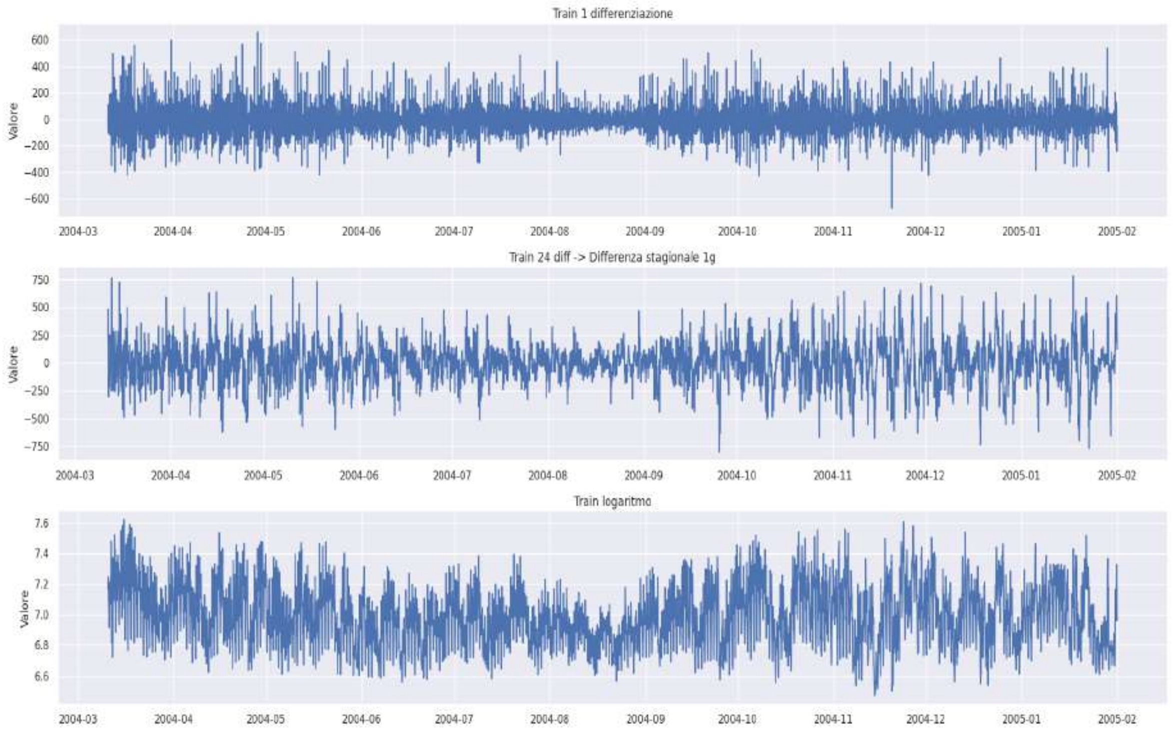


Example time serie seasonal diff

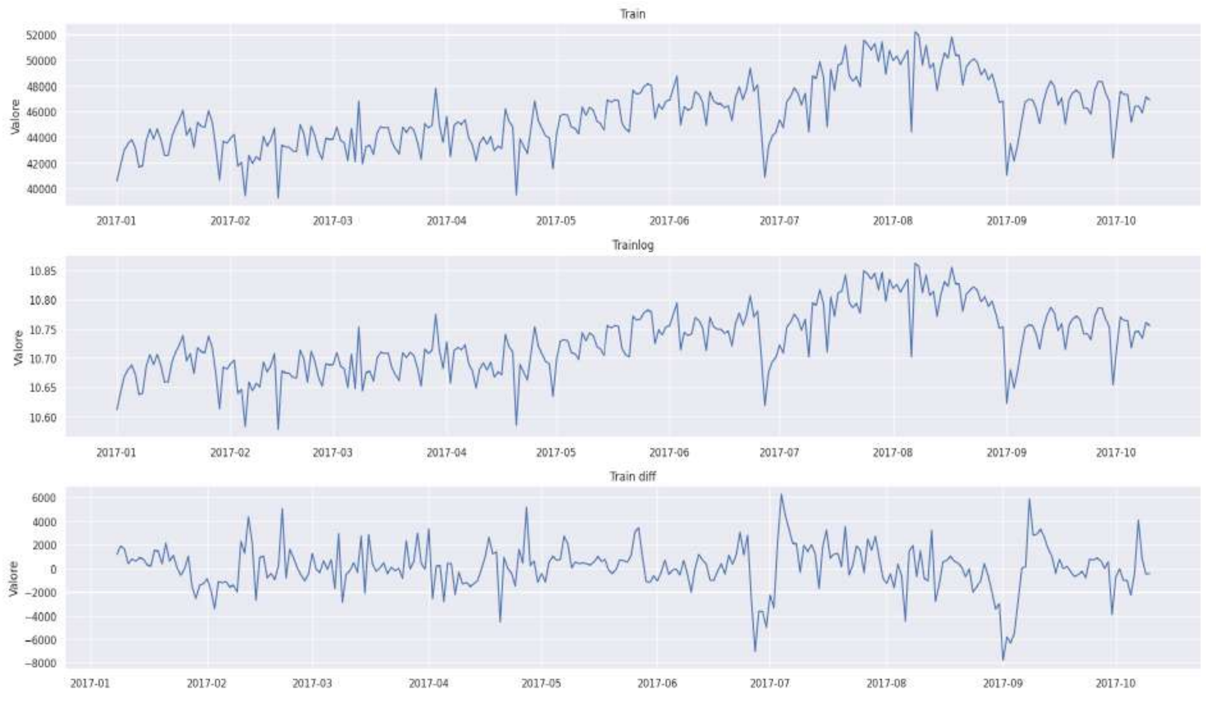
Logarithmic transformation

As a last transformation before proceeding with the tests it was decided to implement the logarithm of the series.
The logarithmic transformation of a time series is a valuable technique in time series analysis because it helps make the series more stationary. This is crucial for several reasons: firstly, stationary series exhibit constant statistical behaviours over time, making it easier to apply statistical models. Stability in means and variances is essential for the accuracy of forecasts.

Furthermore, the logarithmic transformation reduces the percentage variation in the data, making fluctuations more consistent over time. This can help uncover trends or underlying patterns that might be hidden within the larger fluctuations of the original data.
The logarithmic transformation is particularly useful when data shows exponential growth or rapid increases over time, as it can help linearize the series, making it easier to use linear model-based forecasting methods.



example of comparison between different transformation

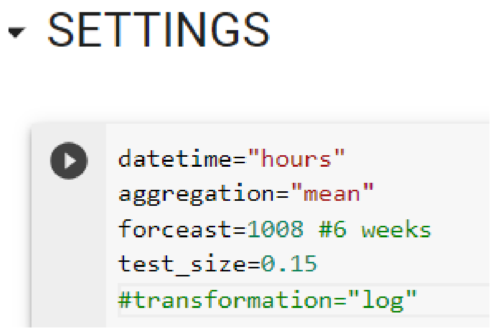


example of comparison between different transformation

Once the types of transformations to be used had been decided, all the tests we mentioned before were carried out in order to understand which was the best transformation to make the series stationary.

Train and Validation test

Among the values to be set manually in the initial phase, in addition to the following, there is also the dimension of separation between train and validation set, which is divided by 85% and 15% based on the dimensions of the dataframe.
If the length of the dataframe is less than a year and the time scale should be larger than the daily one, it is chosen to use the 90% 10% division to use more data for training the model.

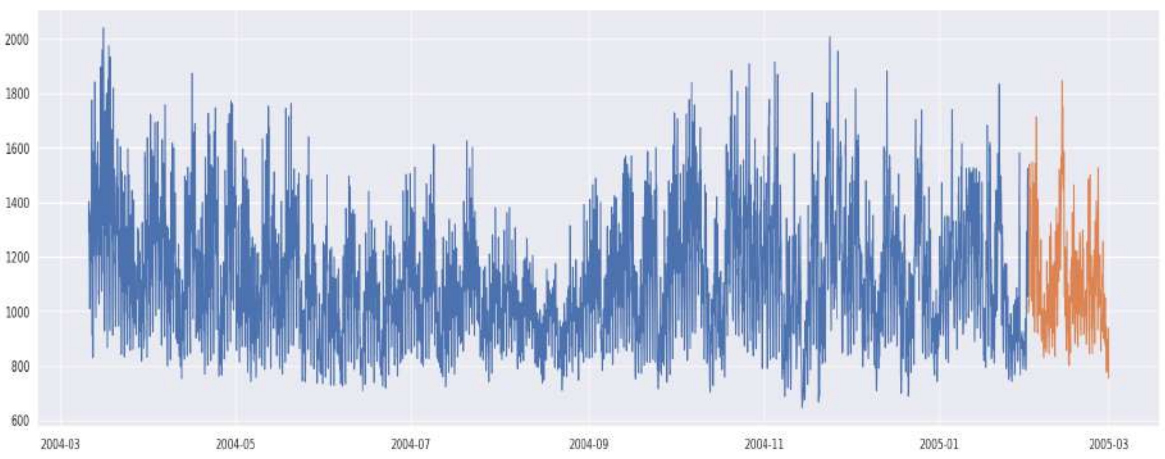


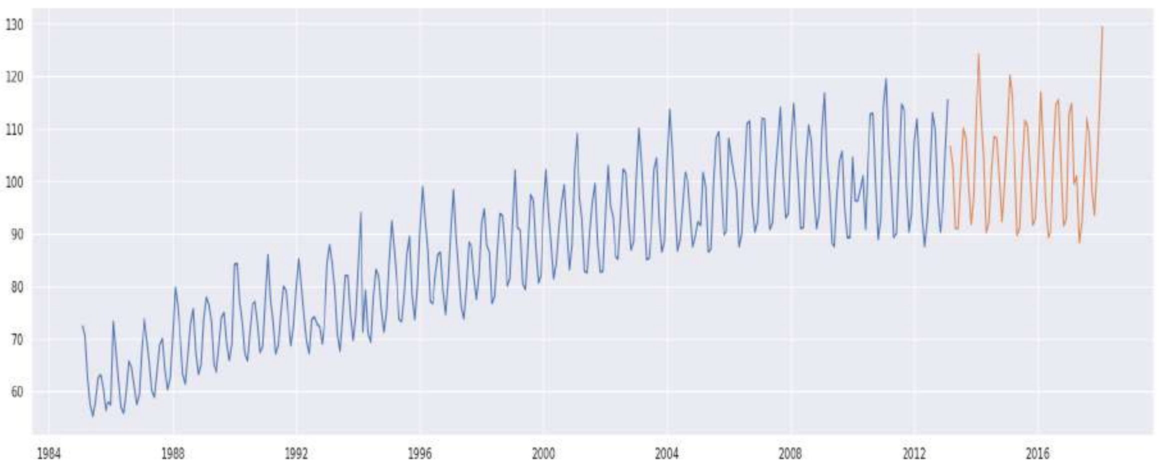
▾ SETTINGS

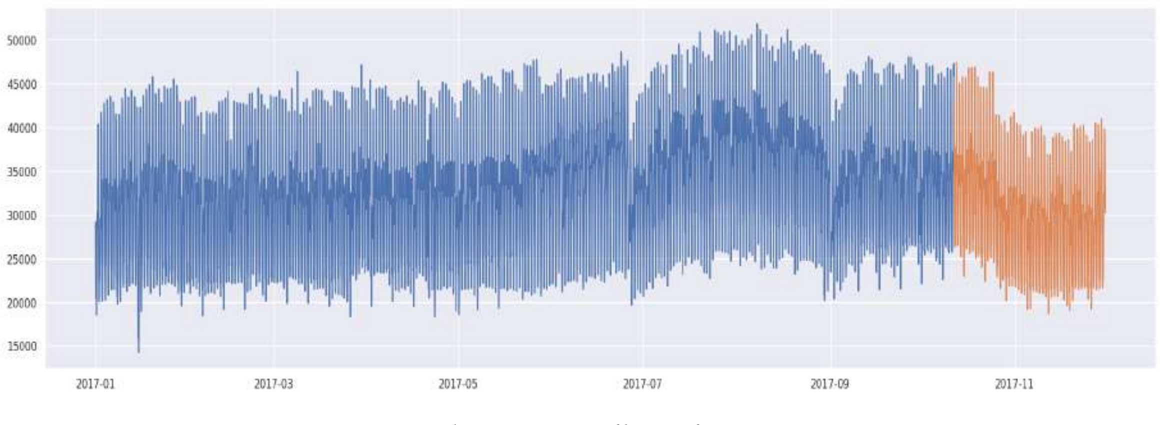
```
datetime="hours"
aggregation="mean"
forceast=1008 #6 weeks
test_size=0.15
#transformation="log"
```

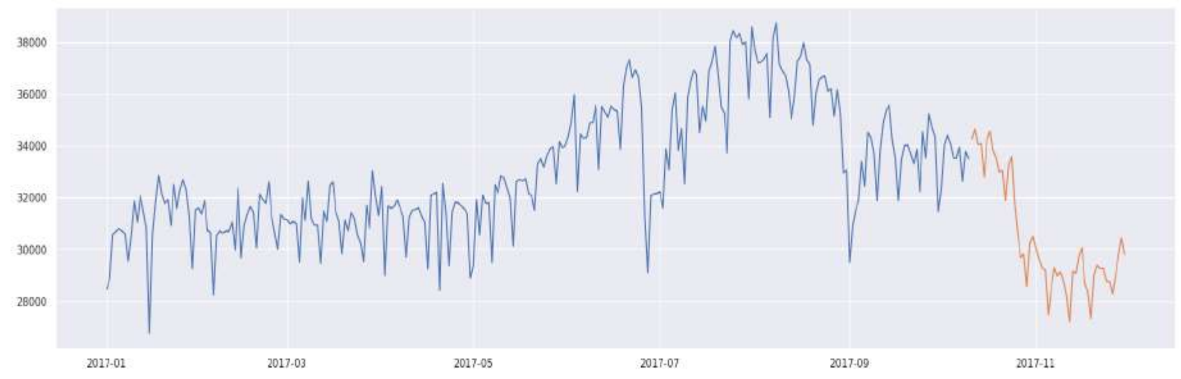
initial features to set

The train and test division must be done before applying the transformation as it must be done only on the train set with which the model will be trained ,not on the entire dataset.
Some examples with dataframes used during test phase:

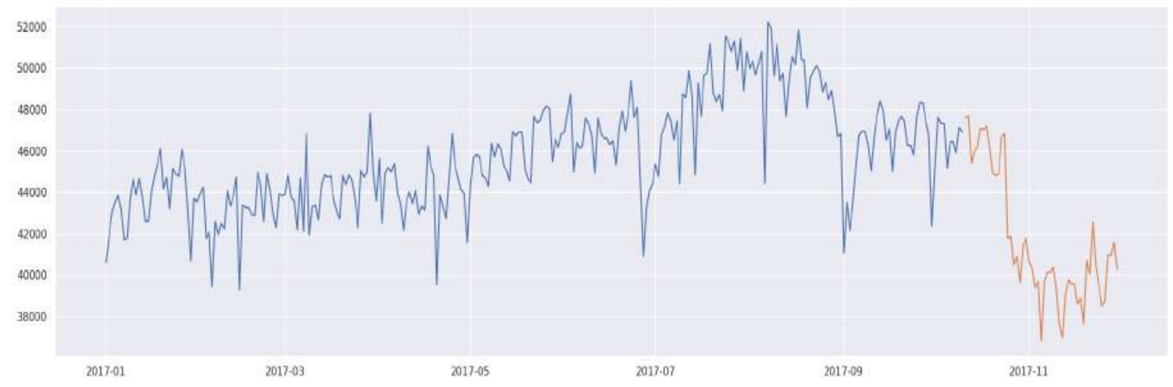

split 90% 10%


split 85% 15%


split 85% 15% (hours)



split 85% 15% (days-mean)



split 85% 15% (days-max)

Transformation on training set

For each type of transformation, 3 tests are carried out to calculate the stationarity of the series.
Two adfuller tests are performed, one of which with autolag "AIC" and regression "n". and then the third test is the KPSS.
The transformation that passes more tests out of 3 present is applied and which therefore makes the series stationary.

```
ADF: 0.40140938634740536 P-value: 0.8012536946479307
Statistiche test ADF : -2.0707564310960422 valore p :
0.25643496889702144 #Lag utilizzati : 7 Numero di
osservazioni : 275 0.25643496889702144 KPSS Statistic:
0.197499 p-value: 0.016938 Critial Values: 10%, 0.119
Critial Values: 5%, 0.146 Critial Values: 2.5%, 0.176
Critial Values: 1%, 0.216
```
test stazionarietà nessuna trasformazione ha superato 0/3
```
ADF: 0.5065880473298748 P-value: 0.8263875615161632
Statistiche test ADF : -2.1061791020692673 valore p :
0.24203722378478226 #Lag utilizzati : 7 Numero di
osservazioni : 275 0.24203722378478226 KPSS Statistic:
0.197730 p-value: 0.016851 Critial Values: 10%, 0.119
Critial Values: 5%, 0.146 Critial Values: 2.5%, 0.176
Critial Values: 1%, 0.216
```
test Stazionarietà trasformazione log ha superato 0/3
```
ADF: -4.415548218427781 P-value: 1.3488586993393774e-05
Statistiche test ADF : -4.413557150934702 valore p :
0.00028085645969753607 #Lag utilizzati : 14 Numero di
osservazioni : 261 0.00028085645969753607 KPSS
Statistic: 0.046162 p-value: 0.100000 Critial Values:
10%, 0.119 Critial Values: 5%, 0.146 Critial Values:
2.5%, 0.176 Critial Values: 1%, 0.216
```
test Stazionarietà differenziazione ha superato 3/3
DATI STAZIONARI diff

example of results

```
ADF: -1.0237275911299917 P-value: 0.278636900929693
Statistiche test ADF : -8.512978830154092 valore p :
1.1540335999365483e-13 #Lag utilizzati : 36 Numero di
osservazioni : 7210 1.1540335999365483e-13

KPSS Statistic: 0.842449 p-value: 0.010000Critial
Values:10%, 0.119 Critial Values: 5%, 0.146 Critial
Values: 2.5%, 0.176 Critial Values: 1%, 0.216
```
test Stazionarietà nessuna trasformazione ha superato 1/3

```
The actual p-value is smaller than the p-value
returned.
result = sm.tsa.kpss(value0.values, regression='ct')
ADF: -0.1533626995346955 P-value: 0.630981129404744
Statistiche test ADF : -8.702428451528421 valore p :
3.77764798214238e-14 #Lag utilizzati : 36 Numero di
osservazioni : 7210 3.77764798214238e-14

KPSS Statistic: 0.773947 p-value: 0.010000Critial
Values:10%, 0.119 Critial Values: 5%, 0.146 Critial
Values: 2.5%, 0.176Critial Values: 1%, 0.216
test Stazionarietà trasformazione log ha superato 1/3
The actual p-value is smaller than the p-value
returned.

result = sm.tsa.kpss(value0.values, regression='ct')
ADF: -14.048469388863852 P-value:
2.2889677007575344e-25 Statistiche test ADF :
-14.04749372796355 valore p : 3.221251684123142e-26
#Lag utilizzati : 29 Numero di osservazioni : 7193
3.221251684123142e-26

KPSS Statistic: 0.005050 p-value: 0.100000 Critial
Values: 10%, 0.119
Critial Values: 5%, 0.146 Critial Values: 2.5%, 0.176
Critial Values: 1%, 0.216
test Stazionarietà differenziazione ha superato 3/3
DATI STAZIONARI diff
```

example of results

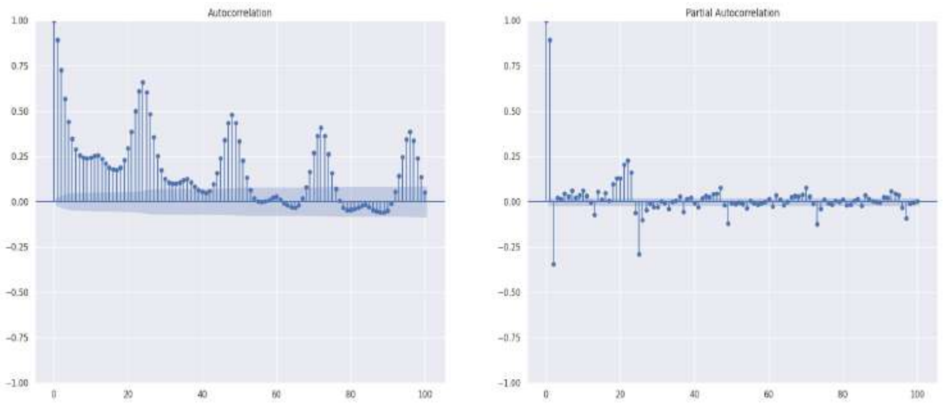
Therefore, based on these tests that were carried out to make the series stationary, the transformation that best transforms the data is automatically performed, making them stationary. However, it is also possible to declare in advance what type of transformation to use, whether logarithmic, seasonal difference or none, in order to have decisional priority with respect to the tests.

In addition, the ACF and PACF graphs were also inspected to check the parameters necessary for the next phase.

- ACF Shows how much the values in the series depend on previous values. In the ACF graph, the x-axis represents the lags and the y-axis represents the value of the autocorrelation. An indication of significant correlation to a given lag may suggest the presence of a possible pattern or seasonal dependence in the series.
- PACF, on the other side, measures the correlation between observed values and lagged values, taking into account the influence of intermediate lags. Shows the direct correlation between two points in time, controlling for indirect effects of other delays. The PACF graph also displays authors on lags, but each author represents the direct correlation between two points in time.

If the ACF graph rapidly reduces to zero and the PACF graph shows only a significant peak at lag 1 and then reduces to zero, it could be indicative of an AR(p) model where the current term depends only on the previous values.

If the ACF graph shows a slow decline and the PACF graph shows a series of significant peaks, it may be indicative of a moving average (MA(q)) model in which the current term depends on past error values.
If both the ACF and PACF graphs show a slow decline, it could be indicative of an AutoRegressive Moving Average (ARMA) model where the current term depends on both past values and past error values.



Example of ACF and PACF graphs

3. Arima

The ARIMA (AutoRegressive Integrated Moving Average) model is a powerful tool used in time series forecasting and analysis. It's particularly effective in capturing and predicting patterns and trends in sequential data, such as stock prices, weather patterns, or economic indicators.
The ARIMA model is defined by three key parameters: p, d, and q.

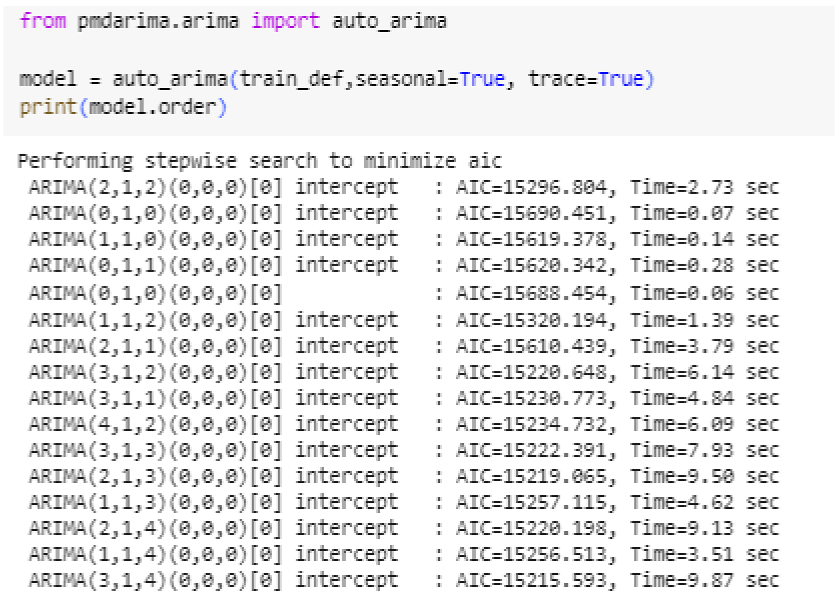
- p (Autoregressive Order): This parameter represents the number of autoregressive (AR) terms in the model. AR terms are essentially lagged values of the time series data. A higher p value means that the model considers more past observations to predict the future.
- d (Integration Order): The "d" parameter represents the number of differences needed to make the time series data stationary. Stationarity is a crucial assumption for ARIMA models, and "d" quantifies how many differencing operations are required to achieve it. A stationary time series has a constant mean and variance over time.
- q (Moving Average Order): The "q" parameter signifies the number of moving average (MA) terms in the model. MA terms capture the relationship between the current observation and past forecast errors. A higher q value implies that the model takes into account more past forecast errors.

In summary, the ARIMA model with its parameters p, d, and q is a versatile approach for time series analysis and forecasting. By adjusting these parameters, analysts and data scientists can tailor the model to suit the specific characteristics and patterns present in their data, ultimately improving the accuracy of predictions.

Using autocorrelation (ACF) and partial autocorrelation (PACF) plots to determine the p (AR order) and q (MA order) orders of the ARIMA model. The differential order d (order of differentiation) can be determined based on the difference needed to make the series stationary.
The definition of the parameters was divided into several parts, improving the previous step each time, then selecting the p, d and q parameters that best optimized the model as regards MAPE and AIC.

Arima 1

As a first step it was decided to use the "autoarima" function of the pmdarima.arima library performed both with and without seasonality. Checking the value of AIC.
From the first moment it emerged that this procedure was particularly expensive and slow, especially with very large datasets.



```
from pmdarima.arima import auto_arima

model = auto_arima(train_def,seasonal=True, trace=True)
print(model.order)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=15296.804, Time=2.73 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=15690.451, Time=0.07 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=15619.378, Time=0.14 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=15620.342, Time=0.28 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=15688.454, Time=0.06 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=15320.194, Time=1.39 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=15610.439, Time=3.79 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=15220.648, Time=6.14 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=15230.773, Time=4.84 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=15234.732, Time=6.09 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=15222.391, Time=7.93 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=15219.065, Time=9.50 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=15257.115, Time=4.62 sec
 ARIMA(2,1,4)(0,0,0)[0] intercept   : AIC=15220.198, Time=9.13 sec
 ARIMA(1,1,4)(0,0,0)[0] intercept   : AIC=15256.513, Time=3.51 sec
 ARIMA(3,1,4)(0,0,0)[0] intercept   : AIC=15215.593, Time=9.87 sec
```

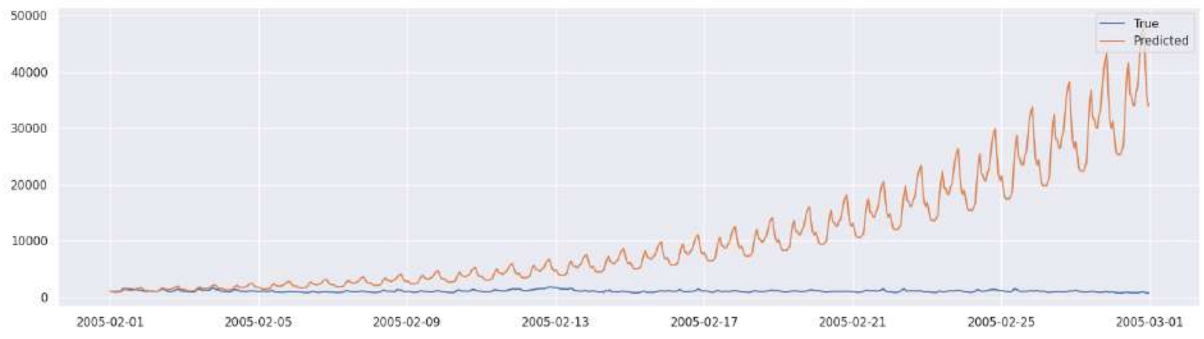
Example of stepwise auto-arima search

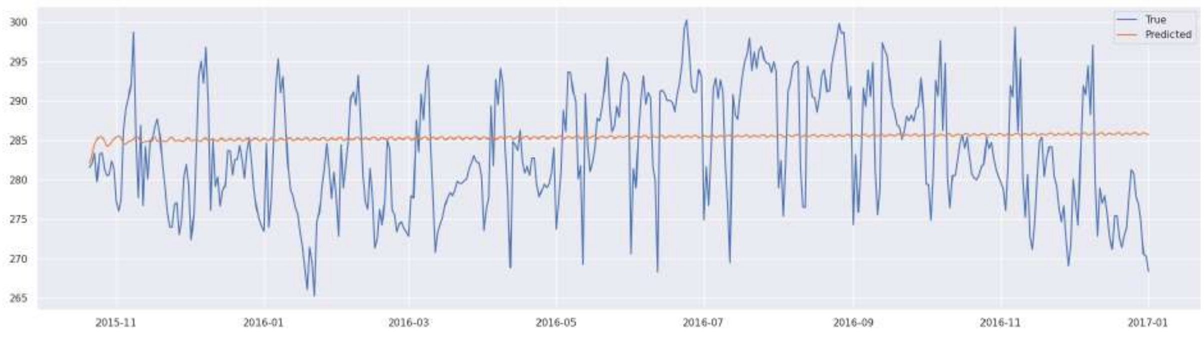
```
Best model:  ARIMA(4,1,1)(0,0,0)[0]
Total fit time: 164.151 seconds
(4, 1, 1)
ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=-17164.240,
Time=12.13 sec
```

```
model_2 = ARIMA(train_log, order=(4, 1, 1),
seasonal_order=(0, 0, 0, s)).fit()
```

Obtaining values that are too centered on the average and therefore difficult to optimize.
Given the linear correlation between variance and mean and having confirmed the need to differentiate the series due to non-stationarity on the mean following the Augmented DIckey-Fuller test, I try to build a new model with d=1.

```
model_3 = ARIMA(train_log, order=(0, 1, 0),
seasonal_order=(0, 1, 0, s)).fit()
```

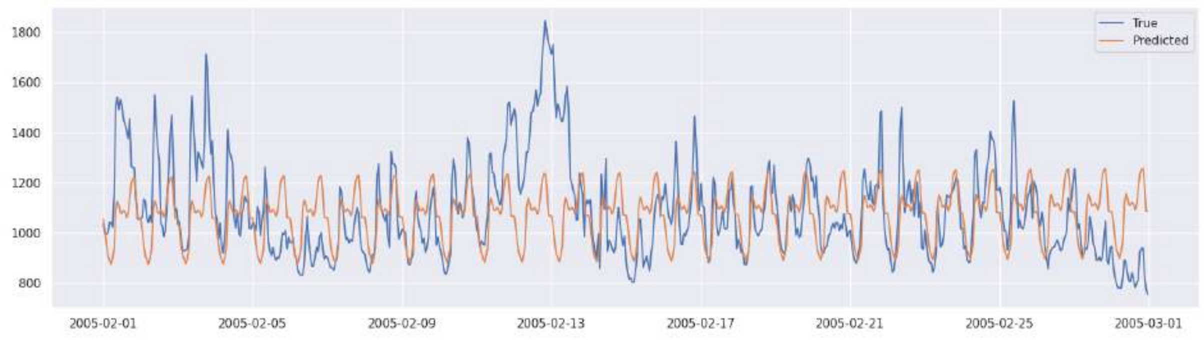




Example of auto-arima predictions on test set

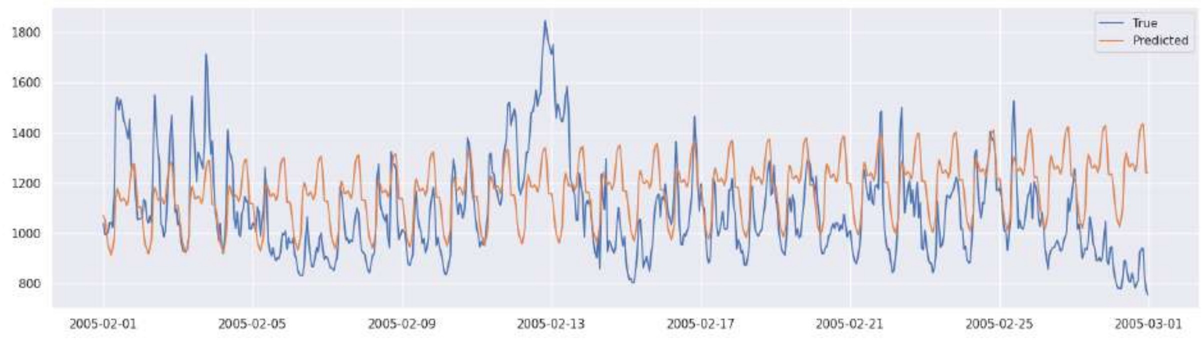
Obtaining values that are too high, I proceed to add SMA(1) as acf goes to 0 after 1 lag with 24 days of seasonality, so I try the model again:

```
model_4 = ARIMA(train_log, order=(0, 1, 0),
seasonal_order=(0, 1, 1, s)).fit()
```



```
AIC:  -19660
MAPE: 0.1187
```

```
model_5 = ARIMA(train_log, order=(4, 1, 0),
seasonal_order=(0, 1, 1, s)).fit()
```



Example of results based on a given time serie

```
AIC:  -19943
MAPE: 0.1617
```

Arima 2

p-->autoregression orders (number of previous data points to predict the current data point)
d-->differentiation (number of times differentiation is applied)
q-->moving average (previous forecast errors , residual correlations between successive data points)

I keep the values previously found with the best results and try to search for the best values of p and q to optimize the model.
Keeping the differentiation number hold to 1.
Here we see an example table of the values obtained from the grid search by keeping the value d fixed and checking the value of MAPE and AIC by scrolling the parameters p and q.

p,q	AIC	MAPE
0, 0	-19660.7015743033	0.11873263010462422
0,1	-19663.53057186855	0.11805237230254015
0,2	-19834.41160319819	0.13547936850029169
0,3	-19945.999634571395	0.17843766838561478
1,0	-19662.280303344778	0.11819551465544526
1,1	-20088.97571089223	0.12449509920733115

1,2	**-20214.212006708607**	0.11713618232820622
1,3	**-20212.59730502269**	**0.11630463283959662**
2,0	-19797.716516327215	0.12564108707226498
2,1	-20209.643133246223	0.12001701935949091
2,2	**-20211.461154348333**	0.11741398699765378
2,3	-20211.378403529074	0.11694986010701676
3,0	-19843.981349844034	0.13907655353460222
3,1	-20210.94583354432	0.11657288811188843
3,2	-20207.398195786212	**0.11624826503758001**
3,3	-20209.699841217684	**0.11647668792459881**

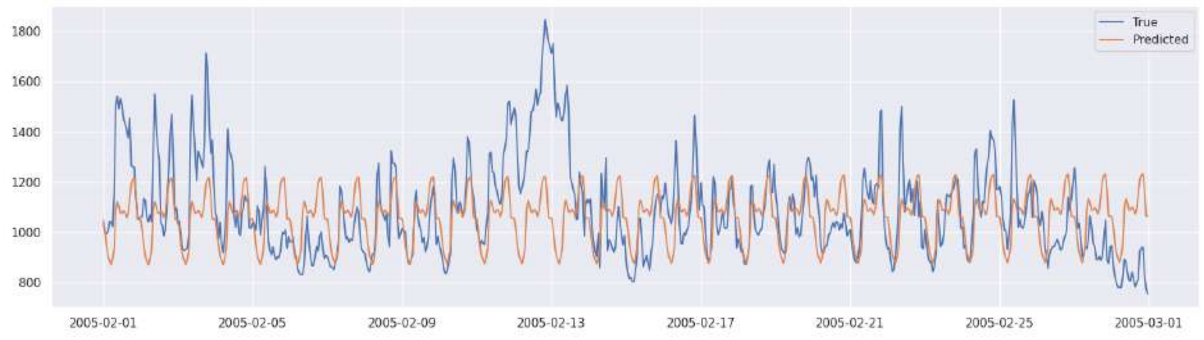
Example of grid search of p and q (hours dataset)

```
    p  q    aAIC      MAPE
9   2  1  4904.5   4.154739
6   1  2  4904.9   4.162337
10  2  2  4907.8   4.233872
5   1  1  4903.8   4.233896
14  3  2  4907.2   4.296502
13  3  1  4904.8   4.333838
7   1  3  4905.6   4.399239
3   0  3  4914.2   4.834357
15  3  3  4967.1   5.384110
11  2  3  4902.0   5.611173
12  3  0  4929.4  24.213694
2   0  2  4929.9  24.454703
1   0  1  4929.1  25.839850
8   2  0  4932.3  26.375190
4   1  0  4931.5  26.953846
0   0  0  4945.0  27.499756
```

Example of grid search of p and q (daily dataset 293 rows)

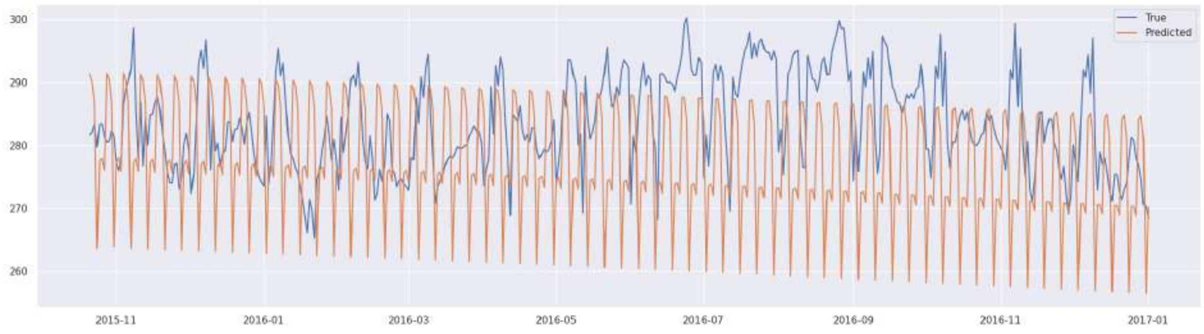
Using minor mape features

```
mod1 = ARIMA(train_def, order=(3, 1, 2), seasonal_order=(0, 1, 1,
24)).fit()
```



```
AIC:  -20207
MAPE: 0.1162
```

Example with logarithmic transformation



Example with seasonal diff transformation

```
AIC:  -20207
MAPE: 0.1162
```

This is to understand how the transformation parameter that is often used, requires a more in-depth and less immediate search for values.
The choice to keep the seasonal order the same (0,1,1,s) was made for two main reasons.

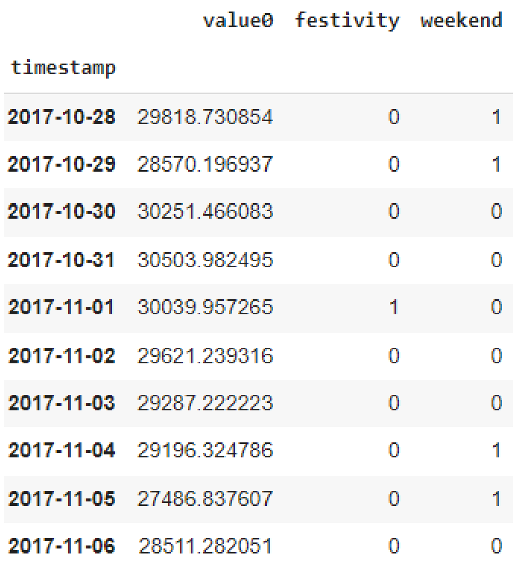
Firstly by carrying out a transformation operation of the time series and very often making a differentiation this turned out to be the most optimal fixed parameter on the basis of different time series on which the tests were carried out.
Secondly, due to a choice of automation in the creation of the model, it would have been very expensive in terms of time and resources to do a general parameter search comparing many parameters together.
Obviously for the initial phase it was decided to carry out tests based on the same time serie, but trying to study the trend in such a way as to make the creation of the model adaptable to more different series with non-similar characteristics.

Arima 3

The third step in this search for optimal parameters is to include additional components in the model to capture holiday-specific seasonal effects.

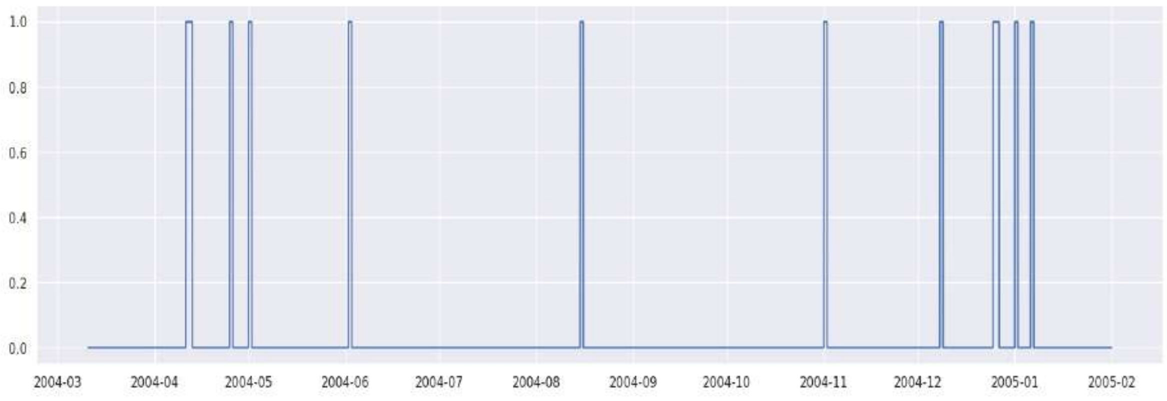
- Use sinusoidal components to represent holiday seasonal patterns.
- Each sinusoidal component will represent a specific holiday and will be characterised by a period length (in this case 1 year), an amplitude and a phase.

In practice, various sinusoidal components with different amplitudes and phases can be defined to represent different holidays. These components will then be included in the seasonal ARIMA model to improve forecasting ability during the holidays.
I add sinusoids for holidays using the holidays library, creating dummy variables for Italian annual holidays and weekends in both the training set and the test set.

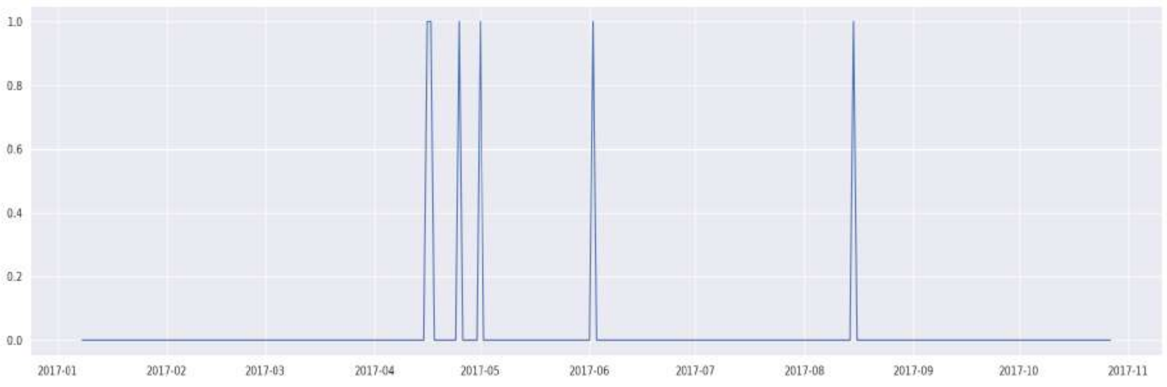


	value0	festivity	weekend
timestamp			
2017-10-28	29818.730854	0	1
2017-10-29	28570.196937	0	1
2017-10-30	30251.466083	0	0
2017-10-31	30503.982495	0	0
2017-11-01	30039.957265	1	0
2017-11-02	29621.239316	0	0
2017-11-03	29287.222223	0	0
2017-11-04	29196.324786	0	1
2017-11-05	27486.837607	0	1
2017-11-06	28511.282051	0	0

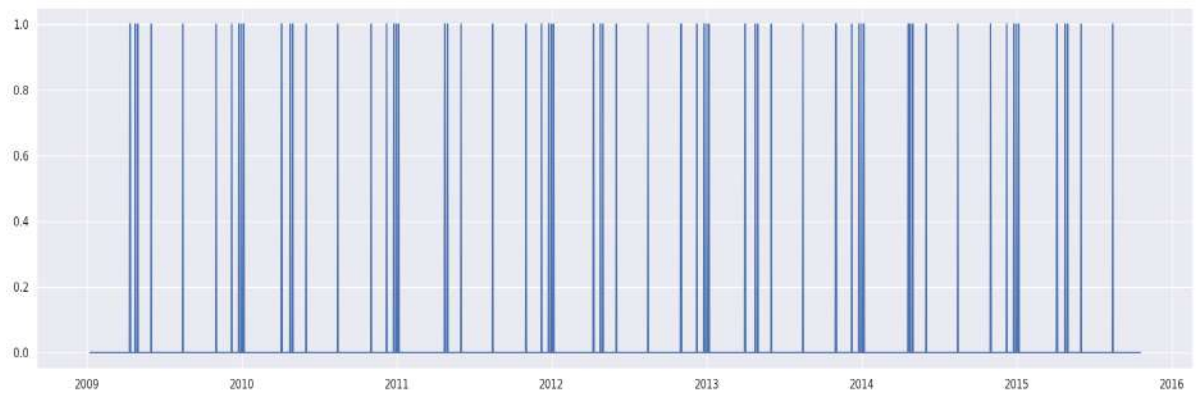
Structure example of festivity dataset



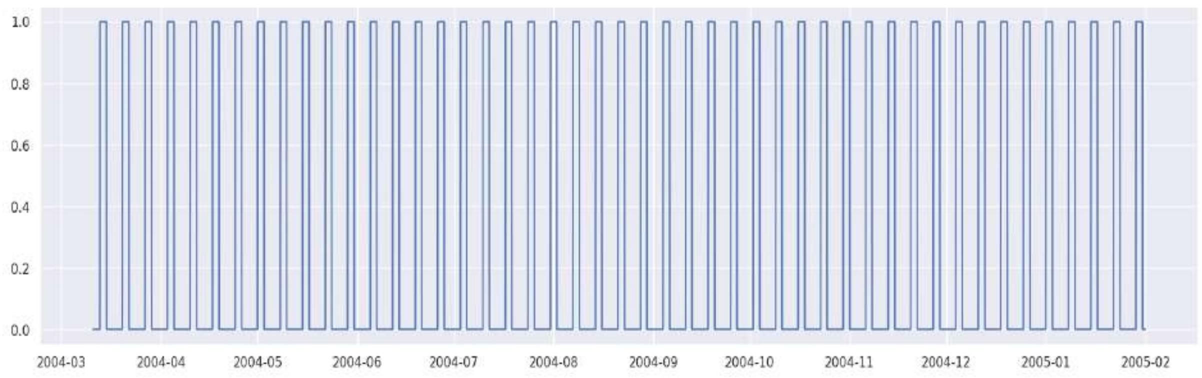
plot festivity (2004-03 to 2005-02)



plot festivity of 2017

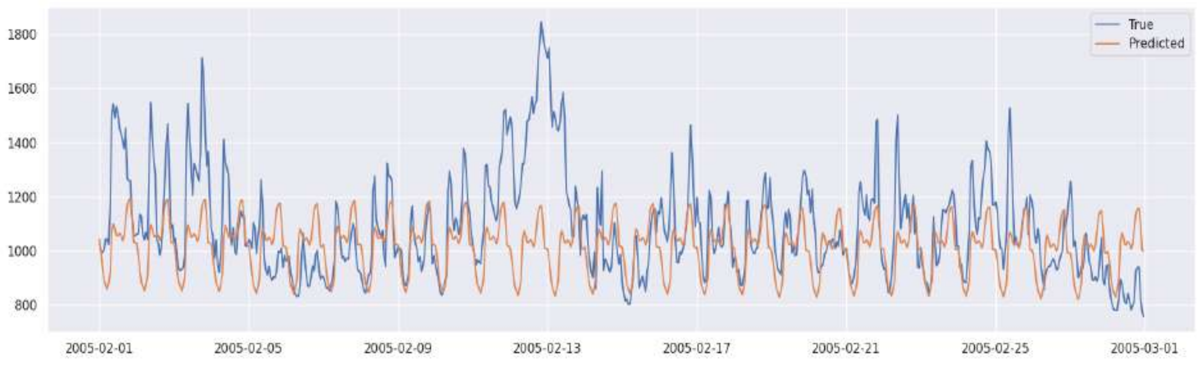


plot festivity from 2009 to 2016



plot weekends (2004-03 to 2005-02)

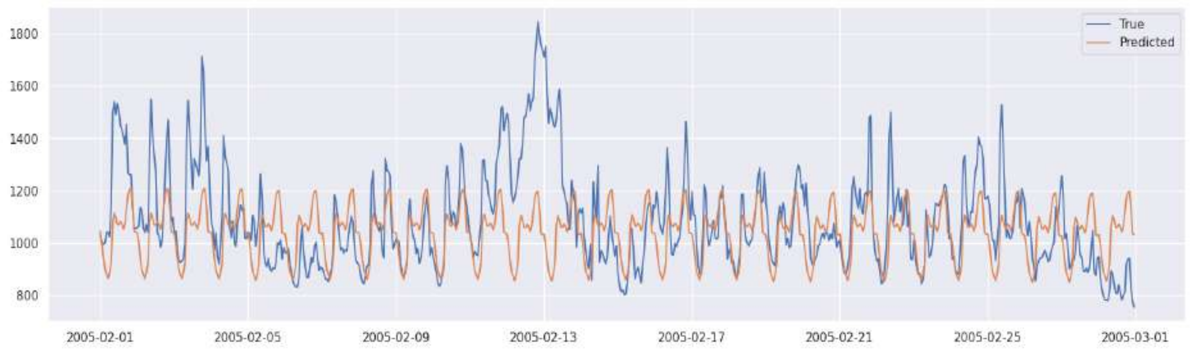
```
3 e 2
model = ARIMA(train_def['value0'], order=(3, 1, 2),
seasonal_order=(0, 1, 1, s),exog=exog_train).fit()
```



example with sinusoidal components to represent holiday based on the best p and q

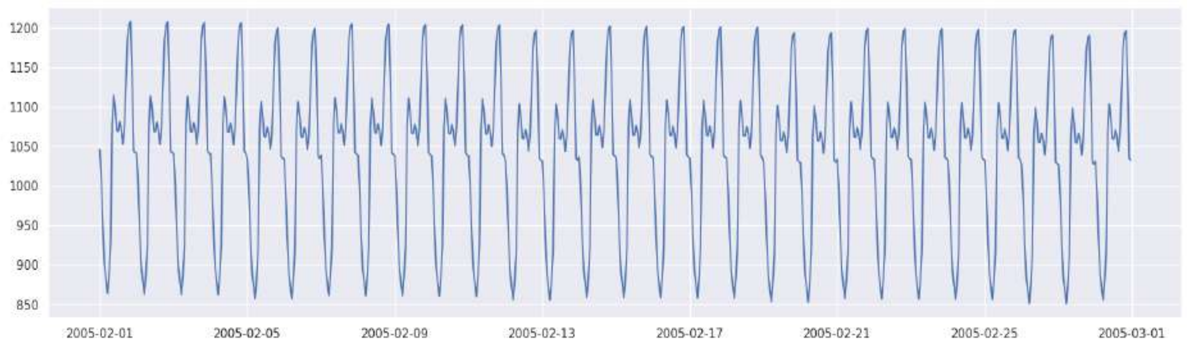
```
AIC:  -20201
MAPE: 11.459999999999999
```

```
model2 = ARIMA(train_log2['CO'], order=(2, 1, 2),
seasonal_order=(0, 1, 1, 24),exog=exog_train).fit()
```



example with sinusoidal components to represent holiday based on others p and q

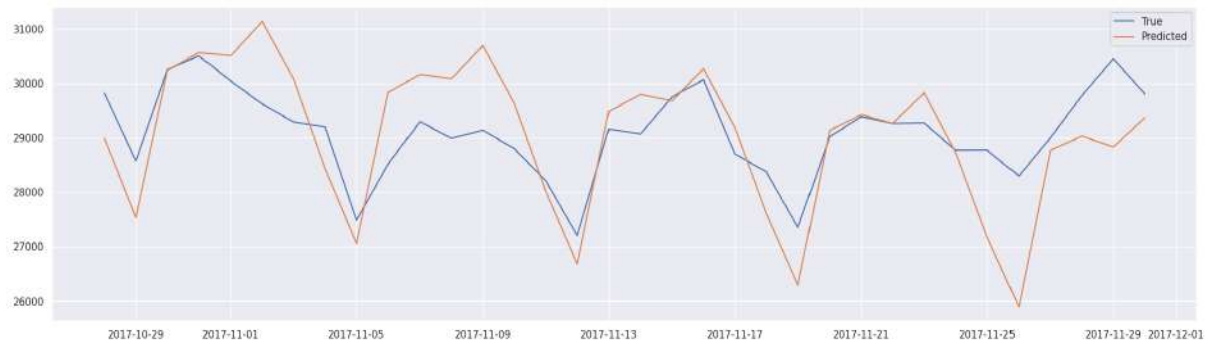
```
mape:11.40749981154510
```



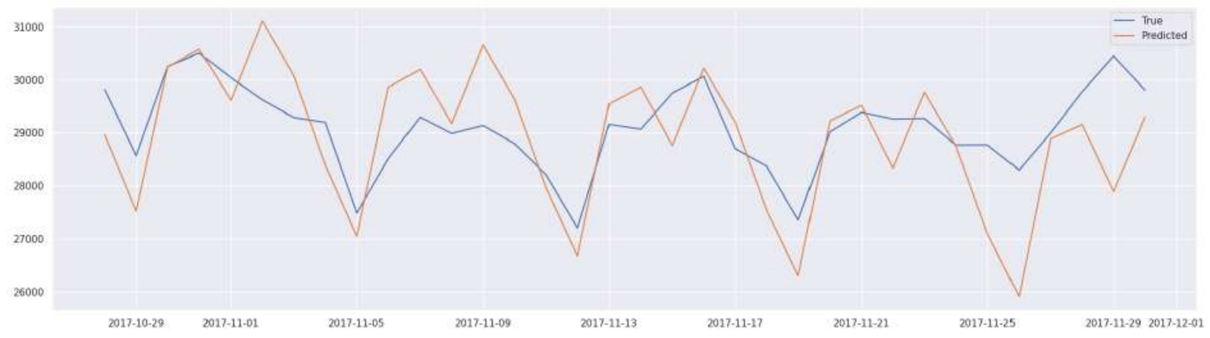
best model test set prediction

We can see how the addition of holidays in this case did not particularly influence the forecast. This choice of using holidays or not can be imposed with a parameter based on the results obtained in the initial exploration and preprocessing phase where the average trend during the week, hours or months using boxplots.

Now let's see with a comparison how the trend changes on example data by creating the model using holidays and weekends or not using them:



Daily dataset predictions without festivity



Daily dataset predictions with festivity

Arima 4

Seasonality refers to repetitive patterns that occur at regular intervals in a time series data, such as daily, monthly, or yearly patterns. These patterns can significantly impact the underlying data and, if not properly addressed, can lead to suboptimal forecasts. To tackle this challenge, one effective technique is to introduce sinusoidal regressors into our forecasting models.
In time series forecasting, seasonality is crucial, and sinusoidal regressors help capture it effectively. They enable us to model repetitive patterns in the data, enhancing accuracy and decision-making. To find the optimal number of sinusoidal regressors, we

use AIC and MAPE metrics to strike the right balance between model complexity and performance. This approach ensures accurate forecasts and a better understanding of time series data.

- Incorporating sinusoidal regressors improves the accuracy of our forecasts, as the model can more precisely estimate the seasonal fluctuations in the data.
- Sinusoidal regressors provide interpretable insights into the nature of seasonality. We can examine the amplitude and phase of the sinusoids to understand how they influence the time series.

```
### RESULT
# [1, -20242.2, 0.113]
# [2, -20213.6, 0.168]
# [3, -20247.6, 0.11]
# [4, -20207.0, 0.168]
# [5, -20276.8, 0.121]
# [6, -20374.5, 0.145]
```

Finding optimal number of sinusoidal regressors

How many sinusoids should we include in our model? The answer lies in finding the optimal number of sinusoidal components that strike a balance between model complexity and performance. One way to determine this is through a grid search approach, where we experiment with various numbers of sinusoidal regressors.

We can assess the performance of different models using two key metrics: the Akaike Information Criterion (AIC) and the Mean Absolute Percentage Error (MAPE).
By systematically varying the number of sinusoidal regressors and observing how AIC and MAPE change, we can pinpoint the optimal number that minimizes both metrics. This approach ensures that our model effectively captures seasonality without introducing unnecessary complexity.
In conclusion, incorporating sinusoidal regressors into time series forecasting models is a valuable technique for addressing seasonality patterns. By conducting a grid search and evaluating models using AIC and MAPE, we can fine-tune our models to provide

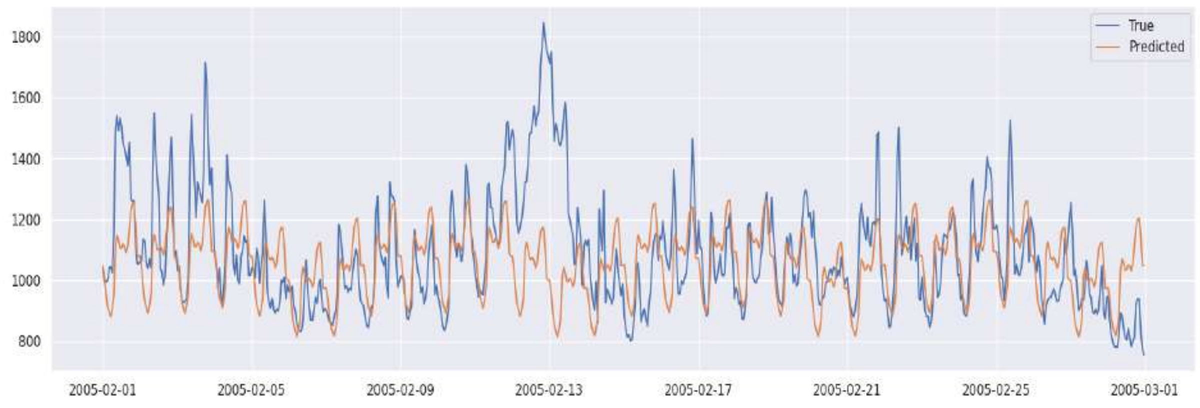
more accurate and reliable forecasts, ultimately assisting in better decision-making and improved understanding of time series data.

Seasonality 24*7 weekly

```
model_sin = ARIMA(train_def, order=(2, 1, 2), seasonal_order=(0, 1, 1, s), exog=sinusoids_train).fit()
```

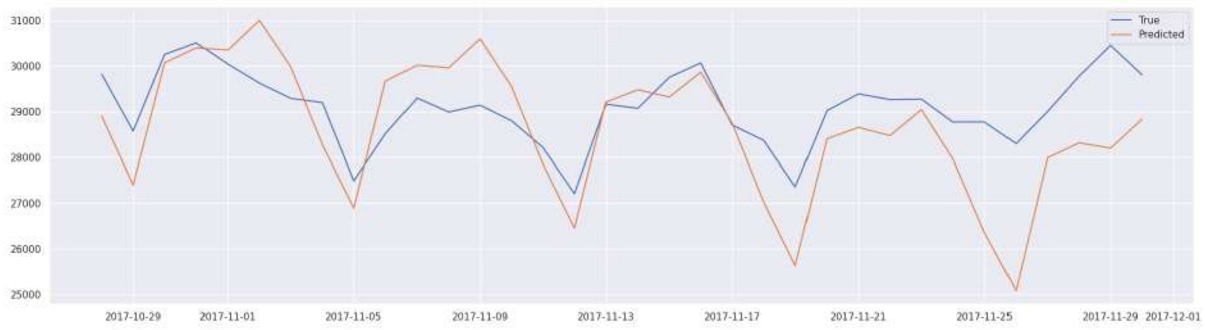
Also in this case, since it is an automated scalar procedure, I use the best parameters that were found in the previous grid searches.

```
AIC:  -20247
MAPE: 0.1101
```



Test set forecasting with regressors

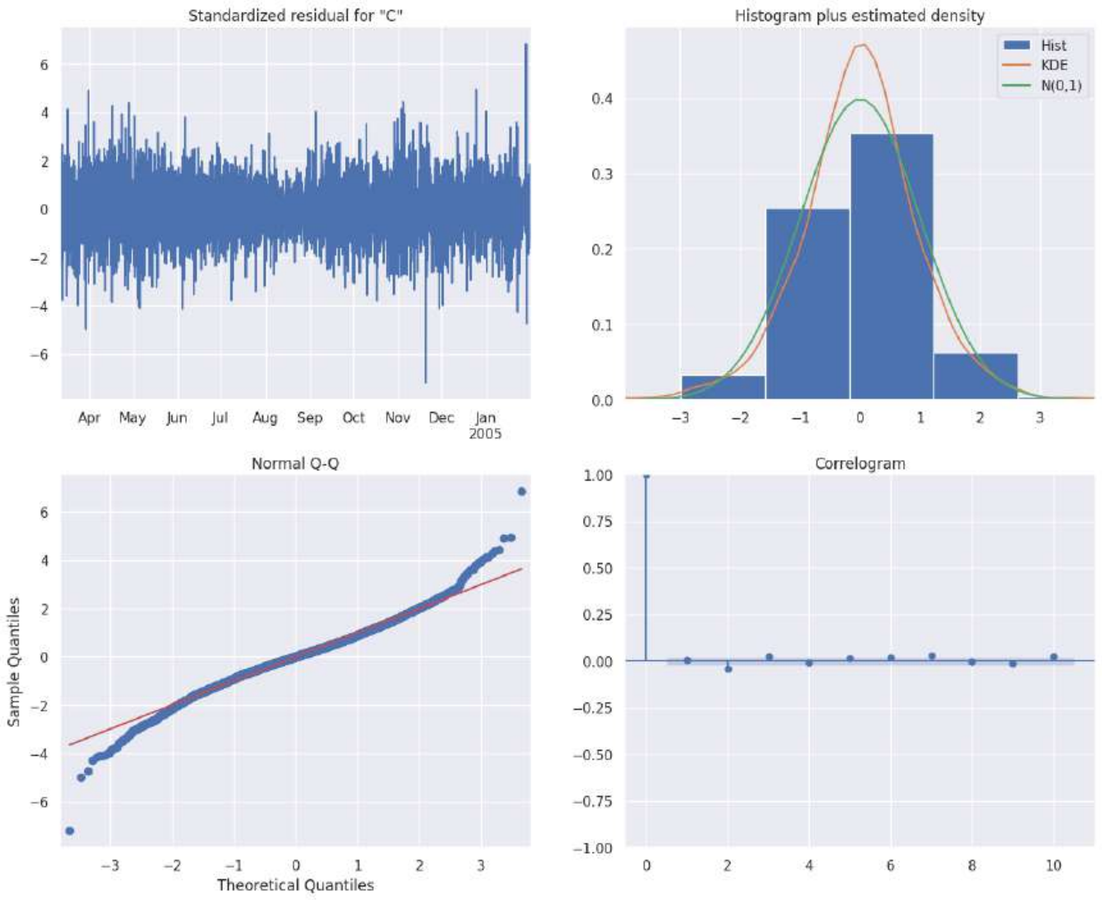
As can be seen in this case, the structure of the series is the same but influenced by a trend that changes its performance based on seasonality and the regressors that were used to map it.



Test set forecasting with regressors on daily dataset

```
AIC: 4911
MAPE: 4.549335018522167
```

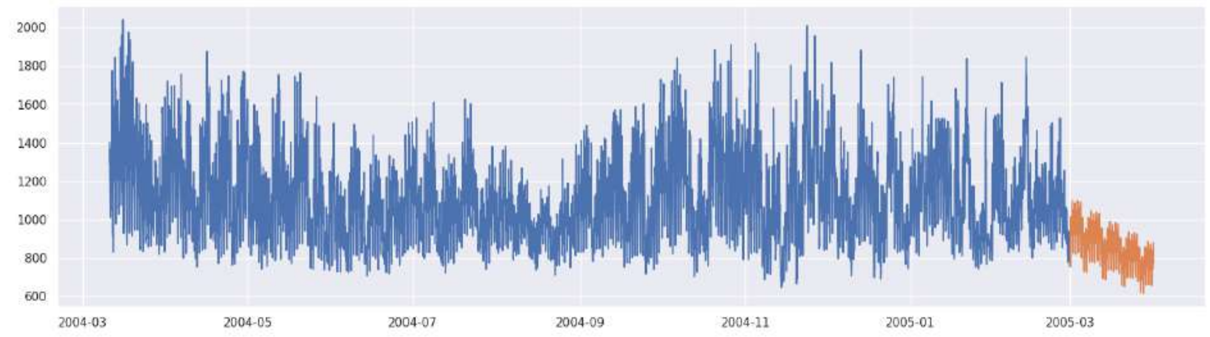
To conclude we can deduce that in this case we can see how by modelling with sinusoids the trend adapts better to the trend offering a better forecast.
model.diagnostic:

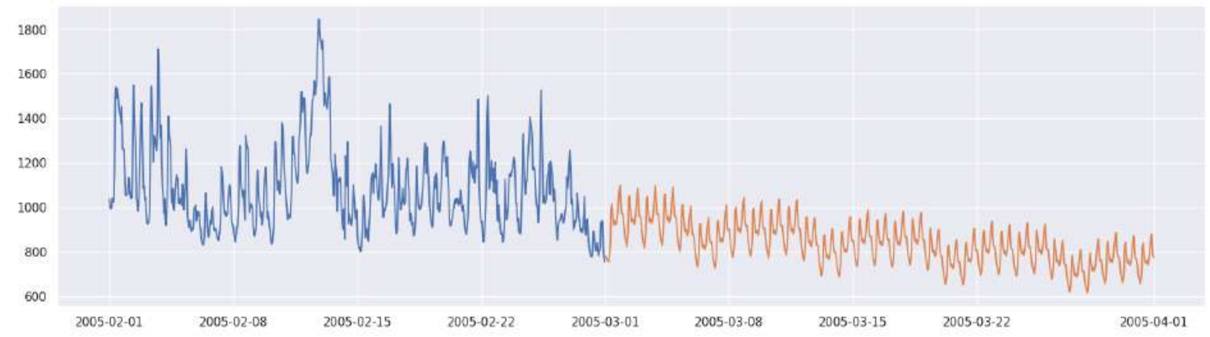


Example of model diagnostics

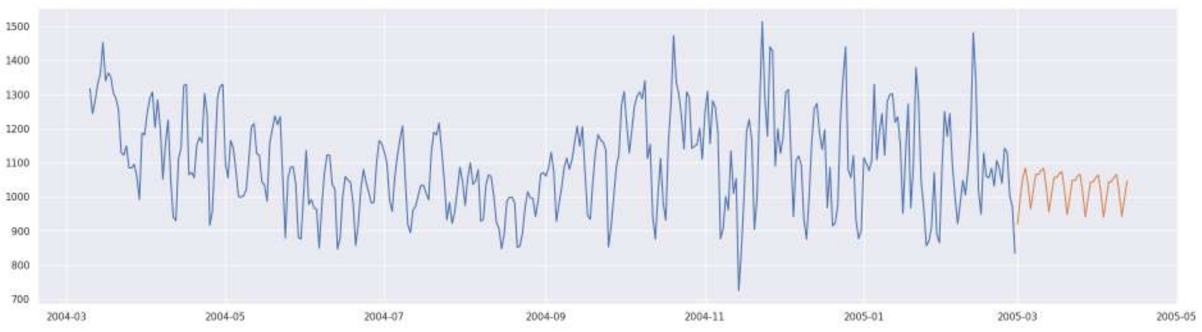
Arima Predictions

```
test_arima_model(order=(p,1,q), season_order=(0,1,1,s))
```

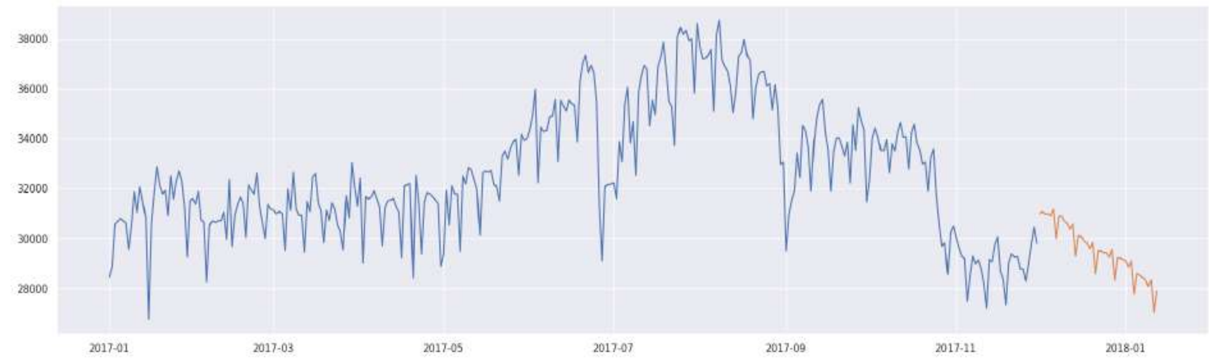




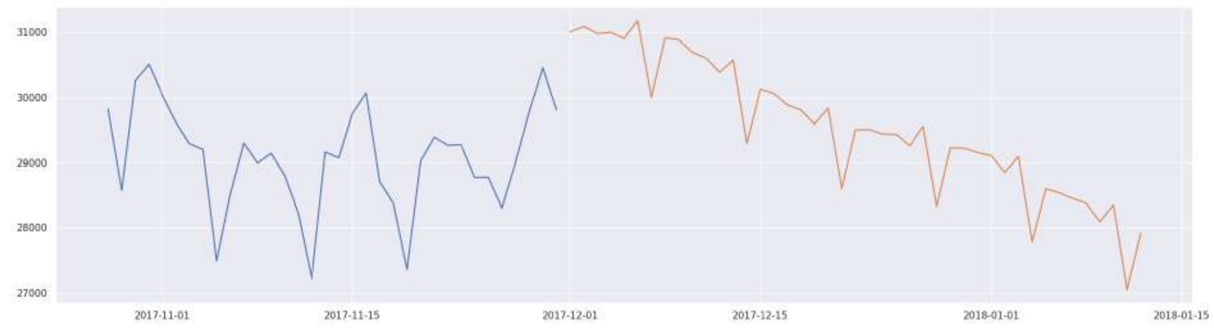
Best model Arima predictions



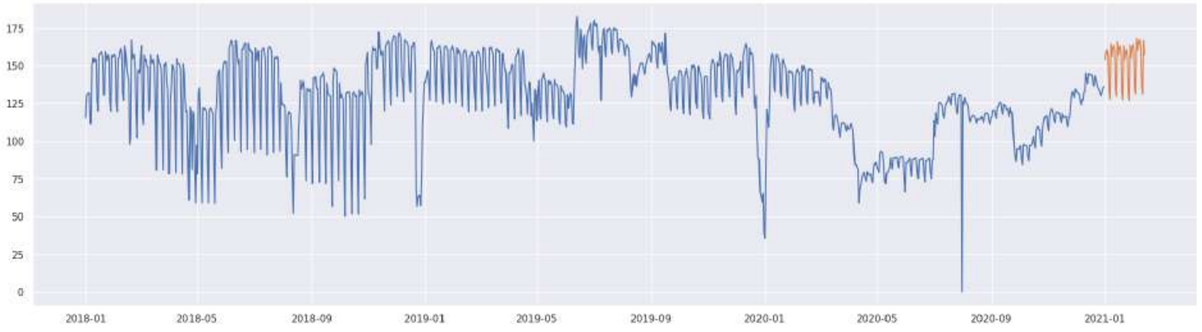
Example of predictions based on different scale



Example of prediction on a daily dataset

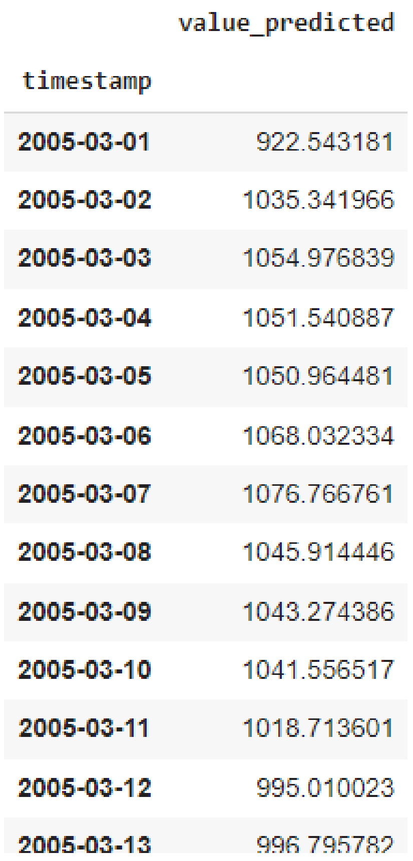


test set and future predictions compared



Daily dataset energy consumption predictions

Once the forecasts are finished and having ascertained which is the best model, a dataframe is created with the future forecasts based on the number or time period of the predictions requested and is automatically saved into the work folder.

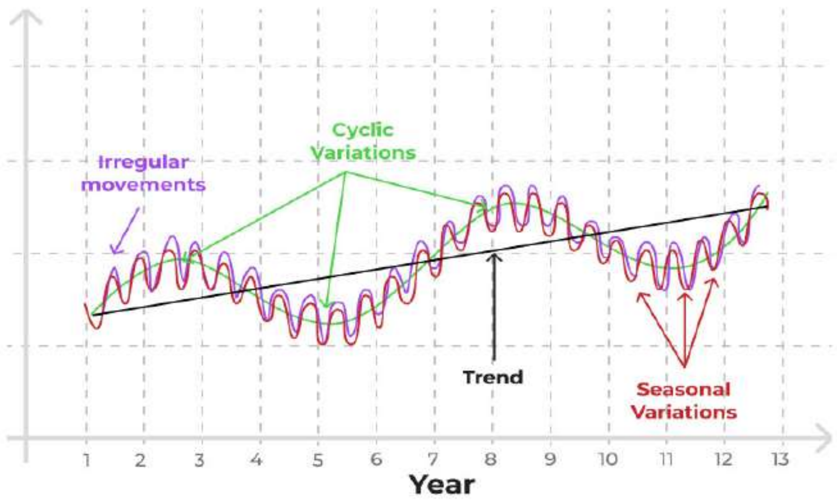


timestamp	value_predicted
2005-03-01	922.543181
2005-03-02	1035.341966
2005-03-03	1054.976839
2005-03-04	1051.540887
2005-03-05	1050.964481
2005-03-06	1068.032334
2005-03-07	1076.766761
2005-03-08	1045.914446
2005-03-09	1043.274386
2005-03-10	1041.556517
2005-03-11	1018.713601
2005-03-12	995.010023
2005-03-13	996.795782

Example of future predictions daily df

6. UCM

The Unobserved Components Model, often abbreviated as UCM, is a powerful tool in the field of time series forecasting. It is a statistical model designed to uncover hidden patterns and components within time series data. The UCM is particularly useful for making predictions when the data is influenced by various underlying factors that are not directly observable.
The UCM forecasting model is based on the idea that a time series is composed of several components, each of which represents a specific aspect of the data generating process.
These components typically include a trend component, a seasonal component, and an error component:



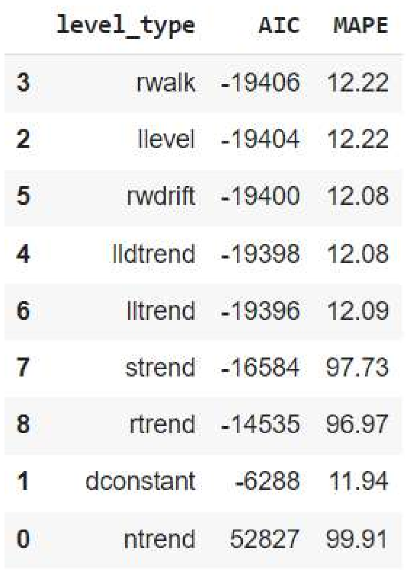
d

Time series data components

- Trend component: represents the general direction of the process, i.e. whether the time series shows an increasing trend, decreasing trend, or no trend. Captures the long-term direction or trajectory of the time series data.
- Seasonal Component: Represents regular or cyclic patterns that repeat in the time series over a fixed time interval.This is especially important for time series data that exhibit regular fluctuations, such as monthly or yearly seasonality.
- Cycle Component: Represents cyclical patterns that occur in the time series with periods longer than seasonality
- Error Component: Represents the residual variability in the time series that cannot be explained by the other components. The random fluctuations or noise in the data that cannot be attributed to the trend or seasonal patterns. It accounts for the uncertainty in the forecasting process.

The UCM model uses mathematical techniques to estimate the parameters of these components, allowing it to decompose the observed time series into its underlying parts. Once these components are identified, they can be combined to make accurate and reliable forecasts for future time points.
It helps analysts understand the various factors influencing time series data and make informed predictions.

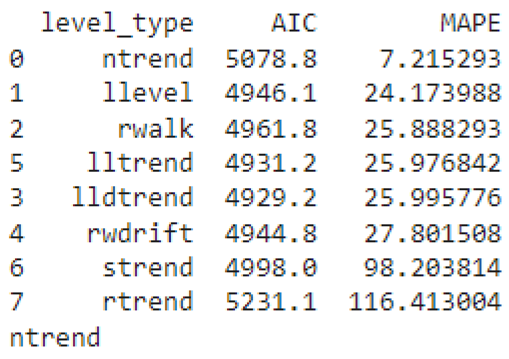
I start by doing grid search to find the best level parameter, obtaining the following results:



	level_type	AIC	MAPE
3	rwalk	-19406	12.22
2	llevel	-19404	12.22
5	rwdrift	-19400	12.08
4	lldtrend	-19398	12.08
6	lltrend	-19396	12.09
7	strend	-16584	97.73
8	rtrend	-14535	96.97
1	dconstant	-6288	11.94
0	ntrend	52827	99.91

grid search example for level type trend

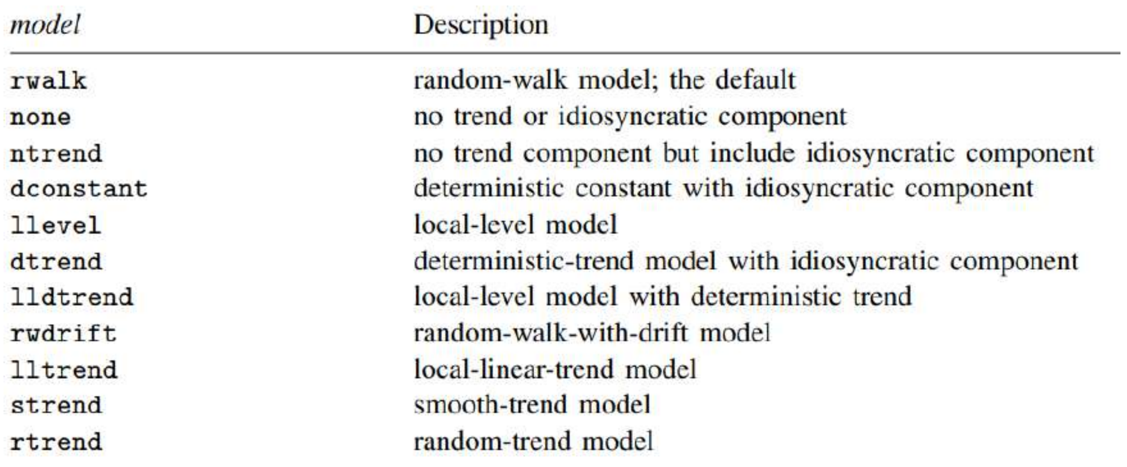
Sometimes the choice of level is more difficult, because the values are similar, other times the choice is more obvious.



```
  level_type     AIC        MAPE
0     ntrend  5078.8    7.215293
1     llevel  4946.1   24.173988
2      rwalk  4961.8   25.888293
5    lltrend  4931.2   25.976842
3   lldtrend  4929.2   25.995776
4    rwdrift  4944.8   27.801508
6     strend  4998.0   98.203814
7     rtrend  5231.1  116.413004
ntrend
```

ntrend level type

It was decided not to use the "dconstant" level as after various tests on different datasets it turned out to be too adaptable for any series, making the model too general and less specific for the dataframe in question that we want to consider.

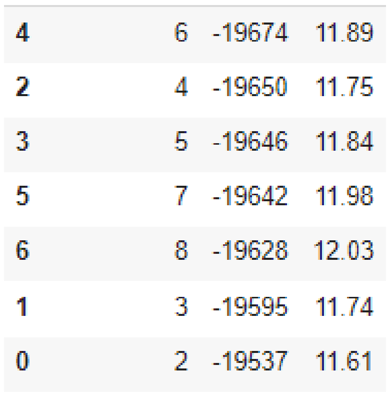


model	Description
rwalk	random-walk model; the default
none	no trend or idiosyncratic component
ntrend	no trend component but include idiosyncratic component
dconstant	deterministic constant with idiosyncratic component
llevel	local-level model
dtrend	deterministic-trend model with idiosyncratic component
lldtrend	local-level model with deterministic trend
rwdrift	random-walk-with-drift model
lltrend	local-linear-trend model
strend	smooth-trend model
rtrend	random-trend model

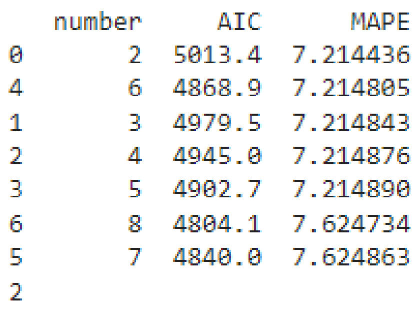
type of levels

I found the best number of harmonics using the level that obtained a lower value than AIC, using AIC and not MAPE or MAE as it turned out to be more reliable after a series of tests.
Harmonics are used to capture the periodic or seasonal structure of the series, the greater the number of harmonics considered, the greater the flexibility of the model in capturing the different seasonal patterns present in the time series.
However, too many harmonics could lead to overfitting of the training data and to a loss of generalisation ability for new data.

I'm looking for the best number of harmonics for the seasonality s:



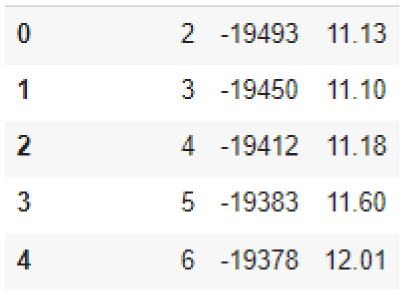
4	6	-19674	11.89
2	4	-19650	11.75
3	5	-19646	11.84
5	7	-19642	11.98
6	8	-19628	12.03
1	3	-19595	11.74
0	2	-19537	11.61



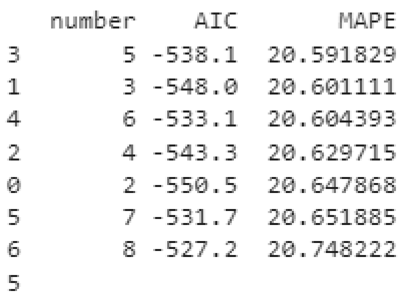
```
   number     AIC      MAPE
0       2  5013.4  7.214436
4       6  4868.9  7.214805
1       3  4979.5  7.214843
2       4  4945.0  7.214876
3       5  4902.7  7.214890
6       8  4804.1  7.624734
5       7  4840.0  7.624863
2
```

Grid search harmonics numbers

I find the best number of harmonics for seasonality s1 (largest scale order) based on the best number of harmonics for seasonality s.

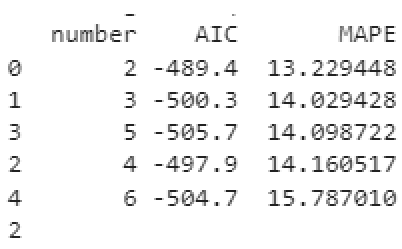


0	2	-19493	11.13
1	3	-19450	11.10
2	4	-19412	11.18
3	5	-19383	11.60
4	6	-19378	12.01



```
   number    AIC       MAPE
3       5 -538.1  20.591829
1       3 -548.0  20.601111
4       6 -533.1  20.604393
2       4 -543.3  20.629715
0       2 -550.5  20.647868
5       7 -531.7  20.651885
6       8 -527.2  20.748222
5
```

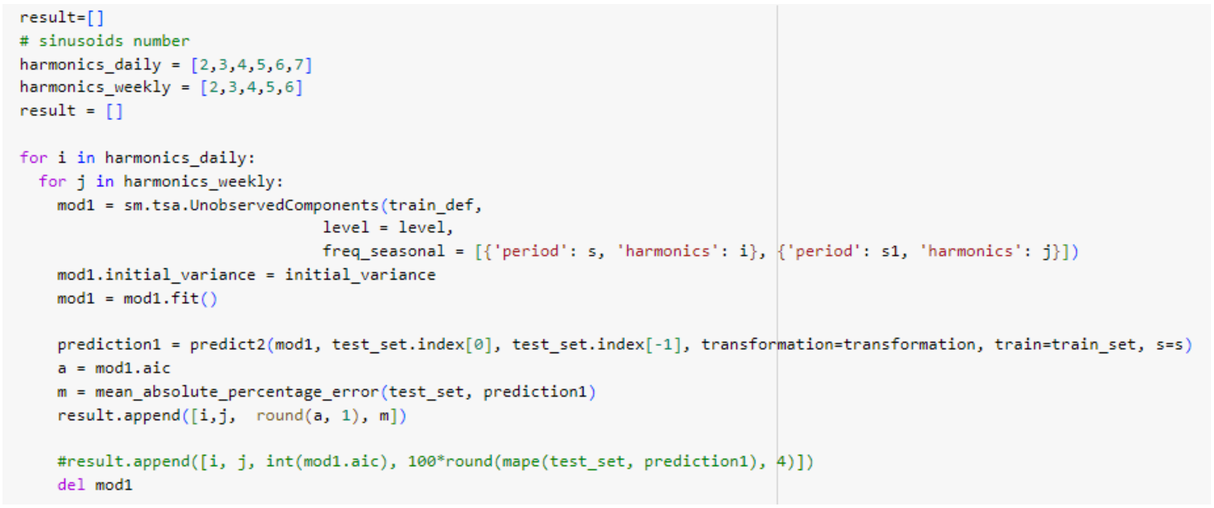
Period s



```
   number    AIC       MAPE
0       2 -489.4  13.229448
1       3 -500.3  14.029428
3       5 -505.7  14.098722
2       4 -497.9  14.160517
4       6 -504.7  15.787010
2
```

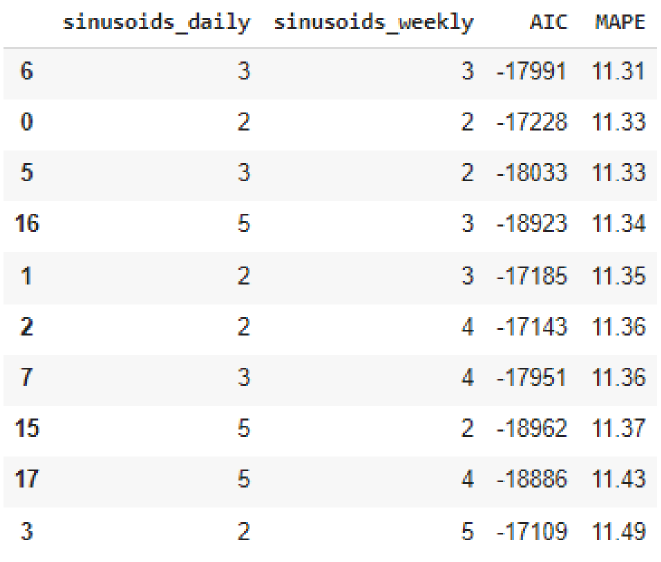
Period s1

Once i have found the best number of harmonics for seasonal scale s and the best number of harmonics for seasonal scale s1 based on the result obtained previously.
I now aim to find the best combination of harmonics for the two different time scales within the series to optimise the results.
S and S1 values are the seasonal periods we talked about previously and are chosen based on the time scale frequency of the series.

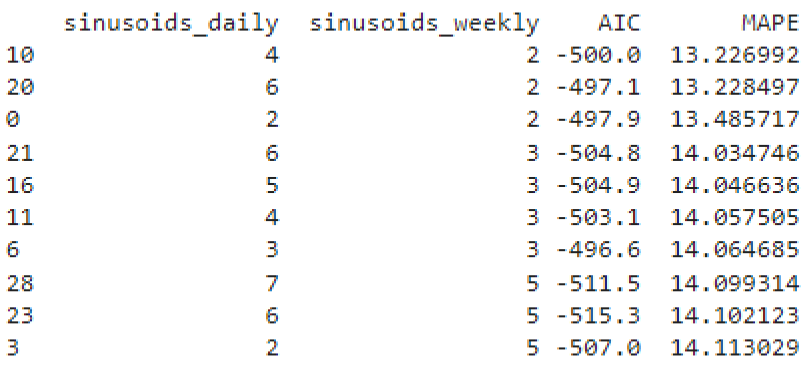


```python
result=[]
# sinusoids number
harmonics_daily = [2,3,4,5,6,7]
harmonics_weekly = [2,3,4,5,6]
result = []

for i in harmonics_daily:
  for j in harmonics_weekly:
    mod1 = sm.tsa.UnobservedComponents(train_def,
                            level = level,
                            freq_seasonal = [{'period': s, 'harmonics': i}, {'period': s1, 'harmonics': j}])
    mod1.initial_variance = initial_variance
    mod1 = mod1.fit()

    prediction1 = predict2(mod1, test_set.index[0], test_set.index[-1], transformation=transformation, train=train_set, s=s)
    a = mod1.aic
    m = mean_absolute_percentage_error(test_set, prediction1)
    result.append([i,j,  round(a, 1), m])

    #result.append([i, j, int(mod1.aic), 100*round(mape(test_set, prediction1), 4)])
    del mod1
```

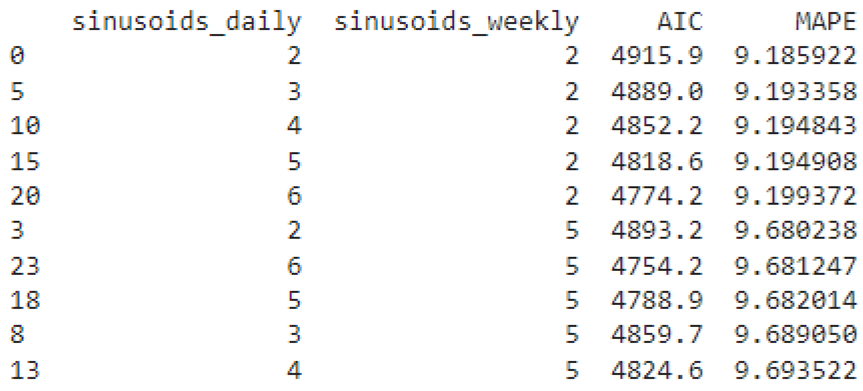
Best combination of harmonics for seasonalities



	sinusoids_daily	sinusoids_weekly	AIC	MAPE
6	3	3	-17991	11.31
0	2	2	-17228	11.33
5	3	2	-18033	11.33
16	5	3	-18923	11.34
1	2	3	-17185	11.35
2	2	4	-17143	11.36
7	3	4	-17951	11.36
15	5	2	-18962	11.37
17	5	4	-18886	11.43
3	2	5	-17109	11.49



```
    sinusoids_daily  sinusoids_weekly    AIC       MAPE
10                4                 2 -500.0  13.226992
20                6                 2 -497.1  13.228497
0                 2                 2 -497.9  13.485717
21                6                 3 -504.8  14.034746
16                5                 3 -504.9  14.046636
11                4                 3 -503.1  14.057505
6                 3                 3 -496.6  14.064685
28                7                 5 -511.5  14.099314
23                6                 5 -515.3  14.102123
3                 2                 5 -507.0  14.113029
```



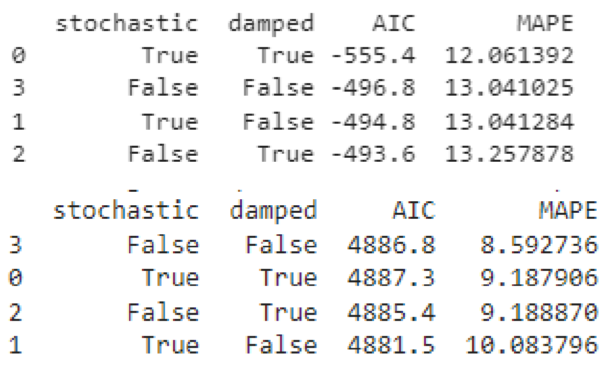
```
     sinusoids_daily  sinusoids_weekly     AIC      MAPE
0                  2                 2  4915.9  9.185922
5                  3                 2  4889.0  9.193358
10                 4                 2  4852.2  9.194843
15                 5                 2  4818.6  9.194908
20                 6                 2  4774.2  9.199372
3                  2                 5  4893.2  9.680238
23                 6                 5  4754.2  9.681247
18                 5                 5  4788.9  9.682014
8                  3                 5  4859.7  9.689050
13                 4                 5  4824.6  9.693522
```

example of results

-cycle → True False

-stochastic cycle → True False

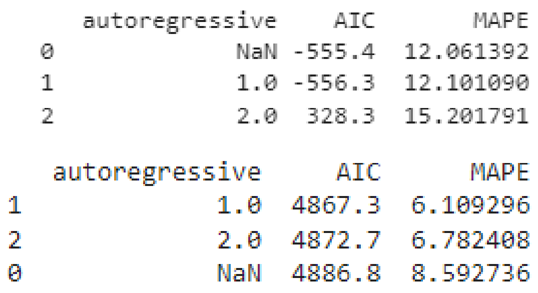
I then perform the same type of search for the Grid search parameters also for the cycle and autoregressive components, obtaining the following results:



```
   stochastic  damped    AIC       MAPE
0        True    True -555.4  12.061392
3       False   False -496.8  13.041025
1        True   False -494.8  13.041284
2       False    True -493.6  13.257878
```

```
   stochastic  damped     AIC       MAPE
3       False   False  4886.8   8.592736
0        True    True  4887.3   9.187906
2       False    True  4885.4   9.188870
1        True   False  4881.5  10.083796
```

Cycle grid search



```
   autoregressive    AIC       MAPE
0             NaN -555.4  12.061392
1             1.0 -556.3  12.101090
2             2.0  328.3  15.201791
```

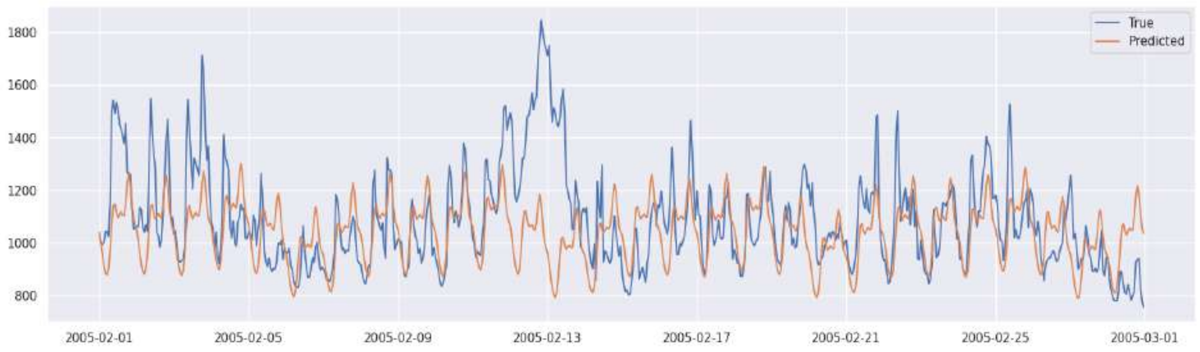
```
   autoregressive     AIC      MAPE
1             1.0  4867.3  6.109296
2             2.0  4872.7  6.782408
0             NaN  4886.8  8.592736
```

Autoregressive grid search

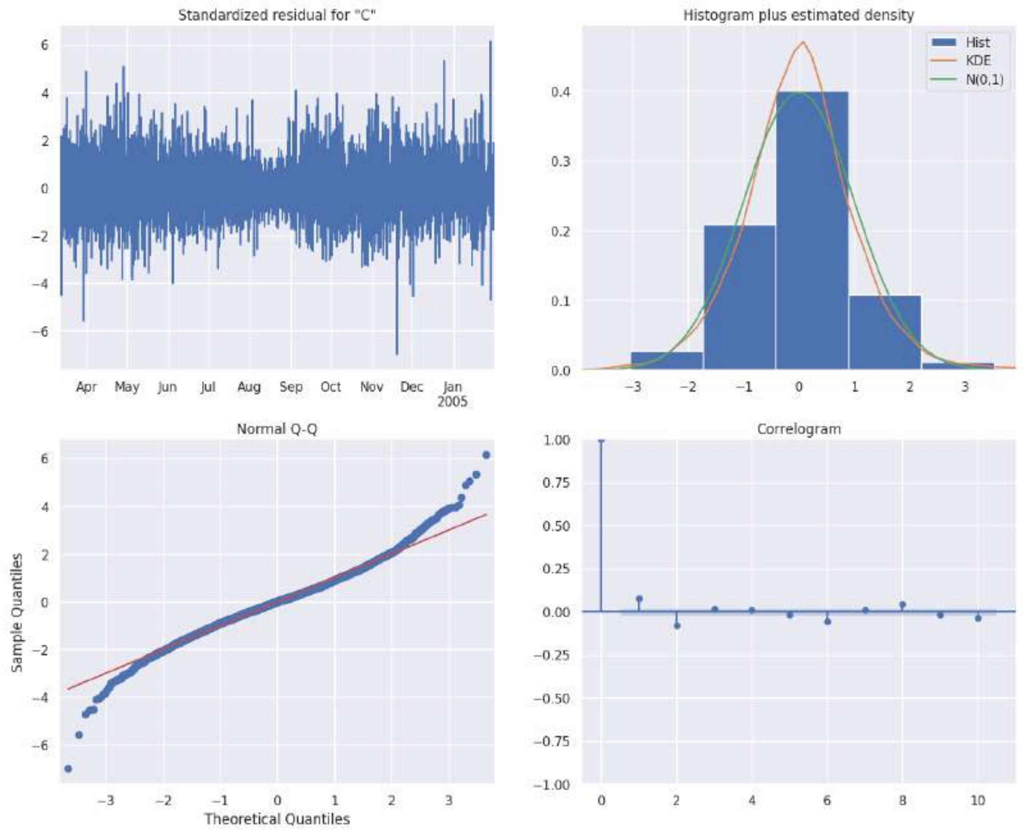
So now in this case we can see how the creation of the model occurs exclusively on the basis of the parameters that were searched automatically first

```
mod1 = sm.tsa.UnobservedComponents(train_def,
                                   level = level,
                                   autoregressive = ar,
                                   cycle = True,
                                   stochastic_cycle = stochastc,
                                   damped_cycle = damped,
                                   freq_seasonal = [{'period': s,
'harmonics': num1}, {'period': s1, 'harmonics': num2}])
```

Model creation

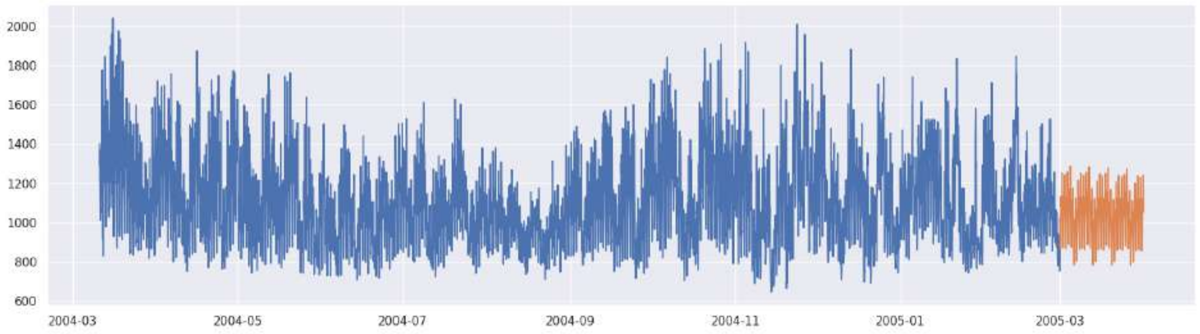


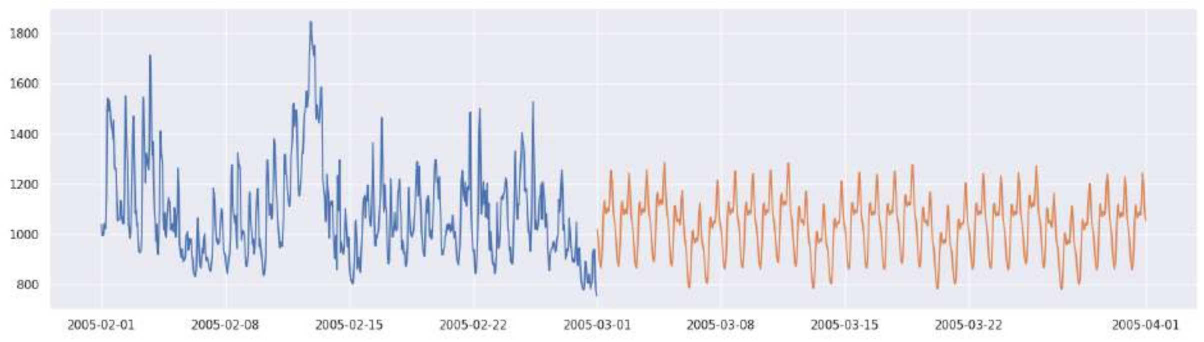
UCM test set prediction with best parameters



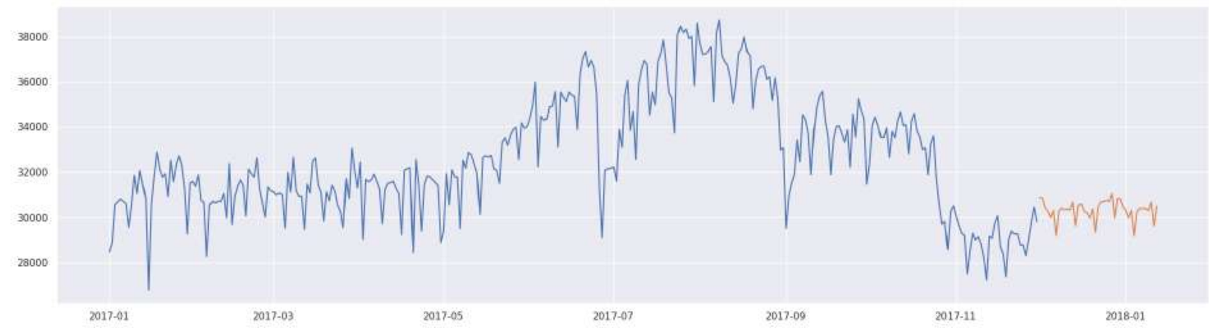
Example of model diagnostics

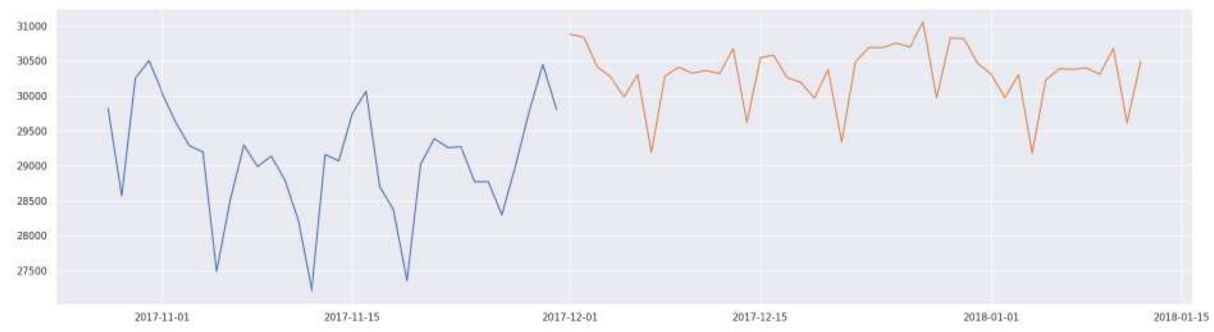
UCM Predictions



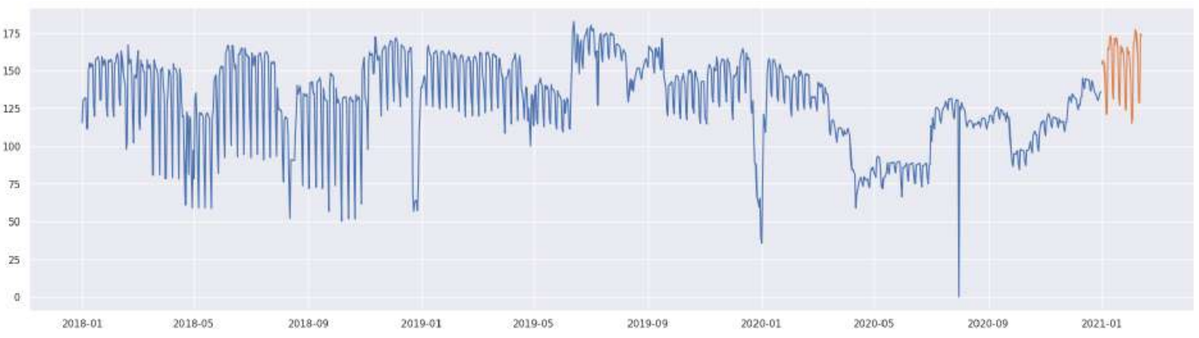


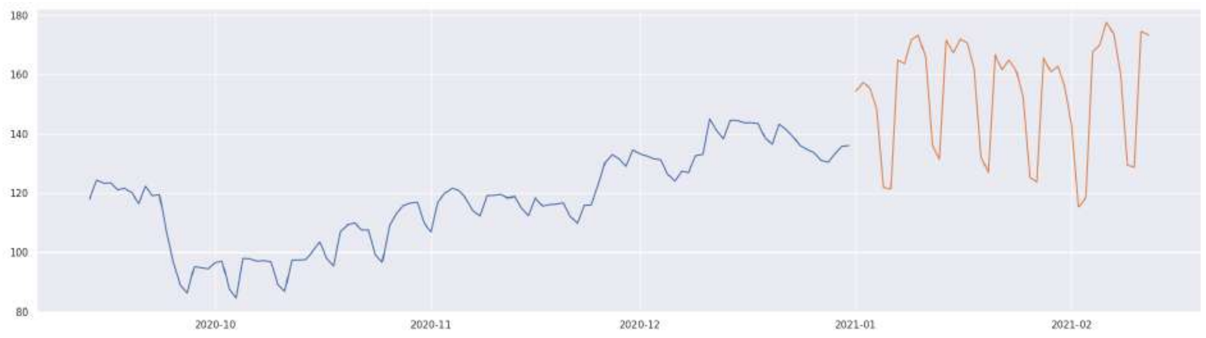
Ucm future forecasting (hour dataset)





Ucm future forecasting (daily dataset)





Daily dataset energy consumption predictions

7. Machine Learning

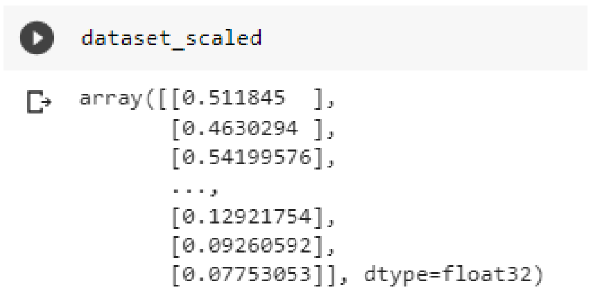
Machine learning is a powerful subset of artificial intelligence that enables computers to learn from data and make predictions or decisions without explicit programming. It can analyze vast amounts of information, detect patterns, and adapt to changing data.There are several types of models that can be used for time-series forecasting. Machine learning forecasting proved to be the most effective in capturing the patterns in the sequence of both structured and unstructured data
In the realm of time series forecasting, the choice of machine learning models can significantly impact the accuracy and effectiveness of predictions. Among the arsenal of tools available, i decided to focus more on these three models that stand out: Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and K-Nearest Neighbors (KNN). In this discussion, we will delve into the specifics of each of these models.

LSTM

LSTM (Long Short-Term Memory) is a variant of recurrent neural networks (RNN) designed to handle the problem of short-term memory.

Unlike traditional RNNs, LSTM is designed to handle long sequences of data while mitigating the vanishing gradient problem, a common issue in training deep neural networks. LSTM networks incorporate memory cells with gates that regulate the flow of information, allowing them to capture and store long-term dependencies in sequential data. This capability makes LSTM particularly well-suited for tasks involving time series forecasting, natural language processing, and various other sequential data analysis tasks.

First I normalized the data on a scale between 0 and 1 and divided it as previously into a train and test set. Time series data is prepared for model training, divided into windows of sequential observations

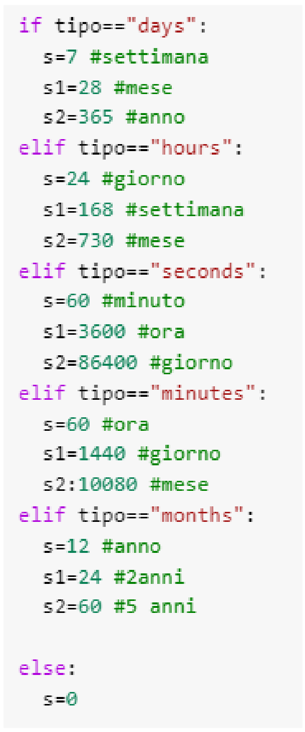


```
dataset_scaled
```

```
array([[0.511845  ],
       [0.4630294 ],
       [0.54199576],
       ...,
       [0.12921754],
       [0.09260592],
       [0.07753053]], dtype=float32)
```

example of new scaled dataset

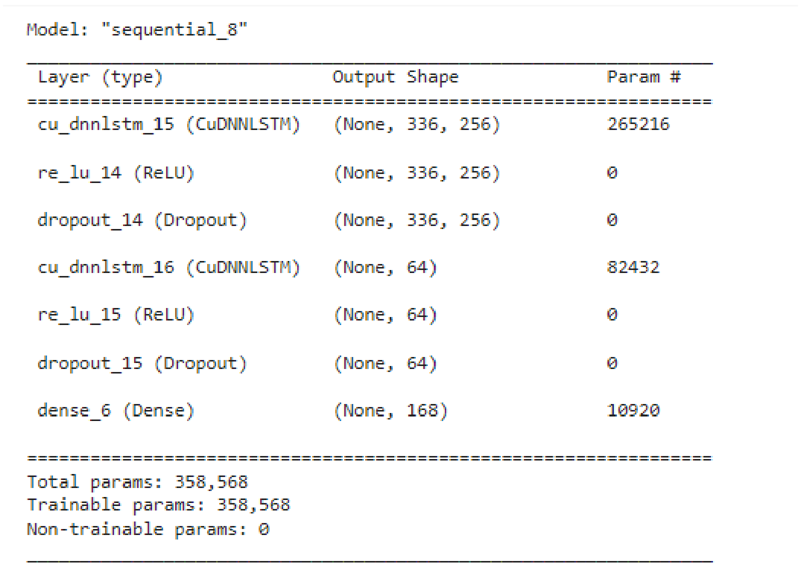
As regards the initial parameters to be set such as "look_back" and "output_size", several tests were carried out by testing with different datasets adopting different scales and in order to try to make the algorithm as automated as possible it was decided to use a combo of operations based on the seasonality parameters that were initially defined on the basis of the type of scale and aggregation of the dataset.



```
if tipo=="days":
  s=7 #settimana
  s1=28 #mese
  s2=365 #anno
elif tipo=="hours":
  s=24 #giorno
  s1=168 #settimana
  s2=730 #mese
elif tipo=="seconds":
  s=60 #minuto
  s1=3600 #ora
  s2=86400 #giorno
elif tipo=="minutes":
  s=60 #ora
  s1=1440 #giorno
  s2:10080 #mese
elif tipo=="months":
  s=12 #anno
  s1=24 #2anni
  s2=60 #5 anni

else:
  s=0
```

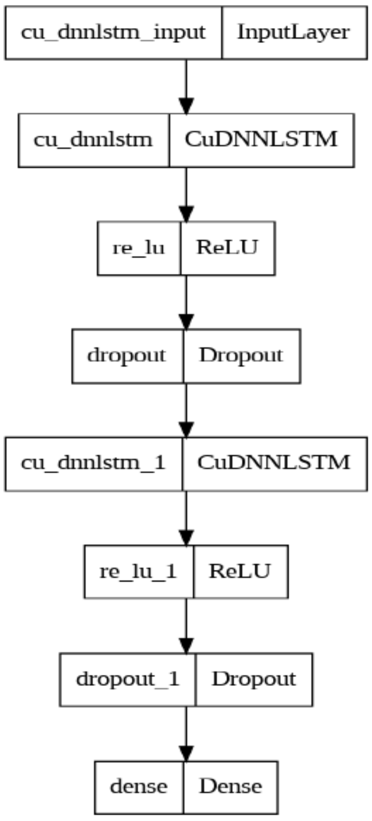
Seasonality features

As regards the structure of the neural network, several experiments were carried out by testing on different datasets with different transformations and different sizes and it was decided to use the following, constituted in this way:



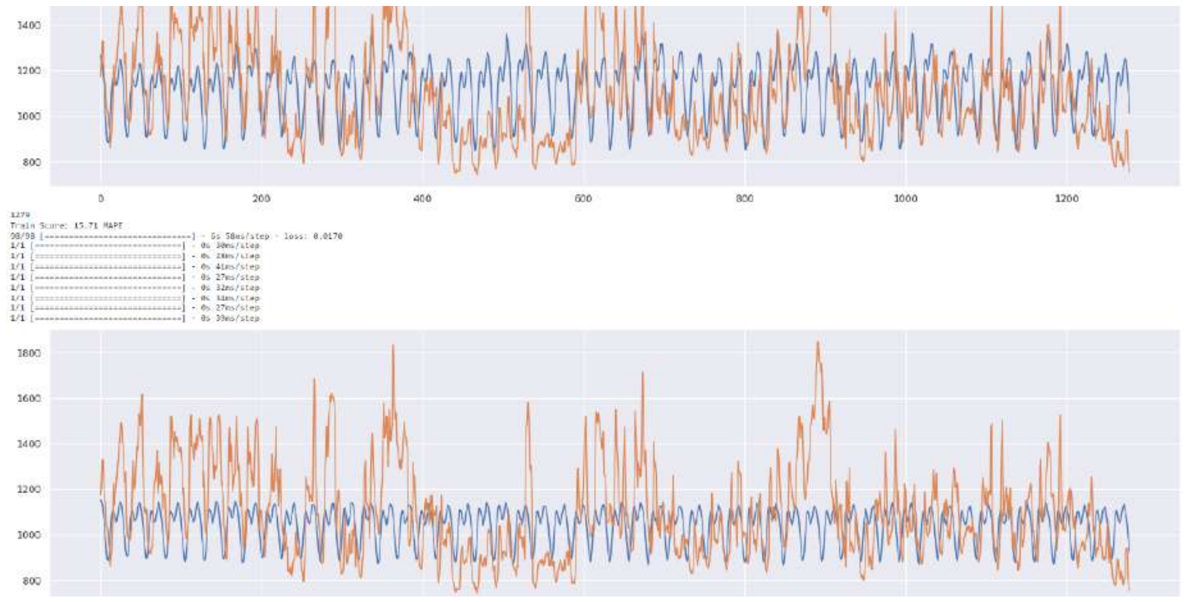
```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 cu_dnnlstm_15 (CuDNNLSTM)   (None, 336, 256)          265216

 re_lu_14 (ReLU)             (None, 336, 256)          0

 dropout_14 (Dropout)        (None, 336, 256)          0

 cu_dnnlstm_16 (CuDNNLSTM)   (None, 64)                82432

 re_lu_15 (ReLU)             (None, 64)                0

 dropout_15 (Dropout)        (None, 64)                0

 dense_6 (Dense)             (None, 168)               10920

=================================================================
Total params: 358,568
Trainable params: 358,568
Non-trainable params: 0
_________________________________________________________________
```

Network structure



Layer structure

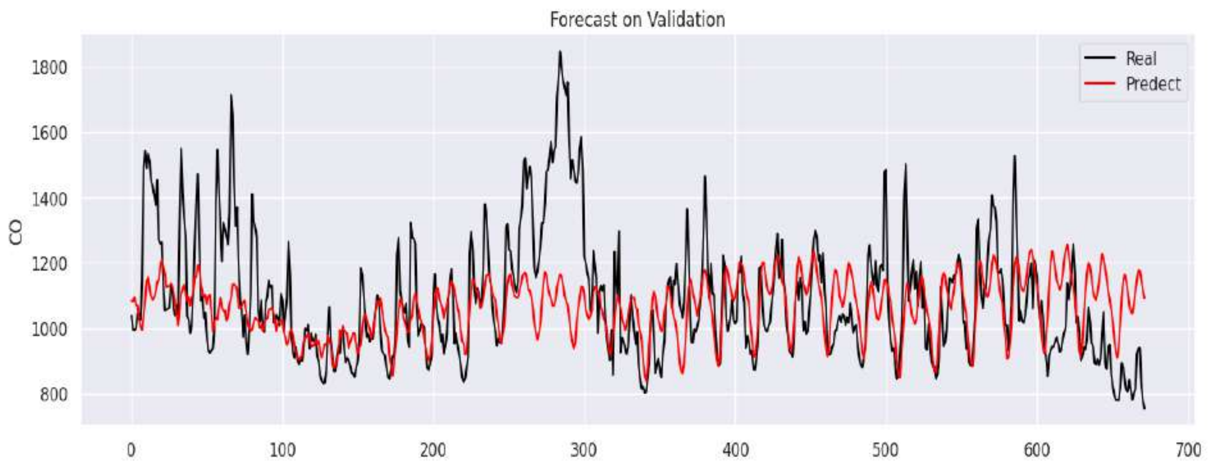
I train for 30 periods, testing with different net structures.
The error between the model predictions and the actual values is calculated, and this error is used to update the weights and biases of the LSTM units through error backpropagation.



```
1279
Train Score: 15.71 MAPE
98/98 [==============================] - 6s 58ms/step - loss: 0.0170
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 39ms/step
```

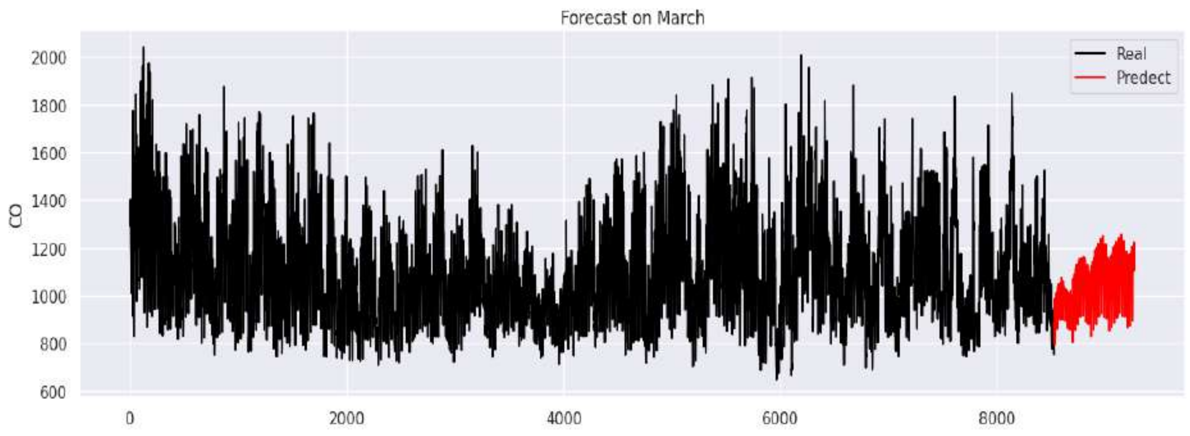
Example of model training

Obtaining a better model and automatically saving it to the drive as a numpy array.

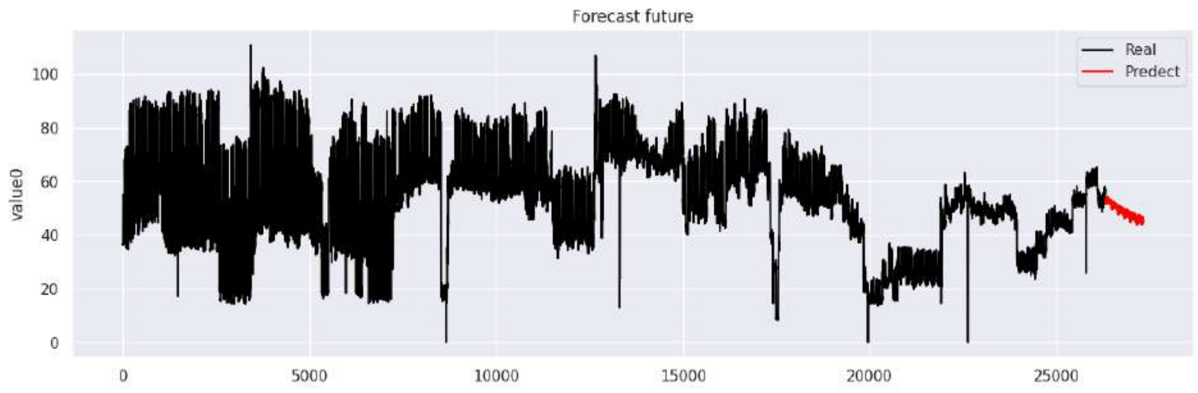


Example of test set forecast

LSTM Predictions



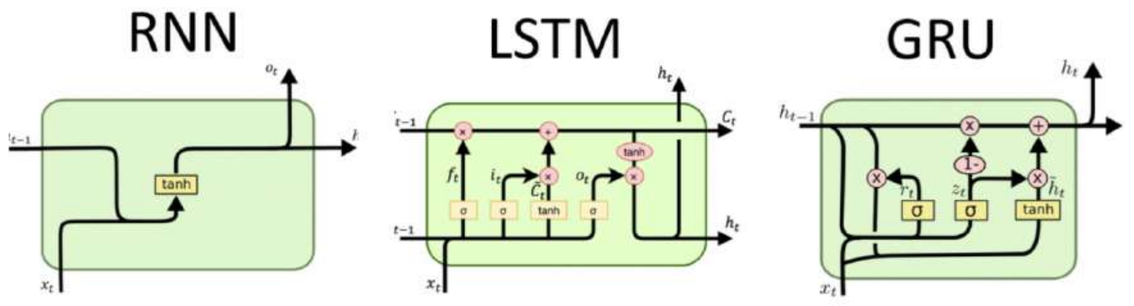
lstm prediction of a test dataset (hours)



Lstm hourly predictions of energy consumptions

GRU

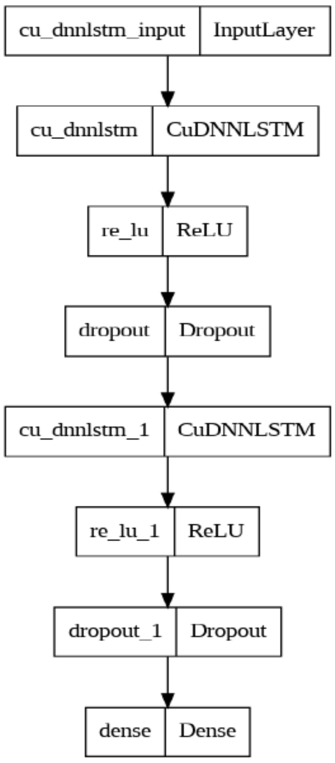
The operation of the GRU model for time series forecasting is similar to that of the LSTM model, but with some differences in the internal components.



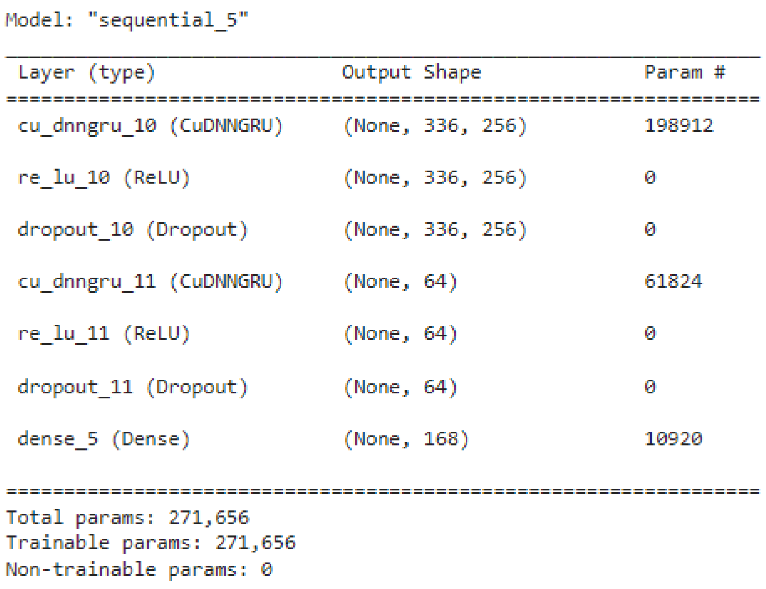
Lstm and gru structures

Gated Recurrent Units (GRUs) are a cutting-edge neural network architecture tailored for time series forecasting. They stand out for their ability to efficiently capture patterns and dependencies in sequential data. Unlike traditional recurrent networks, GRUs excel at handling long-range relationships and avoiding training complications.

GRUs are known for their quicker training convergence due to the efficient flow of gradients, making them suitable for both short and long time series.
GRUs simplify the complexities of time series prediction. Their selective information retention, efficient training, and adaptability across domains make them an indispensable tool for accurate and efficient forecasting.
In summary, GRUs use the reset gate to decide what to forget from the previous hidden state and the update gate to decide how much of the new candidate state to incorporate. This allows GRUs to capture relevant information from the past while adapting to new input, making them efficient for modelling sequential data, including time series.



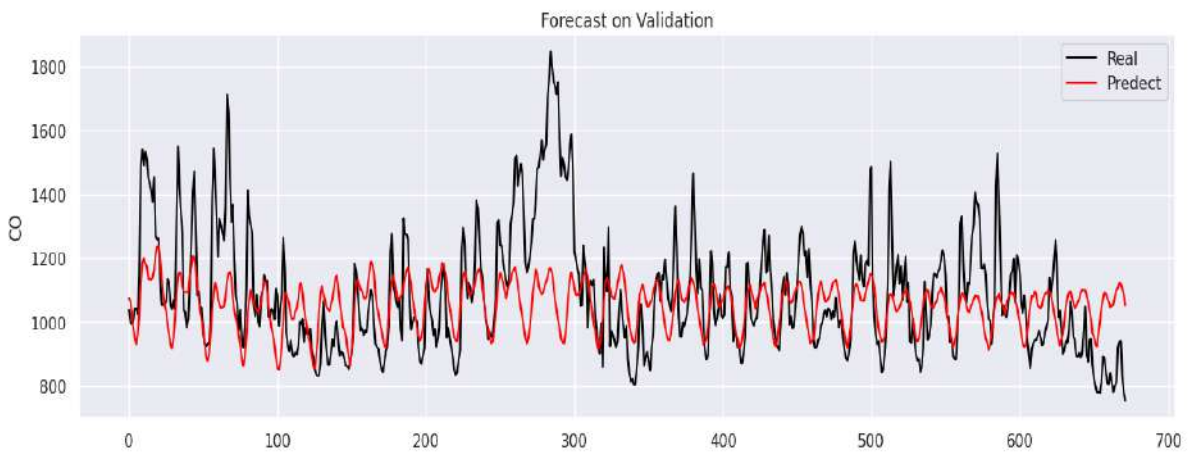
Layer Structure



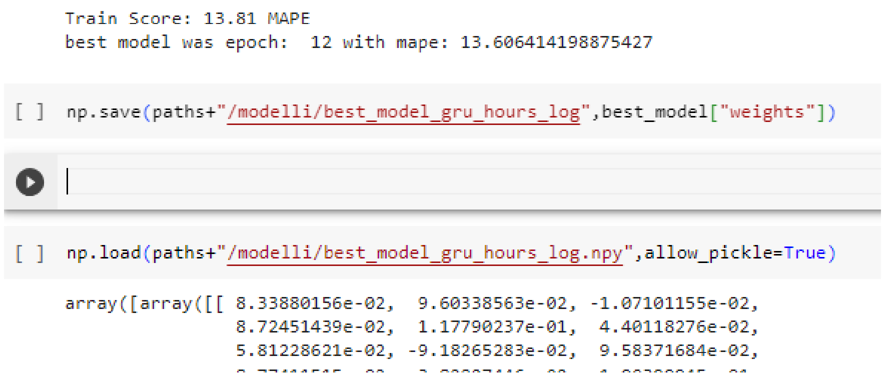
```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 cu_dnngru_10 (CuDNNGRU)     (None, 336, 256)          198912

 re_lu_10 (ReLU)             (None, 336, 256)          0

 dropout_10 (Dropout)        (None, 336, 256)          0

 cu_dnngru_11 (CuDNNGRU)     (None, 64)                61824

 re_lu_11 (ReLU)             (None, 64)                0

 dropout_11 (Dropout)        (None, 64)                0

 dense_5 (Dense)             (None, 168)               10920

=================================================================
Total params: 271,656
Trainable params: 271,656
Non-trainable params: 0
```

Network Structure

Train on 30 epochs and save the best models:



Forecast on validation



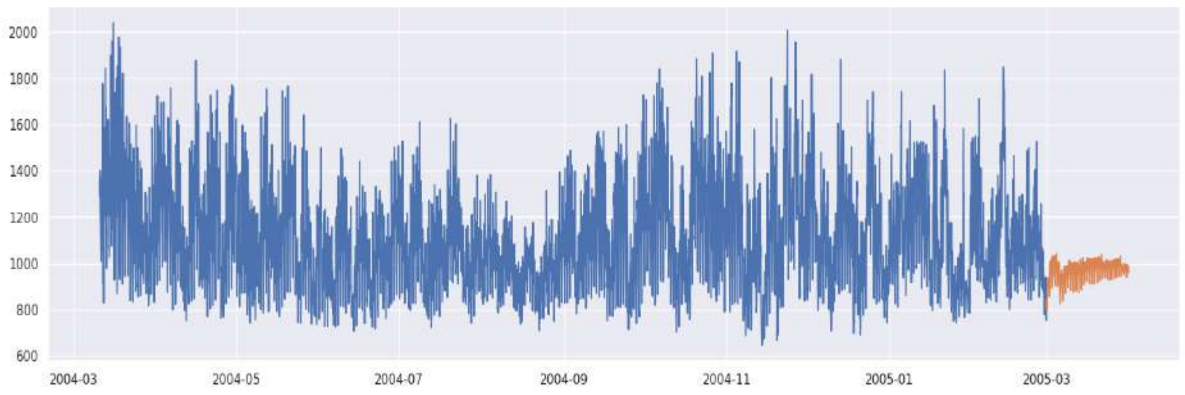
```
Train Score: 13.81 MAPE
best model was epoch:  12 with mape: 13.606414198875427
```

```
[ ]  np.save(paths+"/modelli/best_model_gru_hours_log",best_model["weights"])
```

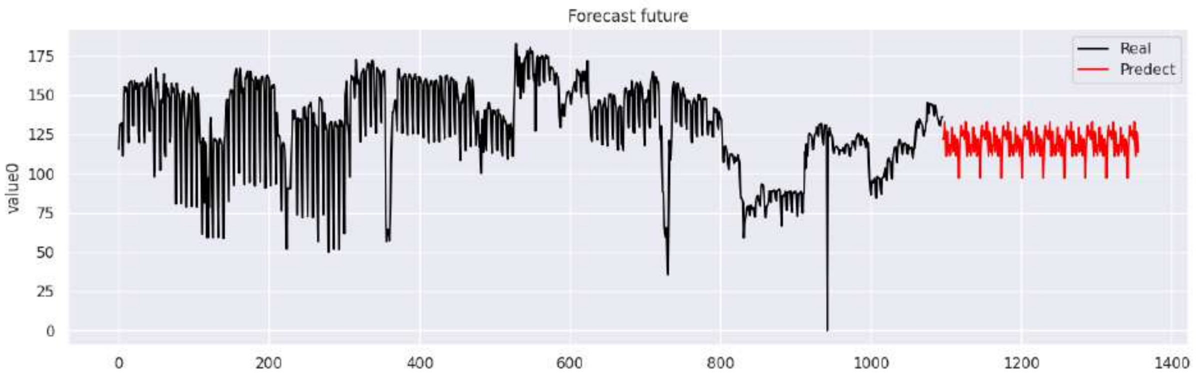
```
[ ]  np.load(paths+"/modelli/best_model_gru_hours_log.npy",allow_pickle=True)

     array([array([[ 8.33880156e-02,  9.60338563e-02, -1.07101155e-02,
                     8.72451439e-02,  1.17790237e-01,  4.40118276e-02,
                     5.81228621e-02, -9.18265283e-02,  9.58371684e-02,
```

Save models

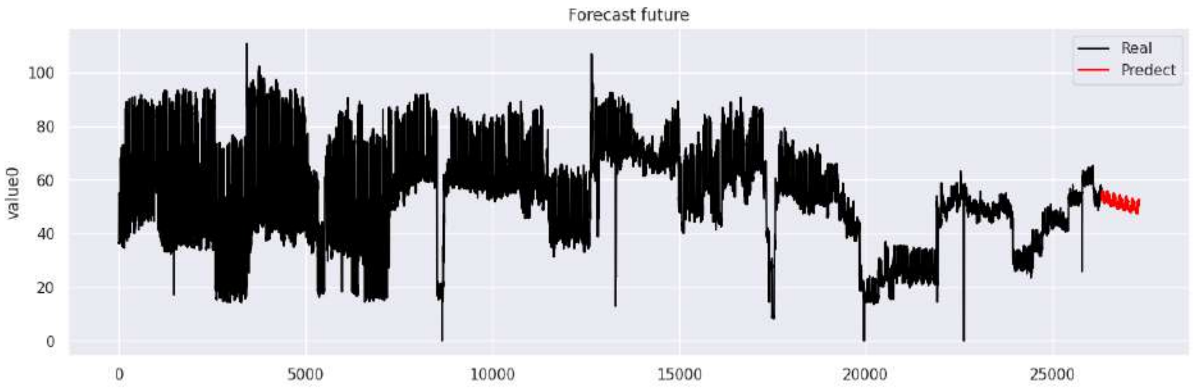
GRU Predictions



Hourly predictions



Daily predictions



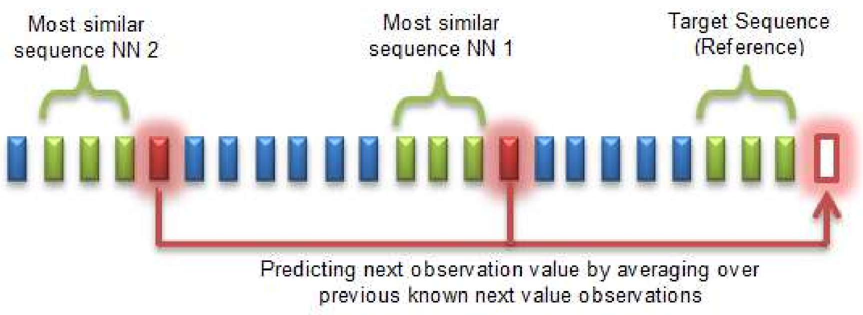
Hourly predictions

KNN

K-Nearest Neighbors (KNN) is a simple yet effective machine learning algorithm used for time series forecasting. Unlike traditional time series models, KNN operates on the principle of similarity. It predicts future values by identifying the most similar historical patterns in the dataset.

Time series data is typically represented as a sequence of observations over time, each data point consists of a timestamp and a corresponding value. The "K" in KNN represents the number of nearest neighbours to consider when making predictions and selecting an appropriate value for K is crucial and involves experimentation. The algorithm identifies the K nearest neighbours to the current data point based on this similarity metric.

Once the K nearest neighbours are found, their values are combined to make a prediction, a common approach is to use weighted averaging, where closer neighbours have a higher influence on the prediction and the predicted value for the next time step is calculated based on the weighted average of the K nearest neighbours.


Most similar sequence NN 2
Most similar sequence NN 1
Target Sequence (Reference)
Predicting next observation value by averaging over previous known next value observations

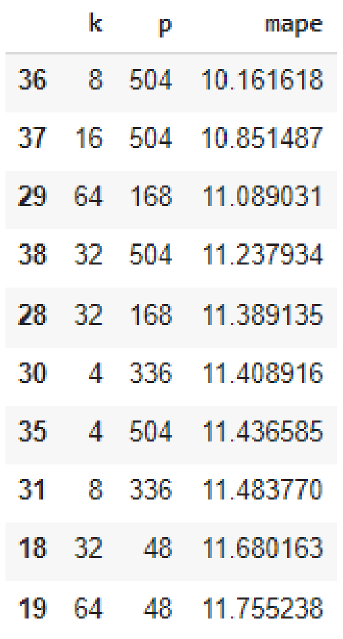
KNN

The process is repeated iteratively for each time step in the forecast horizon.

As new observations become available, older ones are incorporated into the historical dataset.

By identifying similar historical patterns, it provides a different approach to prediction compared to traditional time series models. Careful selection of parameters and an understanding of the data's underlying characteristics are essential for successful application.

The data is divided into windows of sequential observations, and each window is associated with the corresponding value to be predicted.
I try to find the best k and p parameters for the KNN model by doing a search:
k (number of observations passed by consider) and p (how many similar subsequences to look at).

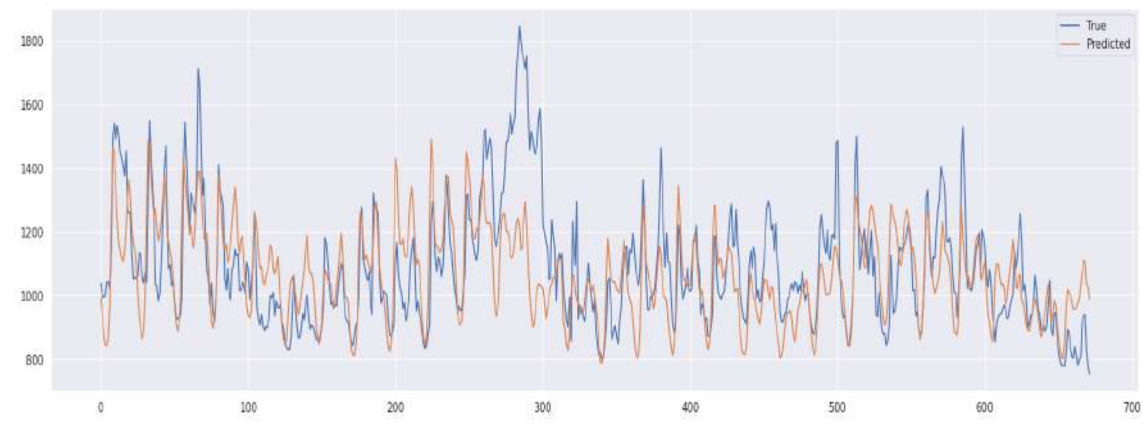


	k	p	mape
36	8	504	10.161618
37	16	504	10.851487
29	64	168	11.089031
38	32	504	11.237934
28	32	168	11.389135
30	4	336	11.408916
35	4	504	11.436585
31	8	336	11.483770
18	32	48	11.680163
19	64	48	11.755238

grid search k and p parameters

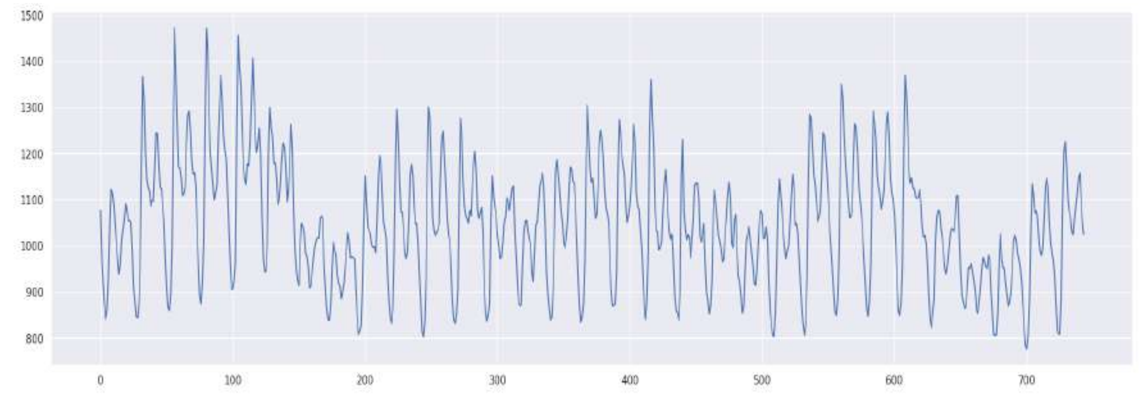
```
prediction = knn_forecasting(train, len(test), k, p)
plot_prediction(list(test["CO"]), prediction)
```

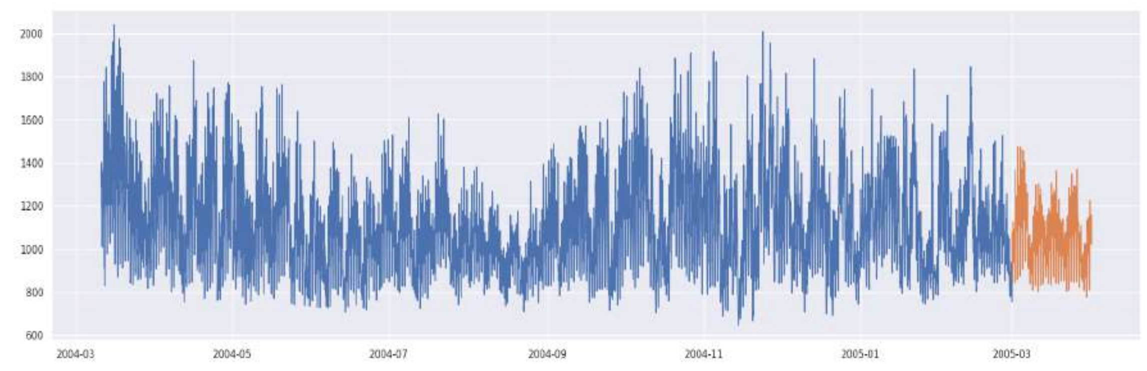
test on validation set:

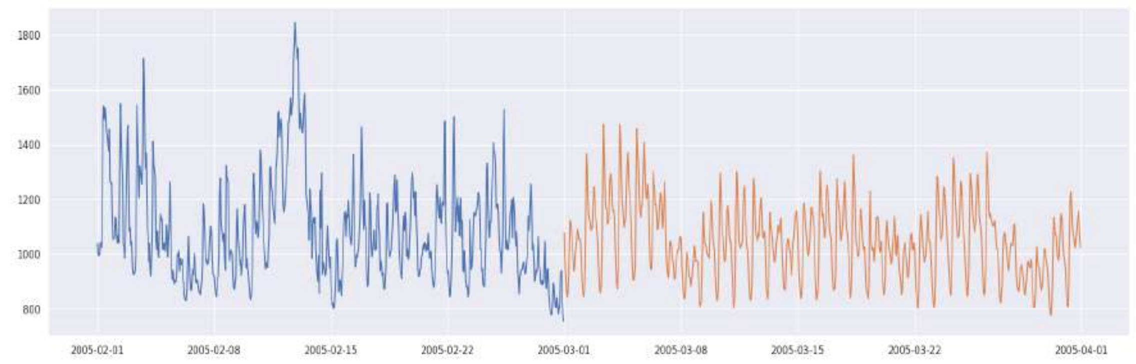


Forecast on validation

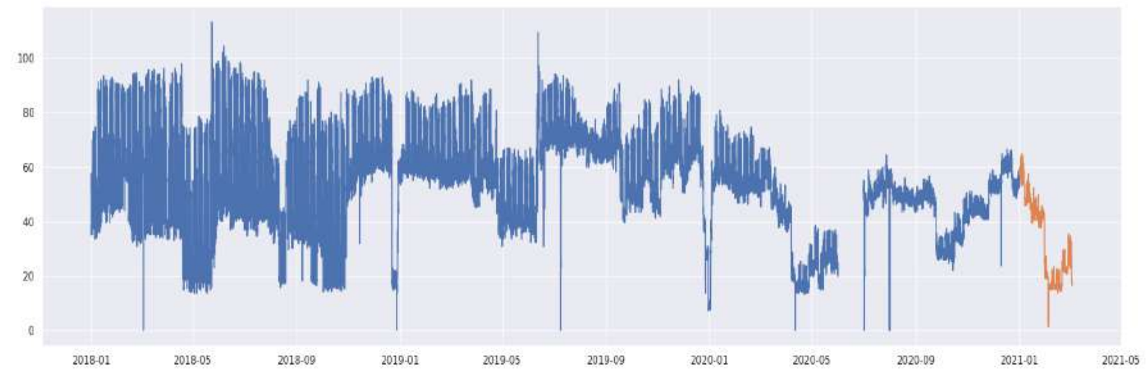
KNN Predictions

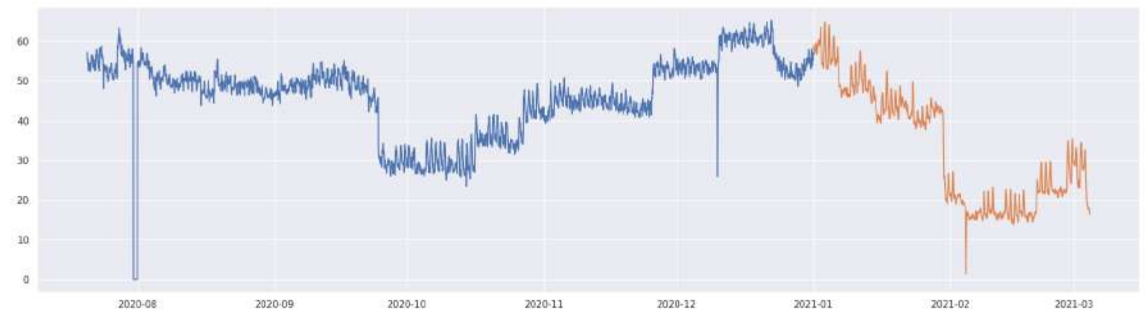




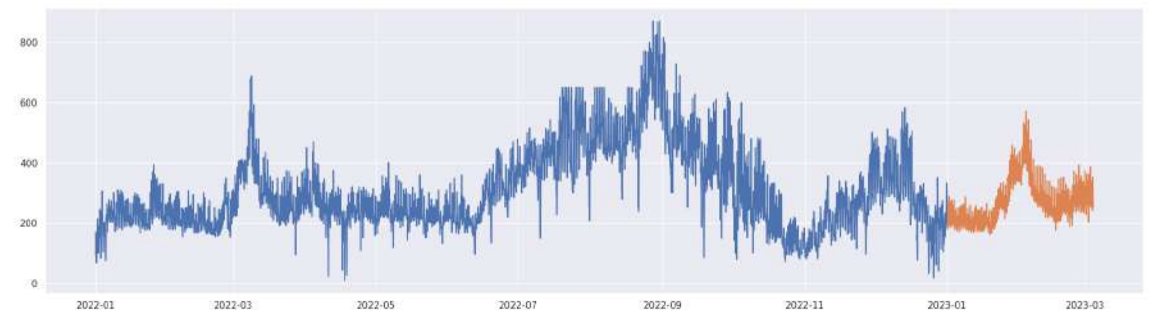


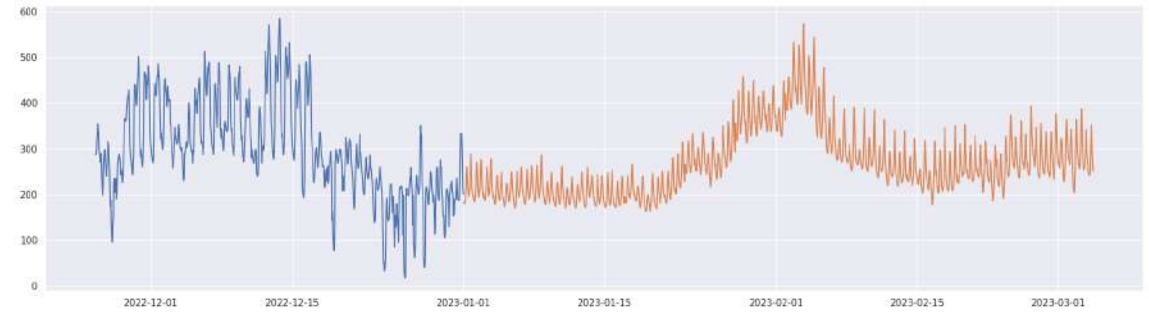
Example of prediction based on best k and p





Hourly energy consumption forecast





Daily energy consumption forecast

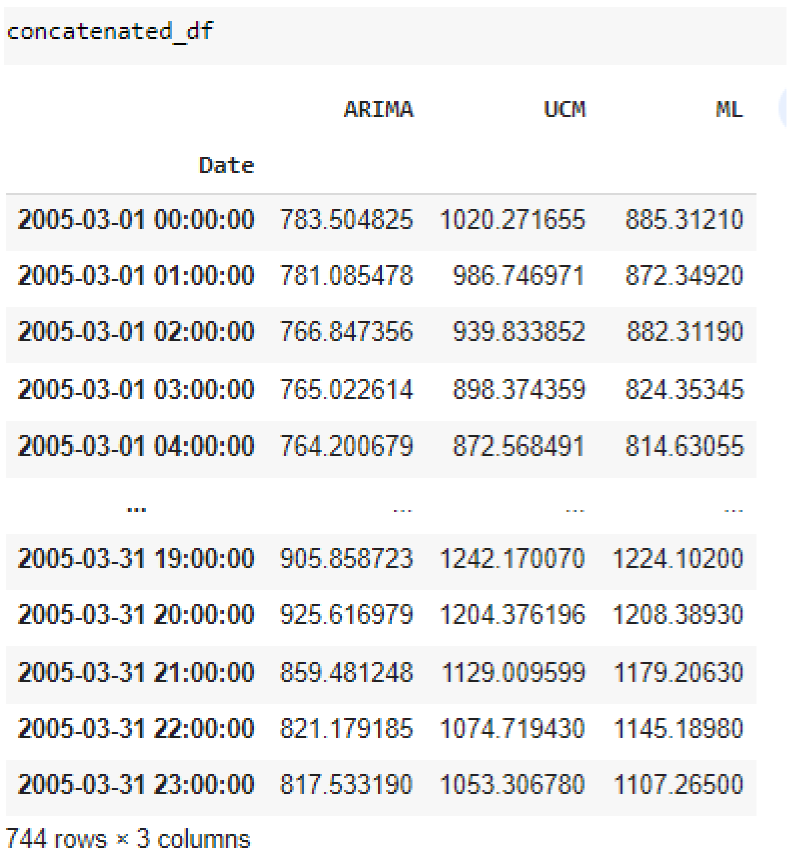
8. Final Predictions

As regards the final phase on the comparison of the forecasts, various checks were carried out, first of all we tried to understand for each series which was the model that found the best positive evaluation in the test set at the mape and mae level, saving in a automatically selects the best model and the best predictions by concatenating and creating a dataset containing all the predictions obtained for the different algorithms used with graphical comparisons, automatically saved in the work folder.

best_ARIMA	11.01
best_UCM	10.88
LSTM	10.34
GRU	11.25
KNN	16.76

Example of MAPE results

Although the ARIMA and UCM models do not obtain excellent results, they are the most computationally expensive and least adaptable to different types of data.
Machine learning models obtain significantly better results on more numerous and more adaptable data. The model that is graphically most correct is the KNN model.

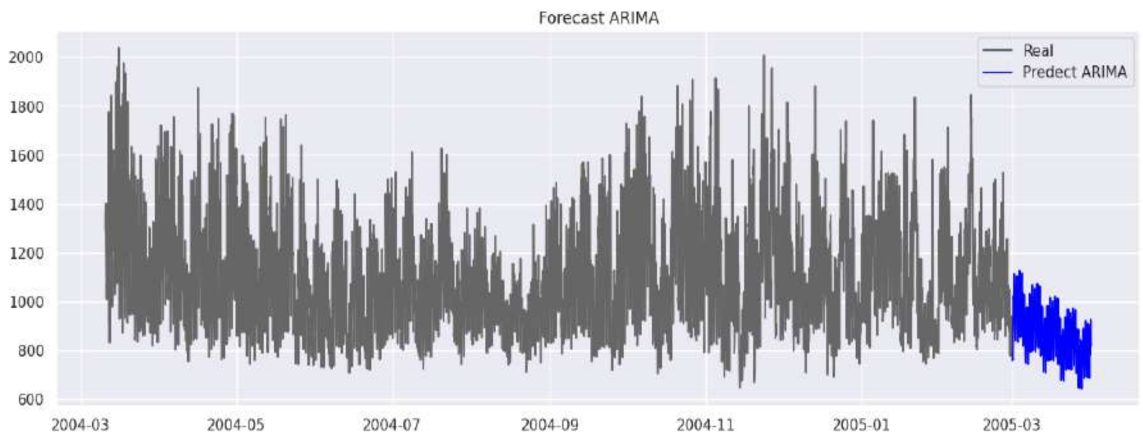


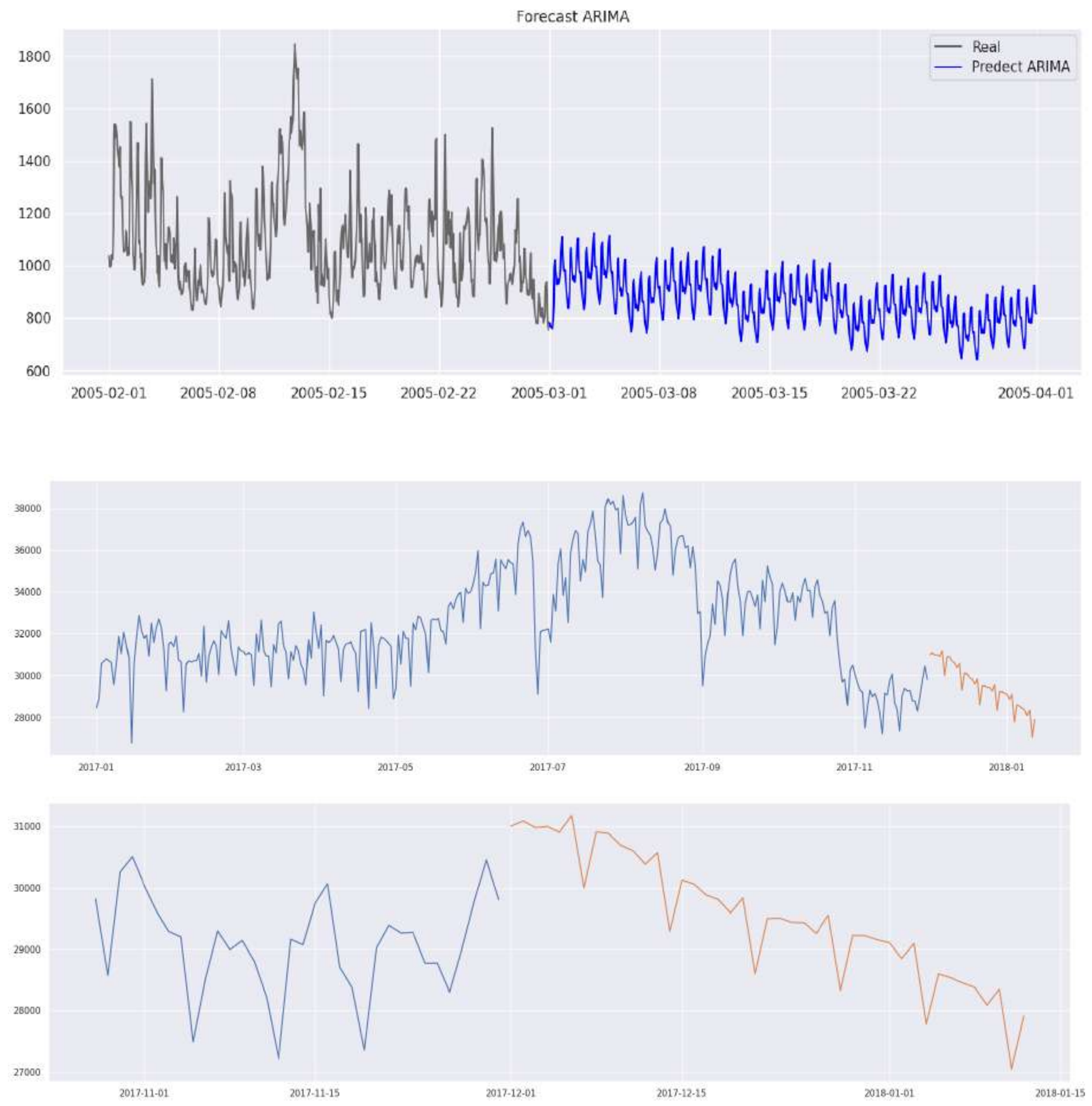
```
concatenated_df
```

Date	ARIMA	UCM	ML
2005-03-01 00:00:00	783.504825	1020.271655	885.31210
2005-03-01 01:00:00	781.085478	986.746971	872.34920
2005-03-01 02:00:00	766.847356	939.833852	882.31190
2005-03-01 03:00:00	765.022614	898.374359	824.35345
2005-03-01 04:00:00	764.200679	872.568491	814.63055
...	...	...	...
2005-03-31 19:00:00	905.858723	1242.170070	1224.10200
2005-03-31 20:00:00	925.616979	1204.376196	1208.38930
2005-03-31 21:00:00	859.481248	1129.009599	1179.20630
2005-03-31 22:00:00	821.179185	1074.719430	1145.18980
2005-03-31 23:00:00	817.533190	1053.306780	1107.26500

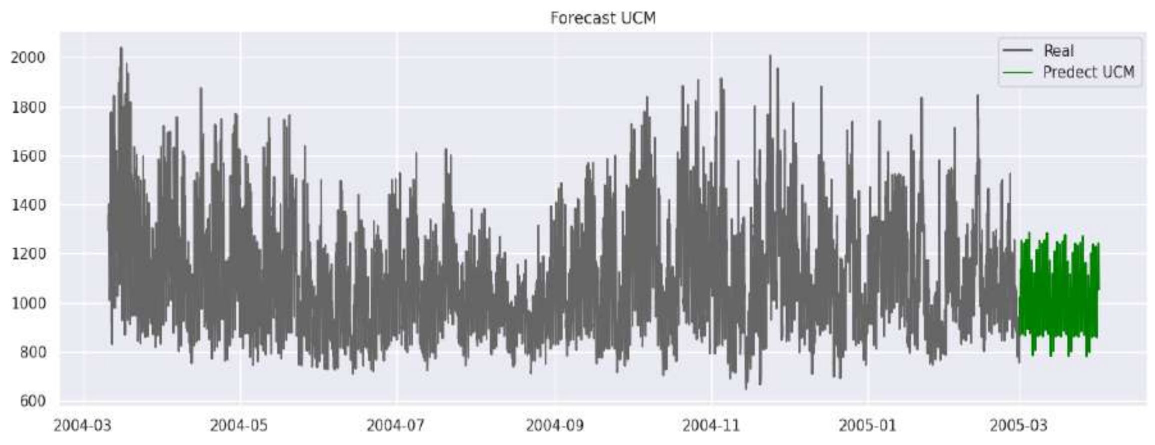
744 rows × 3 columns

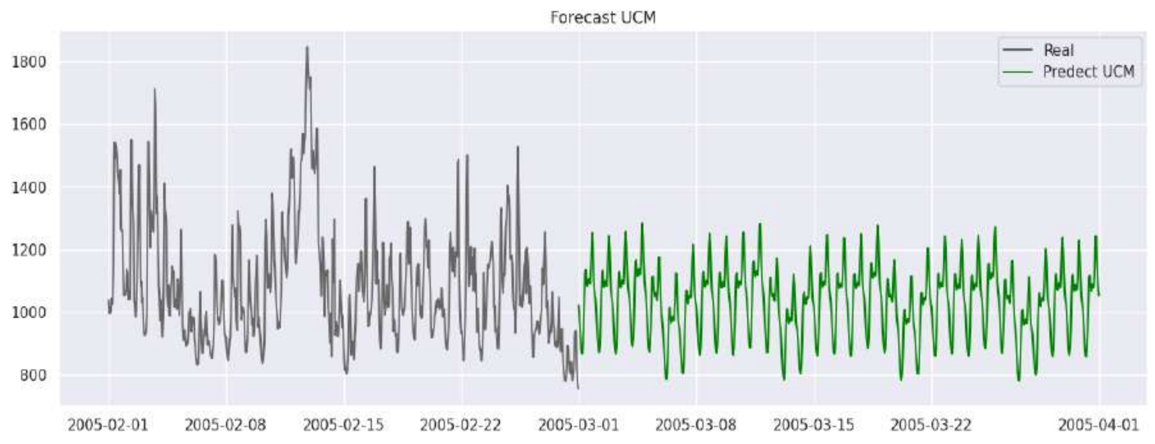
Example of automated saved forecasting df

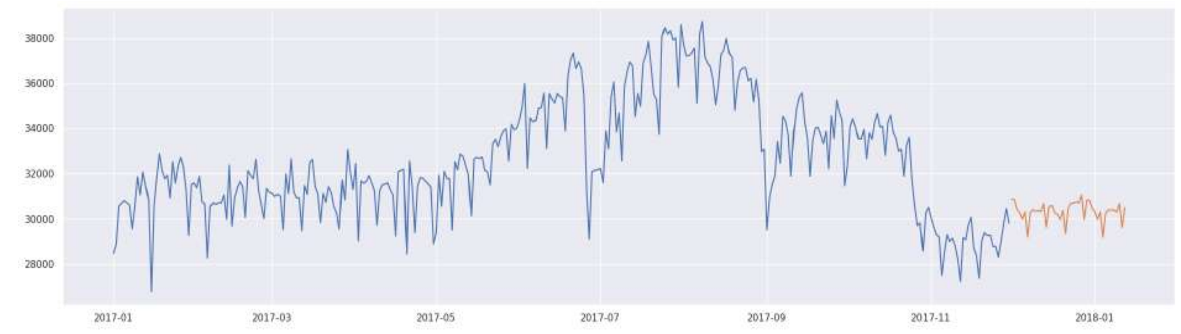


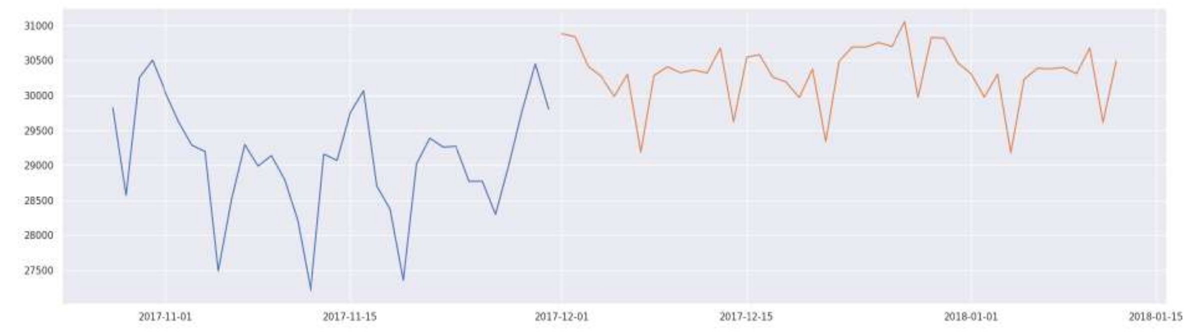


ARIMA forecasts

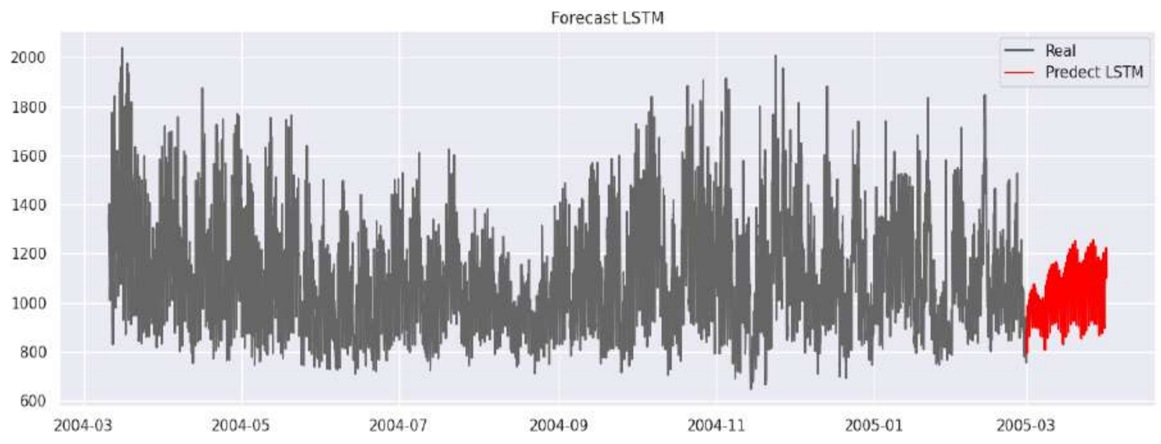


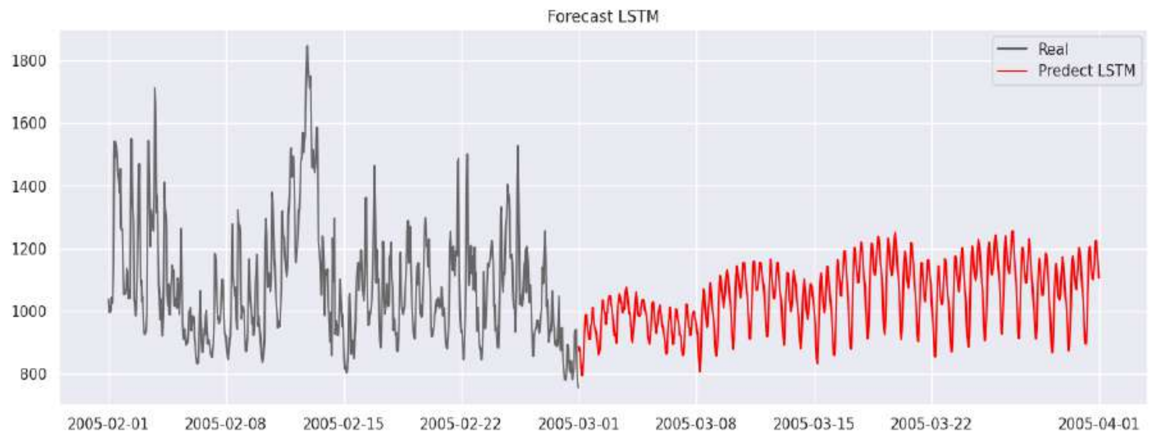


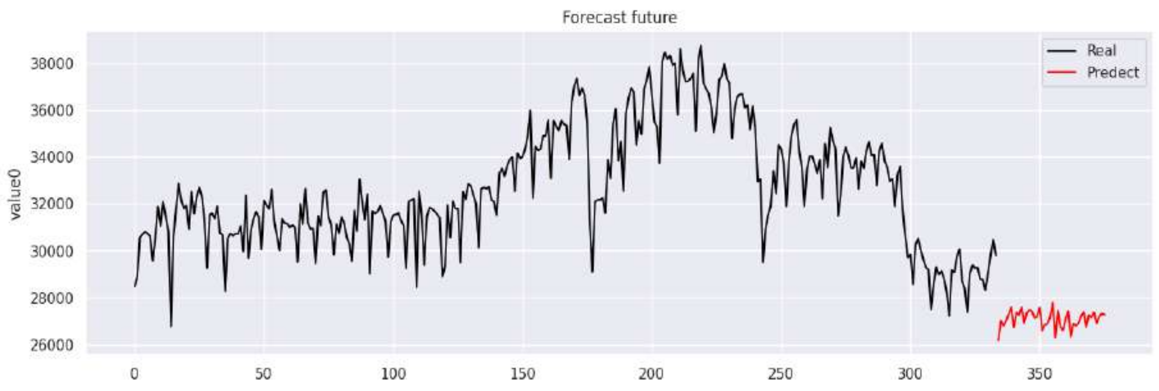




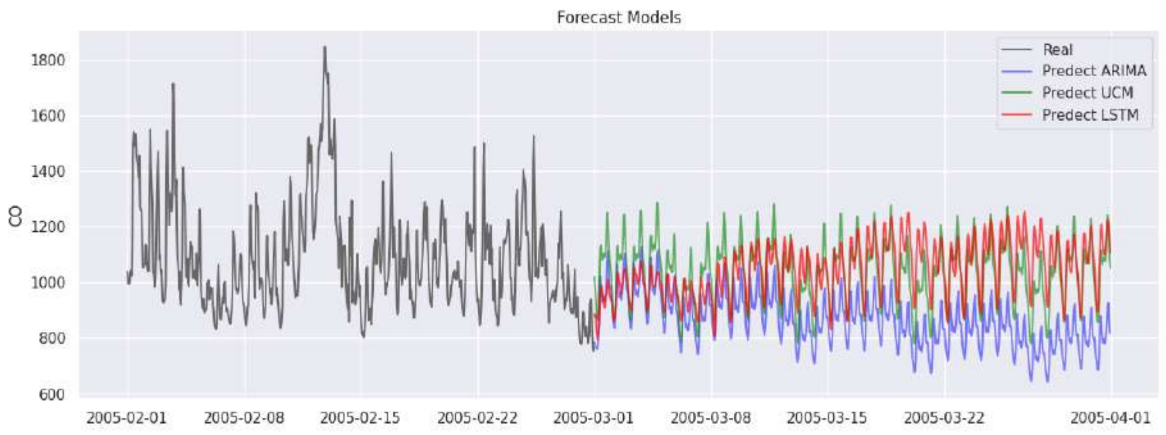
UCM forecasts





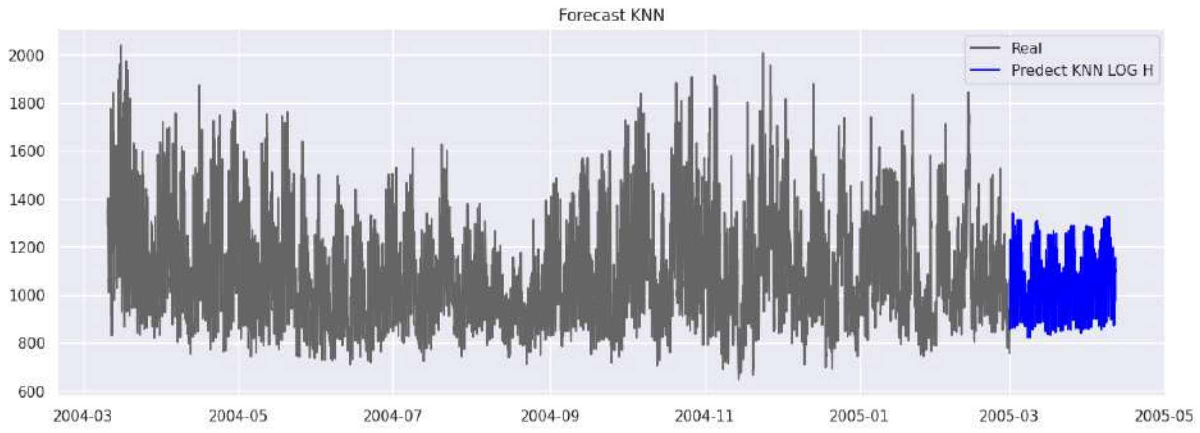


LSTM forecasts

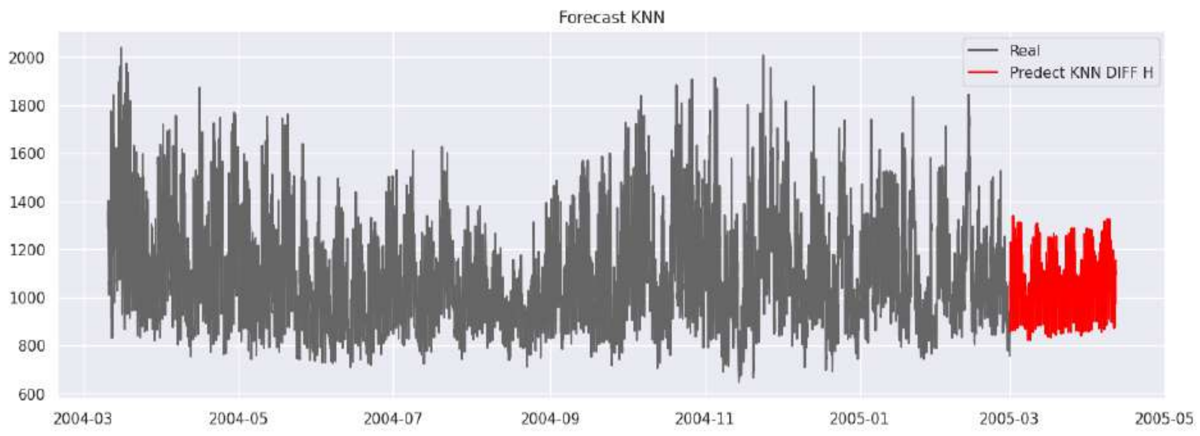


Forecasts comparison

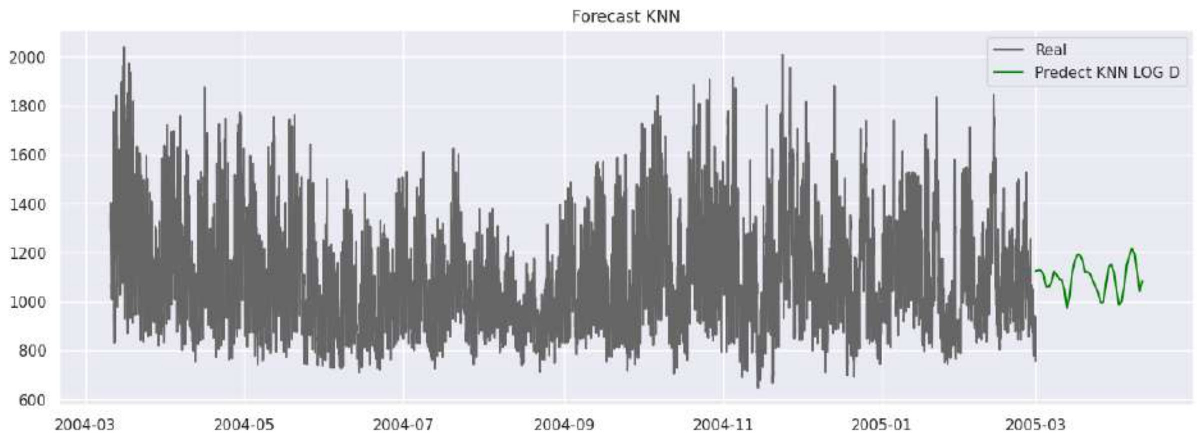
Secondly, the main challenge of this project is the comparison of multiscale results. Therefore, for the same dataframe, a prediction can be made on different scales with different statistical and machine learning algorithms automatically. Furthermore, since it is possible to make multiscale forecasts, it is also possible to compare them with each other by saving them automatically in the work folder. Therefore, using the aggregation method that we have seen previously, it is possible to compare a smaller forecast, for example every hour, with a larger forecast, for example daily, on the basis of the same algorithm used in such a way as to understand whether one is more reliable or not. prediction of one kind or another. By aggregating from one forecast to another to see what the differences may be and understand why.



KNN logarithmic transformation (hours)

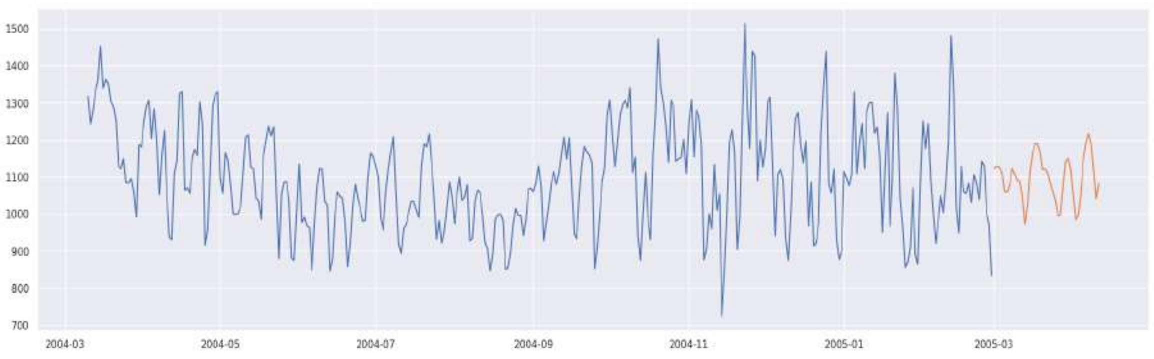


KNN seasonal diff transformation (hours)

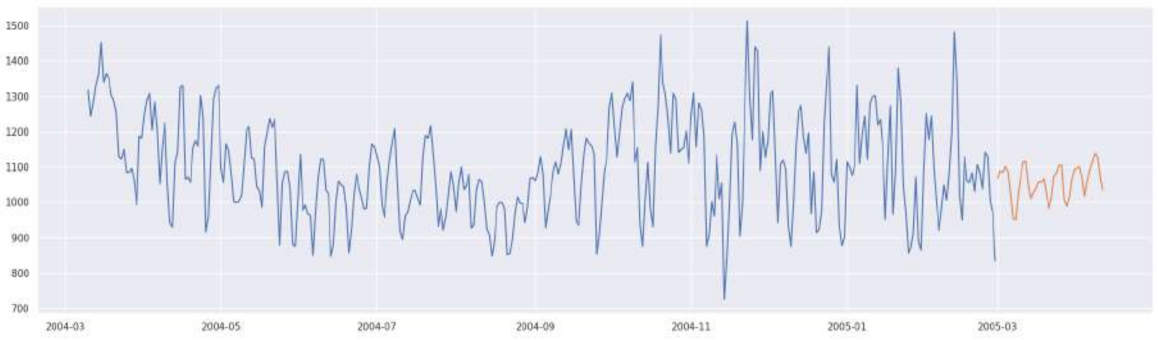


KNN logarithmic transformation (daily)

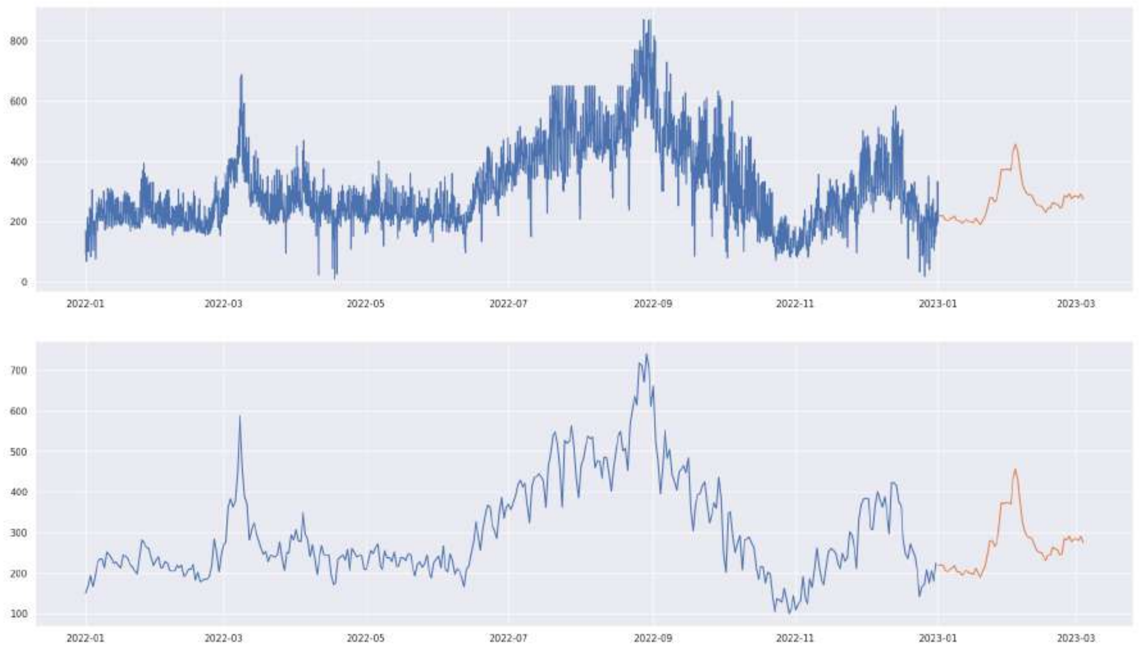
Now what I try to do is take the forecast obtained by using the test dataset scaled on hourly intervals and aggregate the forecast based on the same test algorithm such as the KNN in this case and evaluate and verify whether a forecast is obtained similar to that obtained using the same training dataset but with the values scaled to the daily level. Comparing the two predictions trying to understand which one is actually more reliable by making a comparison on how the model behaves on multiscale datasets.



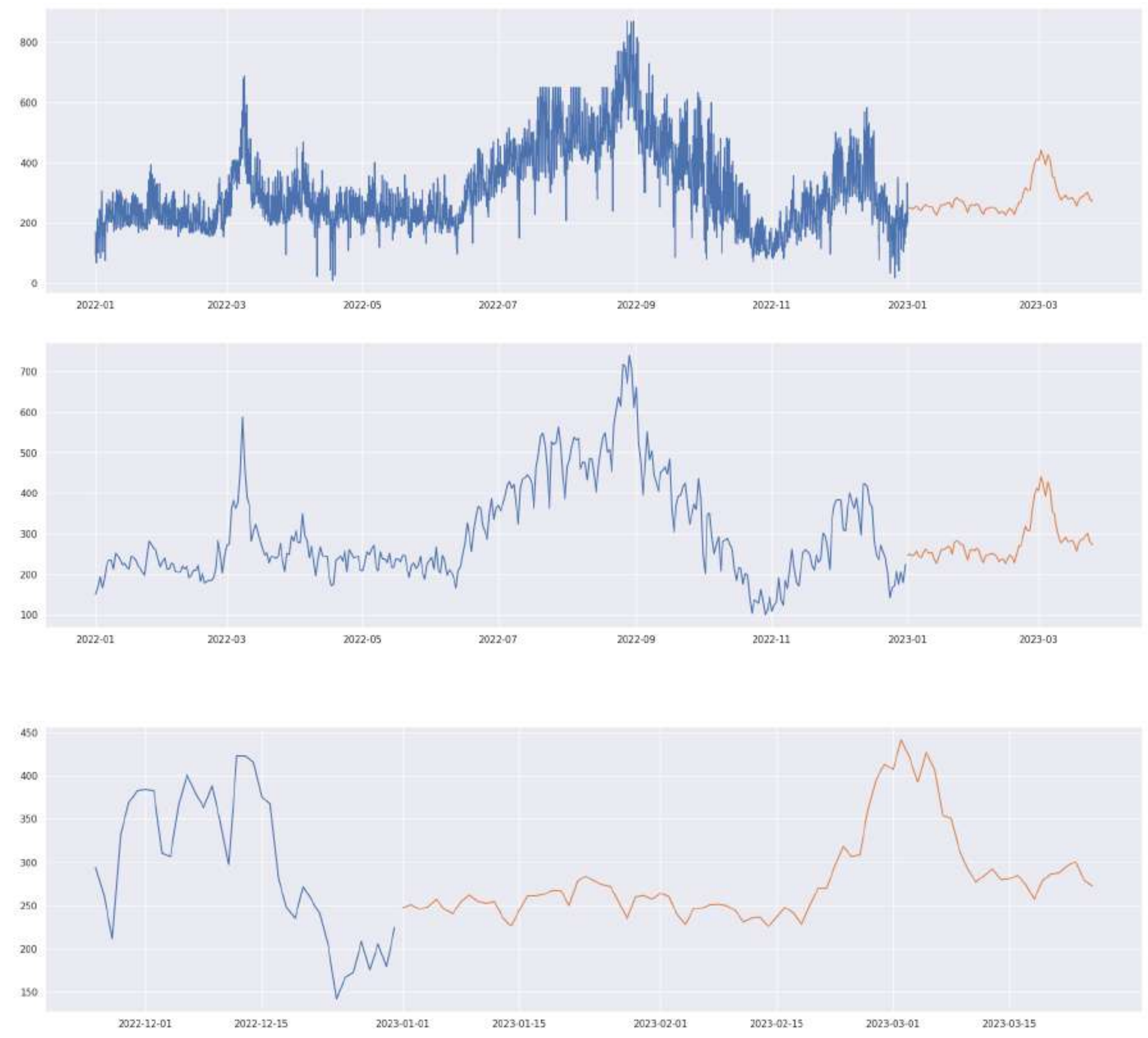
Original forecasting for daily time series



Aggregation forecasting for hours timeseries



Aggregation forecast from hours to day



Daily forecast

Therefore, few differences can be noted at the trend level by comparing the daily frequency forecast of the dataset.. Compared to the hourly forecast by aggregating the data by average.

9. Conclusions

The tests carried out certainly cannot be considered sufficient to fully understand how to best use the right balance of parameters to optimize the performance of the forecasting models.
However, it is possible to show the main solutions obtained and, as shown in the previous chapter, analyze the prediction results both at a statistical level and at the level of computational time required.
Starting from the first statistical model used, it can be said that the ARIMA model was the model that encountered the most problems with the testing phase in which different datasets of all types were used.
The auto.arima model of the statsmodels library, in addition to having a very high calculation time for numerous datasets, has never produced the results obtained, the choice to keep the differentiation value fixed at 1 and cycle the parameters p and q using grid search is turned out to be the choice that best suited different time series, but at the same time takes various problems if a series had different features, obtaining even very high MAE and MAPE values.
We can also talk about the same type of problem for UCM models, in which however slightly more specific models were obtained by trying to obtain the best results that best suited a time series. The problem in this case can be defined as sometimes the parameters obtained were not the best ones for optimization as they were considered individually on the type of time series and not as a combination of them. The

following choice was made primarily due to the computational costs required which would have been too high.
Machine learning models like the ones used require even larger time series to perform better.
Furthermore, the selection of parameters linked to the change in scale of the time series such as the transition from an hourly frequency to a daily frequency did not appear to be optimized correctly, we will talk about this in the next part regarding future developments and implementations in which we will mention the discussion of Automated Machine Learning.

Future developments

The aspects to be considered as future developments are different and could be useful for improving this project from several points of view. First of all, the models currently tested were only 2 as statistical models which are ARIMA and UCM and 3 as machine learning models which are KNN, GRU and LSTM.
Therefore, an important aspect for the development of this project is the possibility of using various new forecasting methods such as gradient boost which it was initially decided to use but then it was decided not to consider or other models in general both at a statistical and at the machine learning level.
Another aspect to consider for improvement is the computational time involved when dealing with very large datasets or time series. Since the parameter selection procedure is automated and also in grid search for some models such as the case of ARIMA in which the parameters to be set are different and difficult to understand automatically. A better method could be used for the optimization of the values and to reduce the computational time by not searching among all the values but only among those that could be most useful to do initially.
Regarding machine learning models, one of the things to consider is the selection of settings. Machine learning models in parameter selection rely heavily on the type and frequency of time series data to find parameters such as output size, learning rate or for

example window size. Therefore, one of the first improvements would certainly be the implementation of a better choice of settings based on the type of time scale and aggregation of the time series.
While the second important aspect would be the implementation of automated machine learning.
Automated machine learning is the process of automating the tasks of applying machine learning. Automation in AutoML aims to allow non-experts to make use of machine learning models and techniques without requiring them to become experts. Automating the process of applying machine learning end-to-end. Common techniques used in AutoML include hyperparameter optimization, meta-learning and neural architecture search.
Automated ML, is the process of automating repetitive and time-consuming tasks related to developing Machine Learning models. It enables data scientists, analysts and developers to create machine learning models with high scalability, efficiency and productivity, while ensuring model quality. You can use automated machine learning to combine techniques and approaches to achieve a high-quality, recommended time series forecast. The list of algorithms supported by AutoML is very extensive.one of the important and useful aspects to consider and explore is the search for hyperparameters.
Hyperparameters are adjustable parameters that allow you to control the model training process, with neural networks, you decide the number of hidden layers and the number of nodes in each layer. Model performance is highly dependent on hyperparameters.
Hyperparameter tuning,, is the process of finding the parameters configuration that results in optimal performance. The process is typically expensive and manual by calculation.
The same kind of machine learning model can require different constraints, weights or learning rates to generalize different data patterns. These measures are called hyperparameters, and have to be tuned so that the model can optimally solve the machine learning problem.
Another aspect that could be talked about could be the automatic reading of graphs such as ACF and PACF for statistical models in such a way as to automatically obtain the optimizations necessary for the creation and adaptation of the models to the data.

Bibliografia

[1] *Matplotlib.pyplot#* (no date) *matplotlib.pyplot - Matplotlib 3.5.3 documentation*. Available at: https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.htm

[2] Documentation User Guide - statsmodels 0.14.0. Available at: https://www.statsmodels.org/stable/user-guide.html

[3] Team, K.*Keras Documentation*, *Keras*. Available at: https://keras.io/guides/

[4] *Statsmodels.tsa.statespace.structural.unobservedcomponentsresults statsmodels.tsa.statespace.structural.UnobservedComponentsResults - statsmodels 0.14.0*. Available at: https://www.statsmodels.org/stable/generated/statsmodels.tsa.statespace.structural.UnobservedComponentsResults.html

[5] Introduzione a tensorflow TensorFlow. Available at: https://www.tensorflow.org/learn?hl=it

[6] Pelagatti, M.M. (2016) *Time series modelling with unobserved components*. CRC Press.

[7] Matteopelagatti (2022) *Time series modelling with unobserved components*. Available at: https://ucmbook.edublogs.org/

[8] *Forecasting: Principles and practice (3rd ed) OTexts*. Available at: https://otexts.com/fpp3/

[9] Pelagatti, M.M. *Previsioni aziendali e valutazione dinamica dell'impatto di campagne pubblicitarie, eventi esterni e investimenti in marketing*. Available at: https://elearning.unimib.it/mod/resource/view.php?id=972897

[10] Hyndman, R.J. 'Forecasting using R', *Non-seasonal ARIMA models* M. University.

[11] Pelagatti, M.M. *Shiny app*, *UCM Simulator*. Available at: https://matteopelagatti.shinyapps.io/ucm_sim/

[12] Pelagatti, M.M.*State space*, *State Space Form Maker*. Available at: https://matteopelagatti.shinyapps.io/ss_maker/

[13] Pelagatti, M.M. *Italian electricity demand in the day-Ahead market*. Available at: https://matteopelagatti.shinyapps.io/sinusoid_approx/

[14] Mnassri, B. (2021) *Jena Climate Dataset*, *Kaggle*. Available at: https://www.kaggle.com/datasets/mnassrib/jena-climate?resource=download

[15] *Applying K-nearest neighbors to time series forecasting: Two new approaches*. Available at: https://arxiv.org/pdf/2103.14200.pdf

[16] Dobilas, S. (2022) *LSTM recurrent neural networks-how to teach a network to remember the past*, *Medium*. Available at: https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e

[17] Fleischer, J.P. *et al.* (2022) 'Time series analysis of cryptocurrency prices using long short-term memory', *Algorithms*, 15(7), p. 230. doi:10.3390/a15070230.

[18] Mohsen, S. (2023) *Recognition of human activity using GRU deep learning algorithm - multimedia tools and applications*, *SpringerLink*. Available at: https://link.springer.com/article/10.1007/s11042-023-15571-y

[19] Multiscale time series (1970) *SpringerLink*. Available at: https://link.springer.com/chapter/10.1007/978-0-387-70898-0_11

[20] Peixeiro, M. (2023) *The Complete Guide to Time Series Analysis and forecasting*, *Medium*. Available at: https://towardsdatascience.com/the-complete-guide-to-time-series-analysis-and-forecasting-70d476bfe775

[21] Athanasopoulos, G. *et al.* (2017) 'Forecasting with temporal hierarchies', *European Journal of Operational Research*, 262(1), pp. 60–74. doi:10.1016/j.ejor.2017.02.046.

[22] Forecast reconciliation: A review (no date) *Rob J Hyndman*. Available at: https://robjhyndman.com/publications/

[23] Kaltsounis, A., Spiliotis, E. and Assimakopoulos, V. (2023) 'Conditional temporal aggregation for time series forecasting using feature-based meta-learning', *Algorithms*, 16(4), p. 206. doi:10.3390/a16040206.

[24] Rostami-Tabar, B., Goltsos, T.E. and Wang, S. (2023) 'Forecasting for lead-time period by temporal aggregation: Whether to combine and how', *Computers in Industry*, 145, p. 103803. doi:10.1016/j.compind.2022.103803.

.