CYO Project: Price forecasting in the energy market

HarvardX PH125.9x - Data Science: Capstone

Emanuela Pannini

2024-12-07

Contents

Introduction

The objective of this project is to forecast the energy price that will be applied in Spain, utilising a combination of other energy market information and meteorological data. The original datasets were downloaded from Kaggle and will be presented below. The decision to pursue this field of research was made considering its significant impact on contemporary life, given the indispensable role of energy in the majority of our activities. Additionally, we were focused in examining a subject matter with real-world applications, excluding the purely academic fields of research, even if that is also interesting. On the contrary, this context presents a challenging environment for the implementation of this project, as numerous private enterprises and public institutions have previously attempted to achieve similar outcomes. The following sections presents my personal proposal, which relies on simple datasets and preliminary methodologies. In details, an exploratory analysis is provided, which first explains the features present in the dataset and demonstrates the points of strength and the anomalies. Subsequently, a seasonal analysis was conducted, which led to the identification of some additional features. After this, some window splits and models of prediction were presented. In attempting to do so, the dataset was divided into training, validation, and test sets using windows that relies on the well-established `caret` package [1]. In all cases, the performance will be evaluated using the `RMSE`, although in this context, other metrics such as `MAE`, `precision`, `recall`, and `F1 score` could also be applicable [2].

Analysis

This section outlines the steps employed in the generation of the Final dataset, which will be used in the models. It begins with an examination of the initial data, stored into two different `csv`, and continues through the various stages of processing and analysis till merging them in the last form. The former dataset contains information related to the energy market while the latter on weather data. In order to ensure consistency, sometimes the same approach may be employed twice, of course with different shades accordingly on the data. As the standard `summary()` was not satisfactory for the information retrieved, an additional summary was built by hand, showing all the information we would like to highlight. So, accordingly with the class of each column, the following statistics were evaluated:

- `min, max, mean, median, sd`: all this values came from the standard functions of R. If the column is numeric, it is a straightforward analysis otherwise some conventions were applied. If it is a date or datetime, only the `min` and `max` are updated, just to let the reader know the temporal range. Instead, if we are dealing with strings, all the evaluations are performed accordingly with the number of characters inside, as in general could be useful to understand the type of data inside
- `unique_values`: if the column is a character, it provides the number of unique values, otherwise it is not applicable
- `num_NAs, perc_NAs`: number of `NA` in the column and relative percentage

Energy dataset

The energy information are initially stored in the `energy_dataset.csv`, a dataframe with 35064 rows and 29 columns, that has the following schema:

Table 1: Energy schema

column_name	class
time	character
generation_biomass	numeric
generation_fossil_brown_coal_lignite	numeric
generation_fossil_coal_derived_gas	numeric
generation_fossil_gas	numeric
generation_fossil_hard_coal	numeric
generation_fossil_oil	numeric
generation_fossil_oil_shale	numeric
generation_fossil_peat	numeric
generation_geothermal	numeric
generation_hydro_pumped_storage_aggregated	logical
generation_hydro_pumped_storage_consumption	numeric
generation_hydro_run_of_river_and_poundage	numeric
generation_hydro_water_reservoir	numeric
generation_marine	numeric
generation_nuclear	numeric
generation_other	numeric
generation_other_renewable	numeric
generation_solar	numeric
generation_waste	numeric
generation_wind_offshore	numeric
generation_wind_onshore	numeric
forecast_solar_day_ahead	numeric

Table 1: Energy schema *(continued)*

column_name	class
forecast_wind_offshore_eday_ahead	logical
forecast_wind_onshore_day_ahead	numeric
total_load_forecast	numeric
total_load_actual	numeric
price_day_ahead	numeric
price_actual	numeric

where:

- `time`: date and time of the observation. In order to coherently manage the daylight saving time present in March and October this is stored in `UTC` timezone
- `generation_%`: all the features that start with generation stores the `MW/h` of generation for each energy source
- `%_forecast_%`: all the features that contain `forecast` in the name stores the forecast for the specific feature
- `total_load_actual`: the actual value of the total energy in the transmission system at that `time`
- `price_day_ahead`: the 1-day forecasted value for the energy price
- `price_actual`: the real price in `€/MWh` applied in the energy market. This column will be the target for all our models

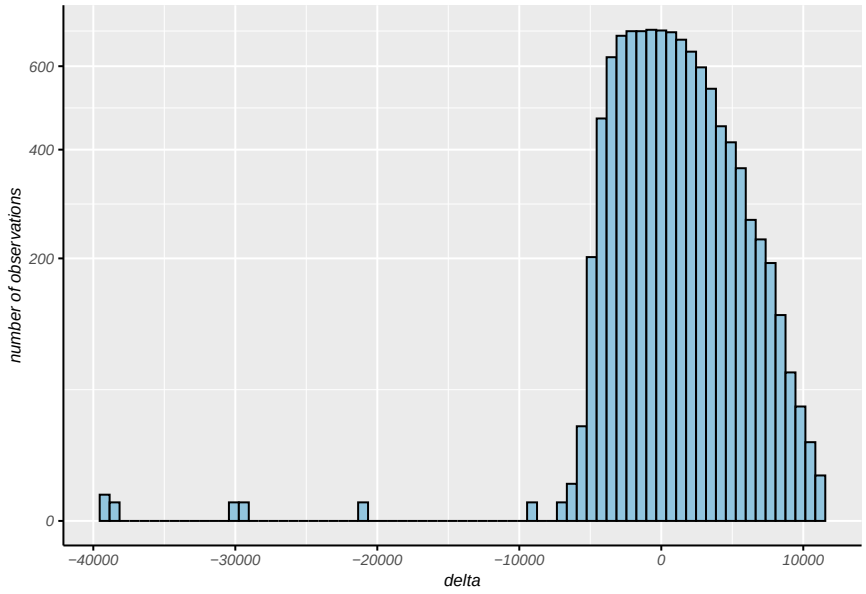
After this preliminary inspection, it was decided to look at the summary in order to understand how the numerical data is distributed and about any `NA`s. So, if we apply our summary to the energy dataset, we can notice the following statistics. We also would like to point out that we are not completely satisfied with this kind of visualisation, however it was decided that it was a good trade-off compared to the necessity of rotating the head to read the values.

Table 2: Energy summary

names	class	min	max	mean	median	sd	unique_values	num_NAs	perc_NAs
time	POSIXct, POSIXt	2014-12-31 23:00:00	2018-12-31 22:00:00	NA	NA	NA	NA	0	0.00
generation_biomass	numeric	0	592	383.51	367	85.35	NA	19	0.05
generation_fossil_brown_coal_lignite	numeric	0	999	448.06	509	354.57	NA	18	0.05
generation_fossil_coal_derived_gas	numeric	0	0	0	0	0	NA	18	0.05
generation_fossil_gas	numeric	0	20034	5622.74	4969	2201.83	NA	18	0.05
generation_fossil_hard_coal	numeric	0	8359	4256.07	4474	1961.6	NA	18	0.05
generation_fossil_oil	numeric	0	449	298.32	300	52.52	NA	19	0.05
generation_fossil_oil_shale	numeric	0	0	0	0	0	NA	18	0.05
generation_fossil_peat	numeric	0	0	0	0	0	NA	18	0.05
generation_geothermal	numeric	0	0	0	0	0	NA	18	0.05
generation_hydro_pumped_storage_aggregated	logical	NA	NA	NA	NA	NA	NA	35064	100.00
generation_hydro_pumped_storage_consumption	numeric	0	4523	475.58	68	792.41	NA	19	0.05
generation_hydro_run_of_river_and_poundage	numeric	0	2000	972.12	906	400.78	NA	19	0.05
generation_hydro_water_reservoir	numeric	0	9728	2605.11	2164	1835.2	NA	18	0.05
generation_marine	numeric	0	0	0	0	0	NA	19	0.05
generation_nuclear	numeric	0	7117	6263.91	6566	839.67	NA	17	0.05
generation_other	numeric	0	106	60.23	57	20.24	NA	18	0.05
generation_other_renewable	numeric	0	119	85.64	88	14.08	NA	18	0.05
generation_solar	numeric	0	5792	1432.67	616	1680.12	NA	18	0.05
generation_waste	numeric	0	357	269.45	279	50.2	NA	19	0.05
generation_wind_offshore	numeric	0	0	0	0	0	NA	18	0.05
generation_wind_onshore	numeric	0	17436	5464.48	4849	3213.69	NA	18	0.05
forecast_solar_day_ahead	numeric	0	5836	1439.07	576	1677.7	NA	0	0.00
forecast_wind_offshore_eday_ahead	logical	NA	NA	NA	NA	NA	NA	35064	100.00
forecast_wind_onshore_day_ahead	numeric	237	17430	5471.22	4855	3176.31	NA	0	0.00
total_load_forecast	numeric	18105	41390	28712.13	28906	4594.1	NA	0	0.00
total_load_actual	numeric	18041	41015	28696.94	28901	4574.99	NA	36	0.10
price_day_ahead	numeric	2.06	101.99	49.87	50.52	14.62	NA	0	0.00
price_actual	numeric	9.33	116.8	57.88	58.02	14.2	NA	0	0.00

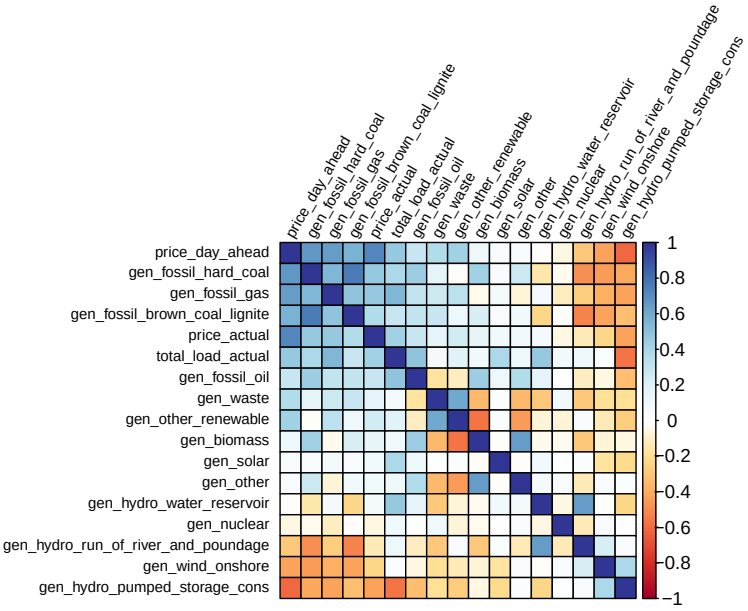
From the table above, it seems that there are no duplication inside, intuition confirmed also by the double check that states 0 as a result. Unfortunately, some columns has all `NA`s or zeros inside, so we will filter them out. The same will be also done for all the `%_forecast_%` ones, as they are not in the scope of this project. As an additional analysis, one could observe how these forecasts are correlated to the actual values and may try to include them in the models. This is not a simple approach, as it should also rely on different subset that could expand the current perimeter of features.

In order to perform a coherence test, it was decided to verify if the sum of all generation features is greater than the `total_load_actual` or not. As you can see:



we do not have a strong trend that could suggest oddities or unexpected behaviour, so we assumed that it is correct. Moreover, we do not have enough sensibility to validate the range of the values for the different energy sources, so the unique consistency test will be the one above. However, as we noted there are some `NA`s that are needs to be interpolated. In order to use different techniques, in this case we will use the `zoo` package [3], while in the following lines we will construct our linear model starting with `caret`. After that, we also verified and there are 0 `NA` values in the whole dataset, so the interpolation was correctly made.

Even though we are dealing with timeseries and the correlation analysis should be finer and more correct taking into account this aspect, for the moment only a standard correlation analysis between features has been carried out. In order to do so, the `corrplot` package was used [4], resulting in the following matrix:



As we can expect, the column `price_day_ahead` is highly correlated with the `price_actual`. Even if it may be a good idea to use also this one as a feature, we will remove assuming that in a real life scenario we don't want this prediction. Furthermore, the concept of forecasting the day-ahead price using in addition different `price_day_ahead` projections could also be a new project, maybe retrieving the values from TSOs but also from different teams or company that operates in the field. From our point of view, the other features are correlated in an acceptable range, so we decided not to remove anything else.

Weather dataset

In accordance with the aforementioned presentation on energy features, a parallel approach was adopted in the analysis of data within the `weather_features.csv` file. At the beginning we have a dataset with 178396 rows and 17 columns, that has the following schema:

Table 3: Weather schema

column_name	class
dt_iso	character
city_name	character
temp	numeric
temp_min	numeric
temp_max	numeric
pressure	integer
humidity	integer
wind_speed	integer
wind_deg	integer
rain_1h	numeric
rain_3h	numeric
snow_3h	numeric

Table 3: Weather schema *(continued)*

column_name	class
clouds_all	integer
weather_id	integer
weather_main	character
weather_description	character
weather_icon	character

where:

- `dt_iso`: date and time of the observation. In order to coherently manage the daylight saving time present in March and October this is stored in `UTC` timezone
- `city_name`: name of the city. Inspecting the values, one can immediately notice an additional blank space before `Barcelona`, so we removed this. Generally speaking, this activity is useful in order to clean character columns
- `temp, temp_min, temp_max`: these features report the average, minimum and maximum temperature in `Kelvin`, with hourly granularity. Although it is not used in Spain, we have not changed the scale to `Celsius` because it is not necessary
- `pressure`: pressure recorded value, in `hPa` scale
- `humidity`: percentage of humidity in the air
- `wind_speed, wind_deg`: these features report the wind speed (in `m/s`) and direction (in `degrees`). In this case, no further information has been provided on the altitude of the measurement
- `rain_1h, rain_3h`: these features report the `mm` of rain recorded in the last hour or last `3` hours
- `snow_3h`: `mm` of snow recorded in the last `3` hours
- `clouds_all`: percentage of cloud coverage
- `weather_id`: weather id assigned relying on different weather conditions
- `weather_main`: high-level summary of the weather in that hour
- `weather_description`: more detailed description of the weather parameters
- `weather_icon`: code of the icon used to display the weather in the site

So, producing the same kind of summary already displayed for energy data, it is possible to notice the paths and values below. In that case, the number of rows does not match our expectations, so we firstly started removing the21 rows, as they were completely doubled. We removed them because in that case they have no clues, as we are dealing with hourly observations. Moreover, it is also possible to notice some unexpected behaviours for some features, anyway they will be properly analysed in the following paragraphs.

Table 4: Weather summary

names	class	min	max	mean	median	sd	unique_values	num_NAs	perc_NAs
dt_iso	POSIXct, POSIXt	2014-12-31 23:00:00	2018-12-31 22:00:00	NA	NA	NA	NA	0	0.00
city_name	character	6	9	7.19	7	NA	5	0	0.00
temp	numeric	262.24	315.6	289.62	289.15	8.03	NA	0	0.00
temp_min	numeric	262.24	315.15	288.33	288.15	7.96	NA	0	0.00
temp_max	numeric	262.24	321.15	291.09	290.15	8.61	NA	0	0.00
pressure	integer	0	1008371	1069.26	1018	5969.63	NA	0	0.00
humidity	integer	0	100	68.42	72	21.9	NA	0	0.00
wind_speed	integer	0	133	2.47	2	2.1	NA	0	0.00
wind_deg	integer	0	360	166.59	177	116.61	NA	0	0.00
rain_1h	numeric	0	12	0.08	0	0.4	NA	0	0.00
rain_3h	numeric	0	2.32	0	0	0.01	NA	0	0.00
snow_3h	numeric	0	21.5	0	0	0.22	NA	0	0.00
clouds_all	integer	0	100	25.07	20	30.77	NA	0	0.00
weather_id	integer	200	804	759.83	800	108.73	NA	0	0.00
weather_main	character	3	12	5.29	5	NA	12	0	0.00
weather_description	character	3	28	11.95	12	NA	43	0	0.00
weather_icon	character	2	3	2.89	3	NA	24	0	0.00

Descriptive features

If we look at the `weather_id`, `weather_main`, `weather_description` and `weather_icon` columns we can imagine that they are highly correlated and provide information that should be neglected. Even if we cannot directly perform a correlation analysis on these characters columns, we can immediately notice that the `weather_description` is just an addition to the `weather_main`. In order to conclude something in a more robust manner, it was decided to display the key steps also there. So, we started adding some relevant features at the distinct combination of the `4` columns above, in particular:

- `simplified_icon`: the `weather_icon` column sometimes has a `d` or `n` suffix that seems to be related to `day` or `night`. This new feature removes this part so it should be easy to understand some relationships
- `icon_day_night`: when present, it extracts `d` or `n` from the `weather_icon` column. It was performed as we would like to understand if it is possible to determine some hourly ranges to confirm that they refers to day and night
- `day_night`: we performed a split based on the hours, trying to understand if we manage to match the values reported in `weather_icon`

In order to confirm the general idea behind `day_night`, we defined quite small hourly intervals (in `CET/CEST` timezone) where we expect day or night as a value and, as shown here:

Table 5: Day-night test

day_night	icon_day_night
	d
	n
d	
d	d
n	
n	n

we have a complete match, when valorised. Of course, this is not a complete check, as we still have blank cases, both on our inspection and on the original data. However, for us it is enough to confirm that the idea is correct and to proceed using the `simplified_icon` in the following steps. At this stage, we expected that the `simplified_icon` always matches with the `weather_id`, however this is not the case, as shown by this exception:

Table 6: Multiple combinations id-simplified icon

weather_id	weather_main	weather_description	simplified_icon
800	clear	sky is clear	01
800	clear	sky is clear	02

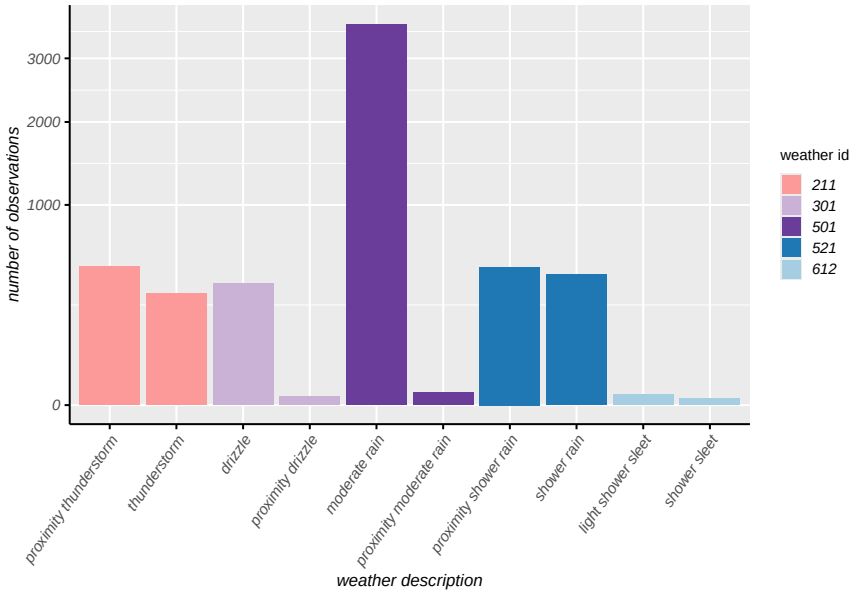
Unfortunately from this picture we can not deduce anything else, so we tried to understand if this difference is related to other non numerical features. For example, it was verified if some are old names (i.e. different ranges for `dt_iso`) or if the site used different icons for different `city_name`, however also this idea is not applicable:

Table 7: Check on non-numerical features

weather_icon	min_date	max_date	num_city
01	2014-12-31 23:00:00	2018-12-27 00:00:00	5
01d	2015-01-01 07:00:00	2018-12-31 17:00:00	5
01n	2014-12-31 23:00:00	2018-12-31 22:00:00	5
02	2015-01-01 15:00:00	2018-12-19 03:00:00	5
02d	2015-01-06 07:00:00	2018-11-28 10:00:00	5
02n	2015-01-02 20:00:00	2018-12-23 01:00:00	5

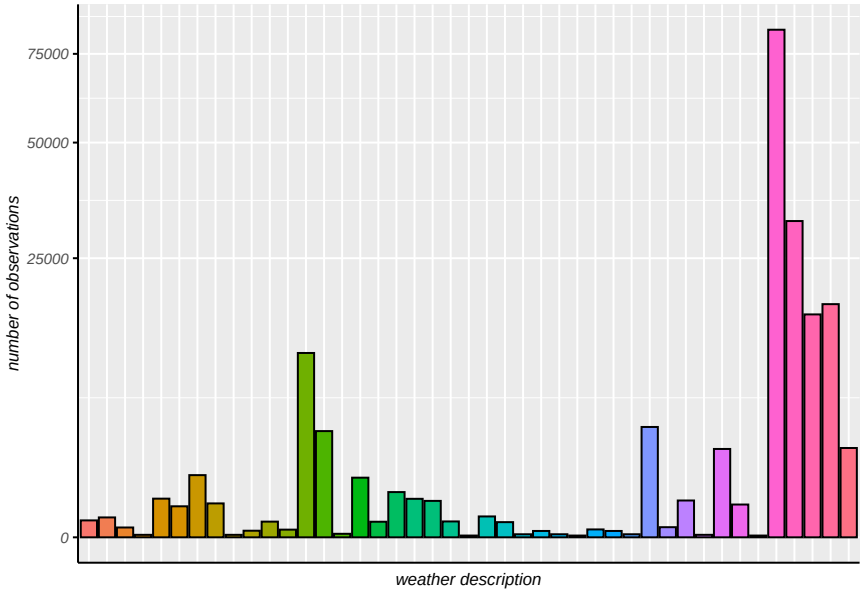
as they are both present at the beginning and the end of the dataset and for all cities.

After that, we tried to understand the existing link between `weather_id` and `weather_description`, resulting in a similar situation where we have more descriptions for the same id (true for 5 ids). Conversely, the reverse is correct, confirmed by the fact that for each `weather_description` we have just 1 `weather_id`. At this point, we may assume that the `weather_description` provides a more detailed information if compared with the `weather_id` or that someone reported in a wrong way some fields. Looking at the data



and comparing the frequencies, in some cases seems the first case, while in others the latter.

Generally speaking, we have the same behaviour also for the cases where we have `1-1` correspondence between `weather_description` and `weather_id`, as you can see from the plot below (each colour is a different `weather_id`):



Moreover, as shown from the table below and as expected, is quite immediate to see that `weather_description` and `weather_main` are coherent and the description is just an addition:

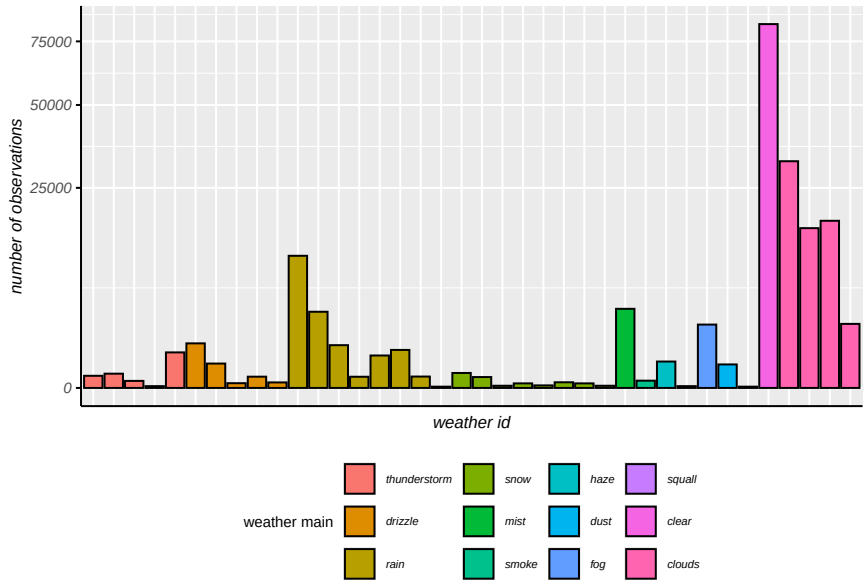
Table 8: General coherence main-description

weather_main	weather_description
clear	sky is clear
clouds	broken clouds
clouds	few clouds
clouds	overcast clouds
clouds	scattered clouds
drizzle	drizzle
drizzle	heavy intensity drizzle
drizzle	light intensity drizzle
drizzle	light intensity drizzle rain
drizzle	proximity drizzle
drizzle	rain and drizzle
dust	dust
dust	sand dust whirls
fog	fog
haze	haze
mist	mist
rain	heavy intensity rain
rain	heavy intensity shower rain
rain	light intensity shower rain
rain	light rain
rain	moderate rain
rain	proximity moderate rain

Table 8: General coherence main-description *(continued)*

weather_main	weather_description
rain	proximity shower rain
rain	ragged shower rain
rain	shower rain
rain	very heavy rain
smoke	smoke
snow	heavy snow
snow	light rain and snow
snow	light shower sleet
snow	light shower snow
snow	light snow
snow	rain and snow
snow	shower sleet
snow	sleet
snow	snow
squall	squalls
thunderstorm	light thunderstorm
thunderstorm	proximity thunderstorm
thunderstorm	thunderstorm
thunderstorm	thunderstorm with heavy rain
thunderstorm	thunderstorm with light rain
thunderstorm	thunderstorm with rain

The same plot can be also developed using `weather_id` and `weather_main`, showing the clusters below:



In conclusion, it is possible to deduct that all these features are correlated with `weather_id` so we will remove all the other ones. Moreover, if we look at the different `weather_description` we can suddenly notice that all the information added in a textual form in this field can be derived from the other numerical features at the beginning of the dataset (i.e. `temp`, `pressure`, etc.). Also, adding these features in a coherent way requires too much effort if compared to the benefits so, for the scope of this project, it was decided to remove

them. In the future we may modify this part to include this section as well, eventually through a numerical standardised form.

Furthermore, as we have found that with the same exact conditions of the parameters the `weather_id` can be different, we removed also this last feature. Of course, it is possible to investigate also this last part and eventually deep dive into the understanding the real meaning of this column, however this is also out of the scope of this project at the moment. Moreover, it should be necessary to properly re-parametrise the `weather_id`s and sort them in way that intrinsically extrapolate how good or how bad is the weather, so that it could be possible to extrapolate a kind of ordering to allow `numeric` comparisons. Hence, after all we can conclude that if we remove `weather_id` we always have the expected number of rows

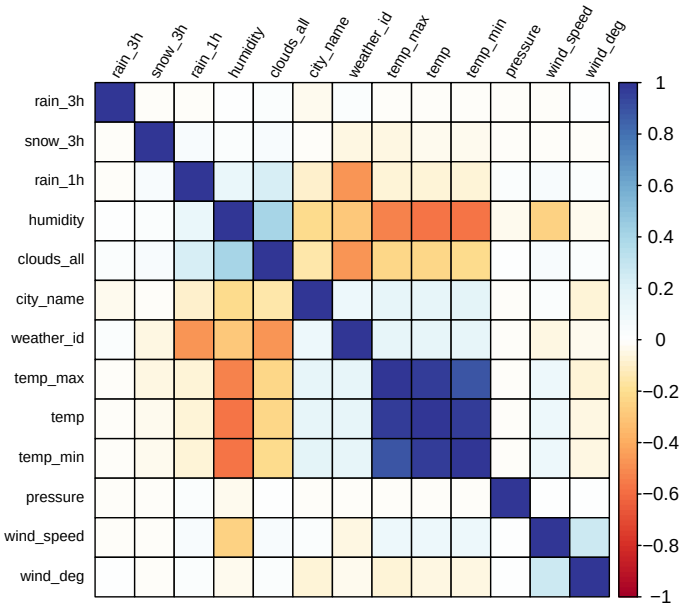
Table 9: Weather numerics

city_name	num_rows
Barcelona	35064
Bilbao	35064
Madrid	35064
Seville	35064
Valencia	35064

confirming that we managed to remove all doubled ones. So, we will take the dataset in this form to start with the real analysis of the others climatological features.

Correlation analysis

As for the energy data, it is better to perform correlation taking into account the timeseries aspect, however for the numerical features it was decided to keep it simple. At this time, it was also established to add `city_name` and `weather_id` even if they are not properly correct in order to verify if some strange behaviour appears and we would like to remove the latter one. The correlation analysis produced the following matrix:

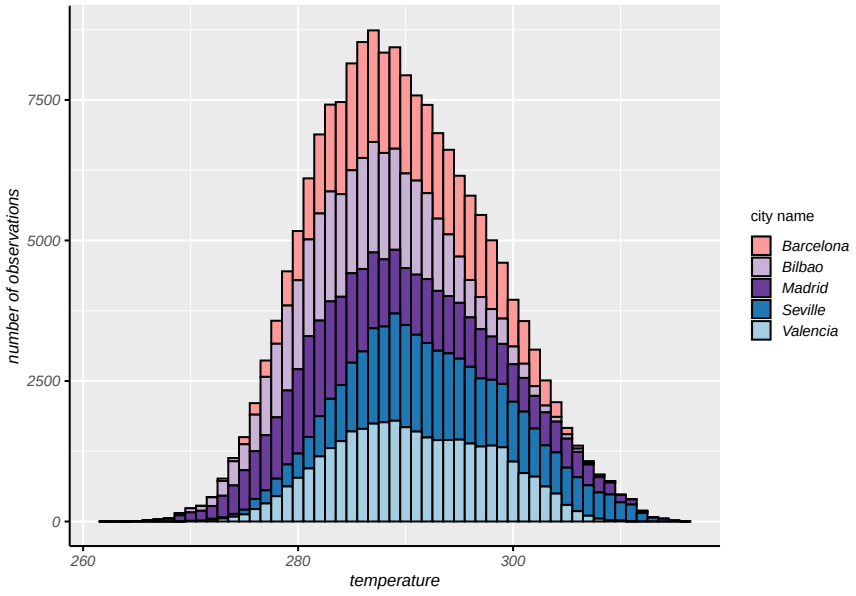


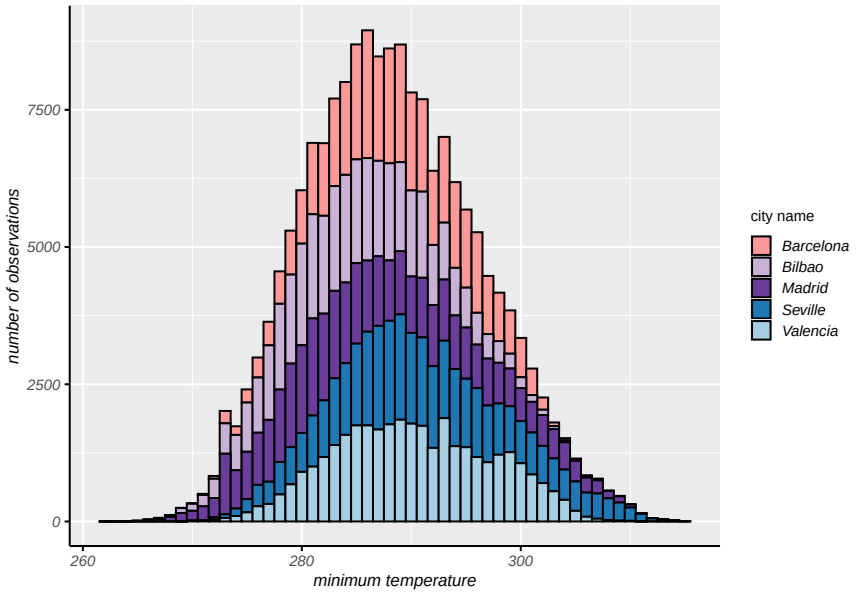
and ones can immediately notice how `temp`, `temp_min` and `temp_max` are correlated, as expected. Moreover, we also expect the same path between `rain_1h` and `rain_3h` but this will be properly investigated in the following. In particular, during the next sections we will report a detailed analysis for each feature, defining some acceptability ranges and general coherence tests. It is important to notice that it is of course possible to define the ranges at a fine level and/or to mix information from different datasets, in order to increase precision and accuracy. However, to limit the scope of the activities, we will present the ranges in a fixed form (i.e. no years or city details) and we have not imported any additional source.

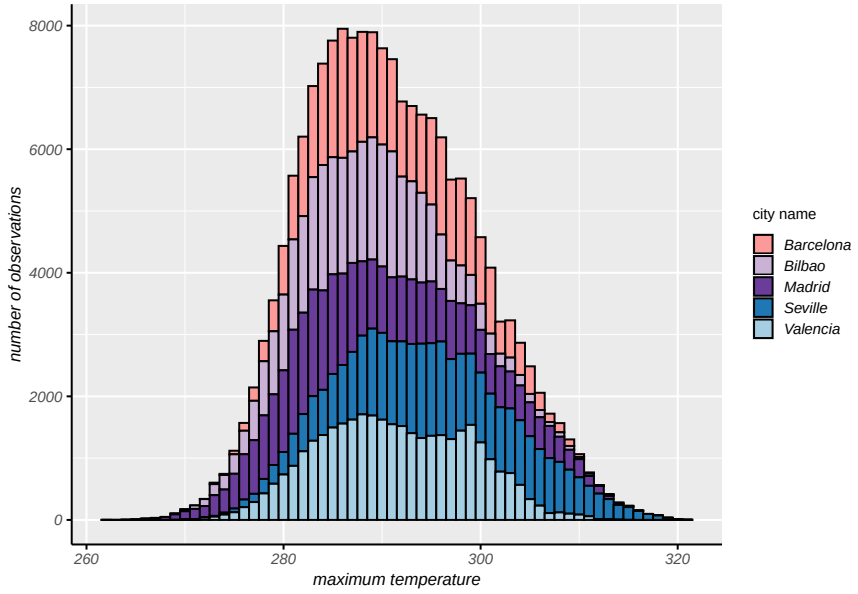
Temperature

As anticipated above, the temperature is expressed in `Kelvin` and we have `3` different features for the same measurement. First of all, we will show the general trend for each one and after the cross coherence.

Below you will see the different temperatures, also divided by cities:



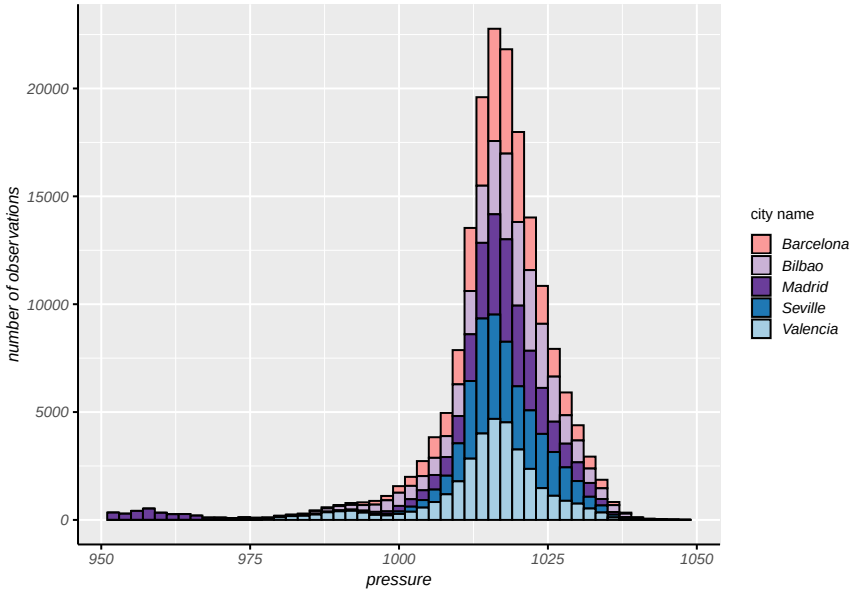




This difference was highlighted as these cities are located in different areas of Spain so we may expect different trends, even if similar. Furthermore, we can assure that these `3` variables are consistent, as we have 0 rows where the temperature is greater than the maximum and 0 rows where is lower than the minimum. So, as we did not see any unexpected behaviour, we can remove the minimum and maximum and leave only `temp` as an indicator for this measurement. After that, we also decided to verify if all the values are within an acceptable range. On our research it was found that on AEMET it is possible to find a double check for recorded temperatures, however to keep this simple it was decided to take all these ranges in a static way [5]. Then, we tried to filter to select the observations outside the range and we found 0 occurrences. Even if it is possible to refine this analysis and double check the values at a finer lever (i.e. yearly, monthly, etc..), this confirmed that all the temperatures are within the range of acceptability so no other activities are needed.

Pressure

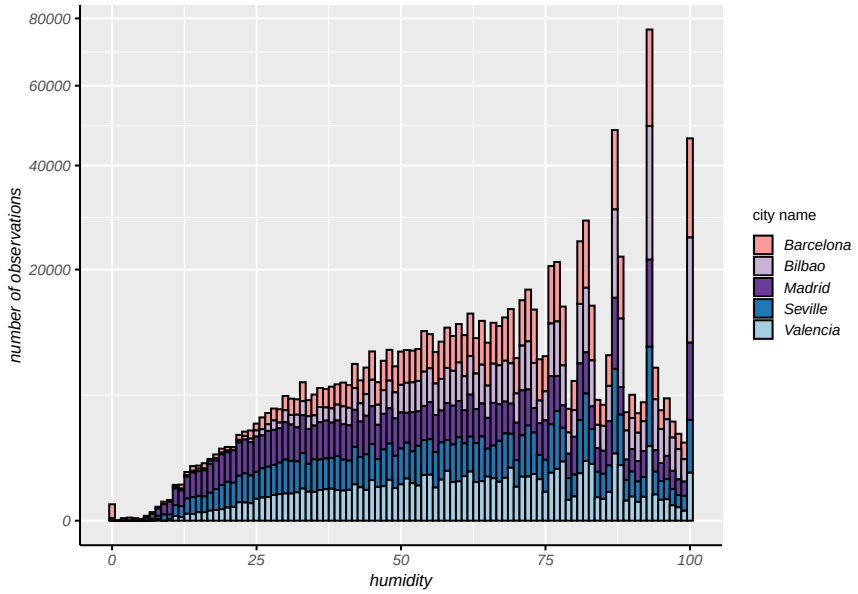
The general idea is the same as above, however for `pressure` there were some unexpected values, also detected on the summary. As they are quite extreme, we can surely predict that they are outliers that need a manual fix however, before filtering them out, the same steps were performed. In this case the acceptable ranges [6] let us clearly detect the oddities, resulting in the following distribution:



So, we decided to replace all these observation with `NA` and then to interpolate to fix the situation. Here we also established to not pivot the dataset for now so that the model can also use the other cities as target and not only the past. For the sake of simplicity, we interpolated just using a linear model, even if other techniques (like splines) may be better. Moreover, we can notice that we have just 1087 `NA`s, that are few if compared to the number of rows. Also, it is possible to point out that sometimes they refers to the same days, maybe suggesting an error in the measurement or in the sensors. From a technical point of view it is clearly possible to use the same technique as above [3], however a linear model using `caret` [1] was implemented in order to explore different approaches. After that, we updated the values in `weather_date` replacing with the prediction if pressure is `NA` (i.e. it was outside the range), otherwise leaving the value as it was. So, when we checked again if `NA`s or outliers were still in the dataset, we found 0 rows, confirming that the interpolation was effective.

Humidity

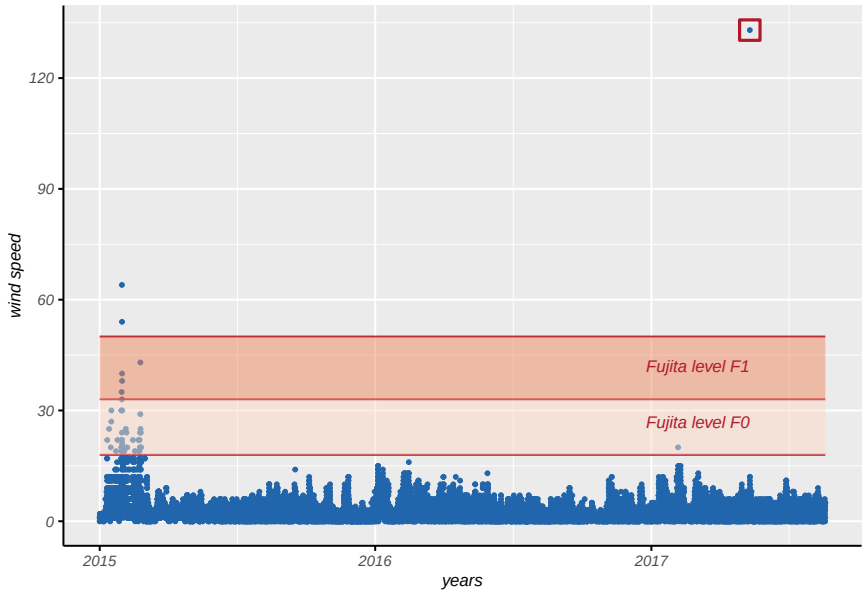
In this case we will simply report the general trend for `humidity`, as we did not found any strange behaviour and all the values are percentages in the range `0-100`:



Unexpectedly, we can notice that for the majority of cases we do not have spikes in correspondence of tens or multiples of `5`, so the reported quantities suggest to be quite accurate.

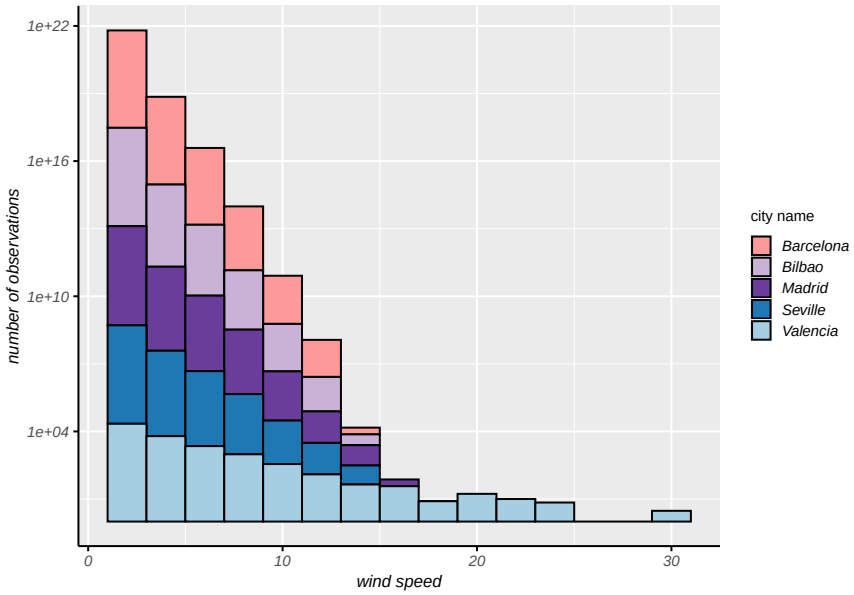
Wind

In accordance with the previously mentioned temperature, also in this case we have two features related to the same measurement. However, it is clear that we can not expect the same correlation between the wind speed (`wind_speed`) and direction (`wind_deg`). Furthermore, similarly to what was reported for `pressure`, from the summary were noticed unexpected values for `wind_speed`, so we can suppose to filter them out. In this case the minimum acceptable value is `0`, so no further checks are needed (excluding precision of measurements and similar issues). On the contrary, the maximum `wind_speed` can not be easily determined as extreme events can happen. Firstly, the Fujita scale was considered, which provides the parameters used to classify tornados. In particular, the minimum `wind_speed` required to classify an event as a tornado is the lower bound of the `F0` band (approximately 18 `m/s`) and it was concluded that all values below this should be accepted as correct. After that, the dataset was inspected more and we found that only `city_outside_wind_speed` has values outside this range. It could be partially explained by the fact that it is a windy city, so we tried to understand which dates are in this situation, noticing that the first one is 2015-01-11 01:00:00 and the last one is 2017-05-11 10:00:00. Taking into account all these considerations, the plot below was created:



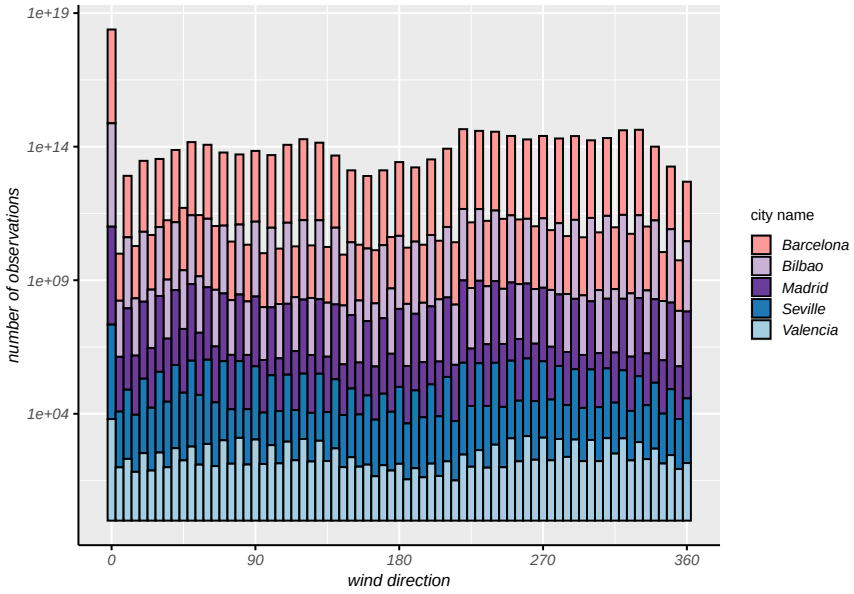
As we can observe, the extreme value in the square on the top right of the plot is clearly an unexpected outlier, maybe related to a conversion issue from `km/h` to `m/s`. However, some of the points at the beginning of `2015` seems to be coherent, also considering that Valencia is a windy city and that at the end of `2014` it unfortunately faced a severe storm. So, we decided to consider acceptable all the values within the `F0` band in Fujita scale and to interpolate the others. In this case it was followed the idea to create a dataset with only values related to `Valencia`, as we noticed that this city has a different trend. Of course, also this time it was possible to use a different approaches however we adopted again the linear model estimation.

So, after the interpolation the general plot for the `wind_speed` is as below, after a logarithmic scale on y-axis



Moreover, we also double checked the results and it was confirmed that now 0 rows does not fill in the range.

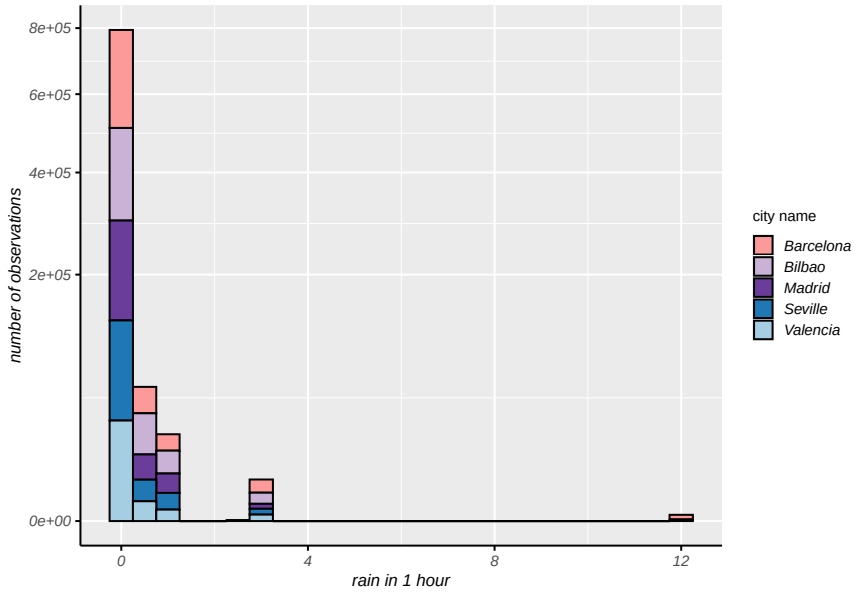
Similarly, an analysis was also performed on `wind_deg` feature, resulting in



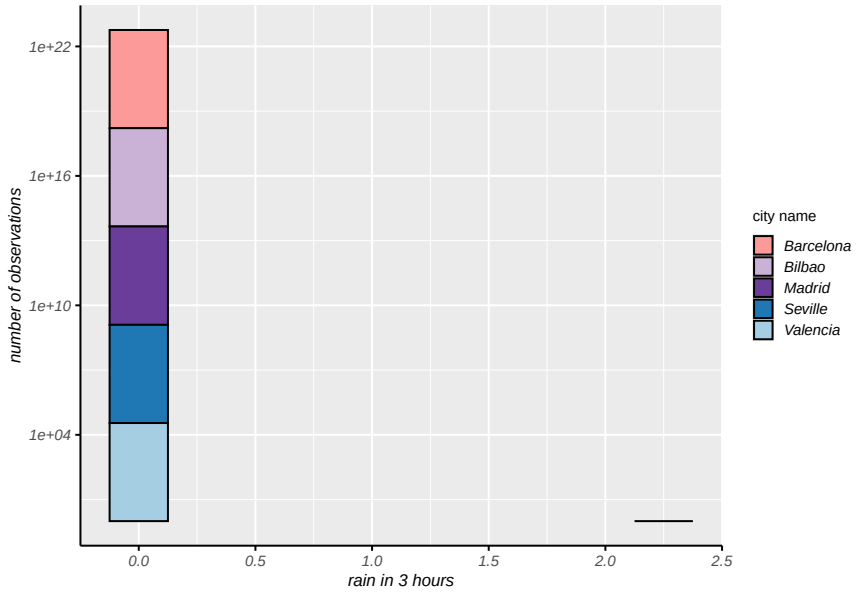
From a technical point of view, this plot is completely coherent as all the values are between `0` and `360` and as we have peaks at multiples of `10`, as observations may be approximated. However, it is quite strange that when `wind_speed` is `0` we still have a value for `wind_deg`, so we may deduct that the sensibilities of the observation operate at different precision. Moreover, the great majority of records is at `0`, so it may suggest that `0` is also used as a base value and sometimes it was not coherently updated. Unfortunately, I personally do not have enough mastery and experience with these kind of data and an extensive research for additional reliable weather data would probably take too much time, so we will just leave these values as they are.

Rain

Once more, also for the precipitation variable we have two different features, `rain_1h` and `rain_3h`, theoretically displaying the water accumulated in the last interval of time. So, we started with defining acceptability ranges for the hourly values and then multiplying them for the `3` hours window, even if this is not extremely precise. Again, the minimum value is `0` and we used the description presented in [7] to define the maximum acceptable value accordingly with the heavy rain condition (`50 mm/h`), resulting in



So, even if the plot is quite shifted, here we will keep these values as they are, trying to understand additional deductions with the `rain_3h` feature. In this project, the ranges were developed simply multiplying by `3` the previous ones, although this is not completely true from a meteorological point of view, this is still a good approximation for this kind of analysis. Again, all the values are within the range and below you can find the plot



As one can easily observe, this plot is quite strange, both when taken as a standalone and also when compared with the `rain_1h` distribution. In general, if we check the values we expect that `rain_1h` is always less or equal than `rain_3h`, however this is not always the case and below you can find the different deltas:

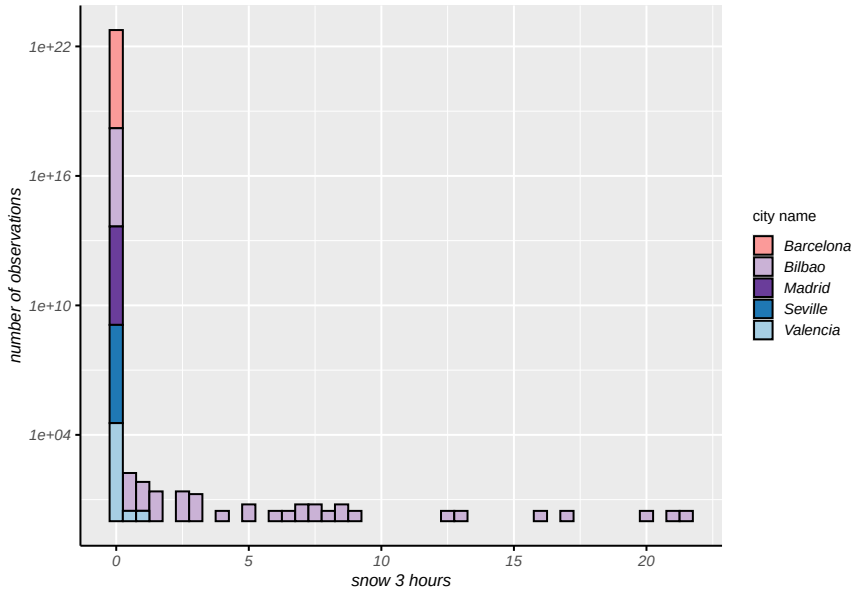
Table 10: Difference rain 1h - 3h

delta	num_observations
12.00	85
3.00	1310
2.29	1
0.90	5196
0.30	12794
0.25	1

Unfortunately, the deltas are not always small and instead sometimes they are significant, especially when related to hourly observations. So, taking into account this aspect and the general plot, we will remove the `rain_3h` feature from the scope of the analysis.

Snow

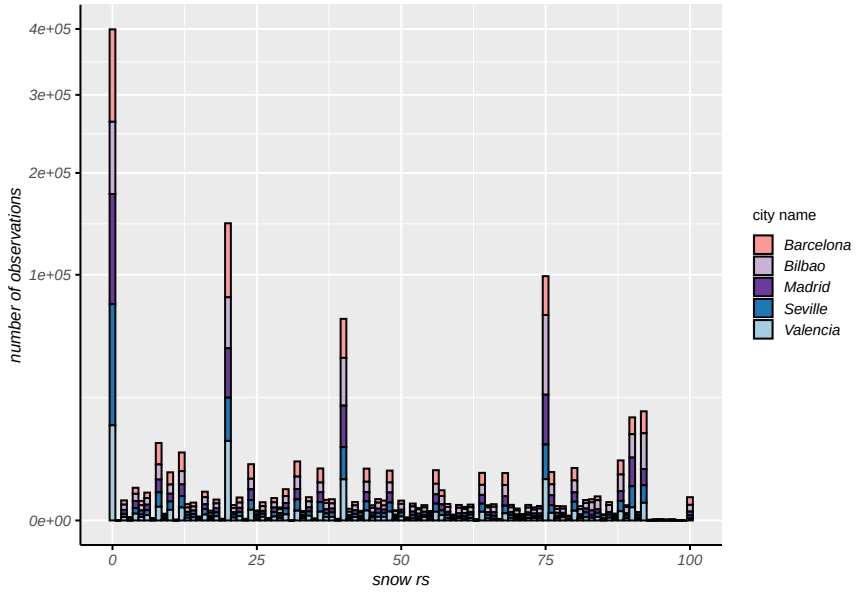
The feature `snow_3h` reports the amount of snow collected during the last `3` hours and one more time we present the plot to show the general trend



As was stated in [8], sometimes areas like `Madrid` reported severe snowfall, so we left the acceptable range quite high to be cautelative. The article specifies the centimeters of snow in `30` hours, so we just divided and changed the unit of measurement. Even if the plot is as above, it is possible to notice that 0 rows do not respect the conditions, so no further actions are needed and will be applied on this variable.

Clouds

At this section, it is quite straightforward to predict the next deduction and, as shown from here



it is easy to see that all the values related to `clouds_all` are in the correct limits for percentage. Furthermore, in this case it is possible to notice some peaks in correspondence of some tens and `75%`. From these spikes we may deduct that it is not always easy to evaluate this percentage with units sensibility, so some approximations also occurred in the input data.

Aggregated dataset

Summarizing, in the previous sections the two original dataset were analysed on their own. Now, the general idea is to merge them in order to provide to the models an unique entity with all the meaningful features. First of all we removed all the unnecessary features and the doubles, in fact at this step the `weather_data` does not have any other repetition:

Table 11: Check no doubles on weather data

city_name	num_obs
Barcelona	35064
Bilbao	35064
Madrid	35064
Seville	35064
Valencia	35064

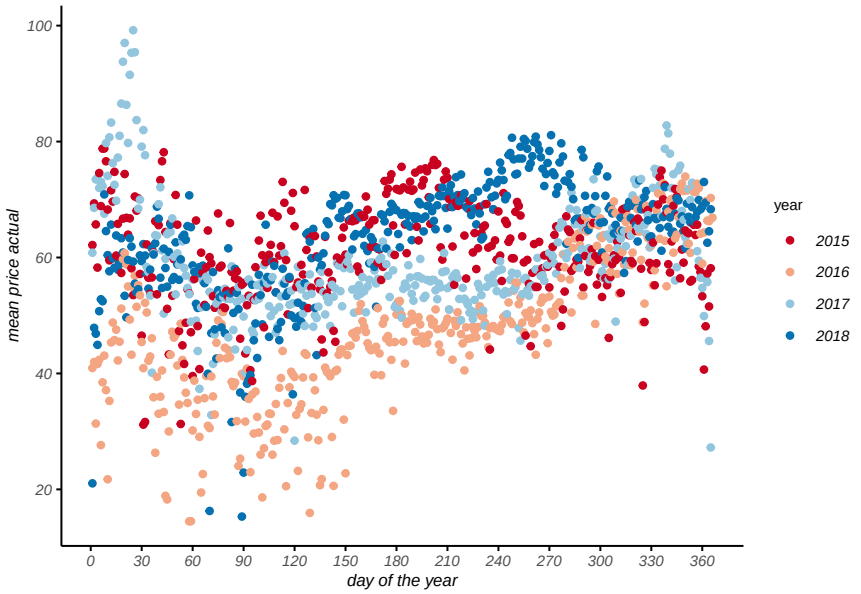
The `price_actual` is the same for every city in Spain and, since this is our target, we decided to weight the weather data of each city accordingly to its population, which varies over the years. We are aware that a more accurate way of weighting the data would also take into consideration the density, urbanisation, industrialisation or the metropolitan area of the city, however we decided that the population count was enough. Our current results are based only on `5` cities but this perimeter could also be further extended with the weather information of the others cities, although this is beyond the scope of this analysis. Here we have retrieved the cities population and the Spain population data through the years and stored them

into a dataframe. At the end, even though the sum of the population of these `5` cities cover nearly the `30%` of the Spanish population, we decided to use the ratio between their populations as a weight of the weather of each city to produce an overall weather data.

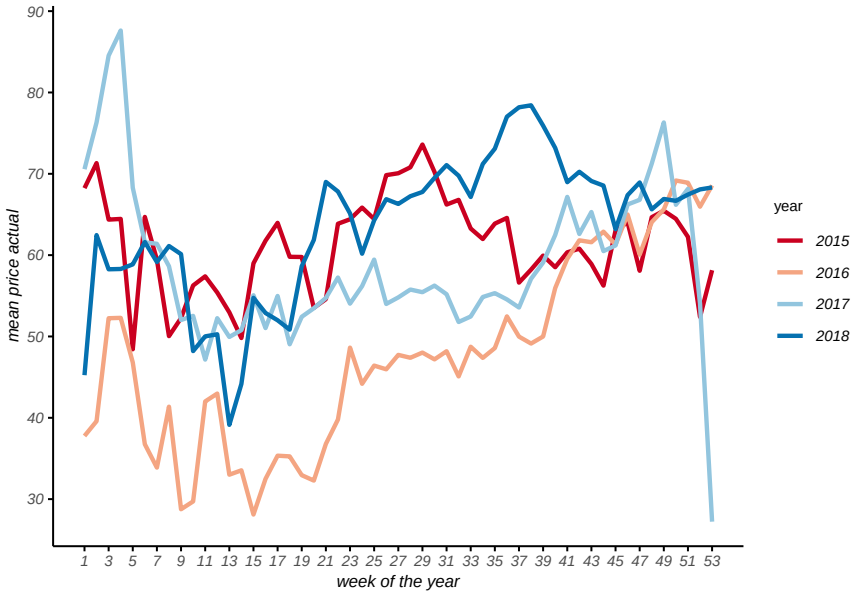
The two datasets were then joined on `time` and `dt_iso` columns. This is one of the key steps where it was necessary to have the datetimes in the `UTC` timezone, otherwise we would not have a unique match when the hour shifts. In the following sections we will use the `CET/CEST` timezone, as this also takes into account daylight saving time.

Focus on timeseries aspects

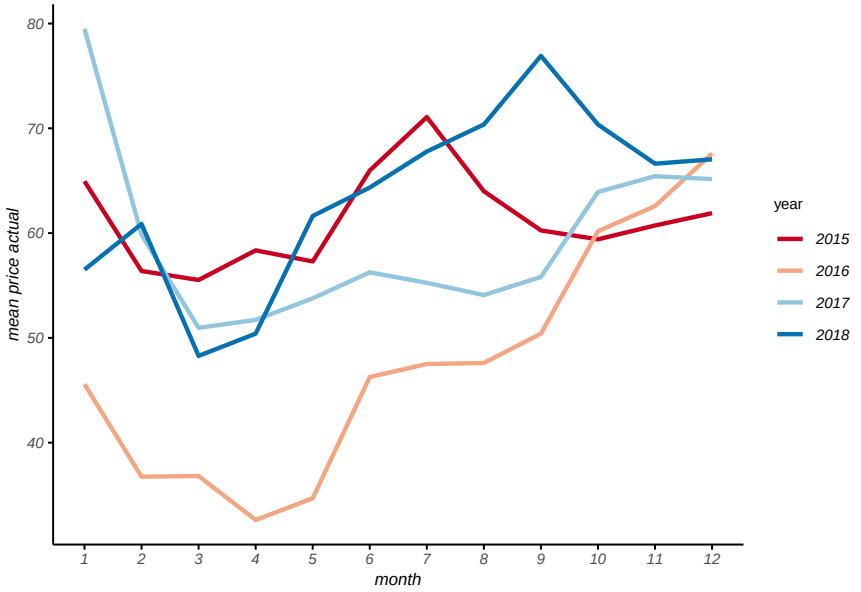
As we are dealing with a timeseries, it should be important to take a look at the seasonal effects, in order to understand which might be the most relevant impacts in the area. From general knowledge of the subject, we expect to see effects related to the `day`, the `month`, the `year` and also the `hour` within the same day. So we started by adding these relevant features and also the ones related to the day of the year, the week of the year, the day of the week and whether it is a business day. For this last feature, we decided to include as non-working day all weekends and the Spanish national holidays. In general, these ideas are confirmed by the plots below and by some punctual observations. The first approach was simply to plot these features trying to see if there were any repeating patterns; hence you can see the similar trend in relation to the days of the year,



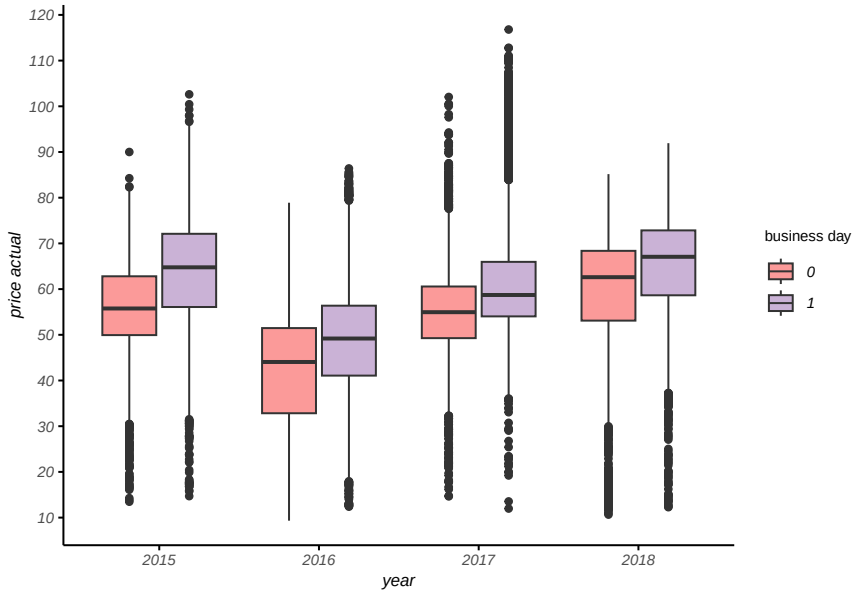
the weeks



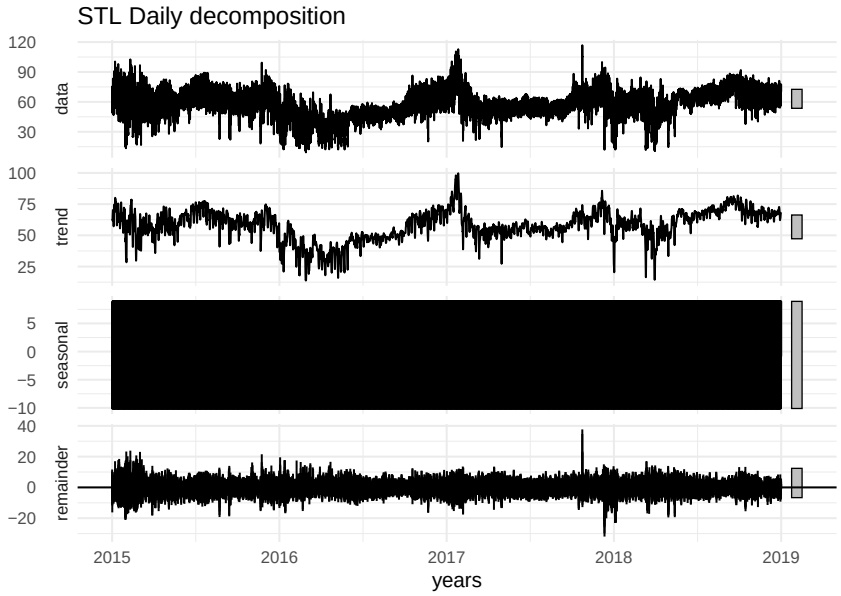
or the months



Moreover, it also seems to confirm that the `business_day` information give us a good insight on the general fluctuations

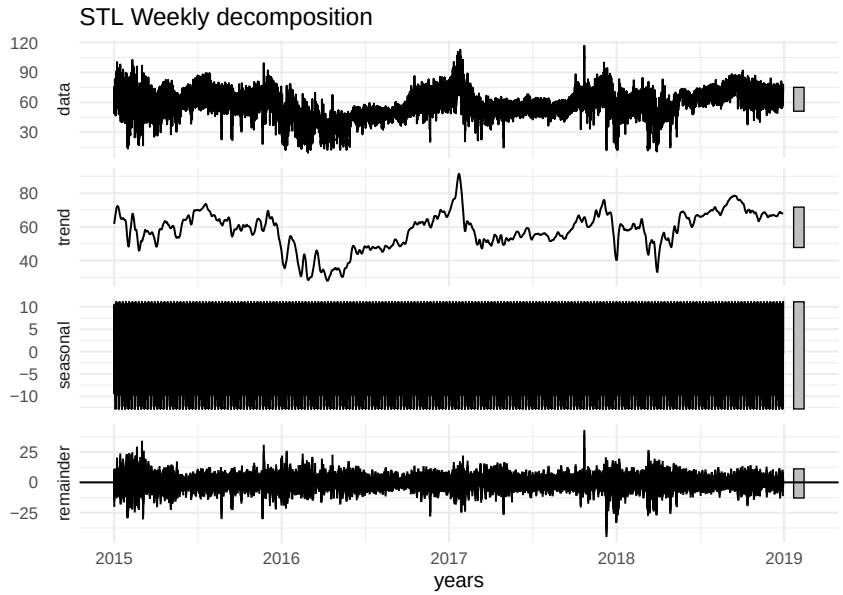


Although these simple plots confirmed the idea that these factors influence the final energy price, it was decided to perform a more detailed analysis of seasonality, in order to better distinguish between the different paths. From a technical point of view we used the STL decomposition in order to perform all the splits, as also explained in [9]. As it is useful to zoom-in and zoom-out in order to understand the dependencies properly, we used the `plotly` library which allows exactly this. However, the `pdf` output does not allow this kind of flexibility, so we have to include either a static version, a pre-determined movement or a link to an external resource. Here you will see the static version, although you may be interested in running the code and looking at these objects. So, for the daily decomposition, it is possible to notice

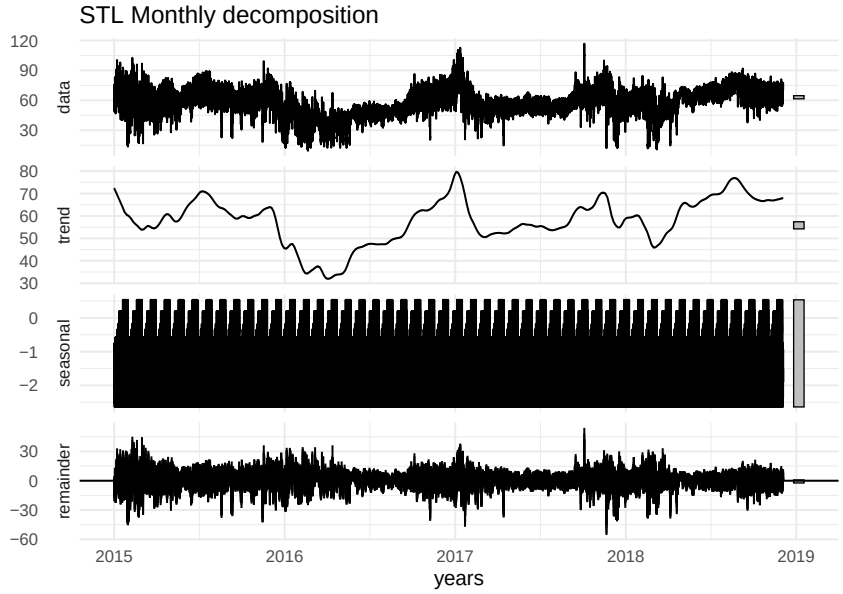


the trend and the seasonality.

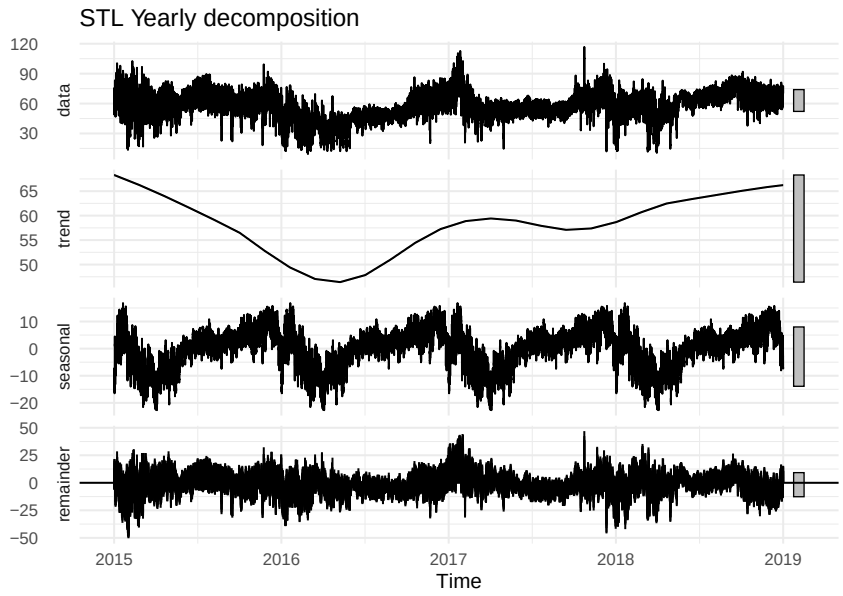
We are aware of the fact that from this screen it is not possible to inspect the seasonal behaviour however, if you look at the `daily_plotly` you can zoom-in and immediately visualise it. The same is also true at a weekly aggregation (`weekly_plotly`),



at a monthly (`monthly_plotly`) level

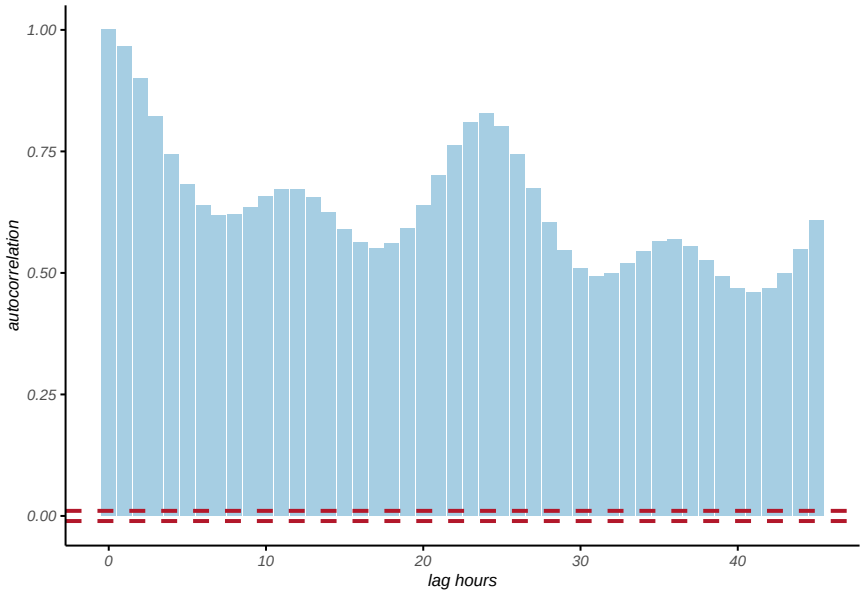


and at an yearly (`yearly_plotly`) one



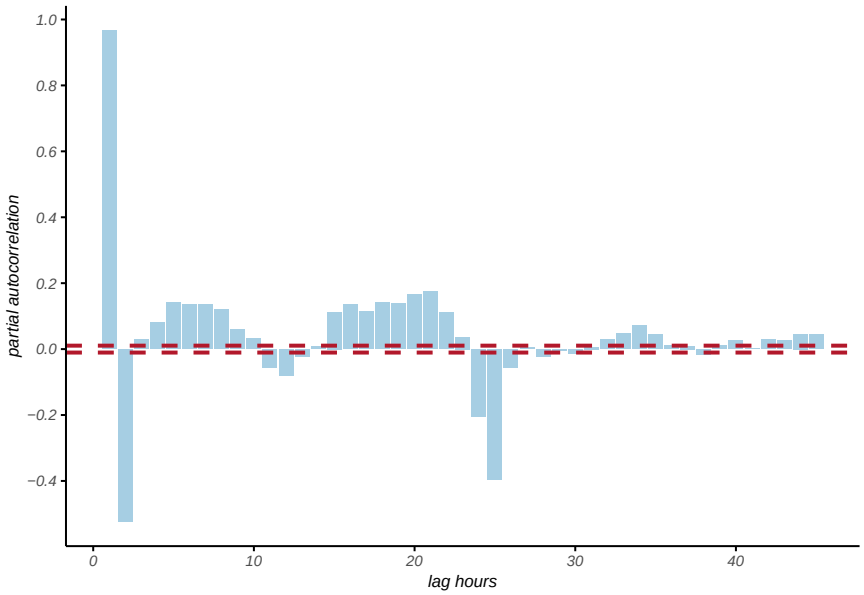
In all the above plots, a clear seasonality pattern and an oscillating trend can be observed, as expected from our initial findings. As we are dealing with timeseries, this should be taken into account also when evaluating the cross-correlation, in order to obtain a better result. As another possible improvement, we have also found a more detailed analysis of seasonality, which can be carried out with the Augmented Dickey-Fuller, Kwiatkowski-Phillips-Schmidt-Shin test or Fourier analysis. Although we honestly believe that these might be good ideas for extending this model, they were considered outside the scope of this project.

An auto-correlation analysis was also carried out to check whether the timeseries was auto-correlated:



and, as explained in [10] and on other online resources, we can deduce that this timeseries is auto-correlated with seasonal path, as expected from a general knowledge of the context and the graphs above. We have

also added the lines that refer to the confidence level of 95% and, as expected since the timeseries is quite long, they define a small range which is clearly exceeded by the plot. Similarly, we have also carried out a test to check for partial autocorrelation, that is shown in the plot below:



In this last picture it is possible to discriminate the correlation between two different observations without taking into account the effects associated with the observation in between, so making it easy to understand for how long the effects propagate over time.

Final dataset

Having completed all the preliminary analysis, we are ready to construct the final form of the dataset that will be used as input for the models that effectively forecast the prices. First, as we have included all the temporal features in a separate column, we will also remove the column `spain_time` to avoid redundancy. So at this point we have in the dataset all the features related to the time information, the relevant energy features and the weather measurements weighted by the population data. Following a fairly standard approach for machine learning studies, we would like to split the whole set into training, validation and test sets. However, as the dataset is a timeseries, it is not appropriate to perform a simple random split. Instead, it is essential that the splits are performed in a chronological order, so that they can accurately reflect the real approach and the seasonality aspects. To do this, it is possible to implement one of the two major ideas for the window function; a sliding window or an expanding window [1]. In the former case, the window is fixed in size and it is simply moved through the dataset to perform multiple runs. In the latter case, each time that the window moves, it increases the size of the training set, while the size of the test set remains fixed. Choosing the sliding window makes it easier to compare performance as the exact same situation is analysed each time, although this is not always easy to manage when the dataset is not so large and the model has to take into account a lot of different seasonal information. From the opposite perspective, the expanding window is an improvement in that it facilitate the model to learn from past data, however it creates a challenging situation when attempting to compare the results and to properly deal with overfitting. When trying to predict the `price_actual` for the following 24 hours, we believe that the dataset is large enough to use a sliding window of 31 days. From a theoretical perspective, we could also adjust the proportion of data in the training set to achieve optimal results, also taking into account performance. However, for the purposes of this analysis, we can assume that the dataset is sufficiently robust, given the results of the analysis phase and the tests conducted manually. Nevertheless, for the simpler model we will also show the difference between these two approaches. First of all, it was decided to keep the last 14 days of the dataset for testing and the rest for training and validation purposes. This idea could properly simulate the real use of the final model after a training period. At this point it is also necessary to split between training and validation sets. All the models and results that we will present below are evaluated using the splitting performed by the `timeslice` of the `caret` package. Nevertheless, at the beginning of the development of this project, there were also constructed two manual expanding window functions that split the data programmatically, by providing the proportion of data in the three sets or the number of observations in the validation and test sets. In order to follow the line of the report we will not explain these details here, but if you are interested you can look at this section in the R code and/or at the definition of the `expanding_window_prop` and `expanding_window_fixed` functions. We have not done the same for the sliding window as we immediately preferred the standard approach suggested in the packages.

Models

As previously mentioned, the `caret` package will be employed for model training, validation and eventually final test on the `test_set`. This package intrinsically enables the visualisation of different metrics, even if it is possible to specify one to ensure that the optimal model is identified in accordance with the expectations. Moreover, the `trainControl` parameter will be specified for all models as follows:

- `method`: it will always be `timeslice` as it is the correct one to be sure that the dataset is split according to the chronological order, which is a requirement
- `initialWindow`: number of rows in the training window at the beginning. In this case we are looking at a month, so for simplicity we will just set 744 hours
- `horizon`: it refers to the number of rows in the validation set, the horizon of the forecast. Since we will be predicting for the following day, this is just 24 hours
- `skip`: this parameter allow you to skip rows within the dataset. As we want to use the whole training dataset, this parameter will simply skip to the next consequent subset. The training dataset already does not contain the test part at the end
- `fixedWindow`: this parameter makes it possible to decide whether to use the sliding window approach (`TRUE`) or the expanding window one (`FALSE`)

Although we already know that the linear model is not complex enough to achieve the flexibility required by the model, it was decided to implement it in order to have a ground truth against which to compare values and performance. So, we defined the following function

```
## linear_model <- function(dataset, target, initial_window, horizon,
##                          window_type, in_parallel){
##
##   # Save start time for performance evaluation
##   time_start <- Sys.time()
##
##   # Use parallel computing, if desired
##   clusters <- makePSOCKcluster(in_parallel)
##   registerDoParallel(clusters)
##
##   # Define the current train control
##   current_train_control <- trainControl(method = 'timeslice',
##                                         initialWindow = initial_window,
##                                         horizon = horizon,
##                                         skip = horizon - 1,
##                                         fixedWindow = window_type)
##
##   # Fit and evaluate the linear model
##   # For technical reasons, the formula has been defined to avoid problems when
##   # specifying column names
##   formula <- as.formula(paste(target, '~ .'))
##   lm_model <- train(formula,
##                     method = 'lm',
##                     data = dataset,
##                     metric = 'RMSE',
##                     trControl = current_train_control)
##
##   # Stop parallel computing, if active
##   stopCluster(clusters)
##
```

```
##   # Save end time and evaluate the minutes required
##   time_end <- Sys.time()
##   total_time <- round(as.numeric(difftime(time_end, time_start, units = 'mins')), 2)
##
##   # Save and return the model and the time required
##   model_summary <- list(lm_model, total_time)
##   return(model_summary)
## }
```

and we decided to run it firstly with the expanding window approach and after with the sliding one. As you may have noticed in the declaration, it is also possible to find the use of the `doParallel` package [11]. We decided to add this feature as well and perform parallel computations with the idea of speeding up the training phase of the different models as much as possible. Of course, we did not find any extraordinary results with the linear model, but it helped in the validation phase and with other approaches. Please note that this parameter is highly dependent on your machine hence, if you want to run the code, we recommend that you set the `current_parallel` variable to a reasonable number for you. Anyway, coming back to the linear model, even if we decided to evaluate performances using the `RMSE` as metric, we decided to display also the `MAE` in order to give some more insight.

Table 12: Models summary

Model	Window	RMSE	MAE	Training time (min)
Linear Model	Expanding	8.765557	7.786211	0.75
Linear Model	Sliding	6.204765	5.315728	0.34

As one could imagine, the approach with the expanding window required more time, even if the linear model is quite fast. On the contrary, it is also possible to detect that this time was not used to better understand the trend behind the data, resulting also in a worst outcome. This phenomena could be explained by the fact that the linear model is too simple to gain flexibility and giving too many observation can be self-defeating. So, the next idea was to use the XG Boost model, as it can learn the different trends inside the series. Generally speaking, this approach constructs an ensemble of decision trees in a sequential manner, where each tree attempts to correct the errors of the preceding ones. The gradient descent represents a focal aspect of this process; however, there are also some hyperparameters that can be appropriately tuned in accordance with the data and specific patterns. Coming back to the source dataset, even if it is sometimes suggested to add lagged features, we decide to avoid this way, as explained before. Of course, it is possible to change the idea and compare the insertion of the temporal features with the lagged ones and verify which one gives a better result. For the moment, we declared the following function to execute the XG Boost model:

```
## xgboost_model <- function(dataset, target, initial_window, horizon,
##                           window_type, in_parallel){
##
##   # Save start time for performance evaluation
##   time_start <- Sys.time()
##
##   # Use parallel computing, if desired
##   clusters <- makePSOCKcluster(in_parallel)
##   registerDoParallel(clusters)
##
##   # Define the current train control
##   current_train_control <- trainControl(method = 'timeslice',
##                                         initialWindow = initial_window,
##                                         horizon = horizon,
##                                         skip = horizon - 1,
```

```
##                                         fixedWindow = window_type)
##
##   # Fit and evaluate the XG Boost model
##   # For technical reasons, the formula has been defined to avoid problems when
##   # specifying column names
##   formula <- as.formula(paste(target, '~ .'))
##   xg_boost_model <- train(formula,
##                           method = 'xgbTree',
##                           data = dataset,
##                           metric = 'RMSE',
##                           trControl = current_train_control)
##
##   # Stop parallel computing, if active
##   stopCluster(clusters)
##
##   # Save end time and evaluate the minutes required
##   time_end <- Sys.time()
##   total_time <- round(as.numeric(difftime(time_end, time_start, units = 'mins')), 2)
##
##   # Save and return the model and the time required
##   model_summary <- list(xg_boost_model, total_time)
##   return(model_summary)
## }
```

that results in the output below:

Table 13: Models summary

Model	Window	RMSE	MAE	Training time (min)
Linear Model	Expanding	8.765557	7.786211	0.75
Linear Model	Sliding	6.204765	5.315728	0.34
XG Boost	Sliding	5.151681	4.332737	22.35

Of course, it is possible to immediately notice that the XG Boost model takes more time than the linear one, however producing better results. Moreover, it is also possible to point out that this approach relies on some hyperparameters that were equal to:

Table 14: XG Boost hyperparameters

hyperparameter	value
nrounds	150.0
max_depth	3.0
eta	0.3
gamma	0.0
colsample_bytree	0.8
min_child_weight	1.0
subsample	1.0

in the current best model. If we better inspect the `xgboost_fixed` model, one can notice that by default the parameter `gamma` was set fixed to `0` and `min_child_weight` to `1`. Looking at [12] we recall that this means no regularisation and a possible risk of overfitting. However, it is the default setting for this well-known model so at this stage it was decided to keep this configuration. In addition, if you are not familiar with this model, we also would like to remember that `nrounds` refers to the number of rounds, `eta` controls the learning

rate, `max_depth` the maximum depth of a tree, `subsample` the subsample ratio of the training instances and `colsample_bytree` the subsample ratio of columns when constructing each tree. Although we are aware that it is also possible to tune these and other parameters, this activity was not performed systematically in this project due to performance issues, however it is an immediate and important improvement for future works. Following a similar idea, the next step was the implementation of the random forest model. This is an algorithm that constructs multiple decision trees in an independent manner, taking into account different subsets of data and features. After that, these trees are combined to generate predictions that will be evaluated to select the best ones. It is good to highlight that from a theoretical point of view this approach is quite good, as it reduces the risk of overfitting and increases the generalisation capabilities with different trees. So, with a similar framework of the previous models, the following function was defined:

```
## random_forest_model <- function(dataset, target, initial_window, horizon,
##                                  window_type, in_parallel){
##
##   # Save start time for performance evaluation
##   time_start <- Sys.time()
##
##   # Use parallel computing, if desired
##   clusters <- makePSOCKcluster(in_parallel)
##   registerDoParallel(clusters)
##
##   # Define the current train control
##   current_train_control <- trainControl(method = 'timeslice',
##                                         initialWindow = initial_window,
##                                         horizon = horizon,
##                                         skip = horizon - 1,
##                                         fixedWindow = window_type)
##
##   # Fit and evaluate the random forest model
##   # For technical reasons, the formula has been defined to avoid problems when
##   # specifying column names
##   formula <- as.formula(paste(target, '~ .'))
##   rf_model <- train(formula,
##                     method = 'rf',
##                     data = dataset,
##                     metric = 'RMSE',
##                     trControl = current_train_control)
##
##   # Stop parallel computing, if active
##   stopCluster(clusters)
##
##   # Save end time and evaluate the minutes required
##   time_end <- Sys.time()
##   total_time <- round(as.numeric(difftime(time_end, time_start, units = 'mins')), 2)
##
##   # Save and return the model and the time required
##   model_summary <- list(rf_model, total_time)
##   return(model_summary)
## }
```

and, also in this case it is possible to tune the hyperparameter to get better results [13]. In details, the `mtry` displays the number of features selected for splitting at each node of a decision tree.

Observing the table below, we can see that this model is slower than the others, while providing better results:

Table 15: Models summary

Model	Window	RMSE	MAE	Training time (min)
Linear Model	Expanding	8.765557	7.786211	0.75
Linear Model	Sliding	6.204765	5.315728	0.34
XG Boost	Sliding	5.151681	4.332737	22.35
Random Forest	Sliding	5.105898	4.310431	33.61

In this scenario, we can also point out that in the final model the `mtry` was equal to 16. Again, it is of course possible to tune the parameters inside, however this phase lasted a bit too long so we decided to do not report the attempts of fine tuning. As an extension, it is of course possible to create a structured framework to perform all these kind of tests, also for different models. In addition, we also investigated other models and approaches to properly deal with this dataset and we would like to report just two of them that are simple but noticeable extensions of what has already been presented here. The first model is the ARIMA [9], a well-known method that is particularly suitable for dealing with timeseries. Unfortunately, it does not take into account the additional features. We performed a more extensive search that revealed that the ARIMAX model is an optimal intermediate solution, as it is capable of handling both the timeseries information and the additional data present in the dataset. We believe that it could be a good intermediate step to understand how far this mixed approach can lead. At the other end of the spectre, a well-structured method of dealing with time series is LSTM [14]. The strength of this approach lies in the fact that it is able to store information over longer sequences, an important capability when we need to understand trends over intervals. It is capable of ignoring noise that produces unexpected behaviour that is not consistent with the general path. In general there are many different approaches in this area that should be investigated and properly analysed.

Results

From the Analysis and Models sections we can see that the best of the models presented can be selected between the XG Boost and the random forest. It was decided to take the random forest as the best one because the additional time required for the training phase is not prohibitive and, even though the `RMSE`s are similar, it was preferred to take the actual best at forecasting. This idea also follows the path that in a real-life scenario it is not necessary to train the model each time, whilst it is required to look at their predictions. So, we also decided to provide the metrics compared to the final and independent `test_set`. Naturally, as we are dealing with a timeseries, it is necessary to use chronological order and, as we also would like to simulate a real use of this model, we removed the last 14 days from the original dataset and stored it in the `test_set`. In this scenario, it is important to anticipate that the horizon is longer than the one used in the training phase, so we are attempting to use the `test_set` to verify results setting a different background. On the other hand, we would generally expect worse performances when compared to the validation sets, although this is not always the case, e.g. we may have casually picked an horizon that is very consistent with the general trend learned by the model, even if it is new and never seen before. Anyway, we compared the prediction and the final result is

Table 16: Independent test metrics

Model	RMSE	MAE
Random forest	3.238131	2.39686

The metrics above were evaluated simply taking into account the whole period of prediction. In addition, it is also possible to evaluate the `RMSE` in a way that mimic the validations performed during the training

phase, splitting each time in a similar manner. This step could be good to understand how the metrics vary through the `test_set`, maybe suggesting limitations and strengths of the current best model.

Moreover, inspecting the slices with spikes in the metrics, it could also be useful to help identify trends and patterns that may be hardly captured by the model. In this case we developed a manual function for `RMSE` and `MAE`, even if this could be expanded as desired:

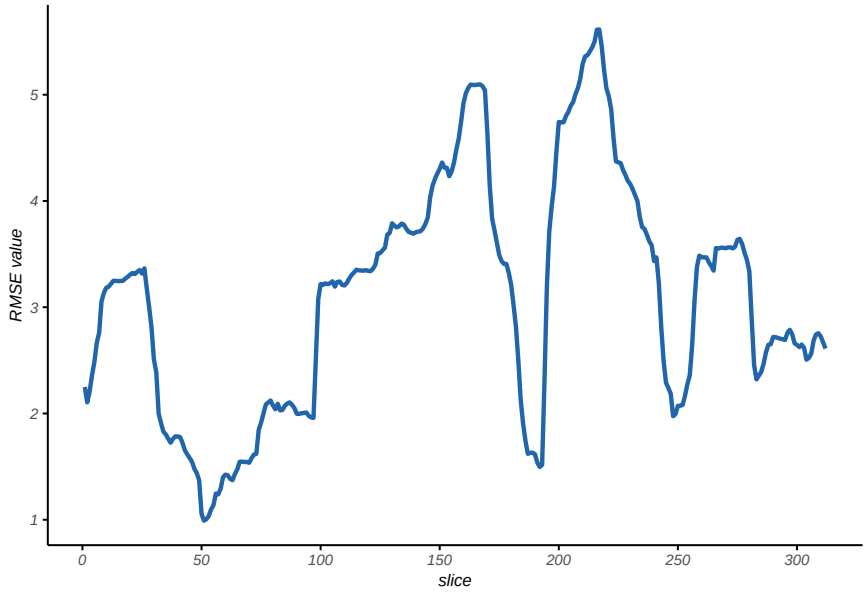
```
## test_set_performance_evaluation <- function(test_set,
##                                             target,
##                                             model_prediction,
##                                             validation_size){
##
##   # Isolate the target feature from the test set
##   test_y <- test_set %>% select(all_of(target))
##
##   # Merge predictions and test set to easily evaluate the metrics
##   final_outcomes <- as.data.frame(cbind(test_y, model_prediction))
##
##   # Define the dataframes to store the metrics
##   final_metrics <- data.frame()
##
##   # Initialise i for the while loop
##   i = 1
##   while(i <= nrow(final_outcomes) - validation_size){
##
##     # Define the current slice
##     # The function dplyr::slice was not used as it was not straightforward
##     # at this stage of the evaluations
##     current_slice <- final_outcomes %>%
##                      head(i + validation_size - 1) %>%
##                      tail(validation_size)
##
##     # Evaluate the metrics on the current slice
##     current_RMSE <- RMSE(current_slice[[1]], current_slice$model_prediction)
##     current_MAE <- MAE(current_slice[[1]], current_slice$model_prediction)
##
##     # Save the results in the final dataset
##     final_metrics <- rbind(final_metrics,
##                            data.frame(slice = i,
##                                       RMSE = current_RMSE,
##                                       MAE = current_MAE))
##     # Update the loop parameter for the next iteration
##     i <- i + 1
##   }
##
##   # Return the results
##   return(final_metrics)
##
## }
```

As you can see below, the actual values depicted are quite different with respect of the previous approach, but still consistent with the general discussion:

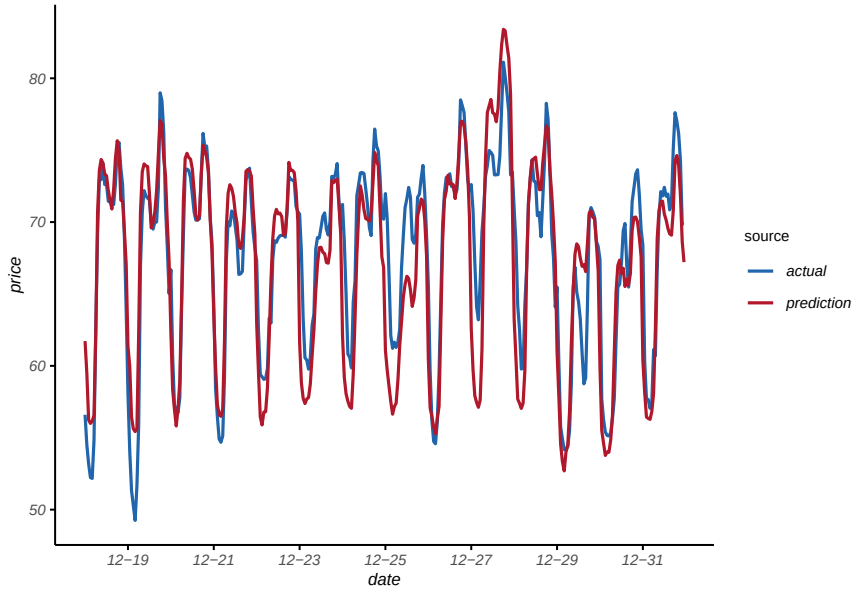
Table 17: Test metrics summary

metric	statistic	value
RMSE	min	0.9918617
RMSE	max	5.6139672
RMSE	mean	3.0940600
RMSE	median	3.2278794
RMSE	sd	1.1047090
MAE	min	0.7957412
MAE	max	4.9382794
MAE	mean	2.4176905
MAE	median	2.3358038
MAE	sd	0.9087300

After that, it was also constructed a general plot that clearly reports the fluctuations of the `RMSE` through the `test_set`:



In addition, we also decided to plot the predictions against the actual values in the `test_set`, in order to have a visual insight on the values and try to extrapolate the edge cases that the model may need to properly tune:



As usual, there is room for improvement, nevertheless it is possible to see that some predictions are very accurate, while others are a little off.

Conclusions

This report presents an analysis of price forecasting in the energy market. All the activities started from two different separate datasets that needed some cleaning steps. So, after a brief summary of the main features, we explored the details related to each specific column, pointing out some anomalies and correcting them as much as possible. The data was then manipulated to give it a good shape that could be used directly in the next steps. Before moving on to machine learning and statistical approaches, we also decided to perform a seasonality analysis to inspect the core entities underlying the timeseries. Regarding the models, we started with a linear one, passing through an XG Boost and resulting in a random forest; each one with different points of strength. At the end, we tested the data on the `test_set` for the final model and it gives an acceptable result, although it could be improved. In addition to the model improvements already mentioned in the Models section, it may be a good idea to perform a PCA analysis to better understand which features we need to take into account, or to perform a complete cross-correlation analysis also related to the time variable. It might also be a good idea to better manipulate the source data to refine it, both in terms of the number of locations and the sensitivity of the values. Furthermore, if we think about historical periods like Covid-19, where the price of energy increases unexpectedly, it might suggest that a good improvement could be to add also some social or demographic information. It is reasonable to suppose that the incorporation of these models and features represents an enhancement under the point of view of reliability and precision. However, it is important to consider the impact on performance, as it is evident that performance is a crucial aspect since the execution of these operations results in costs, both in terms of hardware resources and time. In a real-life application, a perfect model that requires more than one day to run is useless and everyone would rather prefer a less accurate one. This is an extreme case, but it has been highlighted to point out that we can not simply assume that purchasing a more powerful hardware would be enough. Switching to a different however related field of research, it is also possible to manipulate this data in a diverse manner and use it to predict features such as the total load or the energy generation for a renewable park, perhaps adding some specific feature. In conclusion, it can be surely stated that this is a rich topic that could provide interesting results that may also affect our lives, of course if properly analysed with the correct methodologies and technologies.

Notes

In this project, the impersonal form 'we' was employed. However, it should be noted that the project was developed independently, without external interference. As explained during the course and highlighted in [15] and [16], it is important to visualise data in the most readable manner for everyone. So, all the graphical elements in this project are created using colorblind safe palettes, like the RdBu palette, or suggested ones, like Paired palette. In addition, all analyses were performed with a laptop with 8GB RAM and i5-1135G7 CPU so they took quite a long time and some other analyses would have been prohibitive. However, if your machine is better than mine, you can run multiple tests on different models and/or windows functions. Otherwise, if your machine is not that powerful, I would like to inform you that running the whole project could take more than 2 hours, so it might be helpful to execute only the R file, so that you can run only the interesting parts, avoiding useless computations and allowing flexibility in testing.

Bibliography

[1] M. Kuhn, "The caret package." Accessed: Nov. 01, 2024. [Online]. Available: https://topepo.github.io/caret/index.html

[2] O. Calzone, "MAE, MSE, RMSE, and F1 score in time series forecasting." Accessed: Nov. 01, 2024. [Online]. Available: https://medium.com/@ottaviocalzone/mae-mse-rmse-and-f1-score-in-time-series-forecasting-d04021ffa7ce

[3] A. Zeileis, G. Grothendieck, J. A. Ryan, J. M. Ulrich, and F. Andrews, *Package zoo: S3 infrastructure for regular and irregular time series (z's ordered observations)*. 2023-04-13.

[4] T. Wei and V. Simko, "An introduction to corrplot package." Accessed: Oct. 25, 2024. [Online]. Available: https://cran.r-project.org/web/packages/corrplot/vignettes/corrplot-intro.html

[5] "List of extreme temperatures in spain." Accessed: Nov. 10, 2024. [Online]. Available: https://en.wikipedia.org/wiki/List_of_extreme_temperatures_in_Spain

[6] "List of atmospheric pressure records in europe." Accessed: Nov. 10, 2024. [Online]. Available: https://en.wikipedia.org/wiki/List_of_atmospheric_pressure_records_in_Europe#Iberia

[7] "Rain rate intensity classification." Accessed: Nov. 01, 2024. [Online]. Available: https://www.baranidesign.com/faq-articles/2020/1/19/rain-rate-intensity-classification

[8] "Unusual snowfall in madrid in january." Accessed: Nov. 01, 2024. [Online]. Available: https://www.ecmwf.int/en/newsletter/167/news/unusual-snowfall-madrid-january

[9] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and practice*. Melbourne, Australia: OTexts, 2018. Available: https://OTexts.com/fpp2

[10] A. Kis, "Understanding autocorrelation and partial autocorrelation functions (ACF and PACF)." Accessed: Oct. 10, 2024. [Online]. Available: https://medium.com/@kis.andras.nandor/understanding-autocorrelation-and-partial-autocorrelation-functions-acf-and-pacf-2998e7e1bcb5

[11] S. Weston and R. Calaway, "Getting started with doParallel and foreach." Accessed: Nov. 17, 2024. [Online]. Available: https://cran.r-project.org/web/packages/doParallel/vignettes/gettingstartedParallel.pdf

[12] J. Yuan, "Package xgboost." Accessed: Nov. 02, 2024. [Online]. Available: https://cran.r-project.org/web/packages/xgboost/xgboost.pdf

[13] B. Boehmke and B. Greenwell, *Hands-on machine learning with r*. Chapman; Hall/CRC, 2020-02-01. Available: https://bradleyboehmke.github.io/HOML/

[14] K. P. Ranjit and Y. Md, "Long short term memory (LSTM) model for time series forecasting." Accessed: Oct. 05, 2024. [Online]. Available: https://cran.r-project.org/web/packages/TSLSTM/TSLSTM.pdf

[15] R. A. Irizarry, *Introduction to data science*. Chapman; Hall/CRC, 2019.

[16] B. Rudis, N. Ross, and S. Garnier, "Introduction to the viridis color maps." Accessed: Sep. 29, 2024. [Online]. Available: https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html#the-color-scales