



Anticipate the crowd at SNCF train station of Ile de France

Sean SCWHAGER
Kamesh VENKATA
Constance FROMONOT



Introduction

I- Project's presentation

1) Context

SNCF-Transilien is the commuter train operator in Île-de-France. They operate more than 6,200 trains enabling 3.2 million passengers to travel. These travelers validate their smart cards on our gates an average of 2.3 million times per day. Over time it has been observed that the number of passengers generally increases, and as such better anticipation of this increase will help SNCF offer more appropriate services and improve the performance of their operations. Thus, the overall objective for SNCF-Transilien is to be able to better anticipate the number of validations at each station, as a proxy for the volumes of passengers to be absorbed in the years to come.

a) Technical context

- Data Collection and management:
 - SNCF-Transilien collects validation data from 448 stations in the Île-de-France region, with data points recorded daily. We are talking about very concrete data.
- Technical Expertise:
 - Involves time-series analysis, machine learning, and deep learning
 - Data scientists and engineers will need to preprocess data, handle missing values, and possibly engineer new features.
 - Expertise in programming

b) Economic context

Economically speaking, this project is crucial for SNCF for different reasons:

- Resource allocation:
 - Efficiently predicting passenger numbers will enable better allocation of resources to stations (avoid overstaffing in low frequented stations and understaffing in high frequented stations).
 - It helps optimize operational costs by reducing underutilization of trains and overcrowding.
- Revenue optimisation:
 - Understanding and anticipating passenger flows can lead to improved ticket sales strategies, pricing models, and marketing campaigns.
 - Better service reliability and reduced congestion can enhance passenger satisfaction, potentially increasing ridership.
- Cost management:
 - Reducing operational inefficiencies can lead to significant cost savings.
 - Predictive maintenance scheduling based on passenger load could reduce downtime and maintenance costs, and so enhance client satisfaction.
- Investment decisions:

- Data-driven insights can guide infrastructure investments and upgrades.
- Helps in making informed decisions regarding expansion projects or enhancements to existing services.

c) Scientific context

- This project contributes to the field of time-series forecasting, particularly in the transportation sector.
- Feature engineering:
 - Utilizing contextual variables like job days, holidays, and school vacations to improve model accuracy.
 - Exploring other exogenous variables like weather, special events, and economic indicators.

2) Objectives

Given the aforementioned context and initial overview of our dataset, our high-level objective for this project can be stated as follows:

To predict, for a certain period, the number of validations per day and per station.

In fulfilling this objective, certain key steps must be highlighted:

- We will need to deal with “dates” and “station” which are the necessary features, and the number of validations which will be our target variable.
- We have existing exogenous variables provided by the initial dataset (weekend or not, bank holidays, school holidays).
- We will enhance our dataset with other data like detail on time (eg: year, month, day, weekdays), weather (eg: temperature, precipitation).
- We also need to impute values due to the nan and outliers present in the dataset.
- We are planning to test different types of models: time series, machine learning, and deep learning.
- Across all of the work steps, we will need to pay attention to the time consumed by the training and prediction.
- Finally, we are going to perform hyper parameter tuning to find the best models to predict the number of validation per day over a certain period.

Chapter 1: Exploratory analysis & pre processing

I- Presentation of the dataset

Our data comes from the SNCF, which has put those data online freely for a challenge.
Here is the typology of the original dataset :

Name of variable	Description	Format	Type
Date	The date of the day the validations were recorded in YYYY-MM-DD format	Datetime	Feature
Station	Station identifier anonymized in 3 characters (eg :7RP)	String	Feature
Job	Indicator worth 1 if the day is a JOB (Basic Working Day), that is to say a Monday, Tuesday, Wednesday, Thursday or Friday, 0 otherwise	Integer	Feature
Ferie	Indicator worth 1 if the day is a public holiday, 0 otherwise	Integer	Feature
Vacances	Indicator worth 1 if the day corresponds to a school vacation day, 0 otherwise	Integer	Feature
Number of validations	Number of validations	Integer	Target

Find below the detail regarding the specificities of the variables:

- **1 229 863 rows** and **6 columns** in the dataset
- **439 stations** (which have been anonymised)
- The available data are between **1st January 2015 and 31st December 2022** (2922 days).
- The period for which we need to predict the number of validations is between **1st January 2023 and 30th June 2023** (181 days).
- No null values in the dataset
- Job/ferie/vacances will have the same value for every stations each days
- The number of validations which will be our target variable.
- "dates" and "station" are the necessary features to our goal of prediction. The other variables add more information but are not mandatory
- Lastly, we still need to mention that Time is not available in the time series. Thus, there is no notion of peak hours during certain days, for which the stations need to have much more staff than other hours.

II- Pre-processing and feature engineering

1) Adding new variables

We have started by adding new variables on:

- **time** : year, month, day, weekday (made directly from the dataset) = 4 variables
- **lag** : lag_1 corresponding to the number of validation on the previous day. We add 8 days of lag variables = 8 variables
- **weather** : precipitation, snow, wind speed, wind direction... etc (using the meteostat python library) = 7 variables
- **economic indicators** : unemployment, consumer expenses, consumer confidence, GDP, transport growth... ect (from the INSEE, the french National Institute of Statistics and Economic Studies) = 57 variables

2) Missing variable and outliers handling

On first inspection, the dataset does not appear to have any nan values present. However, when pivoting the table it makes apparent the incomplete data within the time series: not every station has a value for every day. This was somewhat disguised when the stations were laid out on top of one another, but it becomes very evident in the pivot table. We have elected to call such instances “**maintenance/service days**” (when the station shows no valid data for that day, presumably related to a station shutdown as a result of some technical fault or planned maintenance). It is therefore essentially to come up with a strategy for how to handle these values.

What constitutes these days are also subject to some interpretation, as they can take several forms, outputting either: nan values, zero values, or values that are extreme outliers (e.g. a station going from 20'000 to 5). It is thus insufficient to flag them based purely on identifying easily identifiable nan and zero values, and as such an outlier detecting strategy must be implemented.

There are a few key considerations here:

1. We are only interested in **low value outliers** (not high value ones, as these high extreme values are entirely reasonable given the size of certain stations and the sudden influx that these stations see during high volume periods).
2. We are interested in **sudden drops**, rather than values which are outliers in the general sense (relative to the entire time series), as such a **rolling method** would be most appropriate.

As such, we have elected to use a **rolling median** method by defining both a **threshold** and **window size** variable. This method is designed to flag these sudden drops, and output a mask identifying them. This mask is then used to set all maintenance day values to nan, in order to allow imputation methods. The use of variables in the construction of the rolling median detection method also has the added benefit of providing two new ‘hyperparameters’ which can be optimized via a custom gridsearch

later on, leading to a missing value detection strategy which is robust, evidence based and tailored to our specific approach.

Once these days are identified, there is still the question of how to handle them. We view this approach as being different for the test and training sets.

- **Training set:** to ensure consistency of the data used to train the model, we have elected to impute these values using a **Piecewise Cubic Hermite Interpolating Polynomial (PCHIP)**. This was determined to be the interpolation method which produced the best results (tested against other methods such as linear, quadratic, spline and akima).
- **Test set:** here the primary concern is how these unpredictable days impact the overall MAPE (discussed further in the modeling chapter). These values do not need to be imputed, but rather accounted for in order to get an accurate sense of the model's performance.

3) Isolating some stations

During the imputation phase, we discovered that some stations opened later than most others, meaning they were not all operational from the beginning of 2015. Consequently, we have less data for these stations making training harder, with some only opening as late as 2022 - our initially designated "test dataset" for model evaluation.

As a result, we cannot treat them like the other stations. We have thus designated two broad groups of these anomaly stations which each require specific treatment:

- The 6 stations **starting sometime in 2022** are deemed not to be predictable with any high degree of accuracy, since we would be trying to predict an entire year (2023) based off of a training and test set that together do not even make up a full year's worth of data. Thus, we will **exclude** them from the analysis. To this group we are also explicitly adding the station **TV1**. While this station is open from 01/01/2015, it has very little valid data present in 2022 (the designated testing window for the bulk of the stations), and as such it provides substantial challenges for making predictions for 2023 (the ultimate goal of the project) and will also be excluded.
- The 6 stations starting **between 2015-01-31 and 2021-12-31** will be handled independently, with their own uniquely defined training and test sets that are based off of the available data.

	Opening date
target_48S	2017-07-02
target_BDC	2022-07-01
target_GW7	2015-11-16
target_N9K	2017-07-01
target_OWM	2017-01-17
target_P6E	2022-07-01
target_QD6	2022-08-02
target_RF2	2015-03-15
target_V2P	2017-06-06
target_W14	2022-07-02
target_W80	2022-07-15
target_TV1	2015-01-01

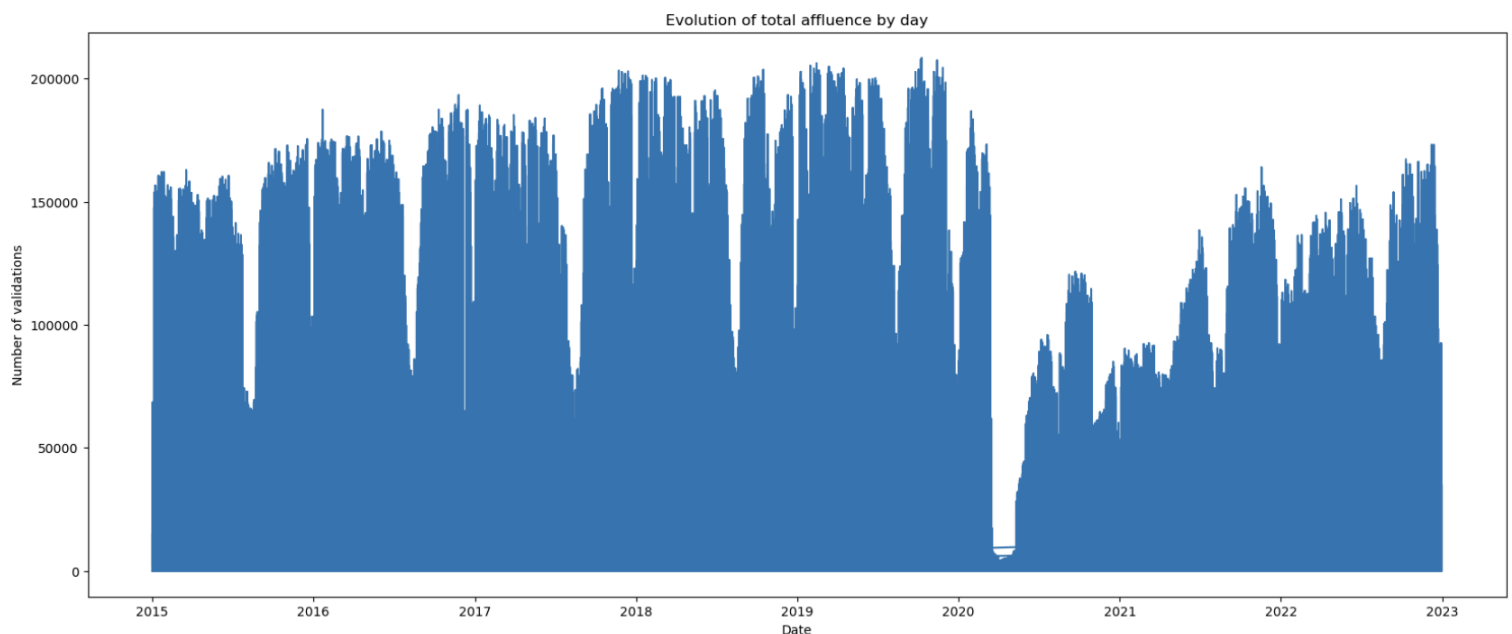
So at this stage, we have a dataset with **2922 rows and 427 columns** containing most of the stations, and another dataset containing the remaining of the stations to predict

individually (6 columns). We will now see what our time series looks like, and select the important features to predict our target variable.

III- Graphical representation of the dataset

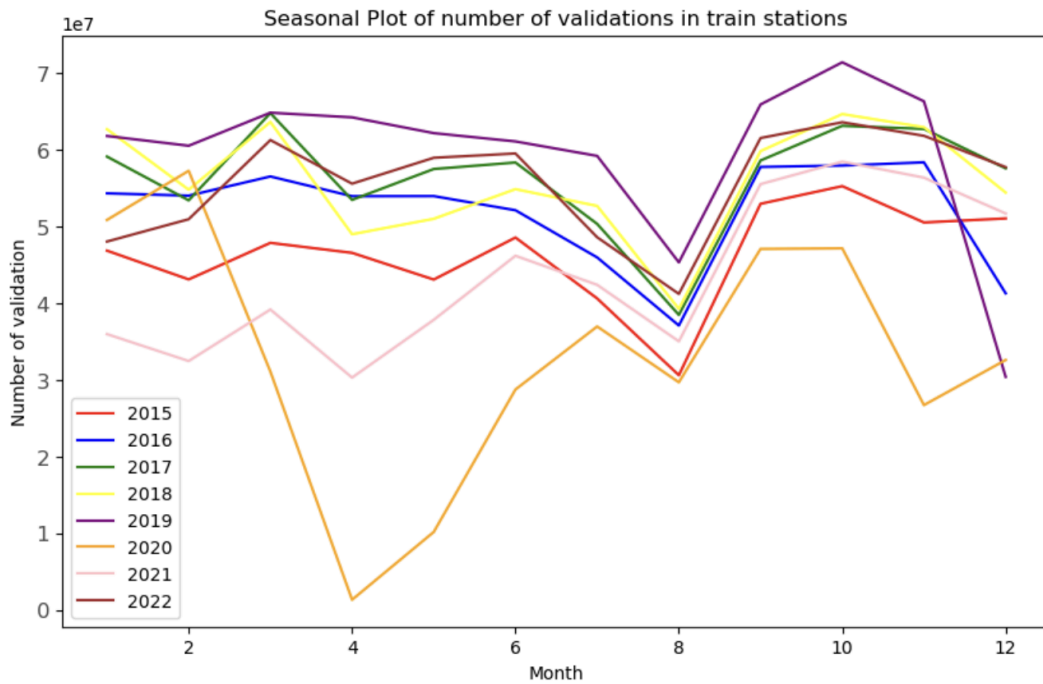
1) Graphical representation 1

This graph presents a simple time series plot showing the daily number of validations across Ile-de-France from 2015 to 2022. It illustrates that the number of validations increased by approximately 6% per year from 2015 to 2019, and continued to rise from 2021 to 2023. However, during the period between these two growth phases, the COVID-19 pandemic caused a significant drop in the number of validations in 2020, and the levels of daily validations have not returned to their previous highs. From a business perspective, it is crucial to anticipate these growth trends while accounting for the unusual drop in 2020, which was an anomaly that could not have been predicted.



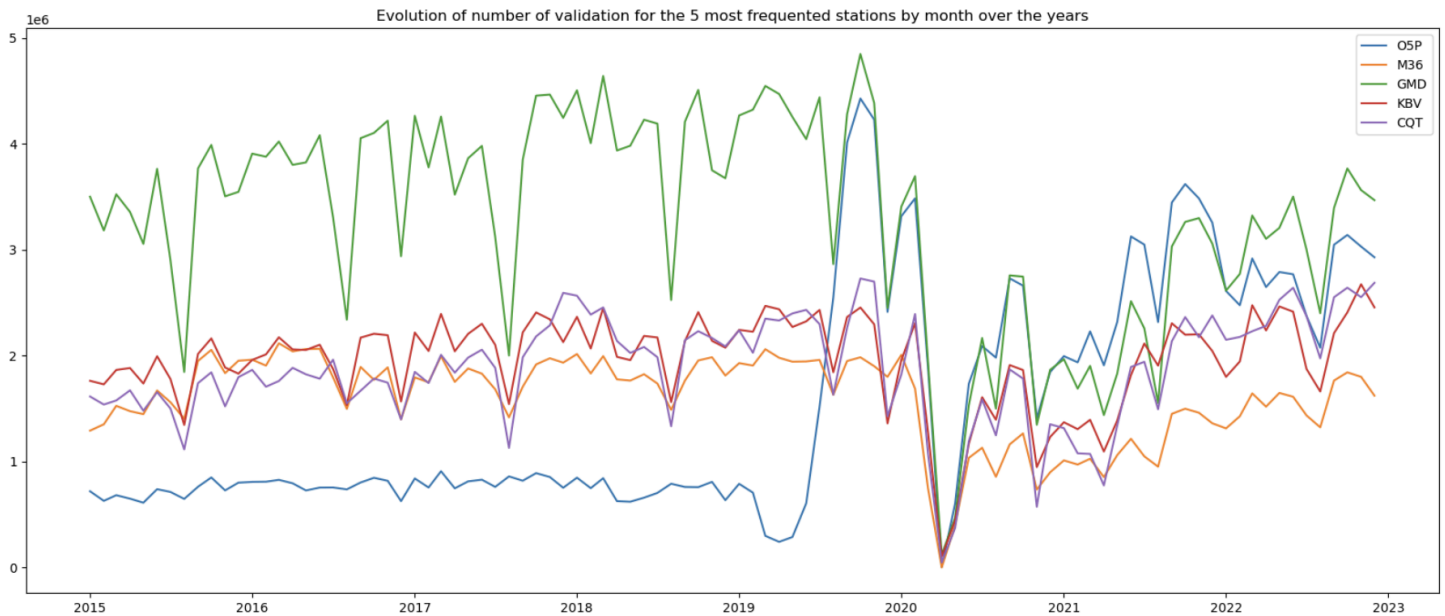
2) Graphical representation 2

Examining this graph reveals a clear seasonal pattern in our time series. We see a relatively steady trend from January to June, a notable drop during the summer with the lowest point in August, followed by a rise in September and a peak in October. There is then a slight decline towards the end of the year. Additionally, the graph highlights a significant drop from March 2020 to July 2020 due to the COVID-19 pandemic. As a result, 2020 and early 2021 stand out as the years with the lowest validation levels between 2015 and 2022.



3) Graphical representation 3

The 5 most frequented stations represent 20% of the total number of validations between 2015 and 2022. Plotting them, we obtain :



This graph illustrates the monthly variations in the top 5 most frequented stations, showing that when there is a drop in one station, it is generally observed across all stations. Notably, at the end of 2019, there is a peak in validation numbers for every

station. However, station O5P, which was previously the least frequented, saw its validation numbers rise to match those of the highest station, GMD. Since then, O5P has consistently had a higher number of validations per month. This suggests that significant infrastructure improvements were likely completed, greatly increasing its capacity.

4) Graphical representation 4

Here we have a graph of combined boxplots which allows us to compare the number of validations against various categorical variables (work_day, holiday, weekday, and school holiday). Each boxplot demonstrates the distribution of the number of validations based on the categorical values of these variables.

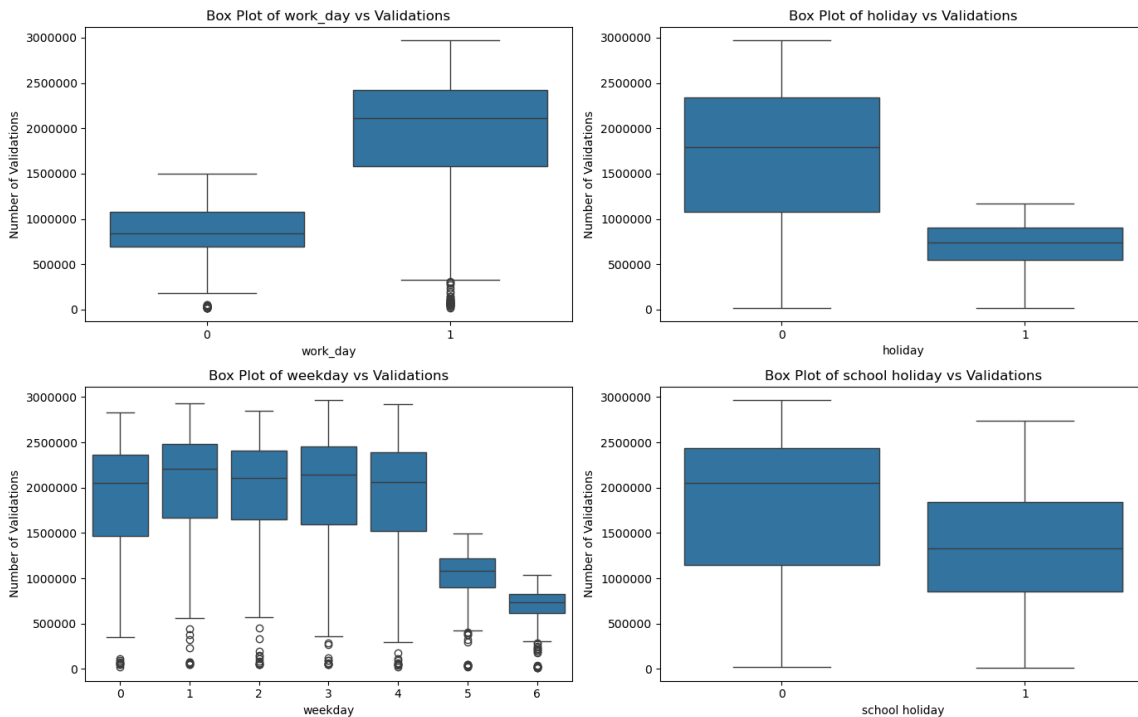
Work_day vs Validations: The subplot shows us that workdays (1) have a higher median number of validations compared to non-workdays (0). Additionally, the interquartile range and the overall spread of validations are larger on workdays, indicating higher and more varied ridership during these days.

Holiday vs Validations: Here we can see that the number of validations tends to be lower on holidays (1) compared to non-holidays (0). The boxplot reveals a noticeable decrease in median validations during holidays, which aligns with expected lower commuter traffic on such days.

Weekday vs Validations: This subplot compares validations across different weekdays (0 to 6, where 0 is Monday and 6 is Sunday). It highlights that weekdays generally have higher validations than weekends, with the median and distribution peaking midweek (Tuesday to Thursday) and dipping towards the weekend.

School Holiday vs Validations: The subplot illustrates that school holidays (1) typically experience fewer validations compared to non-school holidays (0). This is indicative of reduced commuter traffic during periods when schools are on holiday.

Boxplots for categorical variables vs validations



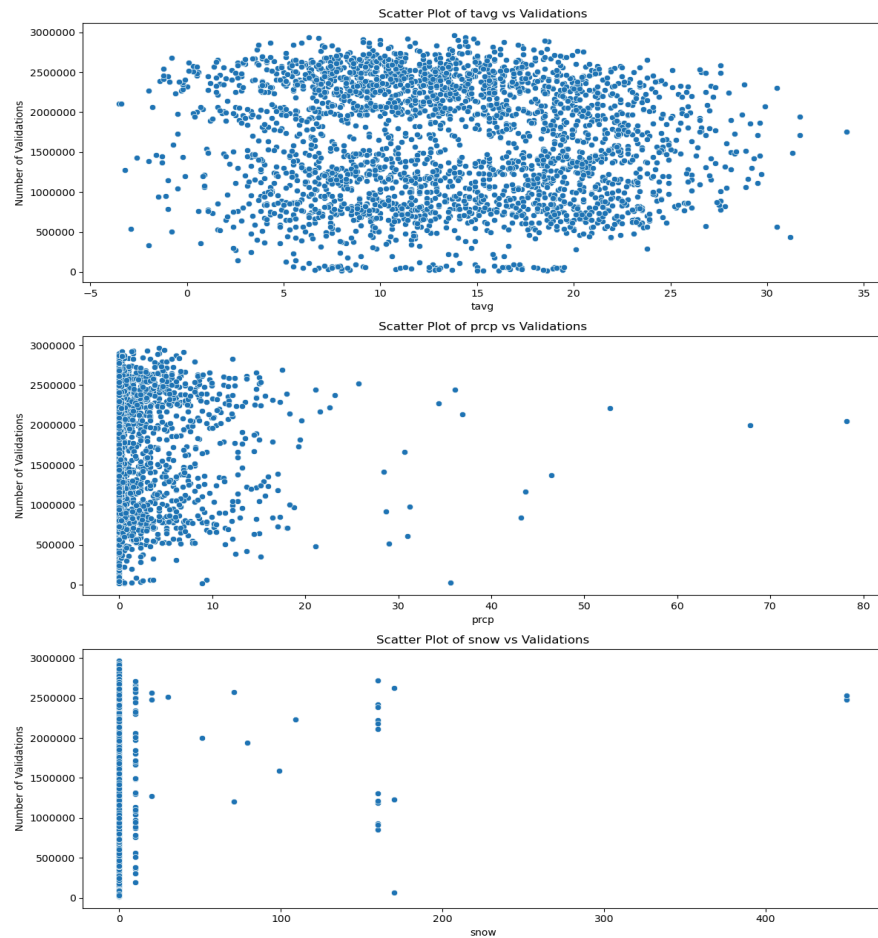
5) Graphical representation 5

The next 3 scatterplots provide examinations of the relationship between the number of validations and three weather-related variables: average temperature (tavg), precipitation (prcp), and snowfall (snow).

Scatter Plot of tavg vs Validations: This scatterplot reveals that while there is no clear linear trend, there is a slight increase in validations with moderate temperatures (around 10-20°C). Extremely high or low temperatures seem to correspond to fewer validations, possibly due to less favorable commuting conditions.

Scatter Plot of prcp vs Validations: This subplot plot suggests that most validations occur when there is little to no precipitation, indicating that heavy rain may discourage commuting. However, some validations still occur at higher precipitation levels, reflecting varied commuter resilience to weather conditions.

Scatter Plot of snow vs Validations: In this subplot we see that most validations are concentrated at zero snowfall, which is expected as snow events are less frequent. There are some validations at higher snowfall amounts, but the overall trend shows a decrease in validations as snowfall increases, likely due to the challenges and disruptions caused by heavy snow.



IV- Select most important features

Now that we have an exhaustive list of complementary variables into a single dataset, we need to make sure which of them are useful for our study. To ease our study, we are assuming that all stations are varying in the same way (see graphical representation 3 for an example) and that they have the same pattern “on the long term”. Thus, we can aggregate all the stations together and have a total number of validations per day between 2015 and 2020.

1) Statistical tests

We started by performing statistical tests to see if there was any significant effect of one of the features (quantitative or qualitative) on the target variable (quantitative). The process is the same for every feature. We start by separating into quantitative and qualitative list of variables, then we can perform the tests:

a) Test between 2 quantitative variables

To check if the 2 quantitative variables are correlated, we can use the Pearson test if the following conditions are respected :

1. Linear relationship between the feature and the target variable.

2. Both variables are approximately normally distributed
3. No presence of outliers in the data

Let's verify all of them.

Linearity condition

To check this condition, we can use the Pearson correlation coefficient. Its value varies between -1 and 1. A value close to 1 or -1 indicates a strong linear relationship (positive or negative, respectively). A value close to 0 indicates no linear relationship. Our goal is to predict y from our features. Then, our tolerance threshold can't be low. Let's set it at 0.5. If any variables have a correlation coefficient lower than 0.5, then the linearity between any feature and the target variable y isn't respected.

We apply this on all our quantitative variables, and none of them have a coefficient higher than 0.5. We conclude that the relationship between the features and the target variable may be more complex and non-linear. Techniques that rely heavily on linear relationships (like linear regression) may not perform well in predicting the target variable. Instead, non-linear models like decision trees, boosting algorithms (e.g., XGBoost), or deep learning models may be more suitable to capture the patterns in the data.

Normality condition

No need to check since the 1st condition isn't met.

Outlier condition

No need to check since the 1st condition isn't met.

As there is no linear relationship between any feature and the target variable y , then we are using the Spearman test. We set our hypothesis:

$$\begin{cases} H_0: \text{The variables X and Y are uncorrelated} \\ H_1: \text{The variables X and Y are correlated} \end{cases}$$

When $p_value < 0.05$, we reject H_0 at level $\alpha = 0.05$ and accept H_1 . We therefore conclude that the variable X and Y are correlated, so we keep the feature in our study to predict our target variable.

For example, the intention to buy a car is significantly correlated to the number of validations, because the $p_value = 0.001$. But the $wind_speed$ is not because it has a $p_value = 0.687$.

b) Test between 1 qualitative and 1 quantitative variable

To check if the 1 qualitative and 1 quantitative variable are correlated, we can use the ANOVA test if the following conditions are respected:

1. The observations should be independent of each other.
2. The residuals (differences between observed and predicted values) should be approximately normally distributed.

3. The variances of the different groups should be approximately equal.
Let's verify all of them.

Independance

Our target variable is independent of any of our categorical features.

Normality of the residual

This can be checked with a Shapiro-Wilk test on the residual. We set our hypothesis:

{ H0: The population is normally distributed
{ H1: The population is NOT normally distributed

When $p_value < 0.05$, we reject H0 at level $\alpha = 0.05$ and accept H1. We therefore conclude that the residuals are not normally distributed.

As none of the variables have normally distributed residuals, then we can't use ANOVA.

Equality of variances

No need to check since the 2nd condition isn't met.

As none of the variables have not a normally distributed residual, then we will use the Kruskal Wallis test, which is a non-parametric equivalent to the ANOVA.
We set our hypothesis:

{ H0: There is no significant difference between the groups
{ H1: There is a significant difference between the groups


When $p_value < 0.05$, we reject H0 at level $\alpha = 0.05$ and accept H1. We therefore conclude that there is a significant difference between the groups, so between the categorical variable (the feature) and the continuous variable (the target). So we keep the feature in our study for prediction.

For example, the precipitation has a significant impact on the number of validations, because the $p_value = 0.004$. But the snow doesn't because it has a $p_value = 0.801$.

2) Dropping duplicated features

On the variables left, we can now drop the "duplicated" features. We had for example a raw indicator, and the same seasonally adjusted. If both were kept individually by the testing phase, then we remove one of them with common sense to avoid having the same information twice.

3) Correlated pairs



Finally, if some features are highly correlated (correlation between 2 features X and $Y > 0.8$ or correlation between 2 features X and $Y < -0.8$), we choose to keep the most accurate one : the feature which has the highest R^2 with respect to y .

In the end, we reduced the dataset to the columns: date, stations, number of validation, and 23 features, down from the original 80. We are now ready to begin the modeling phase.

Chapter 2 : Baseline modeling

I. Classification of the problem

Our problem is time series prediction, which falls under supervised forecasting using regression techniques. Forecasting involves predicting future values based on previously observed data, and it is widely used to anticipate trends, behaviors, or outcomes in various domains.

To evaluate our model, we will use the Mean Absolute Percentage Error (MAPE) on year 2022:

$$MAPE = \frac{1}{n} \sum_{i=1}^n \frac{|y_i - \hat{y}_i|}{y_i}$$

II. Model choices

1) Baseline prediction

As a baseline, we used the data from 2021, transposed to 2022, ensuring that the days are aligned. For example, January 1, 2022, which is a Saturday, corresponds to January 2, 2021, which is a Saturday as well.

We then calculated the Mean Absolute Percentage Error (MAPE) of this "random guess." Our goal now is to improve upon this baseline using more sophisticated models.

MAPE excluding anomalies baseline model = 1.6×10^{11}

2) Time series modeling

We have started by using basic time series modeling. This is usually where we can start to better understand the components of our time series (trend, seasonality, noises). The goal here was to use ARMA, ARIMA, SARIMA, SARIMAX models according to the specific needs of our time series. These types of models are implemented such that they fit and predict time series one by one individually. The advantage is that we can fully optimize the hyperparameters p, d, q, P, D, Q, k to stick to the time series as much as possible, and we can use the exogenous features to help the prediction.

But there are 2 main limits of doing so:

- The model won't be able to capture dependencies between the stations.
- The hyperparameters (p, d, q, P, D, Q, k) are not the same for each time series. Thus, we would have to adapt them for each of the 439 time series that we have, which is not the best solution when a model needs to be industrialized.

Also, the performance of ARIMA-like models deteriorates significantly when the parameters are not selected properly, so they are not suitable for time series data with weak periodic characteristics, as we have in our dataset. Moreover, we would have to find the best hyperparameters for each of the time series, which would be too inefficient. So we did not keep this solution.

3) Machine Learning modeling

We transitioned to a machine learning approach using boosting algorithms like XGBoost and LightGBM for our time series forecasting. These algorithms are well-suited for this task because they are good at capturing complex patterns and non-linear relationships, allowing us to incorporate exogenous features effectively.

We considered two approaches:

- Fitting and predicting on individual time series,
- Fitting and predicting on all time series at once.

For both approaches, we split the data into a training set (2015-2021) and a test set (2022). We normalized the exogenous features and added lag variables, which represent previous days' values and could be optimized for better performance.

In the first approach, we applied the regressor individually to each station by looping over them. However, this was extremely time-consuming because of the number of stations to be predicted, taking 3-5 days to predict for all of them. Due to this, we couldn't apply GridSearchCV for model optimization.

In the second approach, we one-hot encoded the stations, which added over 400 columns to the dataset. Applying the regressor to this larger dataset led to memory issues, making GridSearchCV unfeasible. Without model optimization, the results were disappointing:

MAPE excluding anomalies baseline model = 1.3×10^{17}

The MAPE was extremely high, indicating that the predictions were much worse than those from a baseline model.

4) Adjustment during modeling phase

After this first modeling phase, we have decided to not use all the exogenous features that we have selected before for the following reasons:

- **Technical reason:** Having a model able to fit on 7 years of data for 430 stations, and 33 exogenous features requires too much compute power.
- **Philosophical reason:** If we want to predict the future, we won't have access to economic indicators for the upcoming year, as these indicators typically take several months to finalize after the period has ended. While we could train the model using exogenous features, it wouldn't be feasible to predict future data (such as forecasting 2025 while still at the end of 2024).

For these different reasons, we preferred to remove the economic indicators of our analysis. From that, we will be developing our new approach.

Chapter 3 : Final modeling

As discussed earlier, we realized at this stage that using **all** the exogenous features was not appropriate, both philosophically and practically, so we removed the economic indicators, keeping only the temporal data, weather data, and lag variables. By analyzing the patterns within this data, we aim to predict future values more effectively, avoiding the different issues that we encountered before.

I. New preprocessing

We began by redoing the preprocessing. While we retained certain elements from the previous preprocessing, such as imputation, we removed other aspects like the economic indicators. The 6 time series that opened later and must be treated independently remain unchanged.

1) Pivot table

For the following approaches, the data needs to be structured such that the time series appear in columns side-by-side (rather than on top of one another). This has the added benefit of drastically reducing the number of rows, while also incidentally reducing the number of features (as we do not need to use a method such as one-hot-encoding to differentiate the stations). And thanks to the imputation developed in chapter 1, there are no nan values within the dataset and all stations have data everydays.

II. Evaluation method

Our problem remains a supervised forecasting task, and we are still using MAPE as our evaluation metric. However, this time, we are introducing an additional step to provide a more accurate reflection of our prediction results.

As covered in Chapter 1, we imputed missing values in the dataset. To determine where to apply this imputation, we created a mask that accounts for both missing values (NaNs) and outliers, such as sudden drops (using the rolling median detection method described previously). This imputation is crucial in providing for accurate training of the models, however a similar approach can also ensure that the test set is evaluated appropriately.

In essence, we cannot evaluate data that is unreliable. If a station has a maintenance day within the test set then it wouldn't matter how accurate the prediction was, this will always yield an extremely high MAPE that throws off the overall MAPE, thus clouding our ability to accurately interpret the performance of the models. Therefore, we will use the same mask developed during the imputation step, and apply it during the model performance evaluation to exclude these outliers (zero values, nan values and sudden

drops). As a result the MAPE will be calculated only on values that are not part of the mask, i.e. valid values that are actually predictable.

III. Model choice and optimization

1) SKforecast machine learning modeling, multi-series prediction

a) Presentation of the model

Our model is based on the strategy of **recursive multi-step forecasting**. This involves using the predicted values from previous time steps as input to forecast the values for the subsequent time steps. This approach is easily facilitated by **Skforecast**, a Python library built using the scikit-learn API that provides a comprehensive set of tools for time series forecasting using machine learning models. It works with any scikit-learn compatible regressor such as LightGBM or XGBoost. By leveraging built-in classes such as **ForecasterAutoreg** we are able to automatically transform the time series data into a matrix format suitable for input to a machine learning algorithm.

However, our unique project setup is such that we have a multiplicity of time series. This presents two options:

1. To train over 420 individual forecast models for each of the stations.
2. To train a global forecaster which can be applied to any of the associated time series.

We will explore and compare both options in this report, using the individual models as a baseline against which we hope to improve performance via the use of the global model.

Independent multi-series forecasting

This is our core ML approach, wherein a single model is trained for all time series, but each time series remains independent of the others, meaning that past values of one series are not used as predictors of other series. For this purpose we will make use of the **ForecasterAutoregMultiSeries** class of Skforecast. While there is independence between the time series, modeling provides theoretical benefits should the series follow the same underlying, intrinsic pattern regarding their past and future values. We view this as being plausible in our situation given that all of the time series do appear to share a key common dynamic: that of geographic proximity and the Paris metropolitan area itself.

That being said, we will not take this hypothesis at face value, and will instead adopt a rigorous evidence-based approach to determine what, if any, benefit is derived by the global model over the individual approach (see Results and interpretation below).

b) Implementation of the model

As stated, we will use an implementation based on the **ForecasterAutoregMultiSeries** class of **Skforecast**. This construction has the added benefit of working with any scikit-learn compatible regressor, and as such we have chosen to use an **LGBMRegressor** in order to leverage its inherent ability to handle large, multivariate time series with great efficiency. Given the size of our data set, both in terms of size of the training set, as well as the high multiplicity of time series (over 420 stations), we believe this to be an appropriate selection, however we will test it against alternatives in order to ensure an evidence-based approach is followed to the selection of regressor.

While the initial results were already promising, an extensive grid search was conducted to fine-tune the LGBM specific hyperparameters such as the **learning rate**, **number of leaves**, **number of estimators**, and **max depth**. However, this approach does not comprehensively factor in all of the variables that have resulted from the broader model implementation, and as such we find it to be insufficient on its own. To ensure, as always, an evidence based approach, we have elected to treat these variables as hyperparameters in their own right:

- Given the use of Skforecast, we also need to be able to select the optimal **number of lags**.
- Given the use of the rolling median method for outlier detection, we also have new variables for **outlier threshold** and **window size** which can be optimized.
- Given the preprocessing step to apply weights to the official Covid-19 lockdown periods, this **Covid weight** itself is a variable that can be optimized.

As a result, we constructed a custom grid search which could include these newly created hyperparameters and determine their optimal values as well. The results of the custom hyperparameter grid search are shown in the table below:

Learning Rate	Max Depth	Number Estimators	Number Leaves	Lags	Outlier Threshold	Window Size	Covid Weight
0.05	30	500	100	28	0.8	21	0.9

c) Results and interpretation

Evaluation Approach:

For evaluating the performance of our model we will use a special type of cross-validation called **backtesting**, wherein the model is applied retrospectively to historical data. Skforecast provides for several different constructions of the backtesting approach, which can be tuned to favor either accuracy or efficiency using the “refit” and “fixed-training size” parameters.

In order to strike a balance between the two, we will perform our backtesting using **walk-forward validation**, i.e. **with refit** (to ensure high accuracy at the cost of efficiency) while also using a **fixed training size**. In this approach, the model will be

trained using a fixed window of past observations, and the testing is performed on a rolling basis, where the training window is moved forward in time. This technique is particularly useful when the data is non-stationary, and given the trend and seasonality evident in the data we believe this use case to apply.

Results:

When interpreting the results of the model, it became apparent that while the vast majority of stations show highly accurate performance ($<1\%$ MAPE), there are several “outlier” stations which perform much worse ($10\% < \text{MAPE} < 50\%$). On further exploration to determine which specific predictions (days) are contributing the most to this high MAPE we identified that it was related to the presence of “maintenance/service days” (as discussed in the preprocessing section). Those stations with the highest amount of maintenance days (or with long, continuous periods of such days) were the outlier stations.

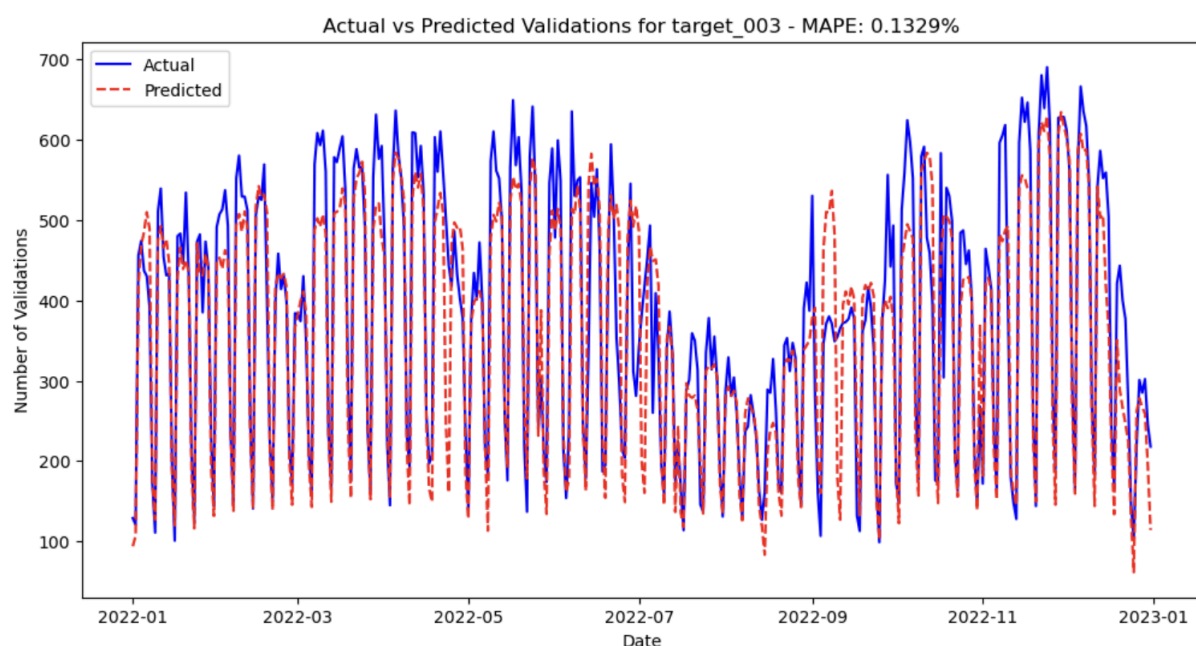
Qualitatively, we do not believe these days provide any true contribution to the realistic MAPE of the model, as they are fundamentally unpredictable (given the current data available). We have no way to determine if or when a station will shut down for technical reasons, and as such these days do not reflect the true performance of the model and should thus not count towards the overall MAPE.

Thankfully, we already have at our disposal a tool to account for this. Using the same outlier mask that was applied for the imputation of these “maintenance days” in the training set, we can also exclude maintenance days from the overall MAPE calculation.

MAPE excluding anomalies SKforecast across all stations = 0.1896%

This means that, on average, our predictions are off by 0.19% for each day at each station, which is a very good result.

To illustrate this result over 1 station (station VBL), we plot the actual VS predicted values:



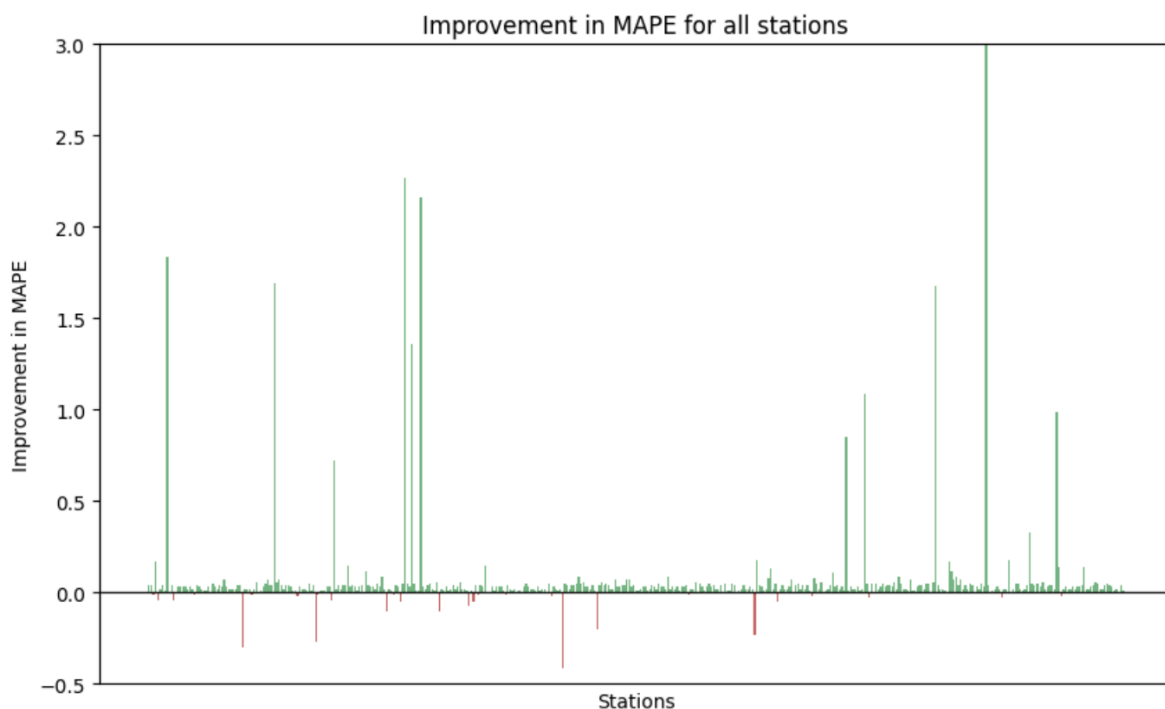
Comparison with single model performance:

SKforecast supports both multi-series and single time series predictions. So why predict all stations together instead of individually?

Modeling them together is useful because the series may follow the same intrinsic pattern regarding their past and future values. While the number of validations at two stations might not be directly related, they may still follow the same external dynamics - that of the greater Paris metropolitan area.

Global forecasting involves building a single predictive model that considers all time series simultaneously. More specifically, independent multi-series forecasting treats each of these time series as independent from one another, meaning that past values of one series are not used as predictors of other series. These types of forecasters attempt to capture the core patterns that govern the various series, thereby mitigating the potential noise that each series might introduce. This approach is computationally efficient, easy to maintain, and can yield more robust generalizations across time series.¹

To see this improvement in practice, we tested both methods (single and multi-series), and the graphs below display the results: they highlight the improvement in MAPE for each station when using the multi-series approach compared to the single-series approach. As shown, most stations experience a significantly greater improvement in MAPE when predicted together rather than individually. The aggregate benefit of this across stations can be seen in the overall MAPE under the **single series** approach: **0.2419%** (vs 0.1896% under the global model).



¹ https://skforecast.org/0.13.0/user_guides/independent-multi-time-series-forecasting

The few stations which see performance decreases were explored individually, and found to all be stations which already had remarkably accurate MAPE, which were made slightly worse under the global model but are all still well below 1%.

Even though the single time series approach has been shown to be less good at predicting most of the stations, it is still useful to predict the outlier stations which we will have to handle independently due to discrepancies with the available data (as discussed in earlier chapters).

2) SKforecast machine learning modeling, single-series prediction

After completing the predictions for the 427 “normal” stations, we can now focus on predicting the “outlier” stations previously mentioned. To recap, 6 of these outlier stations only began operations in 2022, so as explained in the preprocessing section, we cannot predict them. This leaves us with **6 stations** to forecast separately.

We'll apply the same approach as with previous single-series predictions. First, we define the unique training and testing sets for each of these stations. Since the stations didn't all start on the same day, we designate 70% of the available data as the training set and the remaining 30% as the test set. We then implement the forecaster and perform backtesting, considering the exogenous features. After making the predictions, we calculate the MAPE using the mask, as we did previously. We end up with:

	start_date	train_size	test_size	mae	mape
48S	7/2/2017	1406	603	1069.70	7.14%
GW7	11/16/2015	1822	781	3355.99	17.99%
N9K	7/1/2017	1407	603	338.66	2.53%
OWM	1/17/2017	1522	653	26.13	0.64%
RF2	3/15/2015	1994	855	200.55	0.45%
V2P	6/6/2017	1424	611	435.18	2.84%

As can be seen, the MAPE for these stations varies quite widely with some stations performing better than others. 4 of the stations have MAPE below 3% which is still reasonably accurate. However, 2 of the stations have notably higher MAPE which merits further exploration.

3) Deep Learning modeling

In this deep learning section, we explored Long Short-Term Memory (LSTM), a specific type of Recurrent Neural Network (RNN). We did not use any of the exogenous here to be able to treat all stations together.

a) Presentation of the model

Firstly, RNNs are particularly well-suited for time series forecasting because they are designed to process sequences of data by retaining information about previous inputs. This sequential nature makes RNNs effective at capturing temporal dependencies and patterns over time, which is critical for forecasting tasks where future values depend on past observations.

However, standard RNNs are not able to memorize data for a long time and begin to forget its previous inputs. They face challenges when dealing with long sequences, primarily due to the problem of vanishing or exploding gradients during training. This issue makes it difficult for RNNs to remember information from far back in the sequence, which can limit their effectiveness for long-term dependencies in time series data. In our case, we want to predict 1 year in the future thanks to 7 years of data, thus a long-term memory will be necessary.

This is where Long Short-Term Memory (LSTM) networks come into play. LSTMs are a special type of RNN that are specifically designed to overcome the limitations of standard RNNs. They achieve this by incorporating memory cells and gates that control the flow of information. The key components of LSTMs—input, output, and forget gates—allow the network to selectively remember or forget information over longer periods.

These mechanisms enable LSTMs to maintain a long-term memory of previous inputs, making them highly effective for time series forecasting, especially when the relationship between data points spans long intervals.

b) Implementation of the model

We are using all available time series data together for the prediction, aiming to forecast 365 days using 7 years of historical data. In this context, we are handling multiple parallel time series forecasting and multi-step forecasting.

1. First, we need to define our training and testing sets. The model requires input data for both training and testing phases. If we use data from 2015 to 2021 as the training set (which provides 6 years of input), we must also use 6 years of input data for the testing phase. Since we want to predict 2022, we define:

- Train set: 2015 to 2021
- Test set: 2016 to 2022

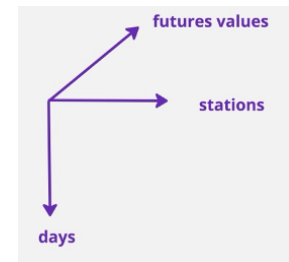
It's important to note that the model will be evaluated solely on the 2022 data.

2. Next, we proceed to normalize the data to prepare it for the model.

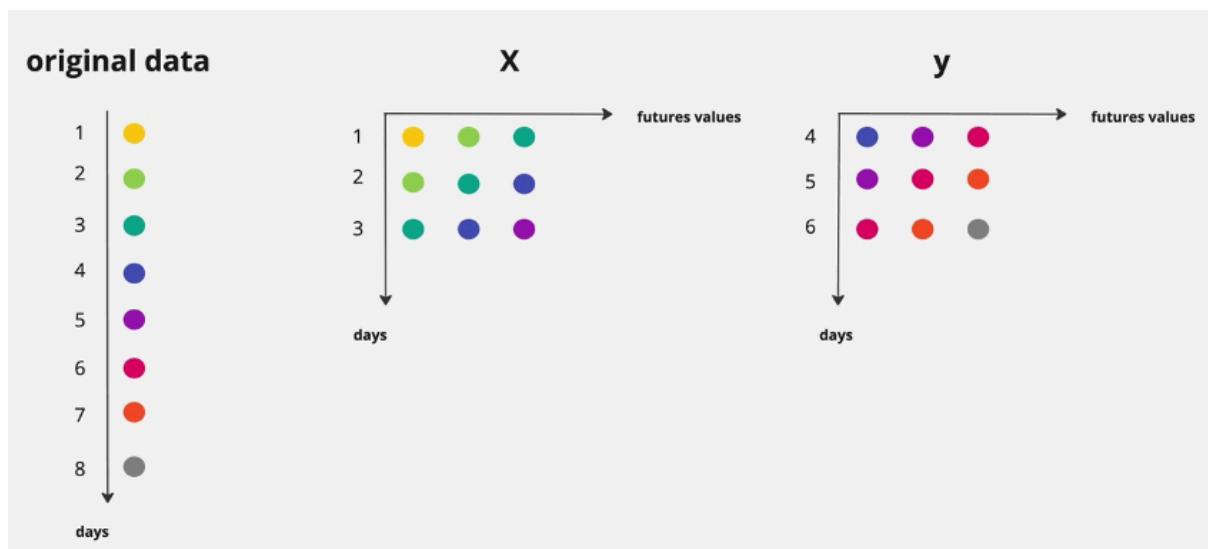
3. Next, we need to prepare our data for the input/output sequence. In practice, the LSTM model learns by mapping a sequence of past observations as input to a corresponding output observation.

Therefore, we must transform the sequence of observations into multiple examples that the LSTM can learn from.

To achieve this, we can define a function that takes the number of input elements and the number of elements to be predicted as output (essentially defining our X and y variables). This function will produce two 3-dimensional matrices, X and y, with the structure (days, stations, future observations).



To clarify this process, let's consider an example where $n_steps_in = 3$ and $n_steps_out = 3$.



After that, we pass our training and testing sets through this function to obtain four matrices in the specified form. These matrices will serve as the input and output for the LSTM model.

4. Once the data is prepared, we can feed it into our model. Here, our model is composed of:
- Sequential layer: this step sets up a linear stack of layers, where each layer is added sequentially, one after the other.
 - Input layer: defines the shape of the input as $(n_steps_in, n_steps_out)$
 - First LSTM Layer
 - RepeatVector Layer: repeats the output from the previous LSTM layer n_steps_out (365) times, to generate sequences of a different length from the input.

- Second LSTM Layer: Another LSTM layer is added after the `RepeatVector` layer. This layer is configured to return sequences of data, allowing it to produce an output at each time step of the sequence.
- Dense Layer: the same dense layer is applied to each time step in the sequence output by the previous LSTM layer. This layer outputs the final prediction for each time step.
- Model Compilation: the model is compiled

Different hyperparameters have been added in order to:

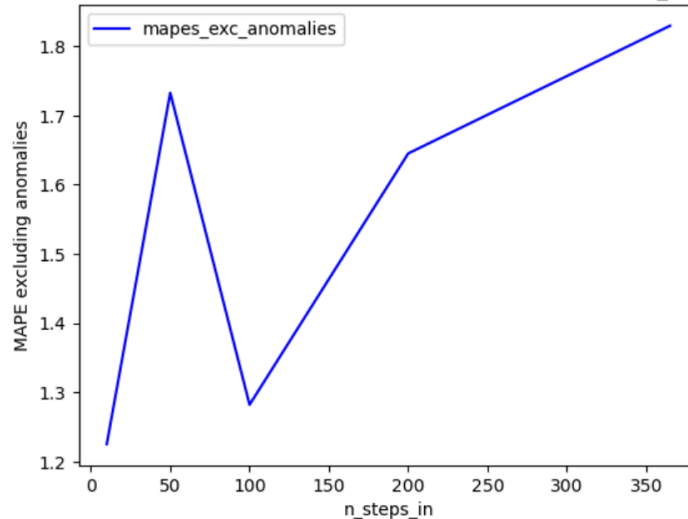
- reduce overfitting: dropout (in the 1st LSTM layer)
- help the model to initialize and avoid vanishing or exploding gradient in the beginning of the training: kernel_initializer (in the 1st LSTM layer)
- keep the model stable and prevent exploding gradient during the process: clipvalue (in the optimizer)

5. Finally, we can make the prediction and evaluate it in the year 2022 using the mask, as explained previously.

c) Optimization

- **N_steps_in:** We knew that `n_steps_out = 365`, but what about `n_steps_in` ? We've tried different values for `n_steps_in` and found an optimal `n_steps_in = 10` as shown below:

MAPE excluding anomalies according to the number of steps in, for `n_steps_out = 365`



- **Hyperparameter tuning:** The hyperparameters that we tried to optimize were the number of nodes in a LSTM layer, the number of epochs, the number of batches, and the loss function. The other hyperparameters were already identified (like the activation function, the learning rate, and the dropout rate). The best set of parameters is the following:

Number of nodes = 100

Number of epochs = 5

Number of batches = 1

Loss function = 'mae'

d) Results and interpretation

Despite going through the optimization phase, the results are still unsatisfactory.

MAPE excluding anomalies LSTM = 123%

This means that, on average, our predictions are off by 123% for each day at each station. While this is an improvement over our baseline model, it remains inadequate compared to the Skforecaster.

Thus, the LSTM approach is not suitable for our time series issue. Indeed, it fails at capturing the intrinsic patterns between the stations, and can not add the exogenous features when deploying a multi series approach.

Conclusion

In summary, our task is time series prediction, a supervised forecasting problem using regression techniques, with 439 stations to predict. Throughout the study, we:

- Imputed outliers and missing values using a Piecewise Cubic Hermite Interpolating Polynomial (PCHIP).
- Incorporated exogenous features like time, lag variables, and weather data to enhance model learning.
- Applied various modeling approaches, including both machine learning and deep learning.

The most effective solution turned out to be the machine learning model, specifically using the SKforecaster approach. The multi-series SKforecaster was applied to 430 stations, while the single-series SKforecaster was used for 7 stations. The remaining 5 stations, which began operating in 2022, could not be predicted.

The results are as follows:

MAPE excluding anomalies for 430 stations (multi-series SKforecaster) = 0.1896%

MAPE excluding anomalies for 430 stations (single-series SKforecaster) = 0.2419%

Thus we conclude that the multi-series approach performed exceptionally well, with a prediction error of just 0.19% per day for each station.


For the outlier stations, it is less meaningful to look at an aggregated MAPE given the extreme variability in performance, and as such these stations will have their MAPE results reported individually:

- Station **48S**: **7.14%**
- Station **GQ7**: **17.99%**
- Station **N9K**: **2.53%**
- Station **OWM**: **0.64%**
- Station **RF2**: **0.45%**
- Station **V2P**: **2.84%**

Key Challenges Encountered:

Despite the strong results and performance of our model, we encountered several challenges:

- The first challenge stemmed from the nature of the problem, which involved predicting multiple time series. This raised the question of whether to encode or pivot the data, with the approach varying depending on the model being tested. So in the beginning, we had to test both approaches with all models we were trying to set up.
- This led to the second challenge: memory issues. In particular, our initial approach of encoding the stations significantly increased the dataset's size. With 2 million rows, expanding the dataset from roughly 30 columns to 435 caused



major memory constraints during modeling, sometimes requiring hours of processing time, even though we were only focused on getting the models to run, not yet optimizing them.

- Finally, in the deep learning approach, the input/output phase also took longer than anticipated to implement and visualize. Unlike the machine learning models, the train and test sets were structured differently, yet we still needed to ensure comparability using the same metric—MAPE excluding anomalies for 2022. And these input/output “new hyperparameters” were also something to optimize.

Some potential improvements to our project could include incorporating the time of day, allowing us to identify when each station is busiest and better anticipate staffing needs, for instance. Additionally, we could introduce variables like maintenance schedules for each station. While we currently rely on assumptions in our analysis, having these actual variables would improve imputation accuracy and lead to more precise predictions.