December 22, 2024

Machine Learning Project: Pedal Predictors Team

Authors: Helena Cobo (23-332-489), Rodrigo Lopez (23-329-683) and Damiano Galassi (24-323-388)

Professors: Sebastian Engelke, Christoph Valvason, Manuel Hentschel

University of Geneva

Faculty of Economics and Management

A.Y. 2024/2025

Abstract

The newly established company Mule is changing urban mobility in Geneva by offering short-term bicycle rentals. To secure bank loans and ensure operational success, Mule must accurately forecast the number of hourly bike rentals over the next three months. Leveraging machine learning (ML) techniques, this project aims at developing predictive models based on temporal and meteorological data provided by a comparable company in another country. A broad range of ML models, from interpretable linear regression to complex ensemble methods like random forests and gradient boosting, is evaluated using the mean absolute error (MAE) metric. Additionally, this project aims at showing the capabilities of the different models in terms of accuracy, computational cost and ability to serve Mule for its operational success. Finally, this study provides actionable business insights by identifying key predictors, inspecting their distributions and patterns, and quantifying their impact on bike rental demand. The results aim to provide Mule accurate forecasts and strategic insights to optimize fleet allocation, ensure financial stability, and contribute to the success of Geneva's soft mobility initiatives.

I. Introduction

This project addresses the challenge of predicting daily bike rental counts using a dataset containing temporal and meteorological information. To achieve this, we explore a variety of Machine Learning (ML) methods with varying levels of complexity to identify the most effective approach for providing accurate forecasts for the next three months.

Our methodology begins with a data preprocessing stage, where we develop techniques to handle missing values and ensure the dataset is clean and robust for analysis. Following this, we conducted an exploratory data analysis (EDA) to uncover insights into the time series nature of the data and the relationships between various predictors and the target variable. This step includes correlation analysis, outliers' detection, and, mostly, feature engineering, where categorical variables are transformed in numerical features, new predictors are generated, and the relevance of each feature is assessed using methods such as correlation analysis and feature importance via baseline decision tree models.

Subsequently, the cleaned and engineered dataset is then used to model the underlying relationship between the predictors (X) and the target variable (y). We apply multiple ML modeling techniques, ranging from simpler approaches to more advanced and complex algorithms, with the objective of training a predictive model.

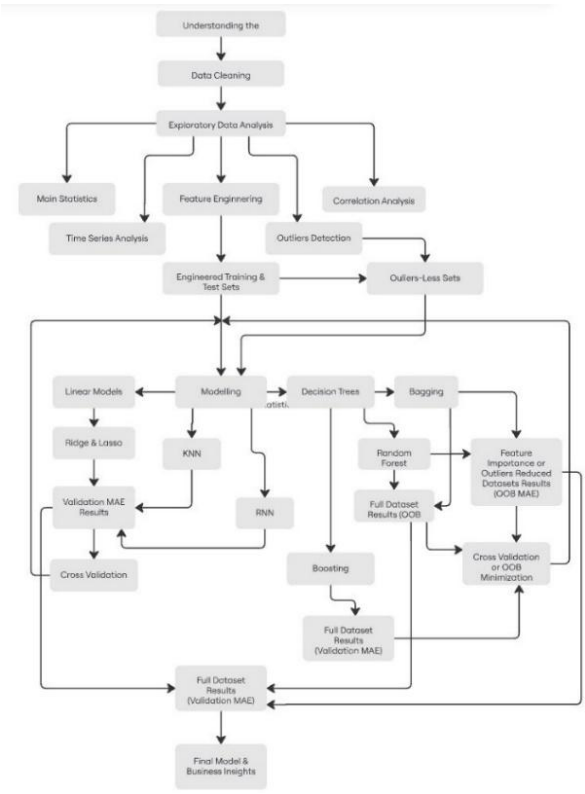


Figure 1: Data Treatment Pipeline and Project Decision Tree.

The performance of these models is evaluated using the Mean Absolute Error (MAE) metric to ensure a fair and consistent comparison. As discussed, Figure 1 reports the tree-structured decision process we applied throughout our analysis. Finally, the report concludes by identifying the model that delivers the most accurate predictions, supported by a thorough discussion of its strengths and limitations.

II. Data Understanding & Cleaning

The train dataset used in this project comprises 15211 hourly observations, structured as a dataframe of 14 columns covering 21 months. In contrast, the test dataset contains 2168 observations, associated with the last three months of 2012. It has the same predictors but, of course, it does not include the response variable, which is the target we are aiming to predict.

Specifically, the predictors are:

- *dteday:* date expressed in a yyyy-mm-dd format.
- *season:* categorical feature assuming 4 possible values (Winter, Spring, Summer, Fall).
- *yr:* assuming either the value of 2011 or 2012.
- *mnth:* categoraical features assuming 12 possible values (January, ..., December).
- *hr:* hour of the day (0, ..., 23).
- *holiday:* binary values being False or True.
- *weekday: c*ategorical variable assuming 7 possible values (Monday, ..., Sunday).
- *workingday:* boolean variable being either True or False.
- *weathersit:* descriptive situation of the weather (assumes only 4 values).
- *temp:* normalized temperature.
- *atemp:* normalized feeling temperature.
- *hum:* normalized humidity.
- *windspeed:* normalized wind speed.

And the response variable *cnt,* representing the number of bikes rented each hour.

Given that the data is observed on an hourly and sequential basis, it can be classified as a time series This means that certain variables remain relatively constant over specific time periods, which will be useful for the missing values management stage.

After understanding the nature of the predictor space and the goal of the work, we proceeded to control the quality of the starting dataset in terms of missing values. Figure 2 reports a bar chart where we can observe the % of missing information in the training dataset for each feature. Given that the number of missing values is quite low, we could have proceeded by dropping the rows containing these blank fields. Still, we decided to retain as much possible the information, hence we proceeded with different data filling techniques:

For *weathersit* we forward filled with the preceding data point if only if there were no two consecutive NaN. This approach assumes that the weather situation is invariant in a 1-hour time span.

For *temp, atemp, hum,* and *windspeed* we filled using the average value between the preceding and the subsequent observation. As weather conditions present an important inertia, these are most likely to stay constrained between the two defined values, while it would be quite strange to observe strange jumps.

For day-invariant predictors such as *yr, season, month, weekday, workingday,* and *holiday* we used data from the same day to fill in the missing information. The dataset has been sorted chronologically, which allowed us to leverage information from both preceding and subsequent data point when they share the dame date. By doing so, we ensured that any imputed missing value corresponded to an hour within the same day.

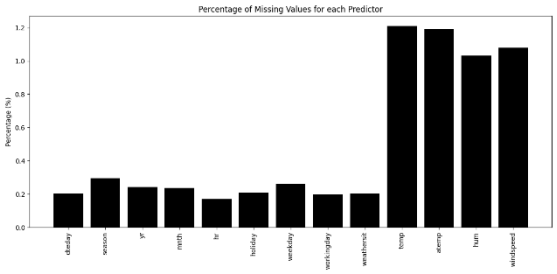


Figure 2: Missing values bars for each of the features in the predictor space bikes rented per day in training data expressed as a % over the total number of observations in the starting data set.

After this process, the missing values accounted for only 0.289% of our dataset, so we opted to remove them

III. Exploratory Data analysis

Exploratory Data Analysis, EDA, is a critical step in data processing. Through EDA the dataset is analyzed and summarized to uncover patterns, detect anomalies and understanding deeply the dataset before proceeding to the modelling stage.

In this project we started exploring the predictors datatypes. The dataset contains categorical and numerical values, Specifically, *yr, hr, temp* and *windspeed* are float 64 datatypes, which do not require any transformation at this stage. On the other hand, categorical variables must be encoded into numerical ones before performing regressions.

A. *Main Statistics*

The data treatment has been performed mostly in the previously mentioned Data Cleaning stage and in the following ones (Feature Engineering). Right now, we decided to focus our attention on descriptive statistics of the main numerical features for our train sample. In Figure 3 we report a bar chart showing median, mean and quartiles of the predictors' distributions. We can observe all the features, other than *windspeed*, present a quite regular distribution, while *windspeed* presents a low value for the key percentiles (quartiles 25, 50 and 75) and rapidly reaches a high maximum value. Another important fact to cite is that, even if the dataset has been normalized (e.g. *temp* has been normalized by the maximum value of 50 °C), we still observe minima and maxima different from 0 and 1. This raised doubt, but probably the maximum and minimum values could belong to the test set, thus we do not see their occurrence here.

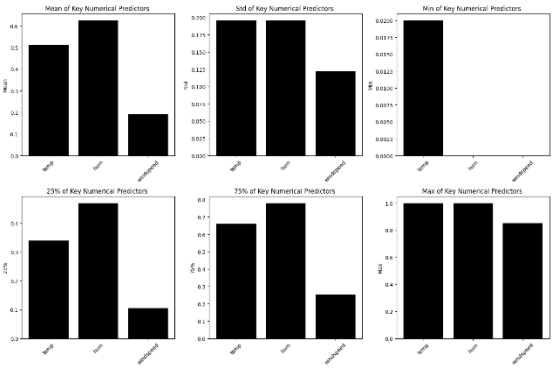


Figure 3: Weather based predictors main statistics.

The skewness of 2.67 confirms the deviation from normality of *windspeed*. This strong deviation from normality can indicate the presence of outliers, or, most likely, the presence of highly windy days which occurred between 2011 and 2012.

An important observation to state is the time dependence of the response variable. As a fact, most of the observations are represented by a series of variables which are hourly discretized. This highlights the time series nature of the dataset. Others, such as *holiday, weekday, workingday, mnth, season* and *yr* are hour independent.

B. *Time series Analysis*

To better highlight the time dependance of the *cnt* response variable we report (Fig. 4) the evolution of the number of rented bikes from 01/01/2011 to 30/09/2012. From this chart we can highlight three key points. First, the overall trend is increasing, this means that during 2012 the firm has rented more bikes increasing, at constant profit margin, the overall profitability. Hence, we expect the same trend to continue also in the test set. Secondly, focusing on 2011 only, we observe that seasonality has a great importance. Furthermore, we observe a sort of cyclicity with rented bikes spiking during the hot seasons and dumping during colder ones. Finally, the targe variable presents a high variability, characterized by significant spikes and drops.

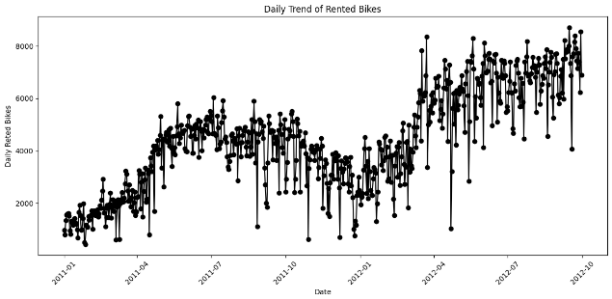


Figure 4: Average bikes rented per day in training data.

To confirm the importance of seasonality we also report a cumulative plot of the bikes rented during 2011 grouped by season. Figure 5 confirms the observed pattern: bikes are mostly rented during hot seasons and experience a sharp drop during winter.

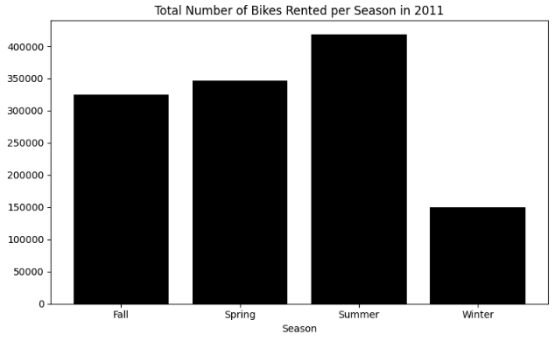


Figure 5: Cumulative number of bikes rented per Season during 2011.

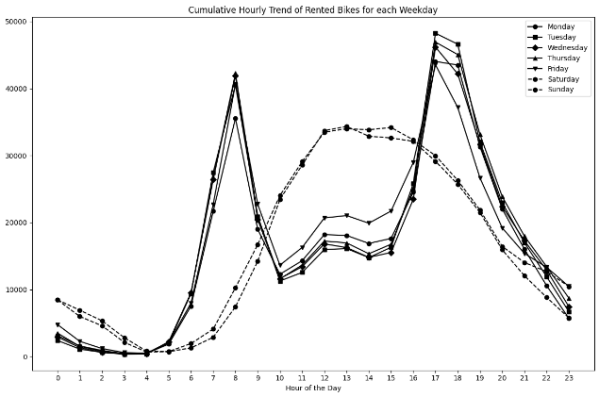


Figure 6: Hourly trend of rented bike during working and non-working days.

To narrow down the daily trend we inspected the evolution of rentals during the day. Figure 6 reveals that during working days the number of rentals follows a bimodal distribution with peaks around 8 AM and 5 PM. This captures the fact that most bike users rent bikes to cover their quotidian activities such as going/coming back from work. Meanwhile, during weekends, the distribution is

smoother and peaks around midday, reflecting more leisure-based rentals.

Finally, the importance of weather is assessed, Weather conditions influence the number of rentals. From Figure 7 we can notice that increasing temperatures favor bike rental almost monotonically. Specifically, the scatter plot reaches a peak around 29 °C, after which users decrease their bike usage due to extremely hot conditions. Humidity dependance is noisier, indicating the absence of a clear trend. Finally, windspeed presents a quasi-monotonic decreasing trend, highlighting that, on average, rentals decrease during windy days. For these reasons, during the feature engineering stage we will introduce interaction terms to capture the non-linear influence of weather conditions on the response variable.

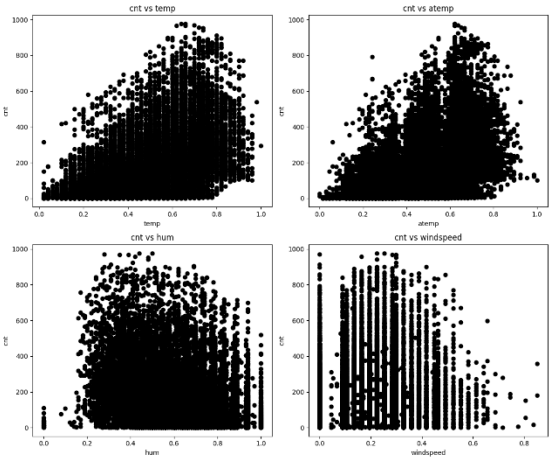


Figure 7: Scatter plots of number of rented bikes vs weather predictors.

C. Feature Engineering

Feature engineering is a crucial step in transforming raw data into a format that enhances the model's ability to learn and make accurate predictions. In this section, we describe the key transformations and derived features created to improve the performance of our machine learning models and reduce prediction errors.

Our focus includes encoding categorical variables, generating interaction terms, and incorporating time-based patterns to capture seasonal and hourly variations in the data. Additionally, we evaluate feature importance to identify and prioritize the most influential predictors. These transformations are consistently applied to both the training and testing datasets to ensure uniformity and avoid data leakage.

Several categorical variables were encoded using mapping dictionaries to convert them into numerical representations. These variables include *season*, *weekday*, *holiday*, *workingday*, and *weathersit*.

To capture relationships between different features, several interaction terms were created: *temperature* and *humidity*, *temperature* and *windspeed*, *rain* and *temperature*, *season* and *temperature* and *windspeed* squared. Additionally, a binary *rain* feature is created to spot the rainy days, as EDA showed that the weather condition has a high influence in bike rental decisions.

Time-related features were transformed to capture seasonality and cyclic effects in the data. Given the daily cyclical nature of bike rentals, we transformed the *hr* feature using sinusoidal functions to capture the periodic behavior, as the behavior of bike rentals varies throughout the 24-hour day. The same method is applied to the *month* feature, using both sinusoids and cosinusoids.

As noted in the EDA, the number of bike rentals exhibits a strong cyclical pattern throughout the day. To better capture these variations, the hour of the day was classified into distinct time-of-day intervals.

We also observed that peak rental times differ between working days and weekends (Fig 6). To account for this variation, a new feature, *hour_pick*, was introduced. This new predictor take specific values during peak hours differentiating between working days and weekends. Additionally, to further enhance the model's ability to capture subtle patterns, additional interaction terms were created: *hour* and *working day*; *hour* and *weekday* and *day* and *hr_sin* interaction

Finally, given the high variability of the number of bikes rented and to improve the model's ability to recognize demand on special occasions, we introduced two binary features, *spike_flag* and *drop_flag*, which capture significant fluctuations in daily bike rental counts. These features were created by identifying days in 2011 with unusually high or low demand (Fig. 8), based on predefined thresholds (absolute change of 50 and percentage 30% with respect to previous day) . By including these features, we account for potential recurring events, such as days before holidays or local festivities, that could affect rental patterns. This allows the model to better predict demand for similar events in 2012.

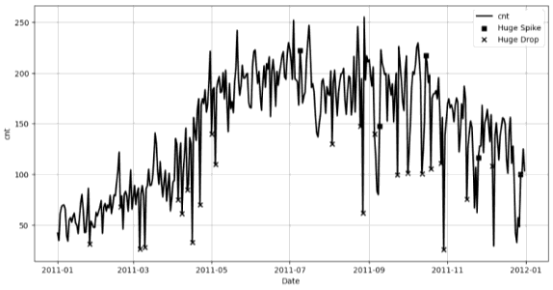


Figure 8: Spikes and Drops in demand during 2011.

Through these feature engineering techniques, we aimed at capturing the temporal, weather, and interaction patterns inherent in the data. By encoding categorical variables, creating interaction terms, and applying time-

series transformations like sine and cosine for cyclical features we provided the model with more meaningful representations of the underlying data. These transformations are expected to improve the model's predictive performance and ability to capture key patterns in bike rental behavior.

D. Outliers Detection

Outliers are data points that significantly differ from the central tendency of the observations in a dataset. They can arise due to various factors, such as errors in data collection, measurement mistakes, or inherent variability in the data. Outliers can have a disproportionately large effect on statistical models, especially those sensitive to extreme values, such as linear regression or decision trees. Dropping outliers can help prevent these extreme values from distorting the model's performance, leading to more accurate and reliable predictions. Furthermore, outliers can skew the distribution of the data, misrepresenting its central tendency and variability. By removing them, we can ensure that the dataset better reflects the general patterns within the data.

However, it's crucial to distinguish between true outliers and data errors before deciding to remove them, as valuable information could be lost if outliers are incorrectly discarded. For such reasons we decided to conduct an outlier removal in parallel with our work on the entire dataset, which, as we will see, will be tested in some models.

To perform this detection, we decided to define outliers all the observations deviating more than two standard deviations from their mean. This is obtained by using the z-score, a common statistical method. The z-score exactly measures how many standard deviations a data point is from its average. Specifically, the formula is:

$$z_i = \frac{x_i - \mu}{\sigma}$$

Where x_i is the i^{th} data point, μ and σ are respectively the mean and standard deviation of the distribution of x. Data points with an absolute value of the z-score above threshold ($z_i > 2$ and $z_i < -2$) have been eliminated. The output of this treatment reduced the vertical dimensionality of the training data by around 18%.

It must be specified that this method assumes the data follows a roughly normal distribution and may not be suitable for heavily skewed datasets. Hence, we decided to transform these to create pseudo-normal distributions before proceeding with the z-score elimination. The applied transformation functions, and the predictors relative skewness before and after transformation are reported in Table 1.

Feature Name	Applied Transf.	Skewness before	Skewness after
windspeed	log	2.67	-0.32
hum_windspeed	cube root	3.13	-0.39
hour_weekday	cube root	1.00	-0.53
temp_windspeed	cube root	1.00	0.77

Table 1: Applied transformations and skewness values before and after the cubic root or logarithmic function application.

E. Correlation Analysis

After the feature engineering step, correlation analysis has been performed to evaluate the relationships between the newly created features and the target variable, as well as among the features themselves. This step helps to identify which engineered features are most relevant for predictive modeling and highlights potential issues such as multicollinearity. Highly correlated features may lead to redundancy, increasing model complexity without improving performance. To address this, correlated features can be dropped, combined, or transformed, ensuring the final feature set is both efficient and interpretable.

Due to the presence of a great number of features we report here two correlation matrixes, the first one (Fig. 9) highlights the correlation for time-based variables, while the second one (Fig. 10) reports the interdependency between the non-temporal features.

It is possible to notice that features such as:

- *temp_hum,*
- *holiday_num,*
- *working_day_num*
- *time_of_day[19.0,23.0]*
- *Weekday*
- *hum_windspeed*
- *rain_temp_interaction*

Present correlations values with the response variable lower than 0.1, indicating a low importance for the prediction. However, correlation implies statistical association but does not infer any causal relationship.

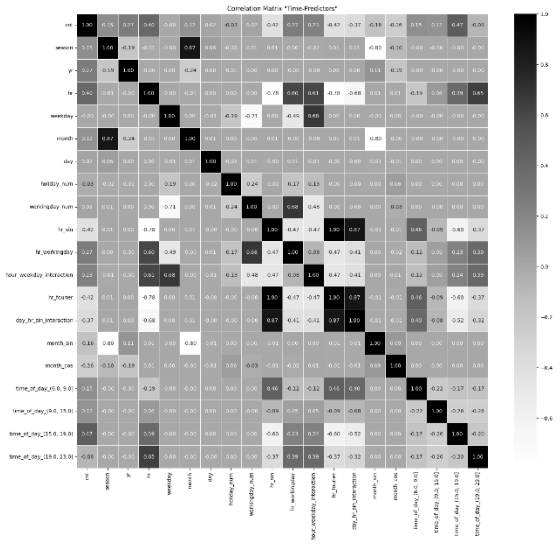


Figure 9: Time-dependent variables correlation matrix.

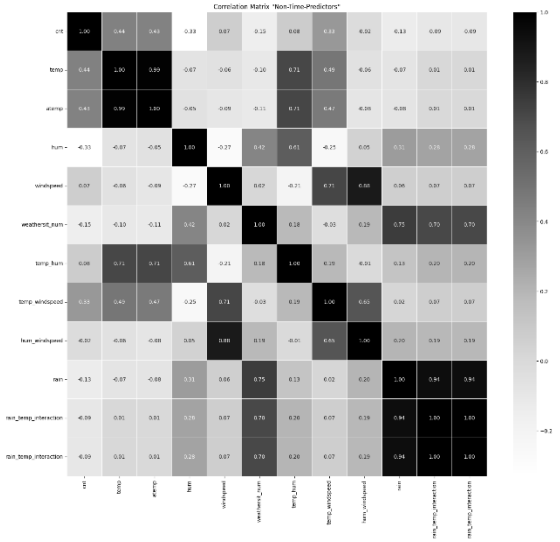


Figure 10: Time-independent variables correlation matrix.

Other than the target-predictors collinearity, the correlation matrix allowed us to identify predictors that were highly correlated with one another. These redundant interactions include:

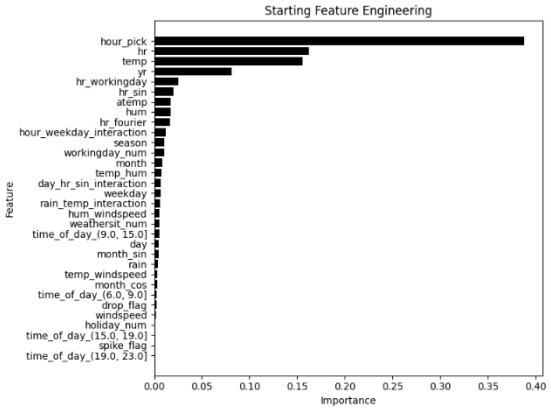
- *rain* and *rain_temp_interaction*
- *hum_windspeed* and *windspeed*
- *temp* and *atemp*
- *hr_fourier* and *hr_sin*
- *hr_sin* and *day_hr_sin_interactions*
- *season* and *month*

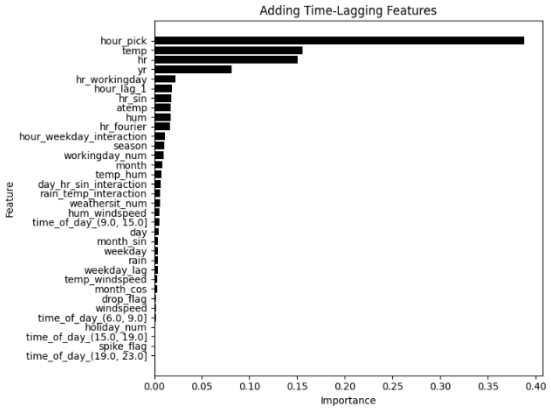
For this reason, we decided to create a restricted dataset by removing both non-informative predictors and highly correlated predictors. This refined dataset has been then used for training and testing some of our models to enhance their performance and reduce redundancy.

F. Additional Engineering: Lagging Features & Rolling Statistics

Lagging features and rolling statistics have been introduced to leverage the time series nature of the dataset. Lagging features, such as *hour_lag_1*, capture the values of predictors at previous time steps. These features help the model identify temporal dependencies and trends, which are crucial for improving predictions in sequential data. In addition to lagging features, rolling statistics defined on weather-based predictors such as *rolling_temp_mean* and *rolling_temp_std* have been incorporated. These rolling metrics provide aggregated insights over a defined window of time (7 observations), smoothing out short-term fluctuations while preserving longer-term trends.

To assess the importance of the last added features we trained a base Random Forest (RF) regressor and evaluated the feature importance. Figure 11 shows how the importance of these new predictors is more relevant than some of the pre-existing ones. For this reason, we decided to keep them for the moment.





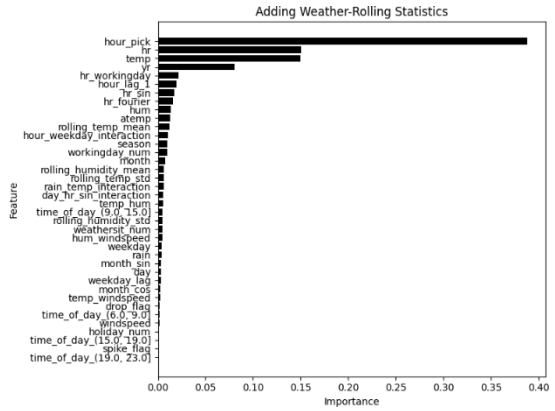


Figure 11: RF derived feature importance bar chart for the feature engineered data set (top), after addition of lagging features (center), after addition of lagging features and rolling statistics (bottom).

IV. MODELLING

A. Train/test split and Cross Validation

Due to the time-series nature of the data, we decided to perform a non-shuffled the train-test split. This approach ensures that we account for potential time trends in bike demand, maintaining the chronological order of the observations.

Similarly, for cross-validation, we employed the TimeSeriesSplit method. This technique ensures that each split incorporates additional consecutive data for the subsequent periods, preserving the temporal structure. As illustrated in Figure 12, this method effectively mimics the real-world scenario where future data is predicted based on past trends.

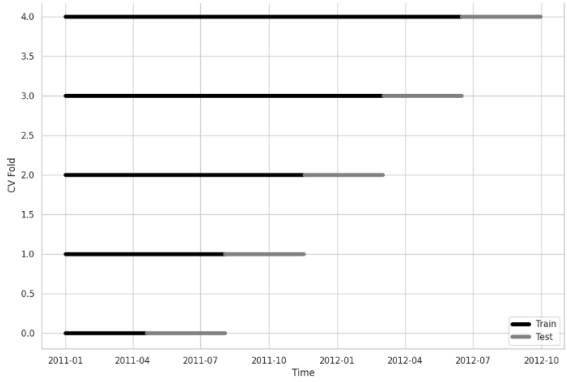


Figure 12: TimeSeriesSplit with 5 folds

B. Linear Regression

To start the modeling process, we implemented the simplest and most straightforward approach: Linear Regression. This model was selected as a baseline to understand the relationships between the features and the target variable. We used train-test split with a 30% test set and 70% training set, ensuring the time series nature of the data was preserved by a non-shuffled train-validation split. This decision reflects a real-world prediction scenario, where future predictions need to be based on past data

The model was fitted on the training data, and we used it to predict both the training and test sets. Since negative rental predictions don't make sense in this context, we took an additional step to ensure that any negative predictions were set to zero. After the predictions were made, we calculated the MAE for both the training and test sets. The train MAE measures how well the model fits the training data, while the test MAE shows how well it generalizes to new, unseen data.

While the linear model provides a quick and simple way to start the analysis, the significant difference in MAE between the training and test sets points to its limitations in capturing complex relationships within the data. This suggests the need for more advanced models, such as KNN, or Gradient Boosting, which could potentially provide better generalization and flexibility to handle the complexities of the dataset.

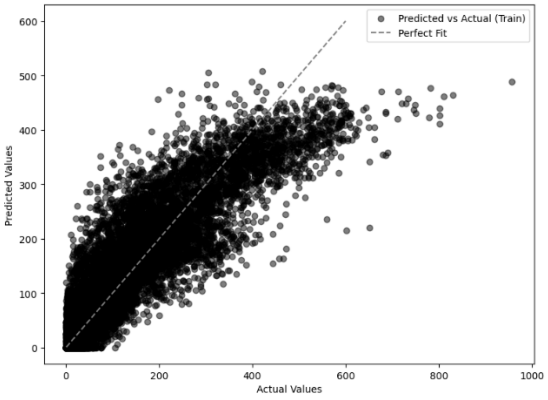


Figure 13: Predicted vs Actual values in Linear regression (Train set).

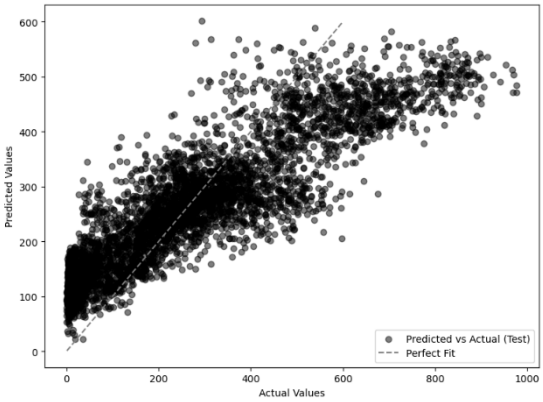


Figure 14: Predicted vs Actual values in Linear regression (Test set).

C. KNN Neighbors

The next machine learning method we applied is the K-Nearest Neighbors (KNN) regression. KNN is a non-parametric algorithm that makes predictions based on the average of the nearest data points, making it simple yet effective for regression tasks. However, selecting the

optimal number of neighbors is crucial for achieving the best model performance. To determine the optimal number of neighbors, we employed cross-validation (CV) using the TimeSeriesSplit method. This technique is well-suited for time series data as it ensures that the temporal order of the observations is maintained during both training and testing. Specifically, we used a 5-fold cross-validation strategy, training and testing the model on each fold.

We used a Pipeline to standardize the features and apply the KNN model. The StandardScaler was used to normalize the input features, ensuring that each feature contributes equally to the distance metric used by the KNN algorithm. A GridSearchCV was then employed to search for the best number of neighbors (k) for a broad range of values (1 to 30).

After cross-validating, the results were extracted and analyzed to identify the optimal neighbor. The MAE was used as the evaluation metric, and we selected the k that minimized the MAE. Upon evaluating the KNN model, we noticed a significant disparity between the training and test errors, with the test error being substantially higher. This large gap is indicative of overfitting, a common issue in KNN where the model is too closely fitted to the training data and fails to generalize well to unseen data. The model captures the noise and minor fluctuations in the training set, which reduces its predictive performance on the test set.

To address overfitting, we implemented the one-standard deviation rule, which selects a model that balances performance on the training set and generalization to unseen data. While this rule improved the model's performance, the MAE still differed significantly between the training and test sets, indicating that overfitting was still a concern.

It is important to note that KNN is not an ideal method when the number of predictors is large, due to the curse of dimensionality. This phenomenon leads to a significant degradation in the algorithm's performance as the dimensionality of the feature space increases.

Considering this limitation and the potential for overfitting, we propose exploring regularized models such as Ridge Regression and Lasso Regression. These regularization techniques introduce penalty terms to the loss function, effectively constraining the complexity of the model. By discouraging overfitting, these methods help the model focus on the most relevant features, improving its ability to generalize to new, unseen data.

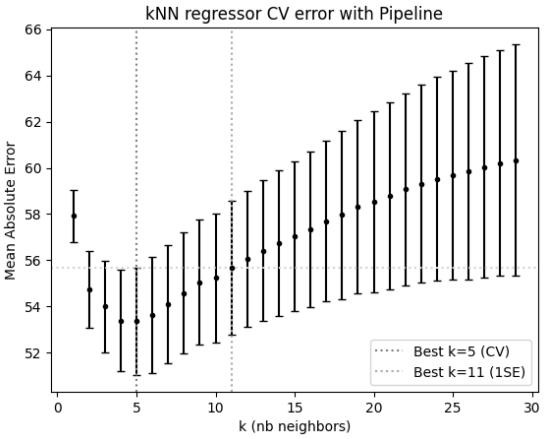


Figure 15: KNN Regressor CV error.

D. Ridge Regression

To avoid collinearity of variables and overfitting, we propose to use Ridge regression, which is also known as the L2 regularization due to the penalization of the squared value of the parameters. One of the main characteristics of this method is the reduction of flexibility or complexity of the model due to the inclusion of a penalty term. This characteristic puts a cost on the squared value of every parameter that is included in the model, multiplied by a tuning parameter that reflects the trade-off between the importance given to bias and variance in our specific model. This adjustment aims for getting better performance of the model and increase accuracy for prediction on unseen data.

Ridge regression minimizes the model parameters β_j for the p predictors with respect to the following function:

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 - \alpha\sum_{j=1}^{p}\beta_j^2$$

In which we aim to minimize the minimum squared error including a penalization for overfitting by adding an α parameter that multiplies the sum of the squared model parameters. This means that adding parameters –which would mean overfitting– will be costly for our function. We should mention that this penalization does not exclude any variables on the dataset, as other methods like Lasso would do. The parameter α reflects the trade-off between bias and variance.

One disadvantage of this method for our specific problem is that we are not able to change the loss function and directly adapt it to our goal of minimizing the mean absolute error. If we would like to do that, we find that the resulting loss function is not differentiable at 0, so it is not

possible to apply gradient descent. Nevertheless, we will be able to adapt the model in the parameter selection.

To obtain the best value for the parameter alpha, we used cross-validation. In this phase we can compare the performance of the different models by means of their MAE performance. Moreover, to obtain a simpler and parsimonious model we use the one-standard-error rule, through which we choose the lambda parameter by selecting the highest parameter in the one-standard-error threshold of the value that minimizes the MAE in the cross-validation process.

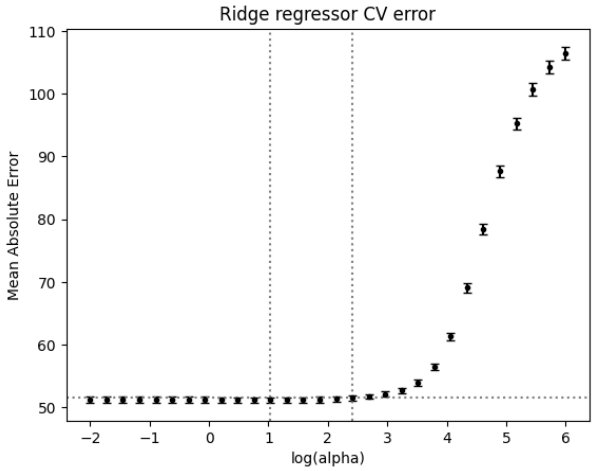


Figure 16: Ridge Regressor CV error.

E. Lasso Regression

Another method of regularization that would help avoid overfitting is lasso regularization. The main idea of this method is penalizing the number of parameters by adding a term compounded of the multiplication of a lambda parameter and the sum of the absolute value of all the parameters:

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 - \alpha\sum_{j=1}^{p}|\beta_j|$$

This kind of regularization is useful not only to avoid overfitting but also for model selection in the sense that it gives us an idea of the importance of the parameters. Lasso forces some feature coefficients to be zero, excluding them from the model. In Figure 17 we show the parameter values of the variables kept by Lasso.

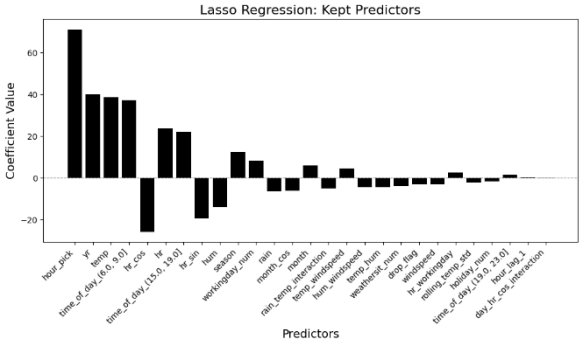


Figure 17: Coefficients for kept predictors for Lasso.

Using this method, we can have an idea of the interpretation of the coefficients of the features that mainly explain the variation of cnt according to Lasso regression. The most significant variable is *hour_pick*, which means pick hours result, on average, in nearly 70 more bike rentals than regular hours. The following important predictors are *yr, temp, time_of_day_(9.0, 15.0]*, which positively influence the target variable.

At the same time, this method has reduced the complexity of the model by getting rid of the least relevant features according to the minimization of the loss function and dropping their coefficients to zero. In our case, Lasso has ignored the following features:

- *weekday*
- *windspeed*
- *day*
- *spike_flag*
- *holiday_num*
- *temp_hum*
- *temp_windspeed*
- *time_of_day_(9.0, 15.0]*
- *time_of_day_(19.0, 23.0]*
- *hour_weekday_interaction*
- *hr_fourier*
- *month_sin*
- *hour_lag_1*
- *weekday_lag*
- *rolling_temp_mean*
- *rolling_humidity_mean*
- *rolling_temp_std*
- *rolling_humidity_std*

Similarly to ridge regression, it is not possible to directly adapt the MAE in the objective function. Nevertheless, we adapted it during the cross-validation process. We compared the performance on MAE of different values of the parameter α. As we did in Ridge regression, we used

the one-standard-error rule to choose the best parameter of regularization as we show in Figure 19.

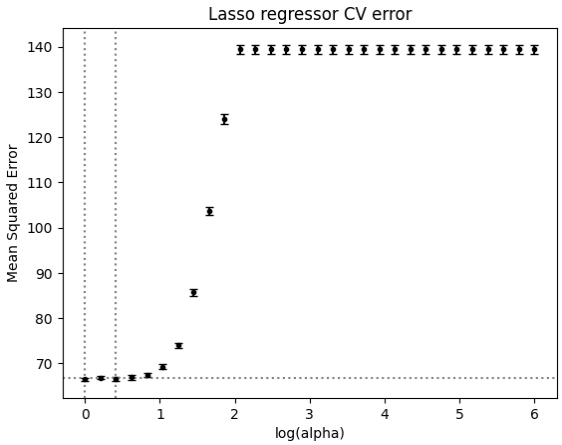


Figure 18: Lasso Regressor CV error.

F. Gradient Boosting

The next machine learning technique we applied is Gradient Boosting, an ensemble learning method that combines multiple weak learners to form a strong predictive model. It works by training models sequentially, with each new model focusing on correcting the errors of the previous one. Gradient Boosting adjusts the weights of the misclassified data points to gradually minimize the model's loss function.

For this task, we used the GradientBoostingRegressor from the scikit-learn library, designed specifically for regression problems. To identify the best hyperparameters, we utilized GridSearchCV with the TimeSeriesSplit cross-validation method, performing an exhaustive grid search over various combinations of hyperparameters. The best combination was determined using the training split, which constituted 70% of the data. The model was then retrained using the optimal hyperparameters and evaluated on both the training and testing splits using the MAE as the evaluation metric.

Despite these efforts, the computed MAE values remained relatively high, suggesting that Gradient Boosting might not perform optimally when the dataset contains a high number of predictors. To address this, we also applied Gradient Boosting to a restricted dataset, constructed in Section 2.E by dropping highly correlated predictors and low importance features. In this scenario, we used RandomizedSearchCV instead of a full grid search to reduce computational costs.

After removing the least informative features, the MAE values for the restricted dataset remained relatively high. Interestingly, the training error was observed to be higher than the test error, suggesting that the use of randomized cross-validation might not have been effective in identifying the optimal hyperparameters, potentially leading to underfitting of the training data.

While Gradient Boosting demonstrated improvement over simpler models in capturing the relationships between the predictors and the target variable, the model faced challenges in fully optimizing predictive performance.

G. Decision Trees

Decision Trees

One of the last families of ML model we tested is Decision Trees. Such models work by recursively partitioning the dataset into smaller subsets based on feature values, creating a tree structure where each internal node represents the decision rule, and the last node (leaf) is the predicted output. The main goal of the algorithm is to minimize the impurity of each split using loss functions such as Mean Squared Error (MSE) or Mean Absolute Eror (MAE).

The main advantage of these models is the interpretability, in fact tree structures can be visualized and it is possible to follow the “algorithm decision making process”. Moreover, they offer great versatility as they can treat both numerical and categorical variables and they are scaling-independent. Unfortunately, building fully-grown trees causes data overfitting, which must be assessed using different techniques.

The simplest technique is pruning. It consists in cutting terminal nodes post training or directly constraining the model not to build full trees ex-ante. In any case here we will mostly make use of regularization techniques such as Bagging (Bootstrap Aggregation) and one of its derivate models, Random Forest (RF).

These two are categorized as ensemble learning models which use low-bias trees (fully grown) and aim at reducing the variance term of the total error by averaging out the fitting process on multiple subsets. In other words, they rely on combining several decision trees to make the final estimation. The way this combination occurs differentiates Bagging from RF.

Bagging creates multiple subsets of the training data by randomly sampling with replacement. Each tree built on the different subsets is independently trained and all the predictions from the different trees are aggregated to produce one single output. This reduces the variance, but all the trees are still strongly dependent on dominant features, leading to correlated errors across the ensemble.

In this case RF comes in help. The main idea is to add an additional layer of randomness by further decorrelating the generated subsets. As a fact, each subset will not be reduced only horizontally, but also a vertical reduction comes into play by randomly selecting the features that each subset will contain. The most common way to proceed is to force the model to only select $m \sim p/3$ features (m is the number of features randomly picked in each tree and p is the total number of features in the training set).

To show how these compare with a basic fully grown tree, we report here a bar chart (Fig. 19) showing the MAE on both training and validation sets. As you can observe, while the MAE for the training dataset using a single decision tree is zero, the errors associated with the validation set are much bigger if we compare it with Bagging and RF. This shows that building a full tree without variance reduction techniques leads to overfitting.

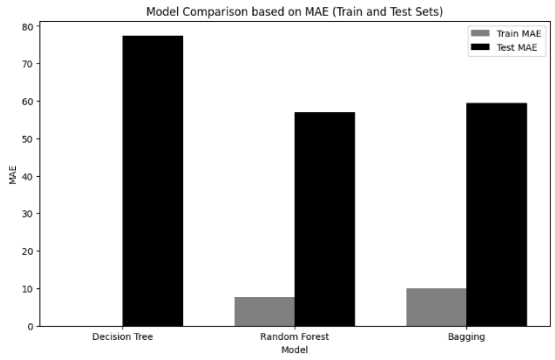


Figure 19: Comparison of MAE for training and test dataset using DT, Bagging and RF.

Before proceeding with variance reduction techniques, we performed a CV on a single decision tree regressor which will be used as base reference model.

The parameters we optimized are *min_samples_leaf* and *min_samples_split*. The first parameter is used to define the minimum number of samples required to be present in a leaf node, which prevents the tree from splitting too deeply creating small and meaningless splits with few samples data. The second parameter deals with a similar problem, but for internal nodes, which prevents overfitting internally. Another key parameter is the tree depth, (*max_depth*) which has been fixed to be fully grown as we need to minimize the bias term before the variance reduction step.

From now on the parameters used are summarized in the following table.

Max_depth	*min_samples _leaf*	*min_samples _split*
None (fully grown)	6	10

Table 2: Best parameters for fully grown Decision Tree Regressor.

Bootstrap Aggregation (Bagging)

After cross-validating the best decision tree regressor we proceed to testing the application of Bagging. As said before, the goal is to create B random subsets, train trees independently and averaging out the results into a single prediction. When using bootstrap aggregation, each subset will contain around 67% of the original training data, while the remaining 33% of the data is used for validation. The error calculated in the validation step is known as the Out-Of-Bag (OOB) error. The OOB measures the model's performance on unseen data without requiring a separate validation dataset. This allows us to train the model on the entire dataset without requiring an ex-ante train-validation split.

Another important feature to specify is that increasing the number of subsamples (B) does not lead to overfitting but only to error stabilization. Hence, the only trade-off to perform here is between OOB error and computational cost.

Using the decision tree defined before we decided to fit the full training dataset (without ex-ante splitting) to test the trend of OOB vs B. Moreover, we kept track of the computational cost which has been proxied by the fitting time for each value of B. The results of this search are reported in Figure 20. As you can see, we can observe an important drop in the OOB error (expressed as MAE) at early stages, and then accuracy of the model finds a plateau. Taking this chart into consideration we decided that the optimal trade-off between accuracy and computational cost (CC) is obtained by fixing B to 50, for which the OOB-MAE error is 26.11. Increasing B to 100 for example will double the computational cost and improve accuracy only by 0.91%.

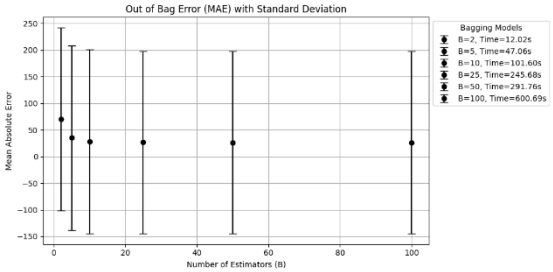


Figure 20: *OOB MAE with relative standard deviation from a Bagging model applied to the full dataset as a function of the number of estimators.*

A negative aspect of ensemble methods is that, due to the random generation of B trees, we lose interpretability. However, such methods offer us a key metric: Feature Importance. Feature Importance (FI) measures how valuable each feature is for making predictions across all subsets in the ensemble, reflecting the contribution of each feature to the final output prediction. This importance is often assessed using permutation importance, where the values of a predictor are randomly shuffled to break their association with the target variable. The resulting decrease in model performance indicates the predictor's significance, providing valuable insights despite the lack of interpretability inherent in ensemble methods.

Hence, for the sake of completeness we explored the effect of dropping the 20 less important features (this means dropping more than 50% of the predictors space) and checked how the accuracy changes. Figure 21 reports the same plot we showed before. We can observe that, while

the OOB MAE is left quite unchanged, (for B = 50 the MAE is 3.44% higher with respect to the utilization of all the features), the CC has been reduced by more than 50%.

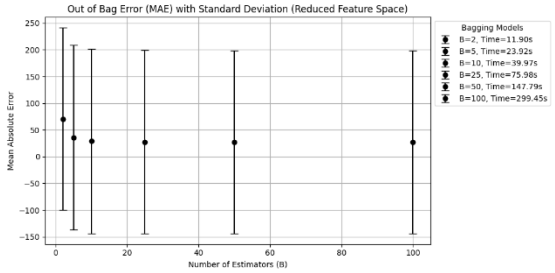


Figure 21: *OOB error as a function of number of estimators when training a Bagging model on the feature reduced dataset.*

Random Forest (RF)

The greatness of RF model lays in the second layer of randomness it introduces by locally reducing the feature space in each tree. To see this effect, we decided to register the OOB score for different estimators for a Bagging and RF model with the same parameters (other than *max_features*).

Figure 22 illustrates OOB error rate (calculated as 1-OOB Score) for both Bagging and Random Forest (RF) as a function of the number of estimators (B). The plot highlights that the use of Random Forest consistently achieves a lower OOB error rate compared to Bagging. This performance improvement can be attributed to the reduction in variance achieved by Random Forest. Specifically, RF effectively reduces the correlation between individual trees in the ensemble, leading to a more robust and less overfitted model.

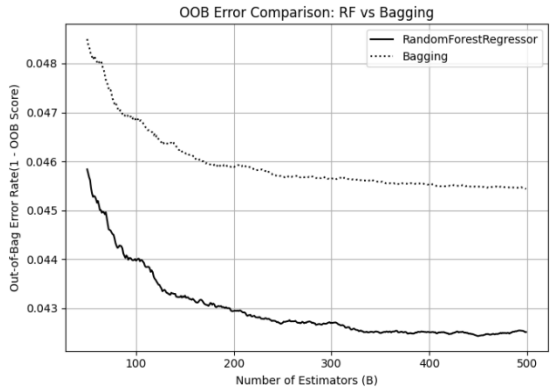


Figure 22: *OOB error rate comparison between Bagging and RF as a function of the number of estimators.*

To check the difference between the two in a feature reduced environment we repeated the process dropping the same features eliminated during Bagging. Moreover, in this case, we decided to report the OOB MAE and the fitting time for both models as a function of the number of generated subsets (B). It is interesting to notice that as the number of estimators increases, so does the computational cost (proxied by fitting time). This confirms, again, the importance of B in the accuracy-cost trade-off.

The results of this search are reported in Figures 23 and 24.

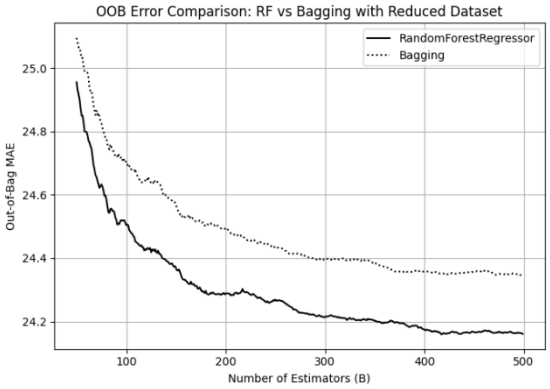


Figure 23: *OOB error rate comparison between Bagging and RF as a function of the number of estimators with the reduced dataset.*

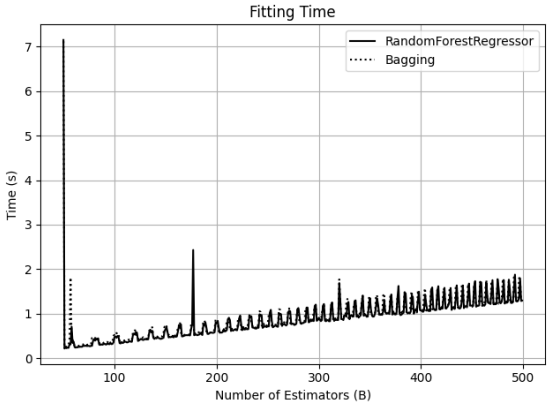


Figure 24: *Computational cost (proxied by fitting time) comparison between Bagging and RF as a function of the number of estimators with the reduced dataset.*

To conclude, a cross-validation has been performed to find out the best tuning parameters for the RF. This time, differently from the bagging case, we decided to use the entire feature space (in terms of number of predictors), but, indeed, we conducted the optimization on two different datasets in terms of number of observations. The first cross validation has been performed on the entire dataset while the second one on a subset of this one from which outliers have been dropped (see Section 3E for more information).

The best OOB MAE, Training MAE and fitting time for both the dataset are reported in Table 3. As we can observe the dimensionality reduction cut down the fitting time by 44% with a small difference in accuracy.

Dataset	OOB MAE	Training MAE	Fitting Time (s)
Full	22.81	8.38	77.23
Outliers-Dropped	23.03	8.48	43.05

Table 3: OOB MAE, Training MAE, and Fitting time (s) for RF when tested on the full and the outliers' removed datasets.

H. Recurrent Neural Networks

Given the time series nature of the target variable, it is especially important to find ways to model the sequentiality and seasonality. This way, we propose to construct a model based on recurrent neural networks. In general, neural networks are powerful because of their flexibility and high predictive power. The main idea lies in the possibility of the model creating different hidden layers. In each of them, the features are non-linearly combined in many hidden variables, which can be defined as:

$$Z_j = g\left(b_j^{(1)} + w_j^{(1)} X\right),$$

For all j's from 1 to m, where m represents the number of hidden variables for the layer.

Sequentially, these hidden variables are combined in hidden layers aiming to find complex relationships through g(x) RELu activation functions. The arguments for the activation function are linear combinations of the features in which each of them has a specific weight W. This way, we get the following structure of the neural network:

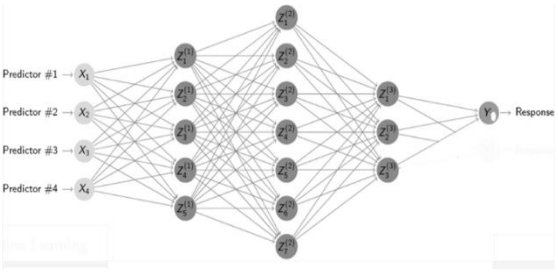


Figure 25: Structure of the Neural Networks.

For our specific case, the RNN has the particularity that the features have a recurrent connection, so each hidden layer is connected not only to the next one but also to itself by the past values. We proceeded to use *MinMaxScaler* to scale all the features and the target variable and ensure all the variables contribute equally to the learning process. Then, we added a new dimension which is the number of timesteps relevant for the model, that we set to 1 meaning that we expect minimal sequence memory.

Next, we constructed three layers, starting from a simple recurrent neural network with 64 units; moreover, to prevent overfitting, we added a dropout layer that deactivates randomly 20% of the neurons in the training phase; finally, we added a fully connected layer using a ReLu function. After that, we compiled our model, using MAE as the metric of performance evaluation. During the training phase, we set the number of epochs to 100, but we use early stopping with patience parameter of 5 epochs to prevent overfitting and to reduce computational cost.

Even with these adjustments, the computational cost stays high, especially when thinking about tuning parameters through cross validation. In that sense, we decided to apply randomized cross-validation through *RandomizedSearchCV*. Nevertheless, due to package compatibility constraints, we manually programmed a randomization method, instead. We defined a grid for the main tuning parameters of the model: number of neurons, number of epochs, dropout rate, activation function and batch size. In total, we performed a cross-validation process of 30 models that randomly chose hyperparameters combination. This allows us to reduce importantly the computational cost and still find a very optimal result. In this stage, we used MAE to compare their performances and select the best parameters for the more optimal model for this method.

V. CONCLUSION

This project has demonstrated the application of various machine learning methods to forecast the number of bike rentals for Mule, a newly established bicycle rental company in Geneva. By leveraging temporal and meteorological data, we tried to develop models that balance accuracy, computational efficiency, and interpretability to address the company's forecasting needs.

The exploratory data analysis (EDA) uncovered key insights about the data, including the significant impact of weather, time-of-day patterns, and seasonality on bike rental demand. These findings have been used for our feature engineering process, enabling us to create meaningful predictors such as interaction terms, cyclical transformations, and lagging features, which enhanced the models' predictive capabilities.

Several machine learning models were implemented and rigorously tested, ranging from simple linear regression to advanced ensemble methods like Random Forest and Gradient Boosting, and even Recurrent Neural Networks for modeling time series datasets. While simpler models like linear regression provide interpretability, they are unable to capture complex relationships resulting in low performances. Ensemble methods, particularly Random Forest, struck a balance by reducing variance and achieving strong performance, as indicated by low out-of-bag (OOB) error rates. Recurrent Neural Networks (RNNs) showcased the potential for capturing sequential

patterns but came at a significantly higher computational cost.

Figure 26 reports a comparison of all the models that have been implemented throughout our research. We observe that the lowest train and test error have been obtained by bagging and random forest models. Nevertheless, these errors were not reflected when uploading our results into Kaggle platform. This may be caused by OOB scores that are based on a random selection basis which is not reflected in the time-series structure of the data we aim to predict. In this sense, the model that gives us the best results for unseen data in our context is Gradient Boosting, which has been selected as our final and definitive model. Using this model we have predicted the series for the next three models, which can be observed in Figure 27.

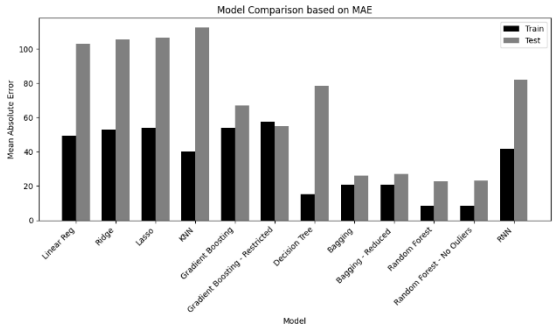


Figure 26: Comparison of train and test MAE of different models.

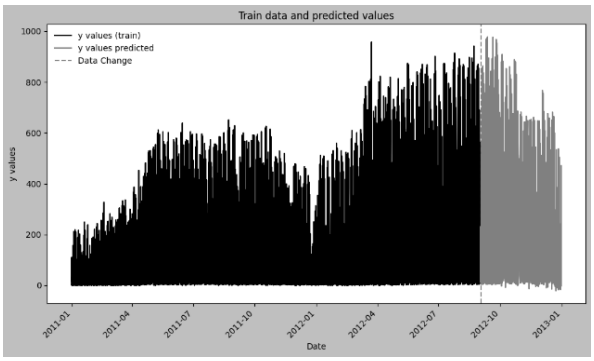


Figure 27: Prediction of next three months using Gradient Boosting.

Through our analysis, we identified the importance of key features, such as temperature, windspeed, and hourly trends, and leveraged feature importance metrics to simplify models without compromising (too much) accuracy. This allowed us to propose solutions that optimize computational efficiency while maintaining predictive power.

From a business perspective, the findings suggest clear action points for Mule.

Firstly, the steady and consistent increase in bike rentals highlights growing demand and suggests the need for future expansion. Shifting the market match between demand and supply, at constant operational costs, can be a great opportunity to enhance profitability, particularly if the bike rental sector presents positive returns to scale.

Secondly, the discussed seasonality patterns suggest the need for strategic planning, such as maximizing the fleet availability in hot seasons. During winter season, instead, considerable as the closing quarter of the short business cycle, can be utilized for checking processes such as bike repairing, stations maintenance and overall operational steps aimed at ensuring full and excellent conditions for the next cycle. Another seasonal opportunity could be offering seasonal promotions during summer, while introducing winter-specific incentives like discounts or enhanced bike usability features during colder periods.

Time-of-day usage patterns further provide valuable insights. Considering the working day peaks due to commuters and the leisure-driven weekend ones, Mule could leverage this information to tailor several initiatives. For example, they could introduce commuter-focused pricing plans on weekdays and leisure-oriented promotions on weekends. Furthermore, partnership with event organizers or tourist agencies could enhance the weekly usage and smooth it down into a wider distribution.

Weather conditions also play a significant role in influencing rentals. Warmer temperatures drive increased usage, windy conditions, however, consistently correlate with decreased demand. These trends highlight the chance of offering dynamic pricing models or weather-adaptive incentives, such as discounts on windy days or promoting the comfort of riding during moderate temperatures. Overall, these insights underscore the potential for targeted interventions to maximize customer engagement, optimize resource allocation, and drive profitability

Regarding the data collection process, Mule could focus on incorporating new variables into the dataset. For instance, taking count of the number of bikes available at each hour as a variable could enhance predictions and give valuable insights for the business. As we already stated, during peak periods, unmet demand might indicate an opportunity for the company to optimize their availability and improve profitability.

Also, knowing the location of the store would allow us to create a variable that measures approximately how many people are doing activities around the store at different times of the day. As a proxy, we could use, for example, parking occupancy rate, so we know the demand intensity that we would wait for the company.

Finally, adding variables related to traditional transportation costs, such as fuel prices, could provide information about the demand for substitute goods. This would allow the model to better account for

macroeconomic factors influencing rental behavior and improve its forecasting accuracy.

In conclusion, this project provides Mule with robust forecasting tools and actionable insights to support operational planning and financial stability. By optimizing its rental fleet, adjusting to weather and seasonal patterns, and targeting customer segments effectively, Mule can position itself as a leader in Geneva's sustainable urban mobility landscape.

AI Assistance Statement:

Throughout this project, AI tools, particularly ChatGPT, were utilized during the analysis phase to assist in debugging the code and generating insightful graphs and figures. Nevertheless, all written content, analyses, and core project contributions were independently developed by the group members, ensuring their originality and integrity.