Hertie School

Bike Sharing Rental Prediction

with Data from Capital Bikeshare (Washington, D.C.)

Mona Borth
Linus Hagemann
Luis Windpassinger

GRAD-C24: Machine Learning

Dr. Chris Carmona

GitHub

Introduction

Bike sharing systems have become a key component of urban transportation networks, including in Washington, D.C., where Capital Bikeshare operates one of the largest programs in the United States[1]. With thousands of daily trips, these systems offer affordable, low-emission mobility options that reduce private car reliance and contribute to lowering urban traffic emissions. In the context of the climate crisis, the importance of sustainable and emission-free transport modes such as cycling has gained increasing attention in both political and public spheres, intensifying the push to optimize and expand bike sharing services.

This project leverages openly available data to develop predictive models of system demand in Washington, D.C. Specifically, we focus on forecasting departures at the station level, as these represent the most immediate and operationally relevant available indicator of user demand. Predicting departures allows for fine-grained, actionable insights that can directly improve system efficiency by informing fleet management, bike redistribution, and rebalancing operations. Accurate forecasts ensure that stations maintain adequate bike availability, reducing the likelihood of stockouts and improving the overall user experience. Beyond daily operations, modeling departure patterns provides critical input for longer-term strategic planning - for business operations as well as municipal decision-makers. By identifying spatial, contextual and temporal demand trends, the models can highlight where existing infrastructure is under strain or where additional investment could yield the greatest benefit.

Incorporating weather data provides an additional layer of contextual information that can greatly enhance the accuracy of our demand predictions. Weather patterns often correlate with changes in cycling behavior - on rainy or cold days, bike sharing demand typically decreases, while mild weather or sunny days encourage more trips. By integrating weather as features, we can refine the model's ability to predict departures, ensuring it accounts for all relevant factors. This ultimately leads to more accurate forecasts that support better decision-making.

The remainder of this paper is structured as follows: We begin by detailing the data collection and preprocessing steps, outlining how our three datasets were obtained, cleaned, and merged into a unified dataset. This is followed by an exploratory data analysis that examines the target's distribution, reveals non-linear feature relationships, and addresses multicollinearity through informed feature selection. We then introduce engineered features designed to capture temporal dynamics and contextual signals relevant to the prediction task, clustering stations according to these. In the methods section, we present a range of models, from baselines and Linear Regressions to advanced tree-based and ensemble methods, including Random Forests and XGBoost. We compare model performance across citywide and clustered subsets, including pooled, unpooled, and Bayesian semi-pooled strategies. The paper concludes with a discussion of key findings, limitations, and directions for future work.

[1] https://grist.org/article/2010-09-20-washington-d-c-launches-the-nations-largest-bike-share-program/

Data Collection and Preprocessing

Our project combines three separate data sets:

1. Data on trips[2] in the Capital Bikeshare system throughout the year 2023
2. Data on stations[3] and their properties in the Capital Bikeshare system
3. Weather data throughout the year 2023 for each station in D.C[4].

Trip Data

Capital Bikeshare provides all trips taken in their system as open data, which is not a given for many bike sharing companies. Therefore, this played a key role in our decision to use data from D.C. for this project, despite the drawback of our having no particular domain knowledge when it comes to the city. It is also worth noting that we chose to use data from 2023. While this comes with some caveats (see the section on combining trips and stations below), we looked at the trip data available for different years and noticed the availability of start- and end-stations of trips dropping drastically for later years. One possible reason for this could be a change in the system towards a “free-floating” fleet, in which the role of stations is less pronounced, instead allowing users to park the rented bikes nearly anywhere. As our prediction task only makes sense at a station level, however, this made the 2023 data most useful for our project.

The trip data is available as .csv files, separated by months, but sharing a common structure so that we could easily combine them to build a table containing one row for each trip taken throughout the entirety of 2023, yielding 4.467.334 in total. We used the following features of the available trip data: Timestamps for both start and end time of a trip, as well as the names of the start and end-stations (as strings). For the chosen timeframe, no unique identifier for individual bikes was available.

Stations

Additionally, a single table containing all stations in the system is available. Unfortunately, this data does not come in a versioned format, but only the most current version is available. This means that stations that might have existed in 2023, but discontinued since then, will be recorded as receiving trips, but we won’t know any details on these stations, such as their geolocation or other station-specific properties.
As there is no stable identifier for each station available to us, we needed to rely on the names of the stations. This also means that, in addition to the problems pointed out above, our matching approach outlined below is not able to account for changes in the names of stations.

[2] https://capitalbikeshare.com/system-data
[3] https://opendata.dc.gov/datasets/a1f7acf65795451d89f0a38565a975b3_5/about
[4] https://open-meteo.com/en/docs/historical-forecast-api

Weather Data

As alluded to above, especially for more casual bike-riders, such as the usual customers of bike-sharing systems, the current weather conditions likely are one of the deciding factors when choosing to take a bike as opposed to other forms of transportation.

We collected hourly weather information for each station in the Capital Bikeshare network, based on their geo-locations. The historic weather data was retrieved with a custom script from the open-meteo API.

Merging and Preprocessing of the Different Data Sources

Our goal was to create a single data frame containing information about the usage of each station in the system on an hourly basis for the entirety of 2023.

First, we aggregated the trips by start- and end-hour, grouping on the respective start- and end-station. There was a small number of trips in our data that did not start or end at a recorded station - either because the bike was actually returned at a non-station location or because of system errors. Either way, we discarded these trips, resulting in 3864783 trips remaining.

Next, we joined these aggregated trips together with the general information about stations. As we did not have unique identifiers for the stations, we simply chose to rely on basic string matching based on their name. As previously mentioned, this led to us dropping all trips that started or ended at a discontinued station or at a station that has changed its name since 2023. We ended up with 3.632.735 trips that we could match to 738 different stations, meaning 6% of trips had to be discarded.

The weather data we retrieved from the API included categorical information according to the WMO weather interpretation codes[5]. There were more than 25 categories present in our initial data. Some of these codes are very granular (e.g., different codes for slight, moderate, and violent rain showers). In order to keep our feature space at a reasonable size during modeling, where we will need to one-hot encode categorical features, we reassigned similar WMO codes into a scheme of 5 manually defined categories (see Appendix II).

Afterward, we could match the weather data to the corresponding station based on geolocation data.

Combining the hourly station trips and the weather data provided a bit of a challenge, as there seemed to be a problem with how the open-meteo API handled time zones. This led to us missing 9 hours of weather data on the 31.12.2023, from 3 pm onwards. After manually adapting for the time-zone difference. We validated our time-handling approach by plotting the average temperature over the 24 hours of all days, showing the lowest temperature at 6 am, which aligns with theoretical assumptions[6]. The structure of our final dataframe is exemplified in Appendix I.

[5] https://www.nodc.noaa.gov/archive/arc0021/0002199/1.1/data/0-data/HTML/WMO-CODE/WMO4677.HTM

[6] https://www.hko.gov.hk/en/education/weather/sunshine-and-uv/00692-What-time-in-a-day-is-highest-lowest-air-temperature.html

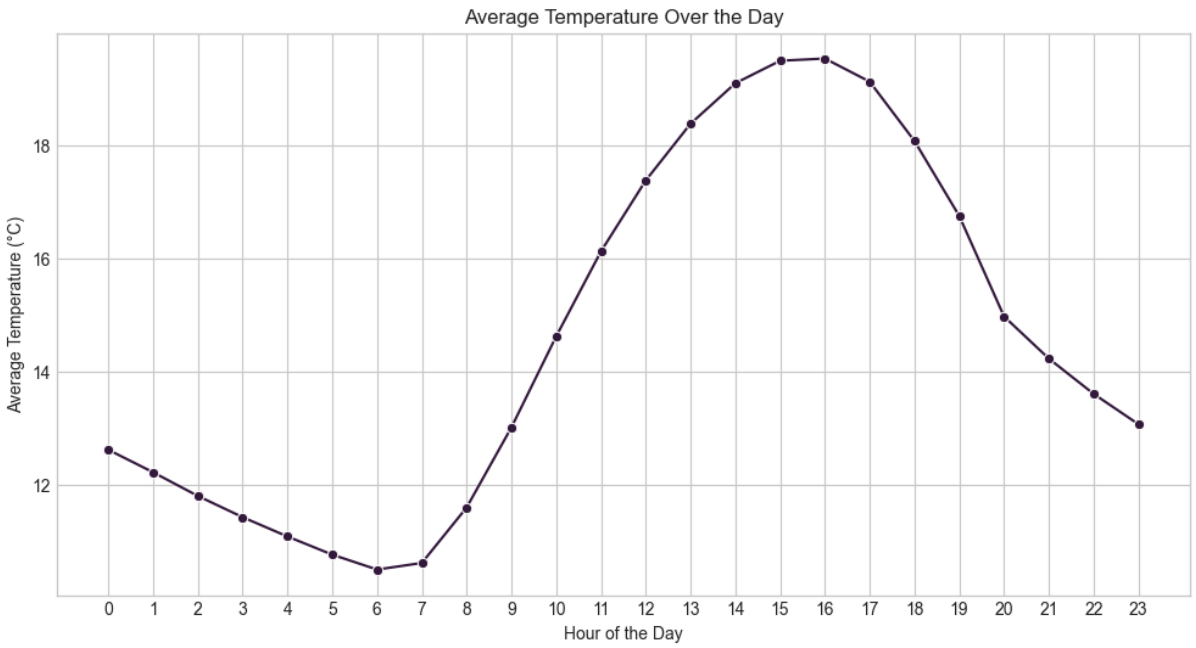


Figure 1: Average temperature per hour of day (mean over the entire year).

Exploratory Data Analysis

Target Distribution

Our analysis of the target variable (departures) reveals a highly right-skewed distribution of hourly bicycle departures. As can be seen in Figure 2, the frequency distribution exhibits a pronounced concentration near zero departures (~77% of all rows represent 0 departures), with rapidly diminishing frequencies as departure counts increase. This pattern indicates that the majority of station-hour combinations experience minimal activity, while a small subset of observations accounts for significantly higher departure volumes (exceeding 10-20 departures). This zero-inflated, long-tailed distribution suggests pronounced temporal and spatial heterogeneity in system utilization, characteristic of urban bike sharing networks with distinct peak usage periods. This pattern differs at cluster level, which is further elaborated on in cluster-specific sections in the following.

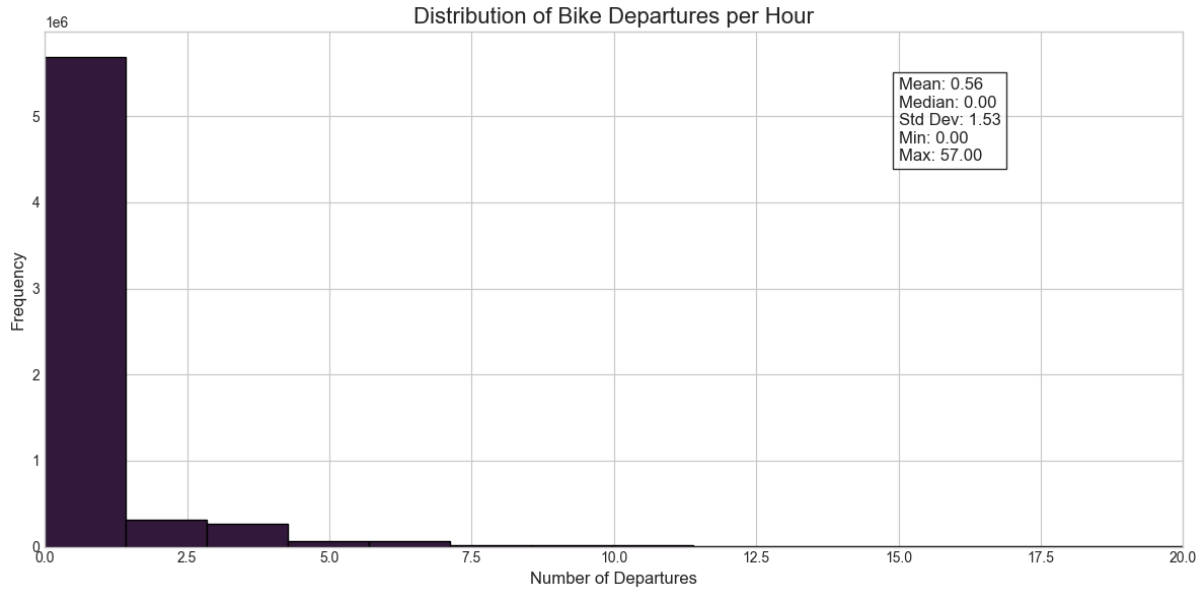


Figure 2: Frequency distribution of the target variable “bike departures”.

An overview of core metrics of the target - both on city- and cluster-level - can be found in Appendix III.

Temporal analysis of aggregate departures reveals a distinct bimodal distribution across hours of the day, providing further context for the previously observed skewed distribution. The histogram exhibits pronounced commuter peaks at 8:00 (approximately 240,000 departures) and 17:00-18:00 (approximately 370,000 departures), with the evening peak substantially exceeding the morning peak. System utilization remains at moderate levels during midday hours (9:00-16:00) before declining significantly during late evening and early morning hours (22:00-5:00).

This pronounced temporal pattern explains much of the zero-inflation observed in the overall distribution, as the majority of zero or near-zero departure counts occur during predictable low-activity periods. The temporal concentration of departures during commuting hours suggests that time-based features are essential predictors in our modeling approach in order

to accurately capture both the cyclical patterns and the significant differences between peak and off-peak periods.

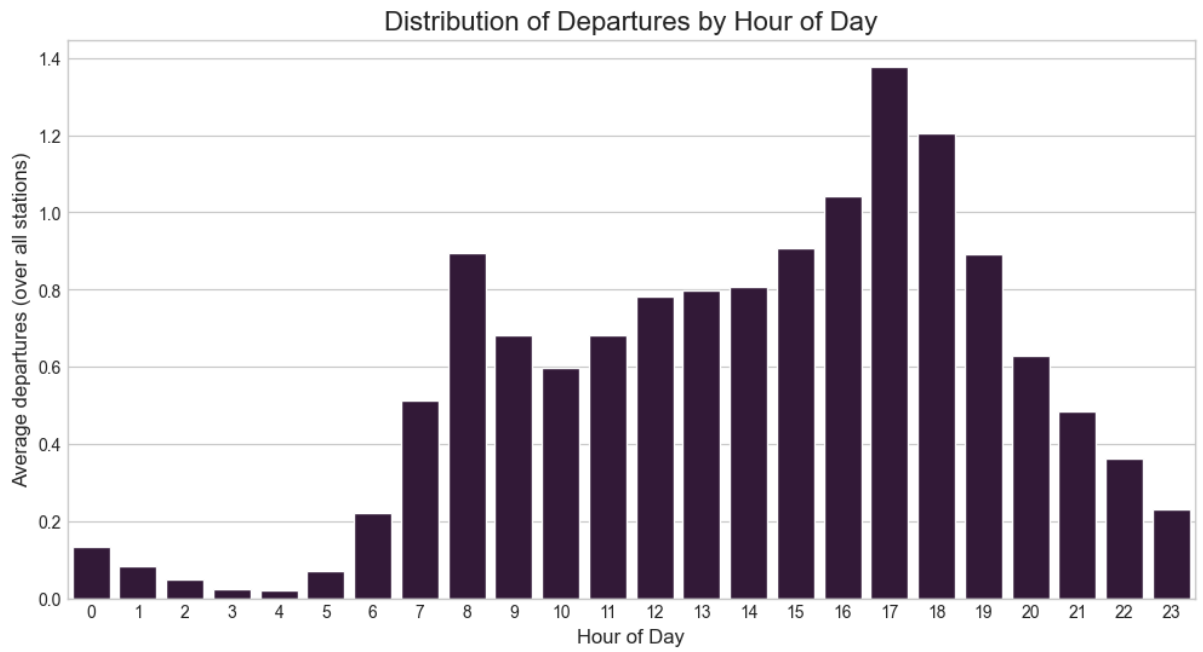


Figure 3: Distribution of departures by hour of day (mean across all stations).

Non-linear Relationships of Features and Target

Our analysis of bivariate relationships between selected numerical features and the target (departures) - illustrated by the array of scatter plots below - reveals complex non-linear patterns that will likely be inadequately captured by simple linear models. Temperature exhibits a notably parabolic relationship with departures, where ridership increases with temperature until approximately 25°C before declining at higher temperatures - suggesting optimal cycling conditions within a specific temperature range.

Precipitation and snowfall demonstrate threshold effects, with sharp declines in departures once precipitation exceeds minimal levels, indicating categorical rather than continuous influence. Wind gusts show a parabolic distribution, with moderate gusts associated with higher departure counts than either calm or strong wind conditions. Cloud cover displays only a minimal discernible pattern, suggesting limited direct influence. The selected spatial variable (latitude) reveals a distinct non-uniform distribution of departures across locations, with concentrated activity in central areas of the city. This could be the result of the common grid-like structure of US cities.

These non-linear relationships underscore the importance of employing modeling techniques capable of capturing complex interactions and threshold effects, such as tree-based methods, rather than relying solely on linear approaches that would miss these nuanced patterns.

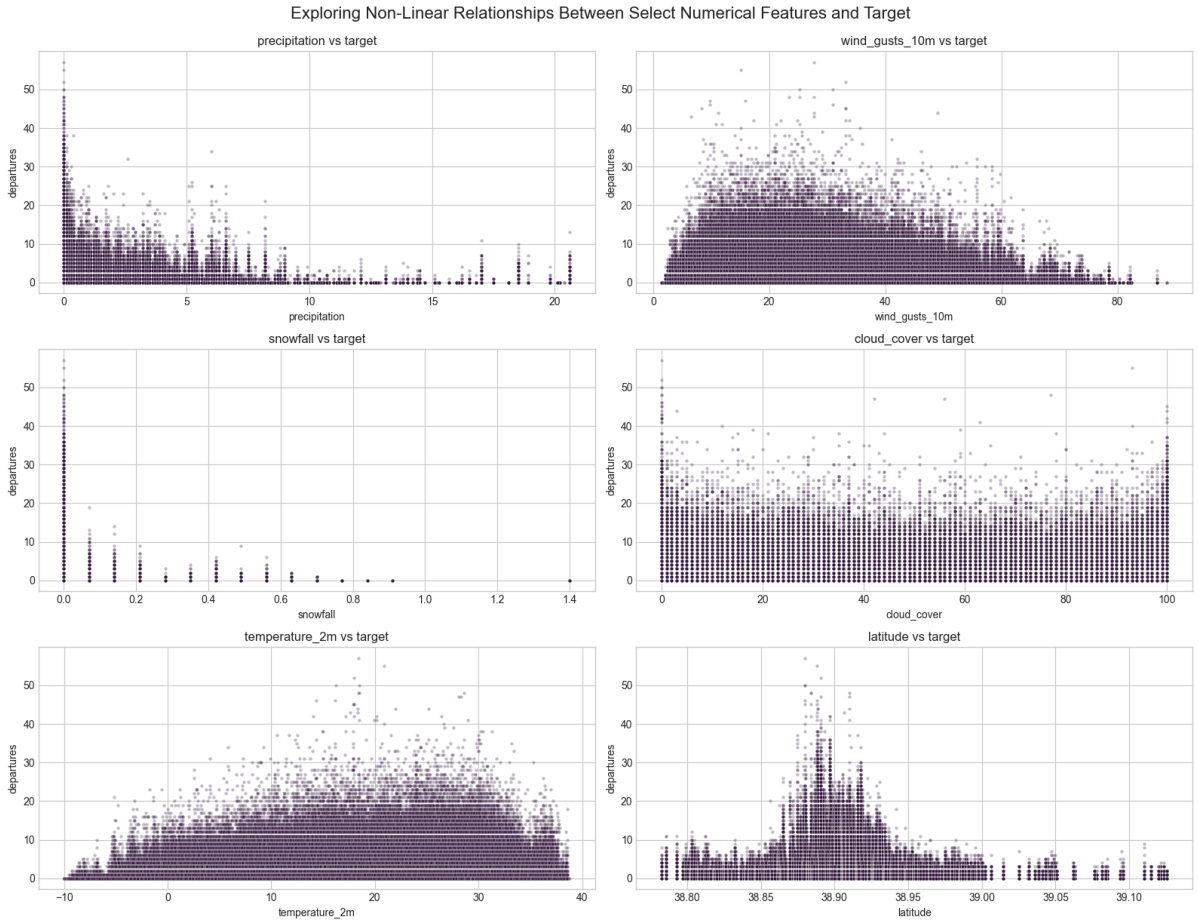


Figure 4: Scatter plots of six selected features with the target variable.

Feature Selection - Collinearity

An analysis of feature correlations revealed considerable collinearity among several features, particularly among time- and weather-related variables. Using a Pearson correlation matrix computed on all numerical features - excluding those engineered solely for clustering - we observed high pairwise correlations that could compromise model interpretability and inflate variance in parameter estimates for certain models, or lead to overfitting and unnecessary complexity.

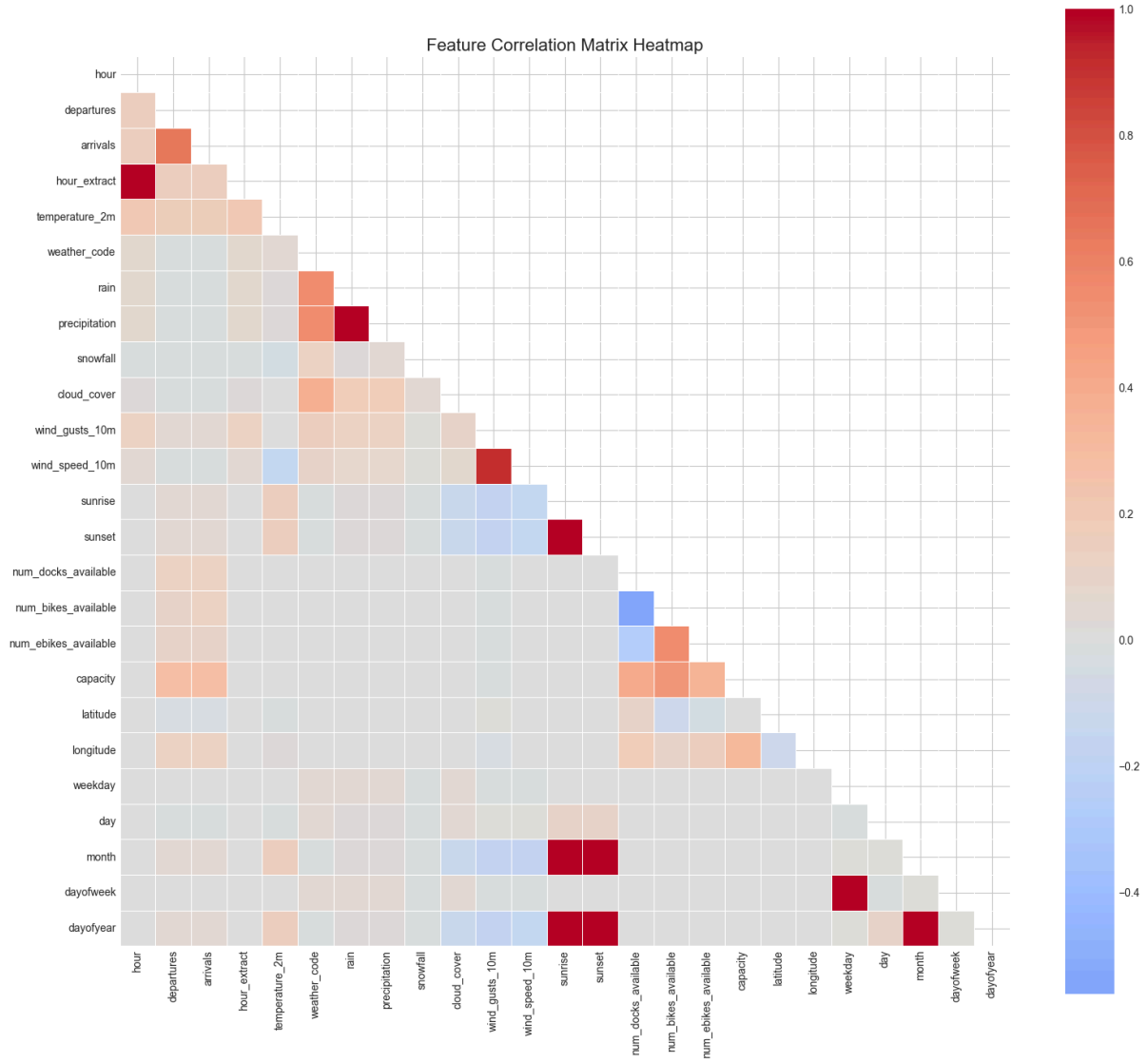


Figure 5: Correlation Matrix (Pearson) between all features (main diagonal & redundant half removed).

To address this, we performed a targeted feature selection process. The following variables were excluded from the training data based on domain knowledge, statistical redundancy, and relevance to the prediction task:

weather_code
To reduce granularity, we decoded the raw weather codes as described above.

timestamp
We dropped raw timestamps in favor of engineered time features such as hour, day, month, and dayofweek, which provide more interpretable and structured temporal patterns relevant for modeling.

precipitation
This variable showed near-perfect collinearity with both rain and snowfall, which were retained as they offer more specific information on the type of precipitation, potentially more relevant to user decisions.

sunrise and sunset
These were excluded because their influence is already indirectly captured through engineered features like time of day and date-based seasonality, which better align with behavioral patterns.

wind_gusts_10m
This metric was highly correlated with wind_speed_10m. We retained the latter, as it is the more commonly reported measure in weather apps, and likely more familiar or impactful from the perspective of a cyclist's decision-making process.

In addition to these, several other columns were removed, including: **station_name** (non-numeric and not generalizable), **capacity** (potentially correlated with station size but not central to our prediction goal), as well as **year**, and **dayofyear**, which were either redundant or already captured through more refined temporal features.

Feature Engineering

Custom Computed Features

To better capture temporal and behavioral patterns in departure volumes across station clusters, we engineered several binary features informed by both usage data and domain knowledge. **isHoliday** flags official holidays in Washington, D.C., which likely indicates reduced commute activity and more irregular usage spikes. **workhours** and **commute** segment the day into periods of structured movement, with **commute** targeting peak hours (6-10h & 15-19h between Monday and Friday) and **workhours** covering the broader 8-16h between Monday through Friday range - both of which are known to drive usage in weekday data. **free** isolates off-hours and weekends, capturing recreational or ad-hoc use. **night** marks low-activity windows (21-6h), where ridership declines. Lastly, **weather_cluster** condenses raw WMO weather interpretation codes into four interpretable categories.

Time Features

Since the training data comes in the shape of a time series, we need to take into account the special characteristics of features encoding time when passing them to any learning algorithm. One key aspect here is the cyclical nature of time variables: Hour 23 and hour 0 should be as close together as hour 1 and hour 2 are. A technique that can encode this relation is transforming time features with trigonometric functions, essentially mapping the clock (or the year, or week, etc.) onto the unit circle. Another possible approach to time variables is treating them like any other categorical variable. Here, one would argue that order plays much less of a role. To find the best method of encoding time features for each model we used, we decided to include both approaches in our grid search to compare their performance. For this, we also included the option of treating the time features as numeric, in case that would yield better predictions, as we are more concerned with prediction quality than with the explanatory power of our models.

We decided not to consider other typical methods of engineering time features (like rolling averages, or lags), because while the time-series nature of our data was a fact we wanted to take into account appropriately, it was not the main focus of this project.

Regarding the non-time-related features, we used one-hot encoding for the categorical features and standard scaling for the numerical ones.

Methods & Models

As we are using data with a time ordering, we cannot randomly select observations as test data, as is usually done in a train-test split. Instead, we need to preserve the order of data, and make sure any test observations are chronologically later than the training set. This is important to ensure that no information “leaks” from the training into the test set, and we get accurate test error estimates. We implemented this using scikit-learn’s TimeSeriesSplit() function, which provides 5 train/test folds. The training set in each fold contains all observations from the beginning until the splitting point, and the test set consists of the following observations, with a 48-hour gap in between. The splitting point is later in time for each consecutive fold, so that later folds contain more data overall, and the training and test sets only add up to the complete data for the last fold.

To find the optimal hyperparameters for each model, we implemented a grid search using cross validation on the folds as described above. For the choice of parameters in the grid, we relied on lab code where possible, and on the scikit-learn documentation and literature where needed. Due to computational and time constraints, we did not have the opportunity to refine the parameter grids further by searching with a finer grid in the vicinity of the optimal parameter values we found.

In order to plot predictions, we created a test set with 5% of the data sampled over the whole year to simulate how the model would perform in production. Note that “year” is not a feature, so this set could technically just as well be data from the following year. This approach, however, was not available to us, as 2023 was the last year with a complete trips data set, containing start- and end-stations for each observation. When evaluating these plots, one has to take into account that they show predictions from models that were not trained exclusively on past data and then predicted exclusively on future data - therefore, there might be information flowing from the training into the test set. For example, if departures for a given day at 16h and 18h are in the training set, we cannot assume that the prediction for 17h in the test set will give us an accurate estimate of the model error. This is acceptable, however, as the focus in this step is on exemplifying a production environment and not on model selection and, for example, recognizing overfitting, for which we used the time series split.

For model evaluation, we chose the root mean squared error, since our target (departures) is numerical and the RMSE preserves scale, therefore allowing for a more intuitive interpretation than the mean squared error. For all our models, we report the root mean squared error on the test observations of the time series split (best_score from the GridSearchCV object), as well as on the training observations (mean_train_score).

Baseline

Given the nature of our data, as described in the section about Explorative Data Analysis, a model that always predicts zero departures would be correct for around 77% of our data. We consider this a reasonable baseline for any model we train: If the model is not better than always predicting zero, it does not pick up on enough of the non-zero signal in the data - and it would be unacceptable to present a model worse than this baseline to any decision-maker.

Linear Regression

We decided to include a Linear Regression model since this is the simplest possible model and therefore a reasonable starting point. This is useful to easily interpret the model and its metrics and for comparing to more complex models later on. Additionally, the interpretability lets us understand which signals in the data the model is picking up on by investigating the coefficients

We do not expect Linear Regression models to perform well for two reasons. Firstly, as described in the section about Explorative Data Analysis, the relations between most of the features and the target are not linear at all. Secondly, the model assumptions of OLS are not fulfilled in our data. For instance, the errors do not appear to be homoscedastic, are unlikely to be uncorrelated, and in some cases, we use one-hot encoded variables that are perfectly collinear. Given that the focus of our project is mostly predictive performance, we proceed to fit the Linear Regression model nevertheless, and empirically assess its performance.

Since our data has many features, especially after one-hot encoding, we additionally use regularized regression models - both Lasso and Ridge Regression.

Polynomial Regression

To account for the non-linear relationships between many of our features and the target, we included a Polynomial Regression model. However, since the expansion of the features requires a lot of memory, we were only able to run this on a very small subset of our data to present a proof of concept. We also had to restrict the degrees in the parameter grid to less than five to be able to run the model at all.

Regression Tree

Tree models seemed most appropriate to this prediction problem to us for a number of reasons. Firstly, they do not rely on functional form, so the different types of relations of our features with the target should present less of a problem. Secondly, they can account for interactions, which are very likely to occur in our setting: for example, the weather most likely matters more on holidays, when leisurely cyclists use the bike sharing system, than during workdays, when people have to get to their workplace no matter if the weather is bad. We decided to include a simple Regression Tree since this, in contrast to tree ensembles, is an interpretable model, which would be simple to explain to a decision maker, which carries some importance in policy settings. Theoretically, a simple Regression Tree is likely too simple and would underfit the data.

XGBoost

We included an Extreme Gradient Boosting (XGBoost) model, as this is often one of the leading models in prediction tasks. Since this implementation of a boosted tree ensemble is well-optimized, it is also computationally feasible to run on our machines. While its predictions are not fully explainable, we can still generate feature importance plots to obtain a general image of which predictors mattered most for the model.

Random Forest

As we include a boosted tree ensemble above, we also wanted to include a bagged ensemble. Random Forests even present an improvement over simple bagging by training each tree on a random subset of features, which seemed useful to us given the large number of features after one-hot encoding. However, training Random Forests is computationally very expensive, so we were only able to try this model on a very small subset of our data in order to present a proof of concept.

Parameter Grids

For parameter selection, we utilized a grid search together with the above-described time series split. The following table showcases our entire grid of parameters for all models. Additionally, for each model, we tried different time processing strategies: trigonometric, one-hot, and simply including the numerical information, as described above.

Model	Hyperparameter Grid
Linear Regression	n/a
Lasso Regression	**alpha:** np.logspace(-4, 4, 20) **max_iter:** [1000, 2000]
Ridge Regression	**alpha:** np.logspace(-4, 4, 20)
Polynomial Regression	**degree:** [2, 3, 4]
Decision Tree	**max_depth:** [3, 5, 10, 20] **min_samples_split:** [2, 5, 10, 15, 20]
Random Forest	**n_estimators:** [50, 100] **max_depth:** [10, 20] **min_samples_split:** [2, 5, 10, 15, 20]
XGBoost	**n_estimators:** [50, 100] **max_depth:** [3, 6] **learning_rate:** [0.01, 0.1]

Table 1: Hyperparameter grids per model.

Results[7]

Citywide Models

To begin our analysis, we generated predictions for the simplest possible, yet still interesting context. This means predicting our target variable, bike departures, aggregated over the whole city of Washington, D.C, i.e., grouping our dataset by hours and excluding any station-level information. All models improve upon the baseline, with even the worst model (Regression Tree) reaching an RMSE of 51% of the baseline model. All other models have very similar RMSE values, with the Lasso Regression achieving the best one (272.32 - 49.1% of the baseline). This is interesting, since a priori, we expected to achieve better results with more complex models like XGBoost. However, we could also observe that the Lasso model has the highest (i.e., worst) train RMSE out of all the models, so it seems like it does not overfit the training set, indicating that the other models might actually be too complex for the task and do not generalize well to unseen data.

To understand which features the models take into account and which patterns they pick up on, we analyze the coefficients of the Linear and the Lasso model. This also illustrates the effect of the L1 regularization.

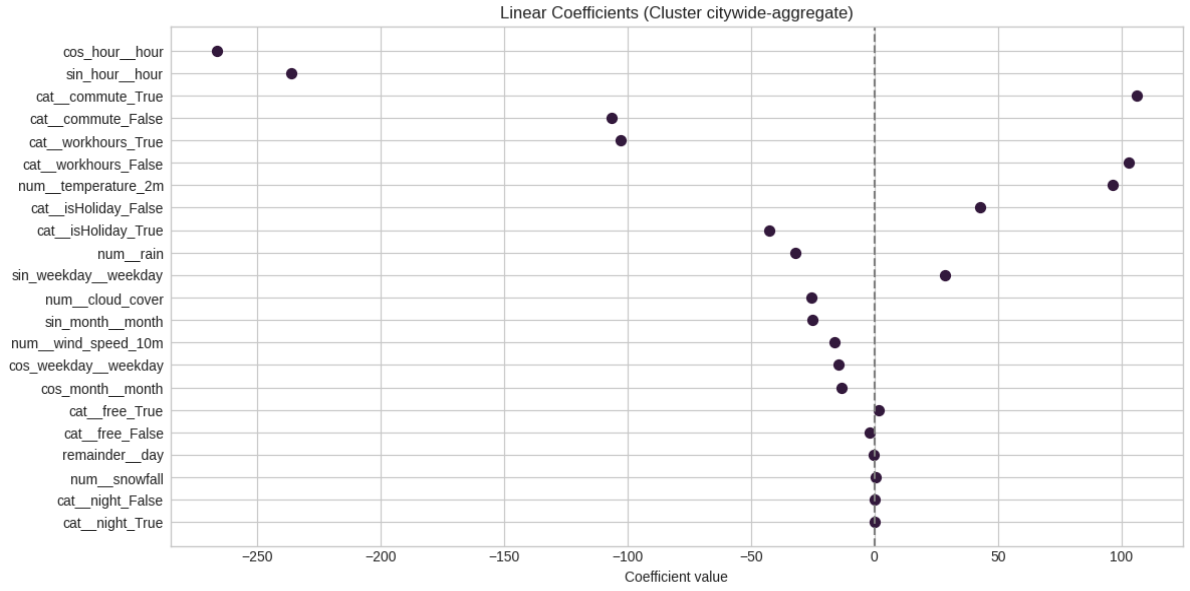


Figure 6: Coefficient Plot for Linear Regression Model for Citywide Aggregate Departures. *Time processing strategy: Trigonometric.*

[7] A subset of our results can be found in Table 2. For a complete overview of concrete parameterizations of all models and their evaluations, see this table on GitHub.

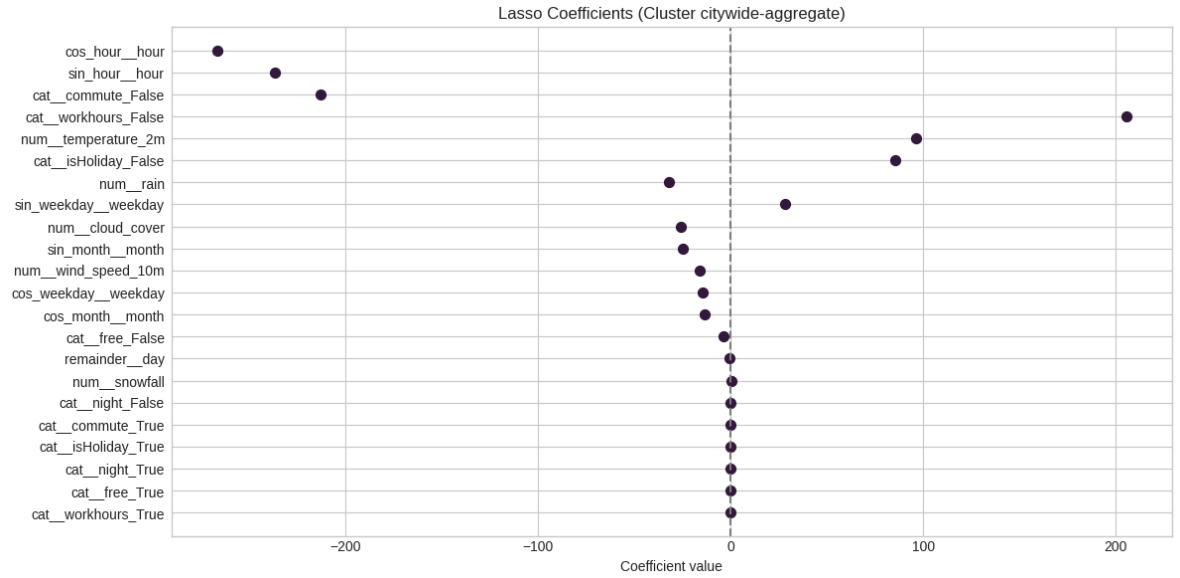


Figure 7: Coefficient Plot for Lasso Regression Model for Citywide Aggregate Departures. *Time processing strategy: Trigonometric.*

We can see that Lasso (second plot) sets the redundant columns created in the one-hot encoding to zero: **commute_true** and **commute_false** encode exactly the same information, so one of them is redundant. Analyzing the coefficients, we see that the model has learned that the time of day (**sin_hour** and **cos_hour**) matters most (measured by the magnitude of the coefficient), though the sign of the coefficient on these features is not straightforward to interpret. Furthermore, the variable **workhours_false**, indicating that it is not between 8h and 16h on Monday to Friday, has a large positive coefficient, indicating that there tend to be more departures outside working hours. Similarly, **commute_false**, indicating it is not rush hour (6-10h or 15-19h on Monday to Friday), has a large negative coefficient, so there are many departures when people arrive at or leave from work. Additionally, a higher **temperature** means more departures, and more **rain** means fewer departures, indicated by their positive and negative coefficients, respectively. The model has also learned that there are more departures when **isHoliday** is true.

To illustrate what this implies for model predictions, we investigate the predictions of the XGBoost model for the second half of December. Note that to obtain this plot, we modified the train-test split to ensure that the 5% test observations form a contiguous block at the end of the year, deviating from the practice in the rest of the report for the sake of obtaining a clear illustration.

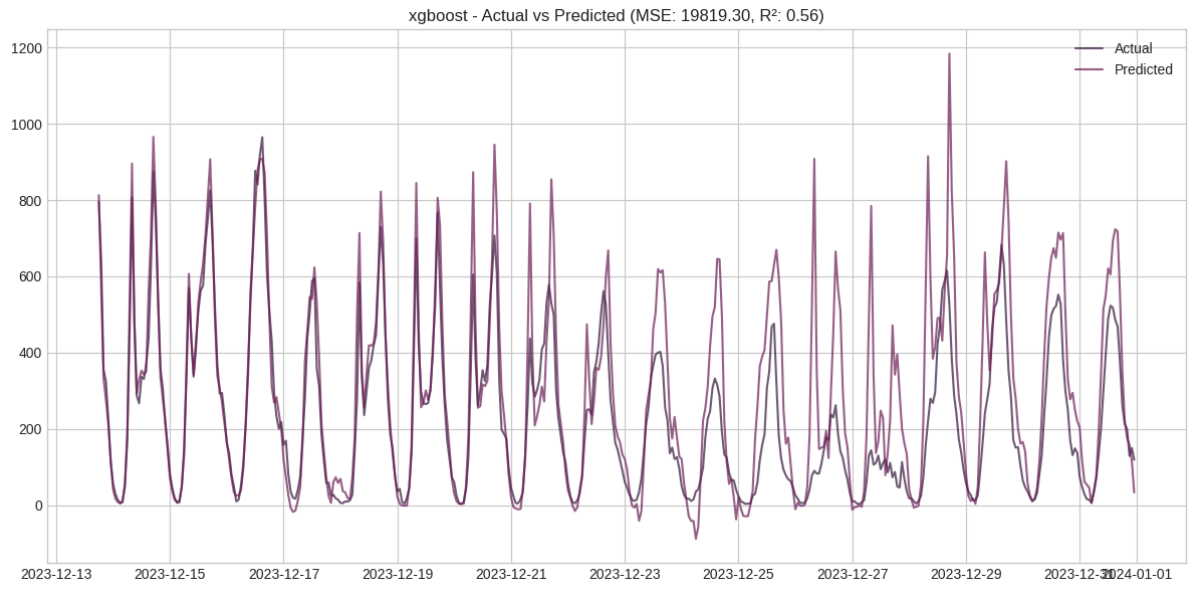


Figure 8: Predicted and actual departures for XGBoost model for Citywide Aggregate Departures, on a 5% test set at the end of the year. *Time processing strategy: Numerical, learning_rate: 0.1, max_depth: 6, n_estimators: 100.*

We see that the model performs quite well, with larger errors over the Christmas days. This is expected given that there are no or very few features in our data that the model could use to understand that this is a time when people are more likely to stay inside, take time off from work, or travel to other states.

We can also see in this plot that the model has likely learned that commuting only happens on days that are not weekends or holidays, since for weekdays (e.g. 18th-22nd) it shows two peaks in departures (for morning and afternoon commute) - for example, but for weekends (e.g. 16th-17th) and holidays (25th, a Monday) it shows only one. The model picks up these intricate patterns quite nicely.

Clustering

As our project relies on the difference in usage patterns across stations, in an ideal case production scenario, each individual station's demand would be predicted using a separate model. However, with limited time and computing resources, this was out of the scope of this project. In order to make the prediction task more manageable, we decided to cluster stations into groups, reducing the scope to one model per cluster. As a positive side effect, we could retain all the data, instead of omitting most of it, if we had selected only a subset of clusters or stations instead.

Our underlying assumption for this approach was that within bike sharing networks, there are usually different “types” of stations: Consider, for example, stations that are located very close to tourist attractions, recreational parks, office-buildings or nightclubs - all of these will show very different usage patterns, e.g. more rides on free days with good weather to and from the park and more rides during typical commute hours to and from the office district.

With limited domain knowledge about D.C. and in the absence of further geocoded information, we chose to engineer features that capture these differences in usage patterns, and then apply a clustering algorithm on the stations, utilizing these features.

We calculated the average number of departures and arrivals per station in different scenarios, such as on weekends, at nighttime, during typical commute hours, etc.

After having engineered more than 25 features this way, we applied a Principal Component Analysis on them to reduce the space in which the clustering algorithms needed to work to obtain more meaningful results. We chose 6 principal components that together captured more than 95% of the variance, while still drastically reducing the space spanned by our features.

We then tried applying different clustering algorithms to the data. For each algorithm, we plotted the resulting clusters in the space spanned by the Principal Components, varying the axes plotted against each other in order to visually assess whether the clusters make sense in the dimensions of our problem. We also inspected the number of clusters and distribution of stations between these: In order to reasonably mitigate our computational constraints, we needed a number of clusters that would allow us to run models for each cluster individually, while the number of stations per cluster should not vary too much (e.g., a cluster with only 3 stations versus a cluster with more than 100 stations).

We tried, in this order: K-Means, DBSCan, HDBScan, and OPTICS, running the algorithms with a variety of different parameters. Note that we began with trying the simpler algorithms (e.g. K-Means), increasing complexity each time we found the results to be non-satisfactory. The reasons for the results being non-satisfactory varied by algorithm: For K-Means and OPTICS, the sizes of the clusters were extremely unbalanced, with most of the clusters containing less than approximately 20 stations. DBScan and HDBScan did not identify clusters with face validity when inspecting them in the space spanned by the principal components.

At this point it is worth noting that we had unusual, and, most importantly, conflicting requirements for our clustering solution: While, in principle, we wanted the clusters to capture differences between the stations as well as possible, we also utilized them as a means to reduce the compute necessary for modelling. Hence, while acknowledging that this negatively influences the explanatory power of the clusters, we were willing to enforce some restrictions on our clusters. We will discuss this in more depth under Assumptions.

After the aforementioned algorithms did not provide satisfactory results, we applied a constrained K-Means algorithm[8] on our data. This algorithm works just like the classic K-Means, but adds another optimization problem in the form that the number of clusters, as well as the allowed minimum and maximum number of entities per cluster, is specified by the user. During the update step, the assignment to a cluster is achieved by choosing the nearest cluster centroid given these constraints. Again, we tried different parameterizations of the algorithm. The best results for our requirements were achieved by specifying 10 clusters with sizes between 30 and 150. The resulting clusters were nicely separated in the PC-Space (see Appendix IV), and the silhouette score of our results (0.26) indicates a successful, although weak, clustering.

When mapping the stations color-coded by cluster on a map of D.C., we can see that there are some geographic patterns, for example, dark-purple stations clustering around the very center of the city and the trail of light-green stations towards the southwest of the city. We would not expect perfect geographic clustering using this approach, as, for example, parks and office buildings could be quite close to each other. With the means available to us, and

[8] Utilized implementation: https://pypi.org/project/k-means-constrained/

while acknowledging the limitations of this approach, we were satisfied with this clustering for our purposes.

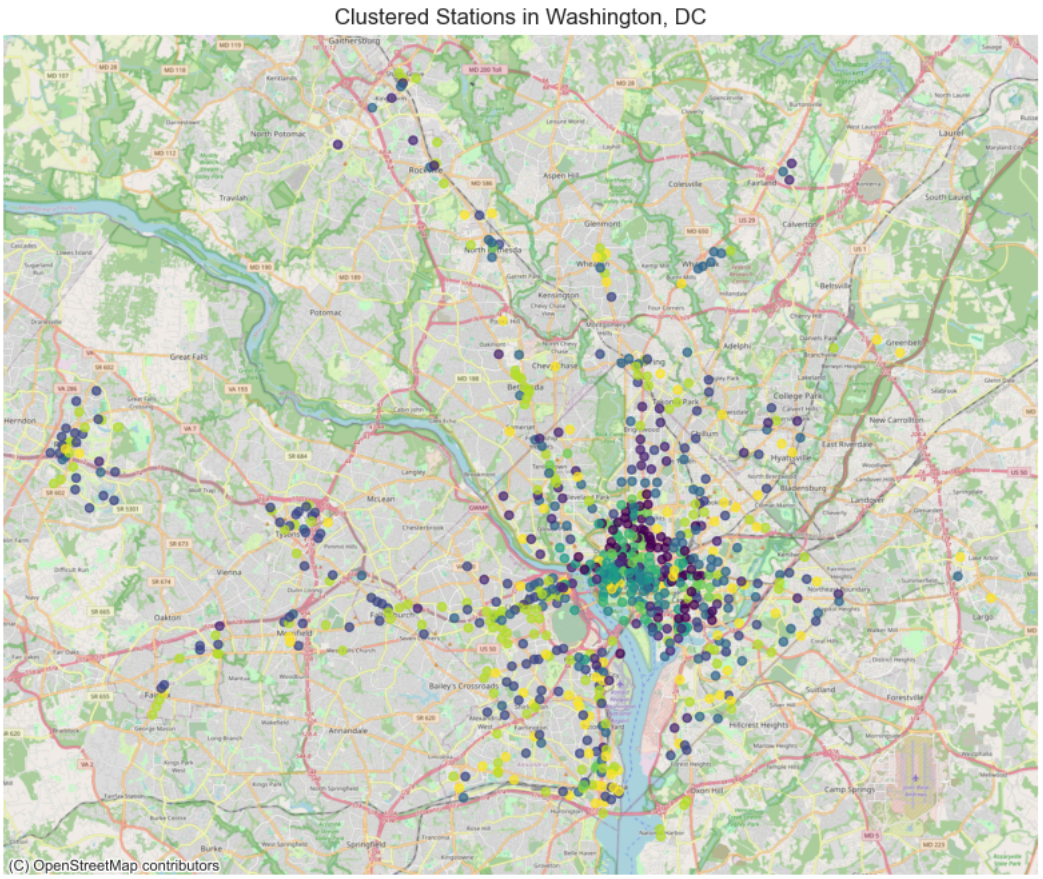


Figure 9: Clusters (marked by color) on a city map of Washington, D.C.

Cluster Characteristics

The clusters exhibit distinct characteristics, as expected from the underlying clustering process. Firstly, size differences are quite significant, as the plot below illustrates. This might have a meaningful impact on modeling, as clusters with fewer trips (rows) have less data available to build a model. That might significantly affect model performance, which we touch upon in later sections.

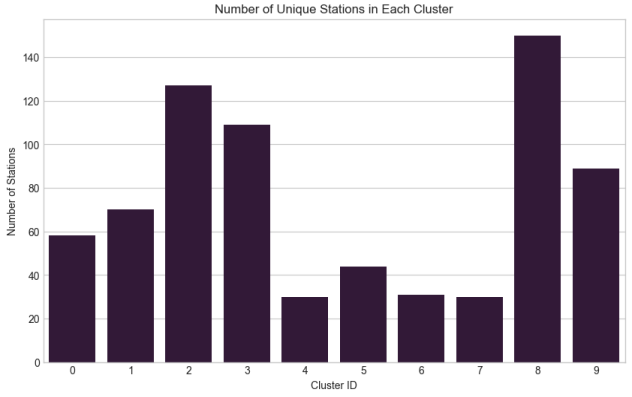


Figure 10: Number of unique stations per cluster.

Secondly, we observe different patterns in departure activity. Clusters vary significantly in the proportion of zero versus non-zero departure values, indicating substantial differences in usage intensity or patterns. This is visualized in the plot below, which highlights the relative share of inactive (zero) versus active time points across clusters. Such variation is crucial for interpreting model performance, as clusters with sparse activity pose different predictive challenges than those with consistent usage. This asymmetry affects both overall error metrics and the comparability of models across clusters, a point that will be revisited in subsequent sections. Note that this asymmetry would persist even when doing predictions on station-level, as the overall activity across stations varies drastically. Proof of this can be seen in Figure 12. Additionally, it is important to keep this in mind when comparing the performance of our models on the different clusters, as our chosen baseline will always have higher RMSEs on the more balanced clusters.

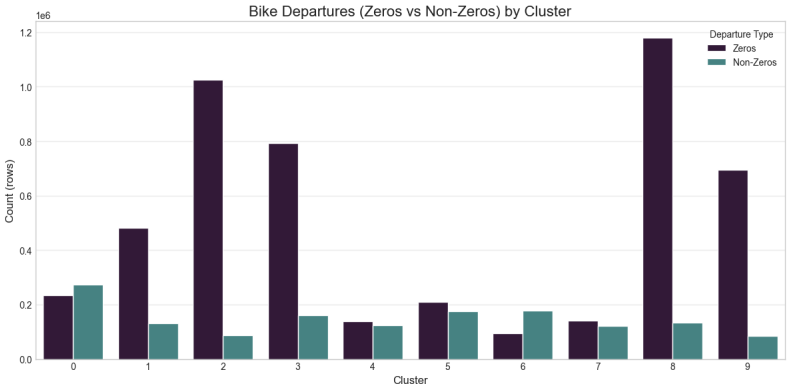


Figure 11: Count of hours with bike departures (zero vs. non-zero) per cluster.

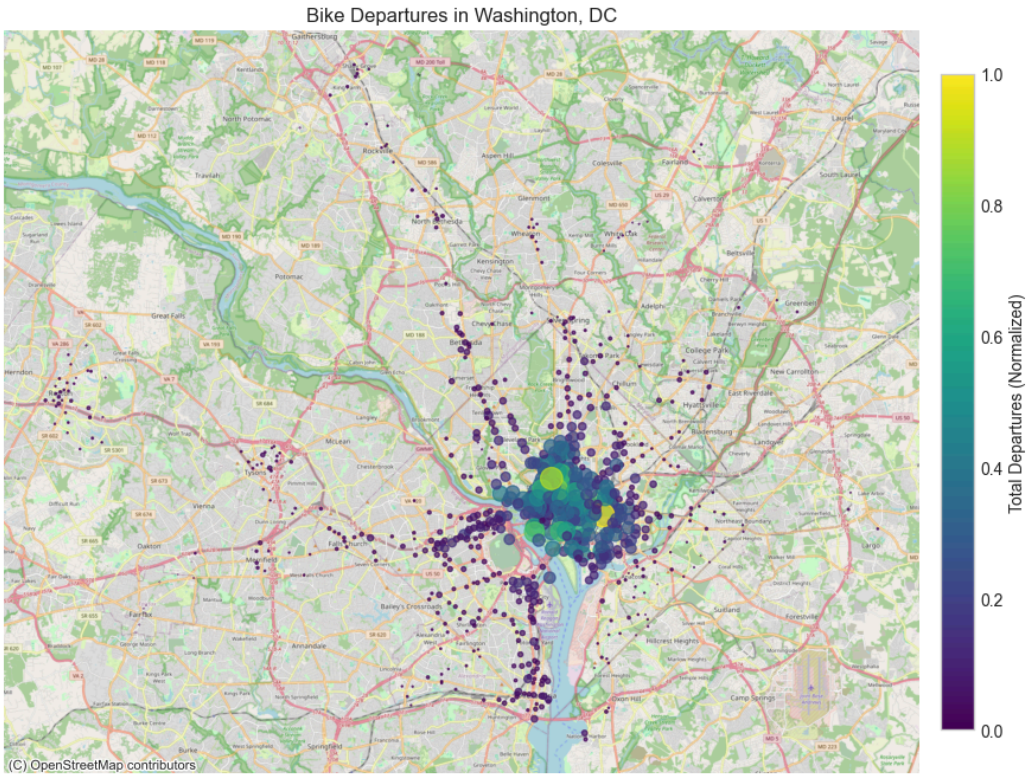


Figure 12: Normalized departure activity on a map (each dot represents one station). Color shows a station's departure activity on a scale from 0 to 1, relative to other stations.

Cluster Models (Unpooled)

We proceeded with training models for each cluster separately, hoping to capture the different usage patterns for each cluster. We find that there are five clusters for which none of our models outperform the baseline, and five clusters for which all of our models (with very few exceptions) do.

The five “bad” clusters are exactly those with a very high share of inactive (zero) time points, i.e., clusters 1, 2, 3, 8, and 9. There is, however, quite some variance in model performance between these clusters: the RMSE values of the best model per cluster range between 3.74 (cluster 2) and 8.58 (cluster 1). Within the group of “bad” clusters, the RMSE is almost perfectly negatively correlated with the share of zero values, meaning that models in clusters with more imbalance do better. One hypothesis explaining this observation is that the models pick up that predicting lower values lowers the evaluation metric, which is solely based on numerical differences and not on picking up trends. To better account for the imbalance present in these clusters, we think that we would need different models that are tailored specifically to predicting sparse targets. Options for this include Poisson Regression and Negative Binomial Regression, which we elaborate on later.

Within the “good” clusters, the share of inactive time points is between 35% and 55%, making them appropriate for classical machine learning methods. This is evidenced by the fact that all models in these clusters outperform the baseline, except for the Linear and Ridge Regression models in cluster 7. The RMSE values range between 1.56 and 2.67 and are, as expected, very weakly correlated (correlation coefficient: -0.18) with the share of zero values. XGBoost is the best-performing model across all clusters, except for cluster 5, where all Linear Regression models outperform all Tree models. Notably, across the “good” clusters, the difference between train and test RMSE is much smaller than in the “bad” clusters, suggesting that overfitting the training set might explain some part of why the imbalance presents such a problem to our models.

Across all clusters, we see that Linear Regression performs the worst, and XGBoost performs best, while the ranking of the other models varies per cluster. Interestingly, the relatively complex XGBoost model does not perform much better than the simpler Linear Regression models, usually only beating them by roughly 5 percentage points with respect to improvement upon the baseline. We expected a larger difference, but have a few hypotheses for why that did not materialize. Firstly, it might be the case that splitting the stations into clusters reduces variation in such a way that the remaining problems per cluster are simple enough for the Linear Regressions, and a more complex model is actually not needed, or might even overfit. Secondly, the problem might not be as complex as we thought due to a lack of features like station characteristics, events in the city, etc. Thirdly, the XGBoost model might just have misspecified hyperparameters and not reach its full predictive capabilities.

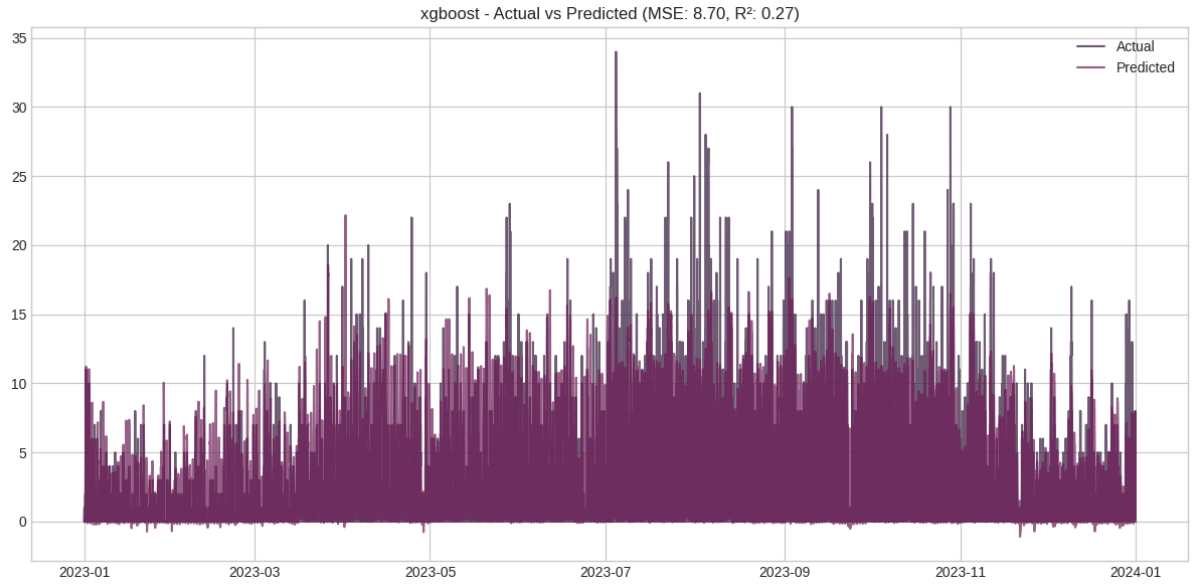


Figure 13: Predicted and actual departures for XGBoost model for Cluster 4, on a 5% test set sampled from the whole year. *Time processing strategy: Trigonometric, learning_rate: 0.1, max_depth: 6, n_estimators: 100*

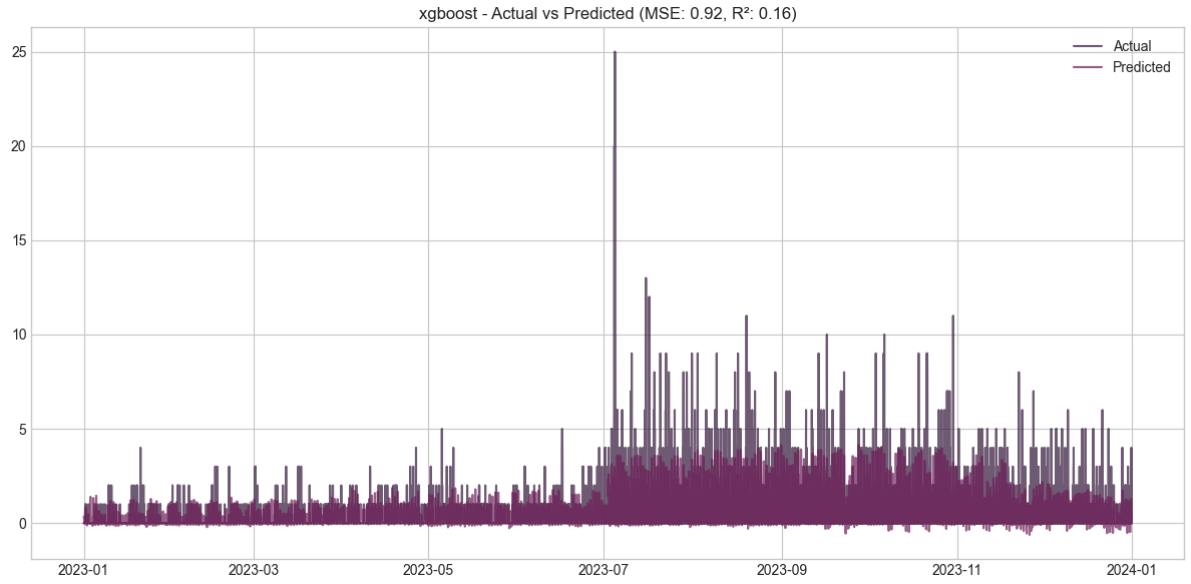


Figure 14: Predicted and actual departures for XGBoost model for Cluster 7, on a 5% test set sampled from the whole year. *Time processing strategy: Trigonometric, learning_rate: 0.1, max_depth: 6, n_estimators: 50*

Comparing the predicted-versus-actual plots for the best models for clusters 4 and 7, we observe that the patterns in actual departures differ significantly between the clusters. A pooled model would have missed this cluster-level information, and the purpose of clustering was exactly to enable the unpooled models to capture the distinct patterns, or even functional forms, common to the groups found by the clustering algorithm.

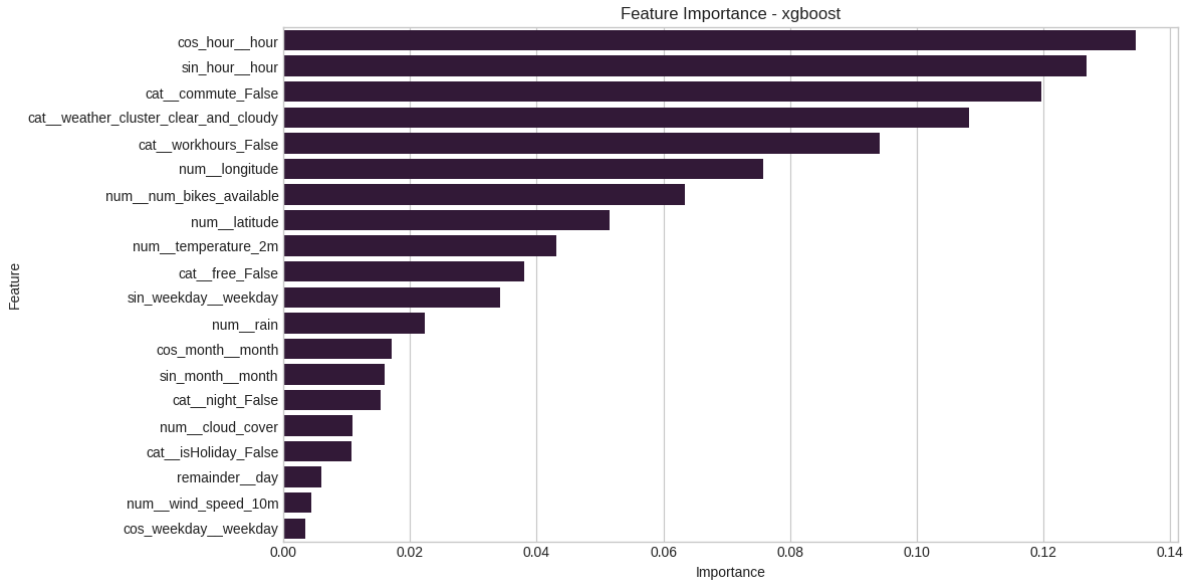


Figure 15: Feature importance plot for XGBoost model for Cluster 4.
Model specifications as in Figure 13.

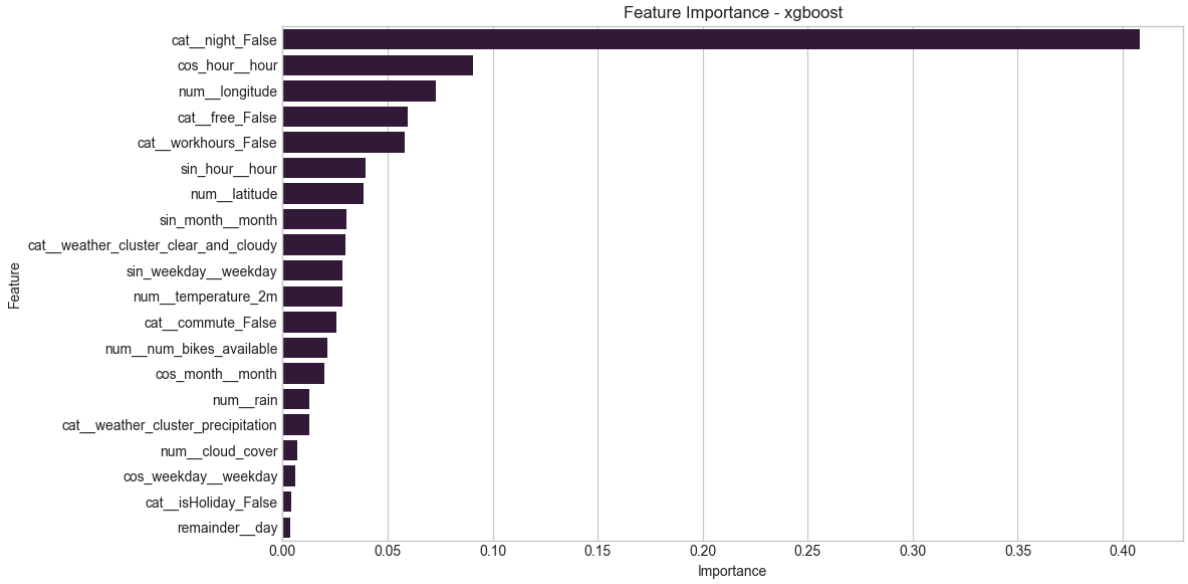


Figure 16: Feature importance plot for XGBoost model for Cluster 7.
Model specifications as in Figure 14.

The models also differ quite significantly on feature importance, again indicating relevant differences across clusters, even beyond purely temporal patterns. In cluster 7, **night_false** is by far the top driver, while the other features (related to time and latitude/longitude) play a much smaller role. In cluster 4, however, the raw hour features **cos_hour** and **sin_hour** matter most, but many other features are similarly relevant, among them **weather_cluster_clear_and_cloudy** (an indicator for good weather) and **commute_false**.

Model	Cluster ID/Subset	RMSE	Mean Train Score	Baseline	RMSE/Baseline
Lasso	citywide-aggregate	272.32	182.575	554.559	0.491
XGBoost	0	1.559	1.566	2.316	0.673
XGBoost	1*	8.580	3.141	1.105	7.766
XGBoost	2*	3.740	1.585	0.464	8.065
XGBoost	3*	6.684	2.358	0.786	8.499
XGBoost	4	2.672	1.853	3.512	0.761
Ridge	5	1.915	1.840	2.597	0.737
XGBoost	6	2.457	2.432	4.126	0.596
XGBoost	7	2.007	1.822	2.578	0.779
XGBoost	8*	4.317	1.807	0.538	8.022
XGBoost	9*	5.112	1.937	0.613	8.346
XGBoost	pooled	1.175	n/a	1.630	0.721

Table 2: Results of the best-performing model for each cluster, as well as Pooled and Citywide Aggregate model. “Bad” (unbalanced) Clusters marked with *.

May Subset

To be able to evaluate the Polynomial Regression and Random Forest, we trained these and all other models on a small subset of our data - specifically, 10 days in May. We chose this date range because we believe it is a time when the weather conditions for cyclists transition to be better but are usually still unstable, which should provide interesting patterns for our model evaluation. Furthermore, we selected this time frame due to the absence of a holiday. With the clusters unchanged, these are still unpooled models as above.

We see that for this small subset, all clusters are “good”, meaning that all models across all clusters beat the baseline (except for Polynomial Regression and Linear Regression in some clusters).

We also see greater diversity in which model performs best than in the full dataset, with Ridge Regression being the best performing model in 2 clusters, Random Forest in 3 clusters, and XGBoost in 5. Polynomial Regression has quite varying performances across clusters, with comparable results to the other models in some clusters, and double or triple RMSE values to the next best model in other clusters. In those latter cases, the train RMSE values are quite similar to the other models’, but then the test RMSE explodes, suggesting that the degree is too large, and that the training set is being overfit.

Evaluating Polynomial Regression and Random Forest, we conclude that given more computing power and memory, it would be reasonable to fit a Random Forest Model for the full dataset, as performance on the subset was quite good. The Polynomial Regression, however, looks less promising to us.

Pooled Models

We also fit a model on the entire city, without grouping the data by clusters. This should help evaluate if clustering added information and aided modelling. As the models fit in this part utilized all 6 million rows of the data frame, we encountered a few computational issues. Apart from crashing many times and eventually taking 10 hours to run for all models, we also observed the problem that some folds of the grid search failed because of an “ArrayMemoryError”. This happened mainly for the Lasso Regression, with about 10% of folds failing, while very few folds failed for the other models. However, as this only means that a small part of the parameter space could not be explored, we hope it does not impact final results too gravely. It also, however, means that we cannot report the mean train RMSE.

All models in the pooled setting beat the baseline, and the best model has an RMSE value of 1.17, which is better than any of the cluster models. This is interesting because across the whole dataset, the share of zero values is 77% (similar to cluster 1), which would place the pooled model closer to the “bad” clusters described above than to the “good” ones. Potentially, being trained on up to ten times more observations helped make up for the imbalance - however, the clusters with the largest share of zeroes are also quite large, so size alone is likely not the determining factor. This curiosity certainly warrants fitting a semi-pooled model, to incorporate both the different patterns captured in the clusters, and whatever makes the pooled model more capable of handling the imbalance.

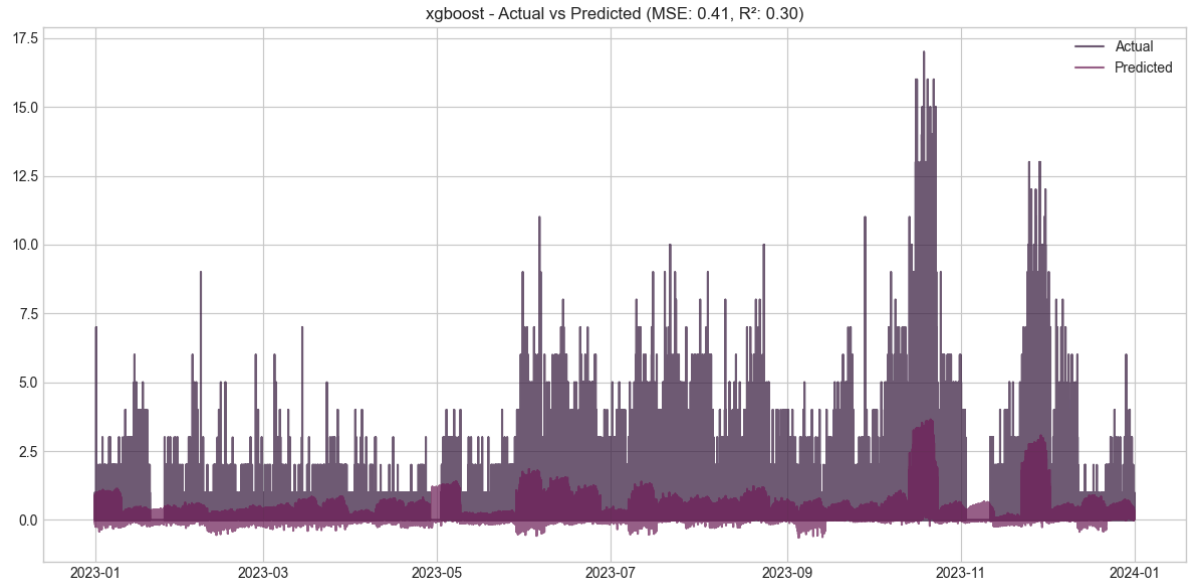


Figure 17: Predicted and actual departures for Pooled XGBoost model, on a 5% test set sampled from the whole year. *Time processing strategy: Trigonometric, learning_rate: 0.1, max_depth: 6, n_estimators: 100.*

In the predictions of the pooled model, we can see that it does not accurately capture the magnitude of departures, but it does identify correctly when peaks and drops in departures occur. We would argue that this is more important in our policy context, since for maintenance and fleet management, decision makers mainly need to know about usage spikes, but not about their exact size at a granularity of 3-5 bikes. However, the model makes negative predictions, which obviously make no sense when the target is departures from bike sharing stations - this is something that could be prevented by using, e.g., Negative Binomial Regression.

This plot, and the fact that this is the model with the lowest RMSE on the full dataset, suggest to us that it would be reasonable to not only consider the RMSE when selecting a model, but also visually inspect predictions, as well as define very clearly the needs of the decision maker in terms of wanting to know only relative changes/outliers or absolute numbers.

Bayesian (Semi-Pooled)

The problem we are modelling is clearly of a hierarchical nature, as each trip departure belongs to a station, which itself is part of a cluster of stations. This makes Bayesian modelling an obvious fit for the problem, due to the intuitive possibility of borrowing strengths over groups when modelling hierarchical data in partial-pooling models.

However, due to the high amounts of compute necessary to run the MCMC samplers modern Bayesian inference utilizes, it was not feasible for us to model nearly a full year of data. Subsetting to a single cluster would be nonsensical, as this leads to losing precisely the hierarchical nature of the data that makes this approach interesting. We therefore subset to a reduced timeframe, choosing a fixed range of 10 days throughout May, that still yielded draws fast enough to run our model in under eight hours. To achieve this, it was also necessary to reduce the number of stations in our data, which we reduced by sampling 100 stations at random.

We chose to implement a varying-intercept-varying-slope model, which can account for the differences in parameters between the different clusters. Building on what we learned about the data throughout our project until this step, we accounted for the dominance of 0 values in the departures column by modelling its outcome through a Negative Binomial distribution. For all other features, we set up standard, weakly informative priors. By setting up a hierarchical model, we pooled all estimates throughout clusters. The full specification for our model, as implemented in *PyMC*, can be found in Appendix V.

Due to the drastic reduction in data size that was necessary to run this model, we decided to only rely on posterior-predictive checks in order to assess how far this idea would take us. We sampled from 4 chains in parallel, with a warm-up period of 1000 draws and sampling from the posterior for 100 draws. We noticed one divergence during sampling on one chain.

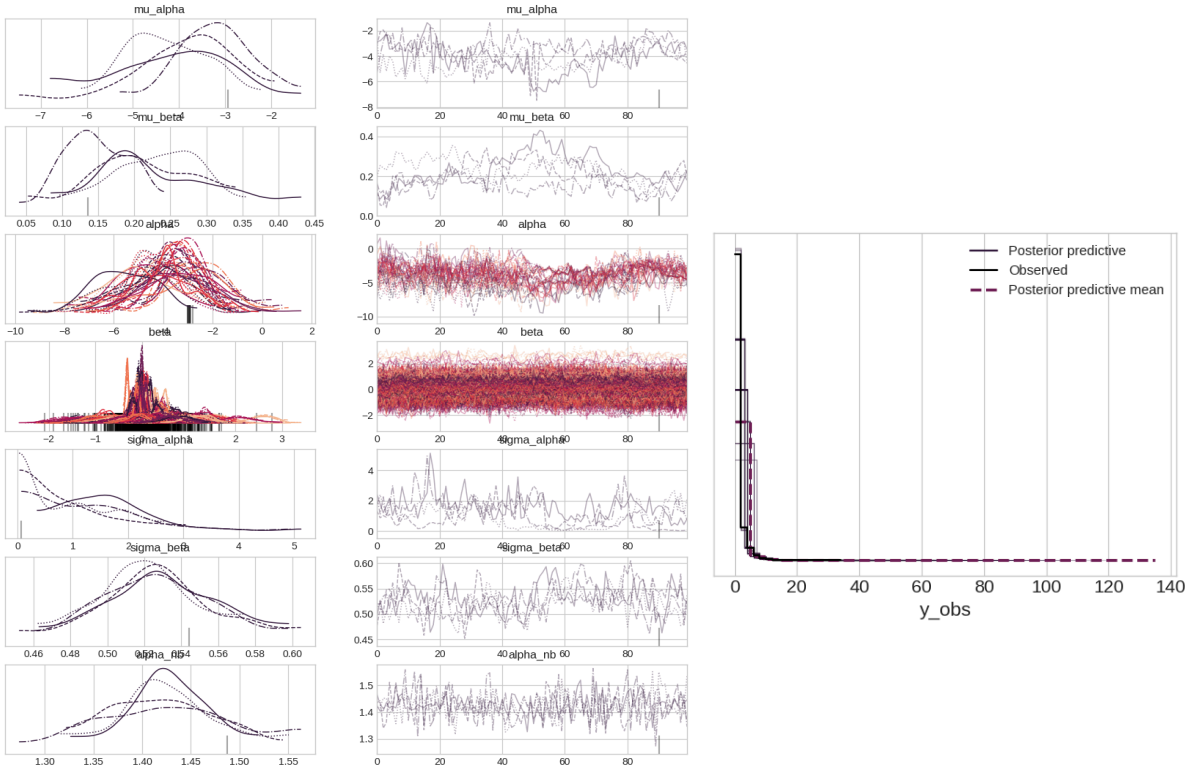


Figure 18: arviz Traceplot of 100 Sampling Draws of our Model after 1000 Warmup Draws on the Left. Posterior-Predictive-Check of the same Model on the right.

From the above trace-plot, we can see that the 4 chains mixed reasonably well for most parameters, and the posteriors for most of the parameters are reasonably well identified. Posteriors for alpha_nb, the parameter of the negative binomial distribution, as well as mu_alpha and mu_beta are not stable across the four chains. However, we still deem this an insightful experiment, as the distributions look like they can reasonably be expected to converge further, when drawing further samples from the posterior. Note that our 100 draws per chain yield neither enough Bulk- nor Tail-ESS to generate exact and reliable insights. Nevertheless, even in this state, our posterior predictive check indicates a well-suited model for our task. To draw definitive conclusions about the suitability of the model and the specified priors, significantly more work would be needed, which was not feasible within the scope of this project.

Future Work

Wider Variety of Models

An interesting avenue that we would have liked to pursue further is the application of Neural Networks. We believe that these could be very well suited for the task, due to their ability to pick up latent and complex functional forms. Interesting experiments could include different architectures, especially on the unclustered - i.e., station-level data - to let the model discover the similarities between different stations by itself. However, due to the limited scope of the project and difficult challenges to solve, not only with regard to setting up the models and the necessary compute times, but also regarding the time series aspect of the data, we chose not to explore this route further at this point.

Quite late in the process of our project, we learned about Poisson Regression and Negative Binomial Regression (see Bayesian approach above). As we are dealing with overdispersed count-data (i.e., count data, $n \geq 0$, with a very drastic left peak in the distribution - in our case the drastic dominance of stations with 0 departures for most hours), both of these could be suitable for our problem. We experimented with Poisson Regression over a wide variety of regularization parameters on a subset of the data, finding it to yield sub-par results. While the error of these models was comparatively low, plotting the observed versus predicted values of a 5% test set, chosen at random, showed that a simple Poisson Regression seldom picked up the actual usage patterns in any meaningful way. The error got "artificially" decreased as an artifact of the model predicting very low numbers of departures. At the same time, the model failed to recognize relative trends and spikes in the data.

However, our experimentation with the Negative Binomial model in a Bayesian setting looked promising, and the mean and variance of our data also indicate that this would be more appropriate than a Poisson Regression[9]. We decided against further exploration of the frequentist Negative Binomial Regression due to the absence of an off-the-shelf implementation in scikit-learn.

Assumptions

Driven by a desire to utilize as much of the data at hand as possible, we focused a lot of our analysis on comparing models among the different clusters we identified. While we think the assumption that there are different types of stations that showcase different usage characteristics still holds, it is important to note that it is not clear whether our clustering approach picked this up sufficiently. While we evaluated the clustering results carefully, the number of stations and high dimensionality of the space that we operate in permit only limited insight. More data sources, beyond the engineered time features, would be helpful to further improve and validate this approach. This could entail data on, for example, tourist attractions or office buildings in the area of a station, public events, or population density.

We acknowledge that an alternative approach could have been to focus on a subset of stations, which would have equally reduced the number of models to be fitted. However, this approach would drastically reduce the data available for training, which could have an impact on model performance. Overall, we still think our approach generated interesting insights that were worth exploring. Additionally, we could observe quite drastic differences in

[9] https://www.bayesrulesbook.com/chapter-12

the performance of different models across different clusters, indicating that they indeed showcased different characteristics.

Conclusions & Learnings

As alluded to above, we chose to utilize as much data as possible from a comparatively (for educational purposes) large data set. This confronted us with some significant challenges, because we could not just “run the model again” in most cases, but getting final results meant investing multiple hours and high amounts of compute.

This made us learn a lot about strategies for writing robust code throughout the entire pipeline, such as automating the storage of model metrics and figures, subsetting data in an automated, but configurable manner and configuring which exact models to run from a larger set of models over grid searches. On the other hand, this meant that we were limited in the number of different models we could try. Initially, we also spent quite a bit of time both understanding the data and thinking about different angles from which actionable insights could be generated, including experimenting with different target variables.

However, our models could generate the following insights:

1. Predicting bike rentals from weather and time data alone already allows quite simple models to pick up a lot of the underlying patterns, e.g., peaks induced by typical commute hours and a general upswing in weather over summer.
2. Different rental stations showcase different types of usage patterns, and for these, different types of models perform best.
3. Both unpooled and pooled models generate insights usable in a decision context. Our unpooled models on clusters serve as a proof of concept for even more granular, station-level models.
4. The unpooled approach can become problematic for clusters (or stations) that are less utilized overall, i.e., that have zero departures in many hours.

Lastly, we want to point out that in a decision-making context, our models could be far more useful than the typical evaluation metrics lead to believe upon a first look: Both for politicians concerned with urban planning, or for rental bike companies themselves, models that pick up general trends such as spikes in usage patterns, are already beneficial: While the concrete number of predicted rentals might be inaccurate, which makes our models unsuitable for allocation purposes, identifying relative spikes already provides insights that can be used, for example, to adapt (temporary) bike-lane planning and traffic light cycles, or for prioritizing which station to repair first under resource constraints.

Appendix

Appendix I: Overview of all Features in Training Set

Each row represents an hour at a given station.

Note: This represents the data before covariate analysis and EDA. Please refer to the Exploratory Data Analysis section to learn which variables were discarded.

Feature name	Sample Value
station_name	Lincoln Rd & Seaton Pl NE/Harry Thomas Rec Center
hour	4
departures	0.000000
arrivals	0.000000
hour_extract	4
temperature_2m	6.600000
weather_code	3.000000
rain	6.000000
precipitation	5.000000
snowfall	1.000000
cloud_cover	100.000000
wind_gusts_10m	20.880001
wind_speed_10m	10.948973
sunrise	1679483288.000000
sunset	1679527275.000000
weather_cluster	clear_and_cloudy
num_docks_available	16
num_bikes_available	2
num_ebikes_available	1
has_kiosk	YES
capacity	19
latitude	38.915000
longitude	-77.007800
isHoliday	False
weekday	2
night	True
workhours	False
commute	False

free	False
day	22
month	3
year	2023
dayofweek	2
dayofyear	81
timestamp	2023-03-22 04:00:00+00:00
avg_delta_station_total*	-0.057771
avg_arrivals_station_total*	0.450775
avg_departures_station_total*	0.508546
avg_delta_station_night*	0.164692
avg_arrivals_station_night*	0.280462
avg_departures_station_night*	0.115770
avg_delta_station_nonnight*	-0.191249
avg_arrivals_station_nonnight*	0.552963
avg_departures_station_nonnight*	0.744211
avg_delta_station_holiday*	-0.025641
avg_arrivals_station_holiday*	0.346154
avg_departures_station_holiday*	0.371795
avg_delta_station_commute*	-0.336151
avg_arrivals_station_commute	0.536736
avg_departures_station_commute*	0.872887
avg_delta_station_free*	-0.040948
avg_arrivals_station_free*	0.411997
avg_departures_station_free*	0.452945
avg_delta_station_unfriendly_weather*	-0.041267
avg_arrivals_station_unfriendly_weather*	0.406419
avg_departures_station_unfriendly_weather*	0.447687
avg_sum_station*	0.959321
avg_sum_station_night*	0.396232
avg_sum_station_nonnight*	1.297
avg_sum_station_holiday*	0.718
avg_sum_station_commute*	1.410
avg_sum_station_free*	0.865
avg_sum_station_unfriendly_weather*	0.854
cluster	3

* the marked features were used for clustering only

First 5 rows of our training data X:

isHoliday	has_kiosk	weather_cluster	workhours	commute	free	night	temperature_2m	rain	snowfall	cloud_cover	wind_speed_10m	num_bikes_available	latitude	longitude	weekday	day	month	hour	cluster
True	YES	precipitation	False	False	True	True	11.936	0.2	0	20	8.47339	12	38.8959	-77.0261	6	1	1	0	0
True	YES	clear_and_cloudy	False	False	True	True	11.486	0	0	0	10.1377	12	38.8959	-77.0261	6	1	1	1	0
True	YES	clear_and_cloudy	False	False	True	True	11.036	0	0	4	11.441	12	38.8959	-77.0261	6	1	1	2	0
True	YES	clear_and_cloudy	False	False	True	True	10.786	0	0	34	12.2241	12	38.8959	-77.0261	6	1	1	3	0
True	YES	clear_and_cloudy	False	False	True	True	10.536	0	0	18	15.1756	12	38.8959	-77.0261	6	1	1	4	0

Appendix II: Weather Code Mapping

The following table shows the mapping of “our” weather codes (first column) to the WMO weather codes (second column) as described in the open-meteo documentation[10].

Weather cluster	**WMO weather codes**
clear_and_cloudy	0,1,2,3
precipitation	51, 53, 55, 61, 63, 65, 80, 81, 82
frozen_precipitation	56, 57, 66, 67, 71, 73, 75, 77, 85, 86
low_visibility	45, 48
severe_weather	95, 96, 99

Appendix III: Key Metrics About the Target Variable

Overall:

Metric	**Value**
Count	6464880.00
Mean	0.562
Median	0.000
SD	1.531
Min	0.000
Max	57.000
Share Zeros	77.250

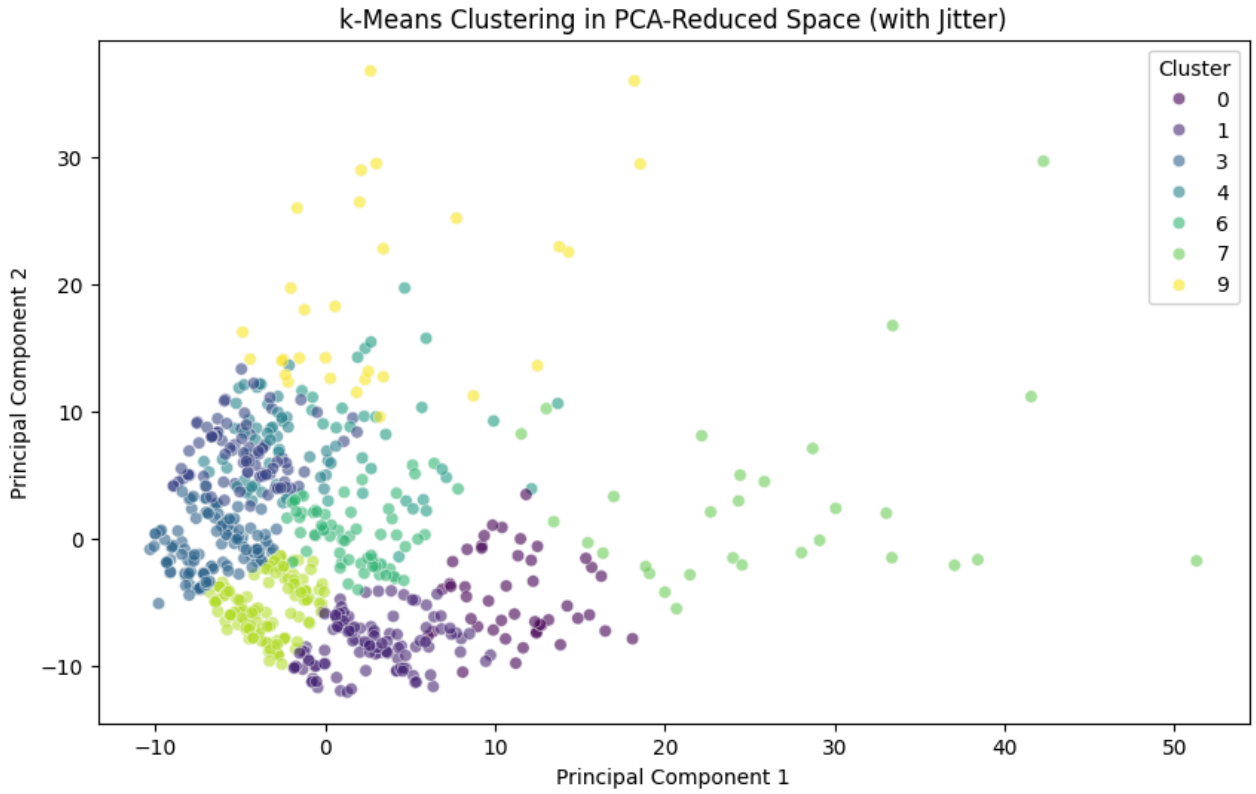
Per Cluster:

Cluster	**Count**	**Mean**	**Median**	**SD**	**Min**	**Max**	**Zeros**	**Share Zeros**
0	508080	1.36	1.00	1.88	0.00	34.00	233681	0.46
1	613200	0.40	0.00	1.03	0.00	23.00	481569	0.79
2	1112520	0.11	0.00	0.45	0.00	16.00	1025205	0.92
3	954840	0.27	0.00	0.74	0.00	43.00	793188	0.83
4	262800	1.70	0.00	3.07	0.00	52.00	138843	0.53

[10] https://open-meteo.com/en/docs

5	385440	1.32	0.00	2.24	0.00	33.00	210853	0.55
6	271560	2.56	1.00	3.24	0.00	57.00	94810	0.35
7	262800	1.31	0.00	2.219451	0.00	48.00	141436	0.54
8	1314000	0.15	0.00	0.518003	0.00	15.00	1180860	0.90
9	779640	0.17	0.00	0.59	0.00	20.00	693701	0.89

Appendix IV: Clusters on two Dimensions of a 6D-PC Space



Appendix V: Full Specification of our Bayesian Model in PyMC

```
partial_pooling = pm.Model()
with partial_pooling:
    # hyperpriors for group-level intercepts
    mu_alpha = pm.Normal('mu_alpha', mu=0, sigma=10)
    sigma_alpha = pm.HalfCauchy('sigma_alpha', beta=5)

    # hyperpriors for group-level slopes
    mu_beta = pm.Normal('mu_beta', mu=0, sigma=10)
    sigma_beta = pm.HalfCauchy('sigma_beta', beta=5)

    # group-level intercepts
    alpha = pm.Normal('alpha', mu=mu_alpha, sigma=sigma_alpha, shape=n_groups)

    # group-level slopes
    # beta parameter for each feature for each group
    beta = pm.Normal('beta', mu=mu_beta, sigma=sigma_beta, shape=(n_groups, b_n_feats))

    # overdispersion parameter for NegBin (alpha > 0)
    alpha_nb = pm.HalfCauchy('alpha_nb', beta=5)

    # linear predictor
    eta = alpha[group_idx] + pm.math.sum(beta[group_idx] * b_X, axis=1)

    # inverse link function: mean mu > 0 for the negative binomial
    mu = pm.math.exp(eta)

    # likelihood
    y_obs = pm.NegativeBinomial('y_obs', mu=mu, alpha=alpha_nb, observed=b_y)
```