Predicting Affluence in Montreal Bike Sharing Graph

Louis Chaubert[1] and Léandre Dubey[2]

[1] University of Bern
louis.chauber@students.unibe.ch
[2] University of Fribourg
leandre.dubey@unifr.ch

Abstract. This project investigates the prediction of bike-sharing demand in Montreal using data from Bixi, a bike-sharing company. Data exploration revealed significant patterns, including a substantial increase in bike usage over the past decade, seasonal trends with higher demand in warmer months, and distinct weekday peaks due to commuter activity. The study compares two predictive models: a Geographically Weighted Regression (GWR) and a multi-linear regression (MLR). While GWR incorporates spatial variability, the simpler MLR model demonstrates superior runtime efficiency and accuracy. The findings indicate that the MLR model is more effective for predicting bike-sharing demand, aiding in better service coverage and balance between bike availability and demand.

Keywords: Temporal Graph · Analysis · Prediction · Bike-sharing · Demand Prediction · Multi-Linear Regression · Geographically Weighted Regression.

1 Introduction

Bike sharing has become one of the criterias that determine the sustainability of cities around the world [3]. Bike sharing programs (BSPs) are an interesting solution to reduce air and noise pollution while promoting physical activity. To support this beneficial travel medium, it is important to provide a satisfying and appealing service for every cities.

Several factors may influence bike sharing popularity. But apart from uncontrollable influences like the weather condition or the landscape formation, bike sharing companies should focus on guaranteeing sufficient service coverage. They can achieve this by trying to predict the demand for bikes by analysing data from previous years.

To this purpose we analyse freely available data from the bike sharing company Bixi in Montréal, Canada. First we have a look at how to efficiently select and load the data. Then, we assess the quality of the data provided and preprocess it. Next we explore the data in the form a network graph, ensure its persistence, and enrich it with some additional information. Finally we carry

out an analysis with the goal to predict out-going and in-going flux, to allow the capacity to ensure balance between offer and demand. The analysis focus on comparing different algorithms in terms of accuracy and runtime efficiency.

The code used for the project is available at `https://github.com/citronpower/Bixi`

2 Pre-analysis Part

2.1 Data Loading

The dataset selection is a crucial step in any data analysis project. For the current project, we chose the freely available dataset from the bike sharing company Bixi in Montréal [2]. For each year since 2014, they provide the information of every travel made with their rented bikes in csv format. We opted for this dataset since it allows to choose the amount of data we want to process and analyse. We loaded the data and performed a primary visual exploration of the data using python libraries like matplotlib and networkx. Although we used the data from the whole decade for some analysis, we focus mainly on the entries for 2023 since it already amounted to the colossal number of more than 11 millions travels between 939 stations. This large amount of data would already prove to be a considerable challenge. Each entry was composed of the start and end stations with their names, arrondissements, latitudes, and longitudes coordinates and each entry also had the start and end timestamp in milliseconds of the travel.

2.2 Data Quality and Pre-processing

Once the data was loaded, we cleaned it and pre-processed it before applying any further analysis. The first issue we encountered concerned the variability in the dataset formats throughout the 10 years. So we created a script that would standardize the format for all years. Roughly it consisted of dropping superfluous columns, renaming others and transforming the timestamps to the ms format. Furthermore, some year had the stations and travels separate whereas some other had them combined. We also dealed with wrong coordinates (e.g. -1, -1) that we reconstructed manually using the name of the station. The entries without departure of arrivals were dropped. Focusing on the 2023 dataset, we dropped all duplicates and entries with missing values which amounted to 67457 rows (0.5%) removed. We added a start date column where we transformed the start timestamp in ms to the date/time format.

Furthermore, to prepare the data for network representation and analysis, we split the data to create a dataset with unique nodes and a dataset with the edges. On one hand, each individual station was transformed into a node with unique ID and attributes name, longitude and latitude coordinates. On the other hand, each travel entry was made into an edge with the IDs of the start and end stations and with attributes start date in date/time format, and start and end time in ms.

2.3 Network Exploration and Visualisation

Using the library networkx, we created multiple instances of graphs. We have a temporal graph on our hands, this means that the edges in our case evolve depending on the time window that we select. First we created the graph with every nodes and edges. This would serve to run analysis on the whole graph, but would not help much for visualisation since the large number of edges made it hard to discriminate them (Figure 1).

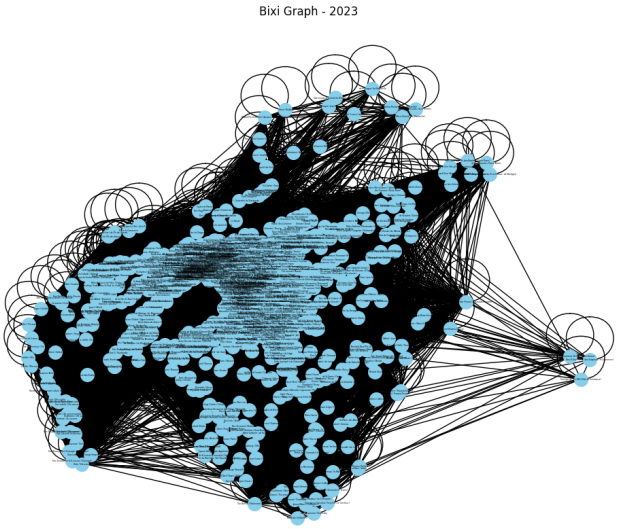


Fig. 1. The Whole Network Graph for 2023

So, we decided to split the whole graph into sub-graphs with every months as time windows. In term this allowed to visualize the graph for months with smaller amount of rentals like in January when the weather is less appealing for bike rental. By computing the betweenness-centrality for each station at each months, we could observe that some stations were recurrently in the the top 5 highest measures (see Table 1 in Appendix). This gave us the initial hypothesis that some stations were acting as bottlenecks and that the network could indeed benefit from a more in-depth analysis.

To understand this temporal network's dynamic, we looked at the rental patterns by plotting the total rentals per week from 2014 to 2024 (see Figure 2). This highlighted two things: (1) first, the demand has almost quadrupled since the last decade, meaning that the company has to be proactive to guarantee the quality of the service, and (2) second, there is a seasonal pattern with dead periods where the weather is probably too unpleasant to bike and rentals drastically

drop. In winter, Bixi even removes the bikes and recently offer another type of service with snow-bikes.

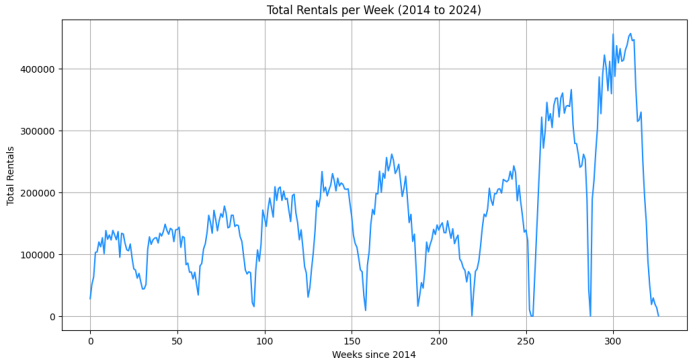


Fig. 2. Total rentals per week during the last decade (2014 to 2024).

Next, we narrowed the time window by plotting the hourly rentals throughout the weeks of 2023 (see Figure 3). In this plot, each peak corresponds to a day of the week. We observe an interesting pattern during the week with two peaks, one early in the morning and one later in the evening. These rush hours are probably created by workers commuting to their job and back, and by people going out in the evening. During the weekend, the pattern changes and seems more normally distributed. Overall the very important lesson we got from these two plots is that there exists indeed underlying patterns with some noise in the data. This makes us confident that it will be possible to predict for each station the amount of arrival and departures to some degree of precision.

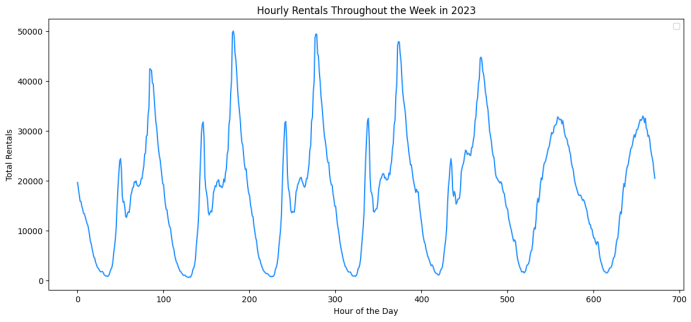


Fig. 3. Hourly rentals throughout the weeks in 2023.

Furthermore, as exploratory analysis, we clustered the stations based on the number of edges they shared. By using the KMeans algorithm, we were expecting to see clusters corresponding to geographically near stations. We could observe some very light clusters in the heatmaps of the travel percentages. (see Figure 4). The analysis was conducted for multiple values of k (see Appendix 11)

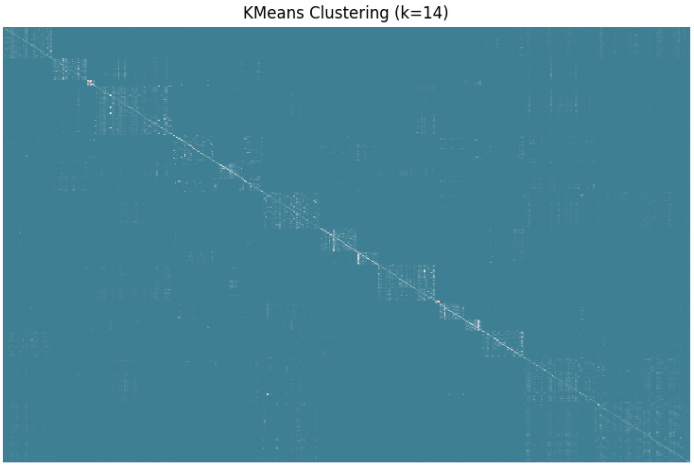


Fig. 4. Kmeans clustered heatmap by the percentage of travels.

2.4 Graph Persistence

Before continuing our analysis, it is interesting to ensure the graph persistence. To that purpose, we used Neo4j to store our network. We tweaked a bit the data to fit the Neo4j format for nodes and edges instances. Then due to the high amount of data we had, we searched for the most efficient way to store it. First, we used the Neo4j library in python to insert directly in the database through queries. It added nodes and edges entry by entry. This first solution was definitely too time consuming and proved to be unusable. Second, we used advanced Cypher queries to upload the whole csv data-file (i.e. using LOAD CSV) at once. This solution showed already better performance but would have required days of runtime. Finally, we successfully managed to store everything swiftly by using the command line admin import tool. We had to prepare the data in order for Neo4j to be able to store them directly in its indexes. The import was then successful in only a few seconds for millions of entries.

Initially, we did some of the explorations and visualisations explained above by using the Neo4j Cypher language. Unfortunately, due to the large amount of data, when we actually had the queries sorted out, they would take hours to execute against minutes for the same analysis with python when the dataset is

directly accessible in memory. Since the goal of this project is a punctual analysis and not to offer real-time analysis of the network, we realized that using Neo4j did not bring much and left it aside.

2.5 Data Enrichment

To improve the dataset with new interesting contextual information and have additional attributes on our edges, we decided to add two new features. The first contextual information consisted of merging a weather dataset of Montréal with our bike sharing dataset [1]. We selected the weather description for every available hour of 2023. Ultimately we added for each travel, i.e. each edge, the description (e.g. sunny or rainy) of the weather, the wind speed, as well as the temperature for the given hour of the travel. This allowed us to plot the number of travels and the weather conditions on the same graph and to visualise the impact of weather on the rentals (Figure 5).

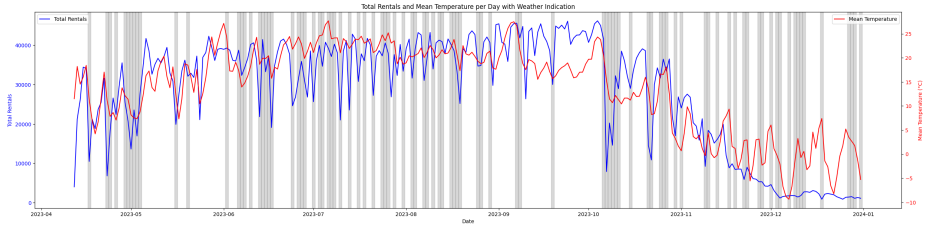


Fig. 5. Impact of the weather condition on the number of rentals. Bad weather in gray, good weather in white, temperature in red and rentals in blue.

The second contextual information consisted of adding an estimation of the distance traveled. First implementation was to use the Haversine formula with the coordinates to have the distance as the crow flies between every stations. However we wanted to have a more realistic estimation of the distance for further analysis. Thus we looked for a way to find the approximate distance that would be traveled if the user was taking the direct biking path to the end station. To that end, we had the choice between many services like Google Maps or Open Route Service for their Time-Distance Matrix API service which calculates the biking distance pairwise of coordinate locations. We opted for Open Route Service since it was free within a given limit. However any API have some constrains. On the case of Open Route Service, we were allowed to compute the pair-wise distance of maximum 59 locations per query, 40 queries per minutes and 2'500 queries per day. One query may contain multiple location and the distance between each possible pair of location is computed. Since we had 939 locations for each stations, and the queries were limited we had to come up with a partitioning algorithm that would efficiently create batches of stations to compute every distances pairwise. Note that every location should be compared at

least one to every other location but the number of comparison should be minimised, optimally the distance between two location should be computed only once. Thanks to the algorithm, we could first reduce the number of comparison from 881'721 to 90'758 and with the addition of a second optimisation, reduce it ultimately to only 4'988, which may be computed in two days given the daily limitation. See Figure 14 and Figure 15 in the Appendix for partitioning size distribution after each step. The algorithm is described bellow.

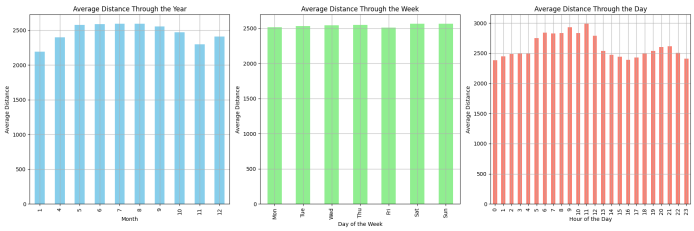


Fig. 6. Distance distribution through time

Partitioning algorithm Previously, we described the problem from the point of view of the Time-Distance Matrix API. However, we may reformulate the problem of partitioning in a formal way. Furthermore, this section present a novel way how to solve this problem using an iterative graph partitioning algorithm.

Formally, given a set of elements S which contains every elements to compare. We derive a number of subsets s_i from S where each element of S appear only once in a subset s_i with any other element of S.

The partitioning algorithm represents all the elements of S as nodes within a fully connected graph (see Figure 7. It then iterates over all nodes within the graph. For each node (called anchor), we first create an array that only contain the anchor, and then iterate over all of its neighbors. We add the neighbors to the array as long as the array does not reach the partition size constrain. In this example, we use the partition size constrain of 3. In the case of the Open Route Service API, it is 59. Each time we add a node to the the array, we remove the edge between **every** nodes within the array. As long as the anchor is not isolated, i.e. has a degree higher than one, we keep iterating over its neighbors. Once the anchor has no more neighbors, we change the anchor with the next node. Note that a partitioning might have less nodes than the partition size constrain, which lead to the second part of the optimization problem explained next. We keep iterating over each anchor until all nodes of the graph are isolated. A complete example of algorithm is available Figure 4 as well as an actual implementation in Appendix Code 4 in the Appendix.

Once the partitions available, we might notice that many partition have a length lower than the partition size constrain. Therefore, in order to have the minimum number of partition, and therefore send the minimum number of

queries to the API, we combine partitions together until it reaches the constrain. Note that we might now compute multiple time the distance between pairs of nodes. However, as we already optimized the partitioning, we don't add additional partitioning (i.e. queries). This algorithm iterates over all the partitions previously computed, create new partitioning and combine them as long as they don't exceed the partitioning size constrain. An example of implementation is available in Appendix Code 4.

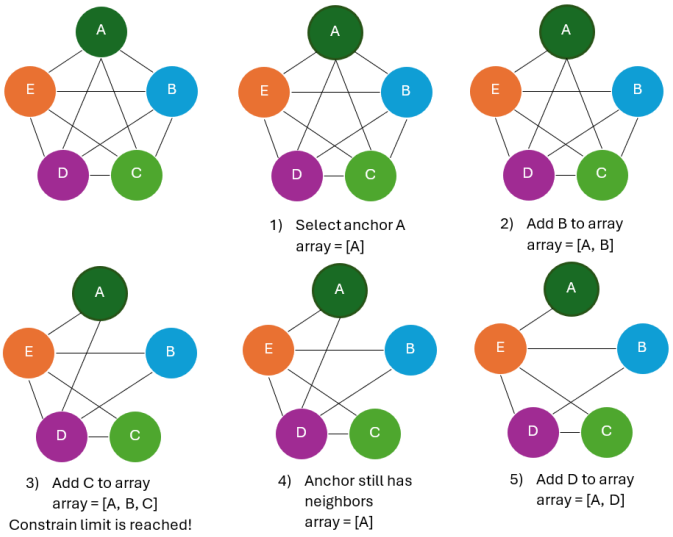


Fig. 7. Example of the first steps of the partitioning algorithm

3 Analysis

The goal of this analysis is to identify bottleneck stations. Typically a bottleneck in the context of bike rental means that the outward flux of bike rentals exceeds the inward flux of bike deposits. This discrepancy limits the network flow and can lead to customers not being able to use it. The benefit of predicting such situations allows to prepare the stations to withstand those high demands moments. As shown in Figure 3, the network's behavior evolves throughout the week, displaying distinct patterns on workdays compared to weekends. On workdays, we observe rush hours due to commuter activity, which would be interesting to predict in terms of their occurrence and location.

Analyzing the workdays (Monday to Friday) throughout the entire year of 2023, we can aggregate the rentals by hour and plot the station-wise difference between inward and outward flux. This difference is visually represented by the size of the marker, indicating the magnitude, and color-coded: red for negative

differences (higher outward flux) and green for positive differences (see Figure 12 in Appendix for every hours). This allows us to identify distinct periods of negative flux in the graph, which vary depending on the hour of the day (Figure 8). To try to predict these flux differences, we can use different type of algorithms.

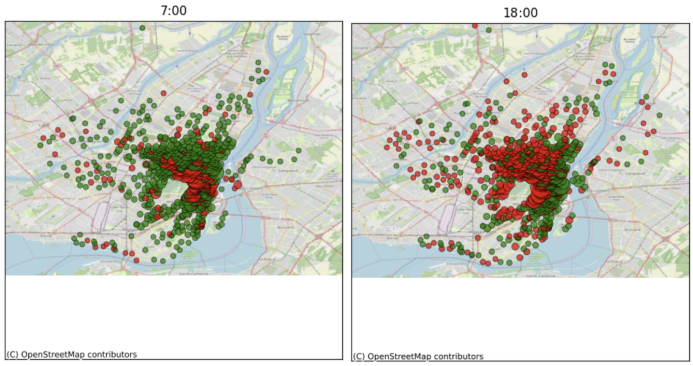


Fig. 8. Visualisation of the difference in inward and outward flux at 7h and 18h.

3.1 Flux prediction

First, we implemented a Geographically Weighted Regression (GWR). It is a type of regression that can be applied to networks with spatial features. It allows for the modeling of spatially varying relationships between a dependent variable and one or more independent variables. Unlike traditional regression models, which assume that relationships are constant across the study area, GWR acknowledges that these relationships can change over space by weighting the data points with their distance from the location of interest. We invested a lot of time into trying to make a GWR algorithm from scratch. With the limited amount of time and the complexity of the task, we failed to make the algorithm work and instead opted for the implementation of a simpler multi-linear regression. We wanted to create two models that would be able to predict (i) the inward flux and (ii) the outward flux for a given station at a given hour of the day. We hypothesized that given the relative simplicity of the algorithm, we would gain in runtime complexity for the training of the models, at the expanse of prediction accuracy.

We compare our implementation of multi-linear regression to a Geographically Weighted Regression (GWR) from the library "mgwr" [5]. Making this algorithm work using the short documentation provided by the authors already proved to be a long and tedious task. This algorithm is more complex, because it incorporates spatial context by using spatial coordinates to model how relationship between variables change over space. With this higher computational

challenge, we hypothesize a higher runtime complexity for the training of the models, with the benefit of higher prediction accuracy.

For our MLR implementation, we used the weather prediction (good or bad), the temperature, the windspeed, the month, the day, the hour, and the coordinates as features. The target value was the count of departures (could be arrivals) in the corresponding hour. For the GWR, we used the weather prediction, the temperature, the windspeed, the month, the day, the hour as features. We also gave the spatial information separately as coordinates and the target values was again the count of departures in the corresponding hour. Ultimately this means that we can use these models to predict the bike flux station-wise by feeding it the time of interest, the weather predictions for this time, and the coordinates of the stations of interest.

To test the accuracy of the models, we split our data into a training set and a test set (respectively 80% and 20%). We use the algorithms to try to predict the outward flux of an hour in the test set and use its data as ground-truth. The exact same algorithm can be used to predict the inward flux.

Algorithm comparison In terms of runtime complexity, training our linear regression model was nearly instantaneous, regardless of the amount of data. In contrast, training the GWR model took significantly longer, exhibiting polynomial complexity (Figure 9). We bench-marked the runtime of the models, using only 2e4 samples (0.2%), 50'000 (0.5%), 100'000 (1%), and 200'000(2%) of our data for the two models to plot the comparison. We were forced to use only small samples since the runtime of the GWR was already of 50 minutes for 2% of the data. Indeed, the time complexity of GWR is depicted as follows[4]:

- Bandwidth selection and training: $O((p+1)^2 n^2 log(n))$
- Predicting: no information found.

Where (p is the number of features and n is the number of observations). Faster GWR have been proposed recently and reduce the time complexity to $O(n^2 log(n))$[6].

Given our implementation of the multi-linear regression, the time complexity are as follows:

- Training: $O(np^2 + p^3)$ where np^2 comes from the multiplication of a $p \cdot p$ matrix with a $p \cdot n$ matrix.
- Predicting: $O(np)$ where come from multiplying X (matrix of shape $n \cdot (p+1)$) with the coefficients vector (vector of shape $p+1$).

Unlike MLR, which fits a single global model, GWR fits many local models, leading to significantly higher computational cost. In our case, the bottleneck is the number observations that is very large. Furthermore, the process of selecting an optimal bandwidth in GWR often involves cross-validation or other iterative optimization techniques that require fitting the model multiple times over a grid of possible bandwidths, further increasing the computation time. In

spatial regression models, the bandwidth determines the range of spatial influence around each data point. In the GWR model we implemented, the optimal bandwidth selection process wasn't possible, so we set a fix bandwidth that is likely suboptimal.

As we hypothesized, the runtime for training and predicting was very short (seconds) for our implementation of MLR, while the GWR needed minutes to hours to do the same (see Figure 9). But what got surprising was that our algorithm actually performed better in terms of accuracy compared to the GWR model (see Figure 10). This could have been due to the suboptimal bandwidth selection that we were forced to do in our project. Also due to time limitations, we couldn't push the comparison to completion by training/testing both models with the complete dataset. The accuracy could have looked differently for both models with more data. If we had to base our choice on the analysis that we made, we would chose our MLR model implemention since it is not only way faster to run, but also more accurate.

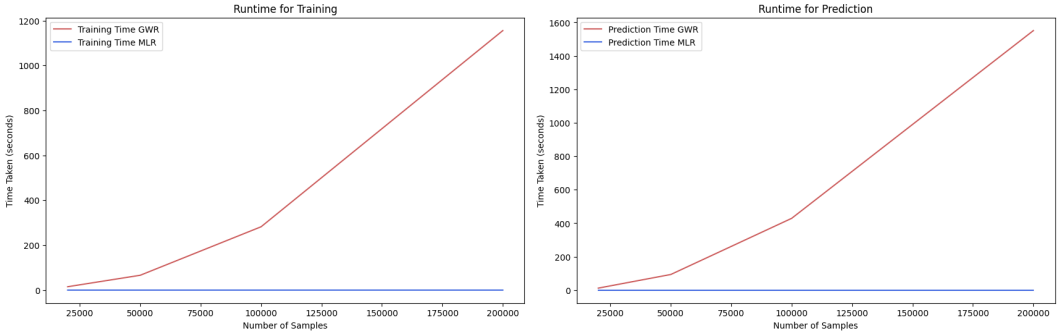


Fig. 9. GWR vs. MLR computational time

4 Discussion

The biggest challenges of this project were the size of the dataset and complexity of the anticipated analysis. First the large size of the dataset meant that every part of the project had to be optimised in order to run in reasonable time. This resulted in a significant loss of time during the data cleaning, pre-processing, and exploration stages, which are already inherently time-consuming tasks. Second, while the prospect of predicting flux in a bike-sharing network was very exciting, it proved to be overly optimistic. Midway through the project, we realized that none of the algorithms presented during the course were suitable for our needs and we had to do literature research. This research revealed that temporal graph prediction is an ongoing field of study with dedicated PhD-level research, far beyond the scope of this course. Consequently, we lost a significant amount of

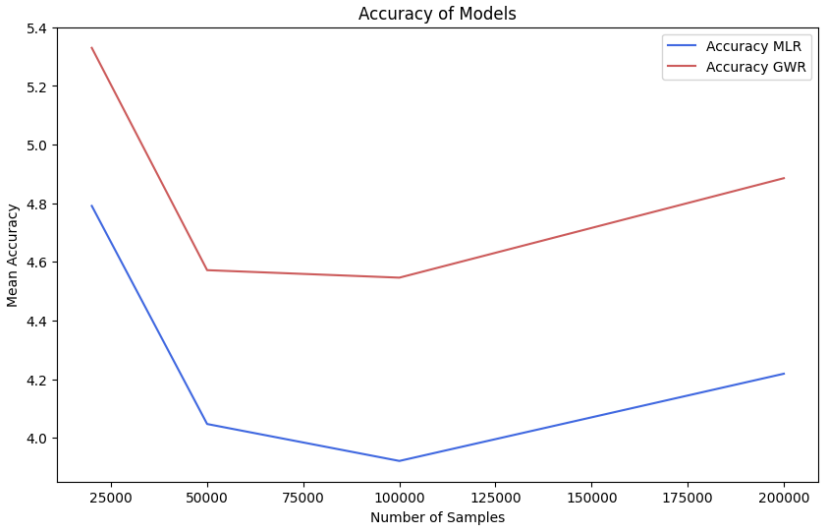


Fig. 10. GWR vs MLR mean accuracy

time attempting to develop a GWR algorithm before ultimately settling for a simpler linear regression approach.

Nonetheless, the project yielded valuable insights for the bike-sharing community. We identified interesting patterns, persevered to find a solution for very large persistent storage, developed a novel algorithm for constrained partitioning, and extracted insights to improve the efficiency of bike-sharing systems through modeling techniques. Future work is always welcome, and with additional time, we would have loved to delve deeper into the modeling algorithms and thoroughly evaluate a range of models suitable for our dataset and objectives.

References

1. Weatherstats.ca based on environment and climate change canada data (2024-05-02 14:32:20), `https://montreal.weatherstats.ca/download.html`
2. Données ouvertes, bixi montréal (2024-05-13 12:18:16), `https://bixi.com/fr/donnees-ouvertes/`
3. Ezgi, E., Emre, U.V.: A review on bike-sharing: The factors affecting bike-sharing demand. Sustainable Cities and Society **54**, 101882 (2020). https://doi.org/10.1016/j.scs.2019.101882
4. Li, Z., Fotheringham, A.S., Li, W., Oshan, T.: Fast geographically weighted regression (fastgwr): a scalable algorithm to investigate spatial process heterogeneity in millions of observations. International Journal of Geographical Information Science **33**(1), 155–175 (2019)
5. Oshan, T.M., Li, Z., Kang, W., Wolf, L.J., Fotheringham, A.S.: mgwr: A python implementation of multiscale geographically weighted regression for investigat-

ing process spatial heterogeneity and scale. ISPRS International Journal of Geo-Information **8**(6), 269 (2019). https://doi.org/10.3390/ijgi8060269
6. Oshan, T., Wolf, L., Fotheringham, A., Kang, W., Li, Z., Yu, H.: A comment on geographically weighted regression with parameter-specific distance metrics. International Journal of Geographical Information Science **33**, 1–12 (02 2019). https://doi.org/10.1080/13658816.2019.1572895

Appendix

Month	Node	Betweenness Centrality (e-3)
January	1185	2.91
	384	2.41
	28193	1.69
	179	1.40
	1017	1.30
April	**299**	3.93
	170	3.41
	662	2.93
	199	2.89
	208	2.64
May	**170**	3.41
	3401	3.31
	187	3.19
	551	2.77
	299	2.36
June	**170**	5.09
	299	2.88
	1022	2.61
	314	2.57
	258	2.54
July	**170**	4.46
	208	2.96
	593	2.74
	784	2.64
	662	2.61
August	**170**	3.82
	199	3.32
	208	2.79
	601	2.69
	370	2.58
September	3495	4.33
	120	4.02
	170	3.19
	299	2.75
	41	2.69
October	1536	4.70
	952	3.23
	551	3.02
	199	3.02
	170	3.01
November	1536	15.52
	14332	5.46
	784	5.11
	147	4.82
	299	4.63
December	0	2.88
	213	2.46
	617	2.37
	96	2.34
	28273	2.08

Table 1. Top 5 nodes with highest betweenness centrality for each month.

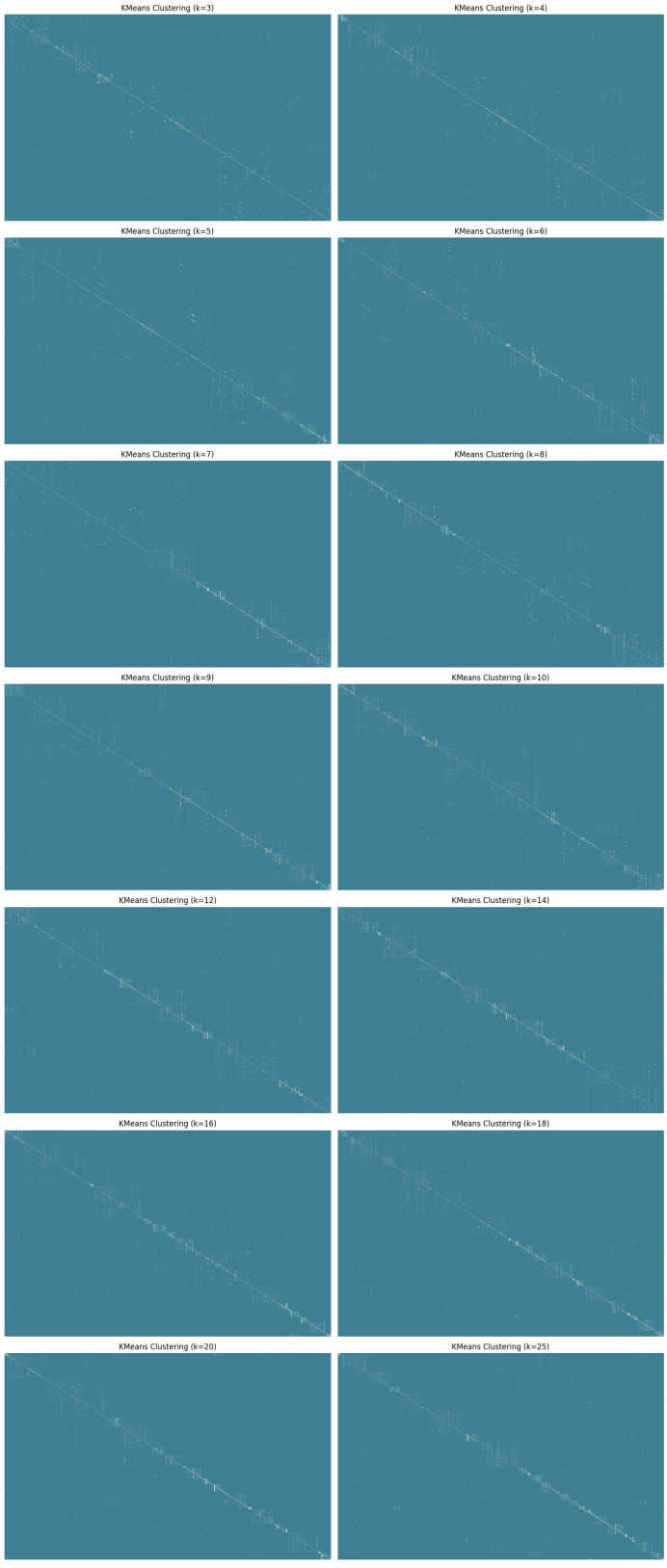


Fig. 11. K-Means clustering

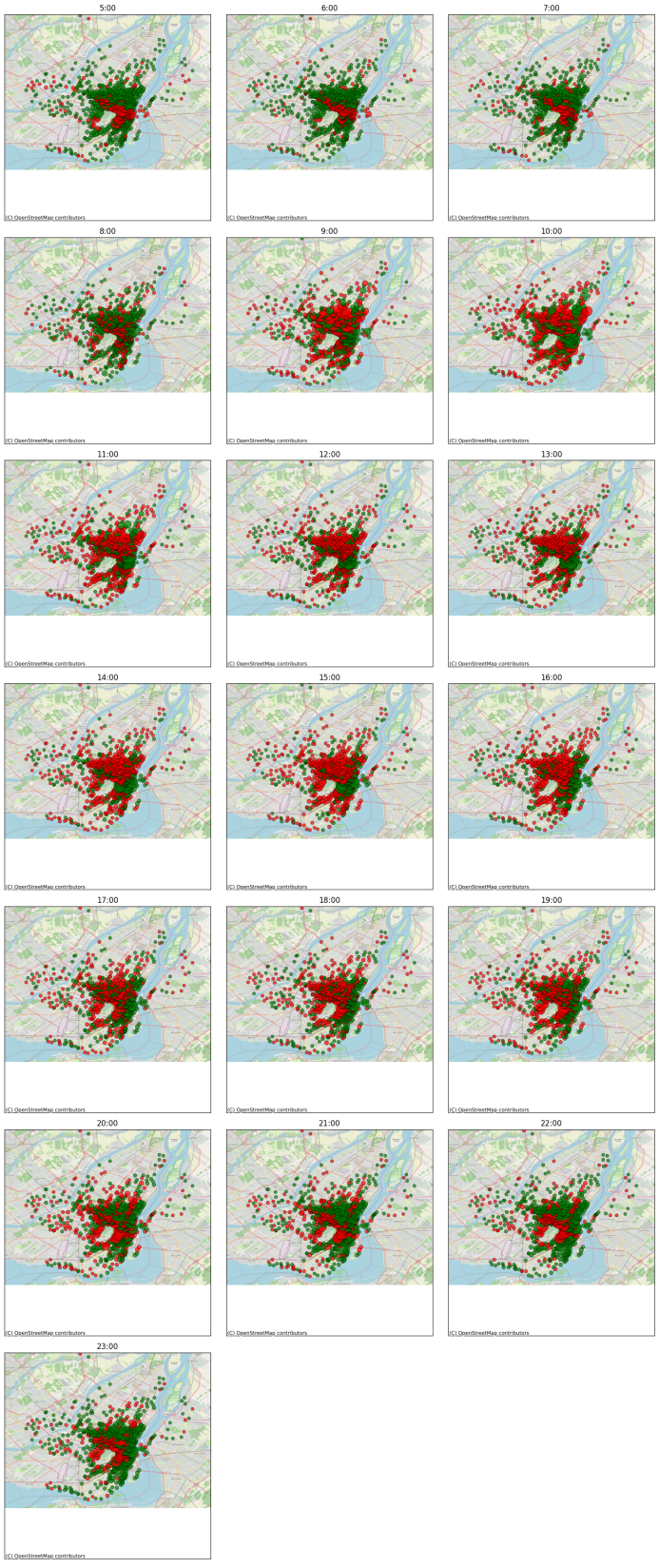
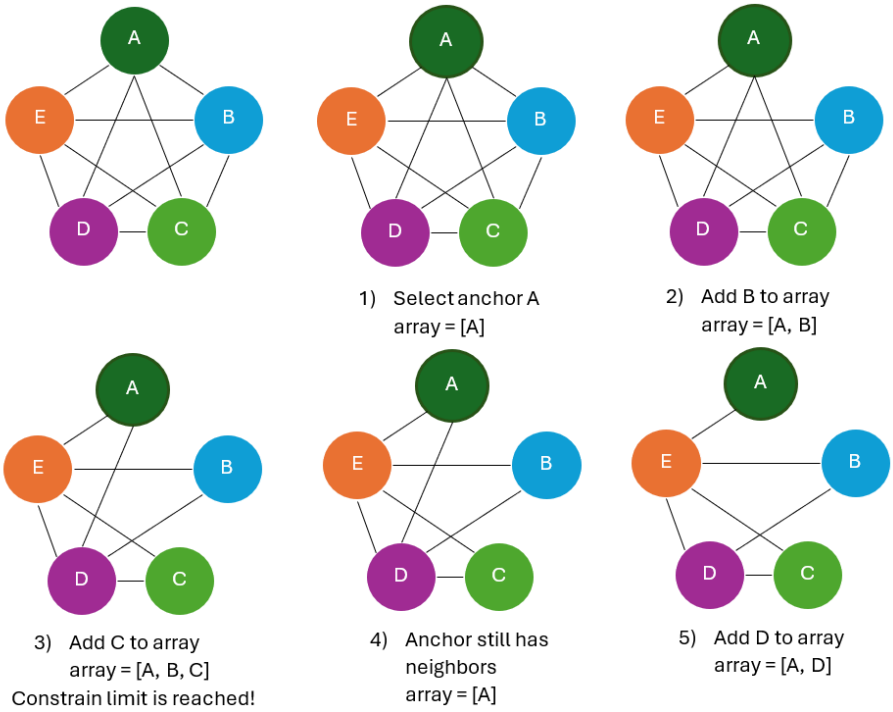
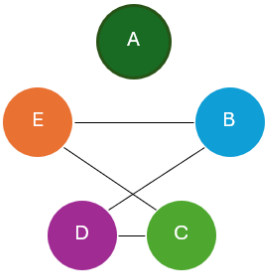
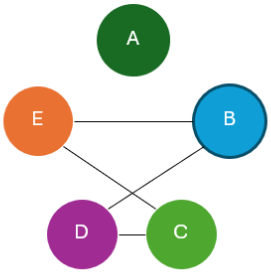


Fig. 12. Station-wise visualisation per hour of the inward and outward flux.

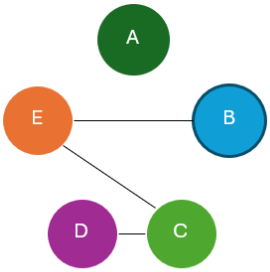

A
E
B
D
C
1) Select anchor A
array = [A]
2) Add B to array
array = [A, B]
3) Add C to array
array = [A, B, C]
Constrain limit is reached!
4) Anchor still has
neighbors
array = [A]
5) Add D to array
array = [A, D]



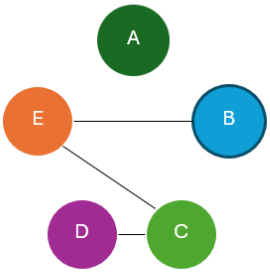
6) Add E to array
array = [A, D, E]
Constrain limit is reached!



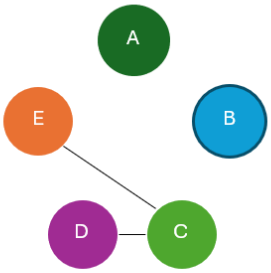
7) Select anchor B
array = [B]



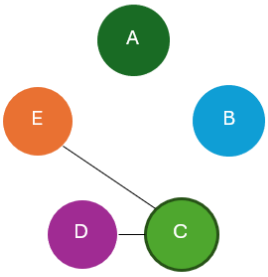
8) Add D to array
array = [B, D]



9) Reset array
array = [B]
We can't add E because it
was already with D (see 6))



10) Add E to array
array = [B, E]



11) Select anchor C
array = [C]

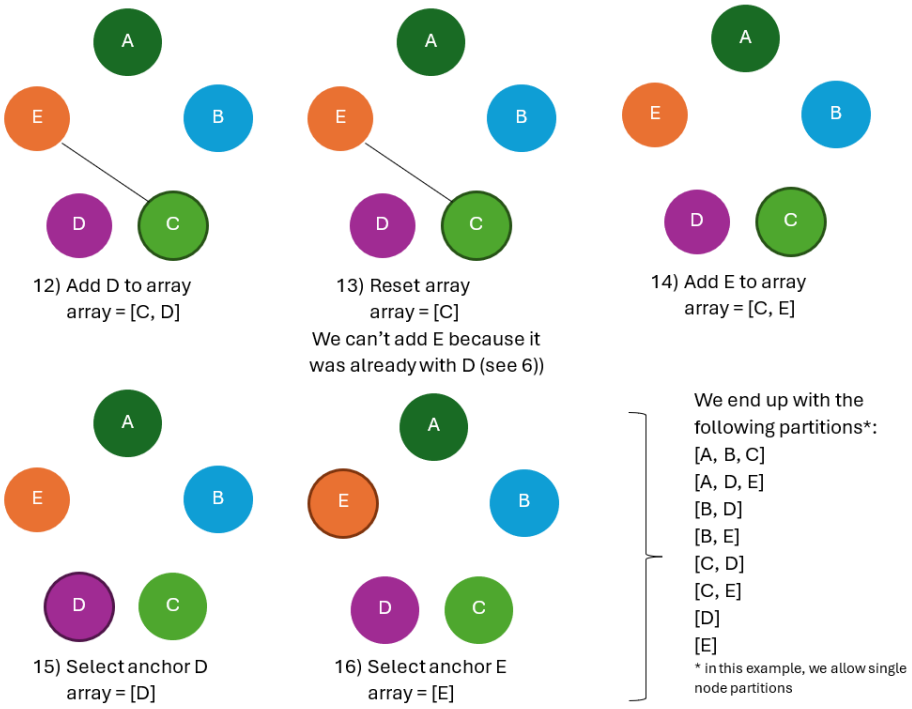


Fig. 13. Example of the partitioning algorithm

```
def create_partitions(input_list, max_size = 3):
partitions = []
memory = {key: set() for key in input_list}
for anchor in input_list:

    temp = input_list.copy()
    temp.remove(anchor)
    while len(temp) > 0:

        partition = [anchor]
        assigned_nodes = []
        for node in temp:
            is_accepted = True

            if node in memory[anchor]:
                # the anchor already know this node
                assigned_nodes.append(node)
            for partition_node in partition:
                if node in memory[partition_node]:
                    is_accepted = False
                    break

            if is_accepted:
                partition.append(node)
                assigned_nodes.append(node)

            if len(partition) >= max_size:
                break

        temp = [x for x in temp if x not in assigned_nodes]

        for partition_node in partition:
            memory[partition_node].update(partition)

        # avoid partition of size 1, not mandatory
        if len(partition) > 1: partitions.append(partition)
return partitions
```

Listing 1.1. Partitioning algorithm

```
def combine_partitions(partitions, max_size=3):
print(type(partitions))
combined_partitions = []
current_partition = []

for sublist in partitions:
    if len(current_partition) + len(sublist) <= max_size:
```

```
            current_partition.extend(sublist)
        else:
            combined_partitions.append(current_partition)
            current_partition = sublist[:]

    # Add the last combined sublist
    if current_partition:
        combined_partitions.append(current_partition)

    return combined_partitions
```

Listing 1.2. Combining partition algorithm

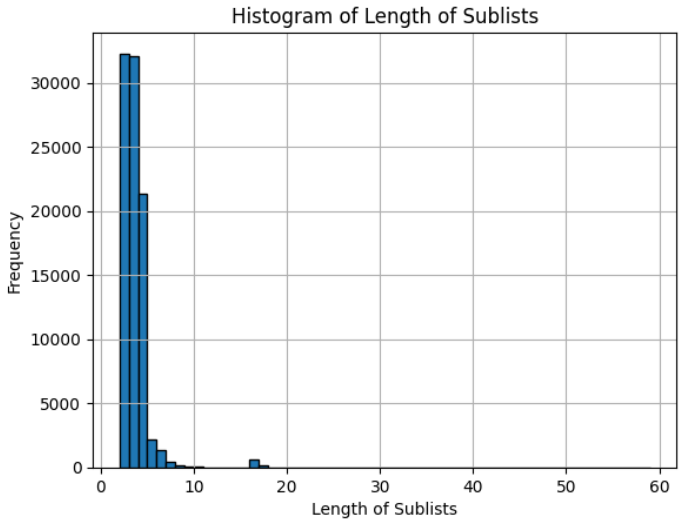


Fig. 14. Partitioning distribution after step 1

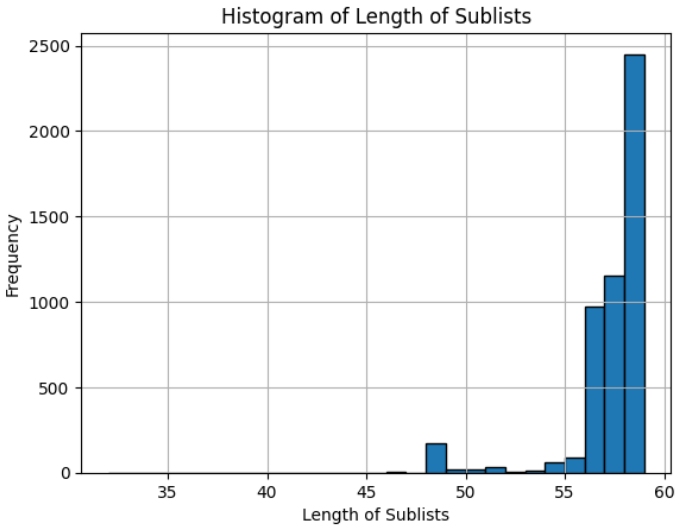


Fig. 15. Partitioning distribution after step 2

```
================================================================================
Model type                                                            Gaussian
Number of observations:                                                   2017
Number of covariates:                                                        7

Global Regression Results
--------------------------------------------------------------------------------
Residual sum of squares:                                            43929.323
Log-likelihood:                                                     -5969.158
AIC:                                                                11952.316
AICc:                                                               11954.388
BIC:                                                                28634.496
R2:                                                                     0.056
Adj. R2:                                                                0.053

Variable                         Est.         SE  t(Est/SE)    p-value
-------------------------------- ---------- ---------- ---------- ----------
X0                              3.184      0.718      4.432      0.000
X1                              0.058      0.015      3.886      0.000
X2                             -0.064      0.019     -3.400      0.001
X3                              0.568      0.226      2.515      0.012
X4                             -0.076      0.052     -1.446      0.148
X5                             -0.000      0.012     -0.006      0.995
X6                              0.122      0.015      8.216      0.000

Geographically Weighted Regression (GWR) Results
--------------------------------------------------------------------------------
Spatial kernel:                                         Adaptive bisquare
Bandwidth used:                                                   429.000

Diagnostic information
--------------------------------------------------------------------------------
Residual sum of squares:                                            35225.031
Effective number of parameters (trace(S)):                             99.425
Degree of freedom (n - trace(S)):                                    1917.575
Sigma estimate:                                                         4.286
Log-likelihood:                                                     -5746.456
AIC:                                                                11693.763
AICc:                                                               11704.397
BIC:                                                                12257.085
R2:                                                                     0.243
Adjusted R2:                                                            0.204
Adj. alpha (95%):                                                       0.004
Adj. critical t value (95%):                                            2.922

Summary Statistics For GWR Parameter Estimates
--------------------------------------------------------------------------------
Variable                   Mean        STD        Min     Median        Max
-------------------- ---------- ---------- ---------- ---------- ----------
X0                        2.837      1.627     -1.318      2.798      7.741
X1                        0.075      0.036     -0.013      0.072      0.164
X2                       -0.075      0.040     -0.214     -0.072      0.020
X3                        0.838      0.476     -0.811      0.817      1.863
X4                       -0.079      0.079     -0.294     -0.069      0.104
X5                        0.001      0.037     -0.072     -0.002      0.091
X6                        0.147      0.063     -0.007      0.155      0.261
================================================================================
```

Fig. 16. GWR Summary with 1% of the dataset