DATA 698 Final Research Project

Predicting Availability of Citi Bike Bicycles in New York City

Philip Tanofsky

2022-11-18

Introduction

New York City offers a multitude of transit options including the subway, buses, rideshare, taxis, and since 2013 a bikeshare option known as Citi Bike. The privately owned bikeshare system boasts a total of 25,000 bicycles and over 1,500 stations in New York City, New York in parts of Manhattan, Brooklyn, Queens, and the Bronx along with stations in Jersey City and Hoboken in New Jersey. The bikeshare option provides a solution for the 'first-' and "last-mile" connectivity in urban areas for individuals wanting to access the subway or bus line without walking. Citi Bike offers real-time availability of bicycles through the Citi Bike app, but the availability of a bicycle at the starting location and availability of a docking station upon arrival is not guaranteed at any time. The availability of bicycles is dependent on the riders themselves and the rebalancing strategies of Citi Bike.

The trip data publicly provided by Citi Bike on a monthly basis can be used to examine ridership patterns and directions of flow based on time, date, and type of user (subscriber or casual). This project will attempt to construct a model of the Citi Bike bikeshare system to predict the availability of one or more bicycles based on location and a future date and time.

The Problem

1. Determine the flow of bicycles throughout the Citi Bike system in New York City to identify neighborhoods with a surplus of bicycles or a dearth of bicycle availability.
2. Identify system rebalancing and its impact on the level of availability at a given station or cluster.
3. Determine the best model to accurately model the availability of bicycles based on day of the week, time of day, location, and number of bicycles needed?

Data Collection & Preprocessing

Data Collection

Citi Bike provides individual bike trip data on a monthly basis available at https://ride.citibikenyc.com/system-data. The September 2022 bike trip data for New York City was downloaded and unzipped. The dataset contains 13 variables for each bike trip originating at a NYC-based docking station. A note on the system data page indicates trips taken by staff to service or inspect the system have been removed from the dataset. Also, any trips below 60 seconds have also been omitted. With this preprocessing by the data maintainers, the remaining trips are considered to be valid bike trips.

- **ride.id:** Unique identifier of the bike trip
- **rideable.type:** Factor variable - classic, electric, and docked
- **started.at:** Timestamp of trip departure
- **ended.at:** Timestamp of trip arrival
- **start.station.name:** Name of departure docking station
- **start.station.id:** Unique identifier of departure docking station
- **end.station.name:** Name of arrival docking station
- **end.station.id:** Unique identifier of arrival docking station
- **start.lat:** Latitude of departure location
- **start.lng:** Longitude of departure location
- **end.lat:** Latitude of arrival location
- **end.lng:** Longitude of arrival location
- **member.casual:** Factor variable for user type - member or casual

Based on the `started.at` and `ended.at` variables, four variables are derived for each bike trip.

- **day:** Day of the month
- **start.hour:** Hour of the trip departure
- **weekday:** Day of the week for the trip
- **trip.duration:** Duration of bike trip in minutes.

The NYC Open Data (free public data published by New York City agencies and partners) provides a GeoJSON file for the polygons defining each neighborhood in NYC according for the 2010 Neighborhood Tabulation Areas (NTAs). Each NTA is associated with one of the five NYC boroughs. (https://data.cityofnewyork.us/City-Government/2010-Neighborhood-Tabulation-Areas-NTAs-/cpf4-rkhq)

The elevation of each Citi Bike docking station is determined using the R library `elevatr` based on the latitude and longitude of each station. The elevation is defined in meters above sea level.

- **elevation: Units above sea level**
- **elev.units: Unit of measurement for elevation**

Data Preprocessing

Upon initial inspection of the 3,507,123 bike trips in September 2022, a total of 8,012 entries do not contain an `end.station.id` and `end.station.name` listed. Of that count, 3,838 do not contain an `end.lat` and `end.lng` values. These 8,012 without a defined destination docking station will be defined as 'Abandoned' meaning the user did not properly dock the bike. For this purpose, the `end.lat` and `end.lng` will be removed as the abandoned bikes temporarily remove a bike from the bikeshare system. Another rider cannot rent an abandoned bike until the bike is properly docked.

Evaluation of the `end.station.id` for the NYC based bike trips includes docking stations located in New Jersey. The Citi Bike bikeshare system does include docking stations in Jersey City and Hoboken. A number of bike trips end in New Jersey which does remove the bike from the NYC-based docking stations of which this research is focused.

Surplus Calculation

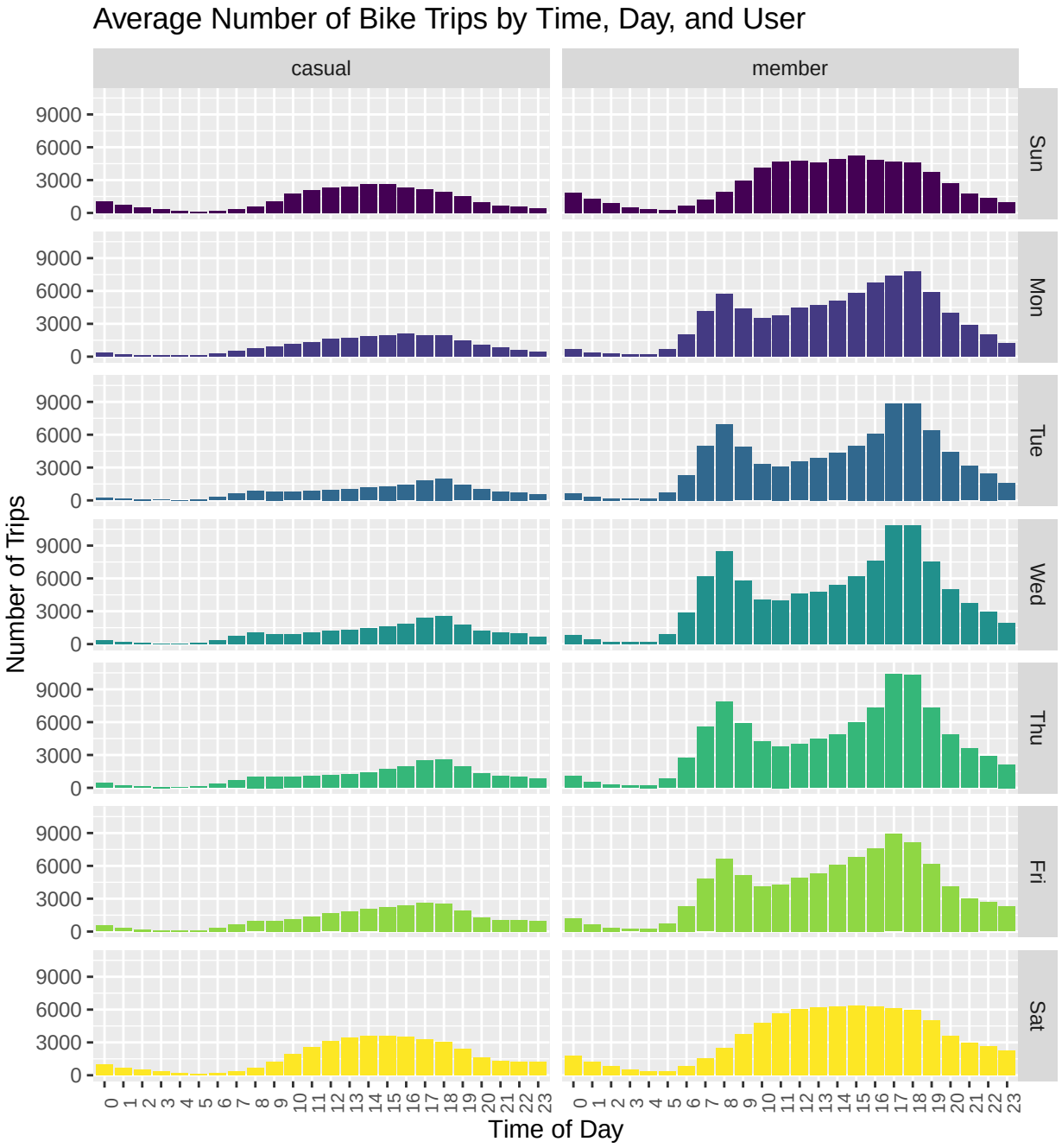
The individual bike trip information was sorted by timestamp and grouped by docking station for each 15-minute interval to count the number of bikes departing and the number bikes arriving. By subtracting the number of departures from the number of arrivals for each station for each interval, we are able to determine the running increase or decrease of bikes at the docking station. This total is defined as the variable `surplus`. A summation of the `surplus` for each docking station is calculated over the course of the month to determine which docking stations are more likely departure stations or arrival stations.

Data Exploration & Analysis

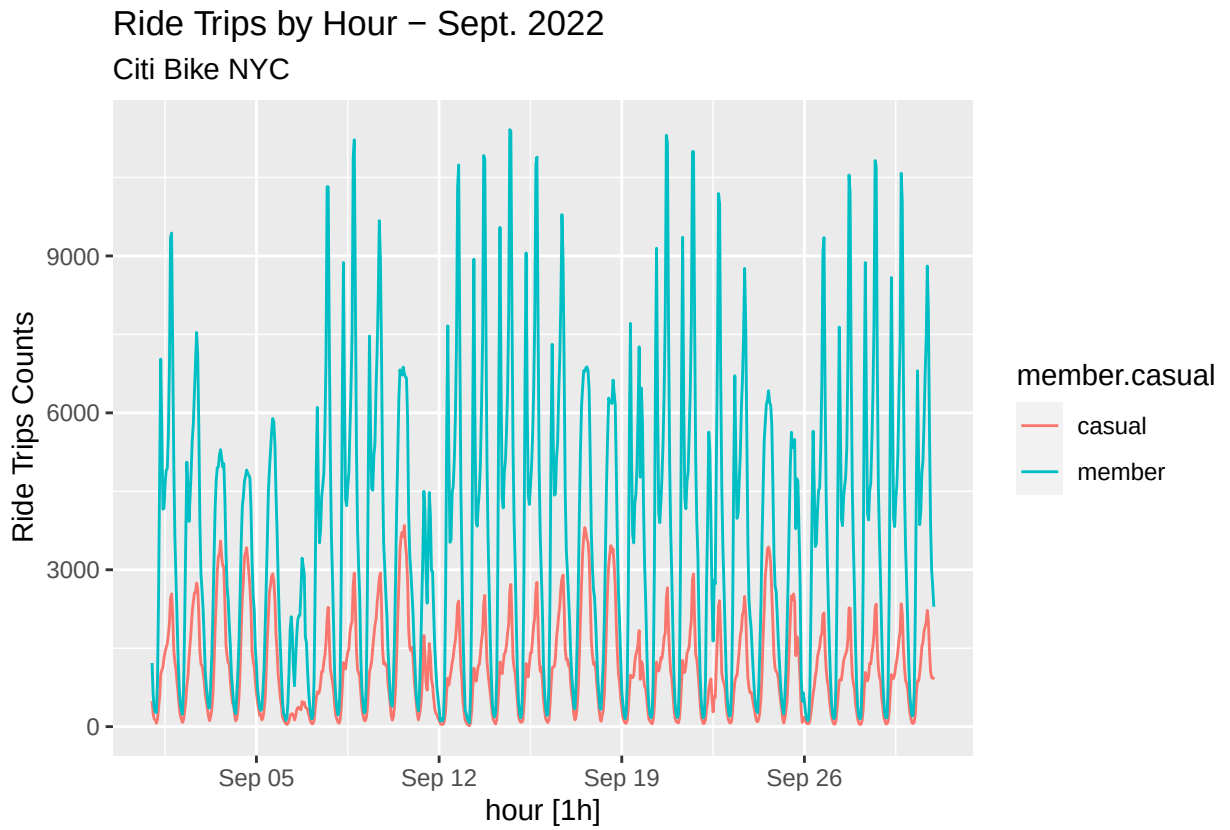
Exploratory data analysis is performed on the Citi Bike trips for September 2022 to evaluate the patterns of bike use and identify docking stations with surplus. First, the count of bike trips are assessed to find the high volume days of the week and time of day. Next, the duration of bike trips are evaluated to assess when bikes are individually likely to be unavailable longer. Finally, the surplus of bikes by docking station and borough are analyzed to determine which areas of the New York City are more prone to having lower bike availability.

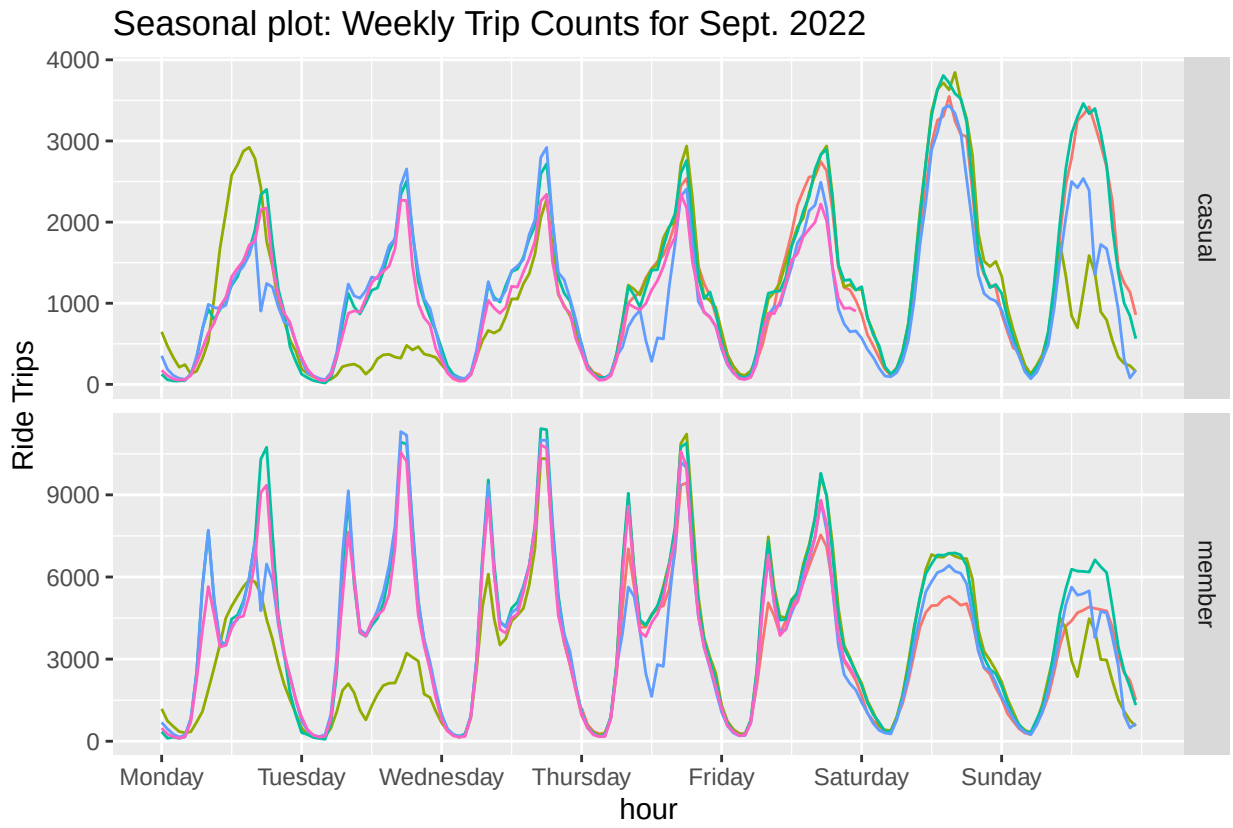
Count of Bike Trips

The count of bike trips are separated by user type - member and casual. With the separation by user type, two distinct patterns emerge of bike use over the course of the day and over the course of a week. The member trips are more likely to follow the workday pattern of spikes in the morning and evening as individuals are traveling to work or from work. The casual users show a pattern not necessarily indicative of the workday but instead of tourist or recreational use. The member trips account for an overall higher volume of trips.



The member user counts show rush-hour spikes for members on weekdays whereas casual users do show higher counts around 5pm and 6pm on weekdays. The comparison of the two user groups also points to greater usage overall by member users. For everyday of the week, the member average member counts per hour are greater than the casual users. On weekend days, both user groups show a pattern indicative of recreational use with plateau use during the middle of the day and without distinct spikes found on the weekdays. Also, the higher usage of by member users throughout the middle of the day may indicate even if someone is a member the primary reason may not be transportation to and from work.


Ride Trips by Hour − Sept. 2022
Citi Bike NYC
Ride Trips Counts
9000
6000
3000
0
Sep 05
Sep 12
Sep 19
Sep 26
hour [1h]
member.casual
casual
member


Seasonal plot: Weekly Trip Counts for Sept. 2022
Ride Trips
4000
3000
2000
1000
0
casual
9000
6000
3000
0
member
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
hour

The time-series chart 'Ride Trips by Hour' confirms the higher usage by members with a clear pattern of weekday volumes corresponding to transportation for work purposes. The casual users follow a daily pattern with increases toward the end of the week - Thursday through Saturday.

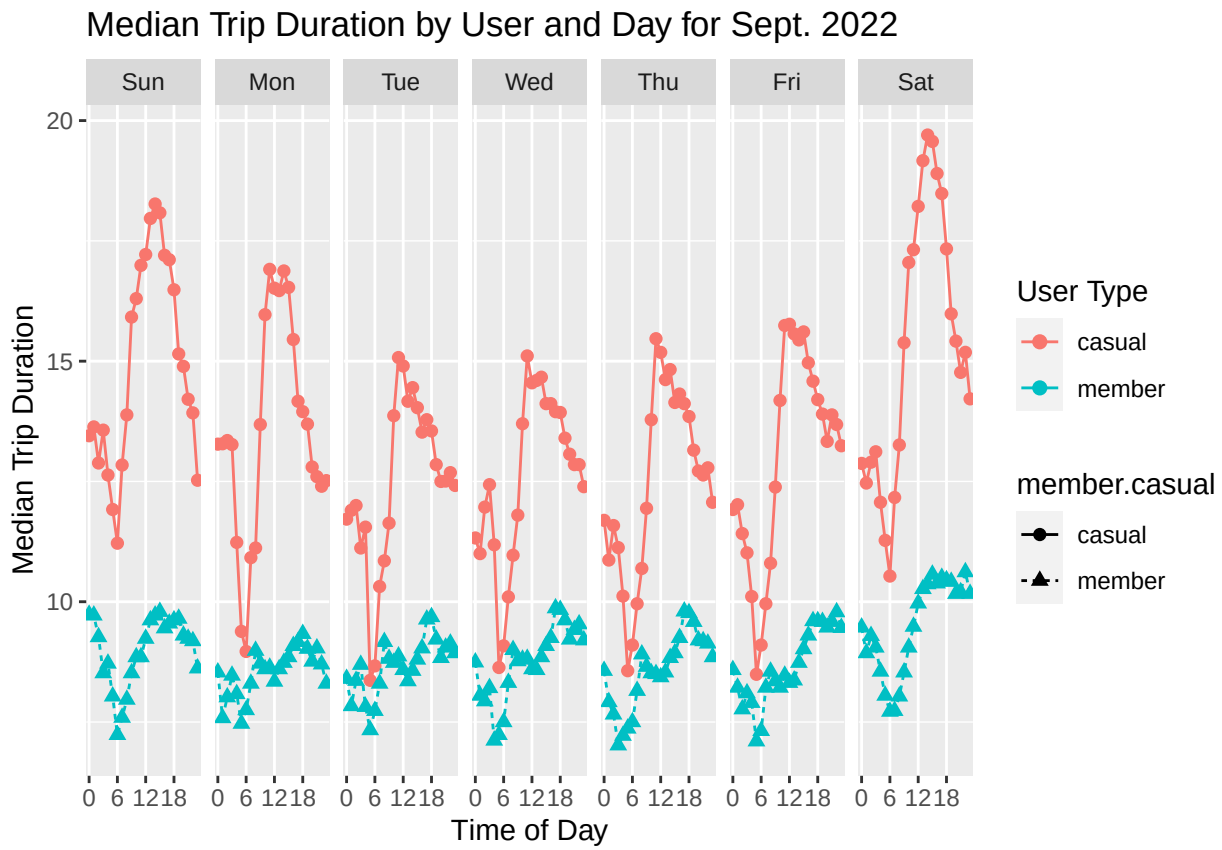
The weekly seasonal plot 'Seasonal plot: Weekly Trip Counts' denotes the same pattern week over week. Based on the plot, the bike trips show consistency from week to week based on member users utilizing the bikes for work and casual users renting the bikes for recreational purposes.

By count of trips per day of the week, Wednesday appears the highest volume day. Given the data comes from September 2022, we believe this observation is a result of pandemic return-to-office policies in which individuals more likely to return to office during the middle of the week instead of Monday or Friday for those with hybrid schedules.

Note: On Tuesday, Sept, 6 2022, the weather consisted of rain the entire day and thus a clear drop in use is evident in both user groups.

Duration of Bike Trips

Next, we evaluate the duration of bike trips to better understand longevity of individual bike unavailability along with reason for trip. As expected, the average bike trip for all users and all times are below 30 minutes as the rental defined time is 30 minutes with users incurring additional fees beyond the base time limit.



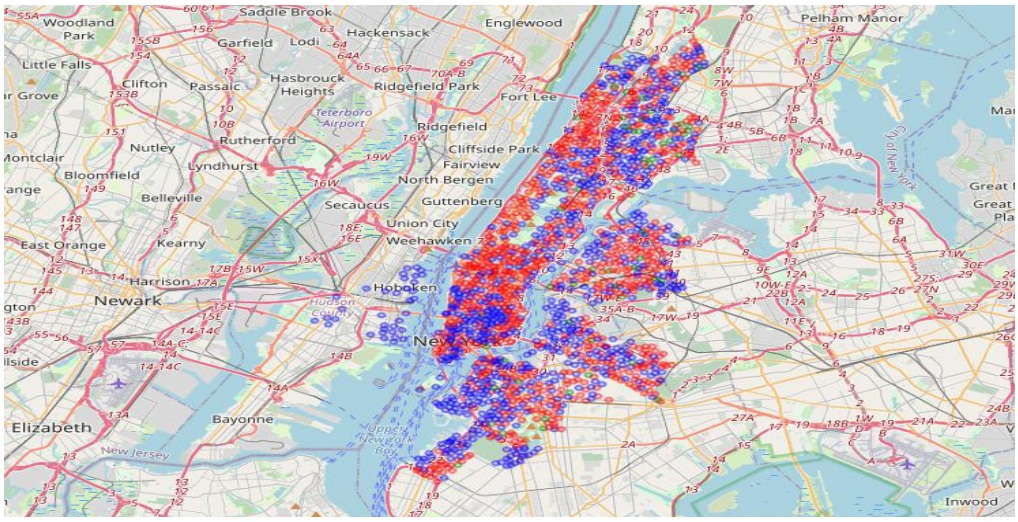
The chart of average trip duration by user type shows a clear distinction between the user types. Member users tend to average bike trips of 10 minutes or less throughout the week, whereas the casual users have a greater variance in average duration based on time of day and day of the week. The member users only average trips greater than 10 minutes during the afternoon and evening on Saturday while casual users typically average even longer trips of greater than 17 minutes on the weekend. Casual users tend to reach of peak of greater than 15 minutes on average everyday of the week, particularly around lunch on weekdays.

We observe that member users tend to have longer average trips during the morning and evening rush hours on weekdays with a slight deviation on Friday evening as the weekend starts. The afternoon and evening hours of Saturday and Sunday show longer trips for members as likely the result of recreational trips.

Docking Station Surplus

After constructing a matrix of bike arrivals and departures for each NYC-based docking station for every 15-minute interval of September 2022, we calculate the overall surplus or shortage of bikes. The researchers note, the shortage of bikes can logically not fall below zero without the introduction of rebalancing throughout the bikeshare system. The following analysis confirms the rebalancing of bikes across the docking stations occurs to ensure popular departure docking stations have bikes available despite the dearth of bike trip arrivals.

Overall Monthly Surplus/Shortage by Docking Station



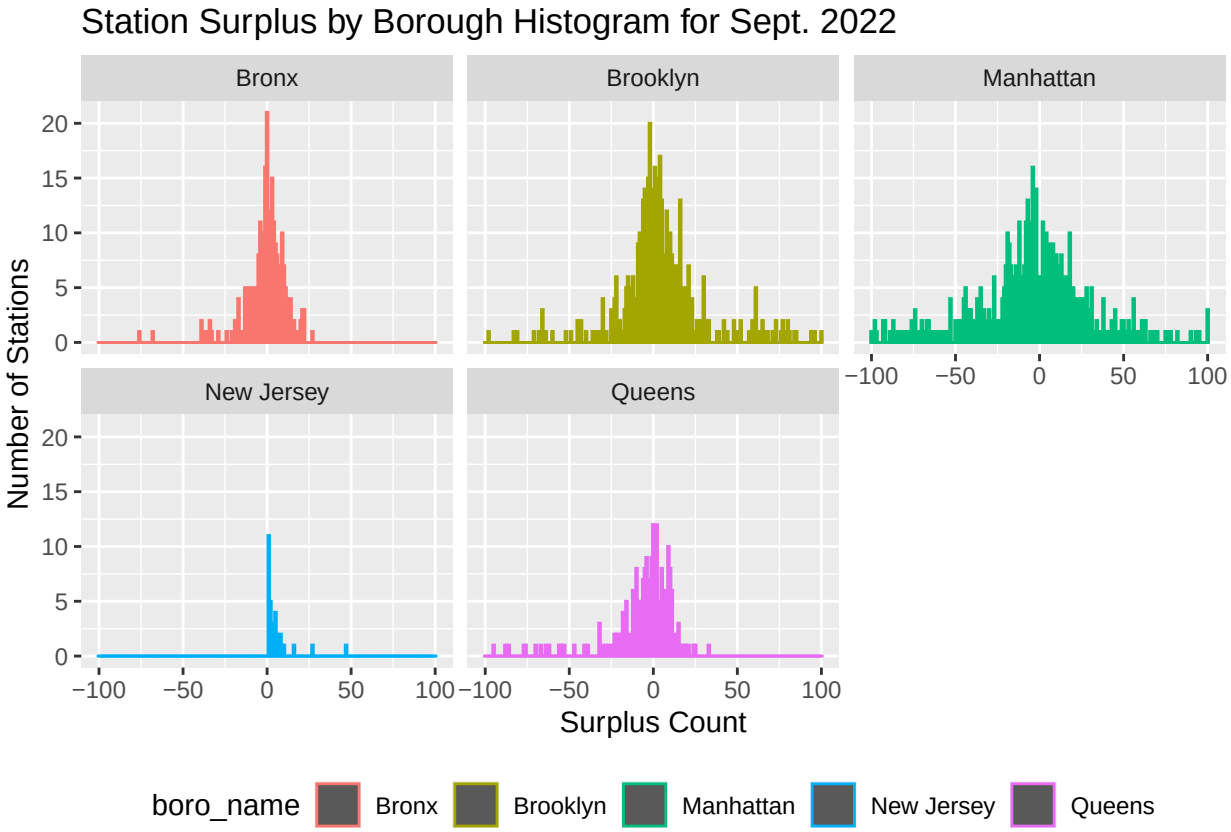
The plot of docking stations across the New York City maps the location of every docking station along with a color to indicate a net positive, negative or even amount of bikes for the given month. The blue stations are net positive, and red stations indicate net negative while green stations are even for the entire month. Of the 1656 docking stations represented, 799 ended with a surplus, 804 with a shortage, and 53 with an even count.

The plot does indicate several blue-colored docking stations in New Jersey. As the dataset does not contain any bike trips originating in NYC, a number of trips have an ending station in Jersey City or Hoboken. The New Jersey based stations are guaranteed to be blue based on that dataset without any departing trips from those docking stations.

The dataset does encompass 472,920 valid combinations of departure and arrival docking stations. Of the valid combinations 15 of the top 20 combinations are the same departure and arrival docking stations, indicative of recreational bike trips. Of the aforementioned valid combinations, 471,313 denote travel between two different docking stations.

Surplus by Borough

With almost an even number of docking stations with a net shortage and a net surplus, we evaluate the shortage and surplus by borough.



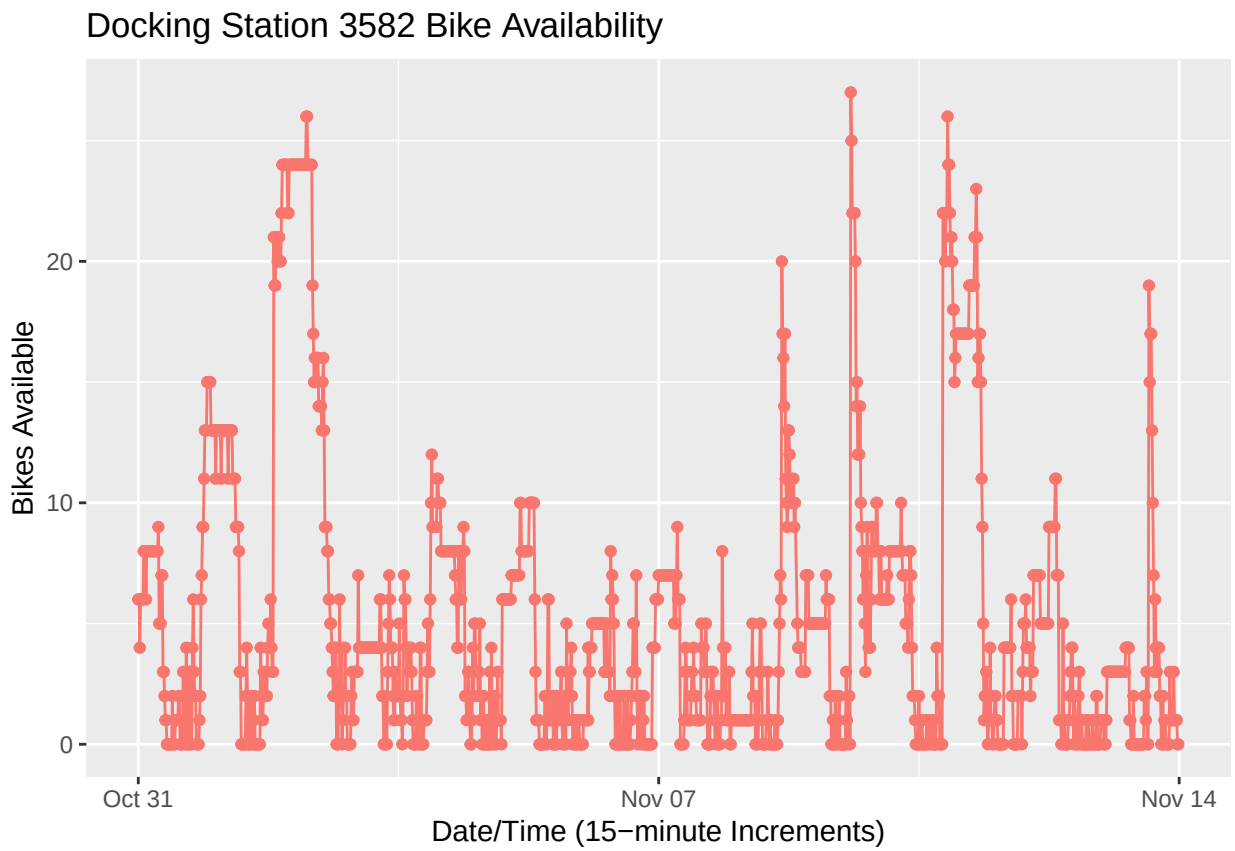
The plot denotes a near normal distribution for the four boroughs included in the dataset - the Bronx, Brooklyn, Manhattan, and Queens. As noted previously, the dataset contains only trips originating in New York City, so the plot for New Jersey only indicates docking stations surpluses. The balanced distribution across the four NYC boroughs likely demonstrates the bike trips are contained within a borough. With average duration less than 10 minutes for member users and less than 20 minutes for casual users, the distance traveled for all bike trips is likely less than three miles. The 30-minute base rental rate dictates a limited travel distance which inhibits users from traveling across boroughs via Citi Bike.

Rebalancing Example

Citi Bike reports a rebalancing of 70117 bicycles in September 2022 (https://mot-marketing-whitelabel-prod.s3.amazonaws.com/nyc/September-2022-Citi-Bike-Monthly-Report.pdf).

From report: There were 1625 active stations at the end of the month. The average bike fleet last month was 24,514.0 with 24,327 bikes in the fleet on the last day of the month.

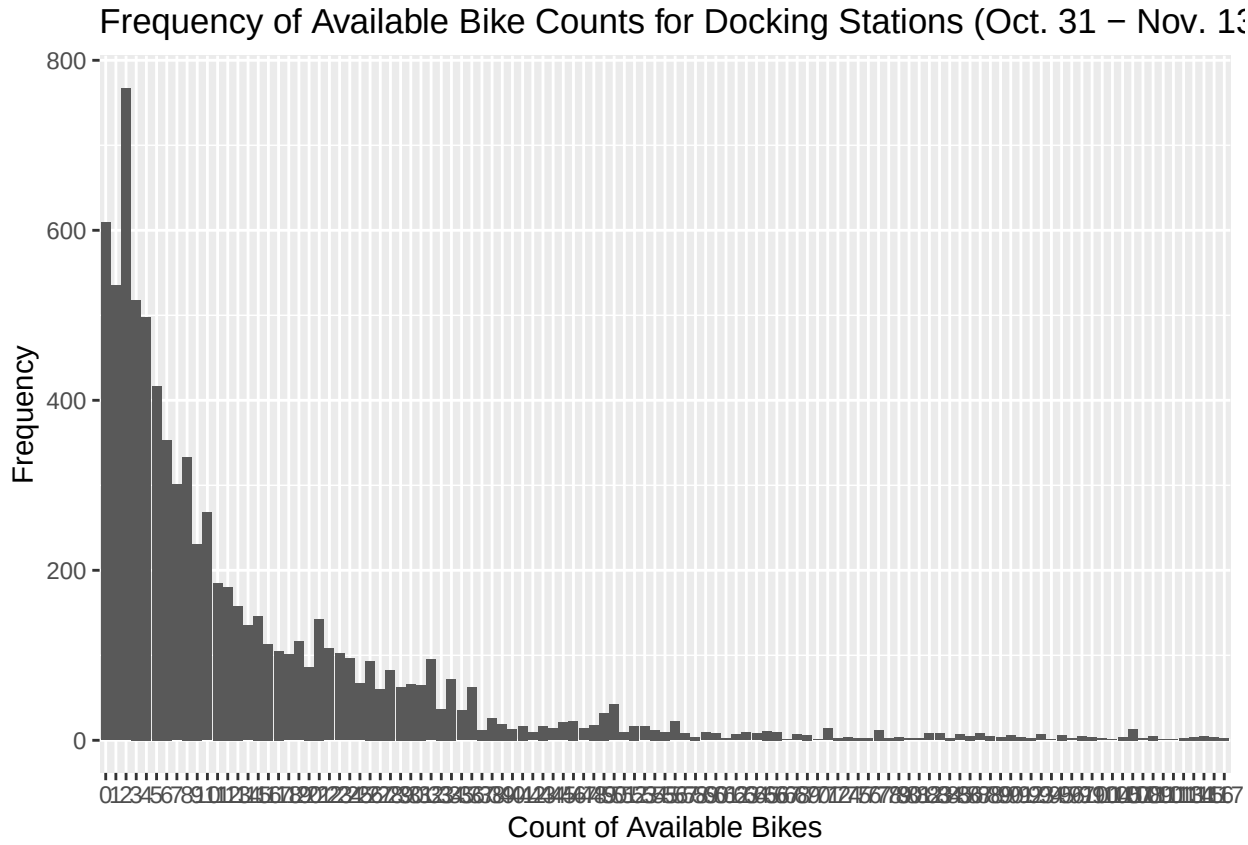
9.6% or almost 10% of bikes rebalanced daily



The above plot shows the occurrence of rebalancing at station 3582 in Brooklyn. Over the course of the two-week period, the sharp spikes which indicate rebalancing are not consistent.

Station Clustering

Filter the stations to NTA name of "Crown Heights North" which results in 11 stations. With over 1650 stations and a walking distance of 400 meters (or one quarter of a mile), the clustering of stations results in over 700 unique clusters. The count of clusters caused vector memory issues in RStudio. In order to continue the research approach, the stations under consideration were decreased to one neighborhood, or NTA, in Brooklyn. The NTA "Crown Heights North" contains 11 docking stations. The clustering of the 11 docking stations based on a distance of 400 meters results in 6 unique clusters, or approximately two docking stations per cluster.



Modeling

Inputs to the models are cluster identifier, day of the week, and time of day. The model inputs would represent a map-based application in which a user selects a point on the map along with day and time to receive an estimation of the bikes available at that future date and time. Based on the user-provided longitude and latitude, a nearest neighbor search is performed to identify the nearest docking station. The translation of the longitude and latitude to the corresponding cluster of the nearest-neighbor docking station would be the input to the model.

For evaluating the models, the two full weeks of data spanning Oct. 31 through Nov. 13 are partitioned into a training set of 80% and a test set of 20%. The dataset follows a time series pattern but in order to predict a specific day and time of the week agnostic of date, regression models for count data are used. The range of regression models for count data considered are Poisson, Quasi-poisson, Negative Binomial, Hurdle, and Zero-Inflated.

The mean of the bikes available by cluster and time interval is 14.0 and the population variance over the dataset is 235.2.

Poisson Model The Poisson model is a generalized linear model for count data with the assumption that the variance is equal to the mean.

```
##
## Call:
## glm(formula = bikes.avail.count ~ cluster + time + weekday, family = "poisson",
```

```
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -7.4597  -1.7847  -0.3791   1.1239   8.4367
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)   3.027394   0.026155 115.750  < 2e-16 ***
## cluster2     -0.212459   0.012601 -16.861  < 2e-16 ***
## cluster3      0.952189   0.009949  95.707  < 2e-16 ***
## cluster4     -0.031446   0.011920  -2.638 0.008339 **
## cluster5     -0.354699   0.013160 -26.952  < 2e-16 ***
## cluster6     -1.356997   0.018381 -73.826  < 2e-16 ***
## time00:15:00 -0.034474   0.036464  -0.945 0.344434
## time00:30:00  0.065799   0.035442   1.857 0.063379 .
## time00:45:00  0.068291   0.035104   1.945 0.051726 .
## time01:00:00  0.088188   0.034829   2.532 0.011341 *
## time01:15:00  0.119915   0.035966   3.334 0.000856 ***
## time01:30:00  0.182831   0.035938   5.087 3.63e-07 ***
## time01:45:00  0.192717   0.033802   5.701 1.19e-08 ***
## time02:00:00  0.190189   0.033742   5.637 1.73e-08 ***
## time02:15:00  0.161024   0.034308   4.693 2.69e-06 ***
## time02:30:00  0.178540   0.034639   5.154 2.55e-07 ***
## time02:45:00  0.194827   0.033833   5.758 8.49e-09 ***
## time03:00:00  0.205296   0.033743   6.084 1.17e-09 ***
## time03:15:00  0.221456   0.035544   6.230 4.65e-10 ***
## time03:30:00  0.189802   0.034041   5.576 2.47e-08 ***
## time03:45:00  0.234113   0.034075   6.871 6.40e-12 ***
## time04:00:00  0.176451   0.033905   5.204 1.95e-07 ***
## time04:15:00  0.211681   0.033561   6.307 2.84e-10 ***
## time04:30:00  0.250103   0.032937   7.593 3.11e-14 ***
## time04:45:00  0.235313   0.035199   6.685 2.31e-11 ***
## time05:00:00  0.214041   0.033406   6.407 1.48e-10 ***
## time05:15:00  0.188136   0.034336   5.479 4.27e-08 ***
## time05:30:00  0.231422   0.033839   6.839 7.98e-12 ***
## time05:45:00  0.169399   0.036119   4.690 2.73e-06 ***
## time06:00:00  0.194328   0.033094   5.872 4.31e-09 ***
## time06:15:00  0.147418   0.035109   4.199 2.68e-05 ***
## time06:30:00  0.144038   0.034176   4.215 2.50e-05 ***
## time06:45:00  0.090176   0.035653   2.529 0.011431 *
## time07:00:00  0.036372   0.035495   1.025 0.305506
## time07:15:00  0.037213   0.035741   1.041 0.297792
## time07:30:00 -0.043794   0.036531  -1.199 0.230606
## time07:45:00 -0.086541   0.038090  -2.272 0.023086 *
## time08:00:00 -0.259651   0.039482  -6.577 4.82e-11 ***
## time08:15:00 -0.405077   0.039600 -10.229  < 2e-16 ***
## time08:30:00 -0.469173   0.041166 -11.397  < 2e-16 ***
## time08:45:00 -0.537492   0.041635 -12.910  < 2e-16 ***
## time09:00:00 -0.662938   0.041871 -15.833  < 2e-16 ***
## time09:15:00 -0.763519   0.044325 -17.225  < 2e-16 ***
## time09:30:00 -0.830695   0.045540 -18.241  < 2e-16 ***
## time09:45:00 -1.025892   0.052359 -19.594  < 2e-16 ***
## time10:00:00 -1.184834   0.053911 -21.978  < 2e-16 ***
```

```
## time10:15:00 -1.119755   0.052105 -21.490  < 2e-16 ***
## time10:30:00 -1.248594   0.059132 -21.115  < 2e-16 ***
## time10:45:00 -1.242307   0.054098 -22.964  < 2e-16 ***
## time11:00:00 -1.322015   0.058593 -22.563  < 2e-16 ***
## time11:15:00 -1.321669   0.056925 -23.218  < 2e-16 ***
## time11:30:00 -1.469130   0.060216 -24.398  < 2e-16 ***
## time11:45:00 -1.458215   0.060140 -24.247  < 2e-16 ***
## time12:00:00 -1.467967   0.062899 -23.338  < 2e-16 ***
## time12:15:00 -1.534490   0.060824 -25.228  < 2e-16 ***
## time12:30:00 -1.590254   0.060821 -26.146  < 2e-16 ***
## time12:45:00 -1.640441   0.063160 -25.973  < 2e-16 ***
## time13:00:00 -1.722559   0.065984 -26.106  < 2e-16 ***
## time13:15:00 -1.757796   0.071332 -24.642  < 2e-16 ***
## time13:30:00 -1.635196   0.064666 -25.287  < 2e-16 ***
## time13:45:00 -1.716339   0.065465 -26.218  < 2e-16 ***
## time14:00:00 -1.642404   0.063804 -25.741  < 2e-16 ***
## time14:15:00 -1.443973   0.058919 -24.508  < 2e-16 ***
## time14:30:00 -1.788588   0.067296 -26.578  < 2e-16 ***
## time14:45:00 -1.626425   0.064762 -25.114  < 2e-16 ***
## time15:00:00 -1.693216   0.062633 -27.034  < 2e-16 ***
## time15:15:00 -1.581433   0.065367 -24.193  < 2e-16 ***
## time15:30:00 -1.461387   0.060441 -24.179  < 2e-16 ***
## time15:45:00 -1.410174   0.058046 -24.294  < 2e-16 ***
## time16:00:00 -1.561763   0.062378 -25.037  < 2e-16 ***
## time16:15:00 -1.630680   0.063709 -25.596  < 2e-16 ***
## time16:30:00 -1.537787   0.063256 -24.311  < 2e-16 ***
## time16:45:00 -1.461326   0.058182 -25.117  < 2e-16 ***
## time17:00:00 -1.327290   0.057539 -23.068  < 2e-16 ***
## time17:15:00 -1.263322   0.057055 -22.142  < 2e-16 ***
## time17:30:00 -1.400349   0.056803 -24.653  < 2e-16 ***
## time17:45:00 -1.283768   0.054550 -23.534  < 2e-16 ***
## time18:00:00 -1.306517   0.055608 -23.495  < 2e-16 ***
## time18:15:00 -1.215287   0.052658 -23.079  < 2e-16 ***
## time18:30:00 -1.232980   0.053105 -23.218  < 2e-16 ***
## time18:45:00 -1.258770   0.052527 -23.964  < 2e-16 ***
## time19:00:00 -1.206720   0.056453 -21.376  < 2e-16 ***
## time19:15:00 -1.197097   0.052758 -22.690  < 2e-16 ***
## time19:30:00 -1.137081   0.053530 -21.242  < 2e-16 ***
## time19:45:00 -1.081515   0.052410 -20.636  < 2e-16 ***
## time20:00:00 -1.026292   0.050684 -20.249  < 2e-16 ***
## time20:15:00 -0.927748   0.048668 -19.063  < 2e-16 ***
## time20:30:00 -0.936490   0.051311 -18.251  < 2e-16 ***
## time20:45:00 -0.895753   0.048087 -18.628  < 2e-16 ***
## time21:00:00 -0.727182   0.045236 -16.075  < 2e-16 ***
## time21:15:00 -0.660247   0.042545 -15.519  < 2e-16 ***
## time21:30:00 -0.590980   0.042269 -13.981  < 2e-16 ***
## time21:45:00 -0.526844   0.044933 -11.725  < 2e-16 ***
## time22:00:00 -0.484159   0.040326 -12.006  < 2e-16 ***
## time22:15:00 -0.424325   0.040513 -10.474  < 2e-16 ***
## time22:30:00 -0.470320   0.042021 -11.192  < 2e-16 ***
## time22:45:00 -0.325715   0.041014  -7.942 2.00e-15 ***
## time23:00:00 -0.275513   0.037938  -7.262 3.81e-13 ***
## time23:15:00 -0.236609   0.039665  -5.965 2.44e-09 ***
## time23:30:00 -0.132927   0.037359  -3.558 0.000374 ***
```

```
## time23:45:00 -0.138450   0.037206  -3.721 0.000198 ***
## weekday.L    -0.111860   0.009594 -11.659  < 2e-16 ***
## weekday.Q    -0.358574   0.009429 -38.029  < 2e-16 ***
## weekday.C     0.098315   0.009126  10.773  < 2e-16 ***
## weekday^4    -0.105129   0.008656 -12.145  < 2e-16 ***
## weekday^5     0.096484   0.008650  11.154  < 2e-16 ***
## weekday^6    -0.113754   0.008367 -13.595  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 108886  on 6463  degrees of freedom
## Residual deviance:  30206  on 6357  degrees of freedom
## AIC: 54262
##
## Number of Fisher Scoring iterations: 5
```

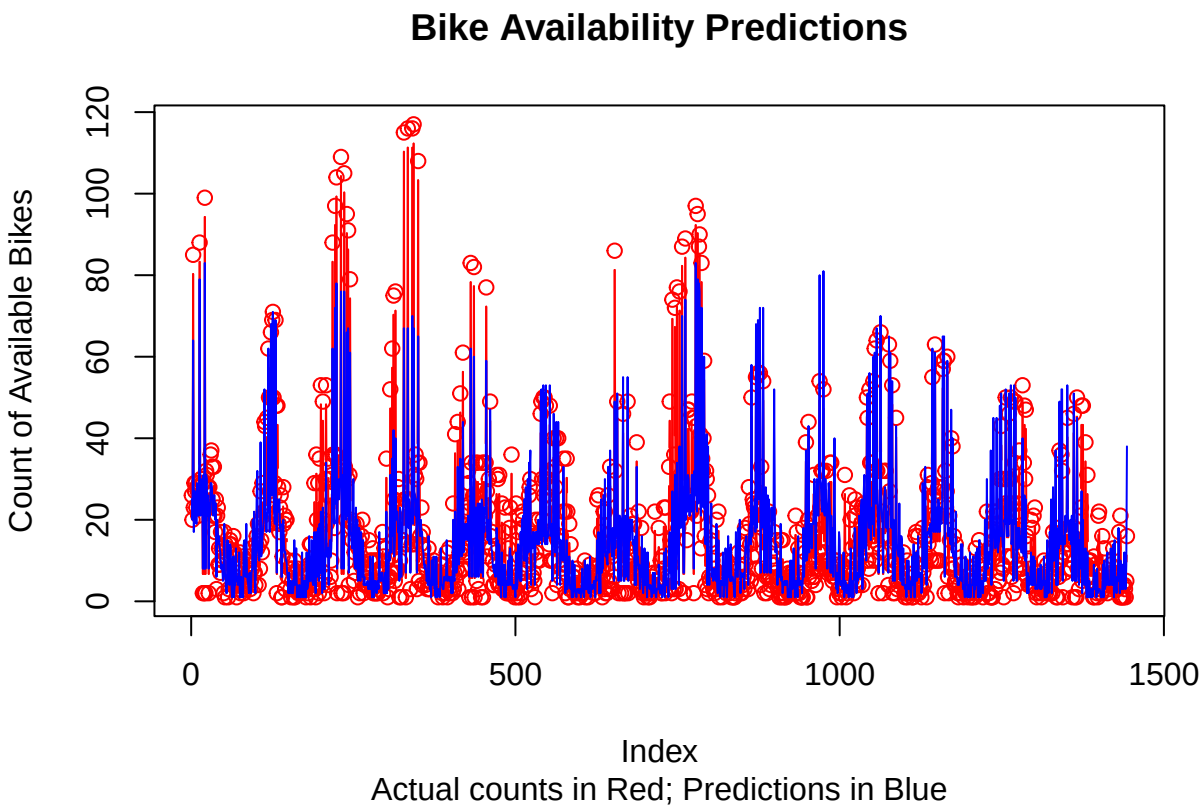
Explanation the result of the model

```
## # A tibble: 1 x 8
##   null.deviance df.null  logLik    AIC    BIC deviance df.residual  nobs
##           <dbl>   <int>   <dbl>  <dbl>  <dbl>    <dbl>       <int> <int>
## 1       108886.    6463 -27024. 54262. 54987.   30206.        6357  6464
```

Glance explanation

```
##
##  Overdispersion test
##
## data:  mod1
## z = 33.687, p-value < 2.2e-16
## alternative hypothesis: true dispersion is greater than 1
## sample estimates:
## dispersion
##   5.031797
```

The overdispersion test indicates a dispersion greater than 5 in which true dispersion is defined as 1 or more.



Quasi-Poisson Model The Quasi-Poisson Model is a generalized linear model to model count data, for overdispersed count variable in which the variance is a linear function of the mean.

Negative Binomial Model The Negative Binomial model is a generalized linear model for count data in which the variance is greater than the mean, and the variance is a quadratic function of the mean.

Hurdle The Hurdle model is another model for count data consisting of two parts - the probability of attaining a value of zero and a second part to model probability of a non-zero value. The Hurdle models used in this research use a base of Poisson and Negative Binomial.

Zero-Inflated Model A Zero-Inflated Model builds a model for a distribution that allows for frequent zero-values observations. The Zero-Inflated models used in this research use a base of Poisson and Negative Binomial.

Model Results

The count model results show the best predictive performance by measuring the Root-Mean-Square Error (RMSE) and Mean Absolute Error (MAE) as the Poisson and Quasi-Poisson models. Overall, the performance of the models falls in a small range with RMSE ranging in 8.68 to 9.42. These results show that the average distance of approximately 9 between the predicted values from the model and the values from the dataset.

```
##                  models     rmse      mae
## 1               Poisson 8.675188 5.961929
## 2         Quasi-Poisson 8.675188 5.961929
## 3        Hurdle Poisson 8.733759 5.965102
## 4 Zero-Inflated Poisson 8.747335 5.996827
## 5     Negative Binomial 9.360348 6.381345
## 6             Hurdle NB 9.389823 6.355330
## 7      Zero-Inflated NB 9.419880 6.421954
```

Prediction Proof of Concept

To make the model functional, we created a function to predict the number of bikes available given a longitude and latitude, time of day, and day of the week. The longitude and latitude are translated into a cluster identifier based on a nearest neighbor search of the docking stations.

The below example returns a predicted availability of 2 bikes at 2:30PM on Saturday within 400 meters from the corner of Classon Avenue and St. John's Place in Crown Heights Brooklyn.

```
# Test case for the function
predict_num_bikes_avail(-73.960859, 40.67355, "14:30:00", "Sat")
```

```
## [1] 2
```

Closing Remarks

The exploratory analysis of the Citi Bike dataset for NYC-based trips in September 2022 shows a consistent pattern of use dependent on time of day and day of the week. The pattern persists week over week for the given month. The majority of users are member which indicates the availability of bikes will be dependent on work schedules during weekdays. The surplus and shortage count of bikes by docking station denotes the uneven direction of bikes in some sections of New York City. The shortage (and surplus) counts confirm the practice of rebalancing by Citi Bike to ensure availability of bikes. The duration of bike trips may not play as large a factor in bike availability compared to time of day, day of week, and directional flow of bikes throughout the city.

Appendix with Code

```
# Required packages
library(tidyverse)
library(ggplot2)
library(skimr)
library(lubridate)
library(fpp3)
library(assertthat)
library(igraph)
library(ggraph)
library(ggmap)
library(leaflet)
```

```
library(rgdal)
library(RColorBrewer)
library(jpeg)
library(data.table)
library(MASS)
library(RANN)
library(geosphere)
library(broom)
library(AER)
library(Metrics)
library(pscl)

# Create additional columns of pertinence
# weekday, day of the month, trip duration in minutes, start hour
citibike <- fread("data/202209-citibike-tripdata.csv", data.table=FALSE, check.names=TRUE) %>%
#citibike <- read.csv("data/202209-citibike-tripdata.csv", check.names=TRUE) %>%
  mutate(day = factor(mday(ymd_hms(started_at))),
         start.hour=factor(hour(ymd_hms(started_at))),
         weekday = lubridate::wday(ymd_hms(started_at), label=TRUE, abbr=TRUE),
         trip.duration = as.numeric(difftime(ended_at,started_at,units="mins")),
         member_casual = factor(member_casual)) %>%
  rename(ride.id=ride_id, rideable.type=rideable_type, started.at=started_at,
         ended.at=ended_at, start.station.name=start_station_name, start.station.id=start_station_id,
         end.station.name=end_station_name, end.station.id=end_station_id, start.lat=start_lat,
         start.lng=start_lng, end.lat=end_lat, end.lng=end_lng, member.casual=member_casual)

# Figure out abandoned and label them
citibike$end.station.name[citibike$end.station.name == ''] <- "Abandoned"
citibike$end.station.id[citibike$end.station.id  == ''] <- "ABAN"

# Output DF if needed
head(citibike)

# Trip by weekday by segment by time of day
citibike %>%
  group_by(day, member.casual, start.hour) %>%
  summarize(n=n(),
            weekday=weekday[1]) %>%
  group_by(weekday, member.casual, start.hour) %>%
  summarize(n.m=mean(n)) %>%
  ggplot(aes(x=start.hour, y=n.m, fill=weekday)) +
  geom_bar(stat='identity') +
  labs(x='Time of Day',
       y='Number of Trips',
       title='Average Number of Bike Trips by Time, Day, and User') +
  facet_grid(weekday~member.casual) +
  theme(axis.text.x = element_text(size=8, angle=90),
        legend.position = 'none')

citibike$started.at.ts <- as_datetime(as.character(citibike$started.at))

citibike_by_hour <- citibike %>%
  mutate(hour=lubridate::floor_date(started.at.ts, "1 hour")) %>%
```

```
  group_by(hour, member.casual) %>%
  summarize(cnt=n())
#citibike_by_hour

citibike_by_hour_ts <- citibike_by_hour %>%
  as_tsibble(index=hour, key=c(member.casual))
#citibike_by_hour_ts

autoplot(citibike_by_hour_ts, cnt) +
  labs(title = "Ride Trips by Hour - Sept. 2022",
       subtitle = "Citi Bike NYC",
       y = "Ride Trips Counts") +
  guides(fill=guide_legend(title="New Legend Title"))
citibike_by_hour_ts %>%
  gg_season(cnt, period = "week") +
  labs(y = "Ride Trips",
       title = "Seasonal plot: Weekly Trip Counts for Sept. 2022")
citibike %>%
  filter(member.casual %in% c('member', 'casual')) %>%
  group_by(weekday, start.hour, member.casual) %>%
  summarize(med.duration=median(trip.duration)) %>%
  ggplot(aes(x=start.hour, y=med.duration, group=member.casual,
             color=member.casual, linetype=member.casual, shape=member.casual)) +
  geom_point(size=2) +
  geom_line(size=0.5) +
  facet_wrap(~weekday, nrow=1) +
  labs(x='Time of Day',
       y='Median Trip Duration',
       title = "Median Trip Duration by User and Day for Sept. 2022",
       color='User Type') +
  scale_x_discrete(breaks=c(0,6,12,18))
# Create table of start to end station IDs
stations_cols <- citibike %>%
  dplyr::select(start.station.id, end.station.id)
stations_table <- as.data.frame((table(stations_cols)))

stations_table <- stations_table %>%
  filter(Freq > 0)

stations_table_order <- stations_table[order(-stations_table$Freq),]

stations_table_dif <- stations_table_order %>%
  filter(as.character(start.station.id) != as.character(end.station.id))

# Create table of start to end station IDs
stations_cols <- citibike %>%
  dplyr::select(start.station.id, end.station.id)
stations_table <- as.data.frame((table(stations_cols)))

stations_table <- stations_table %>%
  filter(Freq > 0)

stations_table_order <- stations_table[order(-stations_table$Freq),]
```

```r
stations_table_dif <- stations_table_order %>%
  filter(as.character(start.station.id) != as.character(end.station.id))

stations_table_dif1 <- stations_table_dif
stations_table_dif2 <- stations_table_dif

# Added all=TRUE to account for one-sided counts
stations_table_dif_merge <-
  merge(stations_table_dif1,
      stations_table_dif1,
      by.x=c('start.station.id','end.station.id'),
      by.y=c('end.station.id','start.station.id'), all = TRUE)

# Set the NA (one-sided trips) to count of 0
stations_table_dif_merge[is.na(stations_table_dif_merge)] <- 0

stations_table_dif_merge$surplus <- stations_table_dif_merge$Freq.y - stations_table_dif_merge$Freq.x
stations_table_dif_merge <- stations_table_dif_merge[order(-stations_table_dif_merge$surplus),]

# Remove rows with start station id equal to ABAN for abandoned, those are a result of the merge all=TR
stations_table_dif_merge <- stations_table_dif_merge %>% filter(start.station.id != 'ABAN')

# Want to identify the surplus (or not) by station for the month
station_surplus_count <- stations_table_dif_merge %>%
  group_by(start.station.id) %>%
  summarize(surplus.sum=sum(surplus))


station_surplus_count <- station_surplus_count[order(-station_surplus_count$surplus.sum),]
colnames(station_surplus_count)[1] <- "station.id"

# Extract just the end station Id, lat, log
# because this had the higher count from the initial dataset, going with end_station_id
end_station_info <- citibike %>%
  dplyr::select(end.station.id, end.lng, end.lat)

end_station_info <- end_station_info[!duplicated(end_station_info$end.station.id),]

# Now I want the coordinates of all those station_ids
station_surplus_count_coords <- merge(x = station_surplus_count, y = end_station_info, by.x = 'station.

colnames(station_surplus_count_coords)[3] <- "lng"
colnames(station_surplus_count_coords)[4] <- "lat"

station_surplus_count_coords <- station_surplus_count_coords %>%
  add_row(station.id = "ABAN", surplus.sum=1363, lng=-73.99, lat=40.67)

# Using basemaps for NYC
m <- leaflet(data=station_surplus_count_coords) %>%
#  setView(zoom=12) %>%
  addTiles() %>%
  addCircleMarkers(
    ~lng, ~lat,
```

```
    popup=~as.character(station.id),
    label=~as.character(station.id),
    radius=.5,
    color = ~ifelse(surplus.sum >= 1, 'blue',
                    ifelse(surplus.sum == 0, 'green', 'red'))
  )

# Display map
#m
img <- readJPEG("stations_surplus_sum.jpg")
plot(1:10,ty="n", axes = 0, xlab='', ylab='', main='Overall Monthly Surplus/Shortage by Docking Station
rasterImage(img,-1,-1,12,12)
stations_with_elevation <- read.csv('stations_with_elevation.csv', row.names = 1, header= TRUE)

stations_with_boro_hood <- read.csv('stations_with_boro_and_hood.csv', row.names = 1, header= TRUE)

# Combine the elevation, borough, neighborhood, and September surplus
# First let's trim the DF for elevation
stations_with_elevation_trim <- stations_with_elevation %>%
  dplyr::select(short_name, station_id, elevation, elev_units)

stations_with_boro_hood_trim <- stations_with_boro_hood %>%
  dplyr::select(short_name, name, station_id, capacity, ntaname, boro_name, lon, lat)

stations_attrs_trim <-
  merge(stations_with_elevation_trim,
        stations_with_boro_hood_trim,
        by.x=c('short_name'),
        by.y=c('short_name'), all = TRUE)

stations_attrs_trim <- stations_attrs_trim %>%
  dplyr::select(-station_id.y)

colnames(stations_attrs_trim)[colnames(stations_attrs_trim) == 'station_id.x'] <- 'station_id'

stations_attrs_trim[c("boro_name")][is.na(stations_attrs_trim[c("boro_name")])] <- "New Jersey"

stations_attrs_trim <-
  stations_attrs_trim %>%
  mutate(ntaname = ifelse(startsWith(short_name, "JC"), "Jersey City", ntaname))

stations_attrs_trim <-
  stations_attrs_trim %>%
  mutate(ntaname = ifelse(startsWith(short_name, "HB"), "Hoboken", ntaname))


stations_attrs_trim_sur <-
  merge(stations_attrs_trim,
        station_surplus_count,
        by.x=c('short_name'),
        by.y=c('station.id'), all.x = TRUE)

stations_attrs_trim_sur <- stations_attrs_trim_sur %>%
```

```
  filter(!is.na(surplus.sum))

stations_attrs_trim_sur %>%
  filter(surplus.sum >= -100 & surplus.sum <= 100) %>%
  ggplot(aes(x=surplus.sum, color=boro_name), ) +
  theme(legend.position="bottom") +
  geom_histogram(bins=201) +
  facet_wrap(~boro_name) +
  labs(x = 'Surplus Count',
       y = 'Number of Stations',
       title = "Station Surplus by Borough Histogram for Sept. 2022",
       fill = "Borough")
# This removed the duplicate rows by transposing the start and end station ids
stations_table_dif_merge_temp <- stations_table_dif_merge %>% select(start.station.id, end.station.id)


stations_for_graph <- stations_table_dif_merge_temp[!duplicated(lapply(as.data.frame(t(stations_table_di

g_stations <- graph_from_data_frame(stations_for_graph, directed=FALSE, vertices=station_surplus_count_

edges_for_plot <- stations_for_graph %>%
  inner_join(station_surplus_count_coords %>% select(station.id, lng, lat), by=c('start.station.id' = 's
  rename(x=lng, y=lat) %>%
  inner_join(station_surplus_count_coords %>% select(station.id, lng, lat), by=c('end.station.id' = 'sta
  rename(xend=lng, yend=lat)

#assert_that(nrow(edges_for_plot) == nrow(stations_for_graph))

citibike_bike_avail <- fread("bike_avail_by_station_and_time.csv", data.table=FALSE, check.names=FALSE)
citibike_bike_avail <- citibike_bike_avail %>%
  dplyr::select(-V1)
citibike_bike_avail_long <- citibike_bike_avail %>%
  pivot_longer(!timestamp, names_to = "station.id", values_to = "bikes.avail")

#head(citibike_bike_avail_long)
citibike_bike_avail_long_trim <- citibike_bike_avail_long %>%
  filter(as.integer(station.id) == 3582)
p<-ggplot(citibike_bike_avail_long_trim, aes(x=timestamp, y=bikes.avail, group=station.id)) +
  geom_line(aes(color=station.id), show.legend = FALSE) +
  geom_point(aes(color=station.id), show.legend = FALSE) +
  labs(x = 'Date/Time (15-minute Increments)',
       y = 'Bikes Available',
       title = "Docking Station 3582 Bike Availability")
p
stations_attrs_trim_bk61 <- stations_attrs_trim %>%
  filter(ntaname == "Crown Heights North")

#(stations_attrs_trim_bk61)
#11 stations in this file
# Let's try the clustering
# https://www.r-bloggers.com/2019/06/hierarchical-clustering-for-location-based-strategy-using-r-for-e-

# Distance matrix for docking stations
```

```r
# Result is in meters
dist_mat <- distm(stations_attrs_trim_bk61[9:10], stations_attrs_trim_bk61[9:10], fun=distHaversine)
dist_mat <- as.data.frame(dist_mat)
dist_mat[is.na(dist_mat)] <- 0
dMat <- as.dist(dist_mat)
#dMat[1:10]

# Now for the clustering
hier_clust <- hclust(dMat, method = "complete")
# 400 meters is about a quarter of a mile (0.248548 miles)
stations_attrs_trim_bk61$cluster <- cutree(hier_clust, h=400)

# Display filtered stations with cluster column
#stations_attrs_trim_bk61
stations_with_bike_avail <- read.csv('bike_avail_by_station_and_time.csv', row.names = 1, header= TRUE,

stations_with_bike_avail_long <- stations_with_bike_avail %>%
  pivot_longer(!timestamp, names_to = "station.id", values_to = "bikes.avail.count")

station_to_cluster <- stations_attrs_trim_bk61 %>%
  dplyr::select(station_id, cluster)

#(station_to_cluster)

# Merge long df with counts with station.id and cluster to label clusters properly
stations_bike_avail_by_time <-
  merge(stations_with_bike_avail_long,
        station_to_cluster,
        by.x=c('station.id'),
        by.y=c('station_id'), all.x = TRUE)

stations_bike_avail_by_time_no_na <- stations_bike_avail_by_time %>%
  filter(!is.na(cluster))

# group by cluster ID for each timestamp
stations_bike_avail_by_time_no_na_group <- aggregate(bikes.avail.count ~ timestamp + cluster, data = sta

stations_bike_avail_by_time_no_na_group <- stations_bike_avail_by_time_no_na_group[order(stations_bike_a

# mutate to convert timestamp into day of the week, etc.
stations_bike_avail_by_time_no_na_group <- stations_bike_avail_by_time_no_na_group %>%
  mutate(time=as.ITime(ymd_hms(timestamp)),
         weekday = lubridate::wday(ymd_hms(timestamp),label=TRUE,abbr=TRUE))

stations_bike_avail_by_time_no_na_group$cluster <- as.factor(stations_bike_avail_by_time_no_na_group$clu
stations_bike_avail_by_time_no_na_group$time <- as.factor(stations_bike_avail_by_time_no_na_group$time)
stations_bike_avail_by_time_no_na_group$weekday <- as.factor(stations_bike_avail_by_time_no_na_group$wee

stations_bike_avail_by_time_no_na_group$time <- factor(stations_bike_avail_by_time_no_na_group$time,
                       levels = c("00:00:00", "00:15:00", "00:30:00", "00:45:00",
                                  "01:00:00", "01:15:00", "01:30:00", "01:45:00",
                                  "02:00:00", "02:15:00", "02:30:00", "02:45:00",
                                  "03:00:00", "03:15:00", "03:30:00", "03:45:00",
```

```
                                  "04:00:00", "04:15:00", "04:30:00", "04:45:00",
                                  "05:00:00", "05:15:00", "05:30:00", "05:45:00",
                                  "06:00:00", "06:15:00", "06:30:00", "06:45:00",
                                  "07:00:00", "07:15:00", "07:30:00", "07:45:00",
                                  "08:00:00", "08:15:00", "08:30:00", "08:45:00",
                                  "09:00:00", "09:15:00", "09:30:00", "09:45:00",
                                  "10:00:00", "10:15:00", "10:30:00", "10:45:00",
                                  "11:00:00", "11:15:00", "11:30:00", "11:45:00",
                                  "12:00:00", "12:15:00", "12:30:00", "12:45:00",
                                  "13:00:00", "13:15:00", "13:30:00", "13:45:00",
                                  "14:00:00", "14:15:00", "14:30:00", "14:45:00",
                                  "15:00:00", "15:15:00", "15:30:00", "15:45:00",
                                  "16:00:00", "16:15:00", "16:30:00", "16:45:00",
                                  "17:00:00", "17:15:00", "17:30:00", "17:45:00",
                                  "18:00:00", "18:15:00", "18:30:00", "18:45:00",
                                  "19:00:00", "19:15:00", "19:30:00", "19:45:00",
                                  "20:00:00", "20:15:00", "20:30:00", "20:45:00",
                                  "21:00:00", "21:15:00", "21:30:00", "21:45:00",
                                  "22:00:00", "22:15:00", "22:30:00", "22:45:00",
                                  "23:00:00", "23:15:00", "23:30:00", "23:45:00"))

tab <- table(stations_bike_avail_by_time_no_na_group$bikes.avail.count)
#tab
# Below shows the range of values for `bikes.avail.count` with 35611 at zero and max of 438 with one oc

tab_df <- as.data.frame(tab)

# Distrubtion of count results
p <- ggplot(data=tab_df, aes(x=Var1, y=Freq)) +
  geom_bar(stat="identity")  +
  labs(x='Count of Available Bikes',
       y='Frequency',
       title='Frequency of Available Bike Counts for Docking Stations (Oct. 31 - Nov. 13)')
p

stations_bike_avail_by_time_no_timestamp <- subset(stations_bike_avail_by_time_no_na_group, select=-c(t
#stations_bike_avail_by_time_no_timestamp
# Start with data partition here.
set.seed(8675309)
index <- sample(2, nrow(stations_bike_avail_by_time_no_timestamp), replace = TRUE, p=c(0.8, 0.2))
train <- stations_bike_avail_by_time_no_timestamp[index==1,]
test <- stations_bike_avail_by_time_no_timestamp[index==2,]
# Sample Variance is 313
var(stations_bike_avail_by_time_no_timestamp$bikes.avail.count)
n <- length(stations_bike_avail_by_time_no_timestamp)
# Population variance
var(stations_bike_avail_by_time_no_timestamp$bikes.avail.count)*(n-1)/n
# Mean is 13
mean(stations_bike_avail_by_time_no_timestamp$bikes.avail.count)

# Overdispersion exists
mod1 <- glm(bikes.avail.count ~ cluster + time + weekday, data = train, family ="poisson")
summary(mod1)
```

```r
#library(broom)
glance(mod1)
#library(AER)
dispersiontest(mod1)
# This indicates dispersion
pred <- predict.glm(mod1, newdata=test[test$bikes.avail.count != 0,],type = "response")
# ggplot
plot(test$bikes.avail.count[test$bikes.avail.count!=0],type = "b",col="red",
     main="Bike Availability Predictions",
     sub="Actual counts in Red; Predictions in Blue",
     xlab="Index", ylab="Count of Available Bikes")
lines(round(pred),col="blue")
pred <- predict.glm(mod1, newdata = test, type = "response")

rmsemodelp <- ModelMetrics::rmse(test$bikes.avail.count,round(pred))
maemodelp <- mae(test$bikes.avail.count,round(pred))

rmsemodelp
maemodelp
mod2 <- glm(bikes.avail.count ~ cluster + time + weekday, data = train, family ="quasipoisson")
#summary(mod2)
predq <- predict.glm(mod2,newdata=test, type = "response")

rmsemodelqp <- ModelMetrics::rmse(test$bikes.avail.count,round(predq))
maemodelqp <- mae(test$bikes.avail.count,round(predq))

rmsemodelqp
maemodelqp
mod3 <- glm.nb(bikes.avail.count ~ cluster + time + weekday, data=train)
#summary(mod3)
prednb <- predict.glm(mod3,newdata=test,type = "response")

rmsemodelnb <- ModelMetrics::rmse(test$bikes.avail.count,round(prednb))
maemodelnb <- mae(test$bikes.avail.count,round(prednb))

rmsemodelnb
maemodelnb
#library(pscl)
modelhp <- hurdle(bikes.avail.count ~ cluster + time + weekday, data=train, dist = "poisson")
#summary(modelhp)
predhp <- predict(modelhp, newdata=test, type = "response")

rmsemodelhp <- ModelMetrics::rmse(test$bikes.avail.count,round(predhp))
maemodelhp <- mae(test$bikes.avail.count,round(predhp))

rmsemodelhp
maemodelhp
modelhnb <- hurdle(bikes.avail.count ~ cluster + time + weekday, data=train, dist = "negbin")
#summary(modelhnb)
predhnb <- predict(modelhnb, newdata=test,type = "response")

rmsemodelhnb <- ModelMetrics::rmse(test$bikes.avail.count,round(predhnb))
maemodelhnb <- mae(test$bikes.avail.count,round(predhnb))
```

```
rmsemodelhnb
maemodelhnb
modelzp <- zeroinfl(bikes.avail.count ~ cluster + time + weekday, data=train, dist = "poisson")
#summary(modelzp)
predzp <- predict(modelzp,newdata=test,type = "response")

rmsemodelzp <- ModelMetrics::rmse(test$bikes.avail.count,round(predzp))
maemodelzp <- mae(test$bikes.avail.count,round(predzp))

rmsemodelzp
maemodelzp
modelznb <- zeroinfl(bikes.avail.count ~ cluster + time + weekday, data=train,dist = "negbin")
#summary(modelznb)
predznb <- predict(modelznb,newdata=test,type = "response")

rmsemodelznb <- ModelMetrics::rmse(test$bikes.avail.count,round(predznb))
maemodelznb <- mae(test$bikes.avail.count,round(predznb))

rmsemodelznb
maemodelznb
rmse <- c(rmsemodelp,rmsemodelqp,
          rmsemodelnb,
          rmsemodelhp,rmsemodelhnb,
          rmsemodelzp,rmsemodelznb)
mae <- c(maemodelp,maemodelqp,
         maemodelnb,
         maemodelhp,maemodelhnb,
         maemodelzp,maemodelznb)
models <- c("Poisson","Quasi-Poisson",
            "Negative Binomial",
            "Hurdle Poisson","Hurdle NB",
            "Zero-Inflated Poisson","Zero-Inflated NB")

data.frame(models,rmse,mae) %>%
  arrange(rmse)
# Function to predict number of available bikes given a longitude and latitude, day of the week, and ti
# Function returns floor value of the result to ensure a whole number as the model predictions are floa
predict_num_bikes_avail <- function(longitude, latitude, time, day) {
  #longitude: -73.960859
  #latitude: 40.67355
  #time: "07:30:00"
  #day: "Mon"

  num_bikes_avail <- 0

  # Convert lon/lat to cluster
  location <- data.frame(matrix(nrow = 1,data = c(longitude, latitude)))
  closest_station <- nn2(stations_attrs_trim_bk61[, 9:10], query=location, k=1)
  inp_cluster <- stations_attrs_trim_bk61[closest_station$nn.idx,]$cluster

  # Create input for prediction
  bike_avail_query <- data.frame(matrix(nrow = 1,data = c(inp_cluster, 0, time, day)))
  colnames(bike_avail_query) <- c("cluster", "bikes.avail.count", "time", "weekday")
```

```
  # Predict using Model: mod1
  num_bikes_avail <- predict.glm(mod1, newdata = bike_avail_query, type = "response")
  # Take floor of prediction to ensure whole number
  num_bikes_avail <- as.integer(floor(num_bikes_avail))

  return(num_bikes_avail)
}
# Test case for the function
predict_num_bikes_avail(-73.960859, 40.67355, "14:30:00", "Sat")
# https://yihui.org/en/2018/09/code-appendix/
```

LaTex help https://tex.stackexchange.com/questions/10684/vertical-space-in-lists