CitiBike Data Analysis

Data 621: Final Project - Group 4

Michael Yampol Ann Liu-Ferrara Sachid Deshmukh Vishal Arora

12/20/2019

Contents

0. Abstract

Bicycling is an activity which yields many benefits: Riders improve their health through exercise, while traffic congestion is reduced if riders move out of cars, with a corresponding reduction in pollution from carbon emissions. In recent years, Bike Sharing has become popular in a growing list of cities around the world. The NYC "CitiBike" bicycle sharing scheme went live (in midtown and downtown Manhattan) in 2013, and has been expanding ever since, both as measured by daily ridership as well as the expanding geographic footprint incorporating a growing number of "docking stations" as the system welcomes riders in Brooklyn, Queens, and northern parts of Manhattan which were not previously served.

One problem that many bikeshare systems face is money. An increase in the number of riders who want to use the system necessitates that more bikes be purchased and put into service in order to accomodate them. Heavy ridership induces wear on the bikes, requiring for more frequent repairs. However, an increase in the number of trips does not necessarily translate to an increase in revenue because riders who are clever can avoid paying surcharges by keeping the length of each trip below a specified limit (either 30 or 45 minutes, depending on user category.)

We seek to examine CitiBike ridership data, joined with daily NYC weather data, to study the impact of weather on shared bike usage and generate a predictive model which can estimate the number of trips that would be taken on each day.
The goal is to estimate future demand which would enable the system operator to make expansion plans.

Our finding is that ridership exhibits strong seasonality, with correlation to weather-related variables such as daily temperature and precipitation. Additionally, ridership is segmented by by user_type (annual subscribers use the system much more heavilty than casual users), gender (there are many more male users than female) and age (a large number of users are clustered in their late 30s).

Keywords

Bikeshare, Weather, Cycling, CitiBike, New York City

1. Introduction

Since 2013 a shared bicycle system known as CitiBike has been available in New York City. The benefits to having such a system include reducing New Yorkers' dependence on automobiles and encouraging public health through the exercise attained by cycling. Additionally, users who would otherwise spend money on public transit may find bicycling more economical – so long as they are aware of CitiBike's pricing constraints.

There are currently about 12,000 shared bikes which users can rent from about 750 docking stations located in Manhattan and in western portions of Brooklyn and Queens. A rider can pick up a bike at one station and return it at a different station. The system has been expanding each year, with increases in the number of bicycles available and expansion of the geographic footprint of docking stations. For planning purposes, the system operator needs to project future ridership in order to make good investments.

The available usage data provides a wealth of information which can be mined to seek trends in usage. With such intelligence, the company would be better positioned to determine what actions might optimize its revenue stream.

- Because of weather, ridership is expected to be lower during the winter months, and on foul-weather days during the rest of the year, than on a warm and sunny summer day. Using the weather data we can seek to model the relationship between bicycle ridership and fair/foul or hot/cold weather.
- What are the differences in rental patterns between annual members (presumably, local residents) vs. casual users (presumably, tourists?)

- Is there any significant relationship between the age and/or gender of the bicycle renter vs. the rental patterns?

The rest of the paper proceeds as follows:

- In **section 2** we review literature on bike share systems, some of which examine the relationship of weather on ridership.
- In **section 3** we examine the methodology from the standpoint of data sources, collection, cleaning, exploratory data analysis (EDA), limitations, aggregation, and statistical modeling choices.
- In **section 4** we discuss results from our modeling.
- In **section 5** we present discussion of the challenges faced in this project and analysis that we commenced but chose to discard.
- In **section 6** we present a conclusion and summary of the results,and discuss proposed future research and enhancements.
- In the **appendix** we provide graphs, tables, and a listing of the R statistical programming code used to perform the data manipulation, modeling, and visualization.

2. Literature review

Westland et al. examined consumer behavior in bike sharing in Beijing using a deep-learning model incorporating weather and air quality, time-series of demand, and geographical location; later adding customer segmentation. (Westland, Mou, and Yin 2019)

Jia et al. performed a retrospective study of dockless bike sharing in Shanghai to determine whether introduction of such program increased cycling. Their methodology was to survey people in various neighborhoods where the areas were selected by sampling, and the individuals were selected by interviewing individuals on the street. (Jia et al. 2019)

Jia and Fu further examined whether dockless bicycle-sharing programs promote changes in travel mode in commuting and non-commuting trips, as well as the association between change in travel mode and potential correlates, as part of the same Shanghai study. (Jia and Fu 2019)

Dell'Amico et al. modeled bike sharing rebalancing programs initially in Reggio Emilia, Italy using branch-and-cut algorithms. (Dell'Amico et al. 2014)

In a more recent paper, **Dell'Amico** et al. examined the bike-sharing rebalancing problem with Stochastic Demands, aimed at determining minimum cost routes for a fleet of homogeneous vehicles in order to redistribute bikes among stations. (Dell'Amico et al. 2018)

Zhou analyzed massive bike-sharing data in Chicago, constructing a bike flow similarity graph and using a fast-greedy algorithm to detect spatial communities of biking flows. He examined the questions 1. How do bike flow patterns vary as a result of time, weekday or weekend, and user groups? 2. Given the flow patterns, what was the spatiotemporal distribution of the over-demand for bikes and docks in 2013 and 2014? (Zhou 2015)

Hosford et al. surveyed participants in Vancouver, Canada and determined that public bicycle share programs are not used equally by all segments of the population. In many cities, program members tend to be male, Caucasian, employed, and have higher educations and incomes compared to the general population. Further, their study determined that the majority of bicycle share trips replace trips previously made by walking or public transit, indicating that bicycle share appeals to people who already use active and sustainable modes of transportation (Hosford, Lear, et al. 2018)

In another paper, **Hosford** et al. determined that that the implementation of the public bicycle share program in Vancouver was associated with greater increases in bicycling for those living and working inside the bicycle share service area relative to those outside the service area in the early phase of implementation, but this effect did not sustain over time. (Hosford, Fuller, et al. 2018)

Schmidt observed that the number of bike-sharing programs worldwide grew from 5 in 2005 to 1,571 in 2018. He further noted that disparities in bike-sharing usage are evident around the country, with users skewing towards younger white men. (Schmidt 2018)

Wang et al. examined the rebalancing problem and determined that the fluctuation of the available bikes and docks is not only caused by the user but also by the operators' own (inefficient) rebalancing activities; they propose a data-driven model to generate an optimal rebalancing model while minimizing the cost of moving the bikes. (Wang et al. 2018)

Vogel and Mattfeld observe that Short rental times and one-way use lead to imbalances in the spatial distribution of bikes at stations over time, and present a case study demonstrating that Data Mining applied to operational data offers insight into typical usage patterns of bike-sharing systems and is used to forecast bike demand with the aim of supporting and improving strategic and operational planning. They analyze both operational data from Vienna's shared bike rental system as well as local weather data over the period. (Vogel and Mattfeld 2011)

Fuller et al. examined the impact of a public transit strike (November 2016 in Philadelphia) on usage of the bike share service in that city. (Fuller et al. 2019)

In an earlier study, **Fuller** et al. examined bikeshare in Montreal by collecting samples prior to the launch of the program, and following each of the first two seasons. [Unlike other cities such as New York, the Montreal bike share system does not operate year-round. Rather, because of the especially harsh winters, their bikeshare system is dismantled each fall and reinstalled each spring.] Fuller's methodology incorporated a 5-step logistic regression in which the weather variables entered at step 4; this rendered nonsignificant the differences between the three survey periods. (Fuller et al. 2013)

Faghih-Imani and Eluru study the decision process involved in identifying destination locations after picking up the bicycle at a BSS station. In the traditional destination/location choice approaches, the model frameworks implicitly assume that the influence of exogenous factors on the destination preferences is constant across the entire population. They propose a *finite mixture multinomial logit* (FMMNL) model that accommodates such heterogeneity by probabilistically assigning trips to different segments and estimating segment-specific destination choice models for each segment. Unlike the traditional destination-choice-based *multinomial logit* (MNL) model or *mixed multinomial logit* (MMNL), in an FMMNL model, we can consider the effect of fixed attributes across destinations such as users' or origins' attributes in the decision process. (Faghih-Imani and Eluru 2018)

An et al. examine weather and cycling in New York City and find that weather impacts cycling rates more than topography, infrastructure, land use mix, calendar events, and peaks. They do so by exploring a series of interaction effects, which each capture the extent to which two characteristics occurring simultaneously exert a combinatorial effect on cycling ridership – e.g, how is cycling impacted when it is both wet and a weekend day or humid day in the hilliest parts of the cycling network? (An et al. 2019)

Heaney et al. examine the relation between ambient temperature and bikeshare usage and to project how climate change-induced increasing ambient temperatures may influence active transportation in New York City. (Heaney et al. 2019)

In the 1990s, **Nankervis** examined the effect of weather and climate on university student bicycle commuting patterns in Melbourne, Australia by examining counts of parked bicycles at local universities and correlating with the weather for each day, finding that the deterrent effect of bad weather on commuting was less than commonly believed (though still significiant.) (Nankervis 1999)

3. Methodology

Data sources and uploading

We obtained data from two sources:

1. CitiBike trip dataset

CitiBike makes a vast amount of data available regarding system usage as well as sales of memberships and short-term passes.

For each month since the system's inception, there is a file containing details of (almost) every trip. (Certain "trips" are omitted from the dataset. For example, if a user checks out a bike from a dock but then returns it within one minute, the system drops such a "trip" from the listing, as such "trips" are not interesting.)

There are currently 77 monthly data files for the New York City bikeshare system, spanning July 2013 through November 2019. Each file contains a line for every trip. The number of trips per month varies from as few as 200,000 during winter months in the system's early days to more than 2 million trips this past summer. The total number of entries was more than 90 million, resulting in 17GB of data. Because of the computational limitations which this presented, we created samples of 1/1000 and 1/100 of the data. The samples were created deterministically, by subsetting the files on each 1000th (or, 100th) row.

2. Central Park daily weather data

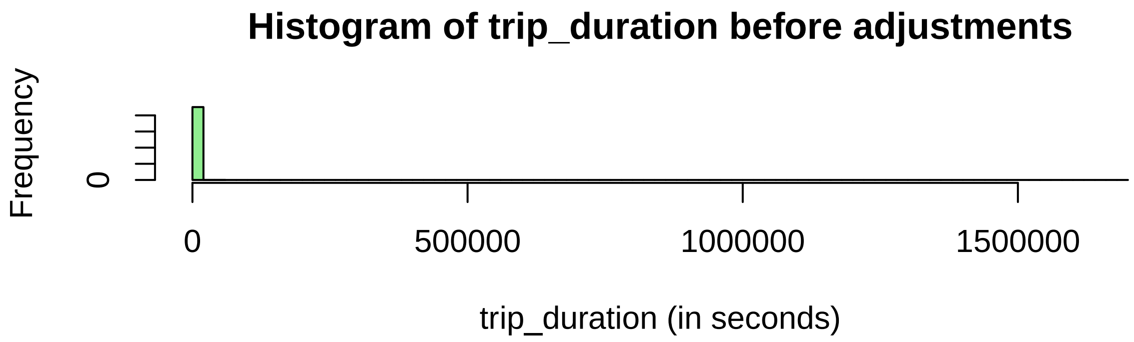
Also we obtained historical weather information for 2013-2019 from the NCDC (National Climatic Data Center) by submitting an online request to https://www.ncdc.noaa.gov/cdo-web/search . Although the weather may vary slightly within New York City, we opted to use just the data associated with the Central Park observations as proxy for the entire city's weather.

We believe that the above data provides a reasonable representation of the target population (all CitiBike rides) and the citywide weather.

Description of Data

In this section, we examine selected individual variables from the CitiBike and Weather datasets. These items require transformation and/or cleaning as there are missing values or outliers which impede analysis otherwise. The detailed data fields tables for weather and CitiBike can be found in the Appendix.

Examine variable trip__duration: The trip_duration is specified in seconds, but there are some outliers which may be incorrect, as the value for Max is quite high: 1688083 seconds, or 19.5379977 days. We can assume that this data is bad, as nobody would willingly rent a bicycle for this period of time, given the fees that would be charged. Here is a histogram of the original data distribution:



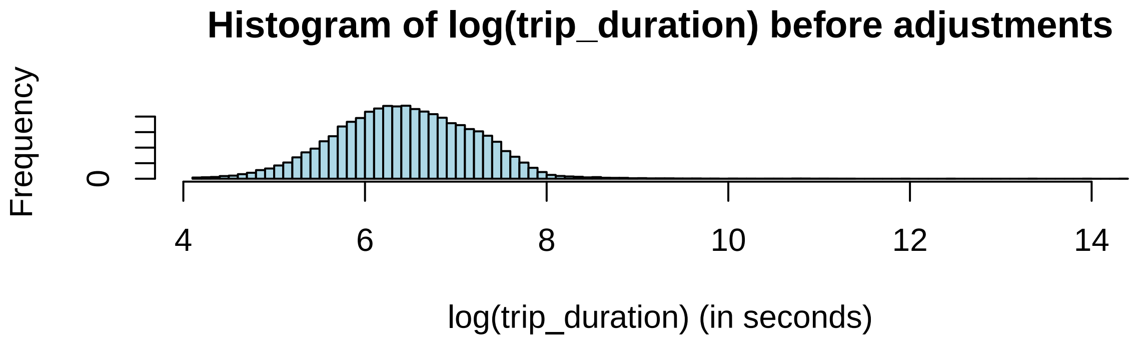


Table 1: Summary of Trip durations before truncation

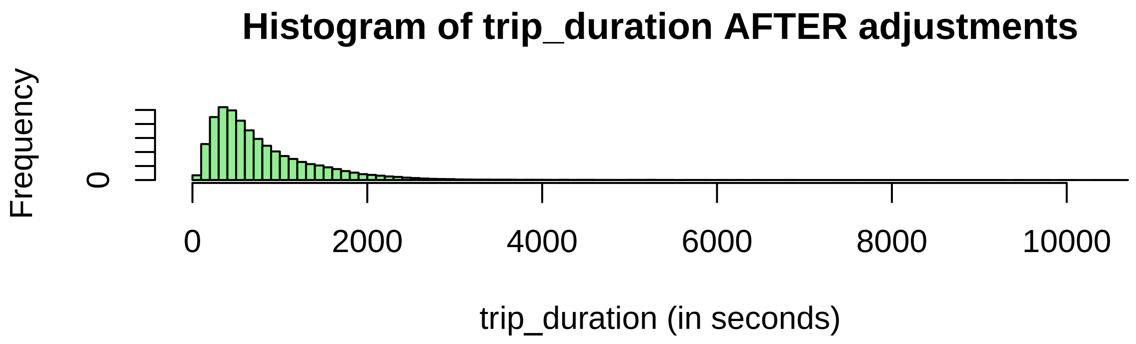
	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
supplied__secs	61.0000000	375.0000000	621.0000000	911.5364944	1064.0000000	1688083.0000
calc__secs	60.0000000	375.0000000	621.0000000	912.0153679	1065.0000000	1688083.0000
calc__mins	1.0000000	6.2500000	10.3500000	15.2002561	17.7500000	28134.7167
calc__hours	0.0166667	0.1041667	0.1725000	0.2533376	0.2958333	468.9119
calc__days	0.0006944	0.0043403	0.0071875	0.0105557	0.0123264	19.5380

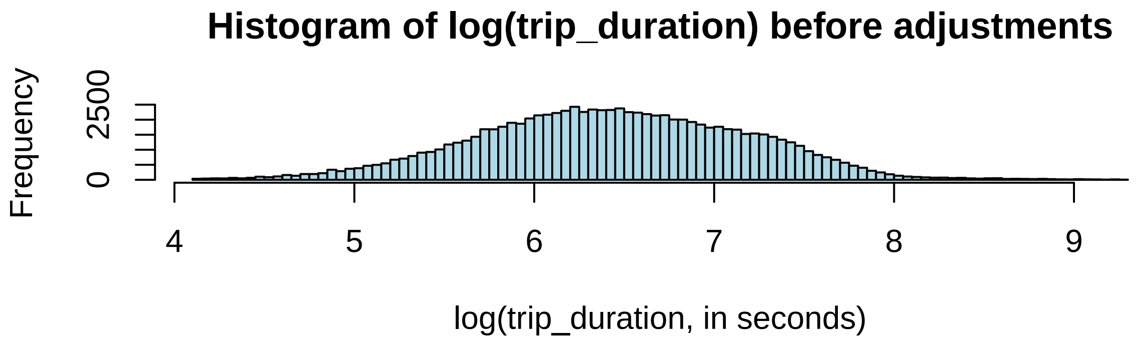
The above indicates that the duration of the trips (in seconds) includes values in the millions – which likely reflects a trip which failed to be properly closed out.

Delete cases with unreasonable trip_duration values Let's assume that nobody would rent a bicycle for more than a specified timelimit (say, 3 hours), and drop any records which exceed this:

```
## [1] "Removed 157 trips (0.174%) of longer than 3 hours."
```

```
## [1] "Remaining number of trips: 90213"
```

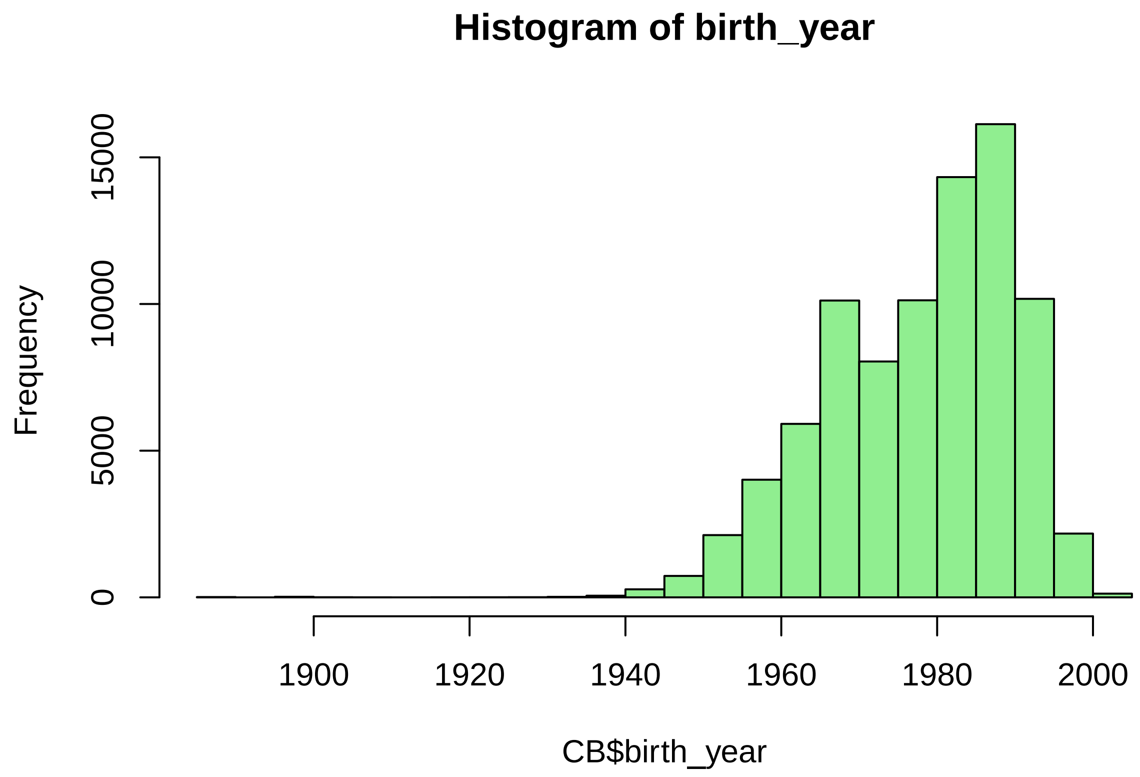


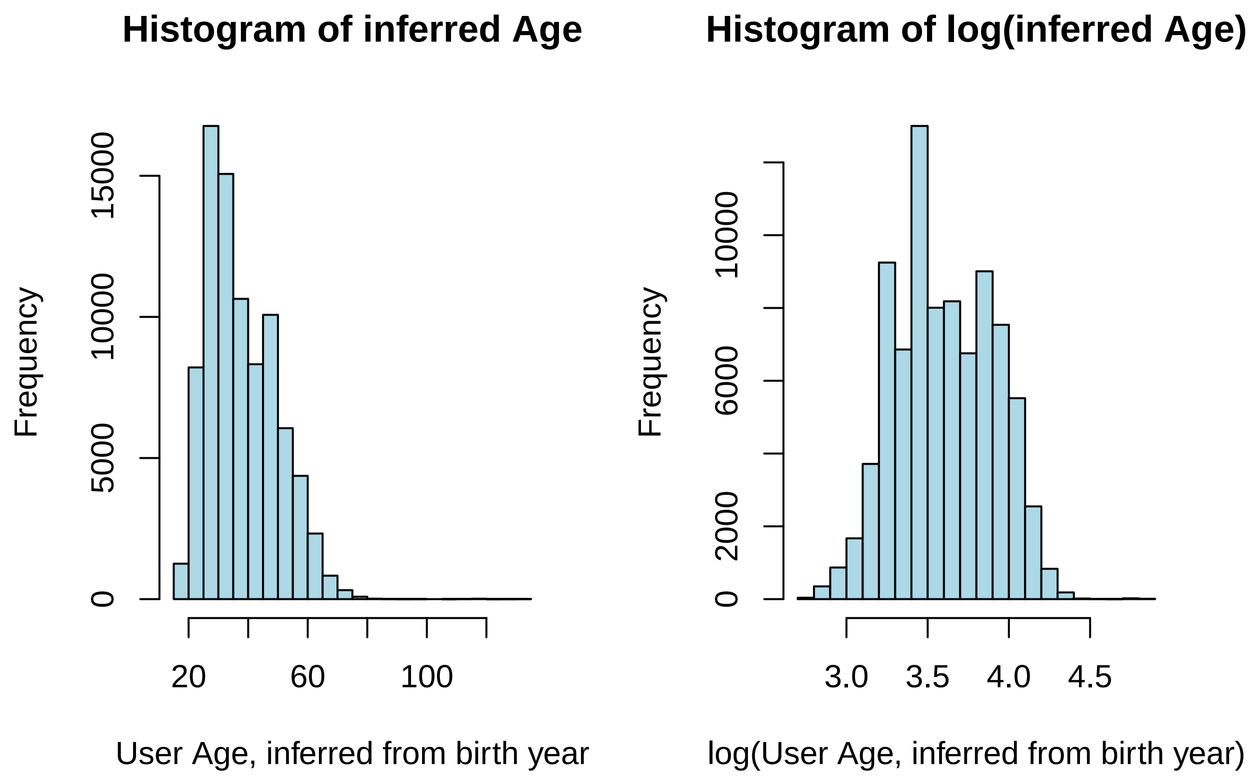


Examine birth__year Other inconsistencies concern the collection of birth__year, from which we can infer the age of the participant. There are some months in which this value is omitted, while there are other months in which all values are populated. However, there are a few records which suggest that the rider is a centenarian – it seems highly implausible that someone born in the 1880s is cycling around Central Park – but the data does have such anomalies. Thus, a substantial amount of time was needed for detecting and cleaning such inconsistencies.

The birth year for some users is as old as 1885, which is not possible:

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    1885    1969    1981    1978    1988    2003    5828
```




Histogram of inferred Age
Frequency
0
5000
10000
15000
20
60
100
User Age, inferred from birth year
Histogram of log(inferred Age)
Frequency
0
2000
6000
10000
3.0
3.5
4.0
4.5
log(User Age, inferred from birth year)

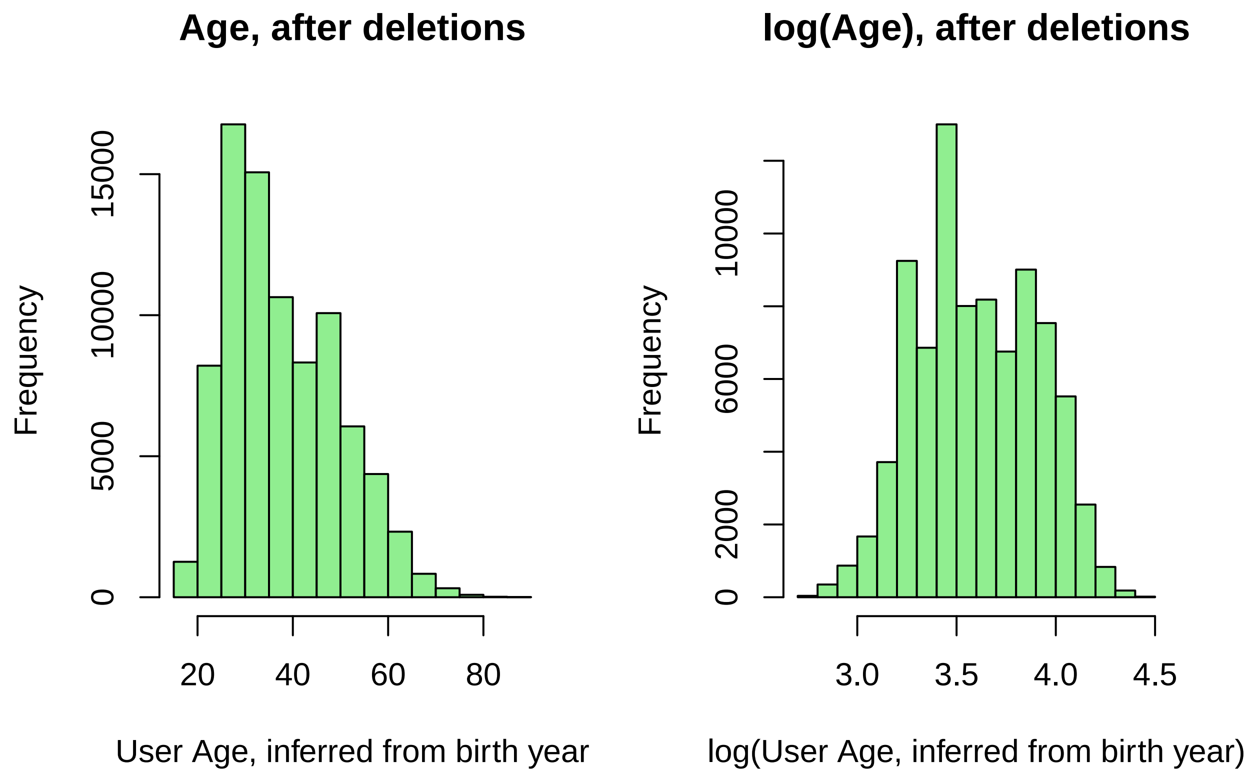
Remove trips associated with very old users (age>90)

(Also: remove trips associated with missing birth_year)

```
## [1] "Removed 40 trips (0.044%) of users older than 90 years."
```

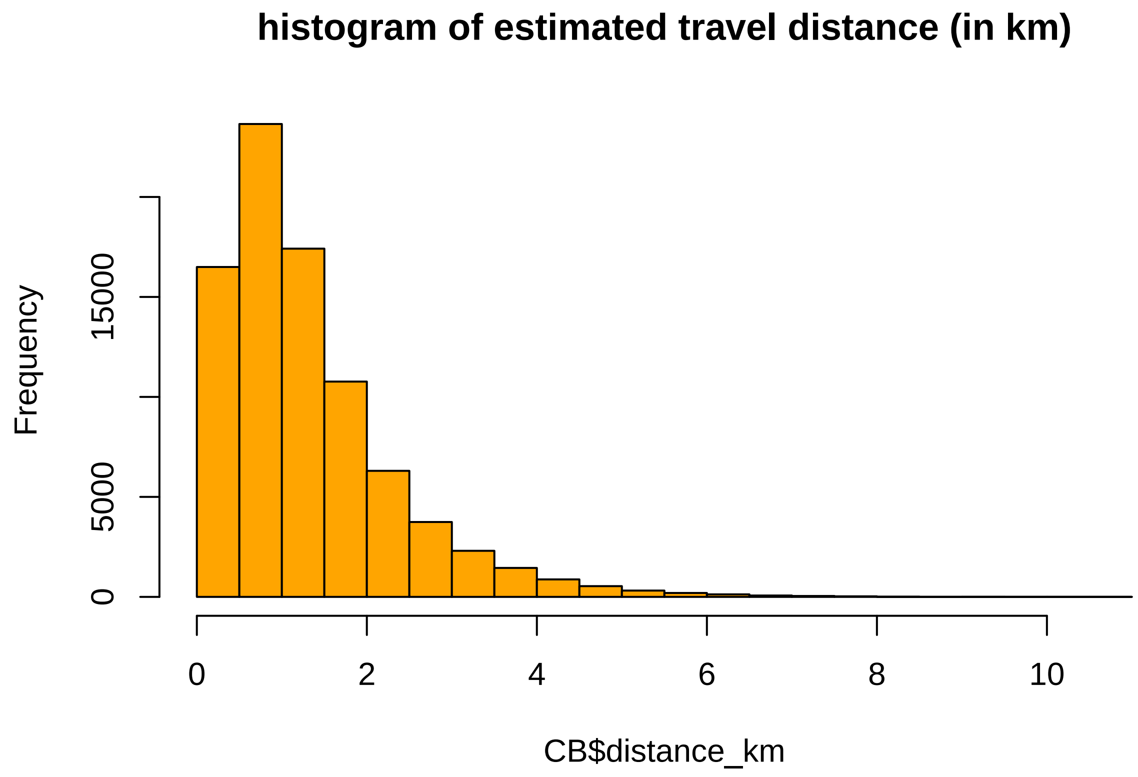
```
## [1] "Removed 5828 trips (6.449%) of users where age is unknown (birth_year unspecified)."
```

```
## [1] "Remaining number of trips: 84345"
```



Compute distance between start and end stations This is straight-line distance between (longitude,latitude) points – it doesn't incorporate an actual bicycle route.
There are services (e.g., from Google) which can compute and measure a recommended bicycle route between points, but use of such services requires a subscription and incurs a cost.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.5894  1.0492  1.3111  1.7316 10.6603
```



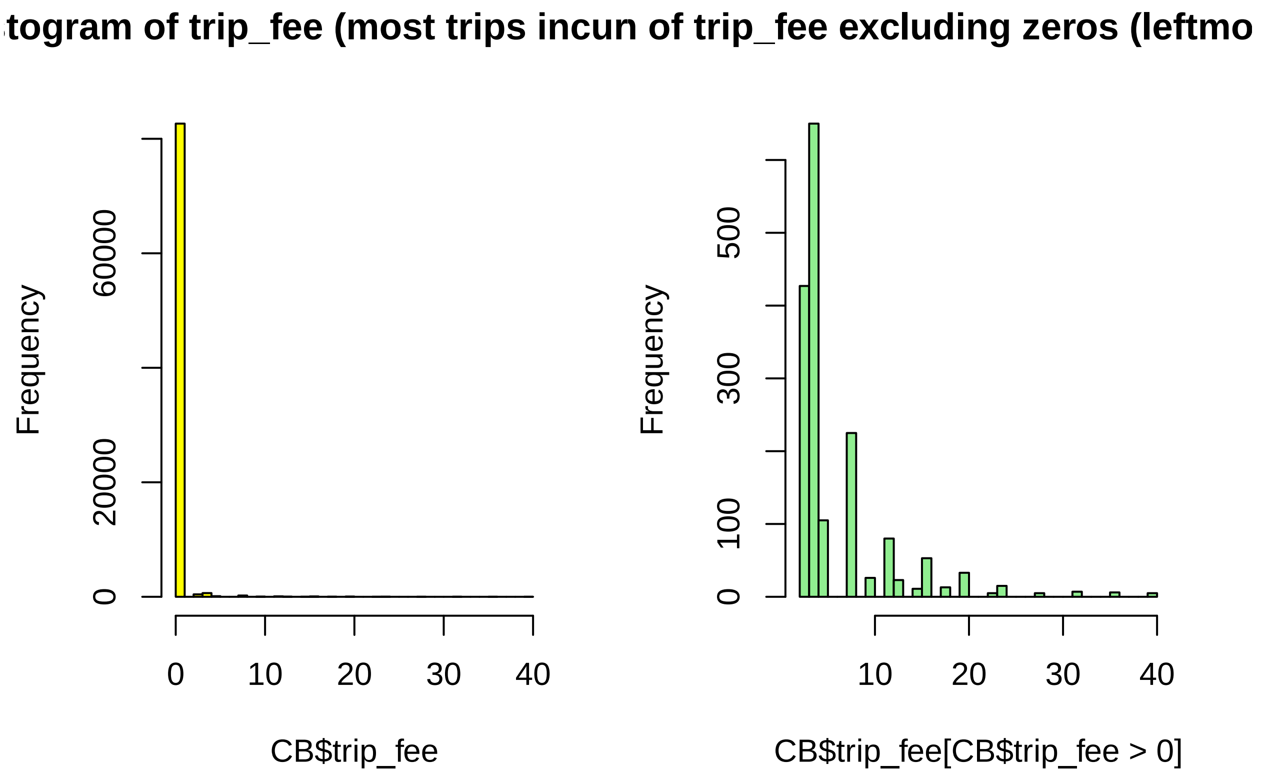
In this subset of the data, the maximum distance between stations is 10.6602792 km. In the data there are some stations for which the latitude and longitude are zero, which suggests that the distance between such a station and an actual station is many thousands of miles. If such items exist, we will delete them:

Delete unusually long distances

```
## [1] "No unusually long distances were found in this subset of the data."
```

Compute usage fee There is a time-based usage fee for rides longer than an initial period:

- For **user_type=Subscriber**, the fee is **$2.50 per 15 minutes** following an initial free 45 minutes per ride.
- For **user_type=Customer**, the fee is **$4.00 per 15 minutes** following an initial free 30 minutes per ride.
- There are some cases where the user type is not specified (we have relabeled as "UNKNOWN", and we do note estimate usage fees for such trips.)



Summary of trip durations AFTER censoring/truncation: After we eliminate cases which result in extreme values, the duration of the remaining trips is more reasonable.

Table 2: Summary of trip durations AFTER truncations:

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
supplied_secs	61.0000000	364.0000000	594.0000000	777.6994724	997.0000000	10617.0000000
calc_secs	60.0000000	364.0000000	594.4970000	778.1873234	997.7920001	10617.5160000
calc_mins	1.0000000	6.0666667	9.9082833	12.9697887	16.6298667	176.9586000
calc_hours	0.0166667	0.1011111	0.1651381	0.2161631	0.2771644	2.9493100
calc_days	0.0006944	0.0042130	0.0068808	0.0090068	0.0115485	0.1228879

We could have chosen to ***censor*** the data, in which case we would not drop observations, but would instead move them to a limiting value, such as three hours (for trip time) or an age of 90 years (for adjusting birth_year).
As there were few such cases, we instead decided to ***truncate*** the data by dropping such observations from the dataset.

Limitations and Challenges in uploading and analyzing this data

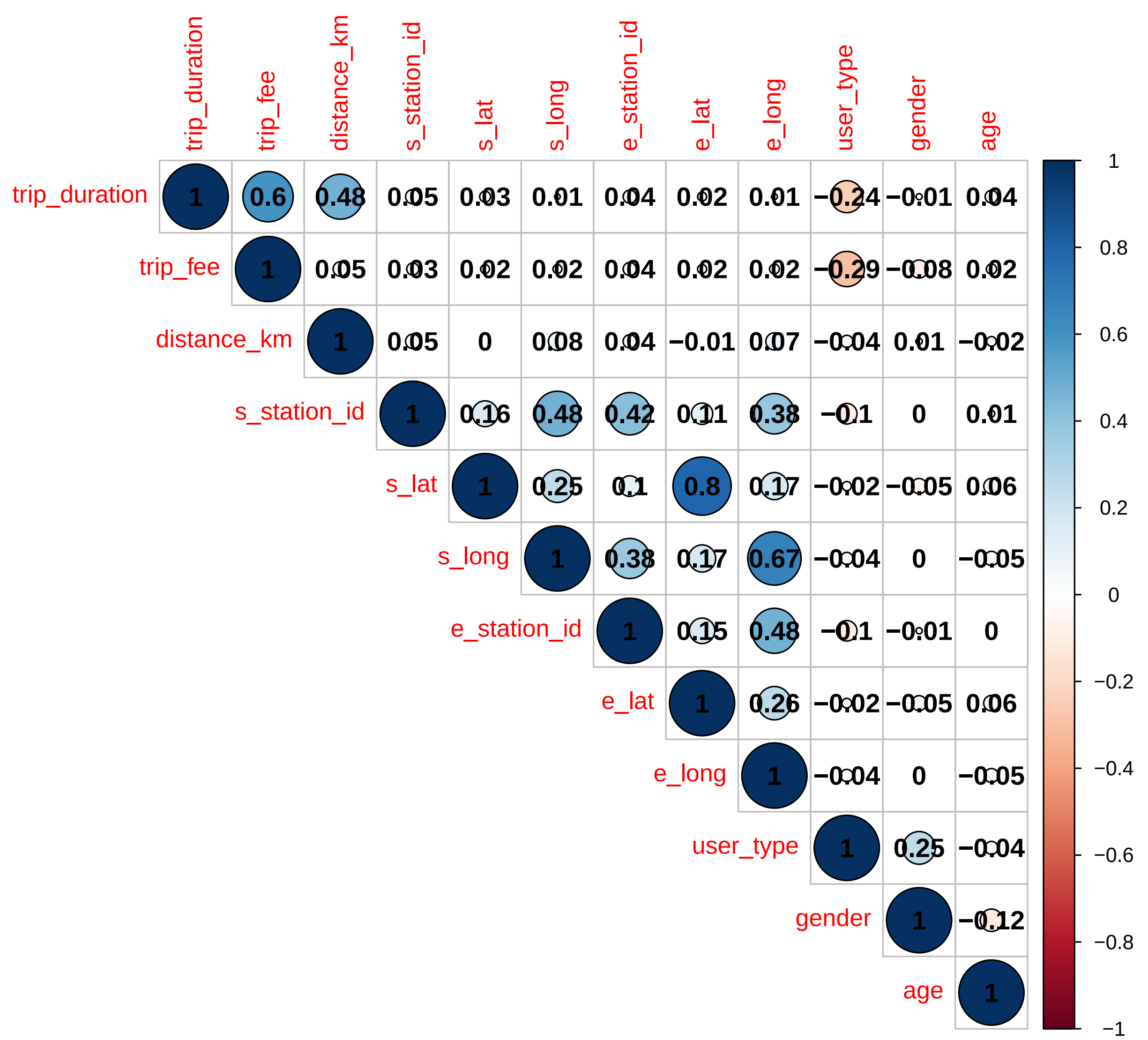
Data Size Because there is so much data, it is difficult to analyze the entire universe of trip-by-trip data unless one has high-performance computational resources.

Data formatting inconsistencies from month to month:

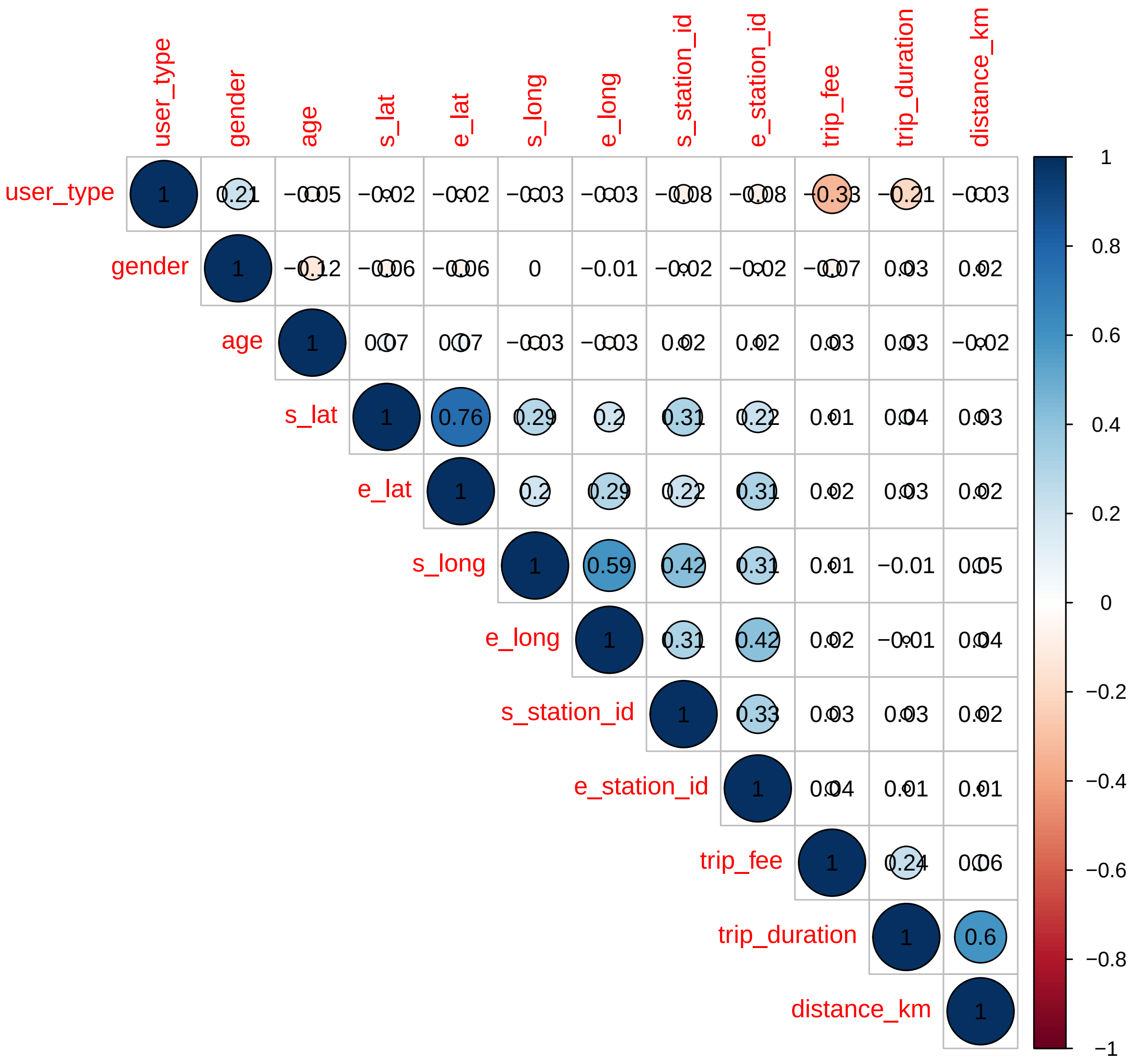
- Data column names change slightly from month to month.
- In some months, CitiBike specifies dates as YYYY-MM-DD, while in other months, dates are MM/DD/YYYY .
- In certain months, the timestamps include HH:MM:SS (as well as fractional seconds) while in other months, timestamps only include HH:MM , as seconds are omitted entirely.
- We encountered an unusual quirk which manifests itself just once a year, on the first Sunday of November, when clocks are rolled back an hour as Daylight Savings time changes to Standard time:
  - The files do not specify whether a timestamp is EDT or EST. On any other date, this is not a problem, but the hour of 1am-2am EDT on that November Sunday is followed by an hour 1am-2am EST.
  - If someone rents a bike at, say, 1:55am EDT (before the time change) and then returns it 15 minutes later, the time is now 1:10am (EST).
  - The difference in time timestamps suggests that the rental was negative 45 minutes, which is of course impossible!
- Sometimes there is an unusually long interval between the start time of a bicycle rental and the time at which the system registers such rental as having concluded.

Correlations of individual trip data features We can examine the correlations between variables to understand the relationship between variables, and also to help be alert to potential problems of multicollinearity. Here we compute rank correlations (Pearson and Spearman) as well as actual correlations between key variables. Here we compute the correlations between key variables on the individual CitiBike Trip data. (Later we will compute correlations on daily aggregated data which has been joined with the daily weather observations.)

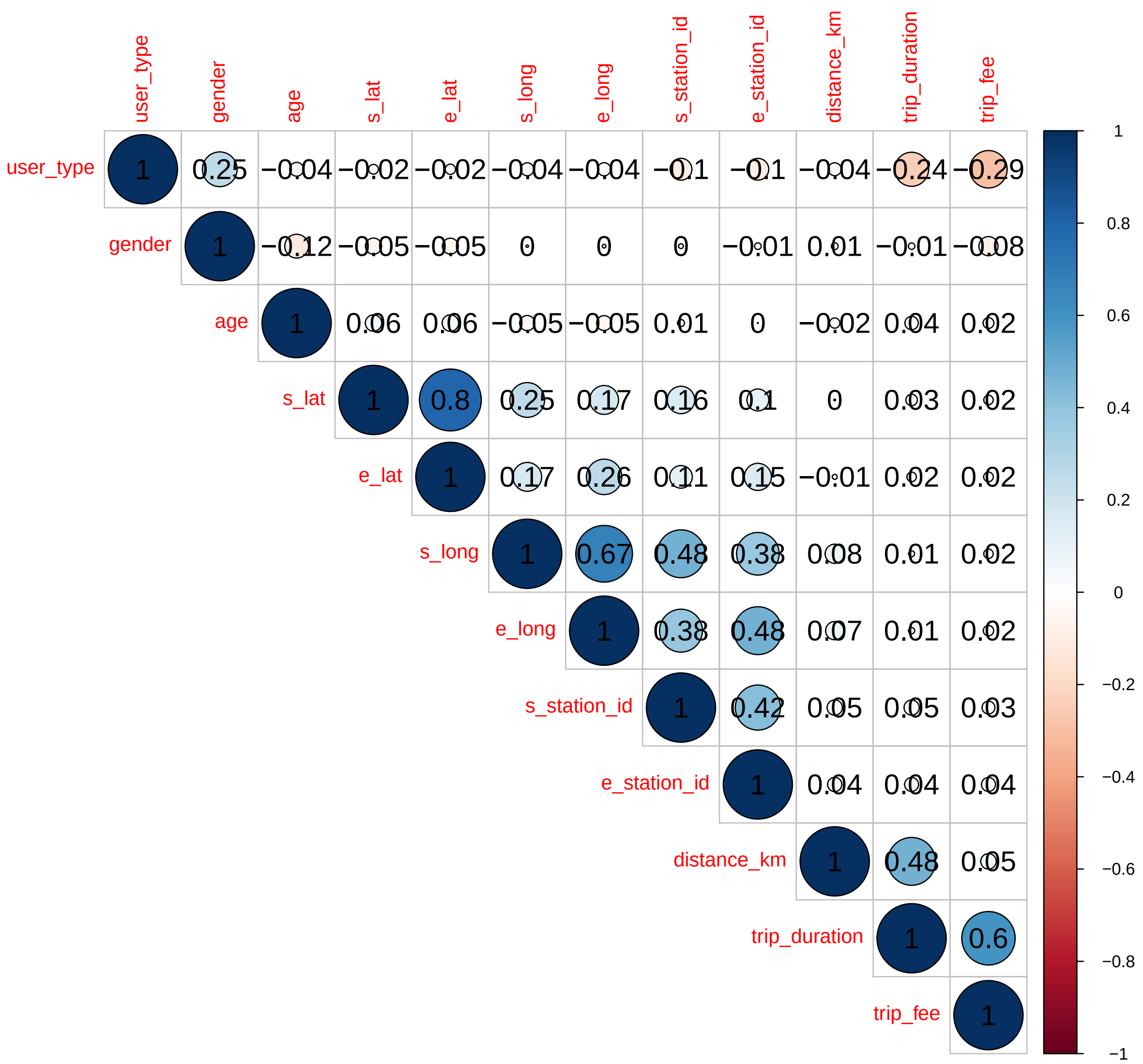
Rank Correlation (Pearson) on individual trip data



Rank Correlation (Spearman) on individual trip data



Actual correlations on individual trip data



Aggregate and join

Aggregate individual CitiBike trip data by day, and join to daily weather data We will perform our calculations on an aggregated basis. We will group each day's rides together, but we will segment by user_type ("Subscriber" or "Customer") and by gender ("Male or"Female"). For each of these segments, there are some cases where the user_type is not specified, so we have designated that as"Unknown." For gender, there are cases where the CitiBike data set contains a zero, which indicates that the gender of the user was not recorded.

For each day, we will aggregate the following items across each of the above groupings:

- mean trip_duration
- median trip_duration
- sum of distance_km
- sum of trip_fee
- mean of age
- count of number of trips on that day

We will split the aggregated data into a training dataset, consisting of all (grouped, daily) aggregations from 2013-2018, and a test dataset, consisting of (grouped, daily) aggregations from 2019.

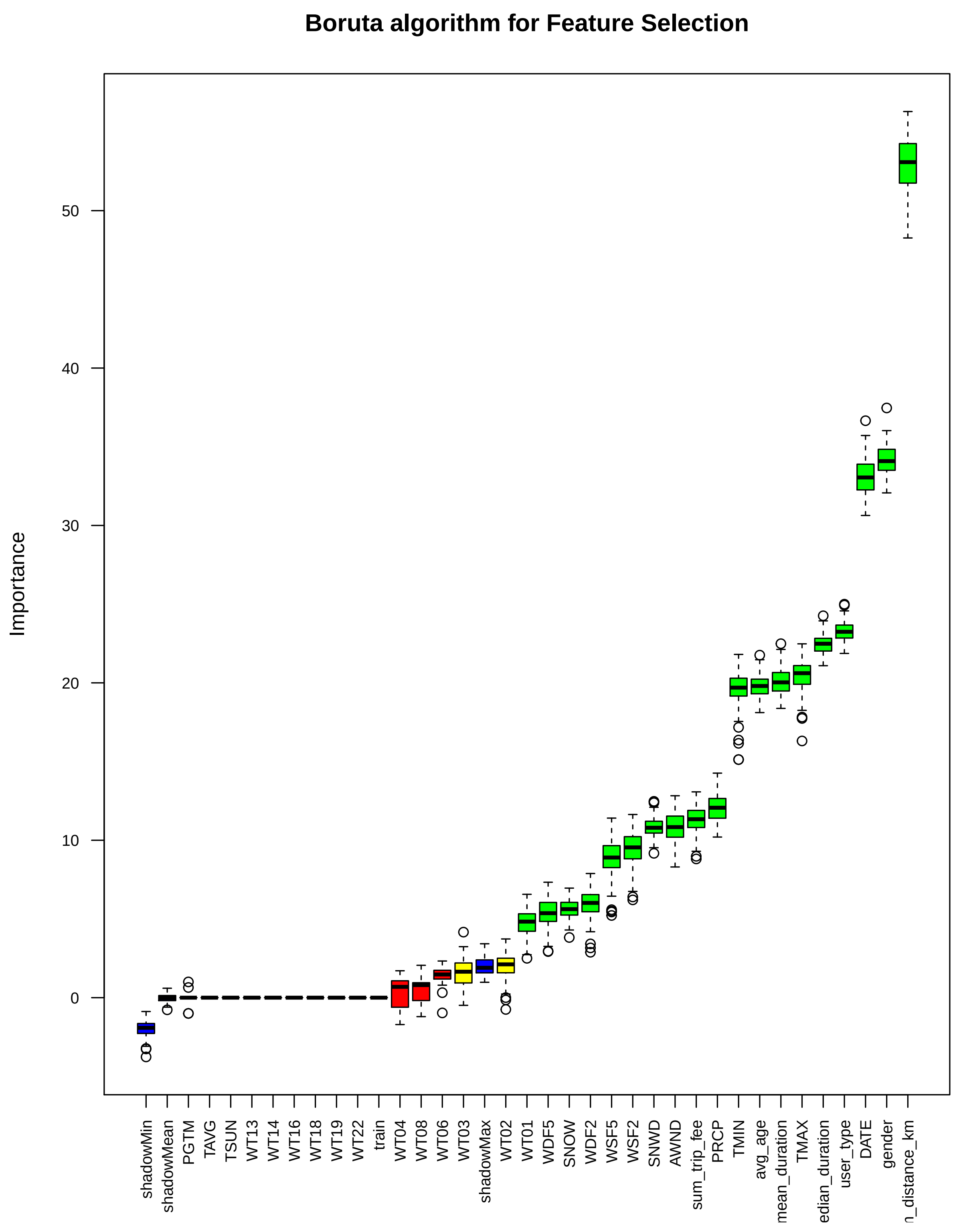
We will then join each aggregated CitiBike data element with the corresponding weather obserservation for that date.

There are 5202 rows of daily aggregated data in the training dataset, and 1621 rows in the corresponding test dataset.

Selection of variables designated as "Important" by the Boruta algorithm

There is a random-forest based algorithm known as "Boruta" which is useful in assessing whether each variable's "importance" in estimation of the target variable is "Confirmed", "Tentative", or "Rejected." Here we execute the Boruta algorithm on the training dataset, which consists of aggregated daily ride metrics (from the CitiBike dataset) joined with daily weather observation metrics.

The graph from the Boruta algoritm indicates the variables confirmed as important in green (on the right) while rejected variables are in red (on the left.) The variables in the middle (in yellow) are "tentative."

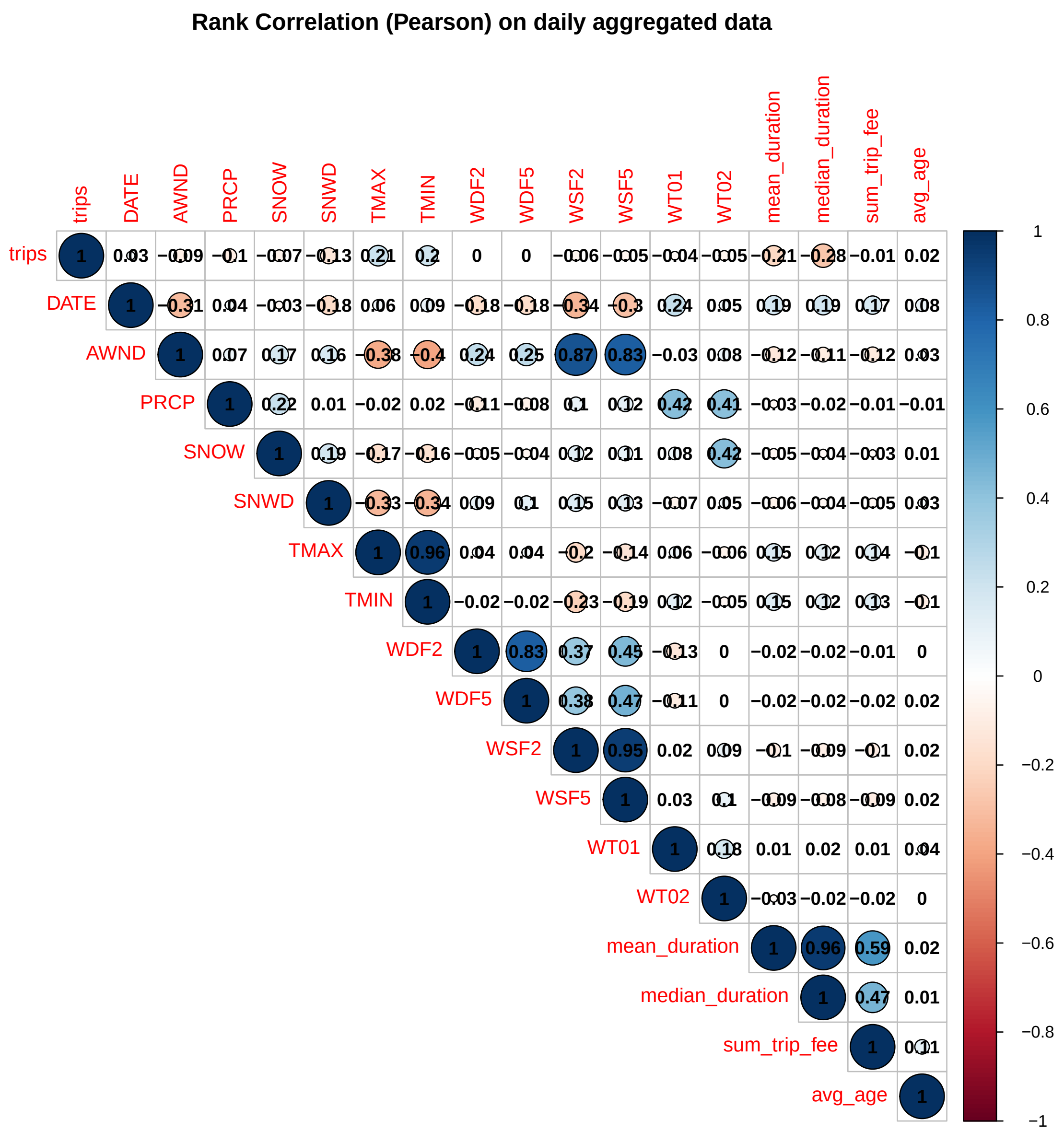


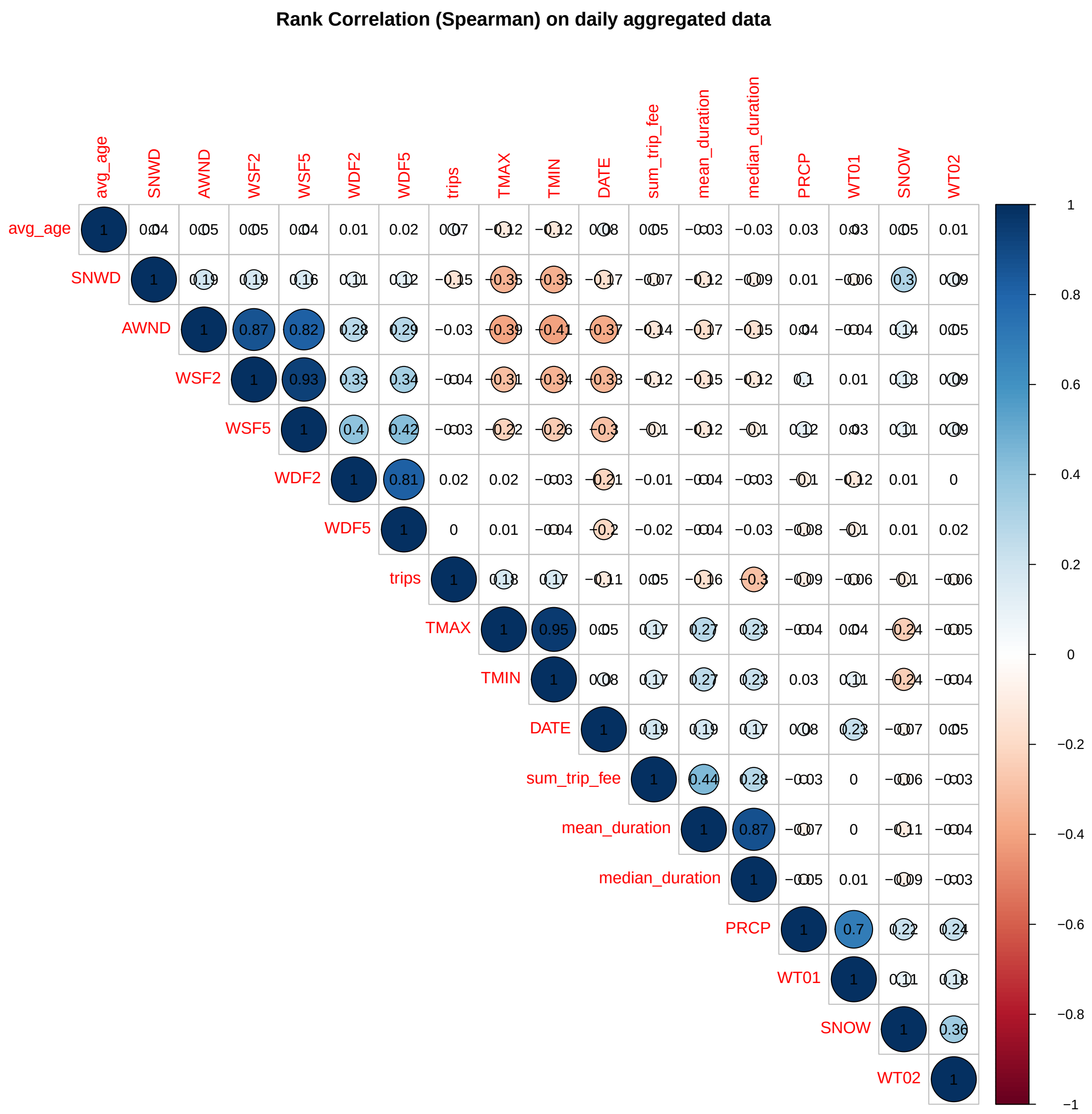
```
## [1] "Num important:  19"
```

Table 3: Boruta results sorted by median value

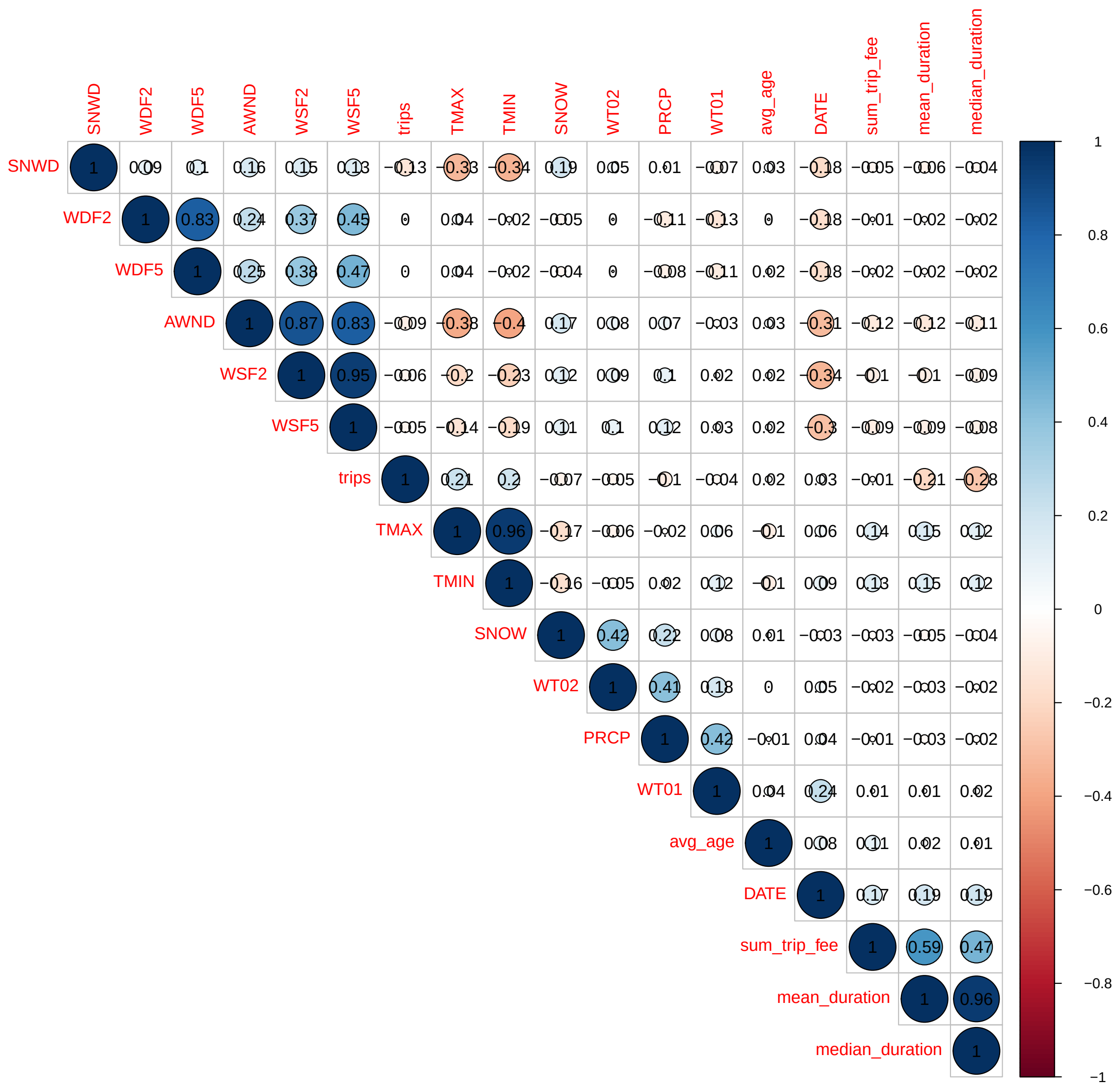
BorutaFinalAlphaNames	BorutaFinalAlphaResults	BorutaMedianAlphaNum
PGTM	Rejected	-Inf
TAVG	Rejected	-Inf
train	Rejected	-Inf
TSUN	Rejected	-Inf
WT04	Rejected	-Inf
WT06	Rejected	-Inf
WT08	Rejected	-Inf
WT13	Rejected	-Inf
WT14	Rejected	-Inf
WT16	Rejected	-Inf
WT18	Rejected	-Inf
WT19	Rejected	-Inf
WT22	Rejected	-Inf
WT03	Tentative	1.64945194243239
WT02	Tentative	2.11633350230996
WT01	Confirmed	4.83249465978541
WDF5	Confirmed	5.36695674002365
SNOW	Confirmed	5.6229979537176
WDF2	Confirmed	6.01907170623564
WSF5	Confirmed	8.90302463691063
WSF2	Confirmed	9.54648436389202
SNWD	Confirmed	10.7949118384744
AWND	Confirmed	10.8365323005427
sum_trip_fee	Confirmed	11.3369707957975
PRCP	Confirmed	12.0672514525478
TMIN	Confirmed	19.6963854571892
avg_age	Confirmed	19.8013140670482
mean_duration	Confirmed	20.0313281605762
TMAX	Confirmed	20.6156043555391
median_duration	Confirmed	22.4861858330595
user_type	Confirmed	23.2447374328727
DATE	Confirmed	33.0493493321453
gender	Confirmed	34.0878629495875
sum_distance_km	Confirmed	53.0821655307592

Correlations of daily-aggregated variables Having aggregated the CitiBike data into daily summarizations, and joined with Weather data, we compute the correlations of the features in the the training dataset.


Rank Correlation (Pearson) on daily aggregated data
trips
DATE
AWND
PRCP
SNOW
SNWD
TMAX
TMIN
WDF2
WDF5
WSF2
WSF5
WT01
WT02
mean_duration
median_duration
sum_trip_fee
avg_age
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1


Rank Correlation (Spearman) on daily aggregated data
avg_age
SNWD
AWND
WSF2
WSF5
WDF2
WDF5
trips
TMAX
TMIN
DATE
sum_trip_fee
mean_duration
median_duration
PRCP
WT01
SNOW
WT02
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1

Actual correlations on daily aggregated data)



Experimentation and Results

Build Models

We will create various models to estimate the total number of trips per day within each category.
We know the actual results (number of trips) in the sampled subset, but we will pull out this column from the test dataset and save into a separate variable for RMSE calculations afterwards.

We will fit standard linear models – both linear and loglinear – using a variety of variables. Then we will fit several geneneralized linear models: Poisson and Gaussian.

The quality of the fit from each model will be assessed using the RMSE calcualtion.

Standard linear models

We will create models using a selection of "important" variables from the Boruta algorithm. At first we will exclude the data, then we will create another model which includes it. Finally we will create a third model which drops those variables deemed not statistically significant.

Because the standard linear model creates predictions which could be negative, we will also estimate a corresponding log-linear model for each of the above models. We will fit log(trips) rather than trips, which will yield a fitted result in log-space. We must then exponentiate the predictions returned by such model in order to get the estimated trip counts.

linear model 1 - without DATE variable

The following variables will be included in this model:

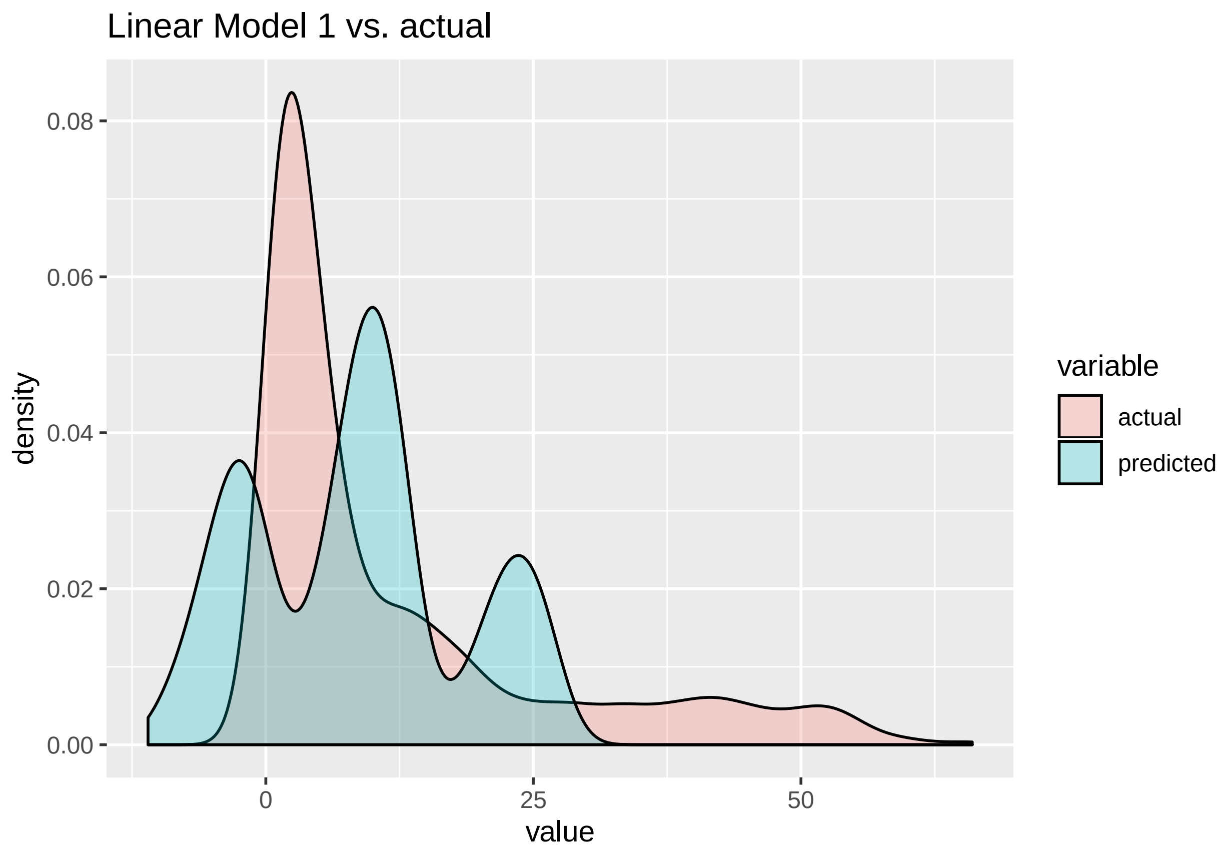
- gender
- user_type
- avg_age
- AWND - Average Daily Wind Speed
- PRCP - Daily Precipitation amount
- SNOW - amount of snowfall
- SNWD - depth of snow on the ground
- TMAX - daily Maximum temperature
- TMIN - daily Minimum temperature
- WDF2 - direction of strongest 2-minute wind gust
- WDF5 - direction of strongest 5-second wind gust
- WSF5 - speed of biggest 5-second wind gust
- WT01 - indicator for fog today
- WT02 - indicator for heavy fog today

The full regression output can be found in the appendix.

Table 4: Linear Model 1 vs. actual

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
predicted	-11.0037	-0.7759302	8.783169	8.338103	12.87871	28.32734
actual	1.0000	2.0000000	5.000000	12.064775	16.00000	66.00000

```
## [1] "lm.1 RMSE: 10.19"
```



This is not a very good model.

LOG linear model 1a - without DATE variable

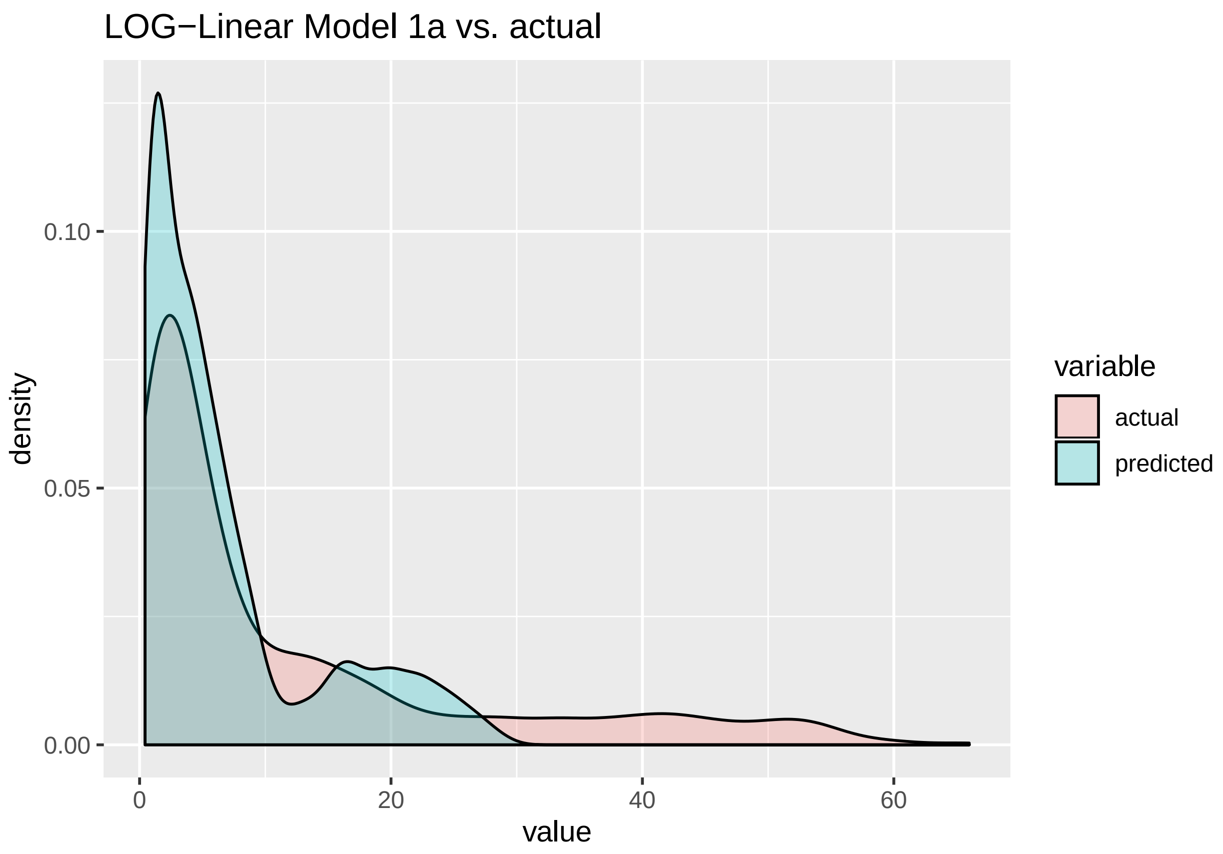
This is the same as the above model, except the dependent variable is log(trips) rather than trips)

The full regression output can be found in the appendix.

Table 5: LOG-Linear Model 1a vs. actual

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
predicted	0.4323213	1.42994	4.210182	6.879069	8.506657	29.55027
actual	1.0000000	2.00000	5.000000	12.064775	16.000000	66.00000

```
## [1] "lm.1a RMSE: 10.2"
```



This model is somewhat improved vs. the previous one. Most notably, there are no negative predctions.

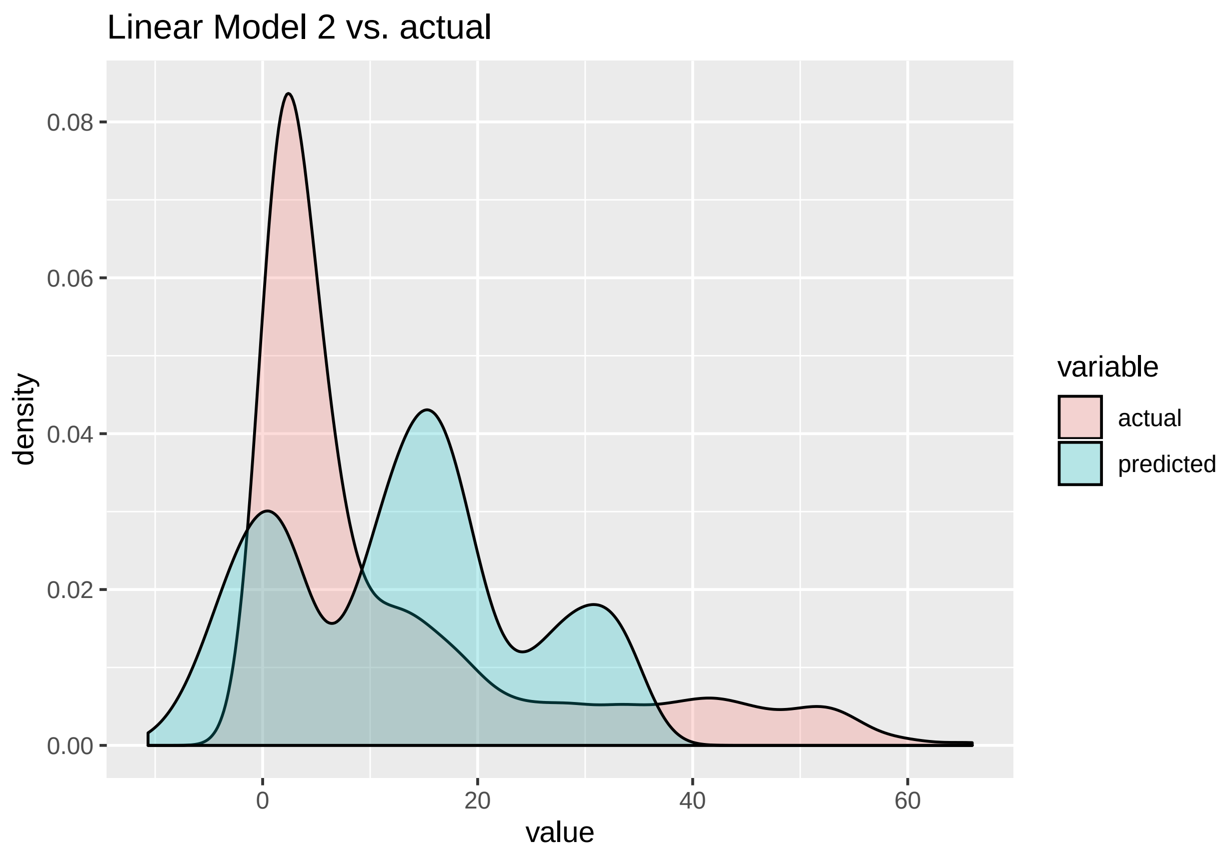
linear model 2 - with DATE variable

This is the same as the first model, except the DATE variable has been included. Doing so reflects the trend over time, as the system has grown and usage has increased over time.

Table 6: Linear Model 2 vs. actual

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
predicted	-10.65884	2.526574	13.72642	13.05295	19.38826	35.76022
actual	1.00000	2.000000	5.00000	12.06477	16.00000	66.00000

```
## [1] "lm.2 RMSE: 9.14"
```



While improved, this is still not a good model

LOG linear model 2a - with DATE variable

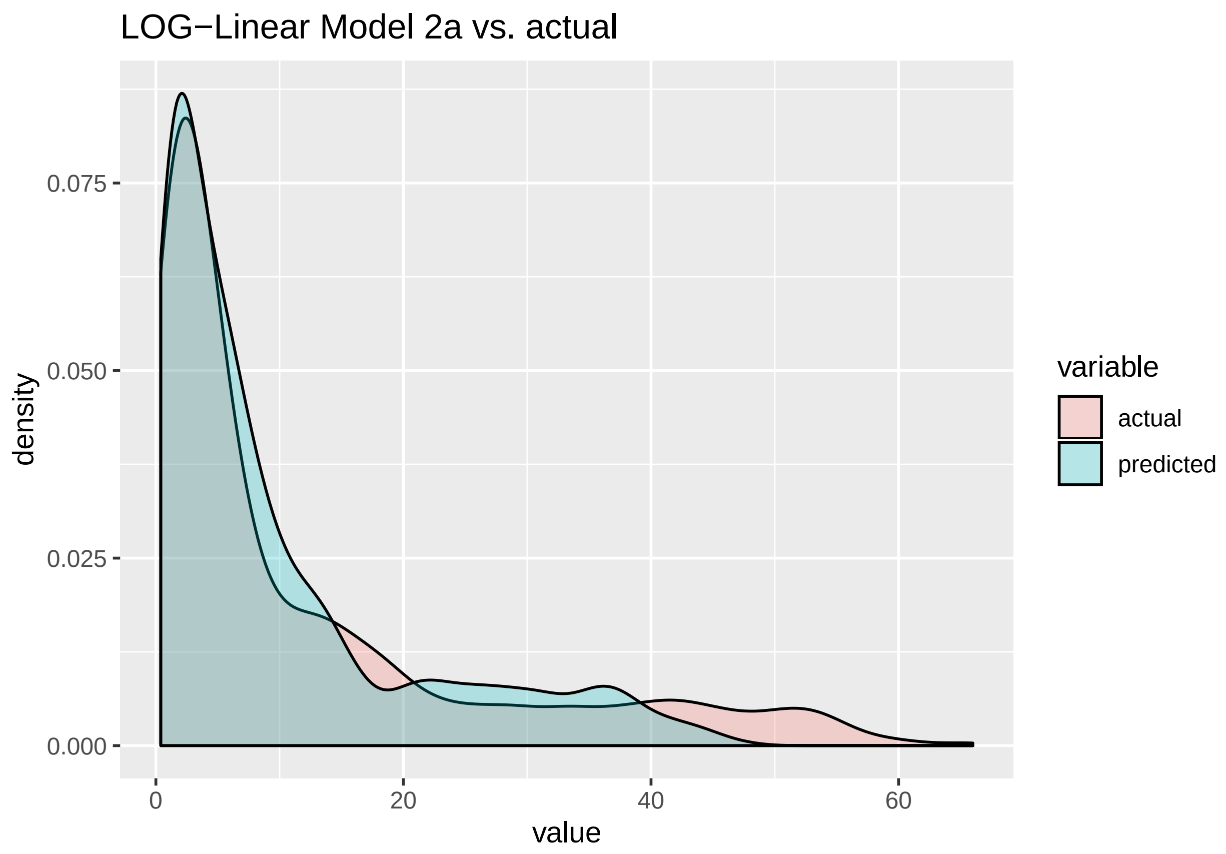
This is the same as the above model, with DATE added, except the dependent variable is log(trips) rather than trips).

The full regression output can be found in the appendix.

Table 7: LOG-Linear Model 2a vs. actual

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
predicted	0.3958614	1.799535	5.587975	9.95053	13.17236	47.6783
actual	1.0000000	2.000000	5.000000	12.06477	16.00000	66.0000

```
## [1] "lm.2a RMSE: 6.59"
```



This model exhibits substantial improvement over the predecessors.

linear model 3 - remove insignificant variables

The following variables will be included in this model:

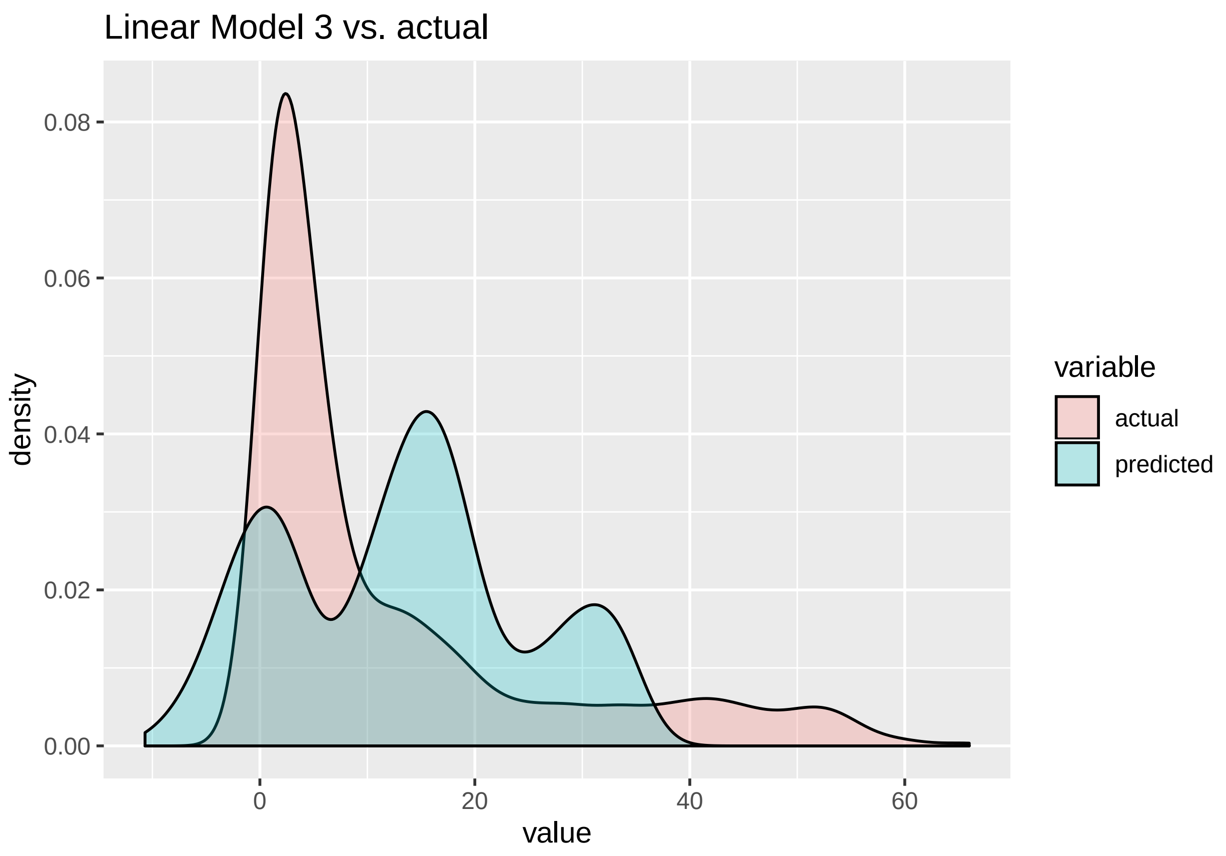
- DATE
- gender
- user_type
- avg_age
- PRCP - Daily Precipitation amount
- SNWD - depth of snow on the ground
- TMAX - daily Maximum temperature
- WT01 - indicator for fog today

The full regression output can be found in the appendix.

Table 8: Linear Model 3 vs. actual

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
predicted	-10.68506	2.608322	13.77932	13.11463	19.44488	35.91916
actual	1.00000	2.000000	5.00000	12.06477	16.00000	66.00000

```
## [1] "lm.3 RMSE: 9.13"
```



Still not a good model

LOG linear model 3a - remove insignificant variables

The following variables will be included in this model:

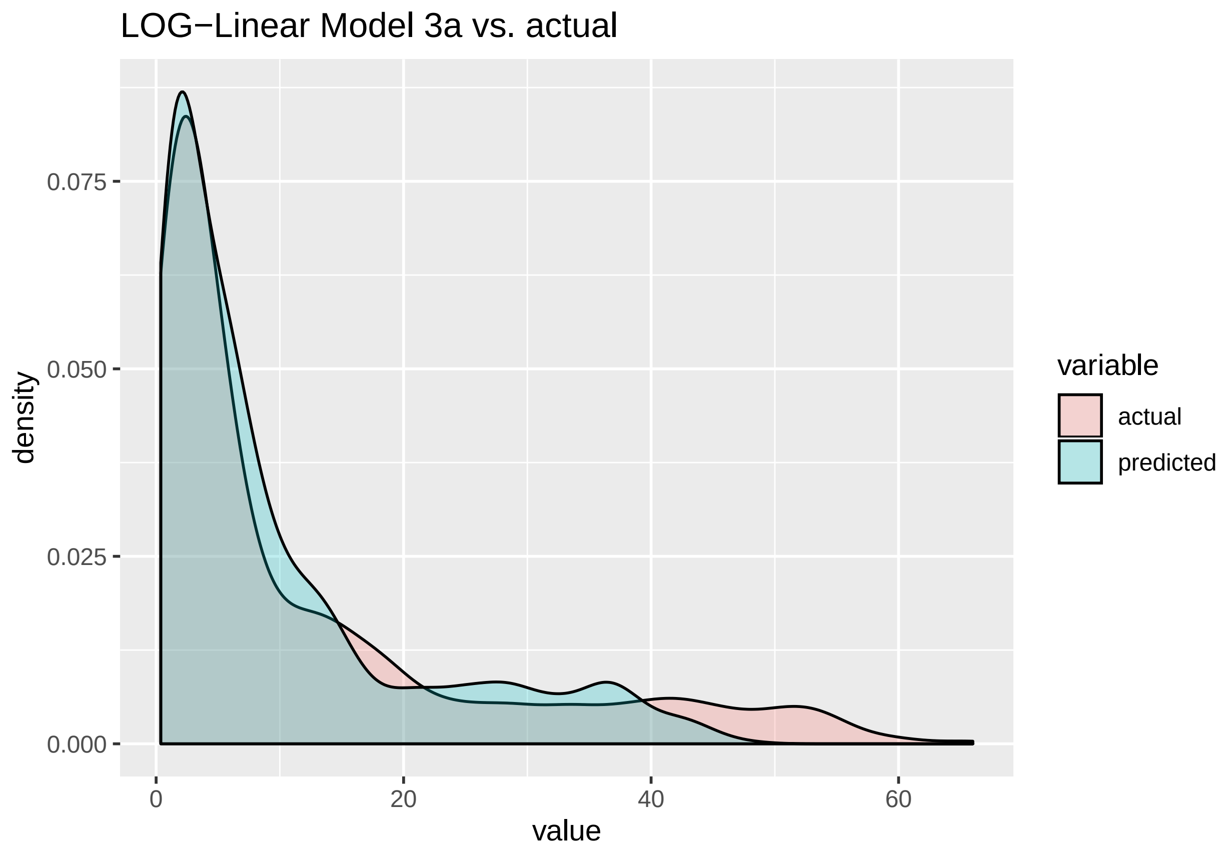
- DATE
- gender
- user_type
- avg_age
- PRCP - Daily Precipitation amount
- SNOW - amount of snowfall
- SNWD - depth of snow on the ground
- TMAX - daily Maximum temperature
- WT01 - indicator for fog today

The full regression output can be found in the appendix.

Table 9: LOG-Linear Model 3a vs. actual

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
predicted	0.377101	1.819508	5.611717	9.986853	13.15124	48.49199
actual	1.000000	2.000000	5.000000	12.064775	16.00000	66.00000

```
## [1] "lm.3a RMSE: 6.53"
```



This model improves slightly over LOG-linear model 2a and appears to fit the actual data well.

Generalized linear models

We will create two poisson models and two gaussian models to estimate the daily number of trips.

The first of each pair will contain all the variables, and the second will contain just those variables deemed significant.

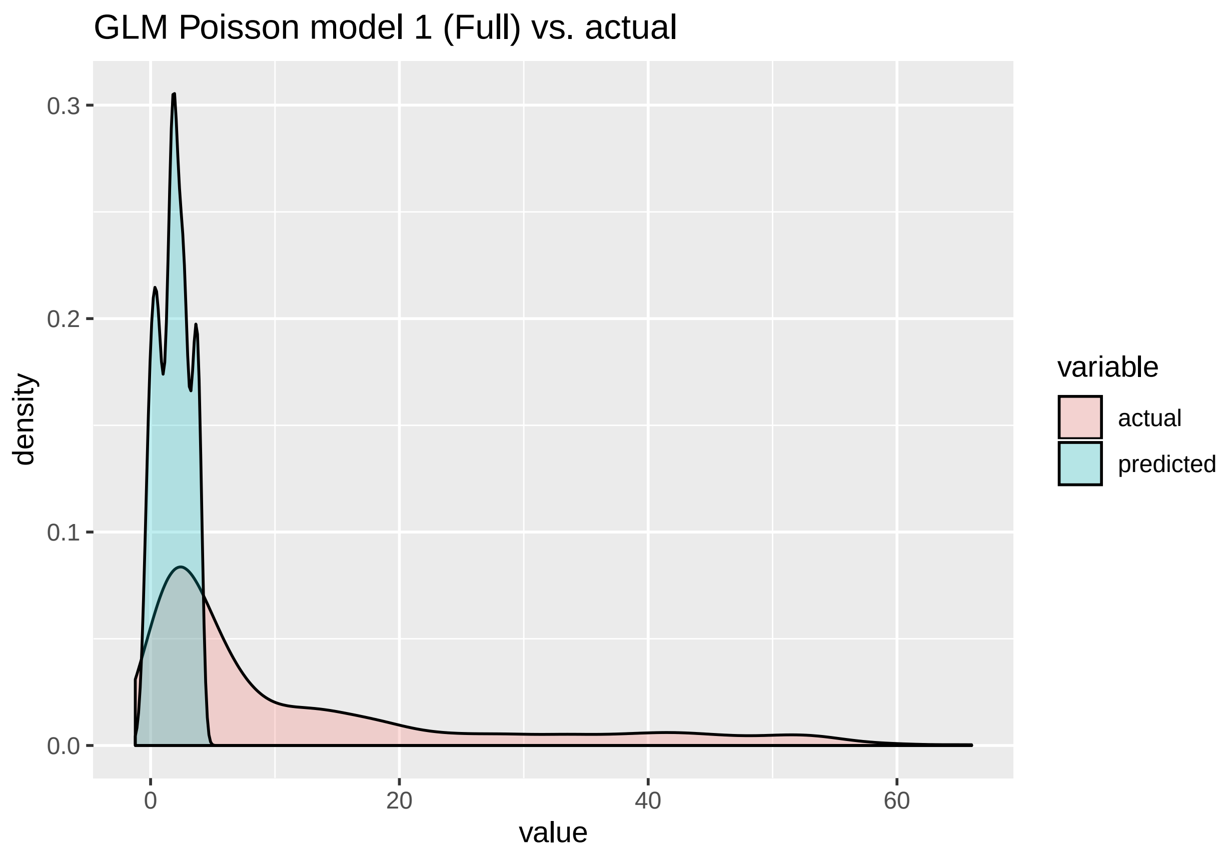
Results will be evaluated by RMSE.

GLM poisson 1 – all variables in the training dataset

Table 10: GLM Poisson model 1 (Full) vs. actual

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
predicted	-1.229372	0.7341554	1.879448	1.842303	2.792683	4.435388
actual	1.000000	2.0000000	5.000000	12.064775	16.000000	66.000000

```
## [1] "glm.poisson.1 RMSE: 17.14"
```



Not a good model at all!

GLM poisson 2 - remove insignificant variables

The following variables will be included in this model:

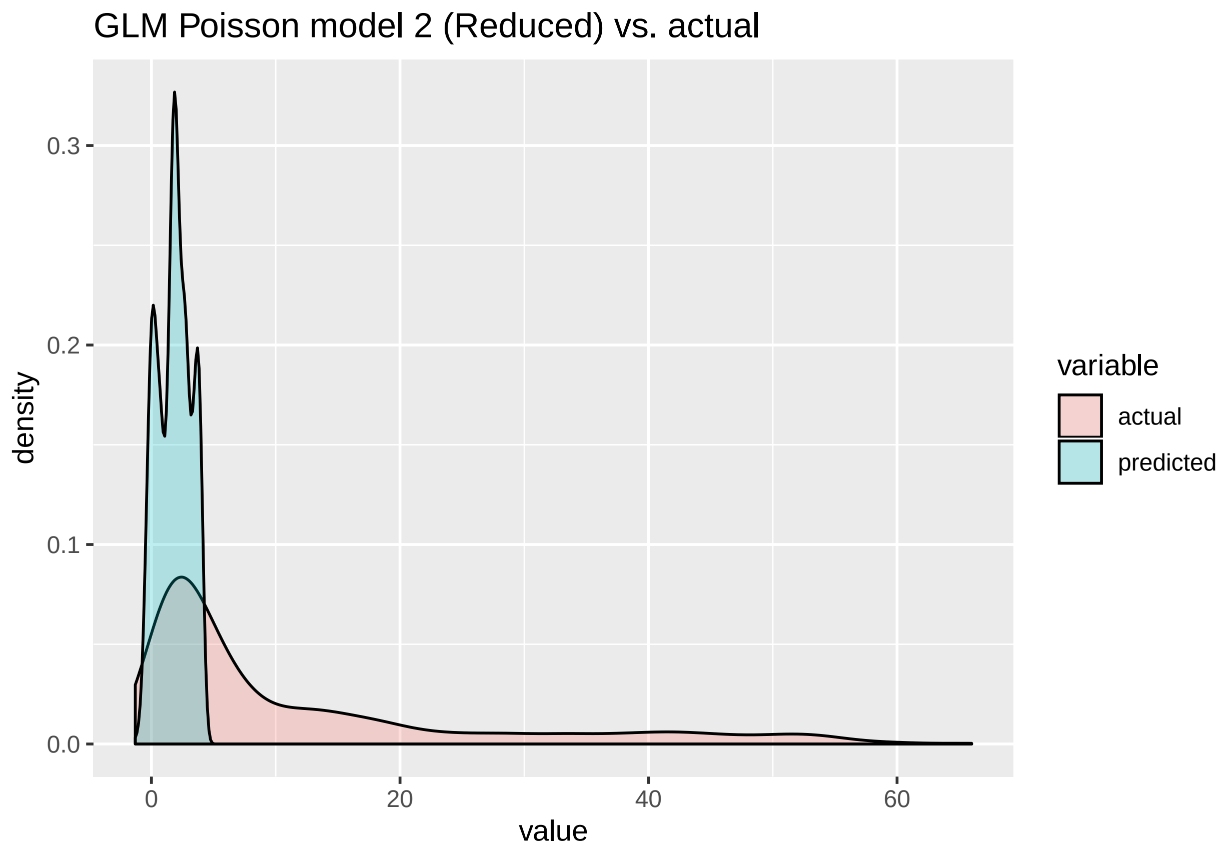
- DATE
- gender
- user_type
- avg_age
- PRCP - Daily Precipitation amount
- SNOW - amount of snowfall
- SNWD - depth of snow on the ground
- TMAX - daily Maximum temperature
- WT01 - indicator for fog today

The full regression output can be found in the appendix.

Table 11: GLM Poisson model 2 (Reduced) vs. actual

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
predicted	-1.300867	0.702728	1.872521	1.825238	2.808056	4.134755
actual	1.000000	2.000000	5.000000	12.064775	16.000000	66.000000

```
## [1] "glm.poisson.2 RMSE: 17.15"
```



Also not a good model!

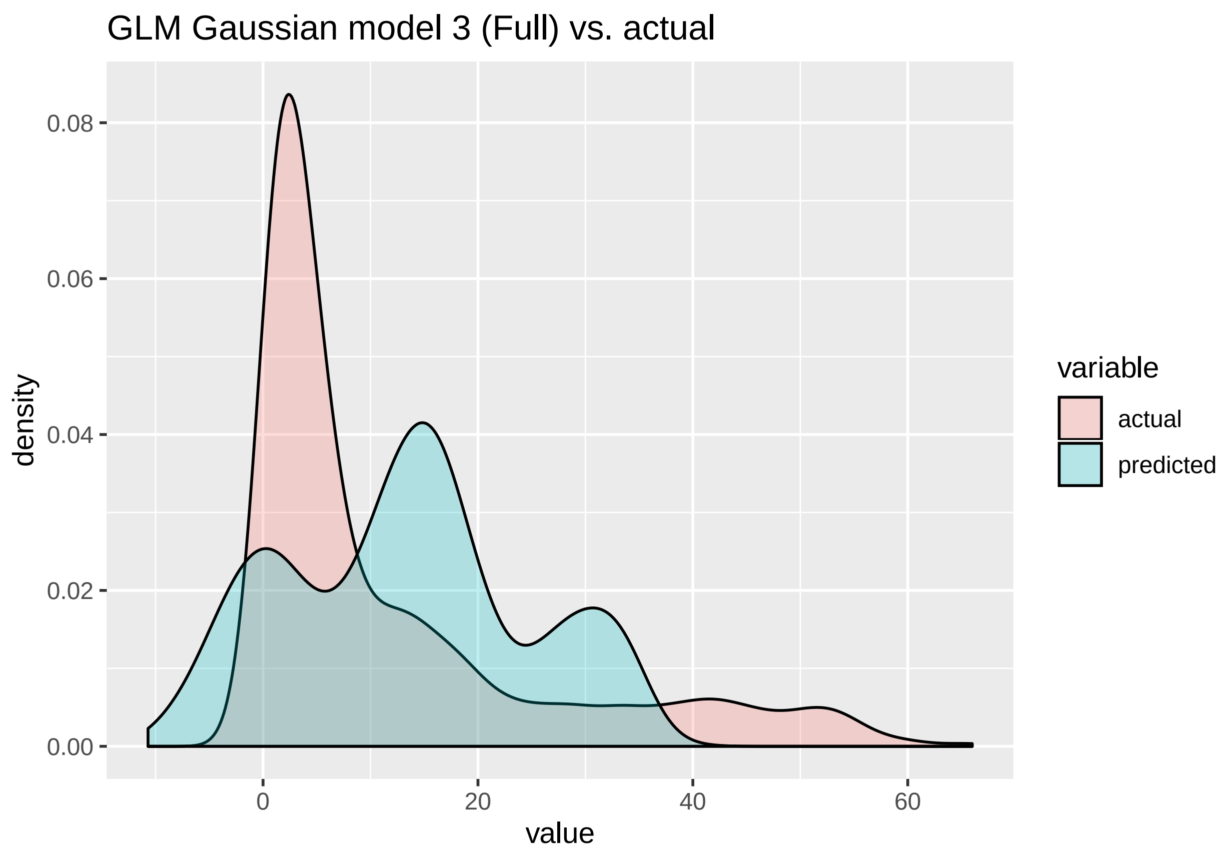
GLM gaussian 3 - – all variables in the training dataset

The full regression output can be found in the appendix.

Table 12: GLM Gaussian model 3 (Full) vs. actual

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
predicted	-10.70314	3.731717	13.63858	13.23580	19.94066	39.80248
actual	1.00000	2.000000	5.00000	12.06477	16.00000	66.00000

```
## [1] "glm.gaussian.3 RMSE: 9.05"
```



Still not a good model.

GLM gaussian 4 - remove insignificant variables

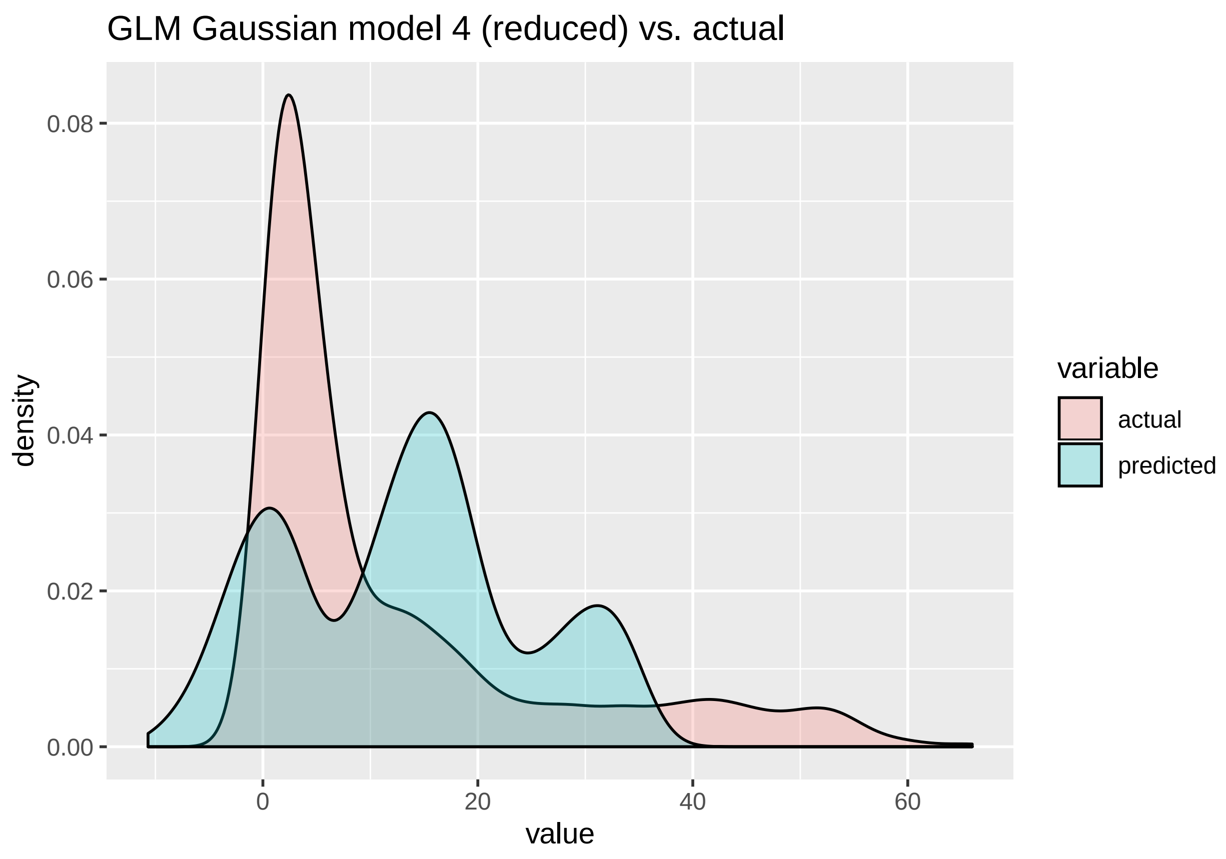
- DATE
- gender
- user_type
- avg_age
- PRCP - Daily Precipitation amount
- SNOW - amount of snowfall
- SNWD - depth of snow on the ground
- TMAX - daily Maximum temperature
- WT01 - indicator for fog today

The full regression output can be found in the appendix.

Table 13: GLM Gaussian model 4 (reduced) vs. actual

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
predicted	-10.68506	2.608322	13.77932	13.11463	19.44488	35.91916
actual	1.00000	2.000000	5.00000	12.06477	16.00000	66.00000

```
## [1] "glm.gaussian.4 RMSE: 9.13"
```



4. Results

Table of RMSE results

Here is a table which summarizes the RMSE values from the 10 models above:

Table 14: Table of RMSE results on 1/1000 of dataset

model_names	model_rmse
Linear Model 1: no dates	10.19
Log-Linear Model 1a: no dates	10.2
Linear Model 2: add dates	9.14
Log-Linear Model 2a: add dates	6.59
Linear Model 3: reduced to significant vars	9.13
Log-Linear Model 3a: reduced to significant vars	6.53
GLM model 1: Full Poisson Model	17.14
GLM model 2: Poisson reduced to significant vars	17.15
GLM model 3: Full Gaussian Model	9.05
GLM model 4: Gaussian reduced to significant vars	9.13

Findings

The findings from comparing the 3 standard linear models (and, their log-linear counterparts) are:

1. DATE is a significant variable in the linear model, when we comparing model 1 and 2 by adding the DATE variable. It can tell strong seasonality in the data as we didn't take holiday and weekends out of the model. The influence is bigger than any other factor. The RMSE improved from 10.19 to 9.14. The improvement is about 10 percent.
2. Removing insignificant variables from linear model 2 to model 3, the model performance is similar; RMSE is essentially same in model 2 and 3 with and without those variables.

However, the problem with estimating daily trips as a straight **linear** model is that such a model can return a negative value, which is nonsensical. Therefore, we also estimate **log(trips)** for each of the above sets of independent variables.

The RMSE for `log(trips)` (i.e, model 1a) is about the same as that of the corresponding linear model, 10.2. There is substantial improvement when we incorporate the date, in model 2a: the RMSE drops to 6.59. which is an improvement of more than 35 percent. There is very slight further improvement on model 3a (where insignificant variables have been dropped): the RMSE improves by a further 1 percent to 6.53 .

The findings from comparing the 4 generalized linear models (Poission and Gaussian) are:

1. With Poisson models 1 and 2, the model with more variables has a better outcome. The first model with all variables has a better RMSE 17.14 vs. 17.15. The slight decrease of RMSE should be considered for parsimoniousness, choosing the second poisson model with only 6 variables. However, with such a high RMSE, both poisson models are quite bad.
2. With Gaussian models 3 and 4, the model with more variables has a slightly outcome. The first model, incorporating all variables has a better RMSE 9.05 vs. the second model 9.13. Again, for parsimoniousness, the preferred model is glm.gaussian.4 .

3. Comparing the models between Poisson family and Gaussian family, the results are perfectly clear: The Gaussian models have a much better outcome with least RMSE 9.05 vs. the best outcome from Poisson family, 17.14.

Select Preferred Model

The overall model selection will be the **LOG-linear model 3a**, which had the lowest RMSE and for which the density plot appeared to most closely overlap that of the actual results.

The equation for this model is

$$\begin{aligned}
log(trips) = &- 7.4150383911 \\
&+ 0.0003888249 \cdot DATE \\
&- 0.9965446415 \cdot I_{gender=Female} \\
&- 1.8523801948 \cdot I_{gender=UNKNOWN} \\
&+ 1.7355605144 \cdot I_{user\_type=Subscriber} \\
&- 0.5164012581 \cdot I_{user\_type=UNKNOWN} \\
&+ 0.0266931302 \cdot avg\_age \\
&- 0.2840324648 \cdot PRCP \\
&- 0.0456678286 \cdot SNOW \\
&- 0.0680191810 \cdot SNWD \\
&+ 0.0156835391 \cdot TMAX \\
&- 0.0691515019 \cdot WT01
\end{aligned}$$

The ***positive*** coefficients indicate the following:

- DATE: this reflects the trend, over time, of increasing daily bike rentals.
- user_type=Subscriber: this confirms that many more rides are taken by subscribers than by customers (the base category, omitted from the above table as it is embedded into the intercept term)
- avg_age: indicates that more rides are taken on days when older riders show up to participate
- TMAX: indicates that more rides are taken on days when the maximum temperature is warmer.

The ***negative coefficients*** indicate the following:

- gender=Female: fewer trips are taken (collectively) by female riders than by male riders (the base category, omitted from the above as it is embedded into the intercept term)
- gender=UNKNOWN: many fewer trips are taken (collectively) by the group of users whose gender is known. (However, this could be because such group of users may be small.)
- user_type=UNKNOWN: fewer trips are taken (collectively) by the group of users for whom it is unknown whether they are annual subscribers or daily customers (the base category.) (As above, this could be because such group of users may be small.)
- PRCP: this confirms that fewer trips are taken on days with heavy precipitation.
- SNOW: this confirms that fewer trips are taken on days when show is falling.
- SNWD: fewer trips are taken on days when there is already substantial depth of snow on the ground.
- WT01: fewer trips are taken on days when fog has been recorded.

5. Discussion

In this CitiBike predictive analysis, we experienced a number of limitations in dataset collection and manipulations as described below, and a plan for further development will be discussed.

1. On the weather data we have only daily reporting, so we couldn't do weather-based hourly or peak-hours predictive analysis.
2. We used 1/1000 sample of the CitiBike data. The data size is very large, around 90 million records from 2013 to 2019. We tried to use AWS Athena to manipulate the data. Because the format of the CitiBike dataset changed from month to month, AWS Athena was not successful in uploading the data compeletly. So we sampled the data using a deterministic process to take each 1000th value from the data files.
3. We used the numeric value of DATE as one of the predictive variables. The positive coefficient on DATA reflects the trend in increased ridership over time. A potential future enhancement would be to model seasonality, which could be achieved using monthly dummies, or a "Winter" vs. "Summer" indicator. As we did not study time series models in this course, this ruled out the use of techniques such as SARIMA (Seasonal ARIMA)-based regression techniques.

6. Conclusion, Summary, and Future Work

Because of limitations with time and computing power, even though we did much research on the CitiBike business, rules, and regulations, we could not contribute all our ideas into the models.
From the comprehensive literature review, in the future we could expand the analysis further, such as prediction of aggregate trip fees to determine revenue generation, etc.

Interestingly, CitiBike just announced that from next month they will be modifying their formula for assessing fees on long trips. Instead of using the 15-minute increment which they have used since they started, they will instead assess fees per minute. This is a welcome change because under the present scheme, a ride of 45 minutes and one second is assessed the same surchage as a ride of 59 minutes and 59 seconds. Under the revised scheme, which takes effect from January 15, 2020, all users will be assessed 15 cents per minute after their initial free period. https://www.citibikenyc.com/blog/electric-bikes-returning . It would be interesting to revisit this analysis in the future as updated data could inform whether users are willing to extend their rides despite incurring small fees.

References

An, Ran, Renee Zahnow, Dorina Pojani, and Jonathan Corcoran. 2019. “Weather and Cycling in New York: The Case of Citibike.” *Journal of Transport Geography* 77 (May): 97–112. https://doi.org/10.1016/j.jtrangeo.2019.04.016.

Dell’Amico, Mauro, Eleni Hadjicostantinou, Manuel Iori, and Stefano Novellani. 2014. “The Bike Sharing Rebalancing Problem: Mathematical Formulations and Benchmark Instances.” *Omega* 45 (June): 7–19. https://doi.org/10.1016/j.omega.2013.12.001.

Dell’Amico, Mauro, Manuel Iori, Stefano Novellani, and Anand Subramanian. 2018. “The Bike Sharing Rebalancing Problem with Stochastic Demands.” *Transportation Research Part B: Methodological* 118 (December): 362–80. https://doi.org/10.1016/j.trb.2018.10.015.

Faghih-Imani, Ahmadreza, and Naveen Eluru. 2018. “A Finite Mixture Modeling Approach to Examine New York City Bicycle Sharing System (Citibike) Users’ Destination Preferences.” *Transportation*, June. https://doi.org/10.1007/s11116-018-9896-1.

Fuller, Daniel, Lise Gauvin, Yan Kestens, Mark Daniel, Michel Fournier, Patrick Morency, and Louis Drouin. 2013. “Impact Evaluation of a Public Bicycle Share Program on Cycling: A Case Example of Bixi in Montreal, Quebec.” *American Journal of Public Health* 103 (3): e85–e92. https://doi.org/10.2105/ajph.2012.300917.

Fuller, Daniel, Hui Luan, Richard Buote, and Amy H. Auchincloss. 2019. “Impact of a Public Transit Strike on Public Bicycle Share Use: An Interrupted Time Series Natural Experiment Study.” *Journal of Transport & Health* 13 (June): 137–42. https://doi.org/10.1016/j.jth.2019.03.018.

Heaney, Alexandra K., Daniel Carrión, Katrin Burkart, Corey Lesk, and Darby Jack. 2019. “Climate Change and Physical Activity: Estimated Impacts of Ambient Temperatures on Bikeshare Usage in New York City.” *Environmental Health Perspectives* 127 (3): 37002. https://doi.org/10.1289/ehp4039.

Hosford, Kate, Daniel Fuller, Scott A. Lear, Kay Teschke, Lise Gauvin, Michael Brauer, and Meghan Winters. 2018. “Evaluation of the Impact of a Public Bicycle Share Program on Population Bicycling in Vancouver, Bc.” *Preventive Medicine Reports* 12 (December): 176–81. https://doi.org/10.1016/j.pmedr.2018.09.014.

Hosford, Kate, Scott A. Lear, Daniel Fuller, Kay Teschke, Suzanne Therrien, and Meghan Winters. 2018. “Who Is in the Near Market for Bicycle Sharing? Identifying Current, Potential, and Unlikely Users of a Public Bicycle Share Program in Vancouver, Canada.” *BMC Public Health* 18 (1). https://doi.org/10.1186/s12889-018-6246-3.

Jia, Yingnan, Ding Ding, Klaus Gebel, Lili Chen, Sen Zhang, Zhicong Ma, and Hua Fu. 2019. “Effects of New Dock-Less Bicycle-Sharing Programs on Cycling: A Retrospective Study in Shanghai.” *BMJ Open* 9 (2): e024280. https://doi.org/10.1136/bmjopen-2018-024280.

Jia, Yingnan, and Hua Fu. 2019. “Association Between Innovative Dockless Bicycle Sharing Programs and Adopting Cycling in Commuting and Non-Commuting Trips.” *Transportation Research Part A: Policy and Practice* 121 (March): 12–21. https://doi.org/10.1016/j.tra.2018.12.025.

Nankervis, Max. 1999. “The Effect of Weather and Climate on Bicycle Commuting.” *Transportation Research Part A: Policy and Practice* 33 (6): 417–31. https://doi.org/10.1016/s0965-8564(98)00022-6.

Schmidt, Charles. 2018. “Active Travel for All? The Surge in Public Bike-Sharing Programs.” *Environmental Health Perspectives* 126 (8): 82001. https://doi.org/10.1289/ehp3754.

Vogel, Patrick, and Dirk C. Mattfeld. 2011. “Strategic and Operational Planning of Bike-Sharing Systems by Data Mining – a Case Study.” In *Lecture Notes in Computer Science*, 127–41. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-24264-9_10.

Wang, Shuai, Tian He, Desheng Zhang, Yuanchao Shu, Yunhuai Liu, Yu Gu, Cong Liu, Haengju Lee, and Sang H. Son. 2018. “BRAVO: Improving the Rebalancing Operation in Bike Sharing with Rebalancing

Range Prediction." *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2 (1): 1–22. https://doi.org/10.1145/3191776.

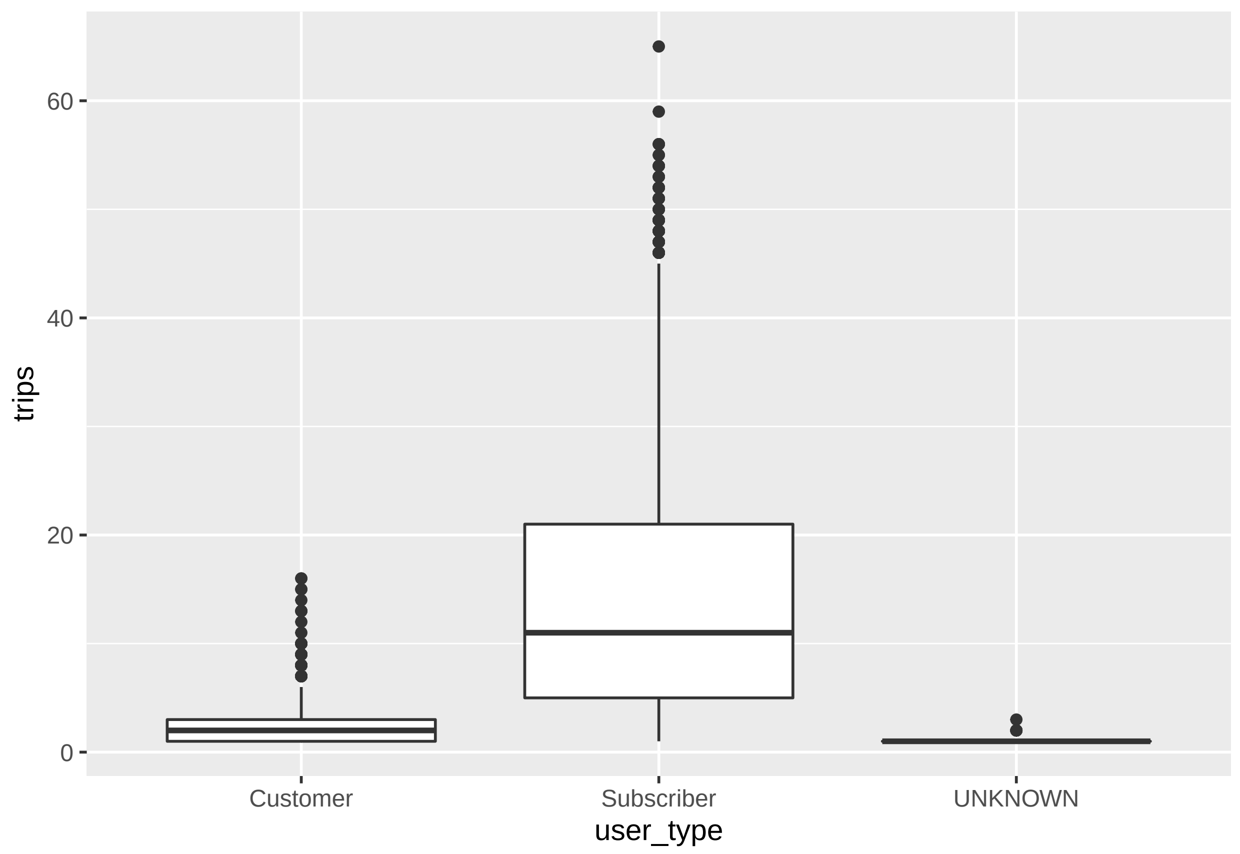
Westland, James Christopher, Jian Mou, and Dafei Yin. 2019. "Demand Cycles and Market Segmentation in Bicycle Sharing." *Information Processing & Management* 56 (4): 1592–1604. https://doi.org/10.1016/j.ipm.2018.09.006.

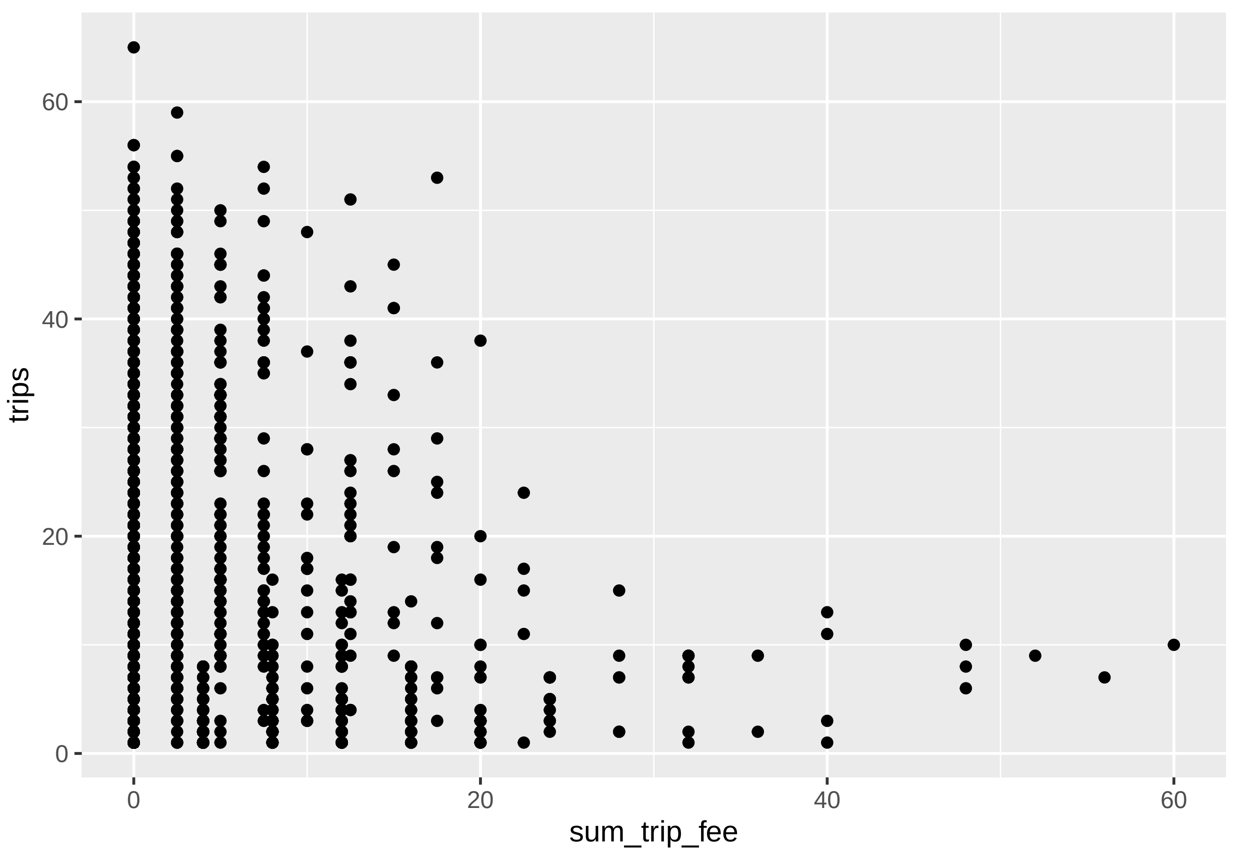
Zhou, Xiaolu. 2015. "Understanding Spatiotemporal Patterns of Biking Behavior by Analyzing Massive Bike Sharing Data in Chicago." Edited by Yanguang Chen. *PLOS ONE* 10 (10): e0137922. https://doi.org/10.1371/journal.pone.0137922.

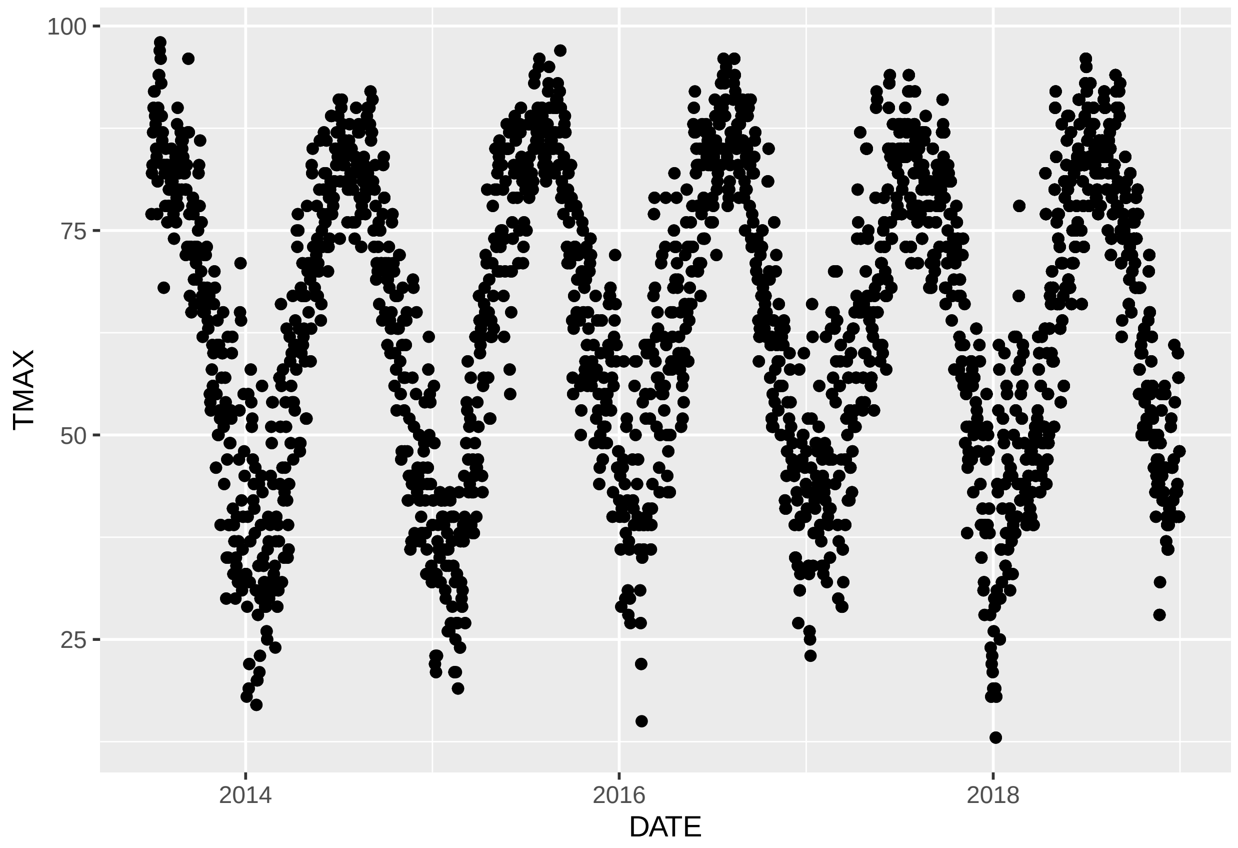
Appendix

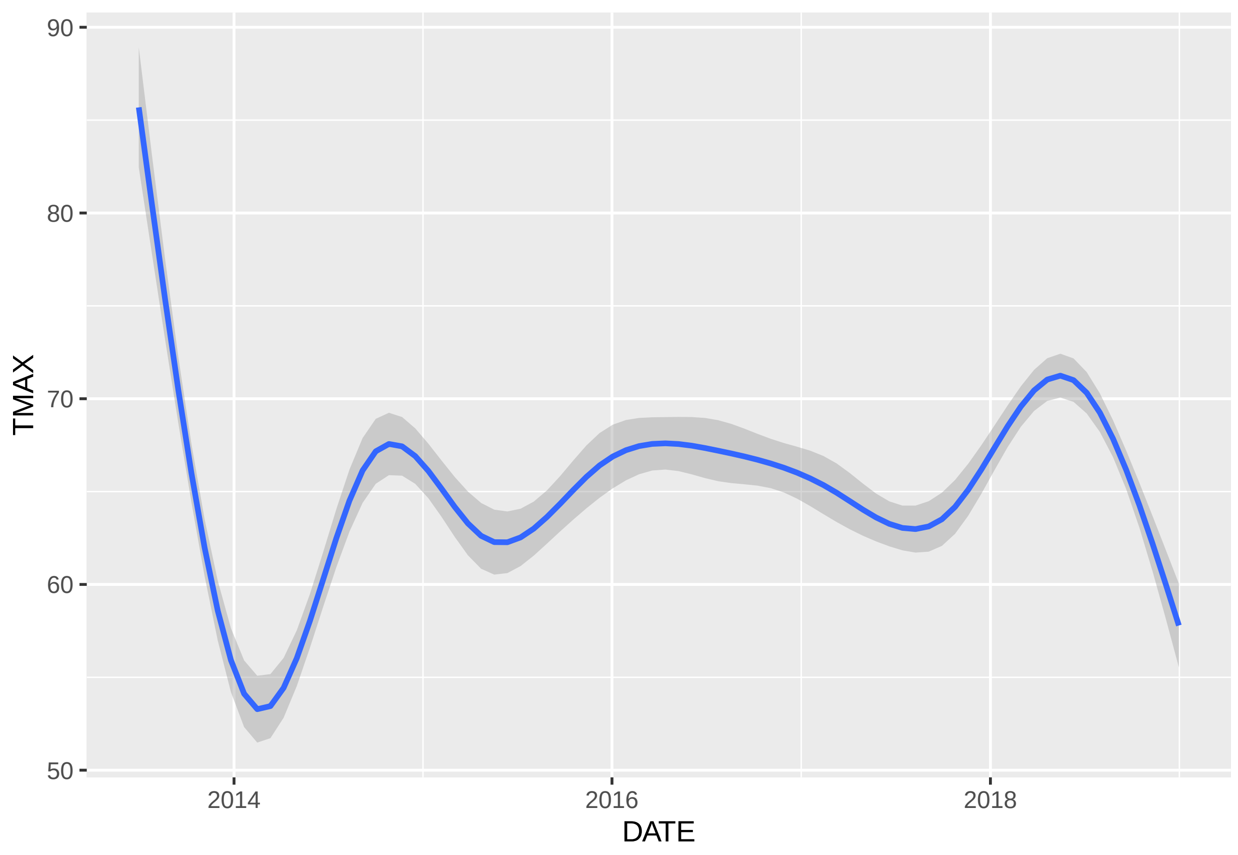
Figures and Graphs

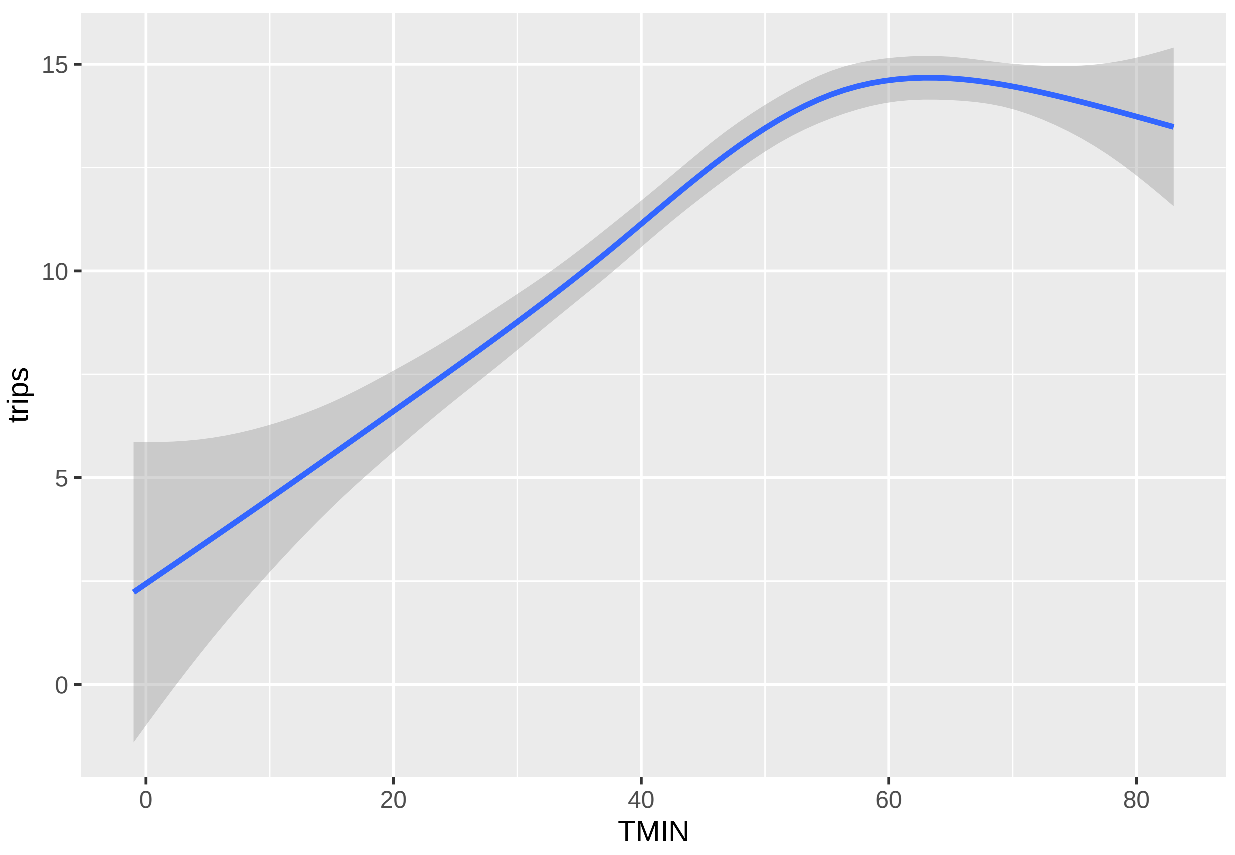
Plot the data - training dataset & weather Let's make some plots of the aggregated data, to gain an understanding of the relationships between various features

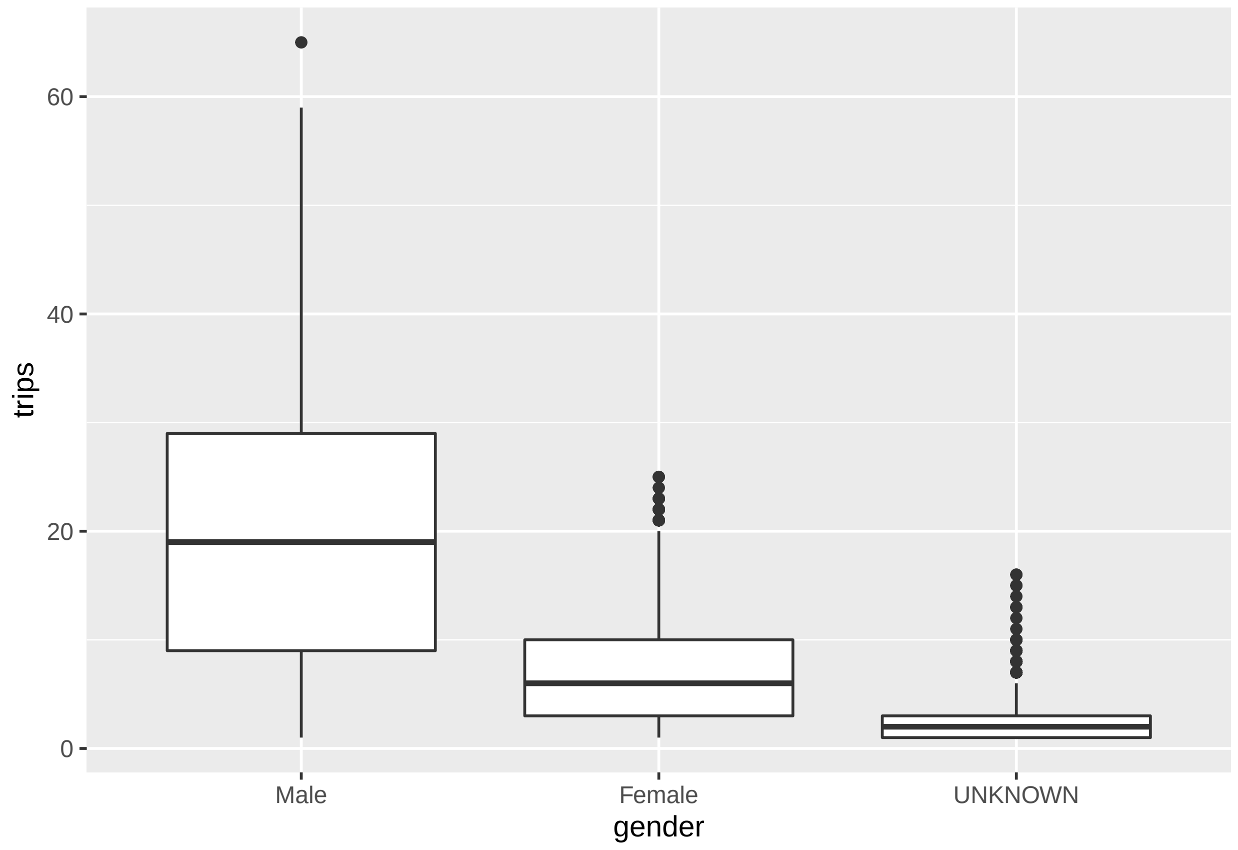


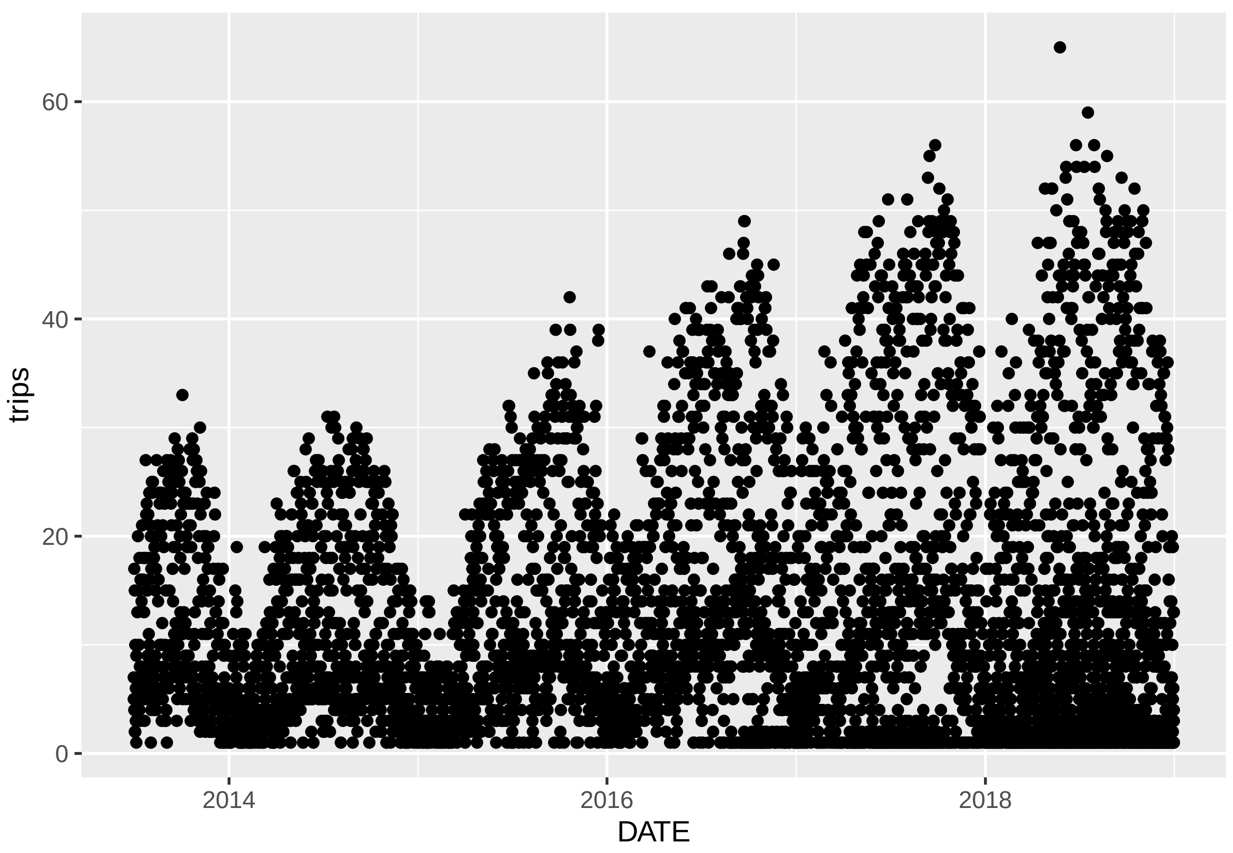

trips
0
20
40
60
0
20
40
60
sum_trip_fee


100
75
50
25
TMAX
2014
2016
2018
DATE

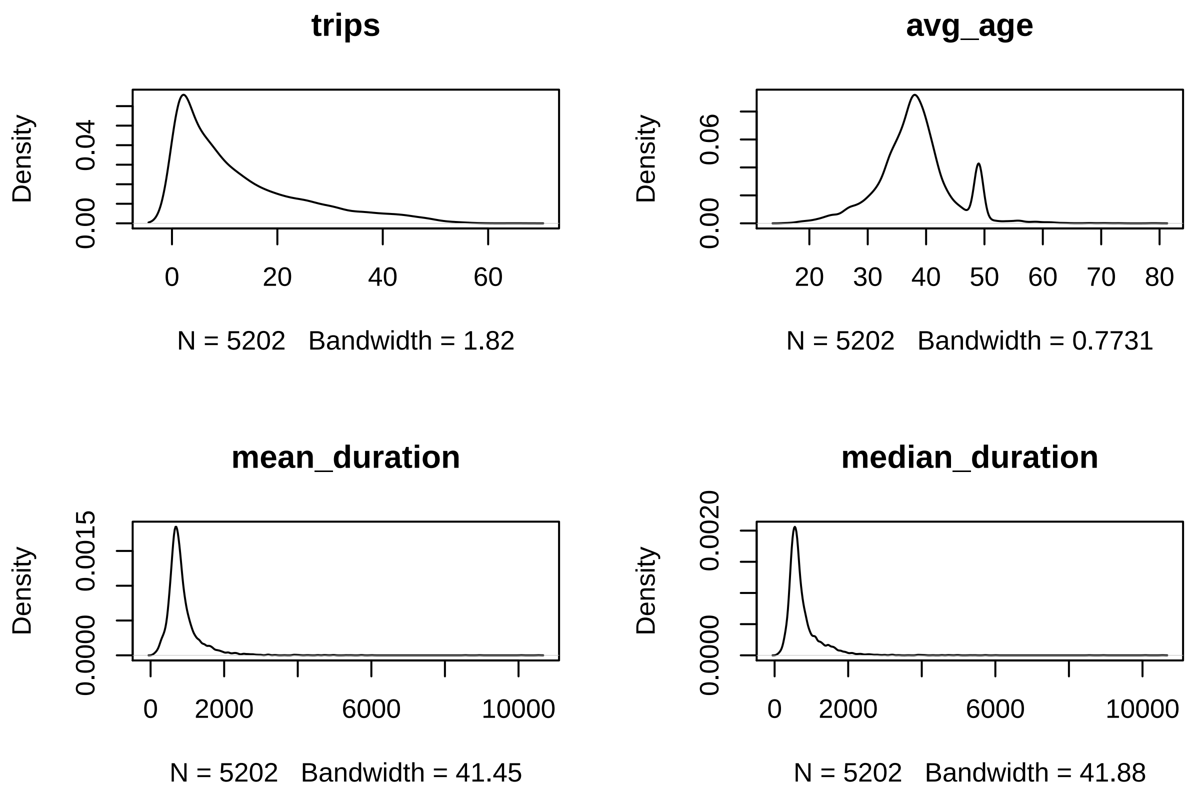

90
80
70
60
50
TMAX
2014
2016
2018
DATE

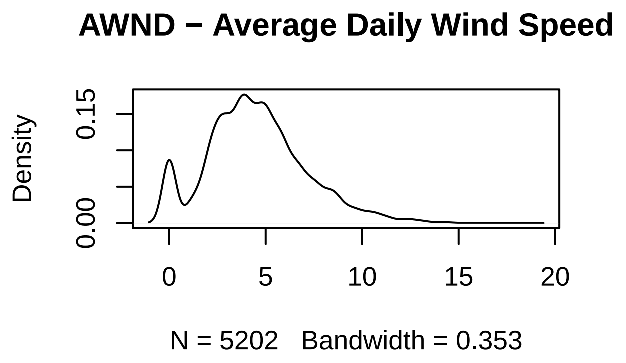

15
10
5
0
trips
0
20
40
60
80
TMIN

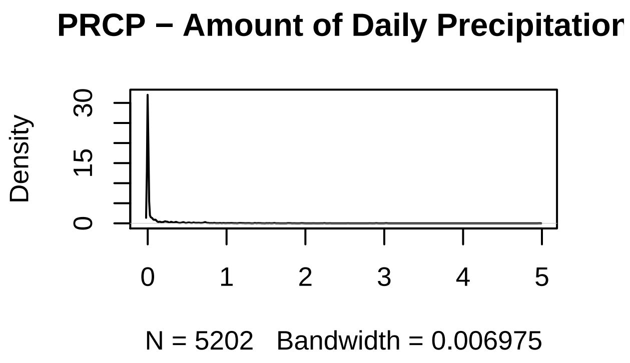

60
40
20
0
trips
Male
Female
UNKNOWN
gender

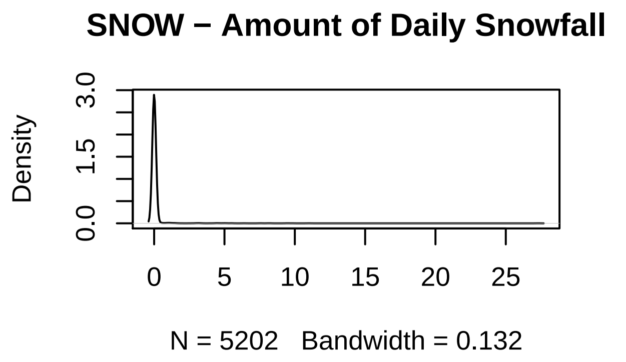


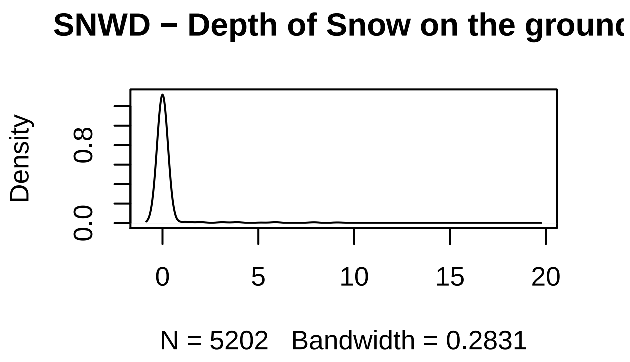
60
40
20
0
trips
2014
2016
2018
DATE

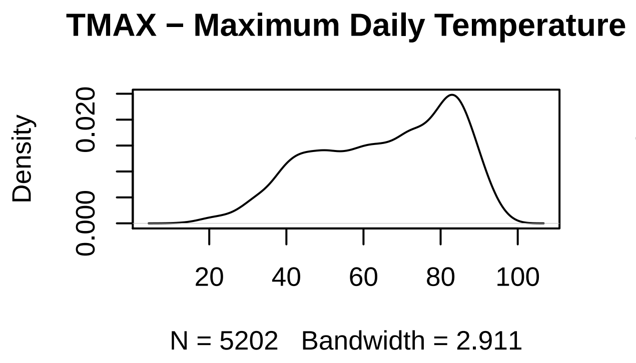

trips
avg_age
mean_duration
median_duration
Density
0.00
0.04
0.06
0.0000
0.0015
0.0020
0
20
40
60
30
50
70
80
2000
6000
10000
N = 5202 Bandwidth = 1.82
N = 5202 Bandwidth = 0.7731
N = 5202 Bandwidth = 41.45
N = 5202 Bandwidth = 41.88

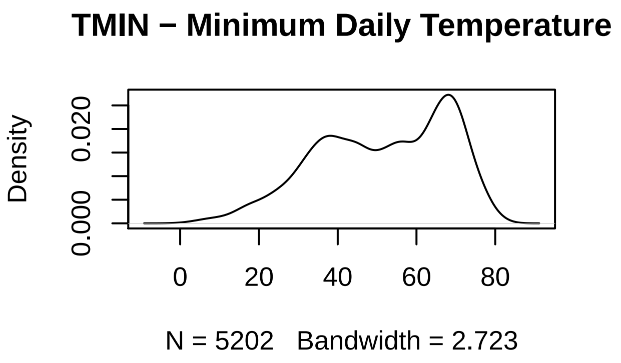

AWND − Average Daily Wind Speed
Density
0.00
0.15
0
5
10
15
20
N = 5202   Bandwidth = 0.353


PRCP − Amount of Daily Precipitatior
Density
0
15
30
0
1
2
3
4
5
N = 5202   Bandwidth = 0.006975


SNOW − Amount of Daily Snowfall
Density
0.0
1.5
3.0
0
5
10
15
20
25
N = 5202   Bandwidth = 0.132


SNWD − Depth of Snow on the grounc
Density
0.0
0.8
0
5
10
15
20
N = 5202   Bandwidth = 0.2831


TMAX − Maximum Daily Temperature
Density
0.000
0.020
20
40
60
80
100
N = 5202   Bandwidth = 2.911


TMIN − Minimum Daily Temperature
Density
0.000
0.020
0
20
40
60
80
N = 5202   Bandwidth = 2.723

Tables

Weather data - NCDC (National Climatic Data Center)

An example of the key weather data elements includes:

Feature	description	|Random date 1	|Random date 2
STATION	Station ID number	USW00094728	USW00094728
NAME	Name of station	NY CITY CENTRAL PARK, NY US	NY CITY CENTRAL PARK, NY US
LATITUDE		40.77898	40.77898
LONGITUDE		-73.96925	-73.96925
ELEVATION		42.7	42.7
DATE		1/30/2019	7/22/2019
AWND	Average Wind Speed		2.68
PRCP	Amount of precipitation	0.01	1.66
SNOW	Amount of snowfall	0.4	0
SNWD	Snow Depth	0	0
TAVG	Average temperature		
TMAX	Maximum temperature	35	90
TMIN	Minimum temperature	6	72
WDF2	Direction of fastest 2-minute wind		10
WDF5	Direction of fastest 5-second wind		340
WSF2	Fastest 2-minute Wind Speed		|14.1
WSF5	Fastest 5-second Wind Speed		|25.1
WT01	Fog, ice fog, or freezing fog?	1	1
WT02	Heavy fog or heavy freezing fog?		1
WT03	Thunder?		|1
WT06	Glaze or rime?		
WT08	Smoke or haze?		

CitiBike data

An example record from the CitiBike dataset includes the following features:

feature name	value
tripduration (seconds)	527
starttime	10/1/2019 00:00:05.6
stoptime	10/1/2019 00:08:52.9
start station id	3746
start station name	6 Ave & Broome St
start station latitude	40.72430832
start station longitude	-74.00473036
end station id	223
end station name	W 13 St & 7 Ave
end station latitude	40.73781509
end station longitude	-73.99994661
bikeid	41750
usertype	Subscriber
birth year	1993
gender	1

Summaries of LM and GLM model results

Standard Linear Models (LM)

linear model 1 - without DATE variable

```
##
## Call:
## lm(formula = trips ~ gender + user_type + avg_age + AWND + PRCP +
##     SNOW + SNWD + TMAX + TMIN + WDF2 + WDF5 + WSF2 + WSF5 + WT01 +
##     WT02, data = train.df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -24.033  -5.422  -0.882   4.341  39.838
##
## Coefficients:
##                        Estimate  Std. Error t value             Pr(>|t|)
## (Intercept)          -3.8058808   0.9707024  -3.921         0.0000894132 ***
## genderFemale        -13.4535068   0.2317749 -58.046 < 0.0000000000000002 ***
## genderUNKNOWN       -15.8676294   0.4061006 -39.073 < 0.0000000000000002 ***
## user_typeSubscriber  12.6305059   0.3107971  40.639 < 0.0000000000000002 ***
## user_typeUNKNOWN     -1.6916326   1.2627877  -1.340               0.1804
## avg_age               0.1853807   0.0188740   9.822 < 0.0000000000000002 ***
## AWND                 -0.2143032   0.0975077  -2.198               0.0280 *
## PRCP                 -3.6553776   0.3736053  -9.784 < 0.0000000000000002 ***
## SNOW                 -0.0538250   0.1526583  -0.353               0.7244
## SNWD                 -0.7086132   0.0677187 -10.464 < 0.0000000000000002 ***
## TMAX                  0.1241192   0.0227272   5.461         0.0000000495 ***
## TMIN                  0.0332403   0.0241589   1.376               0.1689
## WDF2                 -0.0019358   0.0017882  -1.083               0.2790
## WDF5                 -0.0009542   0.0018544  -0.515               0.6069
## WSF2                 -0.1735584   0.0774616  -2.241               0.0251 *
## WSF5                  0.0369182   0.0448243   0.824               0.4102
## WT01                  0.5647002   0.2580385   2.188               0.0287 *
## WT02                  0.9475085   0.9920466   0.955               0.3396
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.815 on 5184 degrees of freedom
## Multiple R-squared:  0.579,  Adjusted R-squared:  0.5777
## F-statistic: 419.5 on 17 and 5184 DF,  p-value: < 0.00000000000000022
```

```
## [1] "lm.1 RMSE: 10.19"
```

LOG linear model 1a - without DATE variable

```
##
## Call:
## lm(formula = log(trips) ~ gender + user_type + avg_age + AWND +
##     PRCP + SNOW + SNWD + TMAX + TMIN + WDF2 + WDF5 + WSF2 + WSF5 +
##     WT01 + WT02, data = train.df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.71887 -0.38814  0.07748  0.46993  2.76734
##
## Coefficients:
##                        Estimate  Std. Error t value             Pr(>|t|)
## (Intercept)         -0.36778948  0.08838219  -4.161         0.0000321491 ***
## genderFemale        -0.99539164  0.02110304 -47.168 < 0.0000000000000002 ***
## genderUNKNOWN       -1.72606984  0.03697535 -46.682 < 0.0000000000000002 ***
## user_typeSubscriber  1.55089976  0.02829800  54.806 < 0.0000000000000002 ***
## user_typeUNKNOWN    -0.59755416  0.11497648  -5.197         0.0000002101 ***
## avg_age              0.02718504  0.00171847  15.819 < 0.0000000000000002 ***
## AWND                -0.01727213  0.00887805  -1.945             0.051770 .
## PRCP                -0.31719214  0.03401666  -9.325 < 0.0000000000000002 ***
## SNOW                -0.04697429  0.01389950  -3.380             0.000731 ***
## SNWD                -0.08317943  0.00616577 -13.491 < 0.0000000000000002 ***
## TMAX                 0.01114559  0.00206930   5.386         0.0000000752 ***
## TMIN                 0.00318698  0.00219966   1.449             0.147440
## WDF2                 0.00001277  0.00016281   0.078             0.937465
## WDF5                -0.00023073  0.00016884  -1.367             0.171821
## WSF2                -0.00737900  0.00705286  -1.046             0.295498
## WSF5                 0.00125685  0.00408124   0.308             0.758126
## WT01                 0.02066406  0.02349433   0.880             0.379153
## WT02                 0.10387867  0.09032557   1.150             0.250177
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7116 on 5184 degrees of freedom
## Multiple R-squared:  0.6343, Adjusted R-squared:  0.6331
## F-statistic: 528.8 on 17 and 5184 DF,  p-value: < 0.00000000000000022
```

```
## [1] "lm.1a RMSE: 10.2"
```

linear model 2 - with DATE variable

```
##
## Call:
## lm(formula = trips ~ DATE + gender + user_type + avg_age + AWND +
##     PRCP + SNOW + SNWD + TMAX + TMIN + WDF2 + WDF5 + WSF2 + WSF5 +
##     WT01 + WT02, data = train.df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -21.742  -4.818  -0.453   4.309  34.189
##
## Coefficients:
##                         Estimate   Std. Error t value            Pr(>|t|)
## (Intercept)         -119.0963136    3.8251189 -31.135 < 0.0000000000000002 ***
## DATE                   0.0064148    0.0002070  30.994 < 0.0000000000000002 ***
## genderFemale         -13.4725090    0.2129059 -63.279 < 0.0000000000000002 ***
## genderUNKNOWN        -18.0482857    0.3796149 -47.544 < 0.0000000000000002 ***
## user_typeSubscriber   15.8213278    0.3034883  52.132 < 0.0000000000000002 ***
## user_typeUNKNOWN      -0.0037088    1.1612560  -0.003               0.9975
## avg_age                0.1815256    0.0173378  10.470 < 0.0000000000000002 ***
## AWND                  -0.0623622    0.0897032  -0.695               0.4870
## PRCP                  -3.0695015    0.3437085  -8.931 < 0.0000000000000002 ***
## SNOW                  -0.0121202    0.1402362  -0.086               0.9311
## SNWD                  -0.4291861    0.0628554  -6.828     0.00000000000959 ***
## TMAX                   0.1479779    0.0208910   7.083     0.00000000000160 ***
## TMIN                   0.0353776    0.0221922   1.594               0.1110
## WDF2                  -0.0012428    0.0016427  -0.757               0.4494
## WDF5                   0.0003457    0.0017039   0.203               0.8392
## WSF2                   0.1103067    0.0717421   1.538               0.1242
## WSF5                  -0.0734449    0.0413286  -1.777               0.0756 .
## WT01                  -1.0639968    0.2427854  -4.382     0.00001196656523 ***
## WT02                  -0.7561984    0.9129358  -0.828               0.4075
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.179 on 5183 degrees of freedom
## Multiple R-squared:  0.6449, Adjusted R-squared:  0.6436
## F-statistic: 522.9 on 18 and 5183 DF,  p-value: < 0.00000000000000022
```

```
## [1] "lm.2 RMSE: 9.14"
```

LOG linear model 2a - with DATE variable

```
##
## Call:
## lm(formula = log(trips) ~ DATE + gender + user_type + avg_age +
##     AWND + PRCP + SNOW + SNWD + TMAX + TMIN + WDF2 + WDF5 + WSF2 +
##     WSF5 + WT01 + WT02, data = train.df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.3673 -0.3354  0.1100  0.4093  2.7818
##
## Coefficients:
##                        Estimate  Std. Error t value             Pr(>|t|)
## (Intercept)         -7.18664576  0.36645779 -19.611 < 0.0000000000000002 ***
## DATE                 0.00037941  0.00001983  19.135 < 0.0000000000000002 ***
## genderFemale        -0.99651553  0.02039702 -48.856 < 0.0000000000000002 ***
## genderUNKNOWN       -1.85504483  0.03636824 -51.007 < 0.0000000000000002 ***
## user_typeSubscriber  1.73962103  0.02907508  59.832 < 0.0000000000000002 ***
## user_typeUNKNOWN    -0.49772185  0.11125178  -4.474       0.000007847345 ***
## avg_age              0.02695703  0.00166101  16.229 < 0.0000000000000002 ***
## AWND                -0.00828557  0.00859383  -0.964              0.33503
## PRCP                -0.28254047  0.03292830  -8.580 < 0.0000000000000002 ***
## SNOW                -0.04450766  0.01343504  -3.313              0.00093 ***
## SNWD                -0.06665270  0.00602173 -11.069 < 0.0000000000000002 ***
## TMAX                 0.01255672  0.00200142   6.274       0.000000000381 ***
## TMIN                 0.00331339  0.00212607   1.558              0.11919
## WDF2                 0.00005377  0.00015738   0.342              0.73264
## WDF5                -0.00015385  0.00016324  -0.942              0.34600
## WSF2                 0.00941021  0.00687311   1.369              0.17102
## WSF5                -0.00527058  0.00395940  -1.331              0.18320
## WT01                -0.07566528  0.02325956  -3.253              0.00115 **
## WT02                 0.00311288  0.08746197   0.036              0.97161
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6878 on 5183 degrees of freedom
## Multiple R-squared:  0.6584, Adjusted R-squared:  0.6572
## F-statistic:   555 on 18 and 5183 DF,  p-value: < 0.00000000000000022
```

```
## [1] "lm.2a RMSE: 6.59"
```

linear model 3 - remove insignificant variables

```
##
## Call:
## lm(formula = trips ~ DATE + gender + user_type + avg_age + PRCP +
##     SNWD + TMAX + WT01, data = train.df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -21.549  -4.867  -0.505   4.335  33.512
##
## Coefficients:
##                        Estimate   Std. Error t value             Pr(>|t|)
## (Intercept)         -121.5296046    3.5737122 -34.007 < 0.0000000000000002 ***
## DATE                   0.0065124    0.0001976  32.953 < 0.0000000000000002 ***
## genderFemale         -13.4705742    0.2130518 -63.227 < 0.0000000000000002 ***
## genderUNKNOWN        -18.0133427    0.3795325 -47.462 < 0.0000000000000002 ***
## user_typeSubscriber   15.7760458    0.3033581  52.005 < 0.0000000000000002 ***
## user_typeUNKNOWN      -0.1839985    1.1595469  -0.159                0.874
## avg_age                0.1789896    0.0173217  10.333 < 0.0000000000000002 ***
## PRCP                  -3.2228165    0.3137280 -10.273 < 0.0000000000000002 ***
## SNWD                  -0.4484124    0.0615698  -7.283    0.000000000000375 ***
## TMAX                   0.1809420    0.0059952  30.181 < 0.0000000000000002 ***
## WT01                  -0.9958636    0.2378873  -4.186    0.000028825110859 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.184 on 5191 degrees of freedom
## Multiple R-squared:  0.6438, Adjusted R-squared:  0.6431
## F-statistic: 938.1 on 10 and 5191 DF,  p-value: < 0.00000000000000022
```

```
## [1] "lm.3 RMSE: 9.13"
```

LOG linear model 3a - remove insignificant variables

```
##
## Call:
## lm(formula = log(trips) ~ DATE + gender + user_type + avg_age +
##     PRCP + SNOW + SNWD + TMAX + WT01, data = train.df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.4065 -0.3324  0.1087  0.4134  2.7816
##
## Coefficients:
##                        Estimate  Std. Error t value             Pr(>|t|)
## (Intercept)         -7.41503839  0.34236383 -21.658 < 0.0000000000000002 ***
## DATE                 0.00038882  0.00001893  20.541 < 0.0000000000000002 ***
## genderFemale        -0.99654464  0.02040631 -48.835 < 0.0000000000000002 ***
## genderUNKNOWN       -1.85238019  0.03635180 -50.957 < 0.0000000000000002 ***
## user_typeSubscriber  1.73556051  0.02905577  59.732 < 0.0000000000000002 ***
## user_typeUNKNOWN    -0.51640126  0.11106395  -4.650           0.00000341 ***
## avg_age              0.02669313  0.00165909  16.089 < 0.0000000000000002 ***
## PRCP                -0.28403246  0.03069048  -9.255 < 0.0000000000000002 ***
## SNOW                -0.04566783  0.01237214  -3.691             0.000226 ***
## SNWD                -0.06801918  0.00595703 -11.418 < 0.0000000000000002 ***
## TMAX                 0.01568354  0.00057827  27.122 < 0.0000000000000002 ***
## WT01                -0.06915150  0.02278528  -3.035             0.002418 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6881 on 5190 degrees of freedom
## Multiple R-squared:  0.6576, Adjusted R-squared:  0.6568
## F-statistic:   906 on 11 and 5190 DF,  p-value: < 0.00000000000000022
```

```
## [1] "lm.3a RMSE: 6.53"
```

Generalized linear models

GLM poisson 1 – all variables

```
##
## Call:
## glm(formula = trips ~ ., family = poisson(), data = train.df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -5.7911  -1.1374  -0.1693   0.8142   7.4511
##
## Coefficients:
##                         Estimate   Std. Error  z value             Pr(>|z|)
## (Intercept)         -7.776331126  0.150469184  -51.681 < 0.0000000000000002 ***
## DATE                 0.000455519  0.000007793   58.454 < 0.0000000000000002 ***
## AWND                -0.009013128  0.003634199   -2.480              0.01314 *
## PRCP                -0.307990680  0.016969147  -18.150 < 0.0000000000000002 ***
## SNOW                -0.076170973  0.012412207   -6.137       0.000000000842 ***
## SNWD                -0.077153166  0.004287252  -17.996 < 0.0000000000000002 ***
## TMAX                 0.011431735  0.000841600   13.583 < 0.0000000000000002 ***
## TMIN                 0.002863570  0.000889506    3.219              0.00129 **
## WDF2                -0.000117778  0.000064844   -1.816              0.06932 .
## WDF5                 0.000048083  0.000067882    0.708              0.47874
## WSF2                 0.008878418  0.002846504    3.119              0.00181 **
## WSF5                -0.004756673  0.001633757   -2.911              0.00360 **
## WT01                -0.059653331  0.009657645   -6.177       0.000000000654 ***
## WT02                 0.050347125  0.044490911    1.132              0.25779
## user_typeSubscriber  1.888032910  0.025310529   74.595 < 0.0000000000000002 ***
## user_typeUNKNOWN    -0.651803607  0.144655002   -4.506       0.000006608654 ***
## genderFemale        -1.053802864  0.009629670 -109.433 < 0.0000000000000002 ***
## genderUNKNOWN       -1.971496347  0.026734872  -73.743 < 0.0000000000000002 ***
## mean_duration        0.000038311  0.000038449    0.996              0.31905
## median_duration     -0.000204581  0.000036797   -5.560       0.000000027016 ***
## sum_trip_fee         0.021186708  0.001547228   13.693 < 0.0000000000000002 ***
## avg_age              0.014701165  0.001004577   14.634 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 55597  on 5201  degrees of freedom
## Residual deviance: 11568  on 5180  degrees of freedom
## AIC: 31592
##
## Number of Fisher Scoring iterations: 6
```

```
## [1] "glm.poisson.1 RMSE:  17.14"
```

GLM poisson 2 - remove insignificant variables

```
##
## Call:
## glm(formula = trips ~ DATE + PRCP + SNOW + SNWD + TMAX + WT01 +
##     user_type + gender + avg_age, family = poisson(), data = train.df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -5.7499  -1.1493  -0.1784   0.8259   7.8763
##
## Coefficients:
##                         Estimate   Std. Error  z value            Pr(>|z|)
## (Intercept)         -8.326639105  0.137427933  -60.589 < 0.0000000000000002 ***
## DATE                 0.000476470  0.000007347   64.851 < 0.0000000000000002 ***
## PRCP                -0.310777838  0.015857706  -19.598 < 0.0000000000000002 ***
## SNOW                -0.071896794  0.011437229   -6.286       0.000000000325 ***
## SNWD                -0.079889088  0.004276864  -18.679 < 0.0000000000000002 ***
## TMAX                 0.014281142  0.000244789   58.341 < 0.0000000000000002 ***
## WT01                -0.055974609  0.009437117   -5.931       0.000000003005 ***
## user_typeSubscriber  1.958258243  0.022122781   88.518 < 0.0000000000000002 ***
## user_typeUNKNOWN    -0.704987057  0.144520457   -4.878       0.000001071058 ***
## genderFemale        -1.088212475  0.009293401 -117.095 < 0.0000000000000002 ***
## genderUNKNOWN       -2.002859910  0.026327958  -76.073 < 0.0000000000000002 ***
## avg_age              0.014139465  0.000998582   14.160 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 55597  on 5201  degrees of freedom
## Residual deviance: 11960  on 5190  degrees of freedom
## AIC: 31965
##
## Number of Fisher Scoring iterations: 6
```

```
## [1] "glm.poisson.2 RMSE:  17.15"
```

GLM gaussian 3 – all variables

```
##
## Call:
## glm(formula = trips ~ ., family = gaussian, data = train.df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -20.851   -4.789   -0.498    4.321   34.478
##
## Coefficients:
##                        Estimate   Std. Error t value            Pr(>|t|)
## (Intercept)         -116.2486608    3.8061767 -30.542 < 0.0000000000000002 ***
## DATE                   0.0063217    0.0002053  30.794 < 0.0000000000000002 ***
## AWND                  -0.0687408    0.0887952  -0.774              0.43888
## PRCP                  -3.1047347    0.3403593  -9.122 < 0.0000000000000002 ***
## SNOW                  -0.0213124    0.1388315  -0.154              0.87800
## SNWD                  -0.4269197    0.0622812  -6.855  0.00000000000797962 ***
## TMAX                   0.1440330    0.0207024   6.957  0.00000000000389849 ***
## TMIN                   0.0339875    0.0219685   1.547              0.12190
## WDF2                  -0.0014924    0.0016263  -0.918              0.35886
## WDF5                   0.0003388    0.0016864   0.201              0.84078
## WSF2                   0.1000231    0.0710231   1.408              0.15910
## WSF5                  -0.0612938    0.0409310  -1.497              0.13433
## WT01                  -1.0036535    0.2404002  -4.175  0.00003029727588279 ***
## WT02                  -0.6820519    0.9036313  -0.755              0.45041
## user_typeSubscriber   15.6767616    0.3453467  45.394 < 0.0000000000000002 ***
## user_typeUNKNOWN       0.8309843    1.1539356   0.720              0.47148
## genderFemale         -13.1846338    0.2146763 -61.416 < 0.0000000000000002 ***
## genderUNKNOWN        -18.0601596    0.3813560 -47.358 < 0.0000000000000002 ***
## mean_duration          0.0009165    0.0007893   1.161              0.24560
## median_duration       -0.0020911    0.0007330  -2.853              0.00435 **
## sum_trip_fee           0.2702818    0.0345898   7.814  0.00000000000000667 ***
## avg_age                0.1682469    0.0172367   9.761 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 50.48266)
##
##     Null deviance: 752136  on 5201  degrees of freedom
## Residual deviance: 261500  on 5180  degrees of freedom
## AIC: 35187
##
## Number of Fisher Scoring iterations: 2
```

```
## [1] "glm.gaussian.3 RMSE:  9.05"
```

GLM gaussian 4 - remove insignificant variables

```
##
## Call:
## glm(formula = trips ~ DATE + PRCP + SNWD + TMAX + WT01 + user_type +
##     gender + avg_age, family = gaussian, data = train.df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -21.549   -4.867   -0.505    4.335   33.512
##
## Coefficients:
##                         Estimate   Std. Error t value            Pr(>|t|)
## (Intercept)         -121.5296046    3.5737122 -34.007 < 0.0000000000000002 ***
## DATE                   0.0065124    0.0001976  32.953 < 0.0000000000000002 ***
## PRCP                  -3.2228165    0.3137280 -10.273 < 0.0000000000000002 ***
## SNWD                  -0.4484124    0.0615698  -7.283    0.000000000000375 ***
## TMAX                   0.1809420    0.0059952  30.181 < 0.0000000000000002 ***
## WT01                  -0.9958636    0.2378873  -4.186    0.000028825110859 ***
## user_typeSubscriber   15.7760458    0.3033581  52.005 < 0.0000000000000002 ***
## user_typeUNKNOWN      -0.1839985    1.1595469  -0.159                0.874
## genderFemale         -13.4705742    0.2130518 -63.227 < 0.0000000000000002 ***
## genderUNKNOWN        -18.0133427    0.3795325 -47.462 < 0.0000000000000002 ***
## avg_age                0.1789896    0.0173217  10.333 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 51.61448)
##
##     Null deviance: 752136  on 5201  degrees of freedom
## Residual deviance: 267931  on 5191  degrees of freedom
## AIC: 35291
##
## Number of Fisher Scoring iterations: 2
```

```
## [1] "glm.gaussian.4 RMSE:  9.13"
```

R statistical programming code

```
knitr::opts_chunk$set(echo = TRUE, fig.pos = 'h')
mydir = "C:/Users/Michael/Dropbox/priv/CUNY/MSDS/201909-Fall/DATA621_Nasrin/20201214_FinalProject/"
#mydir = "./"
setwd(mydir)
knitr::opts_knit$set(root.dir = mydir)
options(digits=7,scipen=999,width=120)
datadir = paste0(mydir,"/Data/")


### This contains all the data -- total 90.4M rows, 17GB size, 77 monthly files
rawdatadir = "C:/temp/CitibikeData/"
### This contains 1/1000 of the rows from each of the data files -- total 90.4K rows, 17MB size
slimdatadir = "C:/temp/CitibikeDataSlim/"
#slimdatadir = "./CitibikeDataSlim/"
```

```
### Load libraries
library(tidyverse)
library(lubridate)
library(sp)
library(Hmisc)
library(corrplot)
library(forcats)
library(kableExtra)
```

```
# 1. Data Exploration

#### Weather data



# Weather data is obtained from the  NCDC (National Climatic Data Center) via https://www.ncdc.noaa.gov
# click on search tool  https://www.ncdc.noaa.gov/cdo-web/search
# select "daily summaries"
# select Search for Stations
# Enter Search Term "USW00094728" for Central Park Station:
# https://www.ncdc.noaa.gov/cdo-web/datasets/GHCND/stations/GHCND:USW00094728/detail
# "add to cart"


weatherfilenames=list.files(path="./",pattern = '.csv$', full.names = T)    # ending with .csv ; not .z
weatherfilenames
weatherfile <- "NYC_Weather_Data_2013-2019.csv"

## Perhaps we should rename the columns to more clearly reflect their meaning?
weatherspec <- cols(
  STATION = col_character(),
  NAME = col_character(),
  LATITUDE = col_double(),
  LONGITUDE = col_double(),
  ELEVATION = col_double(),
  DATE = col_date(format = "%F"),           #  readr::parse_datetime() :   "%F" = "%Y-%m-%d"
  #DATE = col_date(format = "%m/%d/%Y"), #col_date(format = "%F")
  AWND = col_double(),                      # Average Daily Wind Speed
  AWND_ATTRIBUTES = col_character(),
  PGTM = col_double(),                     # Peak Wind-Gust Time
  PGTM_ATTRIBUTES = col_character(),
  PRCP = col_double(),                     # Amount of Precipitation
  PRCP_ATTRIBUTES = col_character(),
  SNOW = col_double(),                     # Amount of Snowfall
  SNOW_ATTRIBUTES = col_character(),
  SNWD = col_double(),                     # Depth of snow on the ground
  SNWD_ATTRIBUTES = col_character(),
  TAVG = col_double(),                     # Average Temperature (not populated)
  TAVG_ATTRIBUTES = col_character(),
  TMAX = col_double(),                     # Maximum temperature for the day
  TMAX_ATTRIBUTES = col_character(),
  TMIN = col_double(),                     # Minimum temperature for the day
  TMIN_ATTRIBUTES = col_character(),
  TSUN = col_double(),                     # Daily Total Sunshine (not populated)
  TSUN_ATTRIBUTES = col_character(),
  WDF2 = col_double(),                     # Direction of fastest 2-minute wind
  WDF2_ATTRIBUTES = col_character(),
  WDF5 = col_double(),                     # Direction of fastest 5-second wind
  WDF5_ATTRIBUTES = col_character(),
  WSF2 = col_double(),                     # Fastest 2-minute wind speed
  WSF2_ATTRIBUTES = col_character(),
  WSF5 = col_double(),                     # fastest 5-second wind speed
  WSF5_ATTRIBUTES = col_character(),
  WT01 = col_double(),                    # Fog
  WT01_ATTRIBUTES = col_character(),
  WT02 = col_double(),                    # Heavy Fog
  WT02_ATTRIBUTES = col_character(),
  WT03 = col_double(),                    # Thunder
  WT03_ATTRIBUTES = col_character(),
  WT04 = col_double(),                    # Sleet
  WT04_ATTRIBUTES = col_character(),
  WT06 = col_double(),                    # Glaze
  WT06_ATTRIBUTES = col_character(),
  WT08 = col_double(),                    # Smoke or haze
  WT08_ATTRIBUTES = col_character(),
  WT13 = col_double(),                    # Mist
  WT13_ATTRIBUTES = col_character(),
  WT14 = col_double(),                    # Drizzle
  WT14_ATTRIBUTES = col_character(),
  WT16 = col_double(),                    # Rain
  WT16_ATTRIBUTES = col_character(),
  WT18 = col_double(),                    # Snow
  WT18_ATTRIBUTES = col_character(),
  WT19 = col_double(),                    # Unknown source of precipitation
  WT19_ATTRIBUTES = col_character(),
  WT22 = col_double(),                    # Ice fog
  WT22_ATTRIBUTES = col_character()
)



# load all the daily weather data
weather <- read_csv(weatherfile,col_types = weatherspec)

summary(weather)
# extract just 2019
#weather2019 <- weather[(weather$DATE>="2019-01-01" & weather$DATE<="2019-12-31"),]


# extract just one month
#weather201906 <- weather[(weather$DATE>="2019-06-01" & weather$DATE<="2019-06-30"),]
```

```
### Fix the factor items
attribute_indexes <- names(weather) %>% grep("ATTRIBUTES",x = .)
attribute_names <- names(weather)[attribute_indexes]
for (atr in attribute_names) {
  #print(paste("atr = ", atr))
  #print(summary(weather[atr]))
  weather[,atr]<-fct_explicit_na(f=pull(weather[atr]),na_level = ",,")
  #print(summary(weather[atr]))
  #print("______________________________")
}
#print(summary(weather))

### Fix the non-factor items by setting to 0
```

```r
non_attribute_indexes <- names(weather) %>% grep("ATTRIBUTES",x = .,invert = T)
non_attribute_names <- names(weather)[non_attribute_indexes]
for (atr in non_attribute_names) {
  #print(paste("atr = ", atr))
  #print(summary(weather[atr]))
  weather[is.na(weather[,atr]),atr]=0
  #print(summary(weather[atr]))
  #print("______________________________")
}
print(summary(weather))
```

```r
#### Drop ATTR columns from weather, as they are not useful (and cause factor headaches)
attr_indexes <- names(weather) %>% grep("ATTR",x = .)
attr_names <- names(weather)[attr_indexes]
# protect against manually running more than once, because it would delete everything if you do
if(length(attr_indexes)>0) {
   weather <- weather[,-attr_indexes]
   summary(weather)
   dim(weather)
   }
```

```r
#### Drop constant columns:  "STATION"   "NAME"      "LATITUDE"  "LONGITUDE" "ELEVATION"
#### Because all the weather data is from one station (Central Park) the values in these columns do not
#### Thus, they are not useful for modeling
#### They are the first 5 columns:

constant_columns=c("STATION",   "NAME",      "LATITUDE",  "LONGITUDE", "ELEVATION")
print("All same:")
summary(weather[,constant_columns])
# guard against accidentally deleting too many columns
if(all.equal(names(weather)[1:5],constant_columns)) {
  weather <- weather[,-c(1:5)]
  summary(weather)
  dim(weather)
}
```

```r
#### Function to load up a CitiBike datafile
read_CB_data_file = function(f){
  Startloadtime = Sys.time()
  print(paste("reading data file   ", f, " at ", Startloadtime))

  ### Extract the year and month from the datafile.  Needed below for inconsistent date/time formats by
  YYYYMM <- sub("^.*/","",f) %>% sub("-citibike-tripdata.csv","",.)
  print(paste("YYYYMM = ", YYYYMM))


  ### Read the datafile according to the format specifications
  datafile = read_csv(f,skip = 1,
#The column names have slight format differences across months.  So, replace all column names with thes
                  col_names=c("trip_duration",            # in seconds
                              "s_time",                   # start date/time
                              "e_time",                   # end date/time
                              "s_station_id",             # station ID for beginning of trip
                              "s_station_name",
                              "s_lat",                    # start station latitude
                              "s_long",                   # start station longitude
                              "e_station_id",             # station ID for end of trip
                              "e_station_name",
                              "e_lat",                    # latitude
                              "e_long",                   # longitude
                              "bike_id",                  # every bike has a 5-digit ID number
                              "user_type",                # Annual Subscriber or Daily Customer
                              "birth_year",               # Can infer age from this
                              "gender")                   # 1=Male,2=Female,0=unknown
#                  ,col_types = "dTTffddffddifif"    # d=decimal; T=datetime; f=factor; i=integer
### specify the data type for each of the above columns
### Note: because of changes in the format across months, we will  have to read the date/time as char f
### also we will have to read the birth_year as char for now because of missing data (either "\\N" or "
                  ,col_types = "dccffddffddifcf"    # d=decimal; c=character; f=factor; i=integer
                  )
  Endloadtime = Sys.time()
  print(paste("done reading data file ",  f, " at ", Endloadtime))
  Totalloadtime = round(Endloadtime - Startloadtime, 2)
  print(paste("Totaltime = ", Totalloadtime))

## Fix format changes on time and birth_year variables
  s_time <- pull(.data=datafile, var = "s_time")
  e_time <- pull(.data=datafile, var = "e_time")

  ### Early and recent files use format "%Y-%m-%d %H:%M:%OS"
  if (YYYYMM < "201409" | YYYYMM > "201609") timeformat="%Y-%m-%d %H:%M:%OS"

  ### time between the months uses format "%m/%d/%Y %H:%M:%OS"
  if (YYYYMM >= "201409" & YYYYMM <= "201609") timeformat="%m/%d/%Y %H:%M:%OS"
  ### except for the first 3 months of 2015, time is only HH:MM -- no seconds!
  if (YYYYMM >= "201501" & YYYYMM <= "201503") timeformat="%m/%d/%Y %H:%M"
  ### Same for June 2015, time is only HH:MM -- no seconds!
  if (YYYYMM == "201506") timeformat="%m/%d/%Y %H:%M"

  datafile[,"s_time"] <- as.POSIXct(s_time, format=timeformat)
  datafile[,"e_time"] <- as.POSIXct(e_time, format=timeformat)

#### note:  on the first Sunday of November, clocks move back 1 hour.
#### This means that the hour 1am-2am EDT is followed by the hour 1am-2am EST.
#### If a bicycle was rented during this hour "EDT",
#### but returned during the subsequent hour "EST",
#### then the trip duration could appear negative.
#### This is because the default loader will assume all times on this date are EST.
#### In this case, the below will force such start-times back an hour:

iii = which(datafile$s_time>datafile$e_time)
if(length(iii)>0) {
  print("***DAYLIGHT SAVINGS PROBLEM***")
  print(datafile[iii,])
  print("**Start times:")
  print(pull(datafile[iii,2]))
```

```r
  print(pull(datafile[iii,2]) %>% as.numeric())
  print(unclass(datafile[iii,2])$s_time)
  print("**End times:")
  print(pull(datafile[iii,3]))
  print(pull(datafile[iii,3]) %>% as.numeric())
  print(unclass(datafile[iii,3])$e_time)


  print("***CHANGING s_time backward***")
  new_s_time <- ifelse(datafile$s_time>datafile$e_time,
                      datafile$s_time-60*60,  # pushes back 1 hour from EST to EDT
                      datafile$s_time) %>% as.POSIXct(., origin= "1970-01-01")
  print("***CHANGING e_time forward***")
  new_e_time <- ifelse(datafile$s_time>datafile$e_time,
                      datafile$e_time+60*60,  # pushes forward 1 hour from EDT to EST
                      datafile$e_time) %>% as.POSIXct(., origin= "1970-01-01")
  before_diff <-  datafile[iii,3] - datafile[iii,2]
  print(paste("BEFORE difference: ", before_diff))

  datafile[,"s_time"] <- new_s_time
  datafile[,"e_time"] <- new_e_time

print("**AFTER CHANGE**")
  print(datafile[iii,])
  print("**Start times**")
  print(pull(datafile[iii,2]))
  print(pull(datafile[iii,2]) %>% as.numeric())
  print(unclass(datafile[iii,2])$s_time)
  print("**End times**")
  print(pull(datafile[iii,3]))
  print(pull(datafile[iii,3]) %>% as.numeric())
  print(unclass(datafile[iii,3])$e_time)

  after_diff <-  datafile[iii,3] - datafile[iii,2]
  print(paste("AFTER difference: ", after_diff))


  }

##
## set missing birth years to NA
  birth_year <- pull(.data=datafile, var = "birth_year")
## Fix missing birth year on early data (occurs when YYYYMM < "201409")
  birth_year[birth_year=='\\N']<-NA
## Fix missing birth year on 2017 (occurs when "201704" YYYYMM < "201712")
  birth_year[birth_year=='NULL']<-NA
## Convert the available birth_years to their integer equivalents (while retaining above NAs)
  datafile[,"birth_year"] <- as.integer(birth_year)

## There are numerous cases between 201610 and 201703 where the usertype is not specified.
## (It should be "Subscriber" or "Customer", but in such cases it is blank.)
## We will set it to "UNKNOWN"
```

```
#library(forcats)   # loaded above
  datafile$user_type<-fct_explicit_na(datafile$user_type, "UNKNOWN")

## There was a trial of DOCKLESS BIKES in the Bronx starting from August 2018:
## https://nyc.streetsblog.org/2018/08/16/a-hit-and-miss-debut-for-dockless-citi-bikes-in-the-bronx/
## https://d21xlh2maitm24.cloudfront.net/nyc/bronx-service-area-map.png?mtime=20180809110452
## https://webcache.googleusercontent.com/search?q=cache:9Xz02WSdeOYJ:https://www.citibikenyc.com/how-i

## For these trips, the latitute and longitude of the bike start and stop is given, but
## the start and end station ID and station name are set to "NULL" in the input datafiles.
## For clarity, we will change such station_id values to 0 and station_name values to "DOCKLESS" :
  levels(datafile$s_station_id)[levels(datafile$s_station_id)=="NULL"] <- 0
  levels(datafile$s_station_name)[levels(datafile$s_station_name)=="NULL"] <- "DOCKLESS"
  levels(datafile$e_station_id)[levels(datafile$e_station_id)=="NULL"] <- 0
  levels(datafile$e_station_name)[levels(datafile$e_station_name)=="NULL"] <- "DOCKLESS"


## for certain months, the datafile is not sorted on s_time (instead it is sorted on s_station_id then 
## ensure that this month's data is sorted on s_time
  datafile <- datafile[order(datafile$s_time),]
  return(datafile)
}


### November 1, 2015
#read_CB_data_file(filenames[29])

### November 6, 2016
#read_CB_data_file(filenames[41])
```

```
#### List the names of available CitiBike data files

filenames=list.files(path=slimdatadir,pattern = '.csv$', full.names = T)    # ending with .csv ; not .z
length(filenames)
t(t(filenames))
```

```
#### Load up CitiBike data file or files
#### Load all the data files, noting how much time it takes to load them

Starttime = Sys.time()
print(paste("Start time: ", Starttime))

### loads up the data for all files -- problem is too much data for my computer to handle if all are lo
print("About to load multiple datafiles:")

#suppress listing
#print(filenames)


# call read_CB_data_files to load the files specified
CB <- do.call(rbind,lapply(filenames,read_CB_data_file))
```

```
### Problem:  loading up multiple files is too much for my computer to handle !!!!



### load up just a single month of data ("filename")
#### Look at just June 2019
##filename = filenames[6]
##print(paste("Loading data for ", filename))
##CB <- read_CB_data_file(filename)

Endtime = Sys.time()
print(paste("End time: ", Endtime))
Totaltime = round(Endtime - Starttime,2)
print(paste("Totaltime for loading above file(s) = ", Totaltime))

## Save a copy of the loaded data, in case we need it during manipulations below
save_CB <- CB
```

```
#### `glimpse` the dataset
glimpse(CB)
```

```
#### `str` - structure of the dataset
str(CB)
```

```
#### Summary of the dataset
summary(CB)
```

```
### **Check for missing values**

#### Let's check whether any variables have missing values, i.e., values which are NULL or NA.
miss.cols = apply(CB, 2, function(x) any(is.na(x)))
print(paste("Number of columns with missing values = ", length(names(miss.cols[miss.cols==TRUE]))))
print(paste("Names of columns with missing values = ", paste(names(miss.cols[miss.cols==TRUE]), collaps

print(paste("Number of rows missing birth_year: ", sum(is.na(CB$birth_year))))
```

```
#### Histogram of trip_duration, log(trip_duration)

summary(CB$trip_duration)
par(mfrow=c(2,1))
hist(CB$trip_duration,col='lightgreen', breaks=100)
hist(log(CB$trip_duration),col='lightblue', breaks=100)
```

It may be easier to think of trip duration in other units (i.e., minutes, hours, or days) rather than in seconds, so lets create such variables. Also, let's confirm that the value shown (in seconds) is consistent with the difference between the start time and the end time:

```
#### Summary of trip durations before censoring/truncation:
#express trip duration in seconds, minutes, hours, days
# note: we needed to fix the November daylight savings problem to eliminate negative trip times

#### Supplied seconds
supplied_secs<-summary(CB$trip_duration)

#### Seconds
CB$trip_duration_s = as.numeric(CB$e_time - CB$s_time,"secs")
calc_secs<-summary(CB$trip_duration_s)

#### Minutes
CB$trip_duration_m = as.numeric(CB$e_time - CB$s_time,"mins")
calc_mins<-summary(CB$trip_duration_m)

#### Hours
CB$trip_duration_h = as.numeric(CB$e_time - CB$s_time,"hours")
calc_hours<-summary(CB$trip_duration_h)

#### Days
CB$trip_duration_d = as.numeric(CB$e_time - CB$s_time,"days")
calc_days <-summary(CB$trip_duration_d)

# library(kableExtra) # loaded above
rbind(supplied_secs, calc_secs, calc_mins, calc_hours, calc_days) %>%
  kable(caption="Summary of original Trip duration data") %>%
  kable_styling(c("bordered","striped"),latex_options =  "hold_position")
```

```
#### Let's assume that nobody would rent a bicycle for more than a specified timelimit,
#### and drop the records which exceed this:
total_rows=dim(CB)[1]
#print(paste("Initial number of trips: ", total_rows))
```

```r
# choose only trips that were at most 3 hrs, as longer trips may reflect an error
# remove long trips from the data set -- something may be wrong (e.g., the system failed to properly re
longtripthreshold_s = 60 * 60 *3  # 10800 seconds = 180 minutes = 3 hours
longtripthreshold_m = longtripthreshold_s / 60
longtripthreshold_h = longtripthreshold_m / 60

long_trips <- CB %>% filter(trip_duration_s > longtripthreshold_s)
num_long_trips_removed = dim(long_trips)[1]
pct_long_trips_removed = round(100*num_long_trips_removed / total_rows, 3)

CB <- CB %>% filter(trip_duration <= longtripthreshold_s)
reduced_rows = dim(CB)[1]

print(paste0("Removed ", num_long_trips_removed, " trips (", pct_long_trips_removed, "%) of longer than
print(paste0("Remaining number of trips: ", reduced_rows))

par(mfrow=c(2,1))
hist(CB$trip_duration,col='lightgreen', breaks=100)
hist(log(CB$trip_duration),col='lightblue', breaks=100)
```

```r
### Examine birth year
summary(CB$birth_year)
hist(CB$birth_year, col="lightgreen")
# Deduce age from trip date and birth year
#library(lubridate) #loaded above
CB$age <- year(CB$s_time) - CB$birth_year

par(mfrow=c(1,2))
hist(CB$age, col="lightblue",  xlab="User Age, inferred from birth year")
hist(log(CB$age), col="lightblue",  xlab="log(User Age, inferred from birth year)")
```

```r
#### Remove trips associated with very old users
#### ALSO removes trips associated with NA ages

# choose only trips where the user was born after a certain year,  as older users may reflect an error
age_threshhold = 90
aged_trips <- CB %>% filter(age > age_threshhold)
num_aged_trips_removed = dim(aged_trips)[1]
pct_aged_trips_removed = round(100*num_aged_trips_removed / total_rows, 3)

unknown_age_trips <- CB %>% filter(is.na(age))
num_unknown_age_trips_removed = dim(unknown_age_trips)[1]
pct_unknown_age_trips_removed = round(100*num_unknown_age_trips_removed / total_rows, 3)

print(paste0("Removed ", num_aged_trips_removed, " trips (", pct_aged_trips_removed, "%) of users older

print(paste0("Removed ", num_unknown_age_trips_removed, " trips (", pct_unknown_age_trips_removed, "%)

CB <- CB %>% filter(age <= age_threshhold)
reduced_rows = dim(CB)[1]
print(paste0("Remaining number of trips: ", reduced_rows))

par(mfrow=c(1,2))
```

```r
hist(CB$age, col="lightgreen",  xlab="User Age, inferred from birth year")
hist(log(CB$age), col="lightgreen",  xlab="log(User Age, inferred from birth year)")
```

```r
# Compute the distance between start and end stations
s_lat_long <- CB %>% select(c(s_lat,s_long)) %>%  as.matrix
e_lat_long <- CB %>% select(c(e_lat,e_long)) %>%  as.matrix
#library(sp) # loaded above
CB$distance_km <- spDists(s_lat_long, e_lat_long, longlat=T, diagonal = TRUE)
summary(CB$distance_km)
maxdistance = summary(CB$distance_km)["Max."]
hist(CB$distance_km, breaks=30, col="orange")
```

```r
### long distances?
long_distances <- CB %>% filter(distance_km>50)

#### Delete unusually long distances

if (dim(long_distances)[1]>0) {
print(paste("Dropping ", dim(long_distances)[1], " trips because of unreasonably long distance travelle
    print(t(long_distances))

### These items have a station where latitude and longitude are zero.
### Drop them:

  CB <- CB %>% filter(distance_km<50)

  summary(CB$distance_km)
  hist(CB$distance_km, breaks=30, col="lightgreen", main="Histogram of distance_km after dropping probl
} else {print("No unusually long distances found in this subset of the data.")}
```

```r
# There is a time-based usage fee for rides longer than an initial period.
# For user_type=Subscriber, the fee is $2.50 per 15 minutes following an initial free 45 minutes.
# For user_type=Customer, the fee is $4.00 per 15 minutes following an initial free 30 minutes.
# There are some cases where the user type is not specified (we have relabeled as "UNKNOWN")

CB$trip_fee <- 0
CB$trip_fee[CB$user_type=="Subscriber"] <- 2.50 * (ceiling(
  CB$trip_duration_m[CB$user_type=="Subscriber"]  / 15)-3)  # first 45 minutes are free
CB$trip_fee[CB$user_type=="Customer"]   <- 4.00 * (ceiling(
  CB$trip_duration_m[CB$user_type=="Customer"]  / 15)-2)  # first 30 minutes are free
CB$trip_fee[CB$user_type=="UNKNOWN"] <- 0   # we don't know the fee structure for "UNKNOWN" so assume z
CB$trip_fee[CB$trip_fee<0] <- 0   # fee is non-negative

#summary(CB$trip_fee)
#print("Table of trip_fee:")
table(CB$trip_fee,dnn="Fee Amount")  %>%
  kable(caption="Table of trip_fees:") %>%
  kable_styling(c("bordered","striped"),
                full_width = F,
                latex_options =  "hold_position")

hist(CB$trip_fee,breaks=40,col="yellow", main="Histogram of trip_fee (most trips incur no fee)")
hist(CB$trip_fee[CB$trip_fee>0],breaks=32,col="lightgreen",main = "Histogram of trip_fee excluding zero
#### Summary of trip durations AFTER censoring/truncation:

#express trip duration in seconds, minutes, hours, days
# note: we needed to fix the November daylight savings problem to eliminate negative trip times

#### Supplied seconds
#print("Supplied Seconds:")
supplied_secs<-summary(CB$trip_duration)

#### Seconds
CB$trip_duration_s = as.numeric(CB$e_time - CB$s_time,"secs")
calc_secs<-summary(CB$trip_duration_s)

#### Minutes
CB$trip_duration_m = as.numeric(CB$e_time - CB$s_time,"mins")
calc_mins<-summary(CB$trip_duration_m)

#### Hours
CB$trip_duration_h = as.numeric(CB$e_time - CB$s_time,"hours")
calc_hours<-summary(CB$trip_duration_h)

#### Days
CB$trip_duration_d = as.numeric(CB$e_time - CB$s_time,"days")
calc_days <-summary(CB$trip_duration_d)

# library(kableExtra) # loaded above
rbind(supplied_secs, calc_secs, calc_mins, calc_hours, calc_days) %>%
  kable(caption = "Summary of trip durations - AFTER truncations:") %>%
  kable_styling(c("bordered","striped"),latex_options =  "hold_position")
```

```r
#### Make a smaller dataset, numeric, without multicollinearities, for correlation calculations
# extract selected fields
CBlite  <- select(CB, c(trip_duration, trip_fee, distance_km,
                        s_station_id, s_lat, s_long,
                        e_station_id, e_lat, e_long,
                        user_type, gender, age))

#make numeric variables
CBlite$user_type <- as.integer(CBlite$user_type)
CBlite$gender <- as.integer(CBlite$gender)

# function to revert factor back to its numeric levels
as.numeric.factor <- function(x) {as.numeric(levels(x))[x]}

CBlite$s_station_id <- as.numeric.factor(CBlite$s_station_id)
CBlite$e_station_id <- as.numeric.factor(CBlite$e_station_id)
```

```r
#### compute correlations
#library(Hmisc) #loaded above
#library(corrplot) # loaded above
res2<-rcorr(as.matrix(CBlite))
respearson=rcorr(as.matrix(CBlite),type = "pearson")
resspearman=rcorr(as.matrix(CBlite),type = "spearman")
```

```r
res3 <- cor(as.matrix(CBlite))
```

```r
#### Pearson rank correlation
  corrplot::corrplot(corr = respearson$r, type = "upper", outline = T, order="original",
           p.mat = respearson$P, sig.level = 0.05, insig = "blank", addCoef.col = "black",
           title = "\nRank Correlation (Pearson)",
           number.cex = 1.1, number.font = 2, number.digits = 2 )
```

```r
#### Spearman rank correlation
  corrplot::corrplot(corr = resspearman$r, type = "upper", outline = T,  order="hclust",
           p.mat = resspearman$P, sig.level = 0.05, insig = "blank", addCoef.col = "black",
           title = "\nRank Correlation (Spearman)",
           number.cex = 0.9, number.font = 1, number.digits = 2)
```

```r
#### Aggregate CitiBike data and join to daily weather data
#### Make train and test dataframes

summary(CB)

##CB$user_type[is.na(CB$user_type)] <- "UNKNOWN"    ## should  not be necessary to do this
CB$gender <- recode_factor(CB$gender, '1' = "Male", '2' = "Female", '0' = "UNKNOWN")

# make the training data set
train <- CB %>%
              mutate(start_date = as.Date(s_time, format="%Y-%m-%d"),
#                     user_type = as.character(user_type),
                     train = 1) %>%
              filter(start_date < '2019-01-01') %>%
              group_by(start_date, user_type, train, gender) %>%
              summarise(
                mean_duration = mean(trip_duration),
                median_duration = median(trip_duration),
                sum_distance_km = sum(distance_km),
                sum_trip_fee = sum(trip_fee),
                avg_age = mean(age),
                trips = n()
              ) %>%
              ungroup()

train_rows = dim(train)[1]
#summary(train)

# make the test data set
test <- CB %>%
              mutate(start_date = as.Date(s_time, format="%Y-%m-%d"),
#                     user_type = as.character(user_type),
                     train = 0) %>%
              filter(start_date >= '2019-01-01') %>%
              group_by(start_date, user_type, train, gender) %>%
              summarise(
                mean_duration = mean(trip_duration),
                median_duration = median(trip_duration),
                sum_distance_km = sum(distance_km),
                sum_trip_fee = sum(trip_fee),
                avg_age = mean(age),
                trips = n()
              ) %>%
              ungroup()
test_rows = dim(test)[1]


# Join train with weather data (there should be no rows with missing values)
train_weather <- weather %>% inner_join(train, by = c("DATE" = "start_date" ))
#dim(train_weather)

# Join test with weather data (there should be no rows with missing values)
test_weather <- weather %>% inner_join(test, by = c("DATE" = "start_date" ))
```

```r
#dim(test_weather)
```

```r
library(ggplot2)
# user_type boxplot
ggplot(data = train_weather,
       aes(x = user_type,
           y = trips)) +
  geom_boxplot()
# trip_fee scatterplot
ggplot(data = train_weather,
       aes(x = sum_trip_fee,
           y = trips)) +
  geom_point()

# weather TMAX over the year
ggplot(data = train_weather,
       aes(x = DATE,
           y = TMAX)) +
  geom_point()

# weather TMAX and smooth over the year
ggplot(data = train_weather,
       aes(x = DATE,
           y = TMAX,
           color = year(DATE))) +
  geom_smooth()

# the number of trips with min. tempareture
ggplot(data = train_weather,
       aes(x = TMIN,
           y = trips)) +
  geom_smooth()
# the number of trips with max. tempareture
ggplot(data = train_weather,
       aes(x = gender,
           y = trips)) +
  geom_boxplot()

# the number of trips by date
ggplot(data = train_weather,
       aes(x = DATE,
           y = trips)) +
  geom_point()

par(mfrow=c(2,2))
plot(density(train_weather$trips), main = "trips")
plot(density(train_weather$avg_age), main = "avg_age")
plot(density(train_weather$mean_duration), main = "mean_duration")
plot(density(train_weather$median_duration), main = "median_duration")
plot(density(train_weather$AWND), main = "AWND - Average Daily Wind Speed")
plot(density(train_weather$PRCP), main = "PRCP - Amount of Daily Precipitation")
plot(density(train_weather$SNOW), main = "SNOW - Amount of Daily Snowfall")
plot(density(train_weather$SNWD), main = "SNWD - Depth of Snow on the ground")
plot(density(train_weather$TMAX), main = "TMAX - Maximum Daily Temperature")
plot(density(train_weather$TMIN), main = "TMIN - Minimum Daily Temperature")
```

```r
# 5. Variable Selection
# We use the "Boruta" package to select those variables which the Boruta algorithm deems "important."
library(Boruta)
# library(kableExtra) # loaded above
set.seed(777)
num_cols <- length(names(train_weather))

#### suppressed because it is very time consuming to run each time.
#### The result is the list of "important" variables, which was copied and hard-coded
#borutaOutput <- Boruta(trips ~ ., train_weather)
print(borutaOutput)
plot(borutaOutput,  cex.axis=0.75, las=2, main="Boruta algorithm for Feature Selection", xlab="")

# Here is the Confirmed/Tentative/Rejected Borua decision:
BorutaFinal       <- borutaOutput$finalDecision
BorutaFinal

# Here is the alphabetized list of Boruta decision:
BorutaFinalAlpha  <- BorutaFinal[order(names(BorutaFinal))] %>% t %>% t
BorutaFinalAlpha

# Extract the numerical median results from the Boruta algorithm
BorutaMedian      <- apply(X = borutaOutput$ImpHistory, MARGIN = 2, FUN = median)

# drop the three "shadow" variables from the list (shadowMax,shadowMean,shadowMin)
BorutaMedian      <- BorutaMedian[BorutaMedian %>% names %>% grep("shadow",.,invert=T)]

# alphabetize the list
BorutaMedianAlpha <- BorutaMedian[order(names(BorutaMedian))]
BorutaMedianAlphaNum <- as.numeric(BorutaMedianAlpha)

BorutaMedianAlpha <- BorutaMedian[order(names(BorutaMedian))] %>% t %>% t

BorutaJoinedAlpha <- cbind(BorutaFinalAlpha,BorutaMedianAlpha)

BorutaFinalAlphaResults <- as.character(BorutaFinalAlpha)
BorutaFinalAlphaNames <- BorutaFinal[names(BorutaFinal) %>% order] %>% names()

# Here's the alphabetical list of the Boruta results:
BorutaByAlpha <- cbind(BorutaFinalAlphaNames,BorutaFinalAlphaResults,BorutaMedianAlphaNum)
BorutaByAlpha %>% kable(caption="Alphabetical Boruta results") %>%
  kable_styling(c("bordered","striped"),
                full_width = F,
                latex_options =  "hold_position")

# Here's the numerical list of the Boruta results (based upon median)
BorutaByNum <- BorutaByAlpha[order(BorutaMedianAlphaNum),]
BorutaByNum %>% kable(caption="Boruta results sorted by median value") %>%
  kable_styling(c("bordered","striped"),
                full_width = F,
                latex_options =  "hold_position")

plot(borutaOutput)
plot(borutaOutput, sort=FALSE)
lz<-lapply(1:ncol(borutaOutput$ImpHistory),function(i) borutaOutput$ImpHistory[is.finite(borutaOutput$Im
names(lz) <- colnames(borutaOutput$ImpHistory)
Labels <- sort(sapply(lz,median))

plot(borutaOutput, xlab = "", xaxt = "n")
axis(side = 1,las=2,labels = names(Labels), at = 1:ncol(borutaOutput$ImpHistory), cex.axis = 0.7)

final.boruta <- TentativeRoughFix(borutaOutput)
important <- getSelectedAttributes(final.boruta, withTentative = F)
```

```
# load important variables
### Because Boruta can take a very long time to run, saving the results here:

important <-  c(
"DATE"           ,
"AWND"           ,
"PRCP"           ,
"SNOW"           ,
"SNWD"           ,
"TMAX"           ,
"TMIN"           ,
"WDF2"           ,
"WDF5"           ,
"WSF2"           ,
"WSF5"           ,
"WT01"           ,
"WT02"           ,
"user_type"      ,
"gender"         ,
"mean_duration"  ,
"median_duration",
##"sum_distance_km",  ## This has 0.97 correlation with number of trips -- not fair to use for predicti
"sum_trip_fee"   ,
"avg_age"
)

impF <- important
train.df <- train_weather %>% select(c("trips", impF))
# we will include "trips" for now but remove it below
test.df <- test_weather %>% select(c("trips", impF))

### glm model
summary(train.df)
```

```
### Make numeric version of training dataset
train.df.numerics <- train.df %>% select(-c(user_type,gender)) %>% mutate(DATE=as.numeric(DATE))


#### Correlation on daily aggregated datasets


#### compute correlations
```

```r
#library(Hmisc) #loaded above
#library(corrplot) # loaded above
res2x<-rcorr(as.matrix(train.df.numerics))
respearsonx=rcorr(as.matrix(train.df.numerics),type = "pearson")
resspearmanx=rcorr(as.matrix(train.df.numerics),type = "spearman")
res3x <- cor(as.matrix(train.df.numerics))
```

```r
#### Pearson rank correlation
  corrplot::corrplot(corr = respearsonx$r, type = "upper", outline = T, order="original",
           p.mat = respearsonx$P, sig.level = 0.05, insig = "blank", addCoef.col = "black",
           title = "\nRank Correlation (Pearson)",
           number.cex = 0.9, number.font = 2, number.digits = 2 )
```

```r
#### Spearman rank correlation of daily aggregated data
  corrplot::corrplot(corr = resspearmanx$r, type = "upper", outline = T,  order="hclust",
           p.mat = resspearmanx$P, sig.level = 0.05, insig = "blank", addCoef.col = "black",
           title = "\nRank Correlation (Spearman)",
           number.cex = 0.9, number.font = 1, number.digits = 2)
```

```
##### set up test variable trips
###### only run this once...
if (!exists("test.values")) {
  # pull the TARGET variable out from the test dataframe, and save it for RMSE calculations
  test.values <- test.df$trips
  # delete the value from the test dataframe
  test.df$trips <- NULL
}
```

```
#### Standard linear models
# use Sachid's RMSE function
model.fit.evaluate.rmse <- function(model, test.data, test.values) {
output = predict(model, test.data)
rmse = round(sqrt(
        sum((output - test.values)^2)
        /
          length(test.values)
        )
        , digits =2)
return(rmse)
}

### evaluation where predicted variable in the model is log(y) ~ dependent variables
model.fit.evaluate.rmse.log <- function(model, test.data, test.values) {
output = exp(predict(model, test.data))
rmse = round(sqrt(sum((output - test.values)^2)/length(test.values)), digits =2)
return(rmse)
}





# linear model 1 without DATE variable
lm.1 <- lm(trips ~  gender+user_type+avg_age+AWND+PRCP+SNOW+SNWD
                    +TMAX+TMIN+WDF2+WDF5+WSF2+WSF5+WT01+WT02,
                    data=train.df)

#lm.1 <- lm(trips ~ gender+user_type+avg_age+TMAX+TMIN+AWND+WSF5+WSF2+WDF5,
#           data = train.df)
summary(lm.1)
(rmse.nb1 = model.fit.evaluate.rmse(lm.1, test.df, test.values))
output1=predict(lm.1,test.df)
summary(output1)
summary(test.values)
plot(output1~test.values)
abline(h=0,col="red")
abline(v=0,col="red")
# Compute RMSE
print(paste("lm.1 RMSE:", rmse.nb1))
plot(lm.1)
```

```r
# LOG linear model 1 without DATE variable
lm.1a <- lm(log(trips) ~  gender+user_type+avg_age+AWND+PRCP+SNOW+SNWD
                    +TMAX+TMIN+WDF2+WDF5+WSF2+WSF5+WT01+WT02,
                    data=train.df)
summary(lm.1a)
(rmse.nb1a = model.fit.evaluate.rmse.log(lm.1a, test.df, test.values))
output1a=predict(lm.1a,test.df)
expoutput1a = exp(output1a)
summary(expoutput1a)
summary(test.values)
plot(expoutput1a~test.values)
abline(h=0,col="red")
abline(v=0,col="red")
# Compute RMSE
print(paste("lm.1a RMSE:", rmse.nb1a))
plot(lm.1a)

###

# linear model 1 with DATE variable
lm.2 <- lm(trips ~  DATE+gender+user_type+avg_age+AWND+PRCP+SNOW+SNWD
                    +TMAX+TMIN+WDF2+WDF5+WSF2+WSF5+WT01+WT02,
                    data=train.df)
summary(lm.2)
(rmse.nb2 = model.fit.evaluate.rmse(lm.2, test.df, test.values))
output2=predict(lm.2,test.df)
summary(output2)
summary(test.values)
plot(output2~test.values)
abline(h=0,col="red")
abline(v=0,col="red")
# Compute RMSE
print(paste("lm.2 RMSE:", rmse.nb2))
plot(lm.2)


# LOG linear model 2
lm.2a <- lm(log(trips) ~  DATE+gender+user_type+avg_age+AWND+PRCP+SNOW+SNWD
                    +TMAX+TMIN+WDF2+WDF5+WSF2+WSF5+WT01+WT02,
                    data=train.df)
summary(lm.2a)
(rmse.nb2a = model.fit.evaluate.rmse.log(lm.2a, test.df, test.values))
output2a=predict(lm.2a,test.df)
expoutput2a = exp(output2a)
summary(expoutput2a)
summary(test.values)
plot(expoutput2a~test.values)
abline(h=0,col="red")
abline(v=0,col="red")
# Compute RMSE
print(paste("lm.2a RMSE:", rmse.nb2a))
plot(lm.2a)
```

```r
###

# linear model 3:
# linear model 1 with DATE variable and remove insignificant variables
lm.3 <- lm(trips ~  DATE+gender+user_type+avg_age+    PRCP+    SNWD
                   +TMAX+    WT01,
                   data=train.df)
summary(lm.3)
(rmse.nb3 = model.fit.evaluate.rmse(lm.3, test.df, test.values))
output3=predict(lm.3,test.df)
summary(output3)
summary(test.values)
plot(output3~test.values)
abline(h=0,col="red")
abline(v=0,col="red")
# Compute RMSE
print(paste("lm.3 RMSE:", rmse.nb3))
plot(lm.3)


# LOG linear model 3a
lm.3a <- lm(log(trips) ~  DATE+gender+user_type+avg_age+    PRCP+SNOW+SNWD
                   +TMAX+    WT01,
                   data=train.df)
summary(lm.3a)
(rmse.nb3a = model.fit.evaluate.rmse.log(lm.3a, test.df, test.values))
output3a=predict(lm.3a,test.df)
expoutput3a = exp(output3a)
summary(expoutput3a)
summary(test.values)
plot(expoutput3a~test.values)
abline(h=0,col="red")
abline(v=0,col="red")
# Compute RMSE
print(paste("lm.3a RMSE:", rmse.nb3a))
plot(lm.3a)
```

```
## Generalized linear models

# glm model 1

glm.poisson.1 <- glm(trips ~ ., data = train.df, family = poisson())
summary(glm.poisson.1)
(rmse.poisson.1 = model.fit.evaluate.rmse(glm.poisson.1, test.df, test.values) )
print(paste("glm.poisson.1 RMSE: ", rmse.poisson.1))

output11=predict(glm.poisson.1,test.df)
summary(output11)
summary(test.values)
plot(output11~test.values, main="glm.poisson.1")
abline(h=0,col="red")
abline(v=0,col="red")

plot(glm.poisson.1)


# glm model 2
# Keep just the significant variables from above
#glm.poisson.2 <- glm(trips ~ DATE+gender+user_type+avg_age+TMAX+WSF5, data = train.df, family = poisso
glm.poisson.2 <- glm(trips ~ DATE+PRCP+SNOW+SNWD+TMAX+WT01
                     +user_type+gender+ avg_age,
                     data = train.df, family = poisson())
summary(glm.poisson.2)
(rmse.poisson.2 = model.fit.evaluate.rmse(glm.poisson.2, test.df, test.values) )
print(paste("glm.poisson.2 RMSE: ", rmse.poisson.2))

output12=predict(glm.poisson.2,test.df)
summary(output12)
summary(test.values)
plot(output12~test.values, main="glm.poisson.2")
abline(h=0,col="red")
abline(v=0,col="red")

plot(glm.poisson.2)







# glm model 3
glm.gaussian.3 <- glm(trips ~ ., data = train.df, family = gaussian)
summary(glm.gaussian.3)
(rmse.gaussian.3 = model.fit.evaluate.rmse(glm.gaussian.3, test.df, test.values) )
print(paste("glm.gaussian.3 RMSE: ", rmse.gaussian.3))

output13=predict(glm.gaussian.3,test.df)
summary(output13)
summary(test.values)
plot(output13~test.values, main="glm.gaussian.3")
```

```r
abline(h=0,col="red")
abline(v=0,col="red")

plot(glm.gaussian.3)







# glm model 4
#glm.gaussian.4 <- glm(trips ~ DATE+gender+user_type+avg_age+TMAX+WSF5, data = train.df, family = gauss
glm.gaussian.4 <- glm(trips ~ DATE+PRCP+SNWD+user_type+gender+avg_age+sum_distance_km,
                      data = train.df, family = gaussian)
summary(glm.gaussian.4)
(rmse.gaussian.4 = model.fit.evaluate.rmse(glm.gaussian.4, test.df, test.values) )
print(paste("glm.gaussian.4 RMSE: ", rmse.gaussian.4))

output14=predict(glm.gaussian.4,test.df)
summary(output14)
summary(test.values)
plot(output14~test.values, main="glm.gaussian.4")
abline(h=0,col="red")
abline(v=0,col="red")

plot(glm.gaussian.4)
```