New York Bike Share Predictive Model and Live Dashboard

City University of New York
School of Professional Studies
Capstone Project, MS in Data Science

Brian Weinfeld
May 2019

Table of Contents

Abstract

Over the past 20 years bike share systems have exploded in popularity in major cities across North America, Europe and Asia. It is estimated that there are more than 700 cities operating over 800,000 bikes in 50 countries around the world.[1] As these programs grow so does their need for up to the minute information on bike operations and strong models that can predict usage rates.

Currently, the most popular bike share systems utilize automated stations. These semi-permanent structures house bikes built and designed specifically for use with the system. Riders use their smartphones to unlock a bike at any station and are permitted to ride it anywhere in the designated operating area. Once the ride is finished, they can dock the bike at any station that has an empty space. They do not need to dock at the station where the bike was initially rented. The rider is automatically charged based on the amount of time the bike was in use. Most bike share companies offer a variety of different rental price points to appeal to everyone from tourists to daily commuters. For the purpose of my capstone project, I will imagine a scenario where I have been hired as a consultant by the Citi Bike bike share program which currently has exclusive rights to operate in New York City and Jersey City, New Jersey.[2]

“Business” Statement

Although the specially built bikes have been designed to ensure stable long term performance, there is a need for employees to be on the ground every day. Broken bikes will need to be fixed or removed from circulation, tires will need to be inflated and, most commonly, bikes will need to be moved from full stations to empty stations to avoid clumping. The company wants to avoid a situation where a person cannot rent a bike because a station is empty or cannot park a bike because a station is full.[3]

The employees who perform this work are often independent contractors who are paid only for the hours they work.[4] As the number of rides taken on any given day can vary widely, it is important for Citi Bike to have an accurate estimate as to the number of rides on a given day and hour. This will allow them to call in enough independent contractors to fulfill the needs for the day without paying workers who are not needed.

Citi Bike requires hourly predictions of the number of bikes that will be rented 24 hours in advance of the beginning of each day. This will provide them sufficient time to call in the appropriate number of workers. For example, at 1 AM on May 5th, 2019, I will need to provide 24 predictions, one for each hour, for the number of bikes that will be rented during the entirety of May 6th, 2019.

Deliverables

First, I will need to collect, organize and clean all relevant available data. From there I will perform an exploratory data analysis (EDA) to draw general conclusions about the raw data I have collected. I will use this knowledge to create a machine learning algorithm that can make predictions representing the number of bikes rented each hour during the next calendar day. The predictions must be made at least 24 hours in advance. Finally, I will create a dashboard that will provide real time updates on bike rides for the current day and provide a live side-by-side comparison of the predicted ride count to the actual ride count. This will need to be simulated as I do not have access to real time data.

Exploratory Data Analysis

Data Collection

I began by downloading all the available rider data from the Citi Bike website.[5] Citi Bike generously shares monthly comma separated value (csv) files with data on every trip taken on its system. This data includes the date, time, location and length of every trip along with a number of nominal statistics on the identity of the rider. These files are not provided in real time but are instead supplied in batch at the end of each month. In total, there are approximately 52 million rides in the provided files. The amount of data was far too much for me to effectively process on my computer and as such I decided to use Google Cloud Platform (GCP) to do the bulk of my data processing. I uploaded all the data to GCP and stored it in a table database.

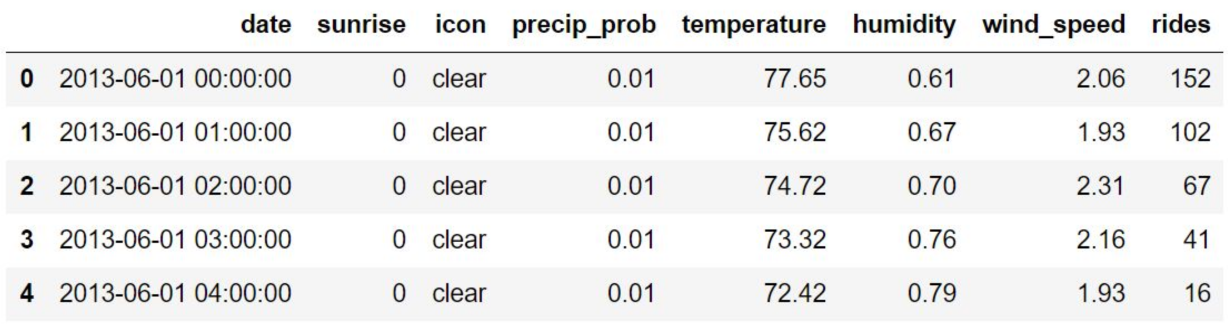
The rider files have been stripped of some sensitive or potentially identifying information. While this is to be expected, there were several other inconsistencies with the data that needed to be addressed. First and foremost, there were a number of hours and days that had missing information. I was unable to determine if there was a problem with the original system resulting in lost data or if the data was mishandled or withheld by the company.

In addition, a number of baffling format changes were applied to the data. For example, the date and time was changed from the standard structured query language (SQL) format to various other formats before being changed back again. All of this needed to be addressed with my ingestion code. I was once again unable to determine why certain files were formatted differently, although I suspect it may have been to dissuade bulk scraping of all the provided data.

Next, I wrote a small Python program to query the website Dark Sky for historical weather information.[6] Exploratory research indicated that weather patterns have a strong effect on the number of rides that are taken. This site offers an applied programming interface (API) for requesting current and historical weather broken down by the hour. The API returns a wealth of useful information about the weather at the given date and time including the sunrise/sunset times, rain probability, humidity and more. This data was also uploaded to GCP into a separate table. In this initial stage I tracked every piece of weather information Dark Sky provided, as I did not yet have a grasp on which information would be useful for making predictions.

With the necessary data collected and cleaned, I created a joined table containing one row for every available hour of riding information combined with the associated weather data. This file is approximately 40 thousand rows starting from the first day of the Citi Bike program on June 1st, 2013. This file was small enough to work with locally.

Data Exploration



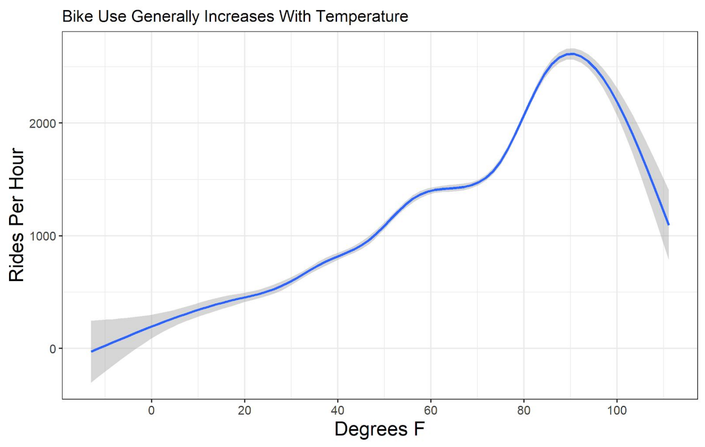
	date	sunrise	icon	precip_prob	temperature	humidity	wind_speed	rides
0	2013-06-01 00:00:00	0	clear	0.01	77.65	0.61	2.06	152
1	2013-06-01 01:00:00	0	clear	0.01	75.62	0.67	1.93	102
2	2013-06-01 02:00:00	0	clear	0.01	74.72	0.70	2.31	67
3	2013-06-01 03:00:00	0	clear	0.01	73.32	0.76	2.16	41
4	2013-06-01 04:00:00	0	clear	0.01	72.42	0.79	1.93	16

After extensive EDA, I settled on 11 predictors to use in my final model. The predictors are listed below with extended discussions on some of the more interesting ones.

1. Year
2. Hour (0 to 23)
3. Month (1 to 12)
4. Day of Week (1 [Sunday] to 7 [Saturday])
5. Sunrise (0 [Sun is Down] or 1 [Sun is Up])
6. Icon (The icon that would be displayed in a weather app. For example: clear or foggy)
7. Precipitation Probability (Float from 0 to 1 representing the chance of rain that hour)
8. Temperature (Temperature in degrees fahrenheit)
9. Humidity (Float from 0 to 1 representing the percent humidity in decimal form)
10. Wind Speed (Float representing wind speed in miles per hour)
11. Holiday (0 [No Holiday] or 1 [Holiday])

Temperature

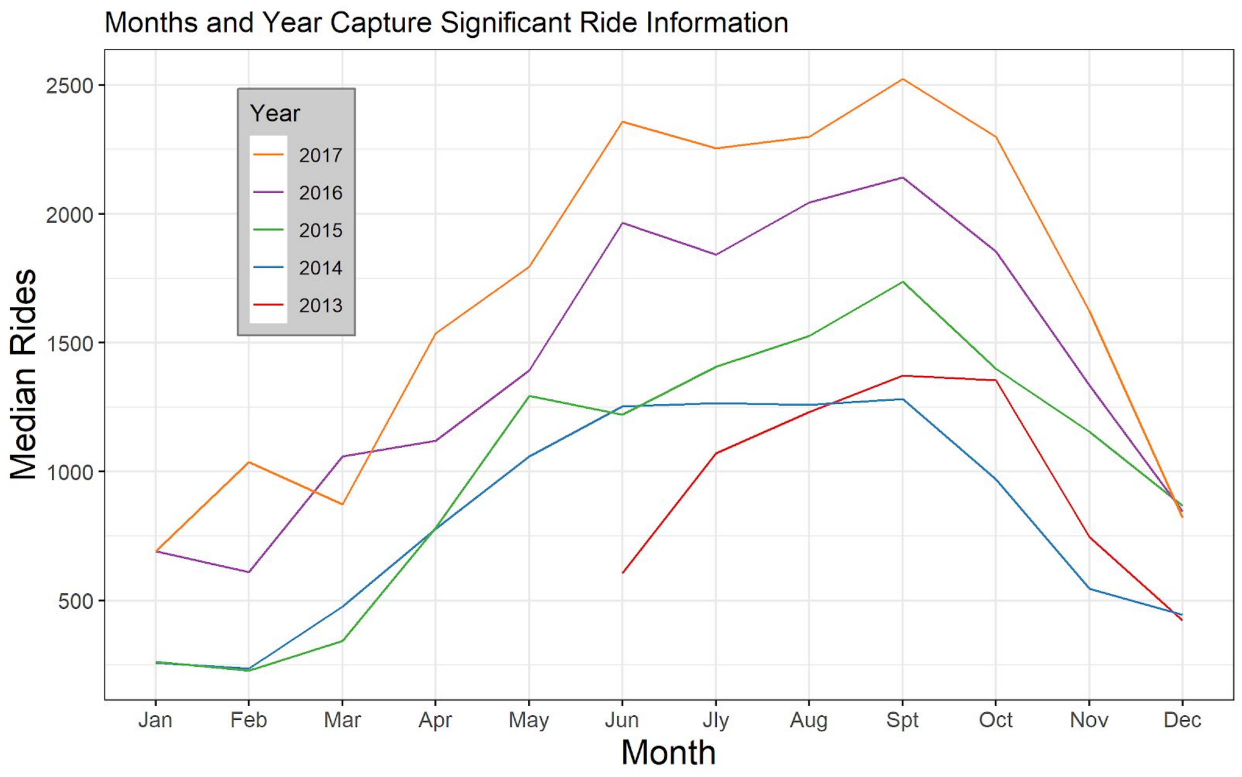
Temperature proved to be one of the most useful predictors in the model. The data fits more or less exactly as one might anticipate. The likelihood of ridership grows as the temperature becomes more pleasant before dropping again once the weather becomes too hot. Higher temperatures are also associated with more daylight hours, which itself is associated with higher ridership.



Month and Year

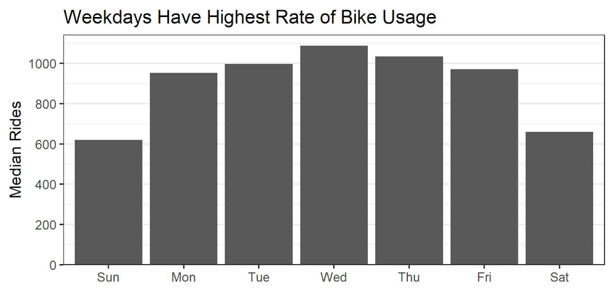
The general trend of the data shows increased ridership year after year. This is due to a number of self-reinforcing trends. As the program becomes more popular, more stations are added which increases popularity and leads to more stations. Other critical events can also affect long term riding usage as well. For example, in the summer of 2018, the so-called “transit apocalypse” wherein subways were affected by long and frequent delays, led to an additional surge in ridership.[7]

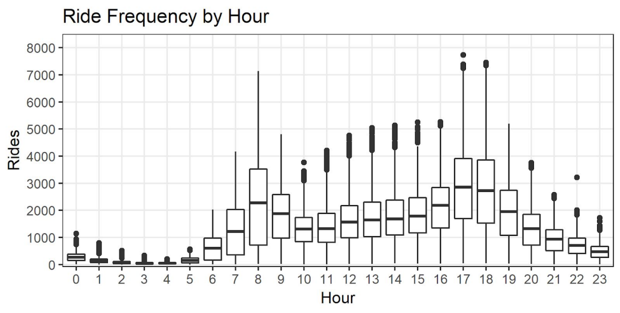
The month was also highly predictive of the average number of rides. Month acts as a great proxy for a number of other important attributes such as weather, holidays, vacations and so forth. Interestingly, spring months have lower overall ridership than fall months of roughly equivalent temperature. April is historically equal in temperature to October but has much lower ridership. In fact, May has lower ridership despite having better weather than October. This indicates that people are slow to resume riding in the spring but continue riding as long as possible before it becomes too cold.



Day of Week and Hour

These predictors provide some of the best evidence yet of the type of usage that drives the bike sharing system. Prior to my investigation, I had anticipated that the highest usage would be during the weekends as I believed that ridership was driven primarily by tourists and those with leisure time. However, the data suggests that most riding is done for commuting purposes. There are large spikes during morning and evening rush hour and the busiest days of the week are the traditional work days.





Holidays

There are a number of days throughout the year when normal activities are interrupted for special events. These interruptions lead to changes in bike rental habits and should be included in the model. However, this turned out to be quite challenging. Unlike the other predictors which have strict definitions and standard forms of measurement, quantifying special days is challenging. I needed to be careful that I wasn't overfitting my training data by simply finding excuses for misses in the model and accounting for the miss with an extra predictor.

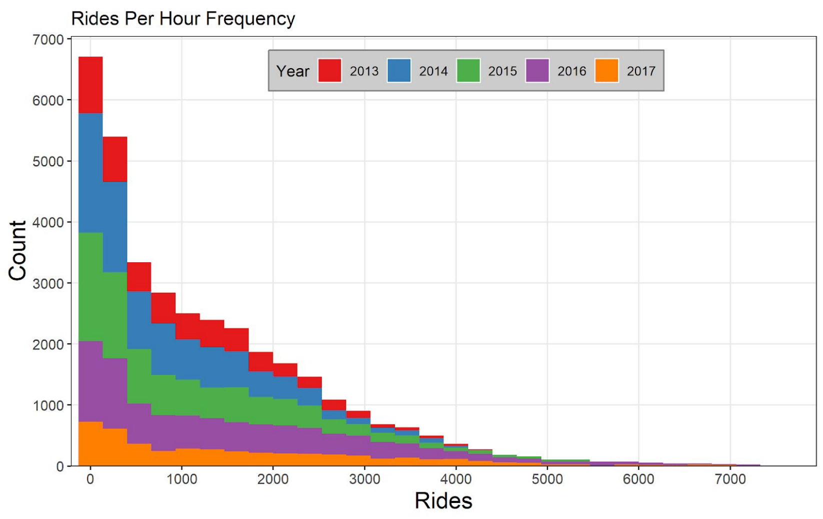
I started by marking all national holidays but this ended up not working because not all holidays are treated equally. People often do not change their daily plans for Veteran's day in the same way they change their plans for Thanksgiving or Christmas.

Making things more difficult is the fact that some holidays change days or months each year. Holiday eves and weekends could also result in changes to riding behaviour. Topping this off is the problem that holidays are rare events. There just aren't enough observations to get a sense of the trends in ridership associated with each holiday. This is all before even considering unofficial holidays such as Fleet Week or Fashion Week.

Ultimately I decided to mark the major celebratory holidays (New Years, Presidents Day, Memorial Day, Independence Day, Labor Day, Thanksgiving and Christmas) along with all of their eves.

Rides

The distribution of the response variable, rides per hour, is heavily skewed. Although the number of rides ranges from 0 to almost 8000, the median value is only about 1000. Thus, for most hours of the day the number of rides taken is quite small. This is logical as there are several hours every night and several months every year where bikes are rarely used. Due to this skewed distribution, I decided to begin my model building with learning models that are robust to skewed data and do not require transformation.



Research

One of the biggest challenges of this project was determining the type of predictive model to build. There are a multitude of options for building classical machine learning and deep learning models. Research into deep learning architectures led me to base my model on GRU (Gated Recurrent Units) modules. The authors of “Using LSTM and GRU neural network methods for traffic flow prediction”[8] demonstrated the efficacy of using these types of modules in a times series based deep learning network. The problem they were exploring, hourly traffic control, is very similar to my problem of predicting bike usage rates.

Another paper titled “A multiple time series-based recurrent neural network for short-term load forecasting”[9] also proved to be helpful. As in this paper, I needed to read data in from multiple sources, some of which was time-series and some of which was not. In my project, this occurred because the information I had access to did not come in at equal intervals. This paper discusses how to join multiple data streams together into a single functional model.

I also explored some options that did not end up being integrated into my final model but that I still found insightful. It is possible that, in the future, these ideas may prove useful in my model. One example is the paper, “Time Series Forecasting Using GRU Neural Network with Multi-lag After Decomposition”.[10] Despite my best efforts, I was not able to use time series decomposition to improve any of my models. Although I am not certain, I believe it may be due to the multiple level of seasonality in the data.

That is, the data is seasonal by hour of day, day of week, and month of year with the interactions between the seasonalities all being different.

Machine Learning

Dummy Predictor

I used Python's Scikit Learn and Keras libraries to develop my model and used mean squared error (MSE) as my performance metric. Following standard machine learning protocol, I split my data into a training, validation, and test set. As I worked with time series data, I split my data in such a way that all the validation data came after the training data and that all the test data came after both the training and validation data.

The next step was to create a dummy predictor. This predictor simulated the type of prediction that might be made by someone with access to the same data but with no knowledge of machine learning or data science. This result acts as a baseline to measure the improvement that machine learning has provided. As the data is highly seasonal, I decided to use recent similar observations to make future predictions. In this case I used the closest observation available from the previous year. For example, I would predict the third Wednesday in March at 4pm in 2015 by using the third Wednesday in March at 4pm in 2014. This dummy predictor set a baseline MSE of approximately 800,000.

Shallow Learner

Preparing the data for ingestion by the model included standard preprocessing such as one-hot-encoding the categorical variable sunrise, but also involved a number of other in-depth transformations. While the hour, day of week and month of year variables could be one-hot-encoded similarly to the sunrise variable, that would not capture the cyclic nature of the data. I wanted to encode these variables in such a way that it was clear that hour 23 was close to hour 0 and that December was close to January. I decided to make copies of each of these variables and apply a sine transformation to one instance and a cosine transformation to the other. As cosine and sine are cyclic functions, this helps the machine learning algorithm identify patterns and relationships within each predictor. The creation of two variables is necessary in order to uniquely identify each value. For example, one cycle of cosine holds every value between -1 and 1 twice. Thus, without the sine component, January would be encoded identically to June.

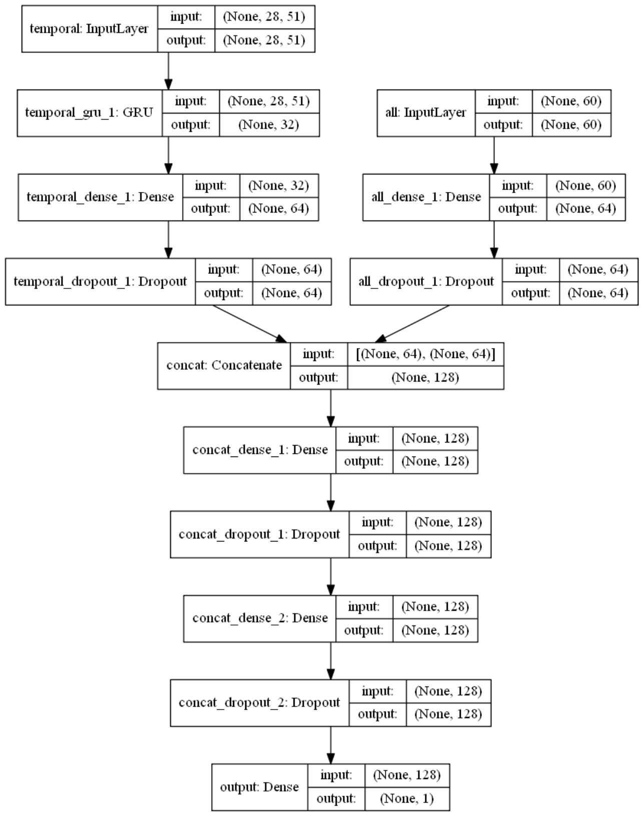
Using these predictors I iteratively created numerous models, testing each for their predictive ability. After attempting to use support vector machines and random forests, I put these aside these in favor of using an XGBoost model.

During this process of development, I discovered that recent historical data is incredibly useful for making future predictions. The number of bikes rented in the past hour is a strong estimate for the number of bikes that will be rented in the next hour. However, since this model needs to make predictions up to 48 hours in advance, I could not use the previous hour's ride data. Instead, I supplied the model with two recent historical values. The first is the number of rides 48 hours prior and the second is the number of rides 1 week prior. I also discovered that the icon predictor did not aid in my model so it was left out.

This shallow learner’s MSE is approximately 270,000 or about 60% lower than that of the dummy predictor.

Deep Learner

After countless iterations, I settled on a deep learning model that has two sets of inputs. The left input utilizes a GRU module to process the historical time series data. I fed it 28 historical values containing every predictor and the corresponding response. This allows the neural net to learn how the predictors and response change through the passage of time. I stabilized for hour of day in order to reduce the amount of historical data I needed to feed the neural net. This was then merged with the right component that contains all of the predictors for the current observation.


temporal: InputLayer | input: (None, 28, 51) | output: (None, 28, 51)
temporal_gru_1: GRU | input: (None, 28, 51) | output: (None, 32)
all: InputLayer | input: (None, 60) | output: (None, 60)
temporal_dense_1: Dense | input: (None, 32) | output: (None, 64)
all_dense_1: Dense | input: (None, 60) | output: (None, 64)
temporal_dropout_1: Dropout | input: (None, 64) | output: (None, 64)
all_dropout_1: Dropout | input: (None, 64) | output: (None, 64)
concat: Concatenate | input: [(None, 64), (None, 64)] | output: (None, 128)
concat_dense_1: Dense | input: (None, 128) | output: (None, 128)
concat_dropout_1: Dropout | input: (None, 128) | output: (None, 128)
concat_dense_2: Dense | input: (None, 128) | output: (None, 128)
concat_dropout_2: Dropout | input: (None, 128) | output: (None, 128)
output: Dense | input: (None, 128) | output: (None, 1)

The two different inputs were necessary because the predictors and the response variable become available at different times. While the predictors can all be obtained in advance, the requirements of the project necessitate making predictions up to 48 hours into the future. This creates a time frame where the predictors can be used but recent historical responses cannot.

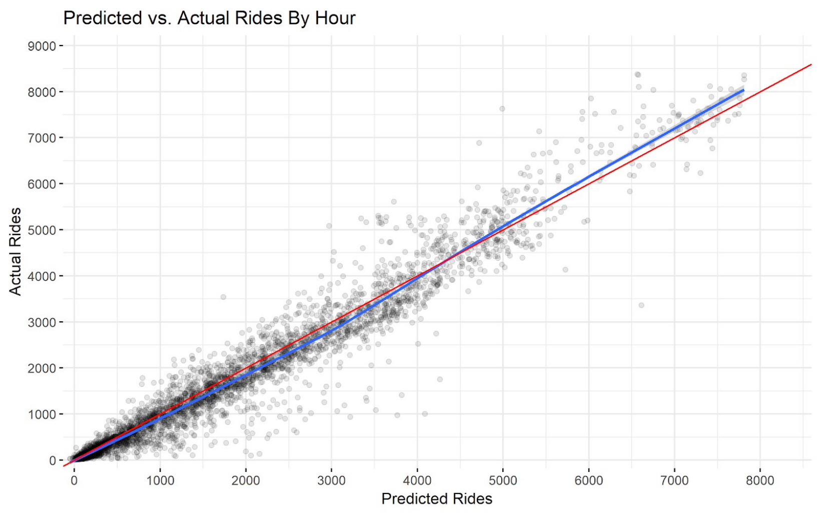
The deep learner’s MSE is approximately 220,000 or about 70% lower than that of the dummy predictor.

Final Model

After completing both models I tested ensembling them together by averaging their predictions. Ensembling can be a powerful technique as it will allow a prediction missed by one model to be “fixed” by the other model. This process of ensembling only works if the models are making different types of errors; otherwise, mistakes are only magnified. It is common practice to ensemble a deep learner with a shallow learner as they tend to make very different types of mistakes.

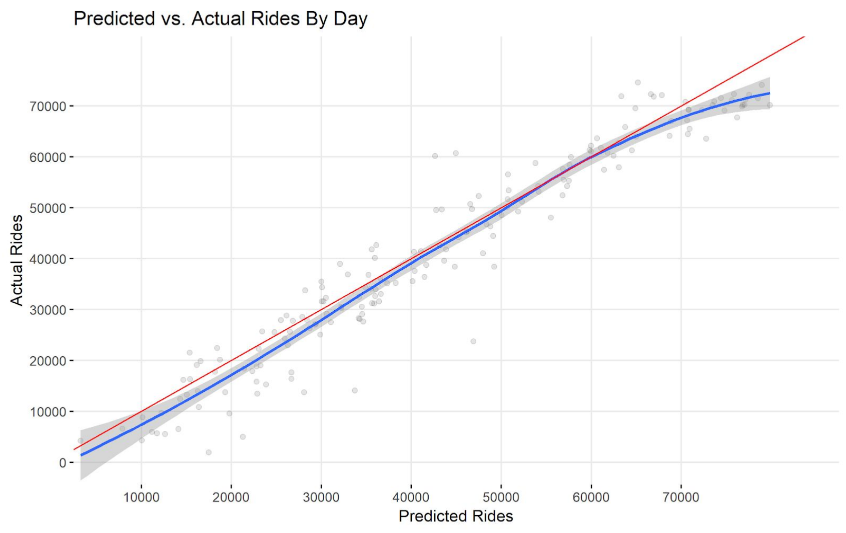
This final ensemble has an MSE of approximately 150,000 or about 80% lower than that of the dummy predictor. This will be the model that I will deploy on the dashboard.

This plot demonstrates the predictions made on the test data. The red line indicates a perfect model while the blue line indicates my model’s predictions. While it appears to be highly accurate, it can be seen that the model is being pulled low by a small number of particularly poor predictions. I inspected a number of these points manually to determine why they were so inaccurate and found a variety of explanations.



In most cases there was some significant but unaccounted for event. For example, during the entire week of September 11th, ridership drops when compared to otherwise similar weeks. The importance of the date is clear, but the association with lower ridership is not. In theory, it would be possible to add additional predictors to attempt to account for events such as these but it would greatly increase the complexity of the model. Other events such as power outages are not really possible to model.

In the second plot, I have aggregated the hours together into days. The model is very close on most days, although once again there is a very small number of misses on the lower end. The misses towards the top are due to the fact that some of the historical data is unrepresentative of the testing data.



Usage of the bike share system is trending upward. Year after year the program becomes more popular and sees higher ridership. Until this stabilizes it is likely that the model will continue to make predictions smaller than the actual values. The model is unlikely to predict ride numbers significantly larger than those which have previously occurred.

Summation

Due to the aforementioned rapid changes in usage, the model quickly falls out of date and needs to be frequently retrained on more recent data. I found that once a month is a good balance point for retraining. Ultimately, the primary reason for this is that the program is still quite young and there is consistent change in rider habits. As a result, data becomes stale in just a few months.

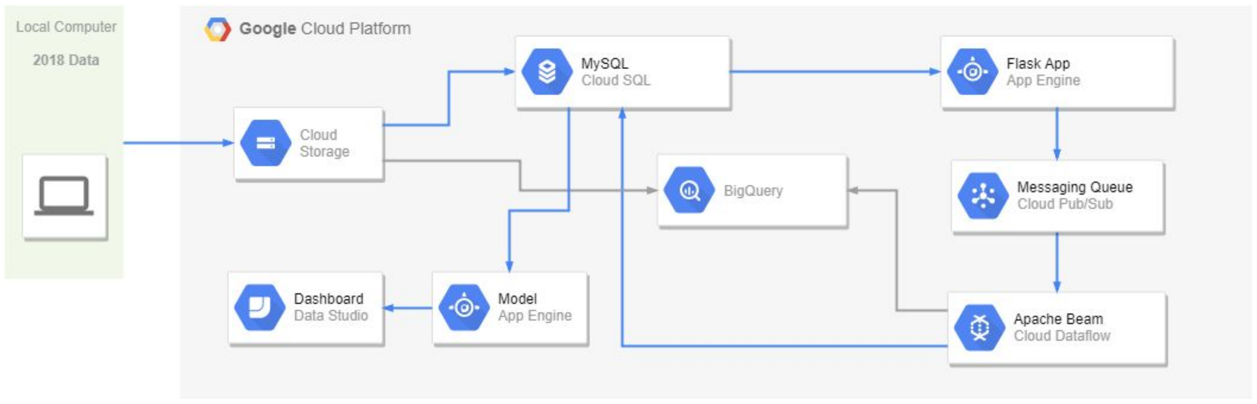
Some of the expected changes include adding of new stations, increasing the relocation options of bikes from full stations to empty ones and implementing new pricing options. Recently, Citi Bike has been exploring a dockless bike share program and a separate app that allows people to act as temporary workers for the company relocating bikes around the city as needed.

Over time, as the utilization of the system begins to settle, the model will not need to be retrained as often. However, if Citi Bike begins to switch to a dockless system, or perhaps gains competition in the form of electric scooters, the model will need to be rebuilt from scratch.

Dashboard

Overview

Creating a dashboard to display live updates comparing my predictions to the actual ride values is a two-step process. The first step is to simulate the live streaming of the bike ride data, capturing and cleaning it as it comes in. As mentioned previously, the raw bike ride data is only served in batch form after the month is over. Thus, I decided to only train my model on data through 2017 and then stream the 2018 data to the dashboard. This effectively recreated what a real live streaming service might look like.



The second step is to take this data and present it on a dashboard. I utilized a variety of services on GCP to accomplish this. It is important to note that most of the steps of the pipeline would not exist in an actual production environment and serve only to simulate the live streaming data. All of the raw data began on my local computer. From there, I uploaded it to Cloud Storage. This area served as my staging area for the other parts of my pipeline. All of the historical (pre 2018) data was stored in BigQuery tables. The new (2018) data was loaded into a MySQL table.

I then created a small Flask app with App Engine. This app queried the 2018 riding and weather data at regular intervals and wrote the information from this time frame out to the Cloud Pub/Sub messaging queue. The app itself did not produce any content or modify the data in any way and only served to move the data into the messaging queue, simulating the live reporting of bike rides.

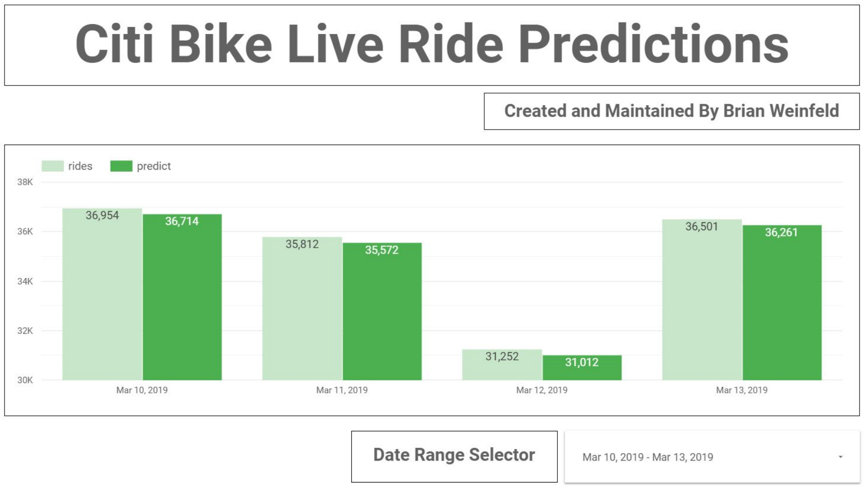
The data was pulled from the messaging queue, cleaned and sent back to a different MySQL table using a program written with Apache Beam running on Cloud Dataflow. Finally, a second app ran the data through the model to produce my predictions, which were displayed on a dashboard using Data Studio.

Apache Beam

Apache Beam is a programming model designed to perform extract, transform, load (ETL) on batch and streaming data. I wrote a Java program that took the raw data from the messaging queue and performed a number of transformations on it. The raw data was passed, unchanged, into BigQuery to join the previously stored historical data. All of the weather data, which came in 72 hours prior to prediction time, was stored in the MySQL table in the same format expected by the machine learning model. Finally, at the end of each hour, all the rides were aggregated and the total number of rides in that hour were appended to the weather data in the MySQL table.

Data Studio

Data Studio is GCP's dashboard service. This service allows a user to query any dataset stored in GCP, aggregate the results and display them in near real time. With the data properly organized in a MySQL table and with the predictions made by the model, all that was left to do was query this information and display it in near real time.

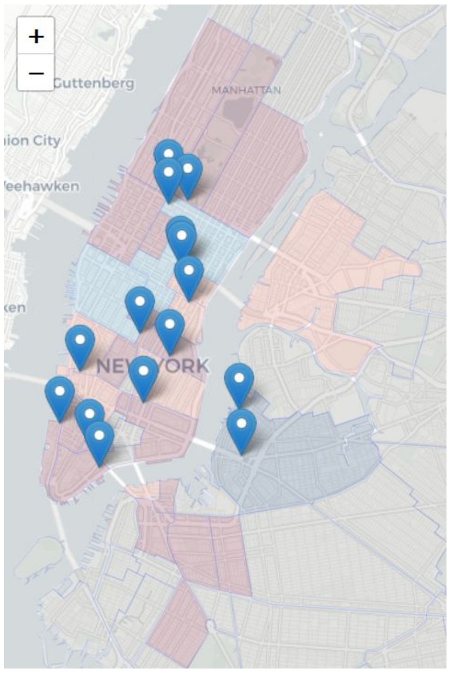


The March 10th, 11th and 12th data shows the past results to allow for comparison between my predictions and the actual results. The current day, March 13th in this example, has the predicted number of results shown immediately while the number of actual rides updates roughly every hour, increasing as time passes to show the actual bike ride count. By default the dashboard will show today and the previous three days of information, but this can be changed with the selector at the bottom of the dashboard.

Critical Stations

Initially, I had planned on adding a live updating map to the dashboard containing neighborhood level information about daily bike usage. The goal was to identify areas that were at critical capacity so that additional resources could be deployed to those areas. During EDA, I discovered that creating a live service using this information was excessive as the key stations and routes remained consistent across seasons and years.

Using PySpark with seven distributed workers to handle the massive calculations, I created a graph with stations acting as nodes and the rides between the stations acting as directed edges. Each edge is weighted based on the number of rides between the stations. I can now apply a number of established graph algorithms for identifying critical stations and routes. Neighborhoods are colored based on the number of critical stations and the most important are marked.[11][12]

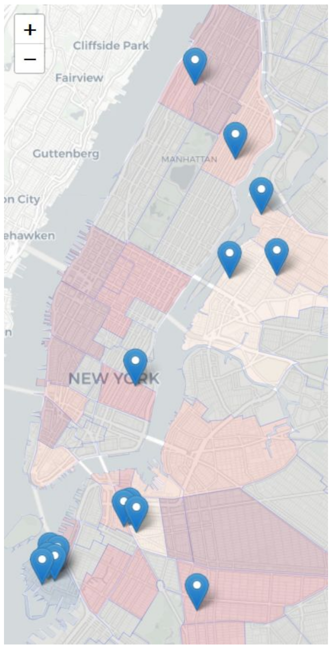
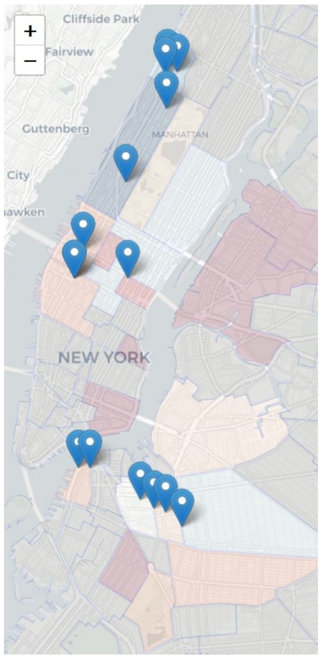


PageRank

PageRank is the famous algorithm created by Google that is used in part to order its search results. The basic version of the algorithm ranks a page's relevance based on how many other pages link to it. In this case, a station is highly ranked if bikes from many other stations ride to it. Stations with a high page rank are integral to the network. They are also potential bottleneck points. There are numerous markers around Grand Central Station and Central Park along with other commonly visited transportation and tourist locations. These stations draw riders from all over the city.

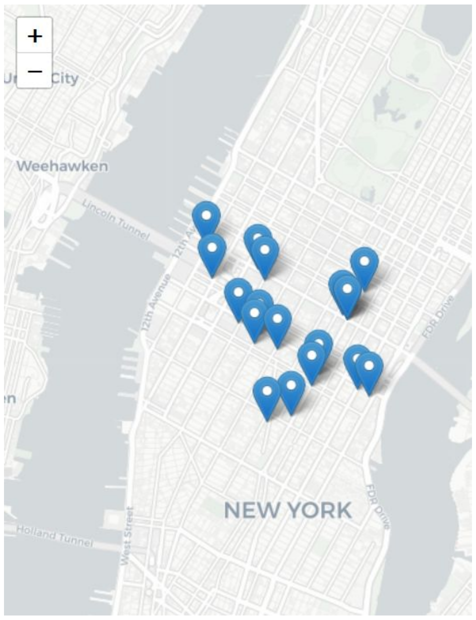
Ratio

A simple statistic like ratio of trips ended to trips started can be quite useful at identifying critical stations. These are the stations that are most likely to be full or empty and must be monitored closely. The left map identifies stations where trips often end but rarely begin, while the right map identifies stations where trips often begin but rarely end. It can be inferred that the stations in Brooklyn are linked and represent people traveling to the Waterfront and Governor's Island. In Manhattan, many rides begin on Columbia's campus and are likely used by students travelling off campus. In general, trips are more likely to start in Manhattan and end in Queens or Brooklyn than vice versa.

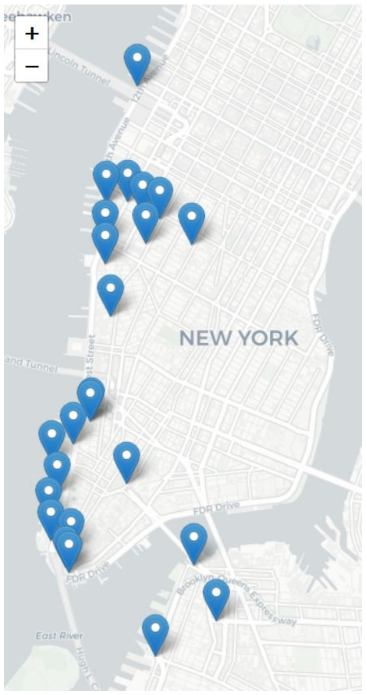
Trips End Often	Trips Begin Often
 Cliffside Park Fairview Guttenberg MANHATTAN NEW YORK	 Cliffside Park Fairview Guttenberg MANHATTAN NEW YORK

Sugraphs

Important subgraphs can be found by removing the edges between nodes whose weight fall below a certain threshold. Through exploration this threshold can be fine tuned until a number of small subgraphs are created. These subgraphs represent some of the most frequented and densely connected corridors in the original graph. The process of selecting a threshold is more of an art than a science. If the threshold is too high, the result will be comprised of many single nodes while a threshold too low will result in a tiny number of subgraphs that contain too many nodes to interpret.



After some exploration, I set a threshold that resulted in 13 subgraphs, the two biggest of which are comprised of 15 and 22 nodes respectively. The first subgraph sits horizontally across Manhattan, and encomposses stations around the Lincoln Tunnel, Penn Station, and Grand Central Station. It almost perfectly parallels the route taken by the 7 and the L trains. This subgraph likely comprises commuters substituting bikes for the last leg of their journey or those that utilize two of the aforementioned locations in their commute.



The second subgraph includes more stations near critical commuting locations. The stations include Battery Park and Brookfield Place, where ferries from New Jersey dock, and stations near PATH stops. The area surrounding the dense pocket of stations is notable for its relatively poor subway coverage.

Conclusion

Overall, I consider the project to be quite successful. The model proved to have strong predictive ability as long as it is frequently retrained on the newest available data. The dashboard component was critical to the project as it allowed me to quickly summarize and display my predictions in an easily understood format. The importance of effective communication cannot be overstated. Even the strongest results are worthless unless they can be clearly interpreted by the decisions makers. While working on this project, I gained valuable insight into the development of time series deep learning models and practical applications of GCP technologies.

Technologies

All available code can be found at: www.github.com/brian-w-projects/citibike-dashboard

- Python was used to scrape the raw data from the internet and its associated data science libraries (incl. Jupyter, Pandas, Scikit Learn and Keras) were used to perform EDA and to create the predictive models.
- R (tidyverse) was used to create the charts and maps used in this paper.
- Flask created the small apps that were used to schedule recurring jobs.
- Java was used to create an ETL pipeline via Apache Beam.
- Apache Spark was used to create the graph network via PySpark.
- Google Cloud Platform was used extensively to aid in every step of the project.

Bibliography

1. Bicycle-sharing system. (2019, April 03). Retrieved from https://en.wikipedia.org/wiki/Bicycle-sharing_system

2. Citi Bike. (2019, March 31). Retrieved from https://en.wikipedia.org/wiki/Citi_Bike

3. Flegenheimer, M. (2013, August 15). The Balancing Act That Bike-Share Riders Just Watch. Retrieved from https://www.nytimes.com/2013/08/15/nyregion/the-balancing-act-that-bike-share-riders-just-watch.html

4. Job Search. (n.d.). Retrieved from https://usr55.dayforcehcm.com/CandidatePortal/en-US/motivate/site/citibikecareers

   The Driver (On-Call) will not receive a regular schedule. They will be contacted on an ad hoc basis to fill open shifts.The Driver (On-Call) will be responsible for operating a vehicle for the purposes of relocating bikes throughout the system. To that end, the Driver (On-Call) will load and unload the vehicle with bikes, and dock/undock bikes at every station stop. Additionally...

5. Motivate International, Inc. (n.d.). Citi Bike System Data. Retrieved from https://www.citibikenyc.com/system-data

6. Weather. (n.d.). Retrieved from https://darksky.net/forecast/40.7127,-74.0059/us12/en

7. Thrillist. (2018, August 28). How to Become a Cyclist and Avoid NYC's Ongoing Transit Apocalypse. Retrieved from https://www.thrillist.com/lifestyle/new-york/biking-in-nyc-guide

8. R. Fu, Z. Zhang and L. Li, "Using LSTM and GRU neural network methods for traffic flow prediction," 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC), Wuhan, 2016, pp. 324-328. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7804912&isnumber=7804853

9. Zhang, B., Wu, JL. & Chang, PC. Soft Comput (2018) 22: 4099. https://doi.org/10.1007/s00500-017-2624-5

10. Zhang X., Shen F., Zhao J., Yang G. (2017) Time Series Forecasting Using GRU Neural Network with Multi-lag After Decomposition. In: Liu D., Xie S., Li Y., Zhao D., El-Alfy ES. (eds) Neural Information Processing. ICONIP 2017. Lecture Notes in Computer Science, vol 10638. Springer, Cham
11. Hofman, J. (n.d.). New York City Maps. Retrieved from https://rpubs.com/jhofman/nycmaps

12. Tsvetovat, M., & Kouznetsov, A. (2012). *Social network analysis for startups*. Beijing: OReilly.