# NYC Taxi Fare Predictions

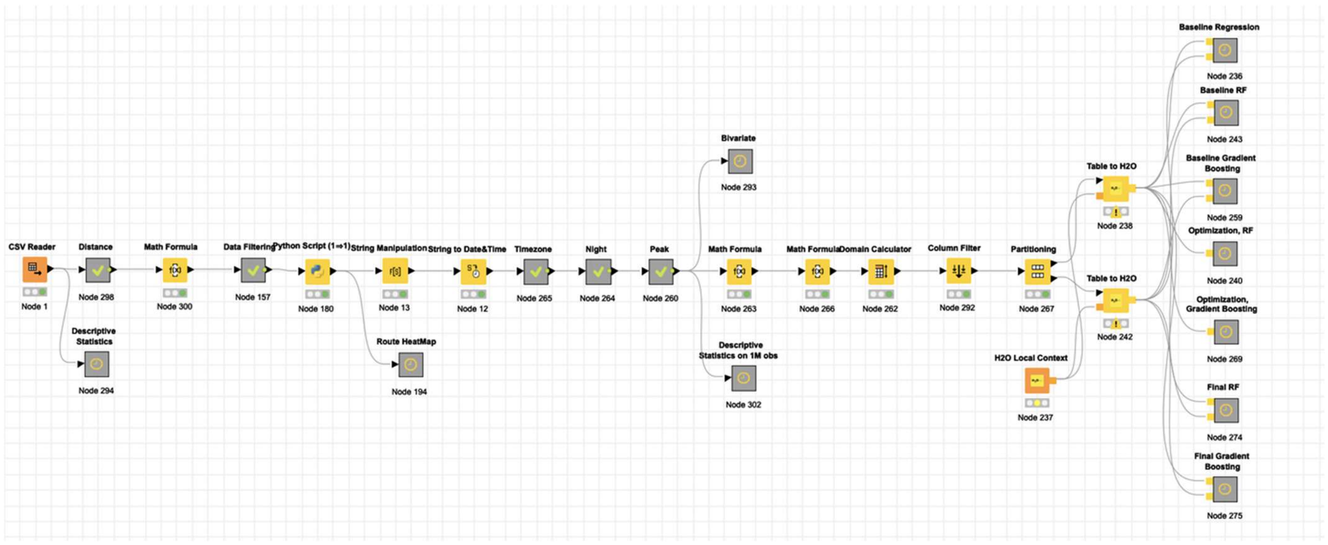
DECEMBER 11

Sophie Bottani
Arturo Cerasi
Giulia Gatteschi
Enrico Giannelli
Anna Pederzani
Aldin Traljic

## Goal of the Analysis

The struggles of the taxi industry have been well documented over the years. The main threat has come from Uber, which has rapidly risen in popularity and become a fierce competitor. The issues have been compounded this year with the coronavirus pandemic, pushing New York City's taxi industry almost to the brink of ruin. However, the taxi industry has served an important purpose for many years and continues to do so today while evolving to meet the ever-changing demands of the market. From famous TV shows and movies set in NYC, to the everyday businesswoman/man hailing a cab to get to their next meeting, to tourists making their way to the next location on their guide map, there is always one ever-present: the iconic yellow NYC taxi, a true symbol of the world's most famous city.

Our analysis will focus on predicting taxi ride fares in NYC based off all the information available to us; there are more details about the dataset in the following section. By being able to give accurate predictions of fares, we could design an app, or improve the existing ones, that would provide customers with a fast and precise estimate of their fare. This would allow the taxi industry to connect with the increasing amount of tech savvy customers that are lured in by the accessible apps that Uber and Lyft offer. Aside from attracting new customers, faithful taxi riders can also benefit by having more readily available predictions about their fares. The implications of the model will be discussed in depth with the results of our analysis and in the managerial implications part.



# Dataset Used and Descriptive Statistics

## DATASET USED

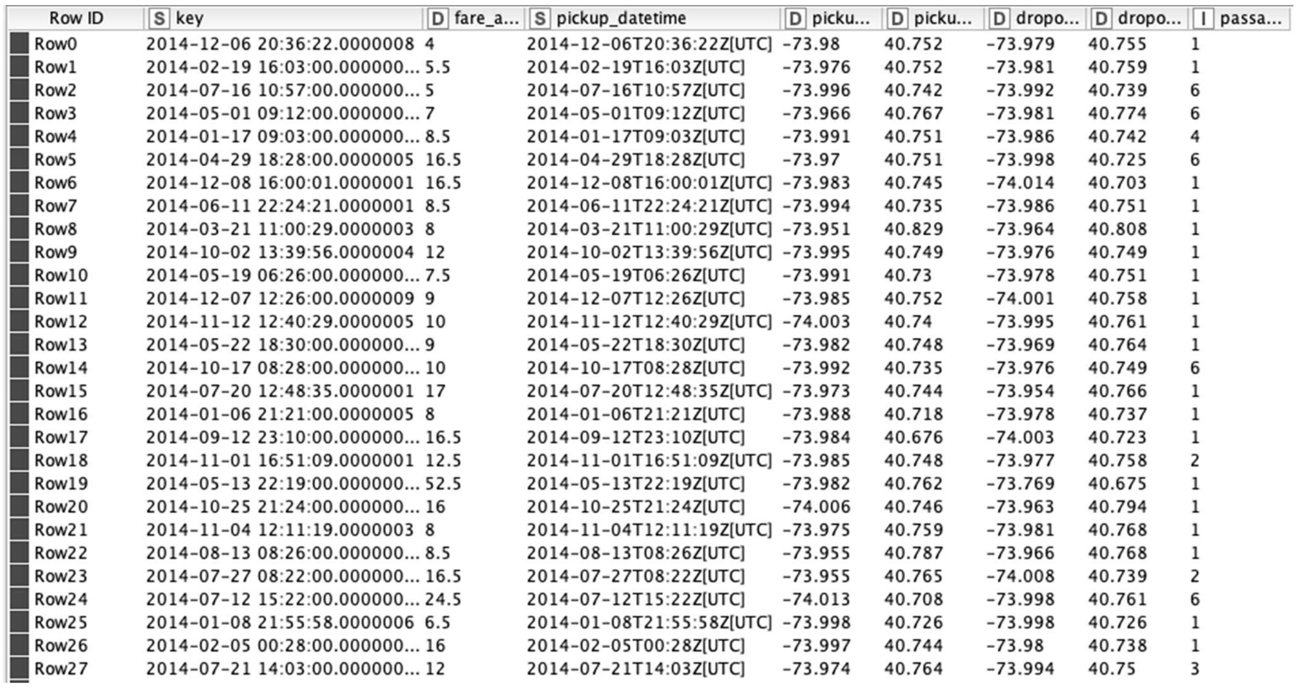
The dataset selected is from Kaggle.com and contains information regarding taxi rides in New York City from 2009 until 2015.
For storage and computational complexity reasons we decided to consider only the data regarding the year 2014 since the original dataset contained about 55 million observations. When evaluating which year to use, we considered the years that best captured the intense competition from competitors, such as Uber, which started in NY in 2011 and by 2014 was already firmly established. We also took into account recent regulations and pricing policies that are still active today, such as the rush hour surcharge. The years that best took these factors into account were 2014 and 2015. In the end, we decided to use 2014 because it still included ample data and the dataset only encompassed half of 2015.

The dataset contains one unique ID and 7 features, and these are:

| name | type | description |
|---|---|---|
| key | STRING | unique string composed of the pickup datetime and a unique integer |
| pickup_datetime | STRING | when the taxi ride started |
| pickup_longitude | FLOAT | longitude coordinate of where the taxi ride started |
| pickup_latitude | FLOAT | latitude coordinate of where the taxi ride started |
| dropoff_longitude | FLOAT | longitude coordinate of where the taxi ride ended |
| dropoff_latitude | FLOAT | latitude coordinate of where the taxi ride ended |
| passenger_count | INTEGER | number of passengers in the taxi ride |
| fare_amount | FLOAT | cost of the taxi ride in dollars |

For the goal of our analysis, we will use fare_amount as the target variable. It is the variable that we want to predict with our models.

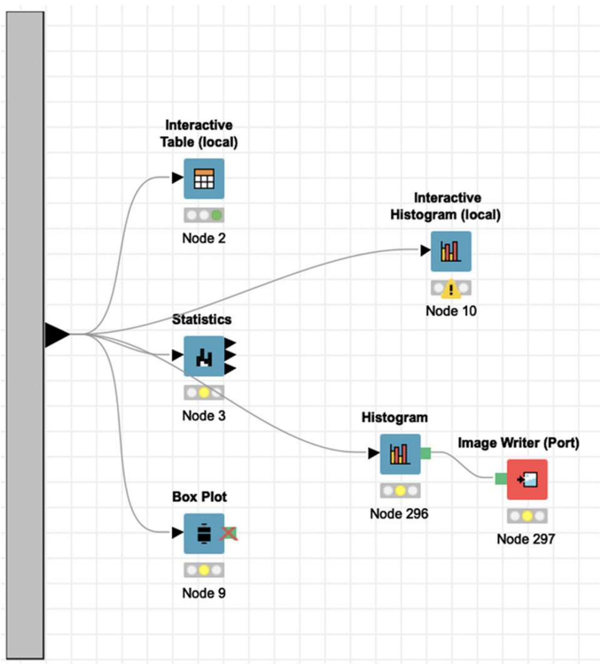
The restricted dataset contains 8,246,270 observations:



| Row ID | S key | D fare_a... | S pickup_datetime | D picku... | D picku... | D dropo... | D dropo... | I passa... |
|---|---|---|---|---|---|---|---|---|
| Row0 | 2014-12-06 20:36:22.0000008 | 4 | 2014-12-06T20:36:22Z[UTC] | -73.98 | 40.752 | -73.979 | 40.755 | 1 |
| Row1 | 2014-02-19 16:03:00.000000... | 5.5 | 2014-02-19T16:03Z[UTC] | -73.976 | 40.752 | -73.981 | 40.759 | 1 |
| Row2 | 2014-07-16 10:57:00.000000... | 5 | 2014-07-16T10:57Z[UTC] | -73.996 | 40.742 | -73.992 | 40.739 | 6 |
| Row3 | 2014-05-01 09:12:00.000000... | 7 | 2014-05-01T09:12Z[UTC] | -73.966 | 40.767 | -73.981 | 40.774 | 6 |
| Row4 | 2014-01-17 09:03:00.000000... | 8.5 | 2014-01-17T09:03Z[UTC] | -73.991 | 40.751 | -73.986 | 40.742 | 4 |
| Row5 | 2014-04-29 18:28:00.0000005 | 16.5 | 2014-04-29T18:28Z[UTC] | -73.97 | 40.751 | -73.998 | 40.725 | 6 |
| Row6 | 2014-12-08 16:00:01.0000001 | 16.5 | 2014-12-08T16:00:01Z[UTC] | -73.983 | 40.745 | -74.014 | 40.703 | 1 |
| Row7 | 2014-06-11 22:24:21.0000001 | 8.5 | 2014-06-11T22:24:21Z[UTC] | -73.994 | 40.735 | -73.986 | 40.751 | 1 |
| Row8 | 2014-03-21 11:00:29.0000003 | 8 | 2014-03-21T11:00:29Z[UTC] | -73.951 | 40.829 | -73.964 | 40.808 | 1 |
| Row9 | 2014-10-02 13:39:56.0000004 | 12 | 2014-10-02T13:39:56Z[UTC] | -73.995 | 40.749 | -73.976 | 40.749 | 1 |
| Row10 | 2014-05-19 06:26:00.000000... | 7.5 | 2014-05-19T06:26Z[UTC] | -73.991 | 40.73 | -73.978 | 40.751 | 1 |
| Row11 | 2014-12-07 12:26:00.0000009 | 9 | 2014-12-07T12:26Z[UTC] | -73.985 | 40.752 | -74.001 | 40.758 | 1 |
| Row12 | 2014-11-12 12:40:29.0000005 | 10 | 2014-11-12T12:40:29Z[UTC] | -74.003 | 40.74 | -73.995 | 40.761 | 1 |
| Row13 | 2014-05-22 18:30:00.000000... | 9 | 2014-05-22T18:30Z[UTC] | -73.982 | 40.748 | -73.969 | 40.764 | 1 |
| Row14 | 2014-10-17 08:28:00.000000... | 10 | 2014-10-17T08:28Z[UTC] | -73.992 | 40.735 | -73.976 | 40.749 | 6 |
| Row15 | 2014-07-20 12:48:35.0000001 | 17 | 2014-07-20T12:48:35Z[UTC] | -73.973 | 40.744 | -73.954 | 40.766 | 1 |
| Row16 | 2014-01-06 21:21:00.0000005 | 8 | 2014-01-06T21:21Z[UTC] | -73.988 | 40.718 | -73.978 | 40.737 | 1 |
| Row17 | 2014-09-12 23:10:00.000000... | 16.5 | 2014-09-12T23:10Z[UTC] | -73.984 | 40.676 | -74.003 | 40.723 | 1 |
| Row18 | 2014-11-01 16:51:09.0000001 | 12.5 | 2014-11-01T16:51:09Z[UTC] | -73.985 | 40.748 | -73.977 | 40.758 | 2 |
| Row19 | 2014-05-13 22:19:00.000000... | 52.5 | 2014-05-13T22:19Z[UTC] | -73.982 | 40.762 | -73.769 | 40.675 | 1 |
| Row20 | 2014-10-25 21:24:00.000000... | 16 | 2014-10-25T21:24Z[UTC] | -74.006 | 40.746 | -73.963 | 40.794 | 1 |
| Row21 | 2014-11-04 12:11:19.0000003 | 8 | 2014-11-04T12:11:19Z[UTC] | -73.975 | 40.759 | -73.981 | 40.768 | 1 |
| Row22 | 2014-08-13 08:26:00.000000... | 8.5 | 2014-08-13T08:26Z[UTC] | -73.955 | 40.787 | -73.966 | 40.768 | 1 |
| Row23 | 2014-07-27 08:22:00.000000... | 16.5 | 2014-07-27T08:22Z[UTC] | -73.955 | 40.765 | -74.008 | 40.739 | 2 |
| Row24 | 2014-07-12 15:22:00.000000... | 24.5 | 2014-07-12T15:22Z[UTC] | -74.013 | 40.708 | -73.998 | 40.761 | 6 |
| Row25 | 2014-01-08 21:55:58.0000006 | 6.5 | 2014-01-08T21:55:58Z[UTC] | -73.998 | 40.726 | -73.998 | 40.726 | 1 |
| Row26 | 2014-02-05 00:28:00.000000... | 16 | 2014-02-05T00:28Z[UTC] | -73.997 | 40.744 | -73.98 | 40.738 | 1 |
| Row27 | 2014-07-21 14:03:00.000000... | 12 | 2014-07-21T14:03Z[UTC] | -73.974 | 40.764 | -73.994 | 40.75 | 3 |

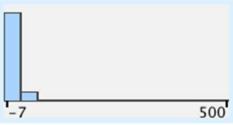
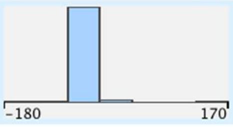
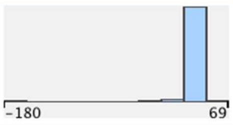
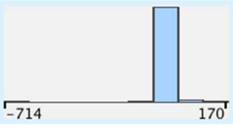
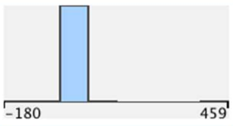
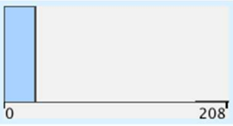
## DESCRIPTIVE STATISTICS

Using the node **Statistics,** we get a summary of all the numeric features.

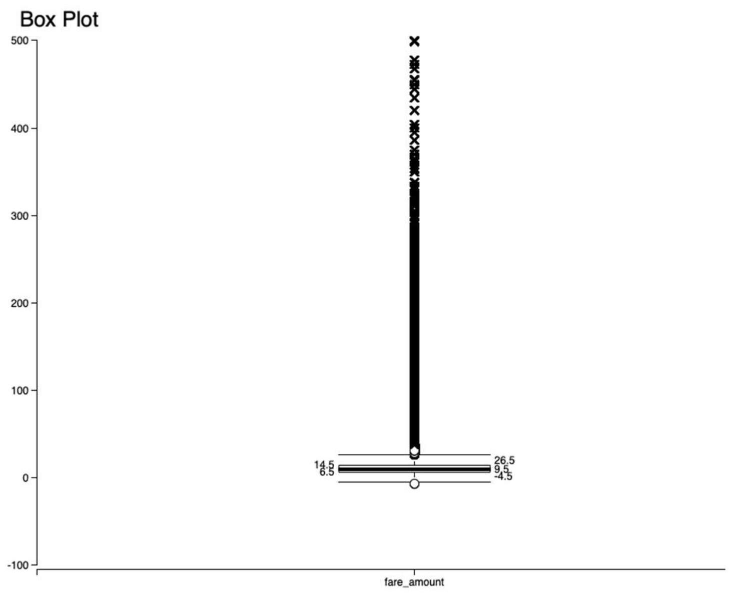


Looking at the output table it can be seen that the dataset contains some outliers: first of all, the fare_amount has some negative values, in fact the minimum fare price is -$7. This must be an error since the base fare is $2.5. Also, the latitude, both for the pick-up and drop-off, has some negative and (0,0) coordinates: this is impossible since New York is in the Northern Hemisphere. Even the longitude feature has some positive values corresponding to locations that are east of the prime meridian, while New York is west of the prime meridian.

Starting from the fare_amount we can observe that the average price of the rides is $12.92. Using the node **Interactive Histogram (local)**, it can be seen that most of the prices (more than 99% of the observations) range between $0 and $60.

| Column | Min | Mean | Median | Max | Std. Dev. | Skewness | Kurtosis | No. Missing | No. +∞ | No. -∞ | Histogram |
|---|---|---|---|---|---|---|---|---|---|---|---|
| fare_amount | -7 | 12.9203 | ? | 500 | 11.2611 | 3.7492 | 33.9321 | 0 | 0 | 0 |  |
| pickup_longitude | -180 | -72.5021 | ? | 169.9728 | 10.3359 | 6.8725 | 45.3046 | 0 | 0 | 0 |  |
| pickup_latitude | -180 | 39.9388 | ? | 69.2804 | 5.6972 | -6.9132 | 47.2484 | 0 | 0 | 0 |  |
| dropoff_longitude | -714.4 | -72.4931 | ? | 169.9728 | 10.3663 | 6.819 | 46.7759 | 4 | 0 | 0 |  |
| dropoff_latitude | -180 | 39.935 | ? | 458.65 | 5.7173 | -6.765 | 54.3156 | 4 | 0 | 0 |  |
| passanger_count | 0.0 | 1.6957 | ? | 208 | 1.3602 | 3.2936 | 195.4272 | 0 | 0 | 0 |  |

However, the box plot shows that there are numerous outliers for the fare of rides, and we will have to treat these, especially the negative amounts, to get better results.



The average longitude for the pick-up is -72.5021 while for the drop-off is -72.4931. The average latitude for the pick-up and for the drop-off respectively is 39.9388 and 39.9350. All these values are in line with the coordinates of New York City.

For the passenger_count instead, we can see in the occurrences table in the **Statistics** node that 70% of rides have only one passenger, 14% have two and 5% have five. The maximum number of passengers observed in the dataset is 208, which is impossible, and it is probably due to a data processing error. However, only three observations list a number of passengers equal to 208, while all the others record values of 8 or fewer people.

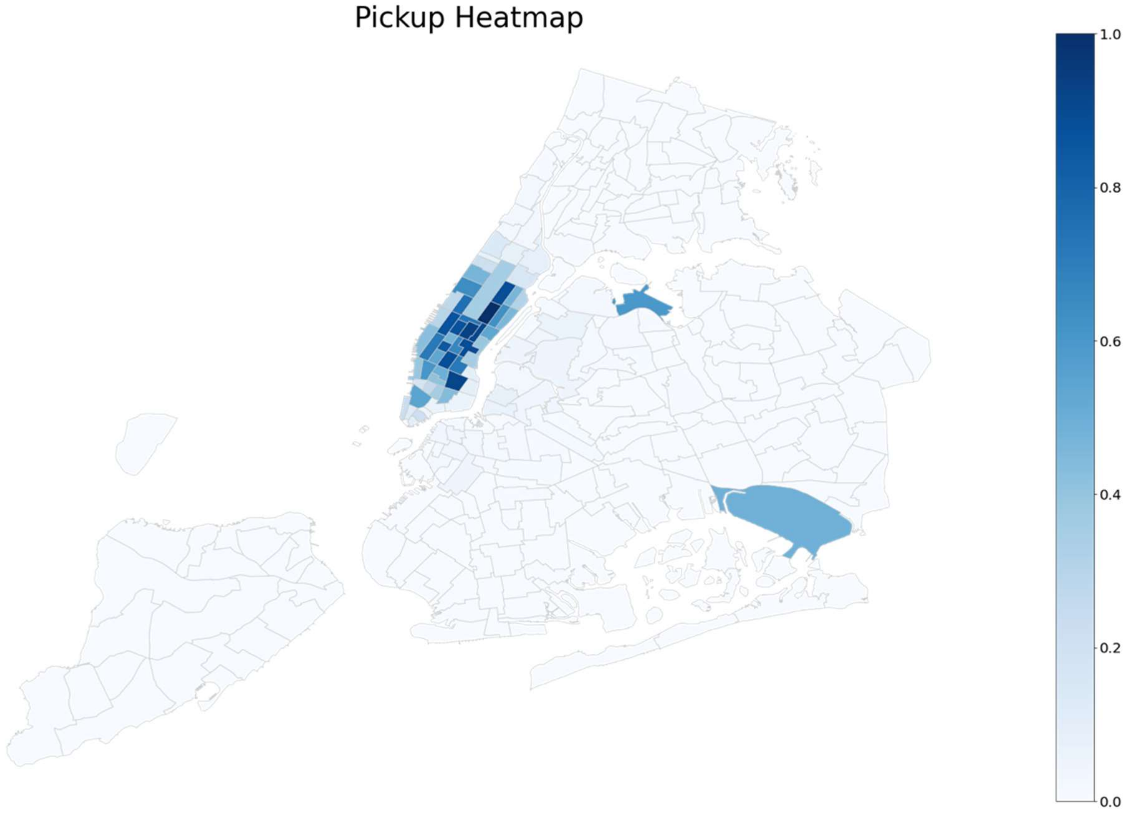
In addition, it can be noticed that the database has very few missing values (only 8 in the entire dataset) which is very good, and these are related to the drop-off coordinates.

To better visualize the data and to understand which areas are more popular in terms of pick-up and drop-off we created two heat maps. To do so we wrote a Python script using the geopandas library.
In particular we used data on the taxi areas in New York[1] to cluster the drop-off and pick-up coordinates into smaller areas. Based on that division we were able to plot some heat maps based on the number of drop-offs and pick-ups in each area.

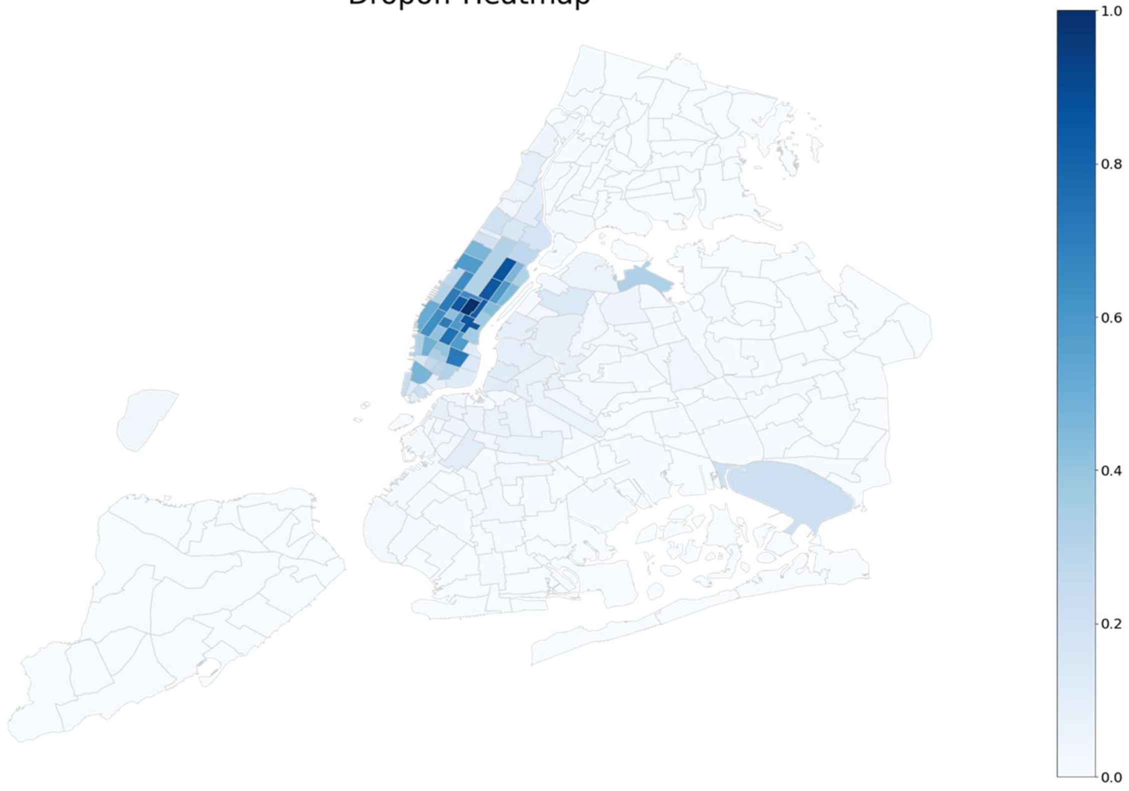
We see that the most affected areas are the Upper East side, the Upper West side, the Theatre district, Midtown Manhattan and in general the areas near Park Avenue and 5th Avenue. Therefore, all those areas where the main city activities are located. We can also notice that the area near the Financial District and the World Trade Center are very popular in terms of pickup and drop-off locations given that there are many offices in the area. In addition, it is interesting how the other hot areas are the ones near transportation hubs, like Penn Station and of course the two airports of JFK and LaGuardia.

[1] https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc

Pickup Heatmap



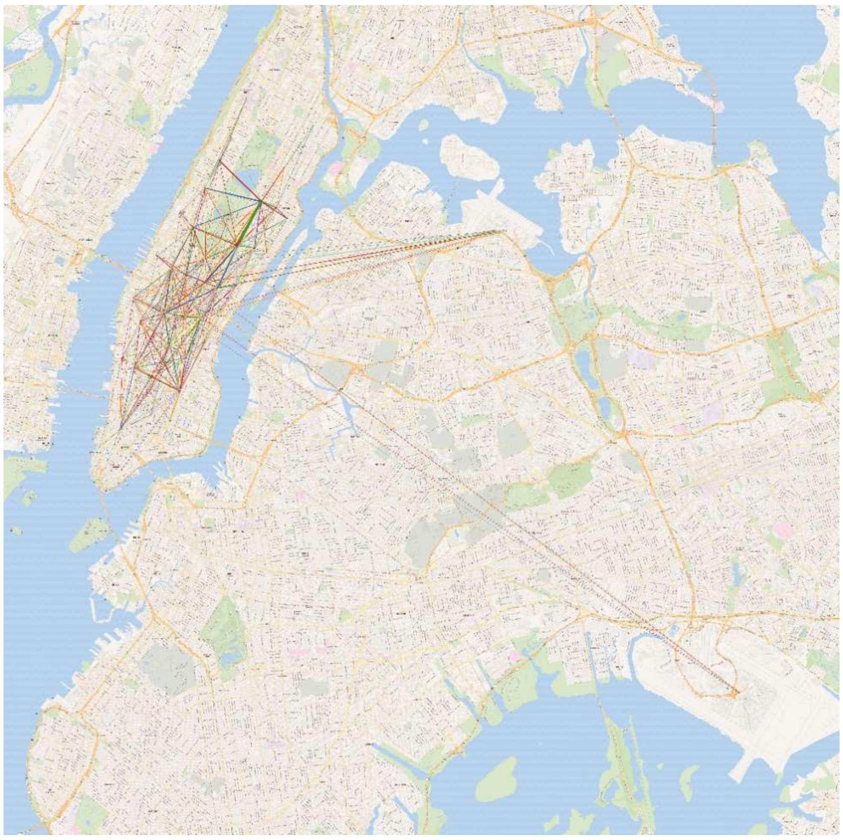
Dropoff Heatmap



We also created a Route Map where we can visualize the most popular routes. To do so we created a Route metanode where we first defined a route variable as (pickup_area, dropoff_area), we then grouped the observations by route. We then created a polygon variable, a line between the barycenter of the observations within each pickup_area and dropoff_area. Next, we drew a map with the **OSM Map View** node with the 90th percentile of the observations by route. In this graph the thicker the line the most popular the route. The thicker the line the more trafficked is the rout.

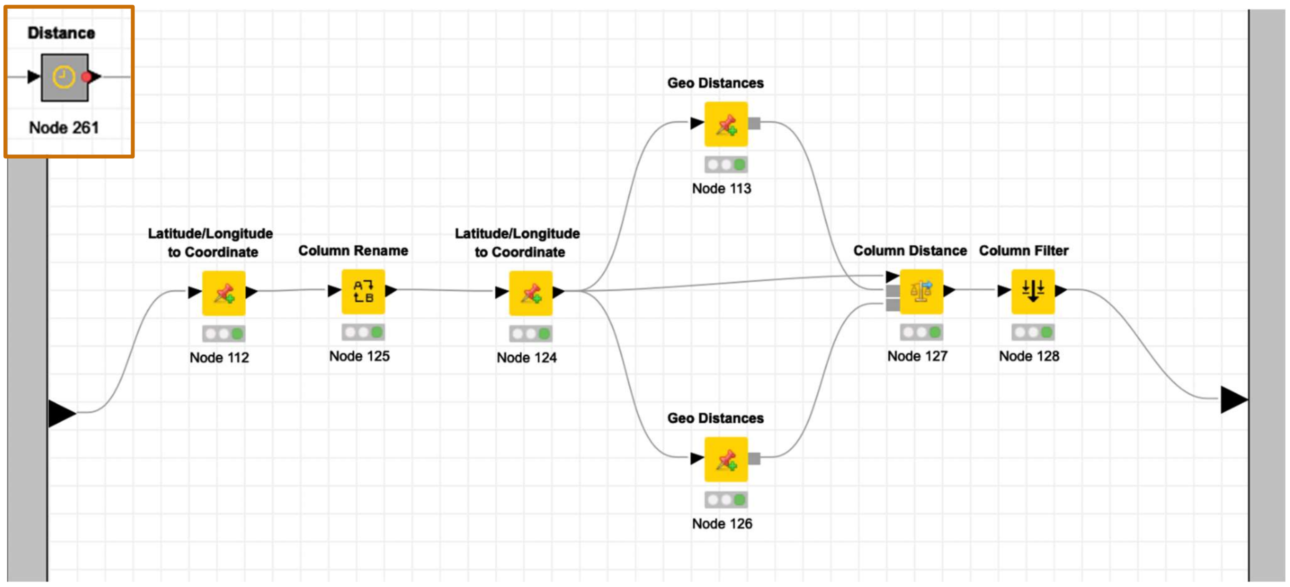
So, we can see that the most popular routes are the ones connecting the Upper East Side with Midtown Manhattan or those within the Upper East Side, the Upper West Side or Midtown. As we saw also in the previous heat maps the area with the highest concentration of routes is Midtown and this is probably because it is where the main attractions are, like the Theatre District, the Diamond District, 5th avenue and all the museums. There are also many routes connecting Midtown Manhattan to the East Village which is known for its nightlife. As we expected, popular routes are also the ones connecting Manhattan to the airports of LaGuardia and JFK. However, in general the most popular routes are those connecting with the Upper East/West Side.

| *Route* | *Count* |
|---|---|
| Upper East Side South TO Upper East Side North | 36375 |
| Upper East Side North TO Upper East Side South | 30627 |
| Upper East Side South TO Upper East Side South | 30118 |
| Upper East Side South TO Upper East Side South | 30053 |
| Upper East Side South TO Midtown East | 18376 |
| Upper East Side South TO Midtown Center | 18097 |
| Upper West Side South TO Upper West Side North | 17966 |
| Upper West Side South TO Lincoln Square East | 17441 |
| Lincoln Square East TO Upper West Side South | 16654 |
| East Village TO East Village | 16589 |
| Upper West Side North TO Upper West Side South | 15711 |
| Gramercy TO Murray Hill | 15678 |
| Penn Station/Madison Sq West TO Times Sq | 15465 |
| Midtown Center TO Midtown Center | 15039 |
| Penn Station/Madison Sq West TO Midtown Center | 14883 |
| Clinton East TO East Chelsea | 14867 |
| Union Sq TO Murray Hill | 14785 |
| Lenox Hill West TO Upper East Side North | 14727 |
| Midtown Center TO Upper East Side South | 14604 |
| Clinton East TO Clinton East | 14219 |



# Data Preparation & Feature Engineering

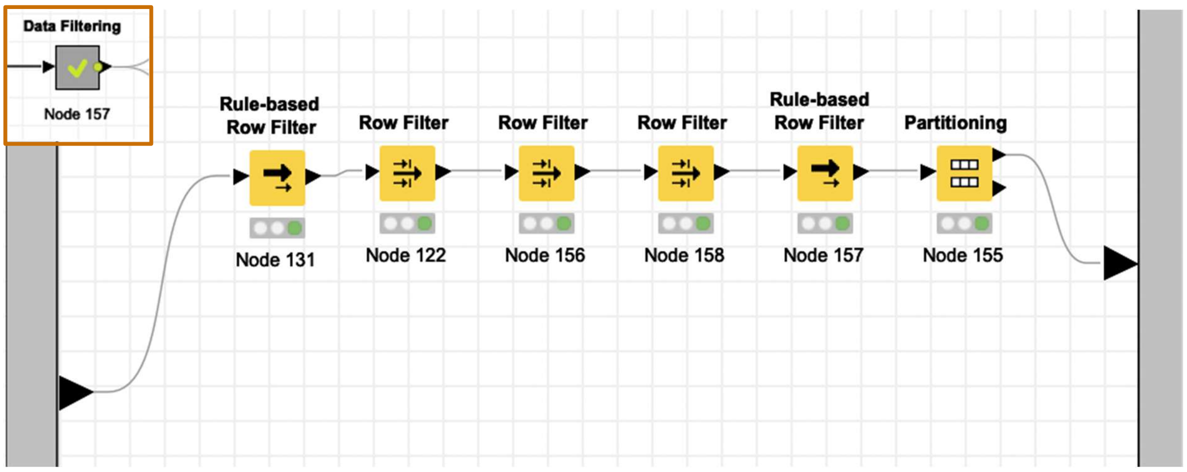
Data preparation and feature engineering were crucial for our project.
First of all, we noticed that our dataset did not include a measure for neither the length driven nor the distance as the crow flies of each taxi ride. However, given the objective of our project a variable that took into account the length of the ride was needed. We thus created a distance variable. To do so we applied the **Geo Distances** node on the pick-up and drop-off coordinates, in particular, we calculated the distance between the two points using the Haversine distance. This distance measure is different from the actual kilometers driven by the taxi, however this proxy was the best that we could do with our computational resources and still fit the scope of our analysis.



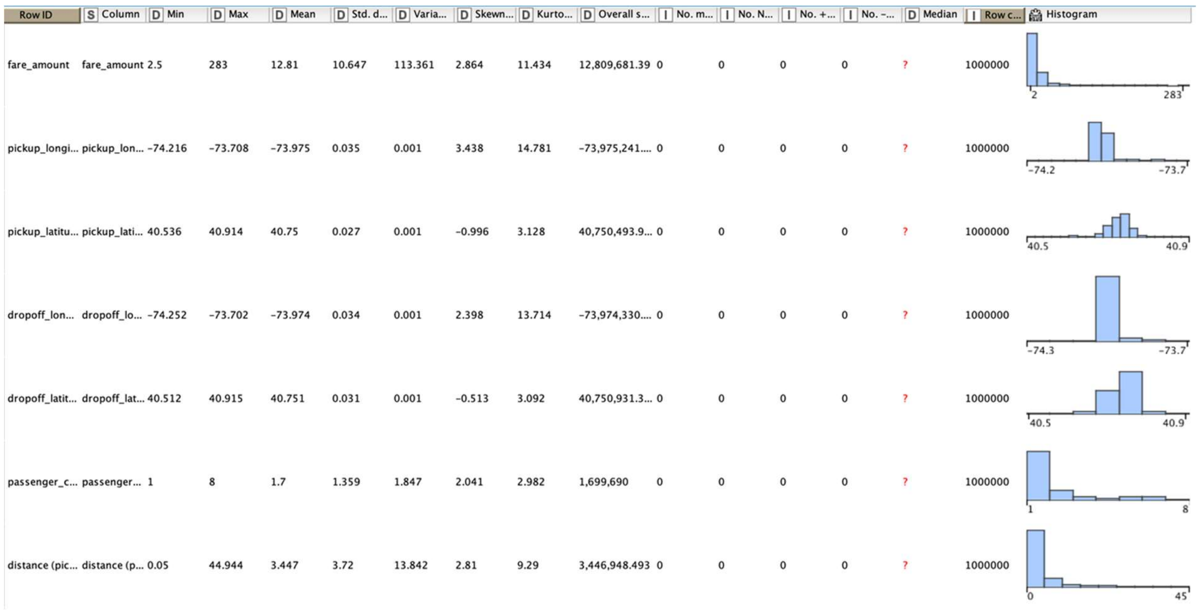
In addition, we created a fare/km variable which is calculated as fare/distance to eliminate outliers. In fact, there were rides that had a very high fare for a short distance and this was probably due to errors or maybe the customer decided to go somewhere and then go back to the original starting point, which is irrelevant to the scope of the analysis. In particular we cut those observations with a fare/distance > 50$/km. This variable will obviously not be included in our model since it contains the target variable.

As we saw in the Descriptive Statistics section the data contained many outliers that we had to treat. We decided to consider only the taxi rides in the New York area in order to simplify our model and to get rid of the outliers regarding the longitude and latitude variables. We did this by defining a rectangle around New York City and removing all those rows which had the pickup or drop off coordinates outside this area. In addition, we also deleted all the observations which had the pickup coordinate equal to the drop off coordinate, since they were probably data entry errors or abandoned rides.

Then we deleted all those rows which had a fare amount under $2.5, since this is the base fare. These observations were probably errors and we decided to simply delete them given that we had many observations and they don't add any additional value.
Then we considered the passenger_count feature which had some impossibly high values. We kept only the observations with a passanger_count between 1 and 9 since these are the only reasonable values for the variable. By doing so we actually only deleted 3 observations that had a value of 208 passengers.

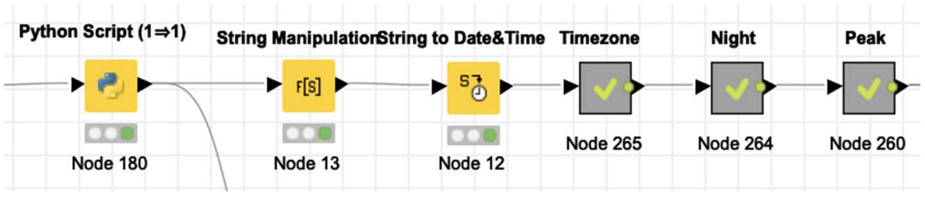


Looking at the distributions after removing outliers and datapoints with errors we see that data is better distributed now and has more reasonable values. Moreover, the average distance of a ride is 3.72 km, while the minimum and maximum distance are of 0.05km and 45km respectively.

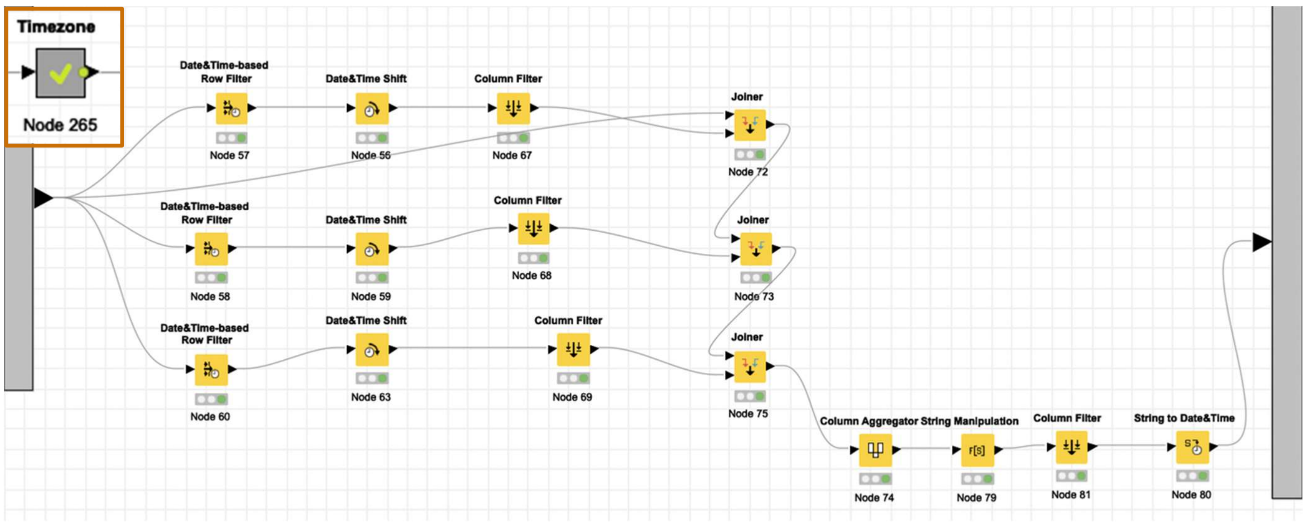


| Row ID | S Column | D Min | D Max | D Mean | D Std. d... | D Varia... | D Skewn... | D Kurto... | D Overall s... | I No. m... | I No. N... | I No. +... | I No. -... | D Median | I Row c... | Histogram |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| fare_amount | fare_amount | 2.5 | 283 | 12.81 | 10.647 | 113.361 | 2.864 | 11.434 | 12,809,681.39 | 0 | 0 | 0 | 0 | ? | 1000000 | 2 283 |
| pickup_longi... | pickup_lon... | -74.216 | -73.708 | -73.975 | 0.035 | 0.001 | 3.438 | 14.781 | -73,975,241.... | 0 | 0 | 0 | 0 | ? | 1000000 | -74.2 -73.7 |
| pickup_latitu... | pickup_lati... | 40.536 | 40.914 | 40.75 | 0.027 | 0.001 | -0.996 | 3.128 | 40,750,493.9... | 0 | 0 | 0 | 0 | ? | 1000000 | 40.5 40.9 |
| dropoff_lon... | dropoff_lo... | -74.252 | -73.702 | -73.974 | 0.034 | 0.001 | 2.398 | 13.714 | -73,974,330.... | 0 | 0 | 0 | 0 | ? | 1000000 | -74.3 -73.7 |
| dropoff_latit... | dropoff_lat... | 40.512 | 40.915 | 40.751 | 0.031 | 0.001 | -0.513 | 3.092 | 40,750,931.3... | 0 | 0 | 0 | 0 | ? | 1000000 | 40.5 40.9 |
| passenger_c... | passenger... | 1 | 8 | 1.7 | 1.359 | 1.847 | 2.041 | 2.982 | 1,699,690 | 0 | 0 | 0 | 0 | ? | 1000000 | 1 8 |
| distance (pic... | distance (p... | 0.05 | 44.944 | 3.447 | 3.72 | 13.842 | 2.81 | 9.29 | 3,446,948.493 | 0 | 0 | 0 | 0 | ? | 1000000 | 0 45 |

We then continued the data preparation by adding four variables to our dataset: pickup_area, dropoff_area, pickup_boro and dropoff_boro. To do so we wrote a script in Python using the geopandas and shapely libraries, the taxi zones database cited above[1] and another database that included the boundaries of the New York borough.[2] Our code comprised of two nested for loops, the outer one assigned to each data point the corresponding borough given the borough boundaries and the inner one looked at each area inside that borough to assign the area. We repeated these two loops both for the pick-up and the drop-off coordinates. We then included the python script in our workflow using the **Python Script** node.

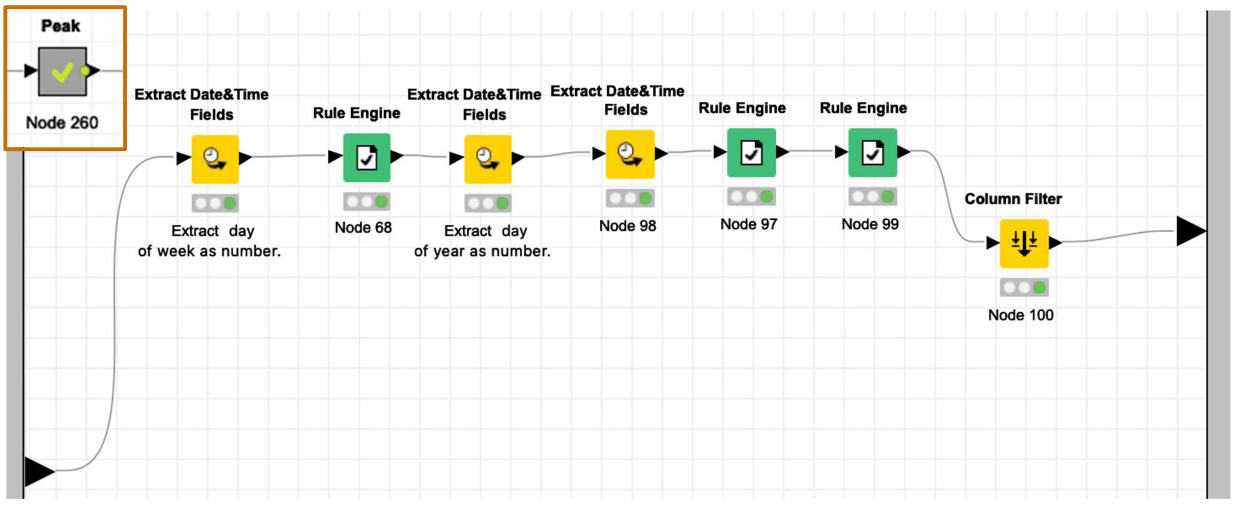


We also realized that we needed to adjust the time zone. First of all, we transformed the pickup_datetime feature from a string to a Date&Time variable. The dataset originally had the dates and times according to the UTC time zone, but this would be a problem for us because we needed to add features that took into account the rush hour surcharge and the night surcharge. We created a sizeable metanode to convert the time zone to ETS5ETD, the correct and standard time zone for New York City since the 'Uniform Time Act of 1966.' We paid attention to correctly convert the time zone by taking into account the differences brought about by the switches to daylight savings time, as UTC remained unaffected while ETS5ETD changed. For this reason, we needed to filter the dates into three separate columns and then apply the **Date&Time Shift** node to each, shifting by 5 hours the dates that were not impacted by daylight savings time and by 4 hours the dates that were. We joined the converted dates and added them to our dataset under the name date_time.



[2] https://data.cityofnewyork.us/City-Government/Borough-Boundaries/tqmj-j8zm

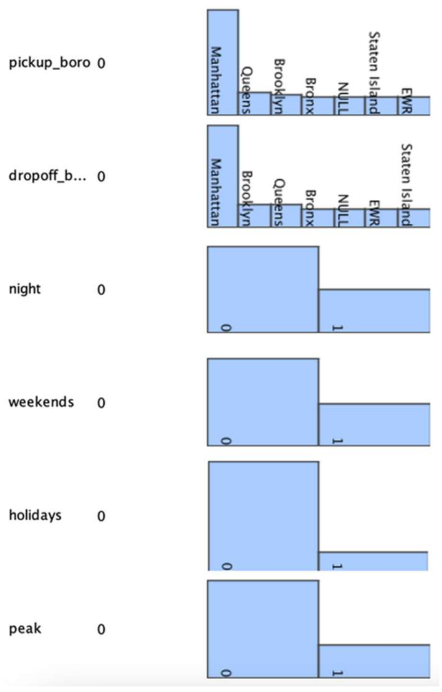
After, we turned our focus towards feature engineering. The first features we added were the overnight surcharge and the rush hour surcharges, plus an additional 30 cents per ride for the improvement surcharge.
The overnight surcharge was an additional 50 cents for rides from 8pm to 6am. We created a metanode to add this feature to our dataset. In the metanode, we used the **Extract Date&Time Fields** node to extract the time field “hour” from our date_time feature. With this, we were able to apply a **Rule Engine** node to create our night dummy variable. 1 was assigned to rides between 0 (included) and 6 (excluded) and from 20 (included) and greater, 0 was assigned otherwise. Then we created a metanode for capturing the rush hour surcharge, which we called peak.



The rush hour surcharge implied an additional $1.00 for rides from 4pm to 8pm on weekdays, excluding holidays. Therefore, we had the task of distinguishing between weekdays and weekends, and also needed to distinguish holidays, for which we looked up and used bank holidays in the US for 2014. In this metanode, we again apply the **Extract Date&Time Fields** node, this time in order to extract the day of the week as a number. Then, we apply a **Rule Engine** node to set the weekends, days 6 and 7 of the week, to 1, and all the other days, 1 to 5, to 0. After this, we applied another **Extract Date&Time Fields** node to extract the day of the year as a number. Then, we applied two **Rule Engine** nodes. The first sets bank holidays for 2014 to 1. We did this by considering the different bank holidays not as a date, but as a number corresponding to the day of the year that given date fell on. This allowed us to effectively mark these dates as create the dummy variable holidays. The last **Rule Engine** of this node ran the following code:

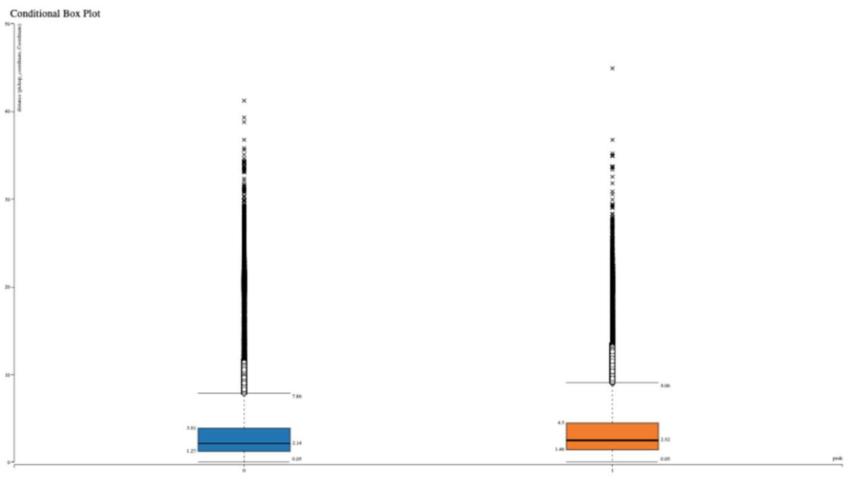
```
$Hour$>=16 AND $Hour$ < 20 AND NOT ($weekends$=1 OR $holidays$=1) => 1
TRUE => 0
```

With this, we created the rush_hour dummy variable.

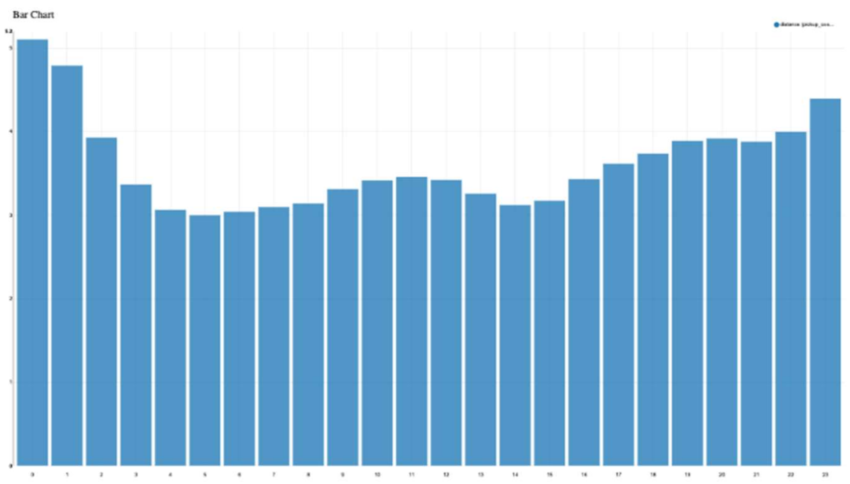


We took a look at the nominal variables created and we can see how the most popular borough in terms of pickup and drop-off is Manhattan, as we also saw in the previous heat maps. We also notice how there are more rides during the day and during the week. In addition, in our data set there are much fewer rides during a holiday, but this is probably because there are not many holiday dates. Finally, the majority of rides are not during peak hour.

With the newly created variables we have performed some bivariate analysis. In particular, we have created a boxplot with the distances of the rides during peak (1) and non-peak(0) hours. From the graph we can see that the average distance during peak hours is larger. An ANOVA test performed on the same variables confirms this result. Indeed, the average distance during peak hours is 3.685km while during non-peak hours is 3.403km. A reason behind this result might be that peak hours coincide with the busiest hours for airports. Thus, the trips to the airport that tare usually much longer than average are more frequent during those hours, contributing to the higher average distance.



Here we have also plotted a bar chart with the distance for each hour. We can observe that during the night we have longer rides. This could be explained by the fact that during the night public transportation is less frequent and consequently that people prefer to use taxi at late hours. However, we must add that the rides recorded at those hours are very few in our dataset.



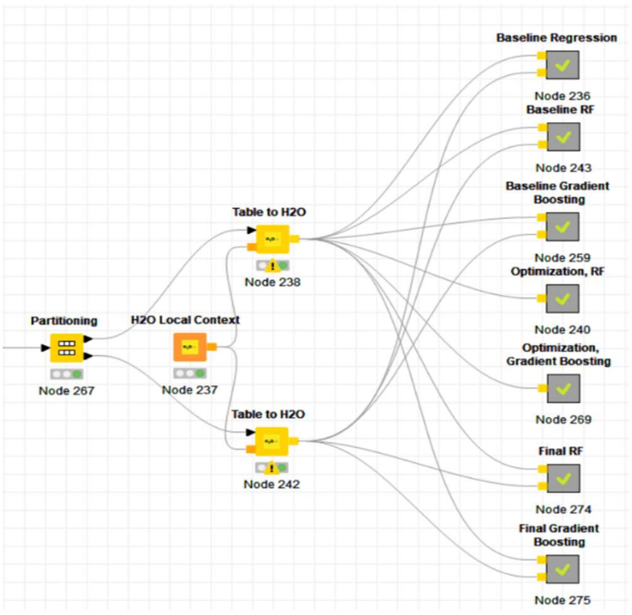
Crucially, our dataset did not include a measure for neither the length driven nor the distance as the crow flies of each taxi ride. However, given the objective of our project a variable that took into account the length of the ride was needed. We thus created a distance variable. To do so we applied the **Geo Distances** node on the pick-up and drop-off coordinates, in particular, we calculated the distance between the two points using the Haversine distance. This distance measure is different from the actual kilometers driven by the taxi, however this proxy was the best that we could do with our computational resources and still fit the scope of our analysis.

# Data Modeling & Model Evaluation

## DATA MODELING

Once we constructed all the necessary variables (like the drop-off and pick-up area category or the distance as the crow flies) and cleaned the data of outliers we moved to the prediction part.

This section was key to our project. Our aim in fact, given all the necessary parameters, was to estimate the cost of rides in advance. This, as we will explain further on, will allow the tax industry to offer a taxi-hailing app where you can get an estimation of the cost of the ride before you even book your taxi.



*Model Estimation Section of the Workflow*

Therefore, we need to find the best model to predict our target variable, fare_amount. To do so we first selected all the independent variables included in the original dataset plus those we built so far:

- passengers_count. A numerical, discrete variable already included in the initial dataset.
- A binary dummy variable to check if the ride is taken during the night (1) or not (0).
- Another binary dummy, called peak, is used to check whether the ride takes place during the rush hour of a weekday (holidays excluded)
- We also kept 2 dummies even though they do not directly impact the fare of a taxi ride:
  - one for holidays, called holidays
  - another one for weekends, called weekend

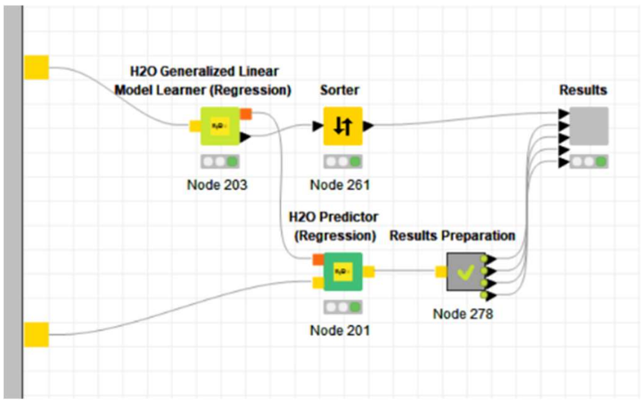
  We included them because during these periods traffic may be more intense, in turn causing rides to be longer which, thus, in turn, affect the cost of a ride.
- We also added the distance (calculated as the crow flies)
- Finally, we have 4 categorical variables related to the starting and ending locations of the rides:
  - pickup_boro → indicates in which New York borough the ride has started
  - dropoff_boro → in which borough it ended
  - pickup_area → same concept as the pickup_boro variable but this time we categorized the starting location using smaller taxi zones, defined by the open data service of NYC.
  - dropoff_area → same as with pickup_area but with the dropoff

Next, we needed to find the most appropriate model for our data set. We therefore evaluated the accuracy of various numerical prediction models. The ones we have chosen are:

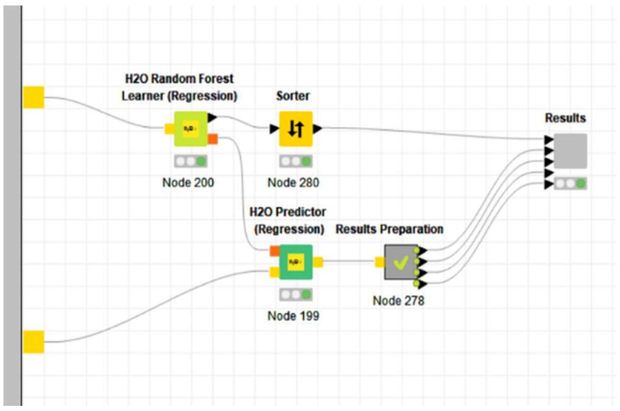
- A basic Linear Regression
- A Random Forest Model
- And a Gradient Boosting Algorithm

First thing we did was then to divide through a **Partitioning** node our training set into 2 sections: a training one comprehending 80% of the dataset and a testing one with the rest of the dataset. Then using the **Table to H2O** node part of the H2O extension we trained our different models using the training set and then evaluated them using the testing set.

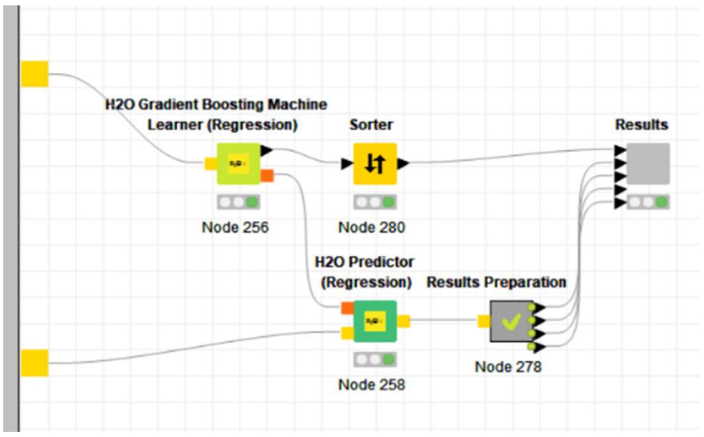
The models work as follows: for each model the training set is connected to the correspondent **H2O Regressor/Learner** node. In these passages the nodes automatically distinguish between numerical and categorical variables and act consequently. For example, the linear regression creates a dummy variable for each possible value of the pickup and dropoff areas. Here below are some screenshots of the metanodes for each of the 3 selected models. We chose H2O as their models supported numerical target variables, contrary to the standard KNIME nodes.


H2O Generalized Linear Model Learner (Regression)
Sorter
Results
Node 203
Node 261
H2O Predictor (Regression)
Results Preparation
Node 201
Node 278

*Baseline Linear Regression metanode*

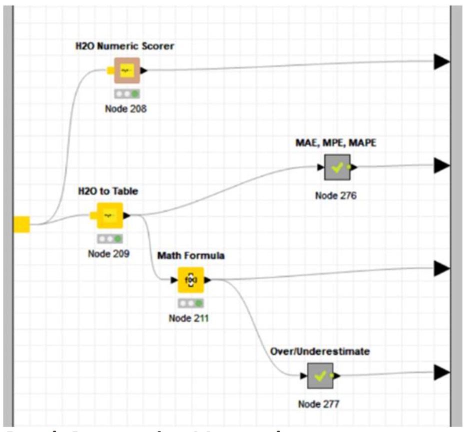

H2O Random Forest Learner (Regression)
Sorter
Results
Node 200
Node 280
H2O Predictor (Regression)
Results Preparation
Node 199
Node 278

*Baseline Random Forest metanode*


H2O Gradient Boosting Machine Learner (Regression)
Sorter
Results
Node 256
Node 280
H2O Predictor (Regression)
Results Preparation
Node 258
Node 278

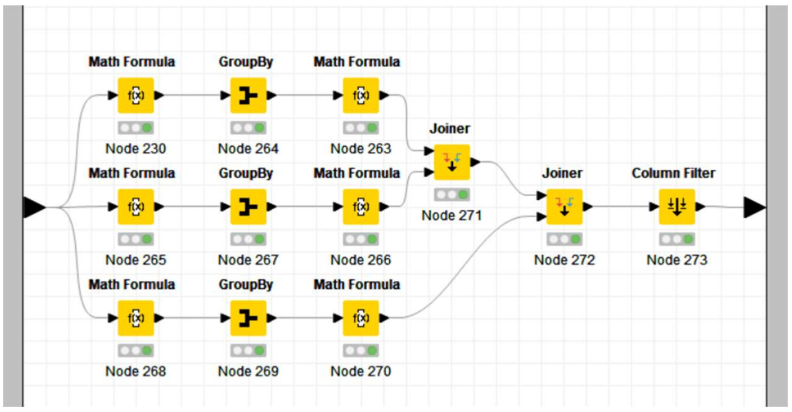
*Baseline Gradient Boosting metanode*

The results of the Regressor/Learner nodes were then passed on to the Predictor node which estimated the fare_amount value for each entry of the test set to which it was connected. The estimations were compared to the true values to generate metrics to evaluate the quality of the regression. This was done inside the **Results Preparation** metanode, which is showed here on the side.



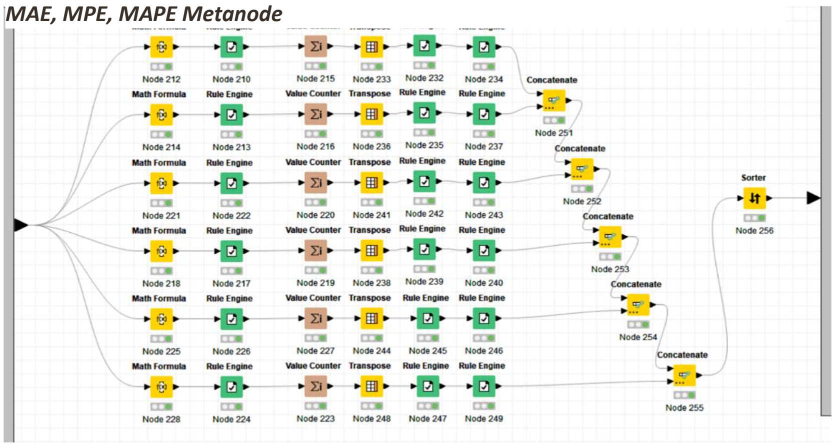
*Result Preparation Metanode*

Inside this metanode, we produced an interactive visualization of the most important statistics of the regression. Other than basic statistics such as Mean Squared Error, R-squared, RMSE, RMSLE, Residual Deviances and other important metrics like MAPE, MAE, MPE, we also included a histogram of the distribution of the error in the prediction which is key to evaluate our model.



*MAE, MPE, MAPE Metanode*

In fact, as we have elaborated on in the managerial implications, we also need to consider the bias of our model. We want to give the most precise fare prediction possible, but it is inevitable that our model will not be perfect. Overestimating and underestimating can both, in different ways, harm the bottom line.



*Over/Underestimate Metanode*

We also produced a table reporting the relative frequencies of fare_values under/overestimated by more than 10,20 or 30%. This was particularly useful because our goal was to reduce to the minimum how often we get significantly wrong estimations. In this sense you may ask why not to rely on the analysis of MPE. In our case this measure was not particularly useful. In fact, the average % error was influenced by big mistakes while we are more concerned that we get an estimation close to reality for a big portion of our test set rather than getting a smaller error for a greater portion of the entries. This is why the analysis of the relative distribution of errors results was much handier in this situation.

With all that said, we set-up the initial regressions/models:

- Baseline Linear Regression
- Baseline Random Forest → for which we arbitrarily chose 50 Trees with a max depth of 10 layers at each iteration
- Baseline Gradient Boosting → in this case we chose the same number of layers as the Random Forest, while we chose to use 75 trees and we also set a Learning Rate of 0.1

Given that the split between the test and train set is performed by the **Partitioning** node which uses a random seed, results vary for each run of the workflow, so those reported in the table below may not be the exact same every time but provides an indication of the accuracy of the initial models and how they improved after fine-tuning their parameters.

| | *MSE* | *R2* | *MPE* | *MAPE* | *MAE* | *Under-Estimated >10%* | *Under-Estimated > 20%* | *Under-Estimated > 30%* | *Over-Estimated > 10%* | *Over-Estimated > 20%* | *Over-Estimated > 30%* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *Initial Regression* | 15.311 | 0.865 | -0.071 | 0.1889 | 2.219 | 22.47% | 10.98% | 4.70% | 41.49% | 25.51% | 14.12% |
| *Initial Random Forest* | 11.477 | 0.898 | -0.076 | 0.1869 | 2.019 | 21.07% | 9.99% | 4.12% | 41.31% | 25.13% | 14.18% |
| *Initial Gradient Boosting* | 10.413 | 0.908 | -0.046 | 0.1530 | 1.770 | 20.16% | 8.94% | 3.69% | 34.99% | 17.06% | 7.40% |

We have observed that the Gradient Boosting and the Random Forest are better algorithms than the Linear Regression. In particular Gradient Boosting had a slight edge on all of them. This is not very clear if we just look at the R2 or the MSE, but it became much more evident taking a look at the measure of Avg. % error (MPE), or avg. % of absolute error (MAPE). This translated to a much
more concentrated distribution of the errors for the Gradient Boosting, as we can see from the second section of the Table. We can notice in particular how the percentage of instances for which the fare_value was underestimated by over 30% drops from 4.12% to 3.69% with a ~11% reduction.

Looking at the results of the MAPE, which are above the recommended 10% threshold for managerial use, we were a little disappointed and therefore tried to further improve our results. We did this by fine-tuning the parameters of our Random Forest Model and Gradient Boosting Algorithm.

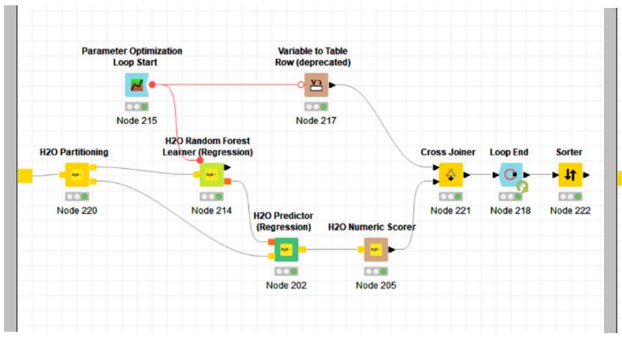
To do so, we set up 2 loops to compare how the two algorithms performed for various values of their parameters. Given the limitations posed by the available computation power we could not run the optimization loops on a cross validation. Instead, we used an **H2O Partitioning** node to split our train set 70/30 and use the smaller set as a test set for the loops. At each iteration, the loop would change the following Random Forests hyper parameters:

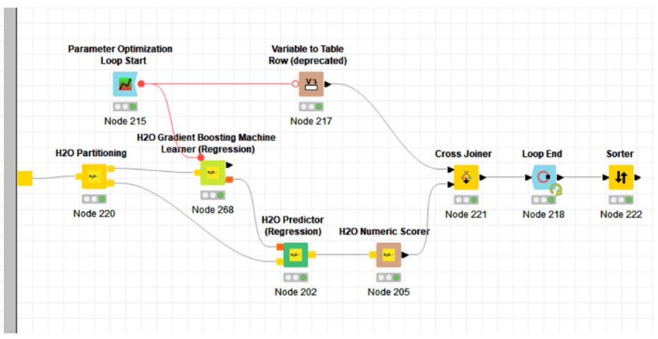
- the number of trees (ranging from 40 to 100, with a stepsize of 5)
- the max depth of the trees (ranging from 10 to 20)

In the Gradient Boosting, in addition to that, we also compared:

- the learning rate (ranging from 0.01 to 0.1, with step sizes of 0.05)

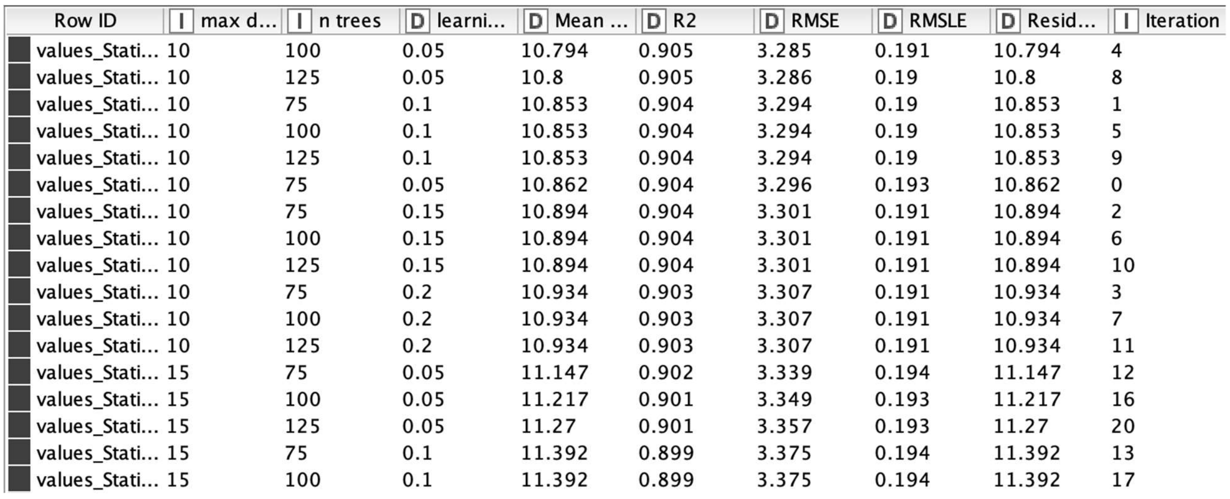
Here we can see the content of the metanodes containing the loops which look for the optimal parameters of the two models.




Parameter Optimization Loop Start
Node 215
Variable to Table Row (deprecated)
Node 217
H2O Partitioning
Node 220
H2O Gradient Boosting Machine Learner (Regression)
Node 268
H2O Predictor (Regression)
Node 202
H2O Numeric Scorer
Node 205
Cross Joiner
Node 221
Loop End
Node 218
Sorter
Node 222

At the end of the loops the sorter node returned a table with the parameters chosen at each iteration and the effectiveness of such model (tested on 30% of the training data). Effectiveness was measured in terms of MSE, R2 and other parameters included in the **H2O scorer**, such as RMSE and RMSLE. Here below we report an example of the tables produced by the 2 loops:

Here is the results returned by the Gradient Boosting Algorithm's loop*:



| Row ID | I max d... | I n trees | D learni... | D Mean ... | D R2 | D RMSE | D RMSLE | D Resid... | I Iteration |
|---|---|---|---|---|---|---|---|---|---|
| values_Stati... | 10 | 100 | 0.05 | 10.794 | 0.905 | 3.285 | 0.191 | 10.794 | 4 |
| values_Stati... | 10 | 125 | 0.05 | 10.8 | 0.905 | 3.286 | 0.19 | 10.8 | 8 |
| values_Stati... | 10 | 75 | 0.1 | 10.853 | 0.904 | 3.294 | 0.19 | 10.853 | 1 |
| values_Stati... | 10 | 100 | 0.1 | 10.853 | 0.904 | 3.294 | 0.19 | 10.853 | 5 |
| values_Stati... | 10 | 125 | 0.1 | 10.853 | 0.904 | 3.294 | 0.19 | 10.853 | 9 |
| values_Stati... | 10 | 75 | 0.05 | 10.862 | 0.904 | 3.296 | 0.193 | 10.862 | 0 |
| values_Stati... | 10 | 75 | 0.15 | 10.894 | 0.904 | 3.301 | 0.191 | 10.894 | 2 |
| values_Stati... | 10 | 100 | 0.15 | 10.894 | 0.904 | 3.301 | 0.191 | 10.894 | 6 |
| values_Stati... | 10 | 125 | 0.15 | 10.894 | 0.904 | 3.301 | 0.191 | 10.894 | 10 |
| values_Stati... | 10 | 75 | 0.2 | 10.934 | 0.903 | 3.307 | 0.191 | 10.934 | 3 |
| values_Stati... | 10 | 100 | 0.2 | 10.934 | 0.903 | 3.307 | 0.191 | 10.934 | 7 |
| values_Stati... | 10 | 125 | 0.2 | 10.934 | 0.903 | 3.307 | 0.191 | 10.934 | 11 |
| values_Stati... | 15 | 75 | 0.05 | 11.147 | 0.902 | 3.339 | 0.194 | 11.147 | 12 |
| values_Stati... | 15 | 100 | 0.05 | 11.217 | 0.901 | 3.349 | 0.193 | 11.217 | 16 |
| values_Stati... | 15 | 125 | 0.05 | 11.27 | 0.901 | 3.357 | 0.193 | 11.27 | 20 |
| values_Stati... | 15 | 75 | 0.1 | 11.392 | 0.899 | 3.375 | 0.194 | 11.392 | 13 |
| values_Stati... | 15 | 100 | 0.1 | 11.392 | 0.899 | 3.375 | 0.194 | 11.392 | 17 |

** we included a screenshot only of the best results for a matter of space*

As we can see from the table above the optimal parameters for the Gradient Boosting are:

- Learning Rate = 0.05
- Number of Trees = 100
- Max depth of each tree = 10

Which are very close to the original values, so that we did not expect much different results in the improved model.

In Random Forest's case, instead, the optimal parameters are:

- Number of Trees in the forest = 100
- Maximum depth of each tree = 15

We used these optimal parameters to train new models. To do so we used similar metanodes to the ones described above for the Baseline Models, the only difference being the parameters values corresponding to those of the **H2O Learner** nodes. The new metanodes were called respectively **Final Random Forest** and **Final Gradient Boosting**.

Here below we present a comparison of the old models with the new ones. The values are again extracted from the interactive view returned at the end of the models' metanodes:

| | *MSE* | *R2* | *MPE* | *MAPE* | *MAE* | *Under-Estimated >10%* | *Under-Estimated > 20%* | *Under-Estimated > 30%* | *Over-Estimated > 10%* | *Over-Estimated > 20%* | *Over-Estimated > 30%* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *Initial Regression* | 15.311 | 0.865 | -0.071 | 0.1889 | 2.219 | 22.47% | 10.98% | 4.70% | 41.49% | 25.51% | 14.12% |
| *Initial Random Forest* | 11.477 | 0.898 | -0.076 | 0.1869 | 2.019 | 21.07% | 9.99% | 4.12% | 41.31% | 25.13% | 14.18% |
| *Initial Gradient Boosting* | 10.413 | 0.908 | -0.046 | 0.1530 | 1.770 | 20.16% | 8.94% | 3.69% | 34.99% | 17.06% | 7.40% |
| *Final Random Forest* | 11.410 | 0.899 | -0.074 | 0.1854 | 2.008 | 21.23% | 10.06% | 4.10% | 40.89% | 24.66% | 13.77% |
| *Final Gradient Boosting* | 10.413 | 0.908 | -0.046 | 0.1530 | 1.770 | 20.16% | 8.94% | 3.69% | 34.99% | 17.06% | 7.40% |

As we can see the results of th Random Forest model improved, but not by much. Most importantly, the number of instances for which we underestimated the fare value drops for any % level considered. Therefore, the updated models are able to predict more precisely the fare amount of each ride.

The improved models confirm the trends seen in the Baseline ones. Gradient Boosting again proved to be the most effective model. Its final version presents underestimates less than 9% of the observation by over 20% and just 3.7% by over 30%. It's important to notice that the model is still slightly biased toward underestimating the fare value, which we can see by looking at the MPE value. Nevertheless, its value is the smallest of all models. The R2 is 0.908 which in general describes quite an efficient model, but looking at the MAPE, we can see it is still above the 10% threshold.

We conclude the analysis of the results by looking at the last element in the interactive view of the results of the model. The importance of each variable in the model is considered here. In the Linear Regression's case it is expressed by the coefficient and p-value assigned to every single variable. While in the other 2 models it is expressed as the number of times that particular variable is used to split the tree.

We can see that in the Linear regression the number of variables listed is very high because, as we said, it creates a random variable for each entry of the categorical variables. Nevertheless, the number of variables for which the coefficient is significantly different from zero is quite low.

Instead, the Random Forest and Gradient Boosting provide interesting insight on how crucial each of the constructed variables is in determining the amount of each fare. In particular we can see that the value of distance variable, pickup_area and dropoff_area are the most used by a good margin. This confirms our expectations that actual distance would be the most crucial element to calculate fare value.

Of course, the final results of our model are not completely satisfactory, in particular if we consider the MAPE threshold. Using the whole 55M entries dataset would probably resulted in much better results. Unfortunately, computational power limitation forced us to settle for a much smaller subset than the original dataset.

The idea at the basis of our model was to get the actual driving distance between the pick-up and the drop-off locations. We tried that by writing a custom Python code, using the APIs of Open Street Map, to return the driving distance as a variable to be added to the dataset. Again, this was too complex to handle and it would have taken days to calculate even 1 million entries like we had in the reduced dataset. So, we fell back on the simpler solution presented above.

Still without calculating the actual driving distance we could have obtained a very good proxy by introducing an interaction effect between the distance (which is calculated as the crow flies) and the route of the ride (given the areas of drop-off and pick-up). But again, the huge number of categories of the routes (over >60k) made this an impossible task.

# Managerial Implications

Uber and Lyft have managed to successfully use technology to gain a competitive advantage over traditional taxis. The popularity of ride-hailing apps cannot be underestimated. In fact, the data fully supports their importance, as: "In January – before the pandemic – ride-hailing apps accounted for 75% of taxi trips in New York City. By April, that figure had risen to 94%, with Uber representing 67.6% of all trips and Lyft accounting for a further 24.9%."[3] We can reasonably assume that once the pandemic passes, the percentage of taxi trips originating from a ride-hailing app will fall back to January levels, but nevertheless this is a very significant percentage. NYC's yellow cabs have followed the trend and developed their own app, but as the data shows it lags behind in terms of popularity.

A part of the problem that taxis are facing is that Uber is able to provide the exact cost of the ride and give the user a take it or leave it deal, which unfortunately for the regulated taxi industry and all holders of the yellow cab medallion, which used to cost 1 million dollars but has fallen in value to around 200 thousand, is often taken[4]. As stated, the taxi industry is regulated, unlike Uber and Lyft, so providing this kind of take it or leave it deal becomes far more complicated, in part due to the lower overall reliance on technology taxis can permit and more importantly ride fares themselves cannot be deregulated in the way Uber's are. If they were, customers that picked up a cab at a station or one passing by would be inclined to bargain over the cost of the fare. Some New Yorkers might actually enjoy this type of bartering, but it would be highly inefficient and create unnecessary clutter for all involved. This is before mentioning the fact that deregulating the taxi industry would devalue the medallions completely, destroying the lives of a large amount of taxi drivers still paying their medallion off. We are seemingly left without many options, but there is a solution that can bridge the gap between taxis and Uber.

This solution is to provide better fare predictions using our model and improve the user-friendliness and capacities of the app. Having accurate and reliable fare predictions will help retain loyal customers and attract new customers, especially tech-reliant younger generations and tourists that rely very much on technology to get around and see new places. This would be significant because a large share of NYC's taxi revenue comes from tourists, which is further reinforced by the significant struggle this year has been for taxis in NYC.[5] It can be more comfortable for tourists to rely on technology to hail an Uber and not worry about where they can find a taxi or if, perhaps, some taxi driver wants to take advantage

[3] https://www.investopedia.com/articles/personal-finance/021015/uber-versus-yellow-cabs-new-york-city.asp

[4] https://www.washingtonpost.com/gdpr-consent/?next_url=https%3a%2f%2fwww.washingtonpost.com%2fbusiness%2feconomy%2fuber-lyft-and-the-hard-economics-of-taxi-cab-medallions%2f2019%2f05%2f24%2fcf1b56f4-7cda-11e9-a5b3-34f3edf1351e_story.html

[5] **https://www.nytimes.com/2020/11/12/nyregion/nyc-taxi-drivers-coronavirus.html**

of their lack of knowledge and take them for a longer than necessary ride. Having the knowledge of how much their fare will approximately cost with a taxi, eliminating the possibility of a driver adding extra kilometers to the ride, and also having the quality assurance and comfort of the yellow NYC cab can actually restore taxi's comparative advantage.

At the moment, tourists and other customers cannot get advanced estimates for taxi fares, and the NYC Taxi and Limousine Commission's official stance is that "it is impossible to pre-calculate a fare, because the meter rate depends on traffic, construction, weather, and route to the destination."[6] While it is true that the fare depends on many factors, our model is able to give reliable results that are more than just a ballpark estimate. This is important because when it comes to these fares, the variability is usually not great, so a reliable model can be trusted by customers even if it is not perfect. One additional factor to consider here is the effect of slight overestimation compared to slight underestimation of the fares. By consistently overestimating, the taxi industry might lose customers to competitors if they see lower prices available. However, customer satisfaction and loyalty after rides might increase if customers pay lower prices than predicted. This might not have much of a long-term effect on some of the clientele, take tourists for example, but the positive word of mouth could definitely be helpful for the taxi industry, especially considering the numerous blogs and social media posts related to travel around NYC. On the other hand, underestimating can lead to an increased risk of making customers angry and making them feel as if they were misled. Nevertheless, this is not likely to occur if the underestimation is minor, and a potential positive of slightly underestimating would be luring customers in with this appealing fare prediction, even if it comes with the possible cost of customer satisfaction decreasing and negative word of mouth spreading about the app. Ultimately, the decision would fall on the manager regarding which bias to favor, or to simply confide with our model, which slightly underestimates the fares. This decision should be based on all available information related to the analysis of the potential effects, which could best be determined after trialing both.

An important aspect, slightly beyond the scope of our analysis and model, would be combining the fare predictions with an improved user interface and adding payment options. This should bring NYC's taxi app to the level of Uber, with the sole exception being that it would not be possible to pay for a ride before it happens. Regarding the payment options, as stated it would not be possible to pay for a ride in advance, but it would be possible to incorporate a PayPal account with the app or add a credit card payment section in order to have the option of paying via the app or directly to the driver when the ride has ended and the fare is finalized. Linking an improved app with reliable fare predictions to the aura and reliability of NYC taxi has the potential to completely change the tide and give taxis a lifeline.

[6] https://citymonitor.ai/transport/uber-lyft-rides-during-coronavirus-pandemic-taxi-data-5232

# Knime Workflow

- Our Workflow can be downloaded at the following link:

  https://bocconi.sharepoint.com/:u:/s/DatabaseGroup/EV3L258YwLJDmOBlU8GkYI8BH8_FXh0PmF2vnwi_uQkaHA?e=XLA4eI

- We have installed the following extensions:
  - Knime JavaScript Views
  - Palladian → installed through NodePit
  - Knime Python Integration
  - Knime H2O Machine Learning Integration

To speed up the computation we have modified the file knime.ini to enable Knime to use at least 6GB of RAM To speed up the computation we have modified the file knime.ini to enable Knime to use at least 6GB of RAM

*P.S. To save space we uploaded a version of the Workflow partially to be run*