MIS 584 Project Final Report

New York City Taxi Fare Prediction

<u>Team 12</u>

Yash Jethmalani
Abhimanyu Paldiwal
Zhuo Huang

Table of Contents

List of Tables

Table of Figures

Abstract

Ridesharing apps like Uber and Lyft can give an estimated fare for the ride before the ride starts. Traditional taxi services cannot provide such a prediction and rely on distance and wait time for calculating their fares using a meter. Therefore, we investigate New York City's taxi fare data from Kaggle and assist future customers of traditional cab services by estimating their fare amount using machine learning. Our predictions are made by analyzing fare amount, passenger count, geopoints for the passengers' pickup and dropoff location, and their equivalent distances. We also evaluate a number of different algorithms like Linear Regression, XGBoost Regression, and Random Forest Regression to arrive at the optimal algorithm for the prediction.

1. Goal

In this project, our team tries to tackle the uncertainty of riding traditional cab services to provide the users an estimated fare based on their destination before their selection of calling Uber or Lyft. The user can then compare the estimated taxi fare with fares provided by Uber and Lyft to make a data-driven decision for their transport. We focus on New York as our pilot city to create the model as the data is readily available, and the fare prices are moderately high compared to other states.

2. Relevant Prior Work

Before starting the project, we have researched some of the previous work related to this project for our overall work plan.

Year	**Author**	**Methods**
2018	Albert van Breemen	Linear Regression Model
2018	Madhuri Sivalenka	Random Forest, GBM, XGBoost
2018	Emanuel Kamali	Multilinear Regression Model, Random Forest, GBM

Table 1: Relevant Prior Work

3. Instance Configuration

3.1. Hardware

We are using a Google Compute VM instance with a configuration of 32 GB memory, 8 CPUs, and SSD for dataset storage. The instance was created with such high RAM and CPU to evaluate distributed dataframes like Dask, but eventually, pandas dataframe turned out to be the right choice for this project.

3.2. WebServer

We have set up an Nginx web server to host the notebook for shared access. The notebook is accessible through https://yash.ai. The webserver acts as a proxy to the notebook server, which handles the authentication.

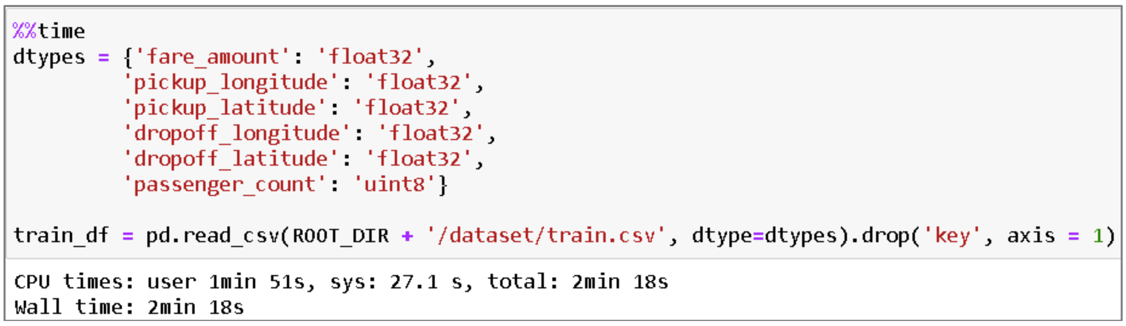
3.3. Notebook

The notebook is installed using miniconda on the instance along with all the packages. The notebook is run using the notebook configuration file, which includes relevant parameters for running the server, including but not limited to the host, port, encrypted password, and remote access.

4. Memory Optimization

To import a CSV file that contains approximately 55 million records, we had to downcast the datatypes of the columns from float64 to float32. Doing this for all the relevant columns reduces the dataframe memory usage by half. Consequently, the CSV file can now be read in under two and a half minutes!

Note: The practical limit of GPS surveying precision is seven (7) decimal degrees, and the precision for float32 is eight (8) (Decimal Degrees, 2020). Therefore, down-casting datatypes from float64 to float32 will not affect the accuracy of the latitudes and longitudes present in the dataset. (Measuring Accuracy of Latitude and Longitude, 2017)



```
%%time
dtypes = {'fare_amount': 'float32',
          'pickup_longitude': 'float32',
          'pickup_latitude': 'float32',
          'dropoff_longitude': 'float32',
          'dropoff_latitude': 'float32',
          'passenger_count': 'uint8'}

train_df = pd.read_csv(ROOT_DIR + '/dataset/train.csv', dtype=dtypes).drop('key', axis = 1)
```

```
CPU times: user 1min 51s, sys: 27.1 s, total: 2min 18s
Wall time: 2min 18s
```

Figure 1: Data Loading Time

5. Dataset Description

The dataset was sourced from Kaggle's New York City Taxi Fare Prediction competition (New York City Taxi Fare Prediction, 2020). It contains approximately 55.4 million rows with a total size of 5.31 GB and includes NYC taxi trip data from 2009 to 2015.

The details of 8 columns in the dataset are as follows:

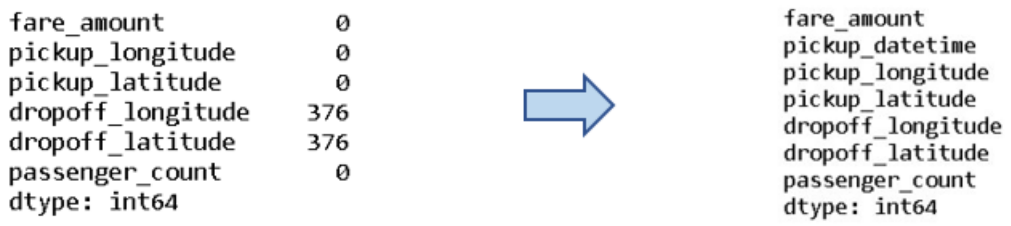
Column	**Data Type**	
Order date (key)	datetime	Identifies each row in both the training and test sets
Fare amount (predicted)	float	Dollar amount of the cost of the taxi ride
Pickup datetime	datetime	Value indicating when the taxi ride started
Pickup longitude	float	Longitude coordinate of where the 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	Indicates the number of passengers in the taxi ride

Table 2: Dataset Description

6. Data Preprocessing

6.1. Null Values

It is important to check for null values in the dataset because it can adversely affect machine learning algorithms' performance. In the dataset, we have 376 null values each for **dropoff_longitude** and **dropoff_latitude** attributes. Therefore, we remove the null values using the **pandas.DataFrame.dropna** function.



```
fare_amount            0
pickup_longitude       0
pickup_latitude        0
dropoff_longitude    376
dropoff_latitude     376
passenger_count        0
dtype: int64
```

```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

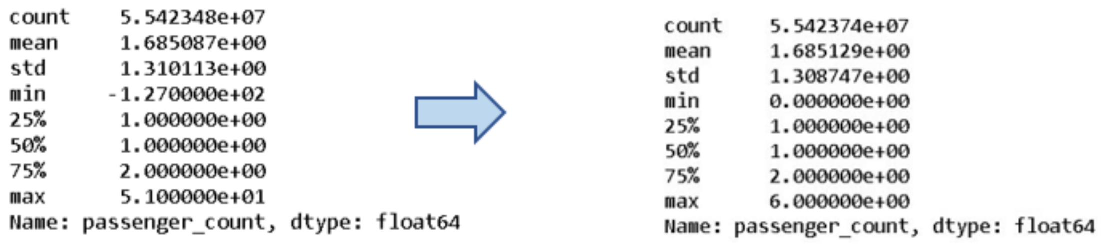
Figure 2: Removing Null Values

6.2. Outliers

6.2.1. Passenger Count

According to the **NYC Taxi Driver Rule 54-15(g) Chapter 54 – Drivers of Taxicabs and Street Hail Liveries**, the maximum amount of passengers allowed in a yellow taxicab by law is four (4) in a four (4) passenger taxicab or five (5) passengers in a five (5) passenger taxicab, except that an additional passenger must be accepted if such a passenger is under the age of seven (7) and is held on the lap of an adult passenger seated in the rear. (Passenger Frequently Asked Questions, 2020) *(Please see figure 17 in the Appendix)*

In the dataset, the minimum and maximum value for **passenger_count** attribute is **-127** and **510**, respectively. We need to prune our dataset with values that adhere to the guidelines laid out by the NYC Taxi & Limousine Commission. Therefore, we remove **passenger_count** values that are greater than six (6). *(Please see figure 13 and figure 14 in the Appendix)*



```
count    5.542348e+07
mean     1.685087e+00
std      1.310113e+00
min     -1.270000e+02
25%      1.000000e+00
50%      1.000000e+00
75%      2.000000e+00
max      5.100000e+01
Name: passenger_count, dtype: float64
```

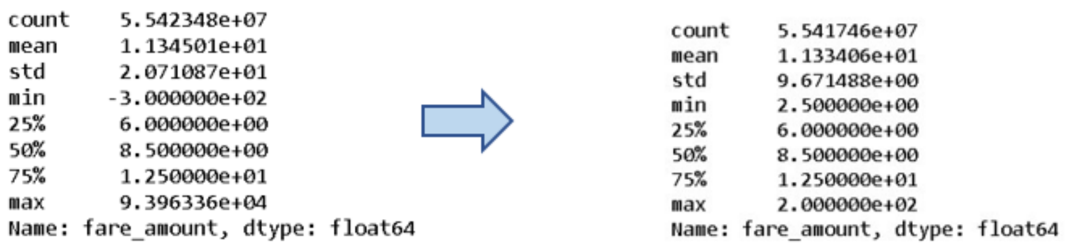
```
count    5.542374e+07
mean     1.685129e+00
std      1.308747e+00
min      0.000000e+00
25%      1.000000e+00
50%      1.000000e+00
75%      2.000000e+00
max      6.000000e+00
Name: passenger_count, dtype: float64
```

Figure 3: Removing Outliers for Passenger Count attribute

6.2.2. Fare Amount

According to the **NYC Taxi Standard Metered Fare**, a base charge of $2.5 is levied on every passenger regardless of the distance traveled during the trip. (Taxi Fare, 2020) *(Please see figure 18 in the Appendix)*

Similarly, the minimum value for the **fare_amount** attribute is **-300,** which doesn't satisfy the criteria laid out by the commission. Therefore, we subset the dataframe with **fare_amount** values greater than or equal to **$2.5**. *(Please see figure 15 and figure 16 in the Appendix)*



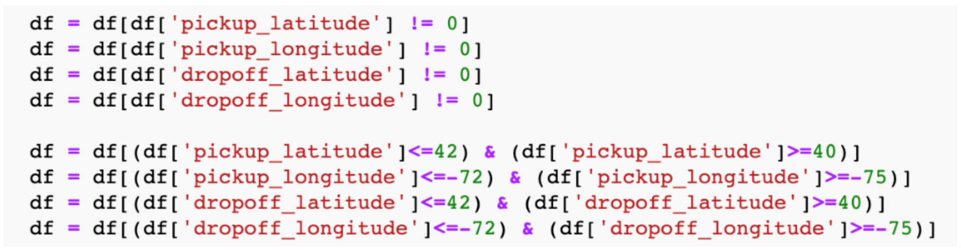
```
count      5.542348e+07
mean       1.134501e+01
std        2.071087e+01
min       -3.000000e+02
25%        6.000000e+00
50%        8.500000e+00
75%        1.250000e+01
max        9.396336e+04
Name: fare_amount, dtype: float64
```

```
count     5.541746e+07
mean      1.133406e+01
std       9.671488e+00
min       2.500000e+00
25%       6.000000e+00
50%       8.500000e+00
75%       1.250000e+01
max       2.000000e+02
Name: fare_amount, dtype: float64
```

Figure 4: Removing Outliers for Fare Amount attribute

6.2.3. Geolocation

The latitudinal and longitudinal range of New York City is approximately at (40,-74). So, we removed values that were close to zero or values that were deviating from the norm. We also decided that all the observations that lie outside the interval formed by 2.5 and 97.5 percentile will be considered potential outliers. After combining these two conditions, we can justify the new upper and lower bounds for latitudes and longitudes, i.e., latitudinal range will be between 40 and 42, whereas longitudinal range will be between -72 and 75 for both the pickup and dropoff geopoints.



```
df = df[df['pickup_latitude'] != 0]
df = df[df['pickup_longitude'] != 0]
df = df[df['dropoff_latitude'] != 0]
df = df[df['dropoff_longitude'] != 0]

df = df[(df['pickup_latitude']<=42) & (df['pickup_latitude']>=40)]
df = df[(df['pickup_longitude']<=-72) & (df['pickup_longitude']>=-75)]
df = df[(df['dropoff_latitude']<=42) & (df['dropoff_latitude']>=40)]
df = df[(df['dropoff_longitude']<=-72) & (df['dropoff_longitude']>=-75)]
```

Figure 5: Applying NYC Latitude and Longitude Bounds

6.3. Feature Engineering

Based on our intuition, algorithms using latitude and longitude may not accurately infer the distances involved in the fare prediction. Therefore, we computed the absolute latitude and longitude distances to give the algorithms some distance related information to aid our predictions. To improve our prediction, we found that we also need to include real-world distances between the pickup and dropoff geopoints in "miles". Consequently, we engineered additional features – Manhattan, Euclidean, and Haversine distances.

6.4. Dataset for Prediction

After preprocessing the dataset, our algorithm implementation stage relies on the features shown below:

Column	**Data Type**
Fare amount (predicted)	float
Passenger count	integer
Absolute longitude difference	float
Absolute latitude difference	float
Manhattan Distance	float
Euclidean Distance	float
Haversine Distance	float

Table 3: Final dataset description

6.5. Binning

Before we split the dataset, we compute a temporary feature called "*fare_bin*" that categorizes the fare amount in the range of \$0 - \$45 into bins of \$5, and everything above \$45 is labeled "45+." Binning allows us to group the outliers in the \$45+ range, the significance of which will be explained in the next section.

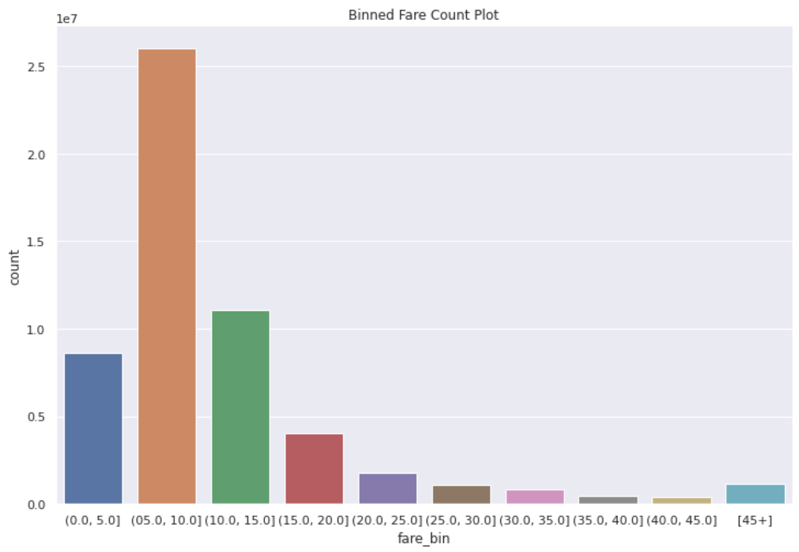


Figure 6: Binned Fare Amount for Stratification

6.6. Stratification

We have partitioned the dataset with a 70/30 split, i.e., 70% for training and 30% for testing. Additionally, we have stratified the split using the "*fare_bin*" feature. This distributes the fare amounts above \$45, ensuring that the distribution of outliers is uniform across our training and testing data.

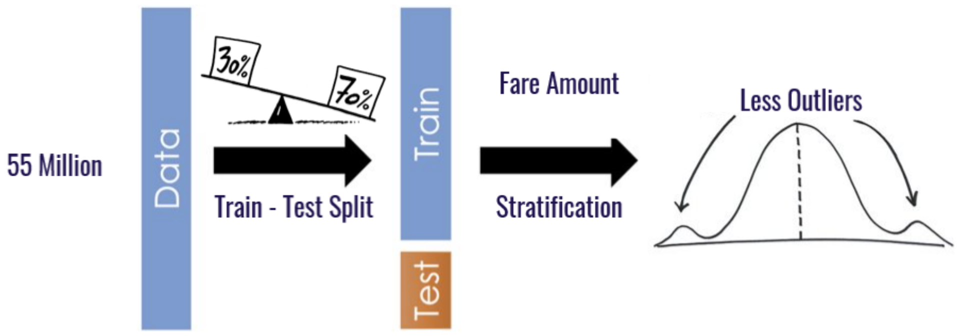


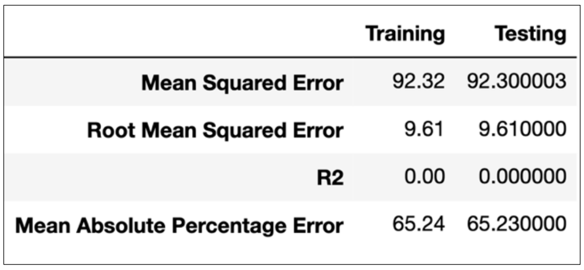
Figure 7: Train-Test split

7. Algorithm Implementation

In order to find the best algorithm to fit our prediction model, our primary evaluation metrics for this stage are Root Mean Squared Error (RMSE), and the coefficient of determination R^2. We have also considered the intercept and the coefficients of the feature used for prediction because the RMSE values are very similar in all of the following cases. Additionally, we chose three common features (passenger count, absolute latitude distance, absolute longitude distance) along with Manhattan, Euclidean, and Haversine distances to determine the correct fit for our model. We chose Linear Regression as our initial model and started with a bottom-up approach to find the right set of features.

7.1. Naïve Baseline

In order to understand how the mean of fare amount would perform as a naïve predictor, we calculated the baseline with the three common features (passenger count, absolute latitude distance, absolute longitude distance) using Scikit-Learn's *DummyRegressor*. The dummy model gave us an RMSE of 9.61, which is very high, and R^2 of 0, which gave us little to no intuition for the predictions.

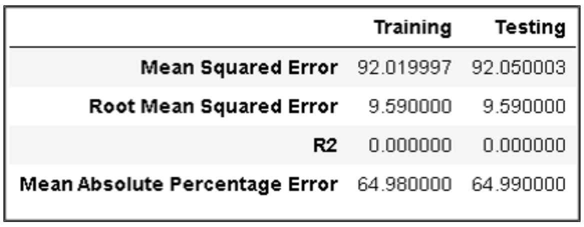


	Training	Testing
Mean Squared Error	92.32	92.300003
Root Mean Squared Error	9.61	9.610000
R2	0.00	0.000000
Mean Absolute Percentage Error	65.24	65.230000

Figure 8: Evaluation Metrics – DummyRegressor

7.2. Old Baseline

Since the mean strategy with the DummyRegressor failed to provide any meaningful intuition for the prediction, we decided to go forward with Linear Regression with the same three common features to serve as our baseline. Our initial run of the Linear Regression model gave us an RMSE of 9.59 and an R^2 score of 0. This indicated some serious errors in preprocessing and/or multicollinearity issues.



	Training	Testing
Mean Squared Error	92.019997	92.050003
Root Mean Squared Error	9.590000	9.590000
R2	0.000000	0.000000
Mean Absolute Percentage Error	64.980000	64.990000

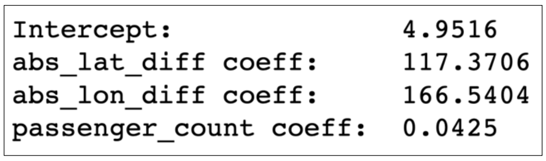
Figure 9: Old Baseline – Linear Regression w/o NYC Lat-Long Bounds

Note: the old baseline did not consider the NYC area bounds for the pickup and dropoff latitudes and longitudes

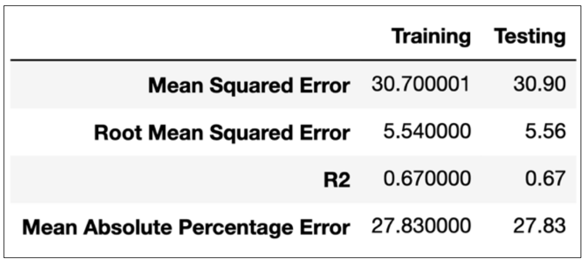
7.3. New Improved Baseline

After investigating the features in-depth, we realized that while we ensured that the latitude and longitude values for pickup and dropoff geopoints were valid, we were not eliminating geopoints outside NYC. Therefore, we decided to only consider geopoints within the NYC bounds as seen in section *6.2.3 Geolocation*. This improved the performance significantly.

The R^2 increased from 0 to 0.67, and the RMSE value reduced from 9.59 to 5.56. This model can now serve as a baseline as we add more features and compare the performance of different algorithms.



```
Intercept:              4.9516
abs_lat_diff coeff:     117.3706
abs_lon_diff coeff:     166.5404
passenger_count coeff:  0.0425
```



	Training	Testing
Mean Squared Error	30.700001	30.90
Root Mean Squared Error	5.540000	5.56
R2	0.670000	0.67
Mean Absolute Percentage Error	27.830000	27.83

Figure 10: New Baseline – Linear Regression with 3 Features

7.4. Linear Regression

Usually, in a bottom-up approach in regression, the analysis starts with a single feature, and additional features are added and compared. For example, we could start with passenger count as the first feature and add absolute distance features in increments, but intuition dictates that the cab's fare will be dependent on both passenger count and distance of the trip. Hence, we opted to start with our 3 features and not with univariate analysis.

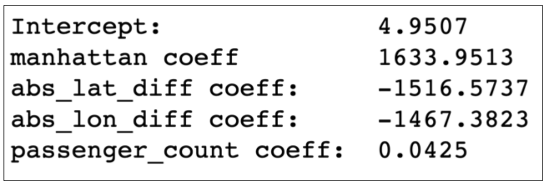
As seen in section 7.3 New Improved Baseline, the three-feature implementation of Linear Regression is an example of this approach. Henceforth, distance features are added to 3 existing features listed below to improve the performance of the Linear Regression algorithm:

- absolute distance between pickup and dropoff latitudes
- absolute distance between pickup and dropoff longitudes
- passenger count

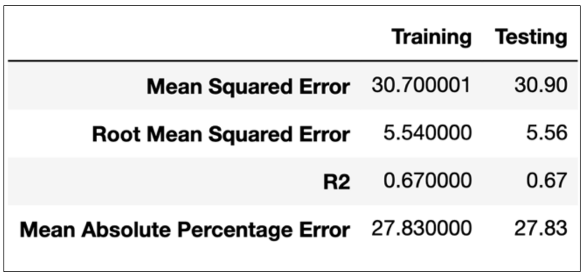
7.4.1. Linear regression: Manhattan

In our first scenario, we add Manhattan to our improved baseline Linear Regression model. Looking at the training performance, intercept and passenger count coefficients remain almost the same as before, but the distance values have varied significantly. We have a Manhattan distance coefficient of 1633.95, which indicates that it is very positively influencing the fare amount. Looking at the coefficients of absolute differences between latitude (-1516.57) and longitude (-1467.38) leads us to believe that the absolute differences and Manhattan distances are balancing out, and there may be multicollinearity issues.

Although the coefficient for the features has changed, the RMSE (5.56) and R^2 values (0.67) have remained the same, which shows that adding Manhattan has not improved the accuracy of the model.



```
Intercept:              4.9507
manhattan coeff         1633.9513
abs_lat_diff coeff:     -1516.5737
abs_lon_diff coeff:     -1467.3823
passenger_count coeff:  0.0425
```



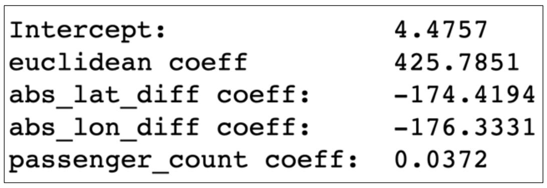
	Training	Testing
Mean Squared Error	30.700001	30.90
Root Mean Squared Error	5.540000	5.56
R2	0.670000	0.67
Mean Absolute Percentage Error	27.830000	27.83

Figure 11: Linear Regression: Manhattan – Evaluation Metrics

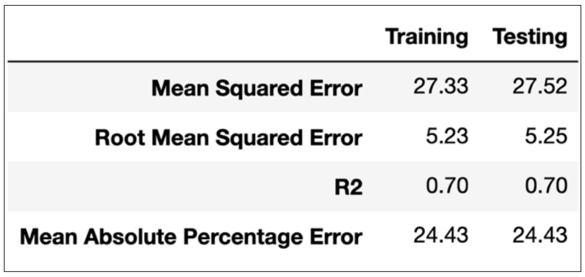
7.4.2. Linear regression: Euclidean

Since Manhattan did not yield favorable results, we replace Manhattan distance with Euclidean distance. The initial 3 features remain the same.

After training the model, we see that the intercept value has changed to 4.47, a marginal decrement, and the passenger count coefficient has also changed a bit. We also see that there are reductions in both the intercept and the passenger count co-efficient. The Euclidean co-efficient has a value of 425.78, and the corresponding absolute latitude and longitude differences show a negative value of -174.42 and -176.33, respectively. We can infer that model is now giving precedence to the Euclidean distance for the prediction, but the multicollinearity remains. The improvement carries on to the RMSE and R^2 scores, which increase to 5.25 and 0.70, respectively. Euclidean is performing better than Manhattan in all of our evaluation metrics.



```
Intercept:               4.4757
euclidean coeff          425.7851
abs_lat_diff coeff:      -174.4194
abs_lon_diff coeff:      -176.3331
passenger_count coeff:   0.0372
```



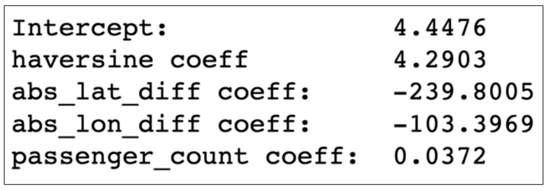
	Training	Testing
Mean Squared Error	27.33	27.52
Root Mean Squared Error	5.23	5.25
R2	0.70	0.70
Mean Absolute Percentage Error	24.43	24.43

Figure 12: Linear Regression: Euclidean

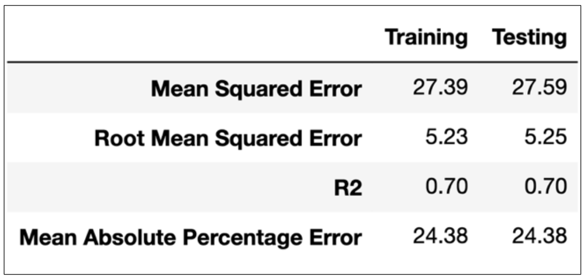
7.4.3. Linear regression: Haversine

To see if we can improve the Linear Regression model more, we use Haversine distance instead of Euclidean distance with the original 3 features. We see that the RMSE and R^2 have improved over Manhattan, but the values remain similar to that of Euclidean. The only change in performance metrics is the Mean Absolute Percentage Error, which is reduced from 24.43 to 24.38.

From the initial evaluation, the performance of Haversine distance seems to be marginally better than the Euclidean.



```
Intercept:               4.4476
haversine coeff          4.2903
abs_lat_diff coeff:      -239.8005
abs_lon_diff coeff:      -103.3969
passenger_count coeff:   0.0372
```

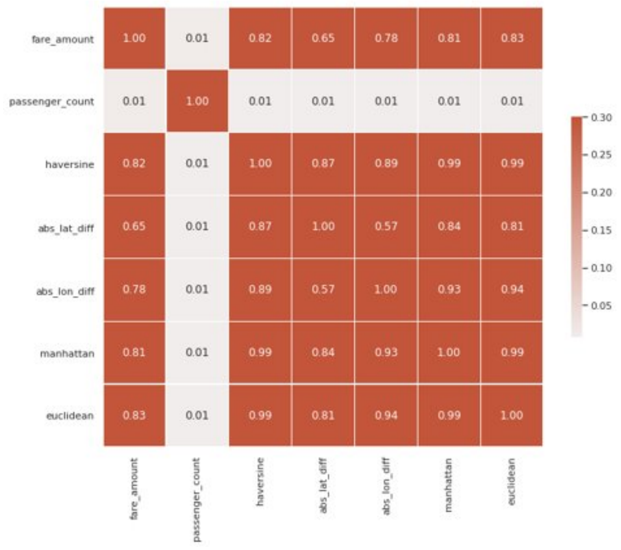


	Training	Testing
Mean Squared Error	27.39	27.59
Root Mean Squared Error	5.23	5.25
R2	0.70	0.70
Mean Absolute Percentage Error	24.38	24.38

Figure 13: Linear Regression: Haversine – Evaluation Metrics

7.5. Multicollinearity

Upon further investigation of the feature engineered dataset, we confirmed the multicollinearity between the absolute latitude and longitude differences and our calculated Manhattan, Euclidean, and Haversine distances. This is evident from the correlation matrix given below.



	fare_amount	passenger_count	haversine	abs_lat_diff	abs_lon_diff	manhattan	euclidean
fare_amount	1.00	0.01	0.82	0.65	0.78	0.81	0.83
passenger_count	0.01	1.00	0.01	0.01	0.01	0.01	0.01
haversine	0.82	0.01	1.00	0.87	0.89	0.99	0.99
abs_lat_diff	0.65	0.01	0.87	1.00	0.57	0.84	0.81
abs_lon_diff	0.78	0.01	0.89	0.57	1.00	0.93	0.94
manhattan	0.81	0.01	0.99	0.84	0.93	1.00	0.99
euclidean	0.83	0.01	0.99	0.81	0.94	0.99	1.00

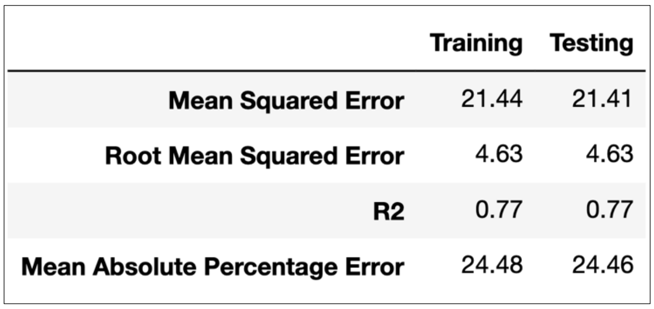
Figure 14: Correlation Matrix Showing Multicollinearity

Due to multicollinearity, our linear regression model's statistical power and precision of the estimate coefficients were affected. Therefore, we decided to use Ensemble-based methods to reduce the effects of multicollinearity on our prediction model.

7.6. Ensemble Algorithms

7.6.1. XGBoost Regression

Our first ensemble-based method is XGBoost Regression, which gives us a slight improvement on both of our RMSE and R^2 values. If we compare it to our baseline, our RMSE value reduced from 5.25 to 4.63, and our R^2 value increased from 0.70 to 0.77.



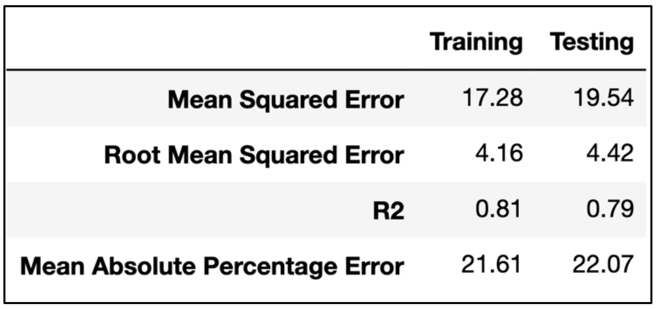
	Training	Testing
Mean Squared Error	21.44	21.41
Root Mean Squared Error	4.63	4.63
R2	0.77	0.77
Mean Absolute Percentage Error	24.48	24.46

Figure 15: XGBoost Evaluation Metrics

7.6.2. Random Forest

Our second ensemble-based method is Random Forest. If we compare it to our baseline, our RMSE value reduced from 5.25 to 4.42, and our R^2 value increase from 0.70 to 0.79.

We ran our Random Forest Regression model with a conservative 20 N-estimators and 20 max depth, which allowed us to have a large number of trees with deeper depth, capturing more details from the training set. Usually, a higher value for the max depth increases the chances for overfitting, but this is clearly not the case considering our R^2 score, and RMSE changed marginally between the training and the testing set as seen below. Consequently, the Random Forest Regression model fared even better than the XGBoost Regression model improving the R^2 value by 0.02 to 0.79 and reducing the RMSE value by 0.21 to 4.42



	Training	Testing
Mean Squared Error	17.28	19.54
Root Mean Squared Error	4.16	4.42
R2	0.81	0.79
Mean Absolute Percentage Error	21.61	22.07

Figure 16: Random Forest Evaluation Metrics

8. Result and Findings

After completing our preprocessing and model implementation, we can summarize all results in the table below.

Metrics	New Baseline	Linear Regression (Haversine)	XGBoost Regression	Random Forest Regression
MSE	3.90	27.59	21.41	19.52
RMSE	5.56	5.25	4.63	4.42
R^2	0.67	0.70	0.77	0.79
MAPE	27.83	24.38	24.46	22.07
Training Time	~ 3s	~ 4s	~ 17min	~ 22min
Prediction Time	~ 2s	~ 3s	~ 14s	~ 1min30s

Table 4: Performance Evaluation

Looking at the table above, we can see improvements in all of our performance metrics from our new baseline to Random Forest. R^2 score improves from 0.67 to 0.79, and RMSE reduces from 5.56 to 4.42.

We also see an exponential increase in training and prediction times. Our training time for 38.8 million records has increased from 3 seconds to 22 minutes, and our prediction time for 55.3 million records has increased from 2 seconds to 1 minute and 30 seconds. This shows that an increase in accuracy is inversely proportional to the training time. Now calculating for 1 record, we get a prediction time of 1.6 nanoseconds. This sub-millisecond prediction time is quite acceptable for real-world scenarios.

Additionally, we compute the average deviation of the predicted values from the actual values and get an average error of $2.22.

9. Conclusion

In conclusion, memory optimization for a dataset this size was crucial for successful and performant prediction of the fare amounts. Real-world distance calculation improved the accuracy significantly. Ensemble-based methods provide a very high accuracy of prediction at a relatively good real-world performance. Hence, our algorithm of choice is Random Forest, and it can predict the fare amounts within an error margin of $2.22.

References

Decimal Degrees. (2020, November 5). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Decimal_degrees

Measuring Accuracy of Latitude and Longitude. (2017, August 15). Retrieved from StackExchange: https://gis.stackexchange.com/questions/8650/measuring-accuracy-of-latitude-and-longitude

New York City Taxi Fare Prediction. (2020, November 5). Retrieved from Kaggle: https://www.kaggle.com/c/new-york-city-taxi-fare-prediction/data

Passenger Frequently Asked Questions. (2020, November 5). Retrieved from Taxi & Limousine Commission: https://www1.nyc.gov/site/tlc/passengers/passenger-frequently-asked-questions.page

Rouman, M. (2012, July 21). *The 70/30 Rule.* Retrieved from Maggie's E-learning Reflections : http://maggiese-learningreflections.blogspot.com/2012/11/the-7030-rule.html

Taxi Fare. (2020, November 5). Retrieved from Taxi & Limousine Commission: https://www1.nyc.gov/site/tlc/passengers/taxi-fare.page

Appendix

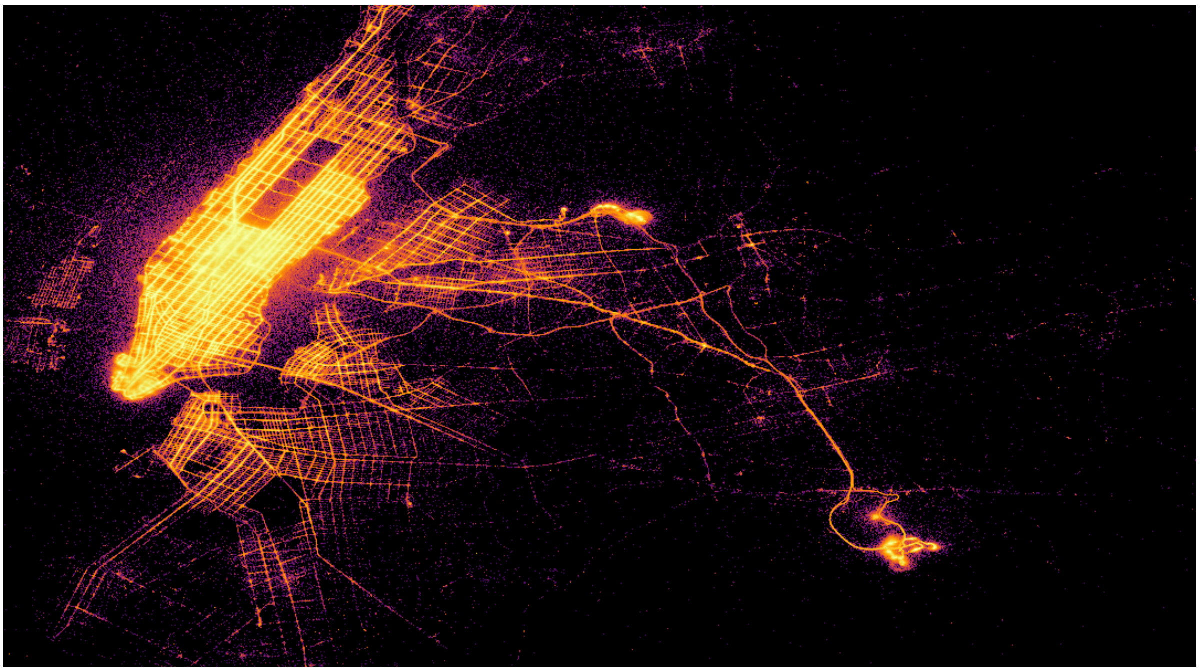


Figure 17: Visualization - Pickup Points

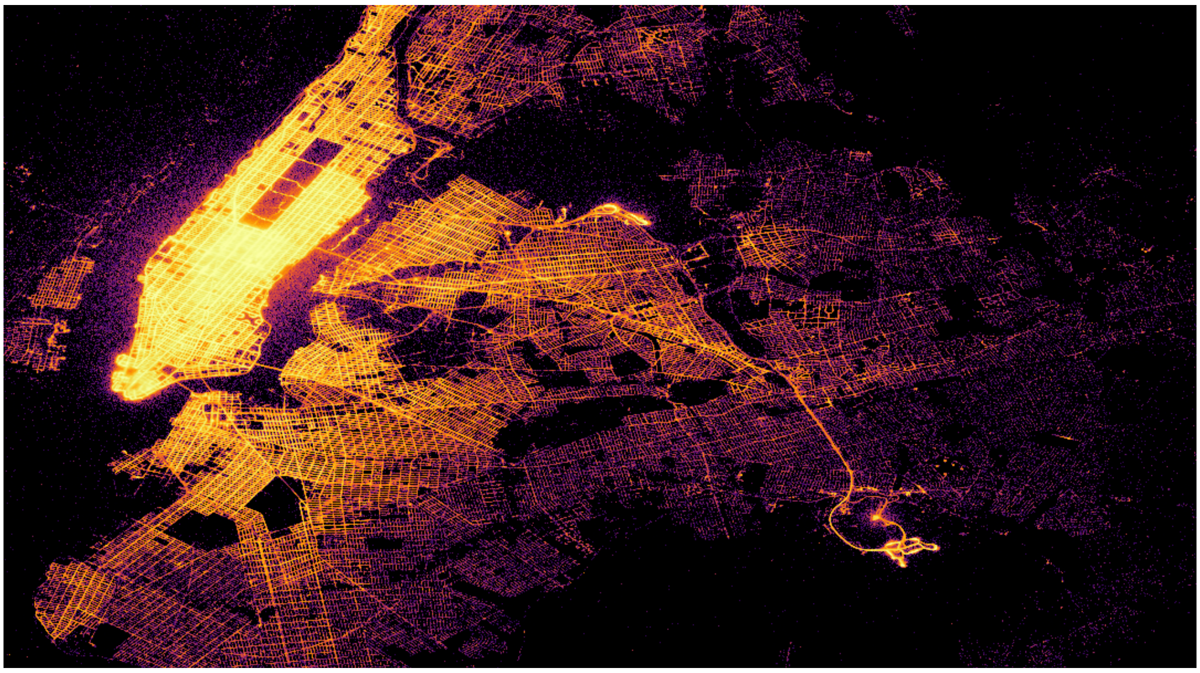


Figure 18: Visualization - Dropoff Points

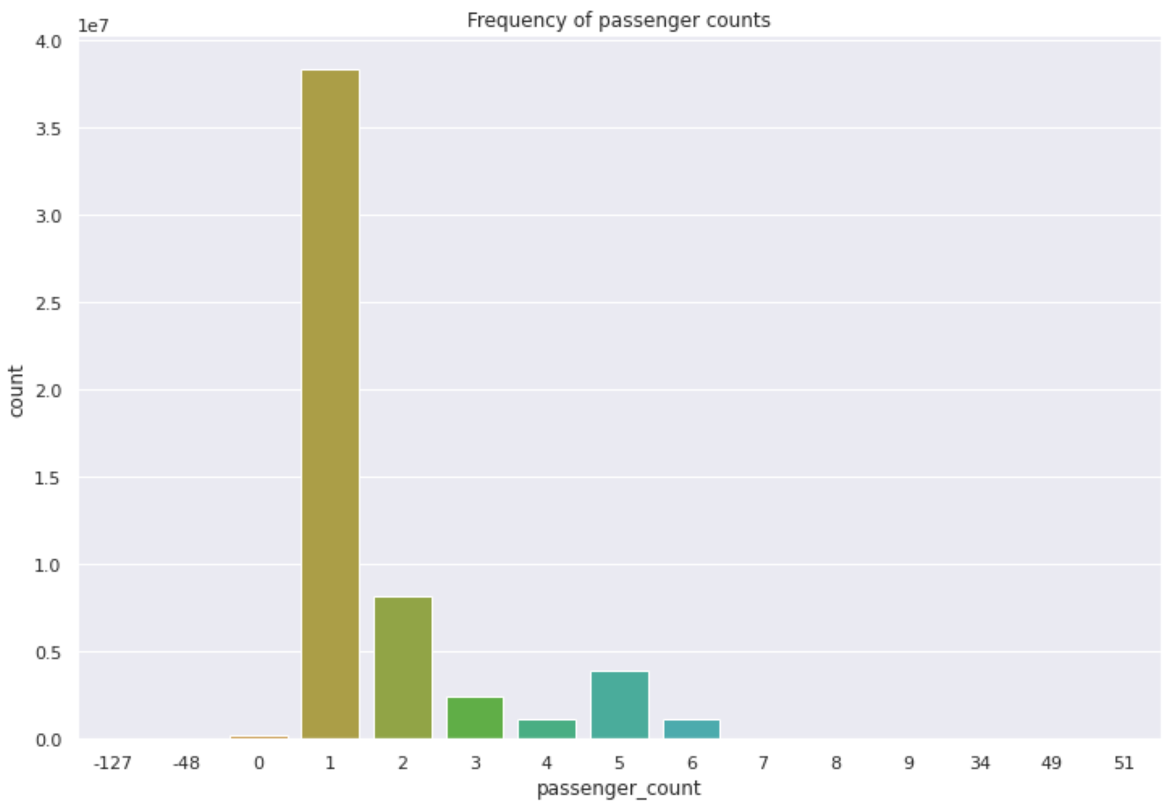


Figure 19: Passenger Count before Preprocessing

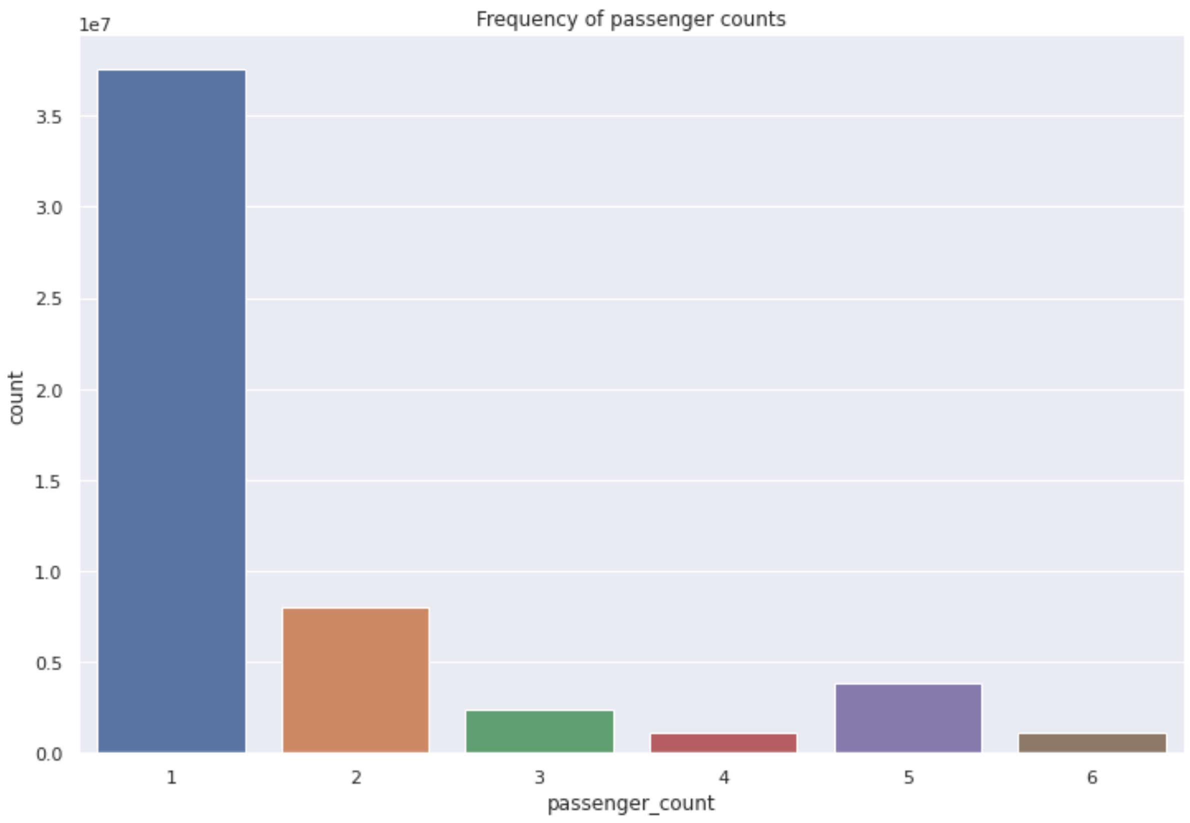


Figure 20: Passenger Count after Preprocessing

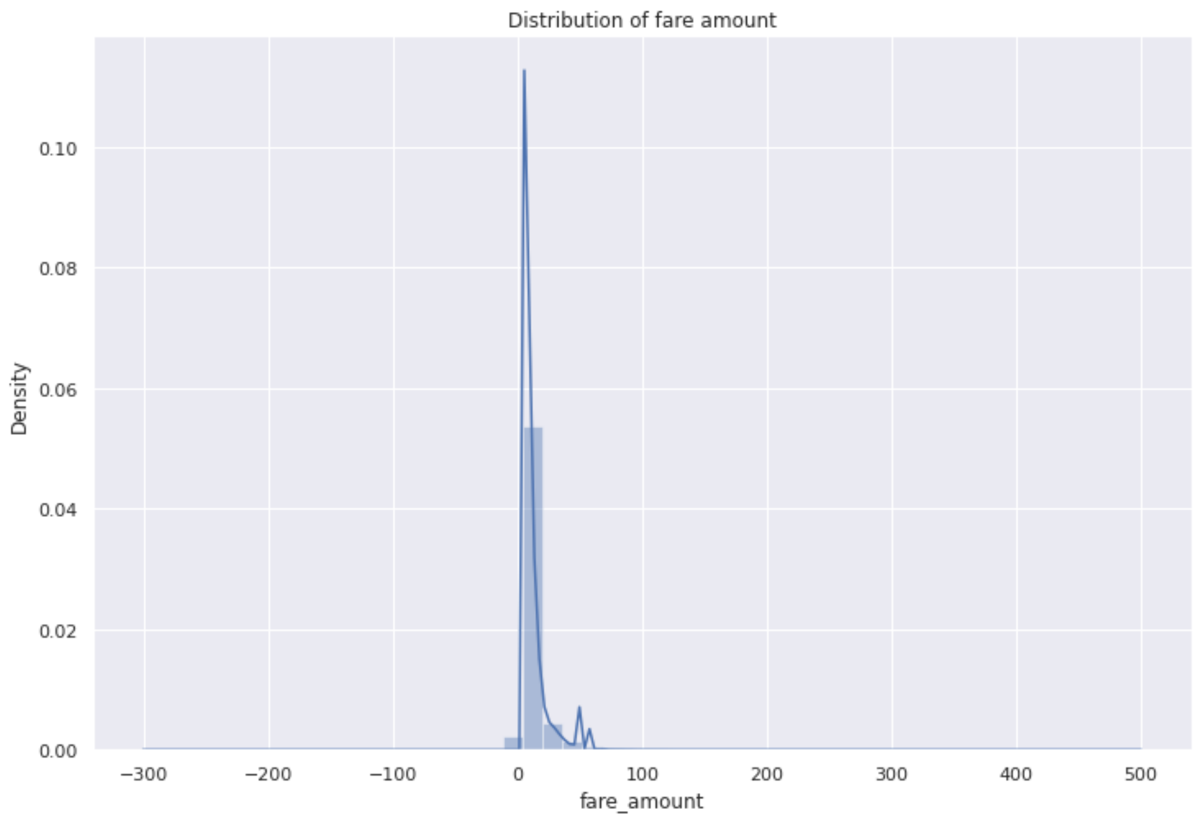


Figure 21: Fare Amount before Preprocessing

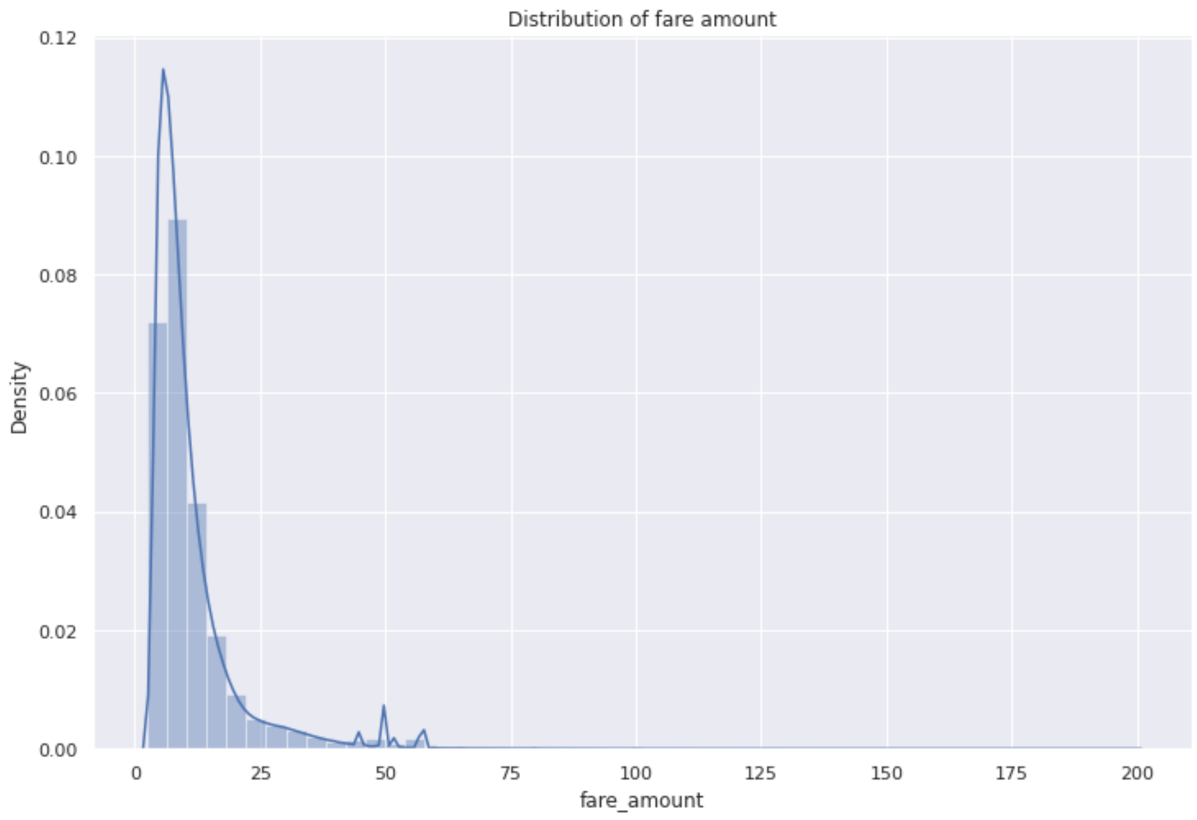
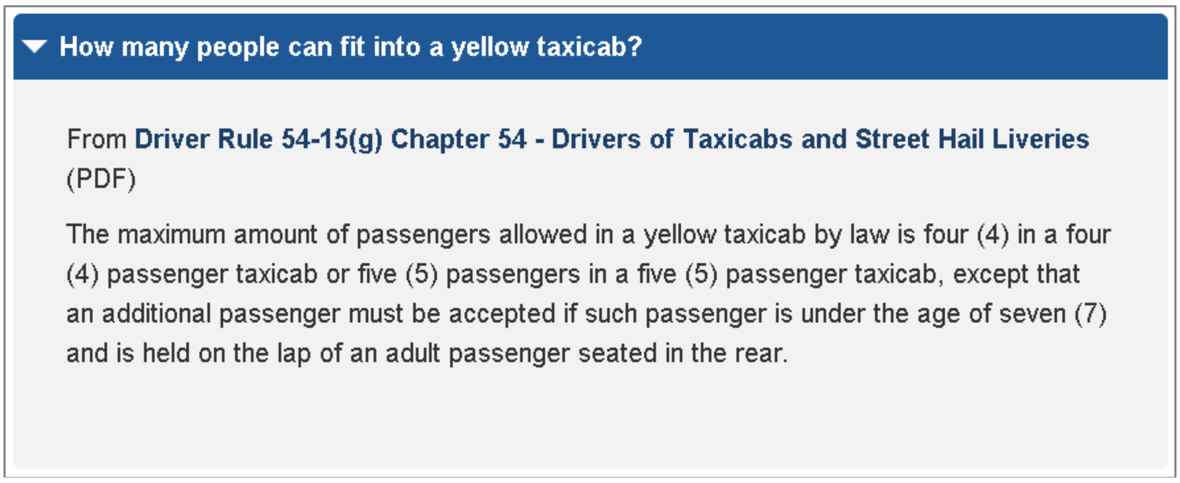


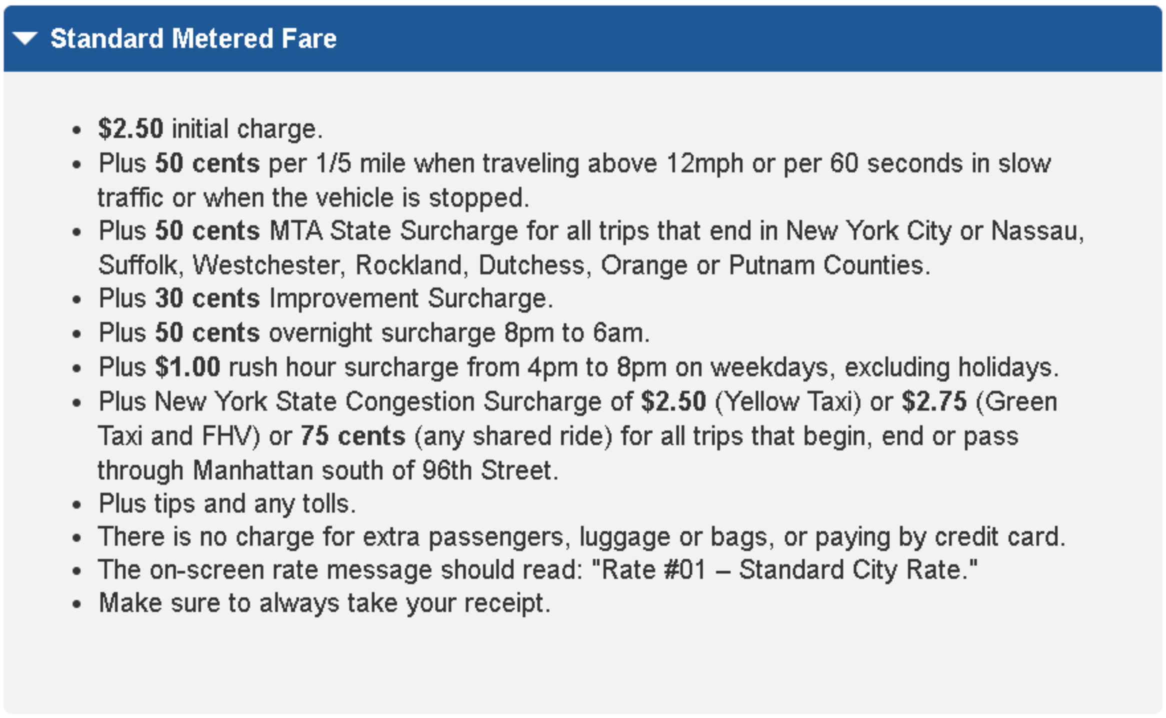
Figure 22: Fare Amount after Preprocessing


How many people can fit into a yellow taxicab?

From **Driver Rule 54-15(g) Chapter 54 - Drivers of Taxicabs and Street Hail Liveries** (PDF)

The maximum amount of passengers allowed in a yellow taxicab by law is four (4) in a four (4) passenger taxicab or five (5) passengers in a five (5) passenger taxicab, except that an additional passenger must be accepted if such passenger is under the age of seven (7) and is held on the lap of an adult passenger seated in the rear.

Figure 23: NYC Taxi Driver Rule 54-15(g)


Standard Metered Fare

- **$2.50** initial charge.
- Plus **50 cents** per 1/5 mile when traveling above 12mph or per 60 seconds in slow traffic or when the vehicle is stopped.
- Plus **50 cents** MTA State Surcharge for all trips that end in New York City or Nassau, Suffolk, Westchester, Rockland, Dutchess, Orange or Putnam Counties.
- Plus **30 cents** Improvement Surcharge.
- Plus **50 cents** overnight surcharge 8pm to 6am.
- Plus **$1.00** rush hour surcharge from 4pm to 8pm on weekdays, excluding holidays.
- Plus New York State Congestion Surcharge of **$2.50** (Yellow Taxi) or **$2.75** (Green Taxi and FHV) or **75 cents** (any shared ride) for all trips that begin, end or pass through Manhattan south of 96th Street.
- Plus tips and any tolls.
- There is no charge for extra passengers, luggage or bags, or paying by credit card.
- The on-screen rate message should read: "Rate #01 – Standard City Rate."
- Make sure to always take your receipt.

Figure 24: NYC Taxi Standard Metered Fare