Project Overview

Goal: The goal of this project is to predict the fare amount of a taxi ride in New York City given the pickup and dropoff dates/times, as well as some additional details about the ride. In a competition-driven market, it's important for a taxi company to be able to predict a fare amount that is both:

1. Competitive, or customers could divert to competitors

2. Profitable, because money matters.

These criteria have likely been thought about in previous fare calculations, so the goal of this project, is to automate this process, whereby a model can predict/propose a fare amount that satisfies the criteria above based on existing data.

Client: While this project specifically applies to the NYC Taxi and Limousine Commission, fare prediction is common in the transport industry in general. Airlines, trains, car pooling services, buses, etc, are all interested in satisfying the conditions above. And with the precedence of data, and the speed of technology, users expect an accurate and immediate price estimation for their travels.

Data: The data will be acquired from the NYC Taxi and Limousine Commission, which has been providing NYC taxi trip information since 2009. I will be training my model from entries in 2017 and testing it against records from the same year.

Trips will be sampled representatively across the entire year to account for any seasonal changes that may affect the fare. Some of the features I will be analyzing include pickup and dropoff times and dates, trip distance, avg speed, among other features as well.

Methodology: This is a supervised problem as we will be training the model using existing data, which it will then use to predict future fare amounts. Specifically, it is a regression problem, and we are looking at out how specific features in the dataset can affect the fare amount.

In making the predictions, I will begin with linear regression, and follow it with Random Forests and XGBoost which I expect to perform much better.

Data Aggregation

The data was collected from the NYC Taxi and Limousine Commission for the year of 2017. In its original form, it was separated by month, with each month containing ~9.5 million records. To aggregate my data, here are the steps I took:

1. For each month, I filtered the records to include regular cab rides only (not airport rides or group rides for example).

2. I sorted the data by pickup date/time and chose every 400th element to generate a representative sample.

3. I concatenated the records for all months into a single DataFrame and saved the data as a CSV file which contained ~265 K records.

Data Wrangling

Upon inspecting the data, there were no missing values, but there were a lot of invalid data points/outliers that needed to be cleaned. There were also some columns that needed to be dropped. Here are the measures I took:

Dropping Columns:

- I dropped *`PULocationID, DULocationID`* (pickup/dropoff IDs), as well as *`passenger counts`* as they were irrelevant for the fare prediction (according to the NYC Taxi Commission).

Cleaning Individual Columns:

- *`Trip Distances`*: Upon inspecting trip distances, there were many short, long, or nill distances. Since the goal of my project was to estimate fare amounts for regular rides, I placed upper and lower thresholds of 0.2 miles and 60 miles for trip duration, which captured ~99.25% of the data. The remaining trips were dropped.

- *`Trip Durations`*: Similarly, upper and lower thresholds were placed on trip durations: 2 minutes and 2 hours respectively, with outliers being dropped.

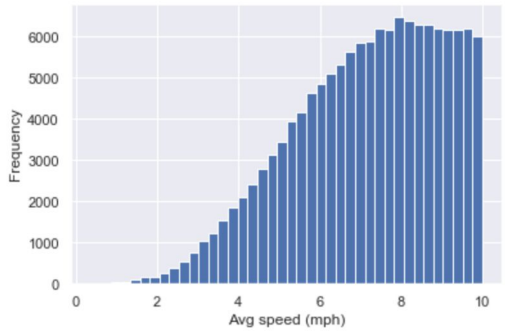
- *Fare Amount*: After the initial cleaning steps, there were no high fare amounts. A few negative fare amounts were found and inverted (changed to positive) as they seemed plausible. Lastly, a minimum threshold of $3.00 was placed on fare amount ($2.50 initial starting price + min $0.50 per 0.2 miles) as per the specifications of the NYC Taxi Commission.

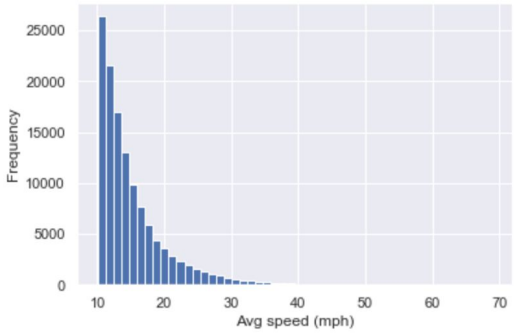
Cleaning/Comparing Dependent Columns:

Next, I compared individual columns against each other to make sure they made sense relative to each other. I compared `distance, duration`, and `fare amount` against each other, thus creating three new features: *avg speed, fare per mile,* and *fare per hour*.

Avg speed: the cleaned data thus far had avg speeds ranging between ~0.3 and 250 miles per hour. As this was an unrealistic range, I inspected the data visually and numerically to propose a more appropriate range.

Below are two PDFs of *avg speed* (0 to 10 mph on the left) and (10 to 70 mph on the right), as well as a 99.5 percent confidence interval of avg speeds.



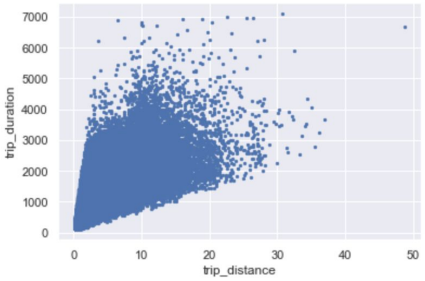


Numerically, the data suggested limits of about 2 mph and 35 mph for min and max average trip speeds. Instead, I extended the upper limit to 60 mph as this is perfectly fine, given that a trip lasts an hour or more for example.

```
# Get 0.25 and 99.75 percentils of avg soeeds
np.percentile(data.avg_speed, [0.25, 50, 99.75])
```

```
array([ 2.17,  9.77, 36.82])
```

To the right is a plot displaying `distance` vs `duration` after the cleaning.



Fare per mile and *fare per hour*: similarly, there were trips where fare vs distance/duration didn't make sense relative to each other. According to the NYC Taxi Commission, fare is calculated using the following equation:

Fare = $2.50 + ($2.50 ∗ miles driven) OR ($0.50 ∗ minutes driven *(in slow traffic)*)

Thus, a car may sit idle for 10 minutes and still be charged $5.00 without moving at all. To deal with this dilemma, I calculated 99.5% confidence intervals for both fare per mile and fare per hour and used them as min and max thresholds. Outliers beyond these thresholds were removed.

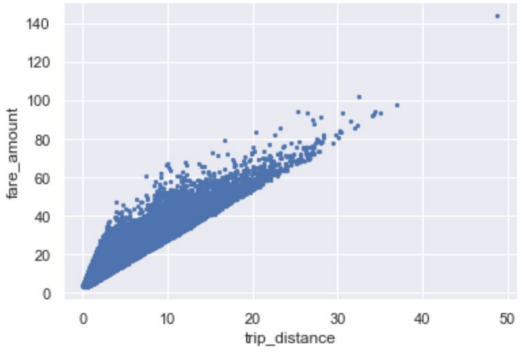
```
# Calculate 0.25 and 99.75 percentiles for fare per mile
np.percentile(data.fare_per_mile, [0.25, 50, 99.75])
```

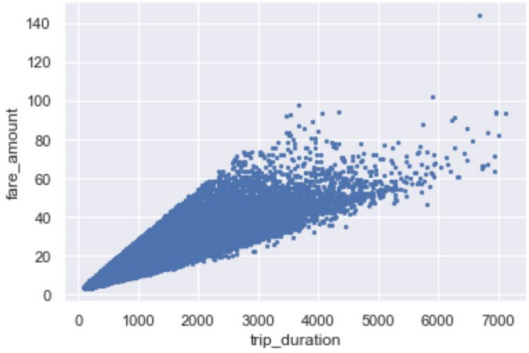
```
array([ 2.22    ,  3.75    , 11.692125])
```

```
# Calculate 0.25 and 99.75 percentiles for fare per hour
np.percentile(data.fare_per_hour, [0.25, 50, 99.75])
```

```
array([24.      , 36.44    , 94.492125])
```

Here are the resulting scatter plots of *fare* vs *duration* and *fare* vs *distance*:





Starting with 267,132 trips, after all data cleaning/wrangling, 258,173 records remained, so about ~3.3% of the original data was dropped.

Creating New Features:

As a final step in the data wrangling process, two new features were extracted from the pickup date/time feature:

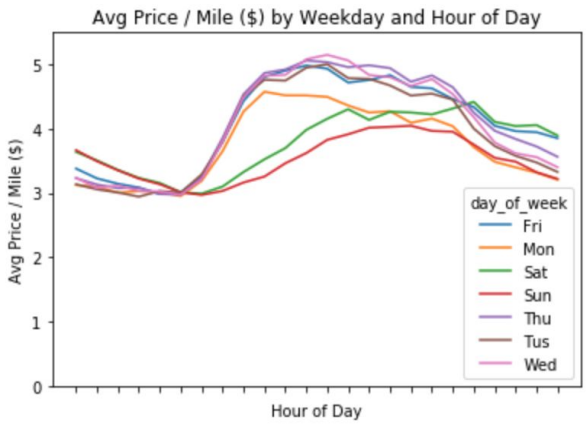
1. *hour*: a numerical data type with discrete numbers ranging from 0 to 23, representing the hour of the day for the trip pick up time.

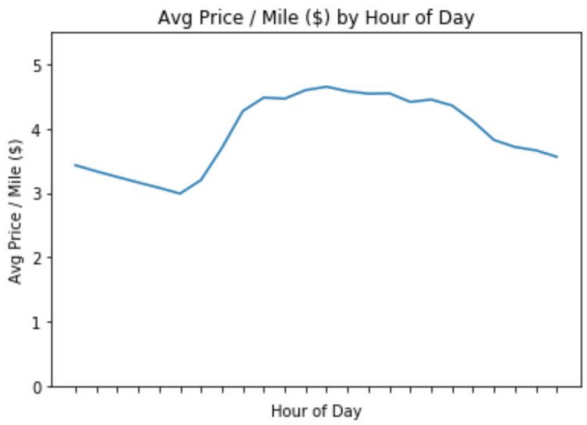
2. *weekday*: a categorical data type: 1 if trip was on a weekday, and 0 otherwise.

Additional EDA

In addition to performing EDA for data wrangling purposes, I explored other facets of my data to gain a better understanding of it. Here are some questions I sought to answer:

1. At what time should you leave to pay the lowest fare?

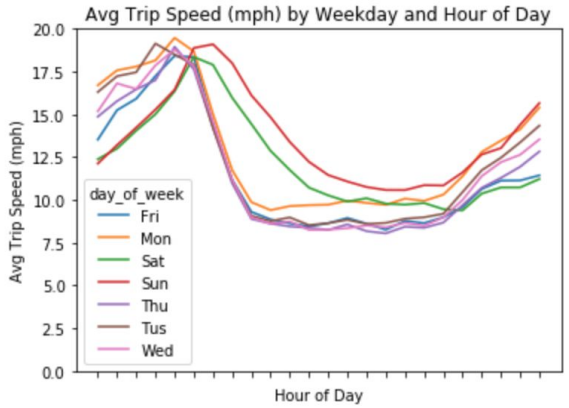


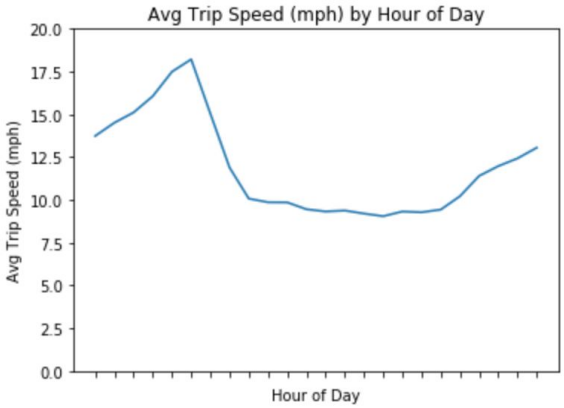


Key insights:

The best time to leave is around 5-6 am (for all days) where you can expect to pay around $3.00 per mile (plus the starting price of course). It quickly increases after that (especially on weekdays), reaching its peak at around lunch time. Expect to pay an additional $1.50 to $2.00 per mile at that time.

2. When should you leave to minimize trip duration?



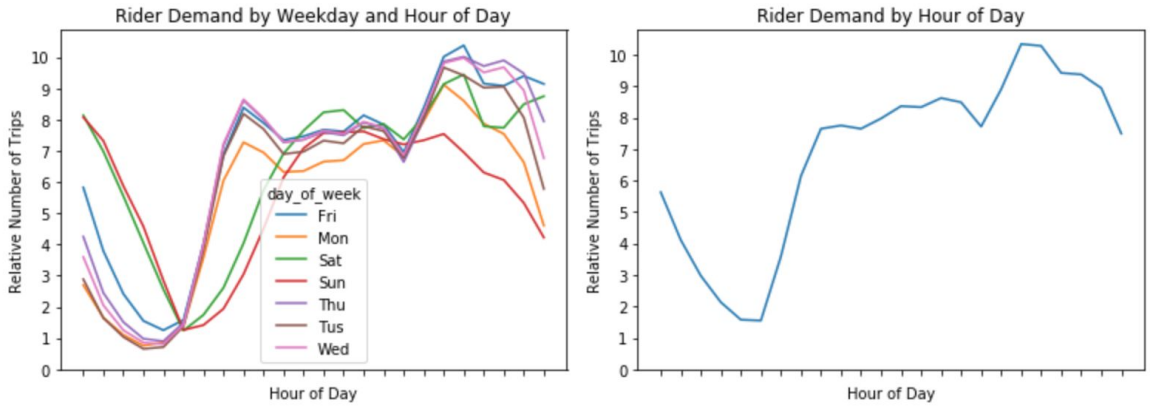


Key Insights:

Average trip speed is inversely proportional to avg price / mile. More clearly stated, faster rides correlate very well with a lower price / mile for the rider.

Again, a time between 5 and 6 am yields the fastest journey overall ~19mph. Waiting as little as 3 hours will almost double your journey time, and this gradually gets worse until around 6pm, until it sharply improves again.

3. When is there the most rider demand?



Key Insights:

While 5-6 am is the best time of day to leave (for both riders and drivers), it is actually the worst overall in terms of rider demand. By 8 am, the demand is about 4 to 5 times than that at 5-6 am, and remains pretty stable until the end of the day.

Additional Insights:

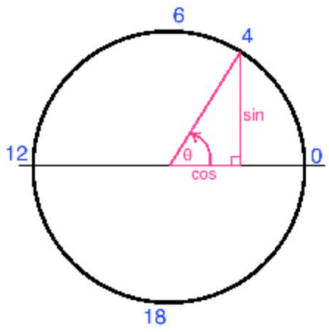
In addition to the highlights mentioned, the graphs above show us that there is variability between weekdays and weekends but not much between individual weekdays. For this reason, I decided to include a feature for trip `weekday` (as mentioned above) where 1 maps to weekday, and 0 maps to weekend.

We also see that features such as hour, distance, and speed are probably strong predictors of our target variable: `fare amount`.

Preliminary Machine Learning Preparation

Converting Hour Into a Cyclical Feature

Given that many machine learning algorithms deal with distance, I wanted to encode the *hour* feature in a more meaningful way. Initially, it was represented with discrete integers 0 through 23. This treats hours 0 and 1 as close to each other, 1 and 2 close, etc, whereas hours 23 and 0 are treated as being distant from each other, when in fact they're close.



To deal with this, and given that hour is a cyclical feature, I decided to represent it in a circular fashion. We know that a circle has 2π radians, so we can divide it by 24 (for each hour of the day). Each hour can be represented by two coordinates: the sin and cos of the given angle. In summary, I created two new features:

1. *hour_cos* = cos(hour * (2π / 24))
2. *hour_sin* = sin(hour * (2π / 24))

Encoding Categorical Variables

Our dataset consists mainly of numerical features, though one feature is categorical: *weekday*. This is 1 if the trip occurred on a weekday, and 0 otherwise. To encode it, I used the *get_dummies()* method from *pandas* and dropped the first column. Essentially, this just changed the feature from type *category* to *uint8*.

Checking Data for Multillinearity

Colinearity or multilinearity can inflate our variance, thus reducing the effectiveness of our model. To check for multilinearity, I used the *variance_inflaction_factor* method from *statsmodels*. After selecting all numerical features (including both versions of hour), the initial VIF scores I received were quite high.

The best combination of VIF scores was found after dropping *hour* and *avg speed*, leaving us with *{trip distance, trip_duration, hour_cos, hour_sin}*. Still, two of the features had a VIF score just above 5. To deal with this, I noticed how

trip_distance and *trip_duration* were highly skewed to the right, so after taking the log of their values, two things happened:

1. Their distribution became approximately normal

2. VIF scores were improved drastically.

Here is the general transformation of my VIF scores:

	VIF	Feature
0	9.968168	trip_distance
1	9.160437	trip_duration
2	8.291084	avg_speed
3	9.288919	hour
4	1.144650	hour_cos
5	2.408603	hour_sin

→

	VIF	Feature
0	5.112218	trip_distance
1	5.542898	trip_duration
2	1.136766	hour_cos
3	1.153591	hour_sin

→

	VIF	Feature
0	2.000753	trip_duration
1	1.782057	trip_distance
2	1.153582	hour_sin
3	1.092506	hour_cos

And here is how taking the log of *trip_distance* and *trip_duration* changed their distributions:

Out-of-the-box Machine Learning Algorithms

At this stage, my data ready was ready, so I began by using out-of-the-box machine learning models. The goal was to find one or two promising models which I could fine tune and optimize. Given that this is a regression problem, I tested three models: linear regression, random forests, and XGBoost.

The goal was to minimize the RMSE. Given that the median fare amount for my training data was $9.00, I wanted my RMSE value to be at most $1, and preferably, much lower.

Here is what each model yielded using the default parameters:

1. Linear Regression: ~$3.60 which was obviously too high

2. Random Forests: ~$0.50 which was drastically better.

3. XGBoost: ~$0.50 which was equally as good as Random Forests.

Hyperparameter Tuning and Final Results

Given these results, it was obvious that I wanted to pursue with Random Forests and XGBoost, ignoring Linear Regression altogether. In fine tuning both models, I began with `RandomizedSearchCV` (once) and ended with `GridSearchCV` (twice) after narrowing down my hyperparameters.

Random Forests: I used a fixed number of estimators: 400, performed 3-fold cross validation, and focused on optimizing the following hyperparameters:

- max_features
- max_depth
- min_samples_split
- Min_samples_leaf

Testing my tuned model on the test set yielded an RMSE of ~$0.46 which was 8% better than the default (untuned model), and overall, I think this was a good estimation.

XGBoost: Similarly, I used a fixed number of estimators: 200 in this case, performed 3-fold cross validation, and focused on optimizing the following hyperparameters:

- learning_rate
- max_depth
- colsample_bytree
- subsample

XGBoost yielded a similar RMSE value of ~$0.46, which again, was a great score given that the median `fare amount` was $9.00.

Summary

Both Random Forests and XGBoost provided an RMSE of $0.46, which in my opinion, was a good estimation. One reason I was able to get such a score was because I thoroughly wrangled my data at the beginning of the project, removing most outliers.

While I could have attempted further hyperparameter tuning, ensemble methods, etc, such attempts would have produced insignificant improvements, if any at all.

THE END.