NYC®
Taxi & Limousine Commission

By
Christopher Leung

TLC Analysis Report

Objective:

- For January 2023 trip record data, is the yellow taxi fare fair when it comes to charging customers?

Hypothesis:

- There might be an opportunity to optimize the pricing structure to promote fairness amongst all customers.

Assumptions:

- Every yellow taxi cab ride trip is independent even if it may be the same cab in multiple records in data. We also do not have a way to identify the cab per the dataset.
- Between each VendorID, there is no difference in the fare rate structure, everything is consistent.

Prerequisites:

- Yellow Taxi data set records for January 2023
- Taxi Zone Lookup table
- Python programming for analysis (Jupyter Notebook)
- Yellow Trips Data Dictionary

On to the analysis!

Analysis:

1. Reading in the data:

In [1]:
```
import pandas as pd
import time
from matplotlib import pyplot as plt
import numpy as np
import datetime
import seaborn as sns
```

In [2]:
```
# Read in data

d = pd.read_parquet("yellow_tripdata_2023-01.parquet", engine='auto')
```

In [106]:
```
[c for c in d.columns]
```

Out[106]:
```
['VendorID',
 'tpep_pickup_datetime',
 'tpep_dropoff_datetime',
 'passenger_count',
 'trip_distance',
 'RatecodeID',
 'store_and_fwd_flag',
 'PULocationID',
 'DOLocationID',
 'payment_type',
 'fare_amount',
 'extra',
 'mta_tax',
 'tip_amount',
 'tolls_amount',
 'improvement_surcharge',
 'total_amount',
 'congestion_surcharge',
 'airport_fee',
 'diff_hrs',
 'diff_min',
 'diff_sec',
 'mph',
 'loc_pu_do',
 'pu_year',
 'pu_month',
 'do_year',
 'do_month']
```

In [58]:
```
d.head()
```

Out[58]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payme
0	2	2023-01-01 00:32:10	2023-01-01 00:40:36	1.0	0.97	1.0	N	161	141	
1	2	2023-01-01 00:55:08	2023-01-01 01:01:27	1.0	1.10	1.0	N	43	237	
2	2	2023-01-01 00:25:04	2023-01-01 00:37:49	1.0	2.51	1.0	N	48	238	
3	1	2023-01-01 00:03:48	2023-01-01 00:13:25	0.0	1.90	1.0	N	138	7	
4	2	2023-01-01 00:10:29	2023-01-01 00:21:19	1.0	1.43	1.0	N	107	79	

5 rows × 28 columns

First before diving into the data analysis, I wanted to view the data and see what kind of condition it is in. The data looks readable and formatted in columns which is good. I noticed a few issues with the data, which will be addressed later.

2. Creating attributes for MPH, Same or different borough and Airport. Then reviewing histogram for MPH.

In [4]:
```
# Time difference between pickup and drop off

#data['diff_days'] = (data['tpep_dropoff_datetime'] - data['tpep_pickup_datetime']) / np.timedelta64(1, 'D')
d['diff_hrs'] = (d['tpep_dropoff_datetime'] - d['tpep_pickup_datetime']) / np.timedelta64(1, 'h')
d['diff_min'] = (d['tpep_dropoff_datetime'] - d['tpep_pickup_datetime']) / np.timedelta64(1, 'm')
d['diff_sec'] = (d['tpep_dropoff_datetime'] - d['tpep_pickup_datetime']) / np.timedelta64(1, 's')

# Distance per second

d['mph'] = d['trip_distance']/d['diff_hrs']
```

In [28]:
```
# Same or different borough

dfmrg['Same_Diff_Borough'] = np.where( dfmrg['Borough_x'] == dfmrg['Borough_y'], "SAME", "DIFFERENT")

# Create Airport field

dfmrg['Airport'] = np.where( (dfmrg['Zone_x'].str.contains("Airport")) | (dfmrg['Zone_y'].str.contains("Airport")), 1, 0)

# Calculated total amount

dfmrg['calc_total_amount'] = dfmrg['fare_amount'] + dfmrg['extra'] + dfmrg['mta_tax'] + dfmrg['improvement_surcharge'] + dfmrg['t
```

- For MPH attribute, I created it by first taking the time difference between pickup and dropoff timestamps. Then converting this into hours and then taking the distance and dividing by the hour. Making this average mph for each trip.
- For Same_Diff_Borough, I created it to indicate whether the pickup location borough was the same as the dropoff location borough. I wanted to see if there are some relationships between the two and what could be analyzed from it.
- For Airport, I created this attribute to indicate 0 or 1 if the pickup or dropoff location had an Airport involved.
- For Calc_total_amount, this attribute is the sum of fare_amount, extra, mta_tax, improvement_surcharge, tolls_amount, congestion_surcharge, and airport_fee. I noticed when I did this I had mismatches between the total_amount and this summed up total. I'm not sure where the discrepancy is from so I will use Calc_total_amount as the reliable variable to find the total amount. Around 84% of the total_amount does not match with my calculated amount which means there are some data issues involved. I trust my logic more as the total_amount attribute looks unreliable, unless there is another fee involved that I wasn't aware of.

In [62]:
```
data_a[data_a['total_amount'] != data_a['calc_total_amount']].shape[0] / data_a.shape[0]
```

Out[62]:
```
0.8392761471097916
```

3. Data Quality issues:

```
In [6]: # Remove dates that don't make sense

        # Create fields to figure out year and month
        d['pu_year'] = d['tpep_pickup_datetime'].dt.year
        d['pu_month'] = d['tpep_pickup_datetime'].dt.month

        d['do_year'] = d['tpep_dropoff_datetime'].dt.year
        d['do_month'] = d['tpep_dropoff_datetime'].dt.month
```

```
In [7]: # Filter for only year = 2023 and month = 1

        dataa = d[((d['pu_year'] == 2023) & (d['pu_month'] == 1)) & ((d['do_year'] == 2023) & (d['do_month'] == 1))]
```

```
In [8]: # Exclude mph greater than 40
        data = dataa[(dataa['mph'] <= 40) & (dataa['mph'] > 0)]

        # Exclude negative total amounts
        data = data[(data['total_amount'] > 0)]

        # Exclude negative tip amounts
        data = data[(data['tip_amount'] >= 0)]

        # Exclude negative fare amounts
        data = data[(data['fare_amount'] > 0)]

        # Create rush hour field
        data['rush_hr'] = np.where( (data['tpep_pickup_datetime'].dt.dayofweek <= 4) & ((data['tpep_pickup_datetime'].dt.hour >= 16) & ((
```

- Removed data that didn't have trips in January 2023
- Removed mph data that was greater than 40 mph and less than 0 as it didn't seem logical at all. No taxi goes negative mph or so fast in a city especially with traffic congestions!
- Removed all amounts that were negative (total, tips, and fare amount)
- Created a field that would indicate rush hour time frame on the weekdays.

4. Utilizing Zone data

```
In [3]: # Read in zone locations

        zone = pd.read_csv("taxi_zone_lookup.csv")
```

```
In [9]: zone.head()
```

Out[9]:

	LocationID	Borough	Zone	service_zone
0	1	EWR	Newark Airport	EWR
1	2	Queens	Jamaica Bay	Boro Zone
2	3	Bronx	Allerton/Pelham Gardens	Boro Zone
3	4	Manhattan	Alphabet City	Yellow Zone
4	5	Staten Island	Arden Heights	Boro Zone

I read in the Taxi zone lookup data then I merged it with the original taxi data set.

```
In [9]: # Join in zone locations data for pickup and drop off to data

        df = data.merge(zone, how = 'inner', left_on = 'PULocationID', right_on = 'LocationID')
        dfmrg = df.merge(zone, how = 'inner', left_on = 'DOLocationID', right_on = 'LocationID')
```

```
In [21]: dfmrg.head()
```

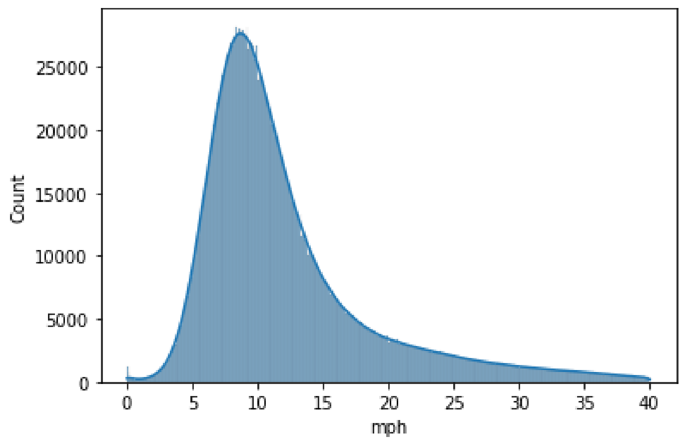
Out[21]:

nID	DOLocationID	payment_type	...	do_month	rush_hr	LocationID_x	Borough_x	Zone_x	service_zone_x	LocationID_y	Borough_y	Zone_y	service_zone_y
161	141	2	...	1	0	161	Manhattan	Midtown Center	Yellow Zone	141	Manhattan	Lenox Hill West	Yellow Zone
161	141	2	...	1	0	161	Manhattan	Midtown Center	Yellow Zone	141	Manhattan	Lenox Hill West	Yellow Zone
161	141	1	...	1	0	161	Manhattan	Midtown Center	Yellow Zone	141	Manhattan	Lenox Hill West	Yellow Zone
161	141	2	...	1	0	161	Manhattan	Midtown Center	Yellow Zone	141	Manhattan	Lenox Hill West	Yellow Zone
161	141	2	...	1	0	161	Manhattan	Midtown Center	Yellow Zone	141	Manhattan	Lenox Hill West	Yellow Zone

As you can see the different boroughs are appended to the taxi data set. Now we can see which locations each ride started (Borough_x) and ended (Borough_y).

5. Histogram plot of average MPH

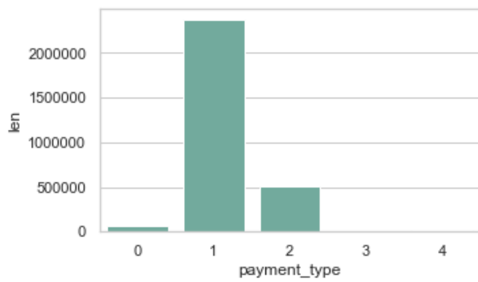
```
In [65]: # Histogram of mph

         set("Histogram of MPH")
         sns.histplot(data = data_a, x = "mph", kde = True)
         ead.
           y = y[:, np.newaxis]

Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x16714bbbcc0>
```



- I wanted to see if there was any value extracted from the histogram of MPH as this would help provide clarity in the distribution of how fast each trip is. As you can tell, most trips go from 8 mph to 12 mph, which makes sense given that a majority of trips occur in Manhattan. With traffic congestion, this speed on average looks pretty accurate for the mph throughout the entire trip.
- Having tested the validity of data, I would say this is a pretty accurate depiction and that the data (once filtered above 0 mph and below 40 mph) are good thresholds to set. Valid data should mimic as close to reality as possible, so I'm glad the distribution looks accurate. If it wasn't then we'd have to look into the sensors/taximeter to understand what problem occurred - why the accuracy of data is not correct for measuring distance and/or time.

6. Pay Type bar chart

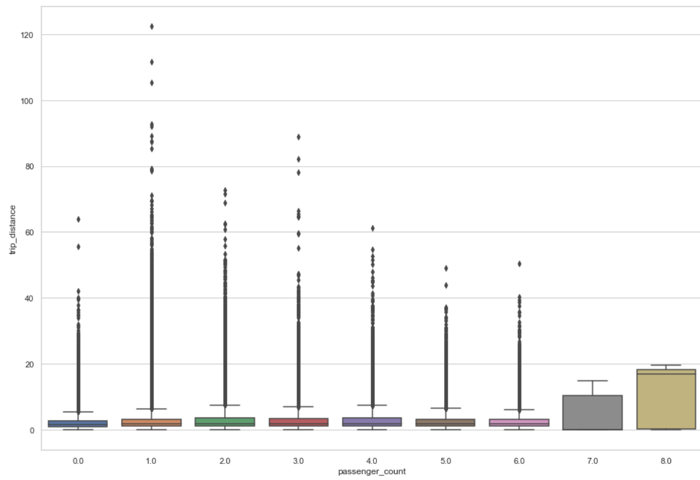
```
In [34]: # Create pay type summary
         pay_type = data_a.groupby(['payment_type'])['VendorID'].agg([len])
         pay_type.reset_index(inplace = True)
         # Barplot of passenger_count
         plt.figure(figsize = (5, 3))
         sns.barplot(x = 'payment_type', y = 'len', data = pay_type, estimator = sum, ci = None, color = '#69b3a2')

D:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:11: FutureWarning:
The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.
  # This is added back by InteractiveShellApp.init_path()

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x1de7d50c7b8>
```



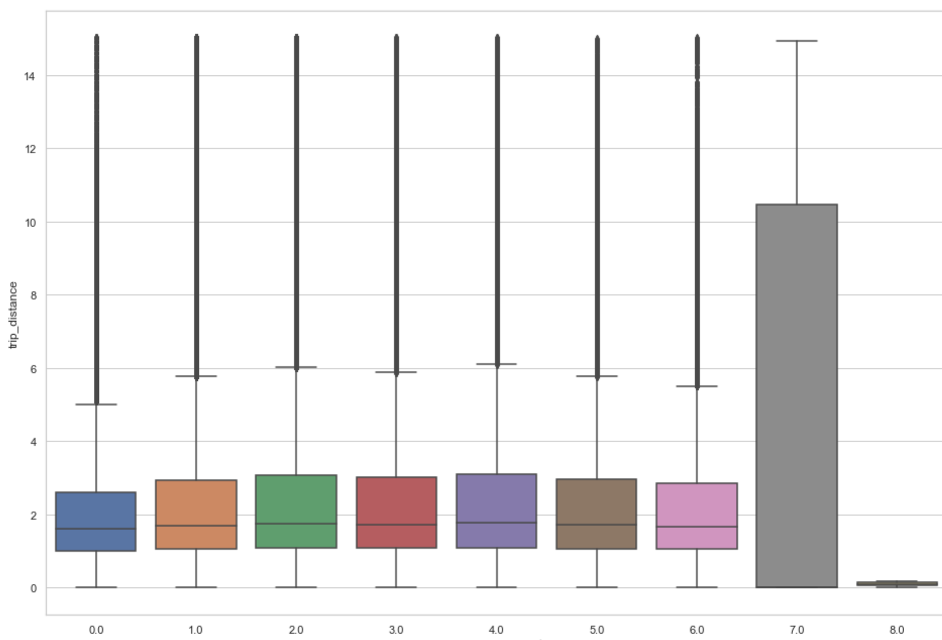
- Looking into the attribute pay_type to get an idea of the distribution, you can see most of the trips are payment type = 1. This wouldn't really help us much in analysis because you ideally want to see more even disparity between each payment type. There's nothing too interesting to note from this.

7. Boxplot for passenger count and trip distance

```
In [69]: # Boxplot of passenger_count and trip distance

         plt.figure(figsize = (16, 11))

         sns.set(style = 'whitegrid')

         sns.boxplot(x = 'passenger_count', y = 'trip_distance', data = data_a)

Out[69]: <matplotlib.axes._subplots.AxesSubplot at 0x16879c8bda0>
```



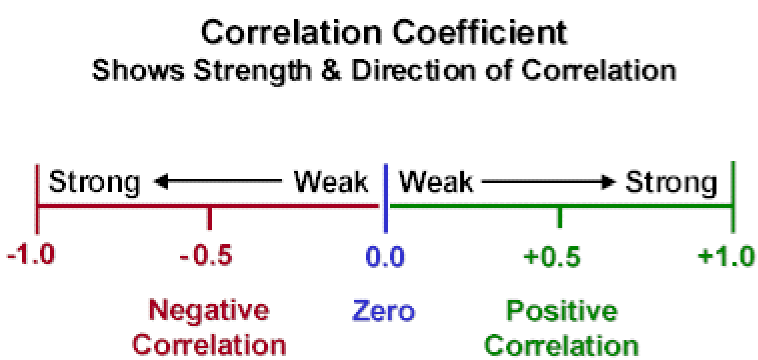
- I wanted to see if there was any relationship between passenger count and trip distance and it appears there is a relationship but there are lots of outliers based on trip distance.
- I decided to remove trip_distances above 15 miles to see if I would get a better result and I see the chart below. It doesn't look any better so I would say looking into such an analysis would not help much.



8. Correlation between variables and fare_amount.

```
In [37]: # Correlation between trip distance and fare amount

         np.corrcoef(data_a['trip_distance'], data_a['fare_amount'])

Out[37]: array([[1.        , 0.95804339],
                [0.95804339, 1.        ]])

In [45]: # Correlation between time and fare amount

         np.corrcoef(data_a['hours'], data_a['fare_amount'])

Out[45]: array([[1.        , 0.23289261],
                [0.23289261, 1.        ]])
```

- Correlation coefficient is a measurement in statistics to find if there exists a linear relationship between two variables, the number varies from -1 to 1 (shown in chart below). Between trip_distance and fare_amount, you can tell it is near 0.96 which indicates a very strong positive correlation between these variables. Meaning they both move in the same direction very closely so if trip distance increases then fare amount increases - this makes a lot of sense since the rate structure in yellow taxis is largely predicated on distance.



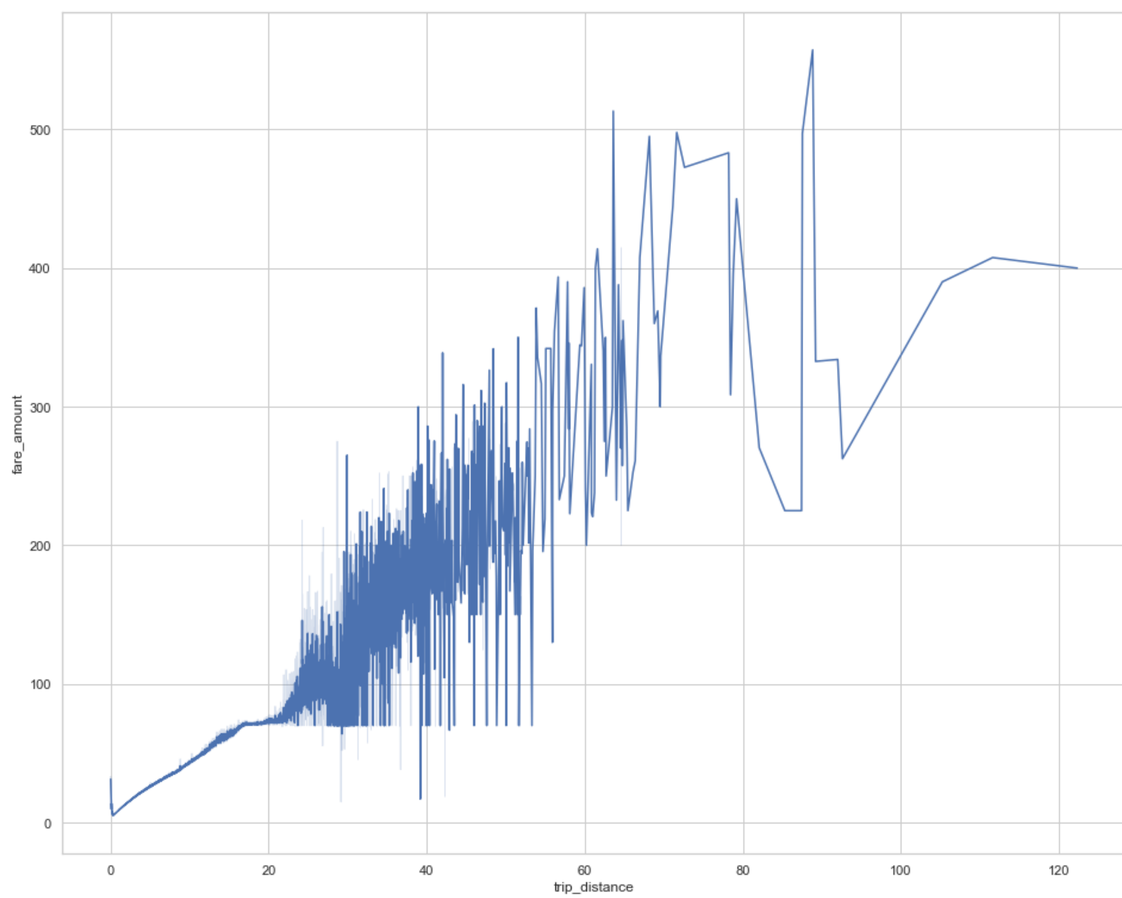
TLC standard meter rate structure:

Standard Metered Fare

- **$3.00** initial charge.
- Plus **70 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 **$1.00** Improvement Surcharge.
- Plus **$1.00** overnight surcharge 8pm to 6am.
- Plus **$2.50** 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.

- As I looked into the correlation between time in hours to fare amount, it is at 0.23 which indicates an extremely low relationship between time and fare amount. I thought this was very interesting because I would've thought time is a big component for taxis rate structure. After all, passengers want to get to the destination as quickly as possible without much congestion.
- I've also plotted a line graph between fare amount and trip distance and you can tell the strength of its relationship.

Out[85]: <matplotlib.axes._subplots.AxesSubplot at 0x1674aacd6a0>



9. Analysis on rush hour

In [33]:
```
# Overall speeds

data_a.groupby(['rush_hr'])['trip_distance','calc_total_amount','fare_amount','mph'].agg(np.mean)
```

```
D:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: FutureWarning: Indexing with multiple
erted to a tuple of keys) will be deprecated, use a list instead.
  This is separate from the ipykernel package so we can avoid doing imports until
```

Out[33]:

rush_hr	trip_distance	calc_total_amount	fare_amount	mph
0	3.363311	23.941584	18.258122	12.380232
1	3.143694	25.627606	18.002427	10.457736

- In my next analysis, I wanted to dive deeper into a potential relationship between rush hour (4 pm to 7 pm, Monday through Friday) and any key performance indicators. As you can see in the chart above, rush hour affects the calculated total amount. It also shows that rush hour generally has slower speeds and shorter trip distances on average. Also I noticed the fare amount is less during rush hour but this is due to not including the congestion surcharge attribute. Therefore, the calculated total amount is reflecting the most ideal scenario to reduce congestion by penalizing passengers for riding during rush hour.

	A	B	C	D	E
1	Pickup Borough	Dropoff Borough	Average	Count	Percentage
2	Bronx	Bronx	10.35649753	1531	0.05%
3	Bronx	Brooklyn	21.63427608	251	0.01%
4	Bronx	EWR	37.97971919	1	0.00%
5	Bronx	Manhattan	14.22359897	995	0.03%
6	Bronx	Queens	22.84749214	235	0.01%
7	Bronx	Staten Island	24.72915872	14	0.00%
8	Bronx	Unknown	19.47825358	26	0.00%
9	Brooklyn	Bronx	20.6314723	250	0.01%
10	Brooklyn	Brooklyn	10.5518643	7943	0.27%
11	Brooklyn	EWR	26.86624141	25	0.00%
12	Brooklyn	Manhattan	15.3691607	6001	0.20%
13	Brooklyn	Queens	19.37674351	1346	0.05%
14	Brooklyn	Staten Island	28.78424843	64	0.00%
15	Brooklyn	Unknown	19.19455533	53	0.00%
16	EWR	Brooklyn	25.2577529	3	0.00%
17	EWR	EWR	12.9069645	34	0.00%
18	EWR	Manhattan	34.6533782	2	0.00%
19	EWR	Queens	37.03159696	2	0.00%
20	EWR	Staten Island	35.22304106	2	0.00%
21	EWR	Unknown	16.19474719	6	0.00%
22	Manhattan	Bronx	20.6666553	8792	0.30%
23	Manhattan	Brooklyn	15.27066521	62987	2.12%
24	Manhattan	EWR	29.0523562	5763	0.19%
25	Manhattan	Manhattan	10.28572893	2497858	83.90%

- As you can see in the chart above, the most trips in NYC for yellow taxis is when the pickup borough is Manhattan and the dropoff borough is Manhattan, so it is safe to say this makes the majority of the dataset.

- The next step was I wanted to look into trips that were within the same borough or different borough (view the chart below).

In [55]:
```
# Borough speeds

bor_speeds = data_a.groupby(['Same_Diff_Borough','rush_hr'])['hours','trip_distance','fare_amount','

bor_speeds
```

```
D:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: FutureWarning: Indexing with mul
erted to a tuple of keys) will be deprecated, use a list instead.
  This is separate from the ipykernel package so we can avoid doing imports until
```

Out[55]:

Same_Diff_Borough	rush_hr	hours	trip_distance	fare_amount	calc_total_amount	total_amount	mph
DIFFERENT	0	0.538650	11.337073	47.618801	57.979078	64.916307	22.320376
	1	0.668175	12.384695	52.617125	66.172094	74.187677	19.154758
SAME	0	0.215253	2.096678	13.594176	18.569354	20.590904	10.801239
	1	0.220831	1.930502	13.458082	20.336001	22.589325	9.315959

- As you can see that everything makes sense as out of borough trip fees are generally higher than travel within the same borough. We also do not have enough data on the other boroughs and their statistics. This could provide additional analysis on distances and trip costs associated with it.
- Other than this, everything appears pretty consistent with the rate structure and customer fairness when it comes to paying for taxi rides.

10. Feature Importance with Random Forest

- In the next step I wanted to see which features were the most important in predicting the fare amount. I have shown the Python code and chart below.

In [13]:
```
# Lets look into feature importance and see which variables affect fare_amount

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.inspection import permutation_importance
from sklearn import preprocessing

# Label encode the location pickup and dropoff

label_encoder = preprocessing.LabelEncoder()

# Encode the location pickup and dropoff

data_a['loc_pu_do_enc'] = label_encoder.fit_transform(data_a['loc_pu_do'])

# Encode Same borough or different borough

data_a['Same_Diff_Borough_enc'] = label_encoder.fit_transform(data_a['Same_Diff_Borough'])
```

```
D:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html
rsus-a-copy

D:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html
rsus-a-copy
```

In [14]:
```
# Transform the variables with numeric value to standardization

from sklearn.preprocessing import StandardScaler
scale= StandardScaler()

data_a['trip_distance_enc'] = pd.DataFrame(scale.fit_transform(data_a[['trip_distance']]))
data_a['congestion_surcharge_enc'] = pd.DataFrame(scale.fit_transform(data_a[['congestion_surcharge']]))
data_a['mph_enc'] = pd.DataFrame(scale.fit_transform(data_a[['mph']]))
data_a['hours_enc'] = pd.DataFrame(scale.fit_transform(data_a[['hours']]))

data_a['fare_amount_enc'] = pd.DataFrame(scale.fit_transform(data_a[['fare_amount']]))
```

- I standardized variables trip distance, congestion surcharge, mph, hours taken, and fare amount as most machine learning models read data cleaner with transformed continuous variables. Typically standardizing your data is a great process. Results may differ if I didn't standardize the variables or show less accuracy within the model.

In [15]:
```python
# Train test split

#X = data_a[['Airport','payment_type','trip_distance_enc','congestion_surcharge_enc','loc_pu_d
X = data_a[['Airport','trip_distance_enc','loc_pu_do_enc','rush_hr','mph_enc','hours_enc','Sam
Y = data_a['fare_amount']


X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random_state=42)
```

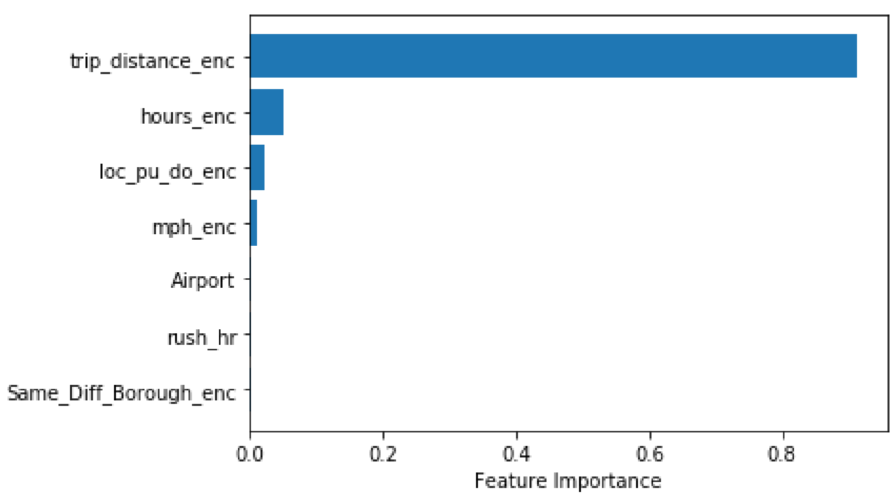
- After standardizing, I will split data into train (75%) and test data (25%) for the model to be trained and tested for data sets to evaluate model accuracy.

In [16]:
```python
# Use Random Forest Regressor

rf = RandomForestRegressor(n_estimators=100)
rf.fit(X_train, y_train)

sort = rf.feature_importances_.argsort()
```

- Once the data split has been established, I fed the training and test data into the Random Forest Regression model with 100 trees.

In [30]:
```python
plt.barh(X.columns[sort], rf.feature_importances_[sort])
plt.xlabel("Feature Importance")
```

Out[30]: `Text(0.5, 0, 'Feature Importance')`

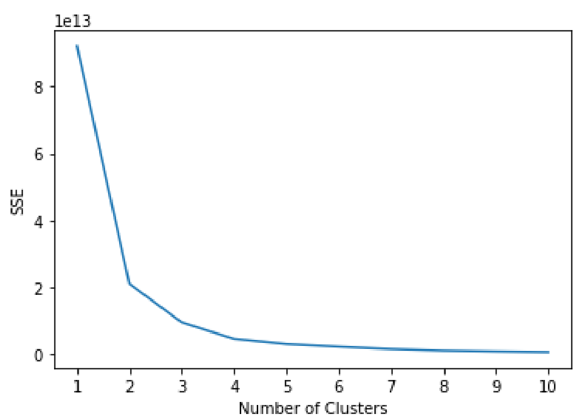


- The results showed that obviously trip distance being the most important feature and next was the hours or time taken for the entire trip.
- There is nothing surprising to note from this but I did want to point out that the hours or time taken for a trip is heavily underutilized which to me should be changed as time is just as important as distance when it comes to taxi rides and carbon emissions! This should enact some policy changes.

11. K Means segmentation analysis

- I decided to segment the data using the k means algorithm which is meant to discriminate a dataset in clusters. In this scenario, I wanted to see if the algorithm would find something interesting in the cluster splits of the data.

In [47]:
```
# K Means analysis

# Elbow method

from sklearn.cluster import KMeans

kmeans_kwargs = {
   "init": "random",
    "n_init": 10,
    "max_iter": 300,
    "random_state": 42}

sse = []

scaled_features = data_a[['Airport','trip_distance_enc','loc_pu_do_enc','rush_hr','mph_enc','h

for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(scaled_features)
    sse.append(kmeans.inertia_)

plt.plot(range(1, 11), sse)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()
```



- In my method to determine the optimal number of clusters for the data set, I applied the elbow method which measures the sum of distances between the cluster and the cluster centroid. As you can see the elbow in the chart above, showing that cluster = 2 is where the optimal cluster exists and that any cluster beyond this number wouldn't provide much more accuracy improvement.

- My final results are shown below to which there are two clusters.

```
In [26]: # Look into mph and fare amount

         df_clust.groupby(['Cluster','Same_Diff_Borough','rush_hr'])['hours','trip_distance','fare_amount',
```

```
D:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: FutureWarning: Indexing with m
erted to a tuple of keys) will be deprecated, use a list instead.
  after removing the cwd from sys.path.
```

Out[26]:

Cluster	Same_Diff_Borough	rush_hr	hours	trip_distance	fare_amount	calc_total_amount	mph
0	DIFFERENT	0	0.470101	8.587233	38.765532	47.463696	19.433120
		1	0.584418	9.182402	42.611973	53.797263	16.026417
	SAME	0	0.212337	2.022997	13.274645	18.207277	10.732563
		1	0.213327	1.861043	13.045430	19.895564	9.420035
1	DIFFERENT	0	0.572854	12.709139	52.036244	63.059687	23.761007
		1	0.700606	13.624644	56.491187	70.796777	20.366073
	SAME	0	0.218490	2.178478	13.948914	18.969543	10.877482
		1	0.228575	2.002182	13.883935	20.788942	9.208555

- The clusters segmented for the same/different borough and rush hour did not display much surprises as the results indicate the same outputs as I had shown from before. During rush hour, passengers are penalized by paying additional congestion surcharges and that distance has a direct effect on total amount or fare amount. Time is still not a factor with the fare amount.
- There isn't much to gather from this as the algorithm isn't disproving anything and it is nothing interesting. These results aren't very promising for our analysis.

Summary:

- My research question was to address the concern whether the taxi rates are fair for customers. My initial take was to find an analysis that would find discrepancy in taxi rates by segmenting on rush hour and same/different boroughs. But my analysis results show otherwise as the rates are properly adjusted for these different criterias and it's largely due to congestion surcharge and distance being the big factor in fare amount. Therefore my analysis failed to disprove the rate disparity for customers.
- However, this analysis is a huge help in providing information on potentially changing the structure of the taxi rate. In my opinion, we had not really factored in time as a component within the fare amount.

Standard Metered Fare

- **$3.00** initial charge.
- Plus **70 cents** per 1/5 mile when traveling above 12mph or per 60 seconds in slow traffic or when the vehicle is stopped.

- Based on the taxi rate from the TLC website, you can see either distance or time to be a factor within the fare amount. But based on my analysis, it is heavily skewed towards distance as time is not really affecting the fare amount. I believe this will show we may need to update the taxi rate as time should be a larger factor.
- When customers order a taxi, they are not budgeting solely on distance but rather time is a bigger component. Most customers want to know how quickly they would get to their destination. Without time factoring into the rate as distance heavily outweighs the taxi rate, we might find customers unhappy with their trip.
- By changing the rate structure to be more balanced (both distance and time), we can incentivize taxi drivers to drive at a better balance as time will be a big factor. It also can be efficient for drivers as they look to get to their destination the quickest route possible rather than the most they can charge.
- If I had more time, I would expand potential formula changes in the rate structure and experiment with this to see if there is a balance between distance and time in the fare rate. Next if policy changes for the rate structure, then I would analyze the data before and after to see if there has been noticeable improvements or efficiencies. Or we could utilize a monte carlo simulation model to simulate what might happen in the future with rate changes before implementing the policy. But this is an entirely new analysis that would take a lot of time.

Side notes:

- The dataset at hand is pretty messy with some dates off, outliers within all features with amounts, passenger count is dependent on driver to input information leading to human error, negative distances, time between pickup and dropoff being the same, and other data quality issues came up due to outliers. I would try to fix some of those issues and try to get to the root cause of the problem.
- If I had more time, I would've analyzed different boroughs other than Manhattan and see if their times and fare differed greatly. Also I would look into compiling more data from other months to see if the analysis occurs not just within January 2023.
- Other data that would help in analysis, is the fail rate of the taximeter or time series data that shows every second the speed in which the taxi is going per trip. This data would help uncover the data quality issues and also the times the taxi is actually stuck in traffic.

I hope you enjoyed my analysis! I spent a lot of time reviewing as I was obsessed with it. Thank you!