TLC TAXI - PROJECT

by : Ana Farida

PROJECT THE TLC TAXI

1. Conduct a complete exploratory data analysis.
2. Perform any data cleaning, data visualizations , and data analysis steps to understand unusual variables (e.g., outliers).
3. Use descriptive statistics (statical analysis) to learn more about the data.
4. Create and run a regression model.
5. Filter down to consider the most relevant variables for running regression, statical analysis, and parameter tuning.
6. Parameter tuning

<

REGRESSION ANALYSIS

Predicting Taxi Fare

A/B TEST

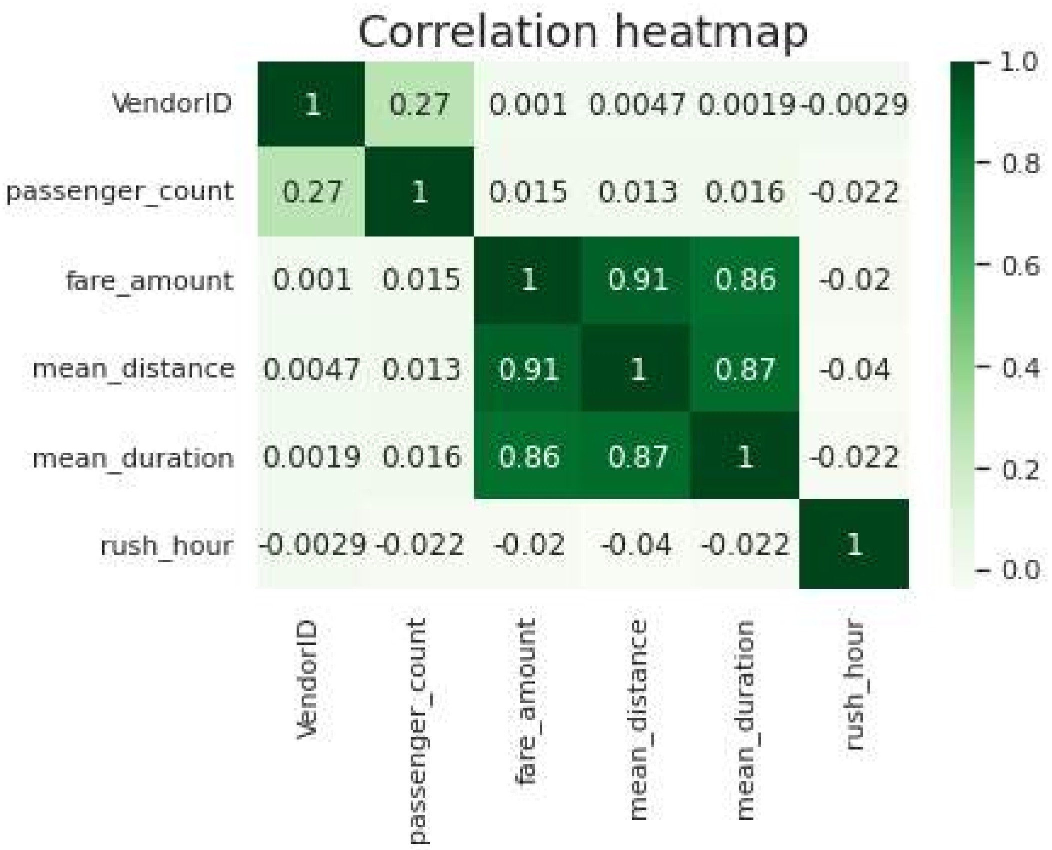
Comparing Credit Card and Cash Payments

CLASSIFICATION ANALYSIS

Predicting Generous Tippers

>

PREDICTING TAXI FARE



- A **multiple linear regression model** was built to predict taxi fares.
- A model uses five features—VendorID, passenger_count, mean_distance, mean_duration, and rush_hour—to predict fare_amount.
- The **mean_distance feature had the greatest impact on the model's prediction.** Both mean_distance (0.91) and mean_duration (0.86) are strongly correlated with the target variable, fare_amount, and also highly correlated with each other (Pearson correlation = 0.87).
- While highly correlated features can widen the confidence interval and complicate statistical inferences, they can still produce accurate predictions. Since the goal is to predict fare_amount for machine learning models, both variables were included despite their correlation.

```
Training data:
Coefficient of determination: 0.8398434585044773
R^2: 0.8398434585044773
MAE: 2.186666416775414
MSE: 17.88973296349268
RMSE: 1.4787381163598285
```

```
Test data:
Coefficient of determination: 0.8682583641795454
R^2: 0.8682583641795454
MAE: 2.1336549840593864
MSE: 14.326454156998944
RMSE: 3.785030271609323
```

- The model achieved 84% performance on the training data and **87% on the test data**

```
# 1. Calculate Standard Deviation of 'mean_distance' in X_train data

print(X_train["mean_distance"].std())

# 2. Divide the model coefficient by the standard deviation

print(7.133867 / X_train["mean_distance"].std())
```

```
3.574812975256415
1.9955916713344426
```

- The model's coefficient **for mean_distance** indicates that for every 3.57 miles traveled, the **fare increases by** $7.13, which averages to **about $2.00 per mile.**

A/B TEST

Comparing Credit Card and Cash Payments

Note: In the dataset, `payment_type` is encoded in integers:

- 1: Credit card
- 2: Cash
- 3: No charge
- 4: Dispute
- 5: Unknown

```
payment_type
1    13.429748
2    12.213546
3    12.186116
4     9.913043
Name: fare_amount, dtype: float64
```

```
credit_card = taxi_data[taxi_data["payment_type"]==1]["fare_amount"]
cash = taxi_data[taxi_data["payment_type"]==2]["fare_amount"]
stats.ttest_ind(a=credit_card, b=cash, equal_var=False)
```

```
Ttest_indResult(statistic=6.866800855655372, pvalue=6.797387473030518e-12)
```

- The goal is to find ways to increase taxi drivers' revenue by analyzing the relationship between payment type and fare amount.
- **Descriptive statistics** compare average fares for each payment type, **showing that credit card users tend to pay more than cash users**. However, this could be due to random chance.
- To confirm the difference, a **two-sample t-hypothesis test** (independent t-test) with **significance level 5%** performed as part of the A/B test to analyze the difference between two unknown population means.
- There is a **significant difference in the average fare amount between customers who use credit cards and those who use cash.**
- The hypothesis test suggests that **encouraging credit card payments could help increase revenue.**

- This project assumes customers were required to use either credit cards or cash and always followed that requirement, although the data wasn't collected this way. To conduct the A/B test, we randomly grouped the data by payment method.
- The dataset doesn't consider other factors, such as customers preferring to pay with credit cards for longer trips because they may not have enough cash. This suggests that the fare amount likely influences the choice of payment method, not the other way around.

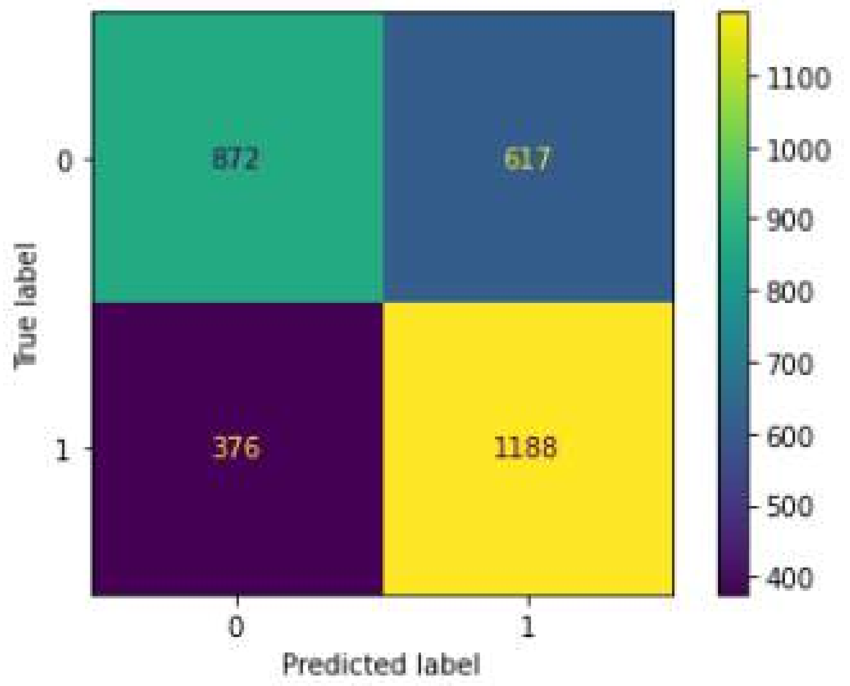
PREDICTING GENEROUS TIPPERS

```
Avg. cc tip:  2.7298001965279934
Avg. cash tip:  0.0
```

```
rf = RandomForestClassifier(random_state=42)

cv_params = {'max_depth': [None],
             'max_features': [1.0],
             'max_samples': [0.7],
             'min_samples_leaf': [1],
             'min_samples_split': [2],
             'n_estimators': [300]
             }

scoring = {'accuracy', 'precision', 'recall', 'f1'}

rf1 = GridSearchCV(rf, cv_params, scoring=scoring, cv=4, refit='f1')

rf1.fit(X_train, y_train)
```

```
xgb = XGBClassifier(objective='binary:logistic', random_state=0)

cv_params= {'learning_rate': [0.1],
            'max_depth': [8],
            'min_child_weight': [2],
            'n_estimators': [100]
            }

scoring = {'accuracy', 'precision', 'recall', 'f1'}

xgb1 = GridSearchCV(xgb, cv_params, scoring=scoring, cv=4, refit='f1')

xgb1.fit(X_train, y_train)
```

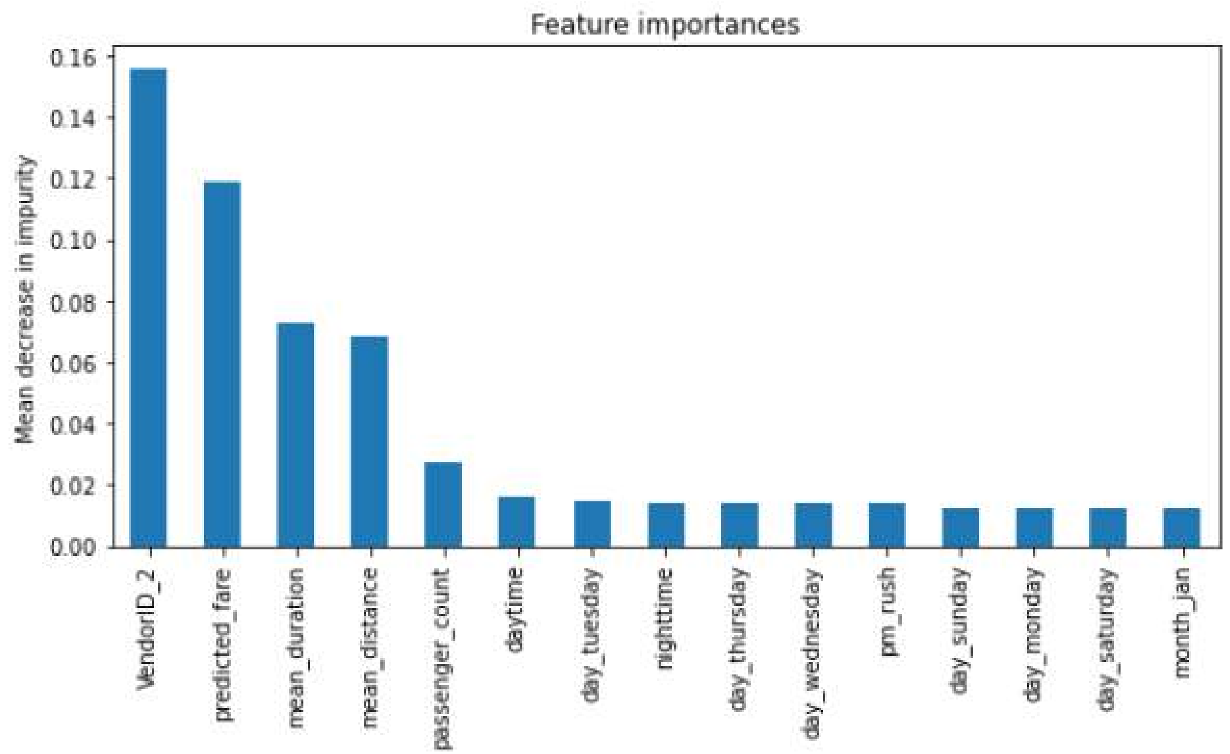
- Tree-based models can predict whether a customer is a generous tipper (those tipping 20% or more).
- This strategy helps drivers increase earnings without excluding anyone.
- Descriptive statistics compare average tip for each payment type, showing that credit card users tend to tip more than cash users.
- Use data such as tipping history, pickup/dropoff times and locations, estimated fares, and payment methods to build models (random forest and gradient boosting).



- **The model is nearly twice as likely to predict a false positive** (predicting a generous tip when it's actually low) **than a false negative** (predicting no generous tip when it is actually generous). This indicates that **type I errors are more common**. While it's better for drivers to be pleasantly surprised by a generous tip than disappointed by a low one, the model's overall performance remains acceptable.

	model	precision	recall	F1	accuracy
0	RF CV	0.679793	0.767111	0.720795	0.685146
0	RF test	0.658172	0.759591	0.705254	0.674746
0	XGB CV	0.689592	0.791221	0.736901	0.700622
0	XGB test	0.675690	0.797954	0.731750	0.700295

- **Gradient boosting model is the champion**, with **F1 score test 73%**, ~0.03 higher than the random forest.



- **`VendorID`, `predicted_fare`, `mean_duration`, and `mean_distance` are the most important features.** `VendorID` is the most predictive feature. This seems to indicate that one of the two vendors tends to attract more generous customers.

EXPLORATORY DATA ANALYSIS - 1

Column name	Description
ID	Trip identification number
VendorID	A code indicating the TPEP provider that provided the record.
	1 = Creative Mobile Technologies, LLC
	2 = VeriFone Inc.
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
PULocationID	TLC Taxi Zone in which the taximeter was engaged.
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged.
RateCodeID	The final rate code in effect at the end of the trip.
	1 = Standard rate
	2 = JFK
	3 = Newark
	4 = Nassau or Westchester
	5 = Negotiated fare
	6 = Group ride
Store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before being sent to the vendor, aka "store and forward," because the vehicle did not have a connection to the server.
	Y = store and forward trip
	N = not a store and forward trip
Payment_type	A numeric code signifying how the passenger paid for the trip.
	1 = Credit card
	2 = Cash
	3 = No charge
	4 = Dispute
	5 = Unknown
	6 = Voided trip
Fare_amount	The time-and-distance fare calculated by the meter.
Extra	Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.
MTA_tax	$0.50 MTA tax that is automatically triggered based on the metered rate in use.
Improvement_surcharge	$0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge begin being levied in 2015. Began in 2015.
Tip_amount	Tip amount – this field is automatically populated for credit card tips. Cash tips are not included.
Tolls_amount	Total amount of all tolls paid in trip.
Total_amount	The total amount charged to passengers. Does not include cash tips.

- Exploratory Data Analysis (EDA) is important because it helps a data professional to get to know the data, understand its outliers, handle its missing values, and prepare it for future modeling.

```
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             22699 non-null  int64
 1   VendorID               22699 non-null  int64
 2   tpep_pickup_datetime   22699 non-null  object
 3   tpep_dropoff_datetime  22699 non-null  object
 4   passenger_count        22699 non-null  int64
 5   trip_distance          22699 non-null  float64
 6   RatecodeID             22699 non-null  int64
 7   store_and_fwd_flag     22699 non-null  object
 8   PULocationID           22699 non-null  int64
 9   DOLocationID           22699 non-null  int64
 10  payment_type           22699 non-null  int64
 11  fare_amount            22699 non-null  float64
 12  extra                  22699 non-null  float64
 13  mta_tax                22699 non-null  float64
 14  tip_amount             22699 non-null  float64
 15  tolls_amount           22699 non-null  float64
 16  improvement_surcharge  22699 non-null  float64
 17  total_amount           22699 non-null  float64
dtypes: float64(8), int64(7), object(3)
```

	trip_distance	fare_amount	tip_amount	total_amount
count	22699.000000	22699.000000	22699.000000	22699.000000
mean	2.913313	13.026629	1.835781	16.310502
std	3.653171	13.243791	2.800626	16.097295
min	0.000000	-120.000000	0.000000	-120.300000
25%	0.990000	6.500000	0.000000	8.750000
50%	1.610000	9.500000	1.350000	11.800000
75%	3.060000	14.500000	2.450000	17.800000
max	33.960000	999.990000	200.000000	1200.290000

- There are 22,699 rows and 18 columns in the dataset.
- No null values.
- tpep_pickup_datetime & tpep_dropoff_datetime are object or non-numeric dtype that must be converted to datetime.
- Regarding trip distance, most rides are between 1-3 miles, but the maximum is over 33 miles and the minimum is 0 miles.
- Regarding fare amount, the distribution is worth considering. The maximum fare amount is a much larger value (999.99) than the 25-75 percent range of values (6.5-14.5). Also, its questionable how there are negative values for minimum fare amount (-120).

EXPLORATORY DATA ANALYSIS - 2

- The first two total_amount values (1,200 and 450) are significantly higher than the others (which are less than 258). The most expensive ride (1,200) is not necessarily the longest, as it covers only 2.6 miles.

trip_distance	fare_amount	tip_amount	tolls_amount	improvement_surcharge	total_amount
2.60	999.99	200.00	0.00	0.3	1200.29
0.00	450.00	0.00	0.00	0.3	450.30
33.92	200.01	51.64	5.76	0.3	258.21
0.00	175.00	46.69	11.75	0.3	233.74

- The vendorID represented in the data:

```
2    12626
1    10073
Name: VendorID, dtype: int64
```

- The mean total_amount for each vendor:

VendorID	total_amount
1	16.298119
2	16.320382

- According to the data dictionary, the payment method was encoded as follows with the corresponding data:

1. Credit card
2. Cash
3. No charge
4. Dispute
5. Unknown
6. Voided trip

```
1    15265    Avg. cc tip:  2.7298001965279934
2     7267    Avg. cash tip:  0.0
3      121
4       46
Name: payment_type, dtype: int64
```

- The average tip for credit card and cash payment types is 2.73 and 0, respectively.

- The credit-card-only data for passenger count:

```
1    10977
2     2168
5      775
3      600
6      451
4      267
0       27
Name: passenger_count, dtype: int64
```

- The average tip_amount for each passenger count (credit-card-only):

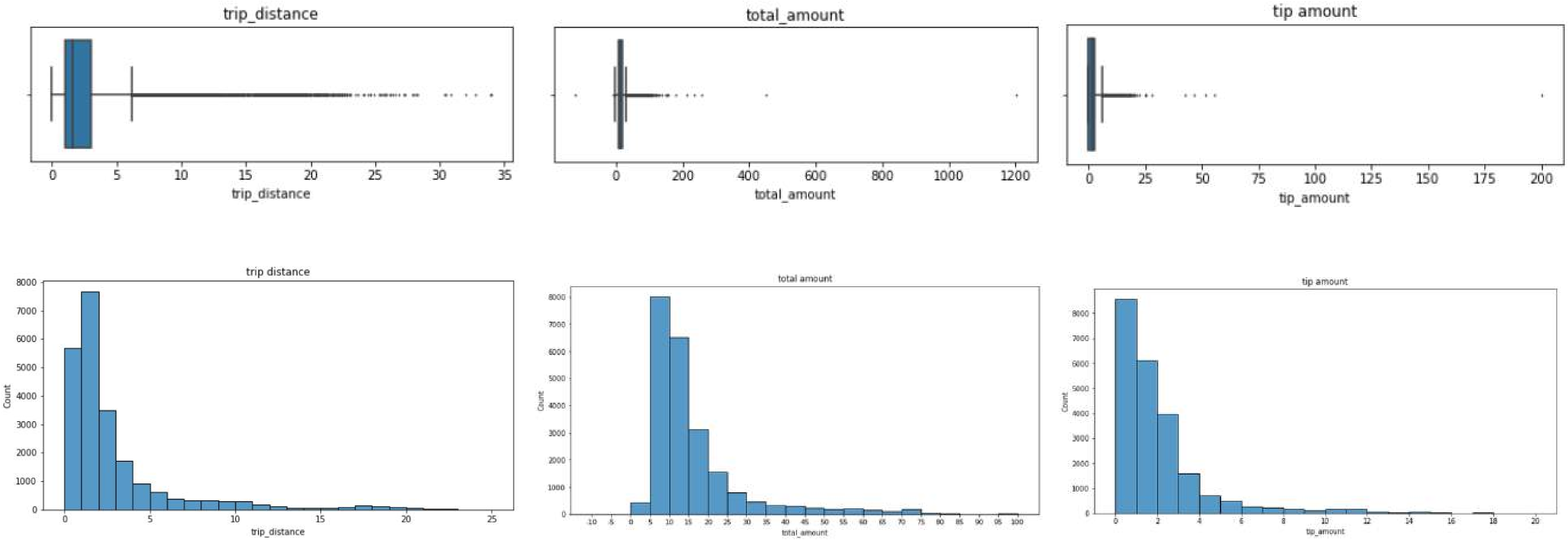
passenger_count	tip_amount
0	2.610370
1	2.714681
2	2.829949
3	2.726800
4	2.607753
5	2.762645
6	2.643326

- **For taxi ride, trip_distance and total_amount are the two variables that are most likely to help build a predictive model.**

EXPLORATORY DATA ANALYSIS - 3

- A box plot will be helpful to determine outliers and where the bulk of the data points reside in terms of `trip_distance`, `duration`, and `total_amount`
- A scatter plot will be helpful to visualize the trends and patters and outliers of critical variables, such as `trip_distance` and `total_amount`
- A bar chart will help determine average number of trips per month, weekday, weekend, etc.

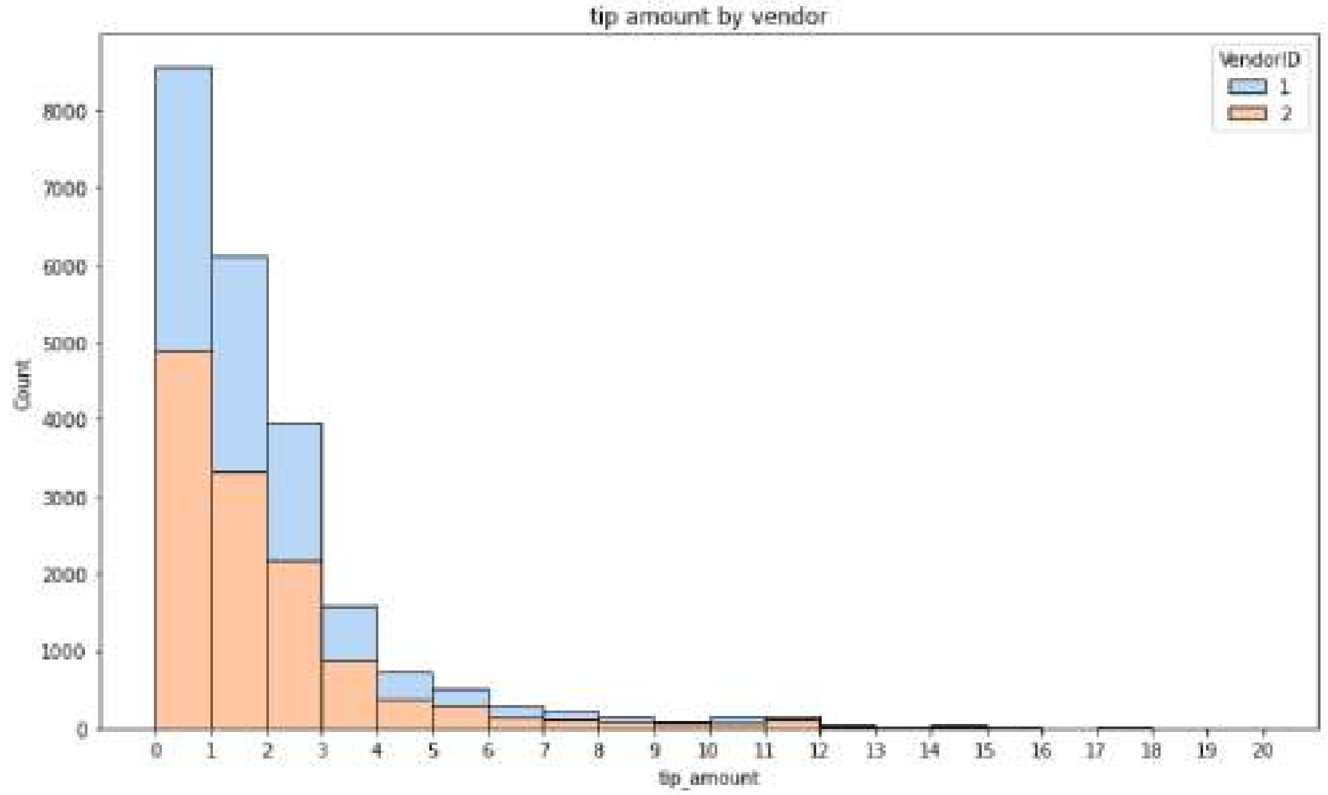
- Data distributions of trip_distance, total_amount, and tip_amount:



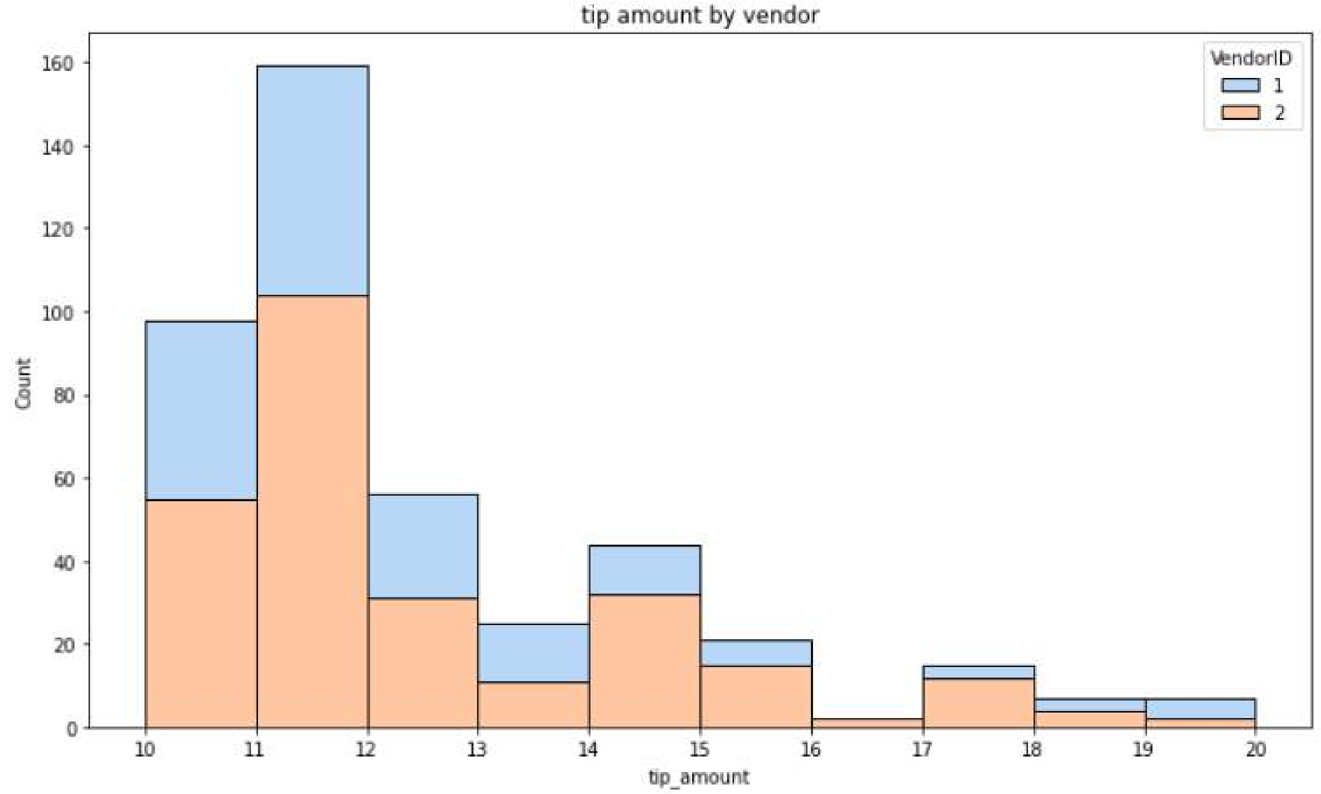
- **Visualizations revealed that trip_distance, total_amount, and tip_amount contain outliers, which need to be addressed before designing a model.**

EXPLORATORY DATA ANALYSIS - 4

- Tip_amount by vendor:



- Tip_amount by vendor for tips more than $10:

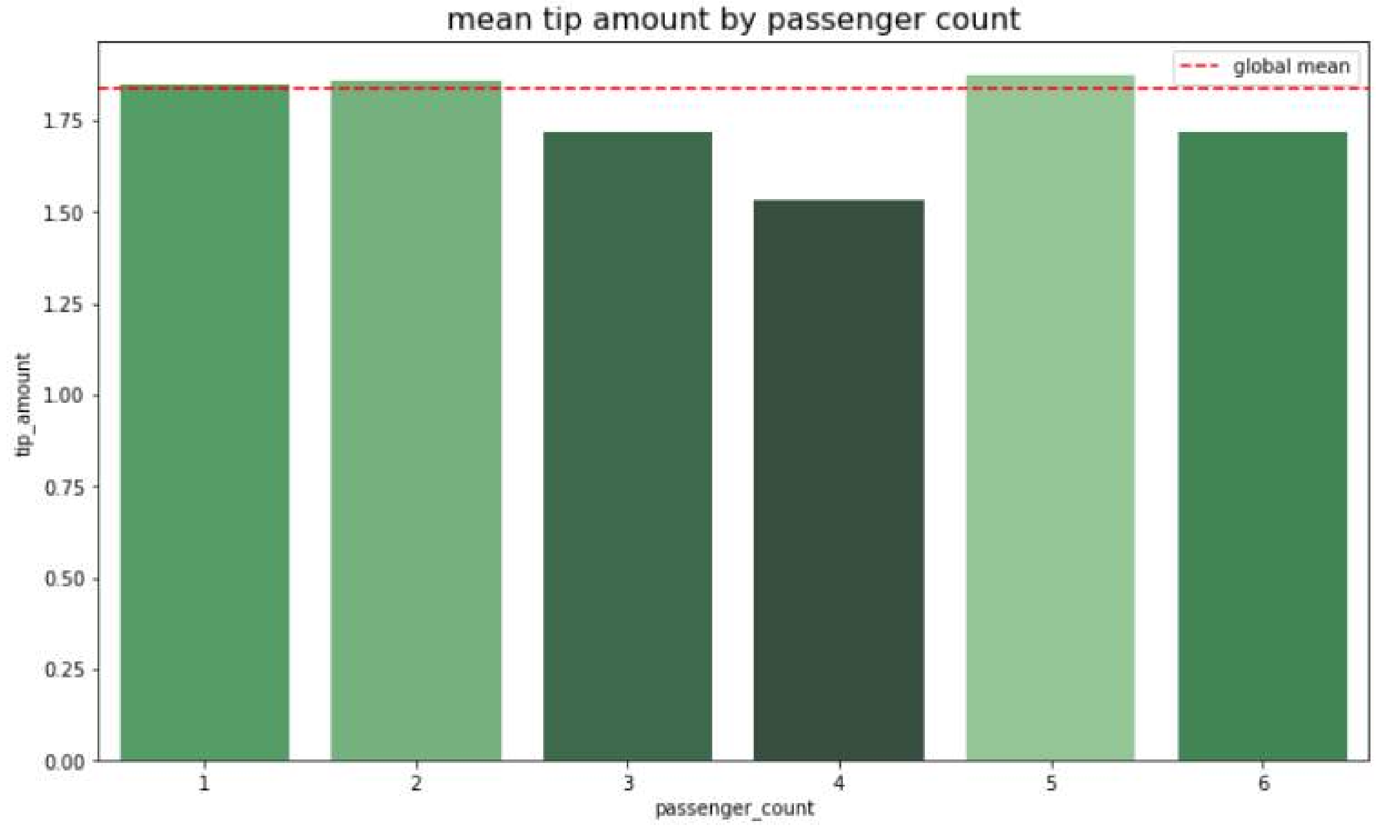


- The passenger_count represented in the data:

```
1    16117
2     3305
5     1143
3      953
6      693
4      455
0       33
Name: passenger_count, dtype: int64
```

- The mean tips by passenger_count:

passenger_count	tip_amount
0	2.135758
1	1.848920
2	1.856378
3	1.716768
4	1.530264
5	1.873185
6	1.720260

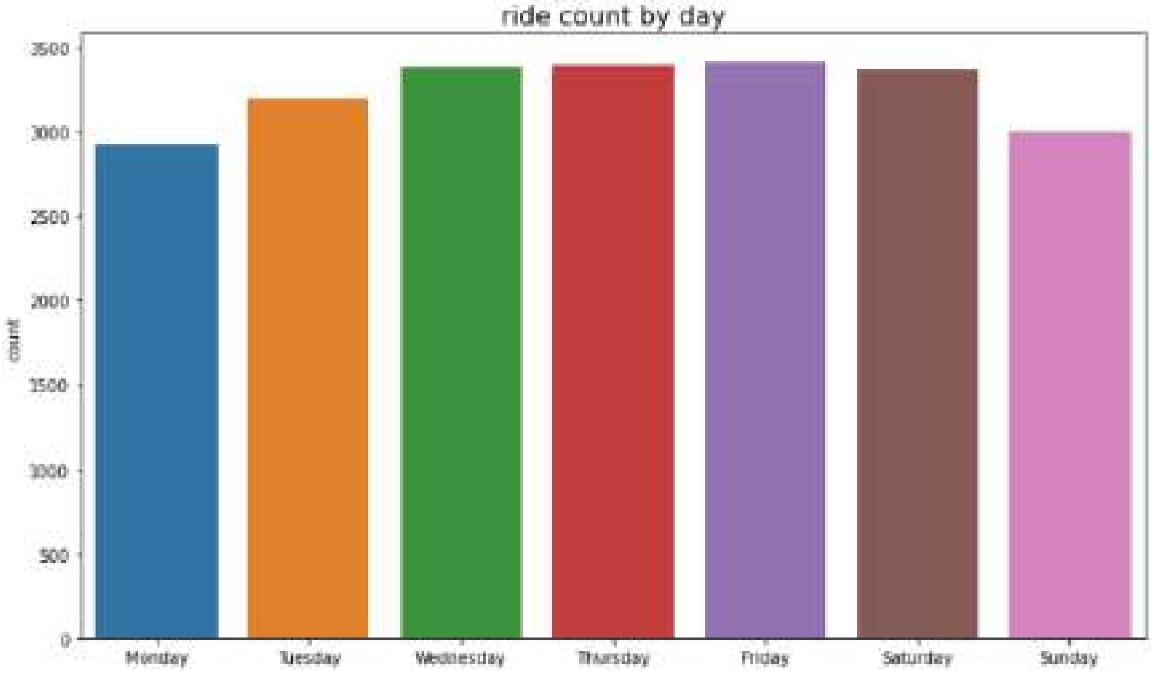


- **The mean tip amount varies little by passenger count, except for a drop in four-passenger rides, likely due to their rarity in the dataset (apart from zero-passenger rides).**

EXPLORATORY DATA ANALYSIS - 5

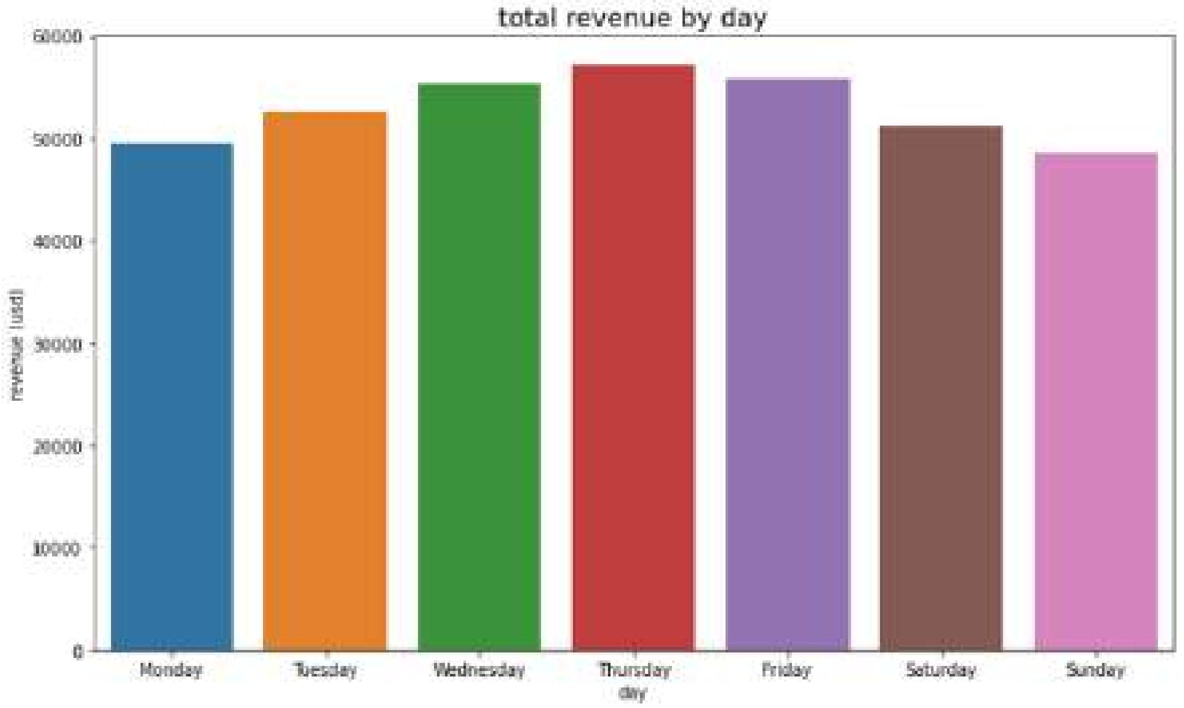
- The ride_count by day represented in the data:

Monday	2931
Tuesday	3198
Wednesday	3390
Thursday	3402
Friday	3413
Saturday	3367
Sunday	2998
Name: day, dtype: int64	



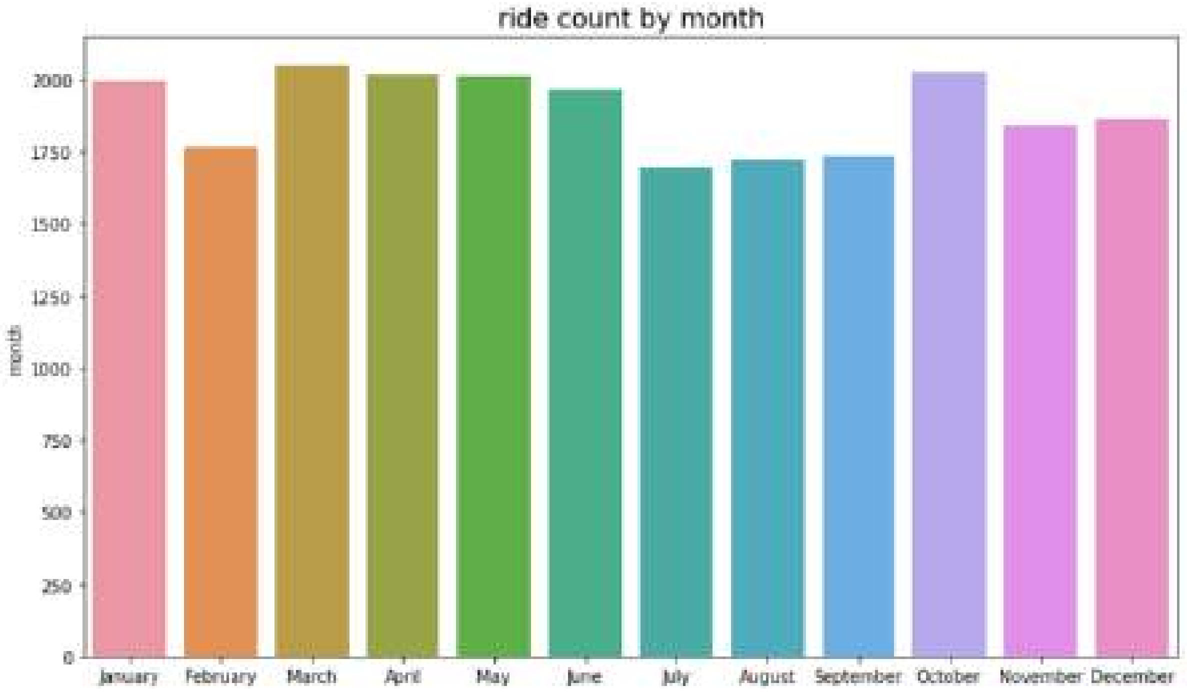
- The total_revenue by day represented in the data:

day	total_amount
Monday	49574.37
Tuesday	52527.14
Wednesday	55310.47
Thursday	57181.91
Friday	55818.74
Saturday	51195.40
Sunday	48624.06



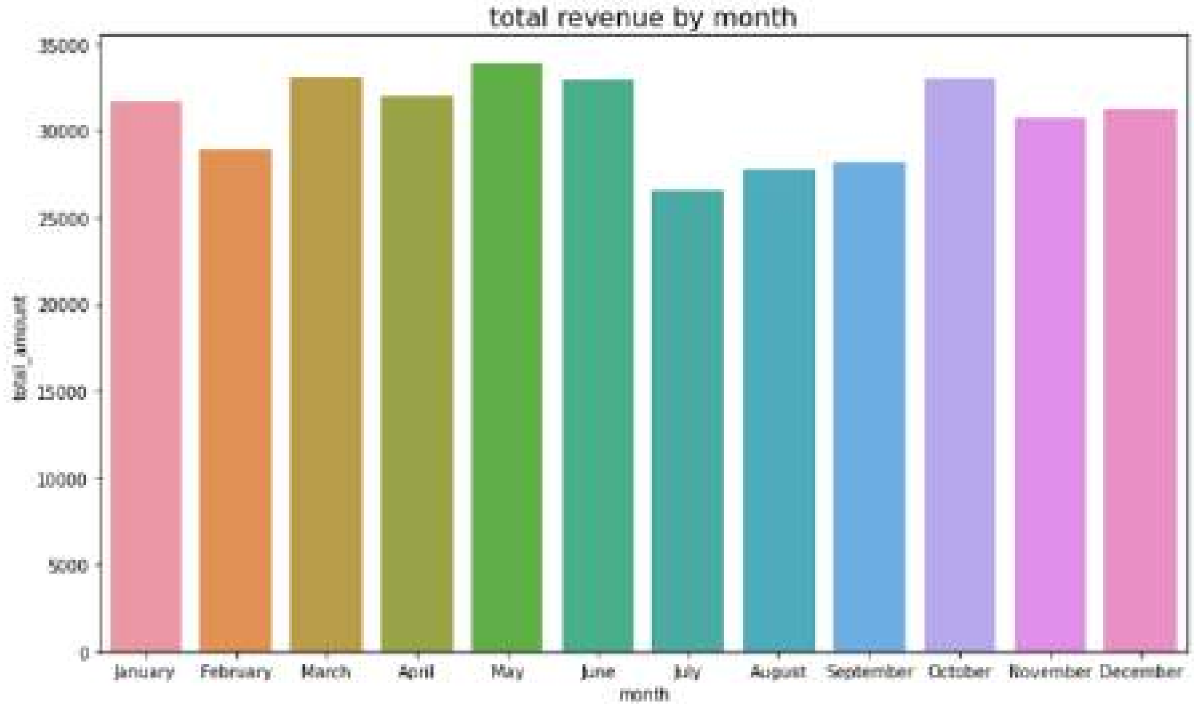
- The ride_count by month represented in the data:

January	1997
February	1769
March	2049
April	2019
May	2013
June	1964
July	1697
August	1724
September	1734
October	2027
November	1843
December	1863
Name: month, dtype: int64	



- The total_revenue by month represented in the data:

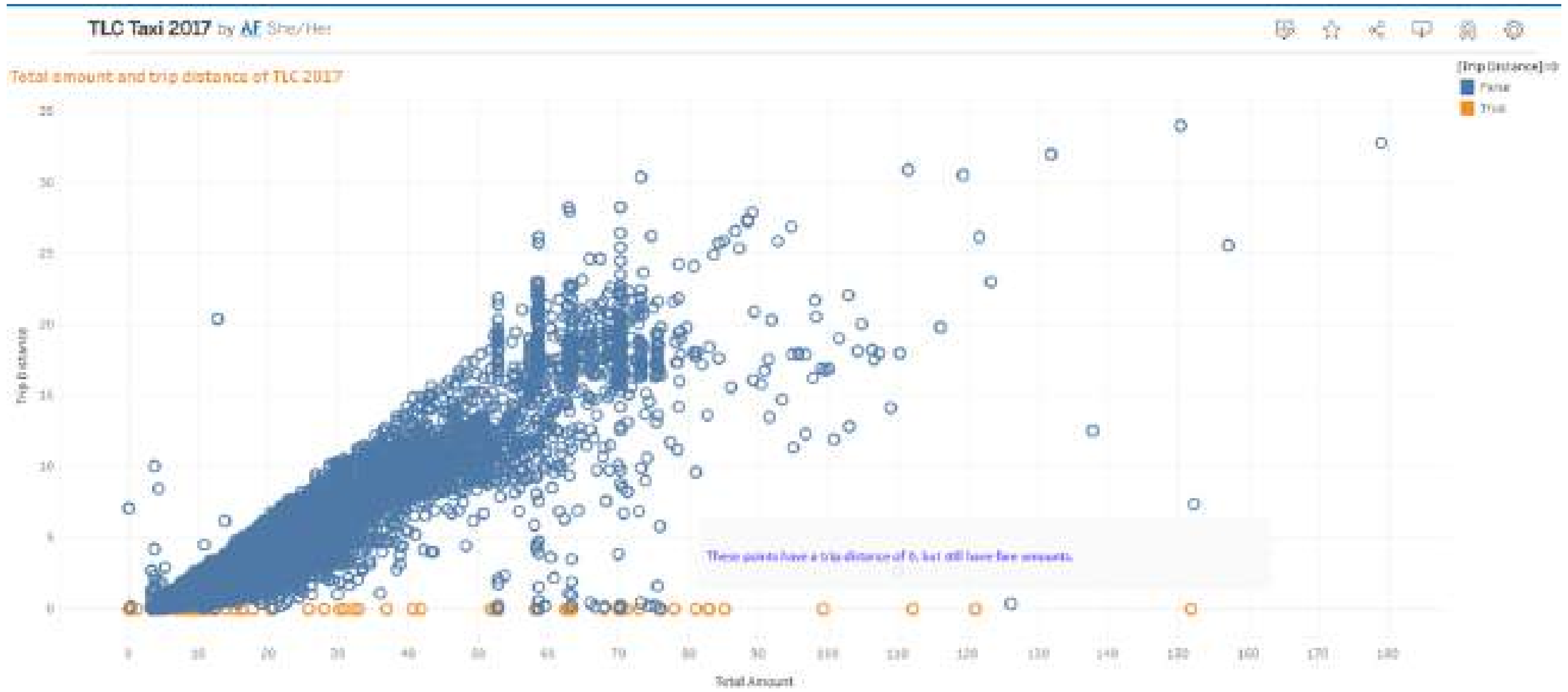
month	total_amount
January	31735.25
February	28937.89
March	33085.89
April	32012.54
May	33828.58
June	32920.52
July	26617.64
August	27759.56
September	28206.38
October	33065.83
November	30800.44
December	31261.57



- **Monthly revenue generally follows the pattern of monthly rides, the lowests are the summer months of July, August, and September, and also one in February.**

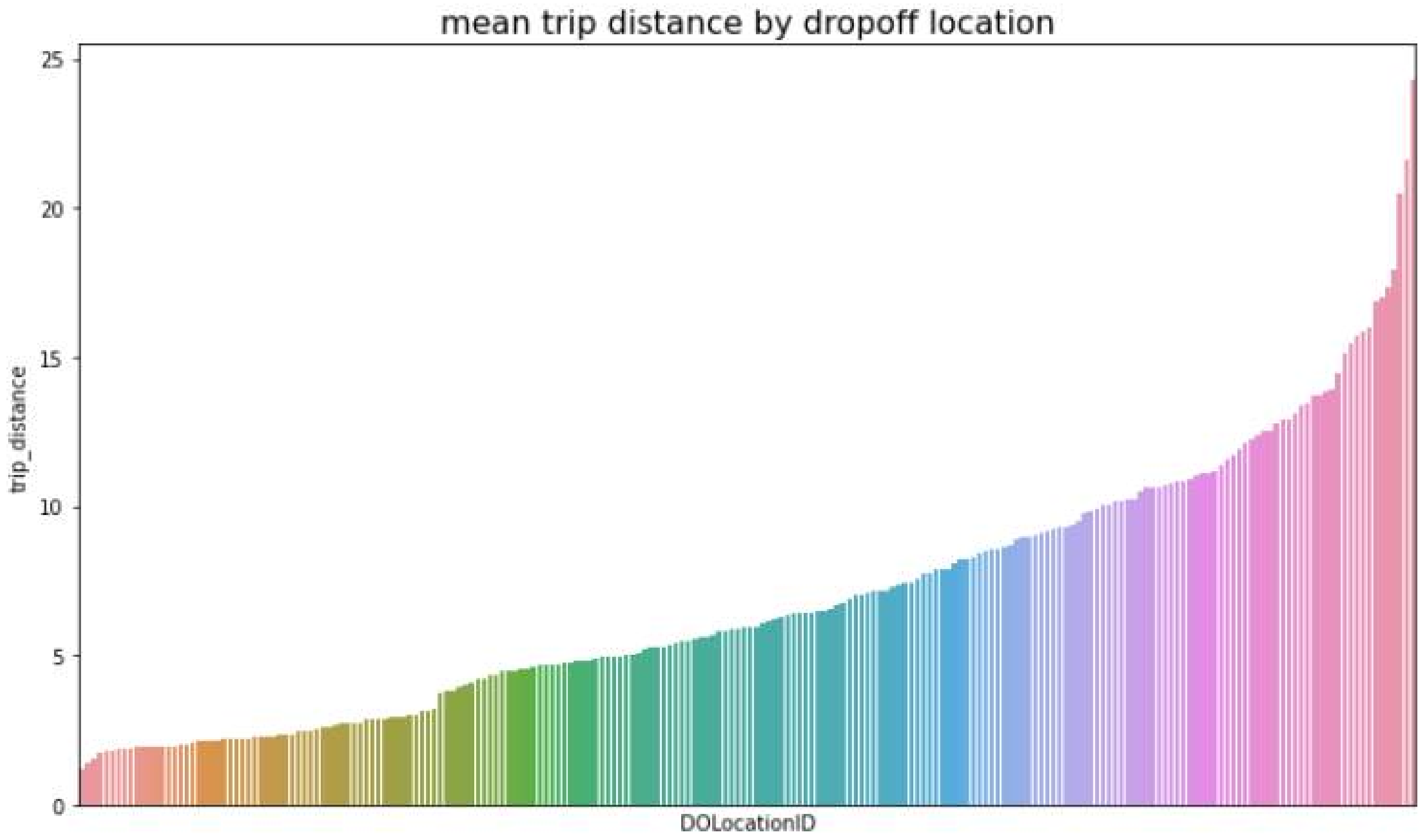
EXPLORATORY DATA ANALYSIS - 6

- Data relationship between total_amount and trip_distance.

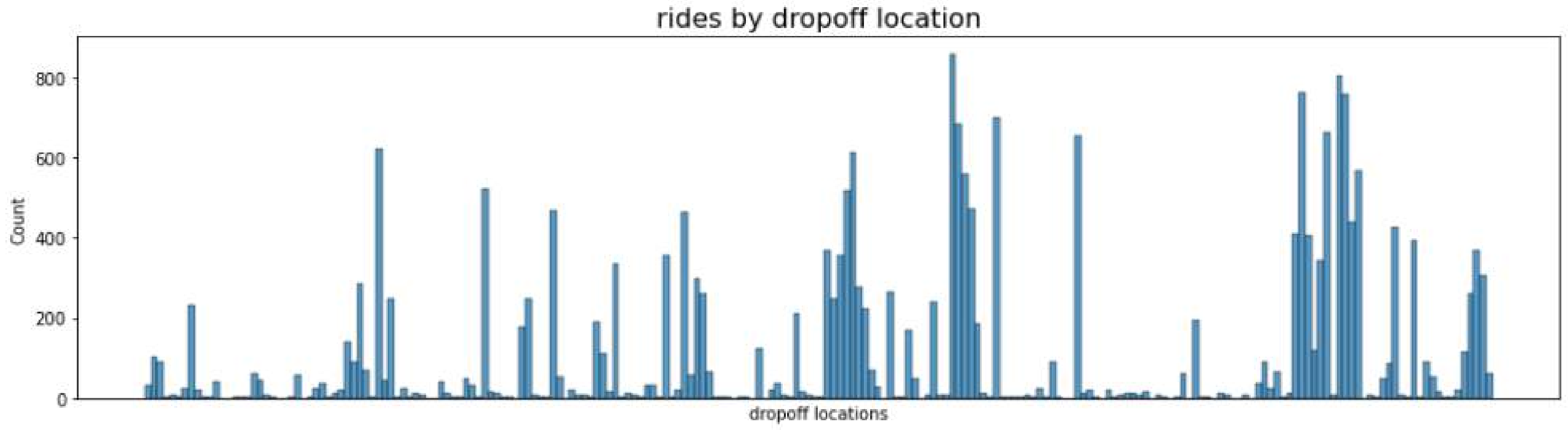


- **There are several trips with a trip distance of "0.0."**
- **What might these trips represent? This issue needs to be addressed before designing a model to prevent it from impacting the results.**

- The mean_trip_distance was calculated for each of the 216 DOLocationID points.



- The total ride count was calculated for each of the 216 DOLocationID points.



A/B TEST - 1

- The goal is to use statistical analysis, including descriptive statistics and hypothesis testing in Python, to examine the relationship between payment type and fare amount. For example, we aim to determine if customers paying with credit cards tend to pay higher fares than those using cash. The A/B test results will help identify ways to increase revenue for taxi drivers.

- **Analysis:**

Descriptive statistics help us understand the data and compare the average fare amounts between payment types. Initial results suggest that credit card users tend to pay higher fares than cash users. However, this difference might be due to random sampling rather than a true effect.

```
payment_type
1    13.429748
2    12.213546
3    12.186116
4     9.913043
Name: fare_amount, dtype: float64
```

Note: In the dataset, `payment_type` is encoded in integers:

- 1: Credit card
- 2: Cash
- 3: No charge
- 4: Dispute
- 5: Unknown

- To confirm if the difference is statistically significant, a hypothesis test is performed.
- Hypothesis test is the main component of A/B test.
- A two sample t hypothesis tests also known as independent t-test is used to analyze the difference between two unknown population means.

- **Experiment Setup:**

The sample data comes from an experiment where customers were randomly assigned to two groups:
1.) Customers required to pay with a credit card.
2.) Customers required to pay with cash.

This random assignment is crucial to draw causal conclusions about the effect of payment methods on fare amounts.

A/B TEST - 2

- The steps for conducting a hypothesis test:

1. State the null hypothesis and the alternative hypothesis
2. Choose a signficance level
3. Find the p-value
4. Reject or fail to reject the null hypothesis

- The null hypothesis (H0): **There is no difference** in the average fare amount between customers who use credit cards and who use cash.
- The alternative hypothesis (HA): **There is a difference** in the average fare amount between customers who use credit cards and who use cash.

- The significance level is set at 5%, and a two-sample t-test is conducted.

```
credit_card = taxi_data[taxi_data["payment_type"]==1]["fare_amount"]
cash = taxi_data[taxi_data["payment_type"]==2]["fare_amount"]
stats.ttest_ind(a=credit_card, b=cash, equal_var=False)
```

```
Ttest_indResult(statistic=6.866800855655372, pvalue=6.797387473030518e-12)
```

- p-value is smaller than the significance level of 5%. It rejects the null hypothesis.
- It means **there is a statistically significant difference** in the average fare amount between customers who use credit cards and customers who use cash.

- **Business Insight:**
- **The results of the hypothesis test suggest that encouraging customers to pay with credit cards can help generate more revenue.**

- **Assumptions and Limitations:**

- This project assumes that customers were required to use a specific payment method (credit card or cash) and always complied with this requirement. However, the data was not collected this way. To perform an A/B test, we had to randomly group data entries based on payment method.
- This dataset does not account for other potential factors. For instance, customers may prefer paying with credit cards for longer or more farther trips because they might not carry enough cash. In other words, it is more likely that the fare amount influences the choice of payment method, rather than the other way around.

REGRESSION ANALYSIS - 1

- Regression analysis simplifies complex data relationships. Multiple linear regression is a method used to estimate the relationship between a single continuous dependent variable (e.g., taxi fares) and two or more independent variables (e.g., trip distance, payment type). This approach allows you to analyze the impact of multiple factors simultaneously, providing a more comprehensive and flexible understanding of the data.

- Project Goal: Build a **multiple linear regression model to predict taxi fares.**

A. Exploratory Data Analysis (EDA)

- Convert tpep_pickup_datetime & tpep_dropoff_datetime datatype from object to datetime
- Create a new column called `duration` that represents the total number of minutes that each taxi ride took.

```
tpep_pickup_datetime"] = pd.to_datetime(df0["tpep_pickup_datetime"], format='%m/%d/%Y %I:%M:%S %p')
tpep_dropoff_datetime'] = pd.to_datetime(df0['tpep_dropoff_datetime'], format='%m/%d/%Y %I:%M:%S %p')
duration"] = (df0['tpep_dropoff_datetime'] - df0['tpep_pickup_datetime'])/np.timedelta64(1,'m')
```

```
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             22699 non-null  int64
 1   VendorID               22699 non-null  int64
 2   tpep_pickup_datetime   22699 non-null  object
 3   tpep_dropoff_datetime  22699 non-null  object
 4   passenger_count        22699 non-null  int64
 5   trip_distance          22699 non-null  float64
 6   RatecodeID             22699 non-null  int64
 7   store_and_fwd_flag     22699 non-null  object
 8   PULocationID           22699 non-null  int64
 9   DOLocationID           22699 non-null  int64
 10  payment_type           22699 non-null  int64
 11  fare_amount            22699 non-null  float64
 12  extra                  22699 non-null  float64
 13  mta_tax                22699 non-null  float64
 14  tip_amount             22699 non-null  float64
 15  tolls_amount           22699 non-null  float64
 16  improvement_surcharge  22699 non-null  float64
 17  total_amount           22699 non-null  float64
dtypes: float64(8), int64(7), object(3)
```

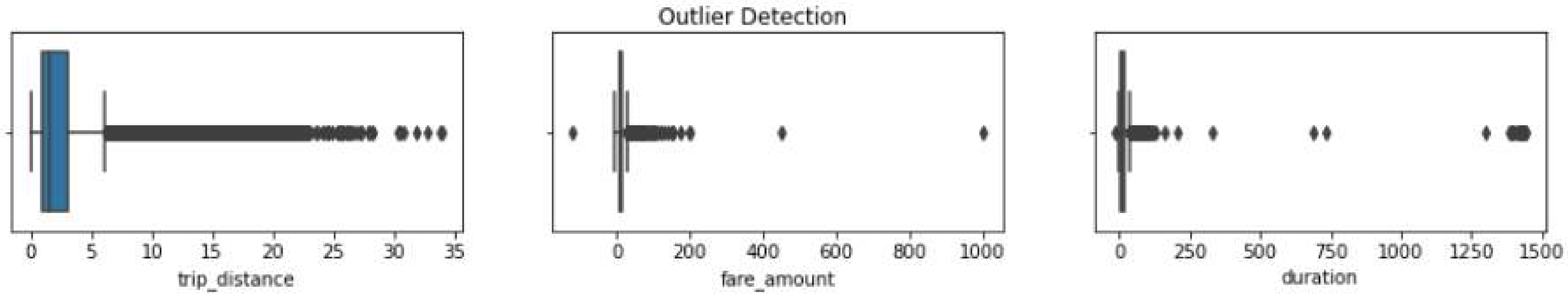
```
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             22699 non-null  int64
 1   VendorID               22699 non-null  int64
 2   tpep_pickup_datetime   22699 non-null  datetime64[ns]
 3   tpep_dropoff_datetime  22699 non-null  datetime64[ns]
 4   passenger_count        22699 non-null  int64
 5   trip_distance          22699 non-null  float64
 6   RatecodeID             22699 non-null  int64
 7   store_and_fwd_flag     22699 non-null  object
 8   PULocationID           22699 non-null  int64
 9   DOLocationID           22699 non-null  int64
 10  payment_type           22699 non-null  int64
 11  fare_amount            22699 non-null  float64
 12  extra                  22699 non-null  float64
 13  mta_tax                22699 non-null  float64
 14  tip_amount             22699 non-null  float64
 15  tolls_amount           22699 non-null  float64
 16  improvement_surcharge  22699 non-null  float64
 17  total_amount           22699 non-null  float64
 18  duration               22699 non-null  float64
dtypes: datetime64[ns](2), float64(9), int64(7), object(1)
```

- The data types of tpep_pickup_datetime and tpep_dropoff_datetime were converted from object to datetime format.

- The dataset now contains 22,699 rows and 19 columns, compared to the previous 18 columns. One new column, duration, has been added.

REGRESSION ANALYSIS - 2

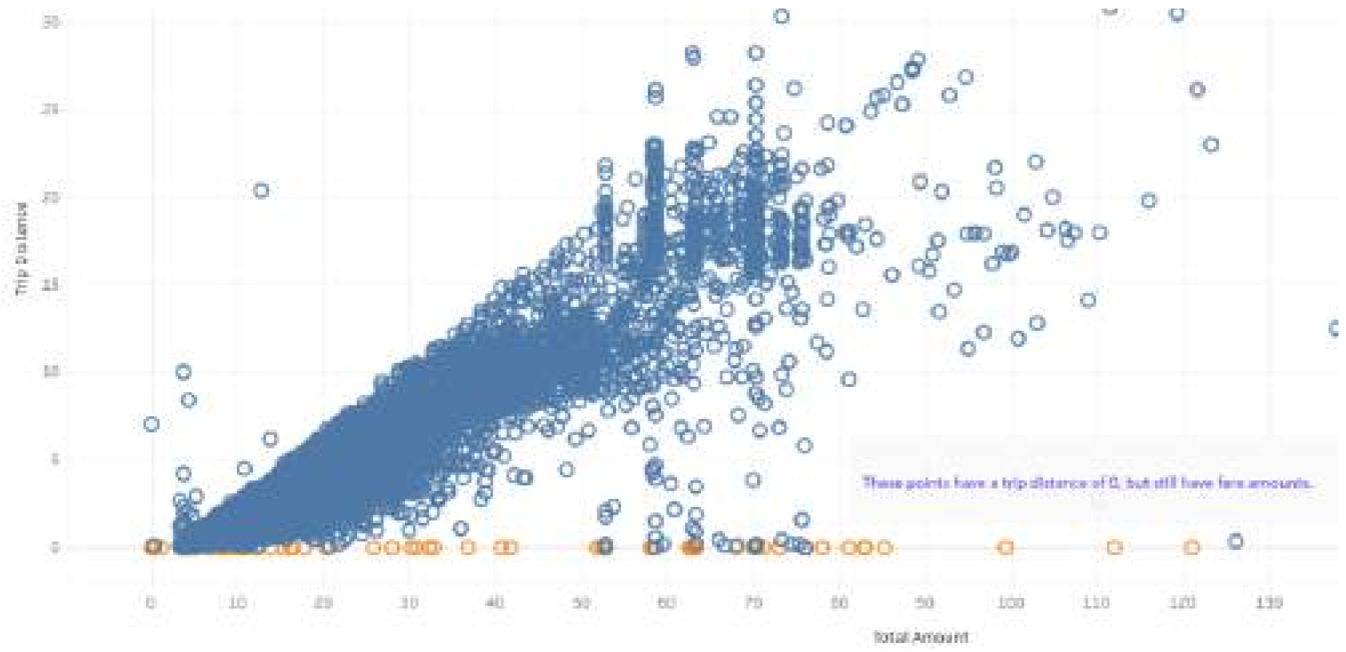
- Identify outliers by visualizing the data distributions for trip_distance, fare_amount, and duration.



	trip_distance	fare_amount	duration
count	22699.000000	22699.000000	22699.000000
mean	2.913313	13.026629	17.013777
std	3.653171	13.243791	61.996482
min	0.000000	-120.000000	-16.983333
25%	0.990000	6.500000	6.650000
50%	1.610000	9.500000	11.183333
75%	3.060000	14.500000	18.383333
max	33.960000	999.990000	1439.550000

- **trip_distance with maximum values less than 35 miles and the overall distribution of the data, it is reasonable to leave these values as they are and not modify them.**
- **The values for fare_amount (including 0 or negative values) and duration appear to have problematic outliers at both the lower and higher ends.**

- Some trips have distances of 0. Are these errors, or very short trips rounded down? To check: sort the column values, remove duplicates, and inspect the 10 smallest values—are they rounded or precise?




```
sorted(set(df0["trip_distance"]))[:10]
```

```
[0.0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09]
```

- Distances are recorded with high precision, but a trip distance of 0 could occur if a passenger summoned a taxi but canceled.

```
sum(df0["trip_distance"]==0)
```

```
148
```

- Additionally, consider whether the number of zero-distance trips is significant enough to be a concern. There are 148 rides have a trip_distance of zero.

REGRESSION ANALYSIS - 3

- The **`fare_amount` and `duration`** column have problematic values at both the lower and upper extremities. **Replace outliers**:
- Low values: There should be no values that represent negative. Impute all negative with `0`.
- High values: Impute high values with `Q3 + (6 * IQR)`.
- 6 is the IQR factor (a parameter representing the value of x in the formula Q3+(x×IQR)). It is used to define the maximum threshold for identifying outliers—data points exceeding this threshold are considered outliers.

- Handle `fare_amount` outliers:

```
count    22699.000000
mean        13.026629
std         13.243791
min       -120.000000
25%          6.500000
50%          9.500000
75%         14.500000
max        999.990000
Name: fare_amount, dtype: float64
```

```
# Impute values less than $0 with 0
df0.loc[df0["fare_amount"] < 0, "fare_amount"] = 0
df0["fare_amount"].min()
```

```
0.0
```

Now impute the maximum value as `Q3 + (6 * IQR)`.

```
outlier_imputer(["fare_amount"], 6)
```

```
fare_amount
q3: 14.5
upper_threshold: 62.5
count    22699.000000
mean        12.897913
std         10.541137
min          0.000000
25%          6.500000
50%          9.500000
75%         14.500000
max         62.500000
Name: fare_amount, dtype: float64
```

- Handle `duration` outliers:

```
count    22699.000000
mean        17.013777
std         61.996482
min        -16.983333
25%          6.650000
50%         11.183333
75%         18.383333
max       1439.550000
Name: duration, dtype: float64
```

```
# Impute a 0 for any negative values
df0.loc[df0["duration"] < 0, "duration"] = 0
df0["duration"].min()
```

```
0.0
```

```
# Impute the high outliers
outlier_imputer(["duration"], 6)
```

```
duration
q3: 18.383333333333333
upper_threshold: 88.78333333333333
count    22699.000000
mean        14.460555
std         11.947043
min          0.000000
25%          6.650000
50%         11.183333
75%         18.383333
max         88.783333
Name: duration, dtype: float64
```

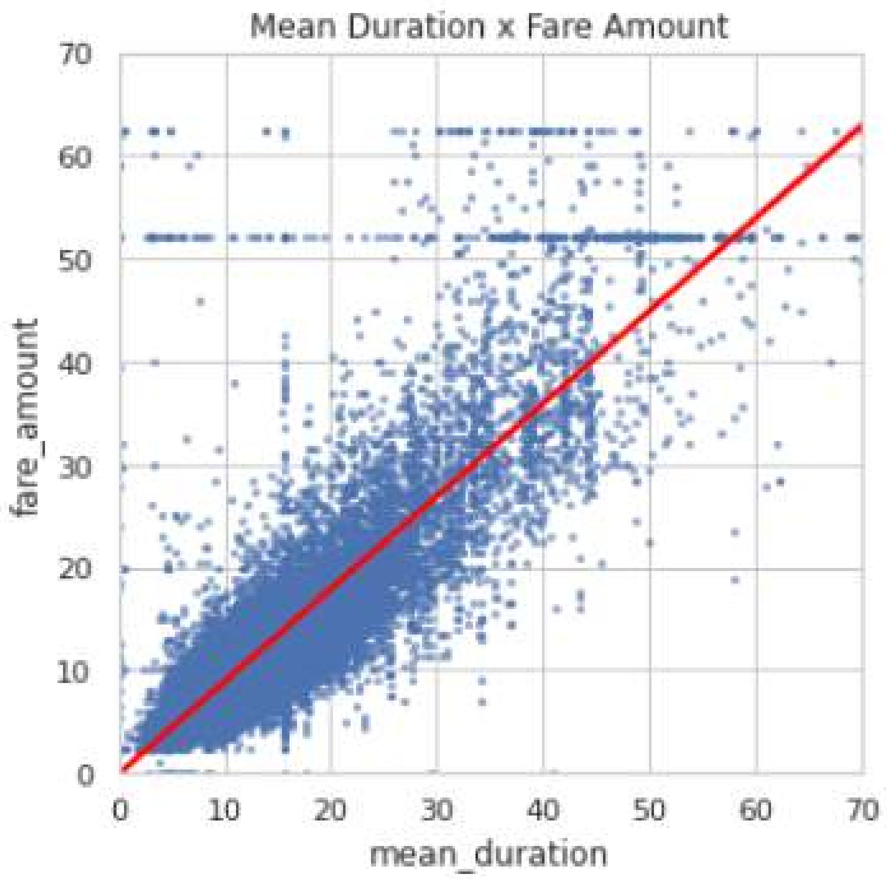
B. Feature Engineering

- The model will not know the duration of a trip until after the trip occurs, so you cannot train a model that uses this feature.
- Create a helper column called pickup_dropoff to store the unique combination of pickup and dropoff location IDs for each row.
- Create a mean_distance column by grouping trips based on the pickup_dropoff values and calculating the mean values based on trip_distance.
- Create a mean_duration column by grouping trips based on the pickup_dropoff values and calculating the mean values based duration column.
- Create 3 columns: day, month, and rush_hour. The rush_hour column includes trips on weekdays (Monday to Friday) during either 06:00–10:00 or 16:00–20:00.

```python
df0["pickup_dropoff"] = df0["PULocationID"].astype(str) + ' ' + df0["DOLocationID"].astype(str)
grouped = df0.groupby("pickup_dropoff").mean(numeric_only=True)
grouped_dict = grouped.to_dict()

grouped_dict1 = grouped_dict["trip_distance"]
df0["mean_distance"] = df0["pickup_dropoff"]
df0["mean_distance"] = df0["mean_distance"].map(grouped_dict1)

grouped_dict2 = grouped_dict["duration"]
df0["mean_duration"] = df0["pickup_dropoff"]
df0["mean_duration"] = df0["mean_duration"].map(grouped_dict2)
```

```python
df0["day"] = df0["tpep_pickup_datetime"].dt.day_name().str.lower()
df0["month"] = df0["tpep_pickup_datetime"].dt.strftime('%b').str.lower()
df0["rush_hour"] = df0["tpep_pickup_datetime"].dt.hour

df0.loc[df0["day"].isin(['saturday', 'sunday']), 'rush_hour'] = 0

def rush_hourizer(hour):
    if 6 <= hour['rush_hour'] < 10:
        val = 1
    elif 16 <= hour['rush_hour'] < 20:
        val = 1
    else:
        val = 0
    return val

df0.loc[(df0.day != 'saturday') & (df0.day != 'sunday'), 'rush_hour'] = df0.apply(rush_hourizer, axis=1)
```

- Visualize the relationship between mean_duration and fare_amount:

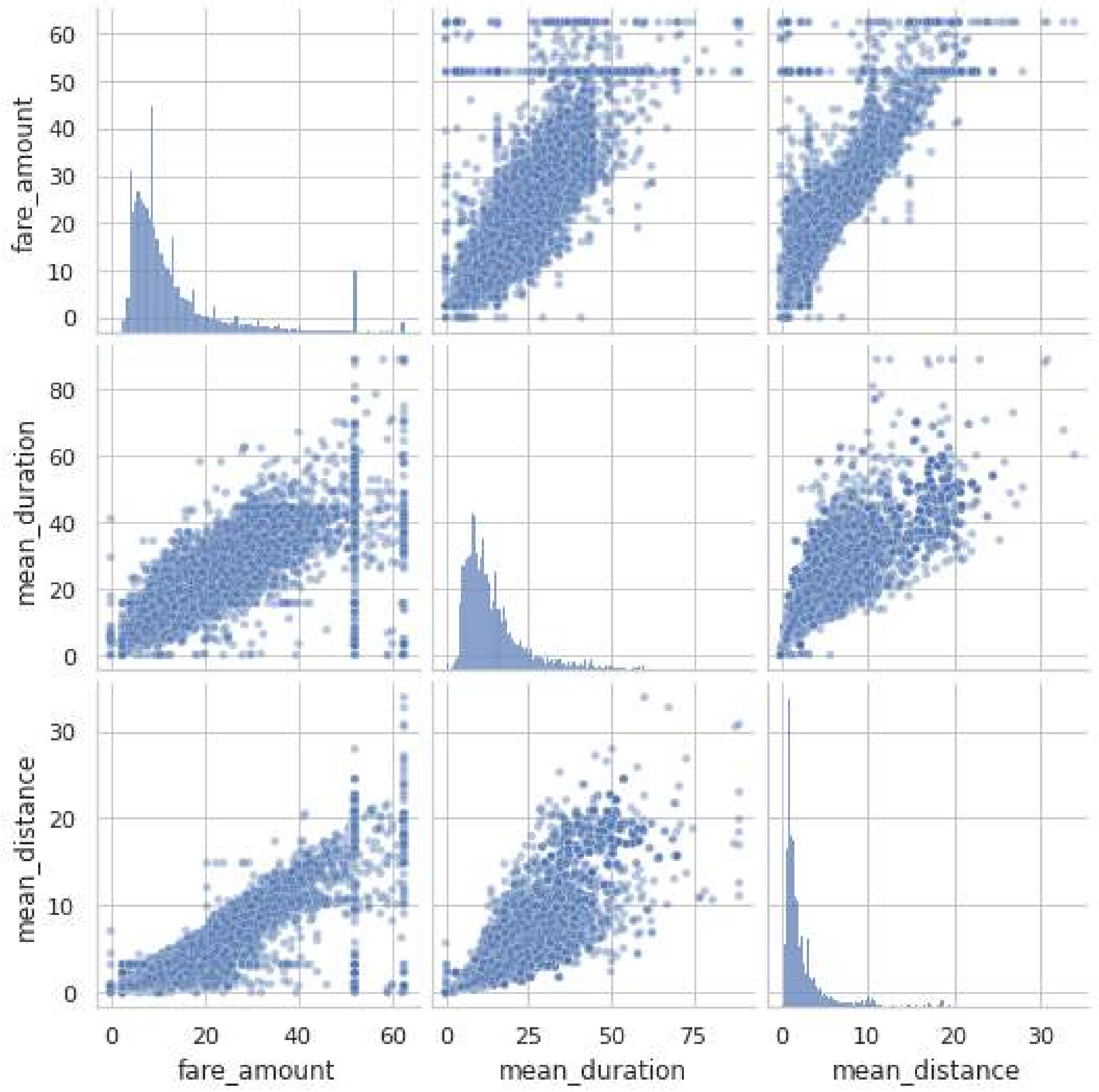


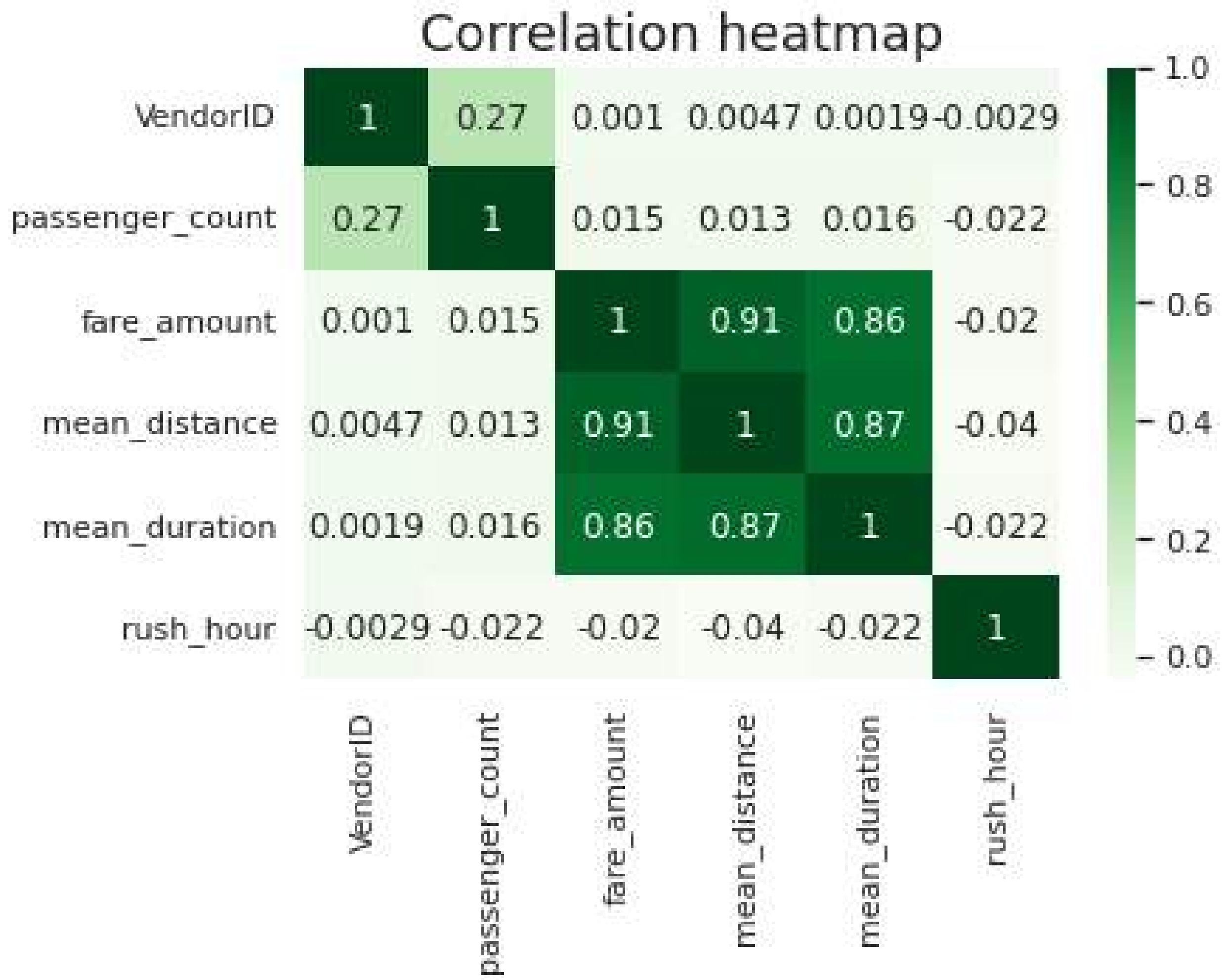
- The graphic shows that mean_duration is correlated with fare_amount.
- Horizontal lines around $63 indicate the maximum value set for outliers, with all former outliers now capped at $62.50.

REGRESSION ANALYSIS - 5

- Visualize pairwise relationships between fare_amount, mean_duration, and mean_distance:

```
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 25 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             22699 non-null  int64
 1   VendorID               22699 non-null  int64
 2   tpep_pickup_datetime   22699 non-null  datetime64[ns]
 3   tpep_dropoff_datetime  22699 non-null  datetime64[ns]
 4   passenger_count        22699 non-null  int64
 5   trip_distance          22699 non-null  float64
 6   RatecodeID             22699 non-null  int64
 7   store_and_fwd_flag     22699 non-null  object
 8   PULocationID           22699 non-null  int64
 9   DOLocationID           22699 non-null  int64
 10  payment_type           22699 non-null  int64
 11  fare_amount            22699 non-null  float64
 12  extra                  22699 non-null  float64
 13  mta_tax                22699 non-null  float64
 14  tip_amount             22699 non-null  float64
 15  tolls_amount           22699 non-null  float64
 16  improvement_surcharge  22699 non-null  float64
 17  total_amount           22699 non-null  float64
 18  duration               22699 non-null  float64
 19  pickup_dropoff         22699 non-null  object
 20  mean_distance          22699 non-null  float64
 21  mean_duration          22699 non-null  float64
 22  day                    22699 non-null  object
 23  month                  22699 non-null  object
 24  rush_hour              22699 non-null  int64
dtypes: datetime64[ns](2), float64(11), int64(8), object(4)
```





	VendorID	passenger_count	fare_amount	mean_distance	mean_duration	rush_hour
VendorID	1.000000	0.266463	0.001045	0.004741	0.001876	-0.002874
passenger_count	0.266463	1.000000	0.014942	0.013428	0.015852	-0.022035
fare_amount	0.001045	0.014942	1.000000	0.910185	0.859105	-0.020075
mean_distance	0.004741	0.013428	0.910185	1.000000	0.874864	-0.039725
mean_duration	0.001876	0.015852	0.859105	0.874864	1.000000	-0.021583
rush_hour	-0.002874	-0.022035	-0.020075	-0.039725	-0.021583	1.000000

- Both **mean_distance (0.91) and mean_duration (0.86) are strongly correlated with the target variable, fare_amount.** They are also highly correlated with each other, with a Pearson correlation of 0.87.
- While highly correlated predictors can complicate statistical inferences in linear regression, they can still be effective for accurate predictions. Since the goal is to predict fare_amount for machine learning models, it’s worth including both variables in the model despite their correlation.

REGRESSION ANALYSIS - 7

C. Construct Model

```
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   VendorID         22699 non-null  int64
 1   passenger_count  22699 non-null  int64
 2   fare_amount      22699 non-null  float64
 3   mean_distance    22699 non-null  float64
 4   mean_duration    22699 non-null  float64
 5   rush_hour        22699 non-null  int64
dtypes: float64(3), int64(3)
```

- There are **6 columns will be used for modeling.**
- Fitting multiple linear regression models may require trial and error to select variables that fit an accurate model while maintaining model assumptions

1. Split data into outcome variable (X = fare_amount) and features (y = VendorID, passenger_count, mean_distance, mean_duration, rush_hour)

```
X = df1.drop(columns=["fare_amount"])
y = df1[["fare_amount"]]
```

2. Pre-process data (dummy encode for VendorID).

```
X["VendorID"] = X["VendorID"].astype(str)
X = pd.get_dummies(X, drop_first=True)
```

3. Split data into training (80%) and test set (20%) with random_state=0
4. Standardize the data use StandardScaler(), fit(), and transform() to standardize the X_train variables.
5. Instantiate the LinearRegression() model and fit it to the training data.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)

lr=LinearRegression()
lr.fit(X_train_scaled, y_train)
```

6. Evaluate model performance by calculating the residual sum of squares and the explained variance score (R^2) also calculate the Mean Absolute Error, Mean Squared Error, and the Root Mean Squared Error for train data and test data.

```
Training data:
Coefficient of determination: 0.8398434585044773
R^2: 0.8398434585044773
MAE: 2.186666416775414
MSE: 17.88973296349268
RMSE: 1.4787381163598285
```

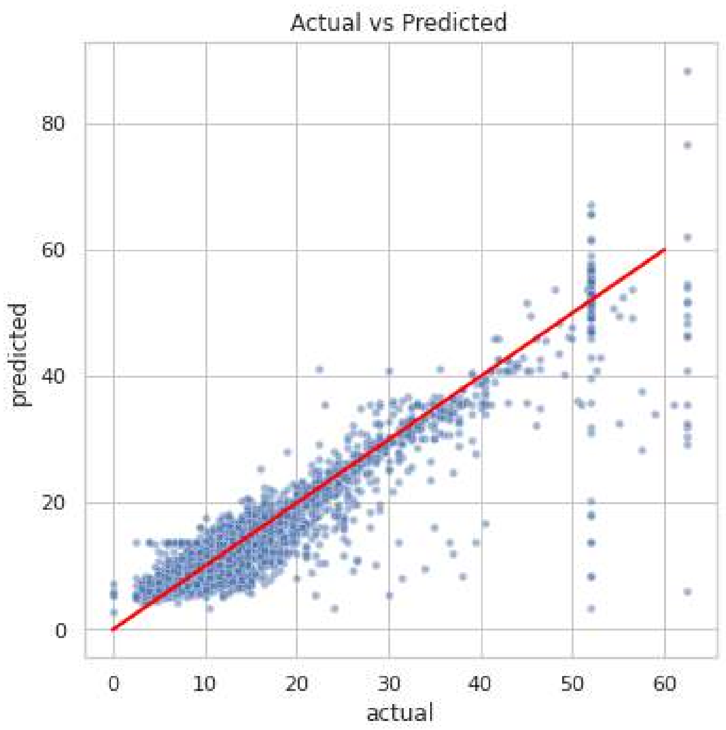
```
Test data:
Coefficient of determination: 0.8682583641795454
R^2: 0.8682583641795454
MAE: 2.1336549840593864
MSE: 14.326454156998944
RMSE: 3.785030271609323
```

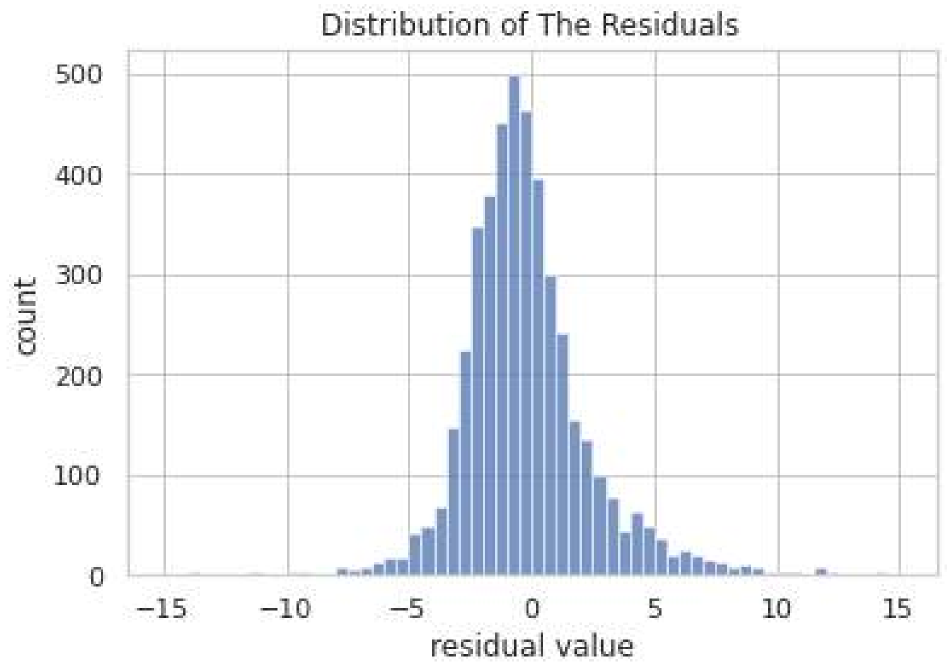
- The **model achieved** 84% performance on the training data and **87% on the test data.**

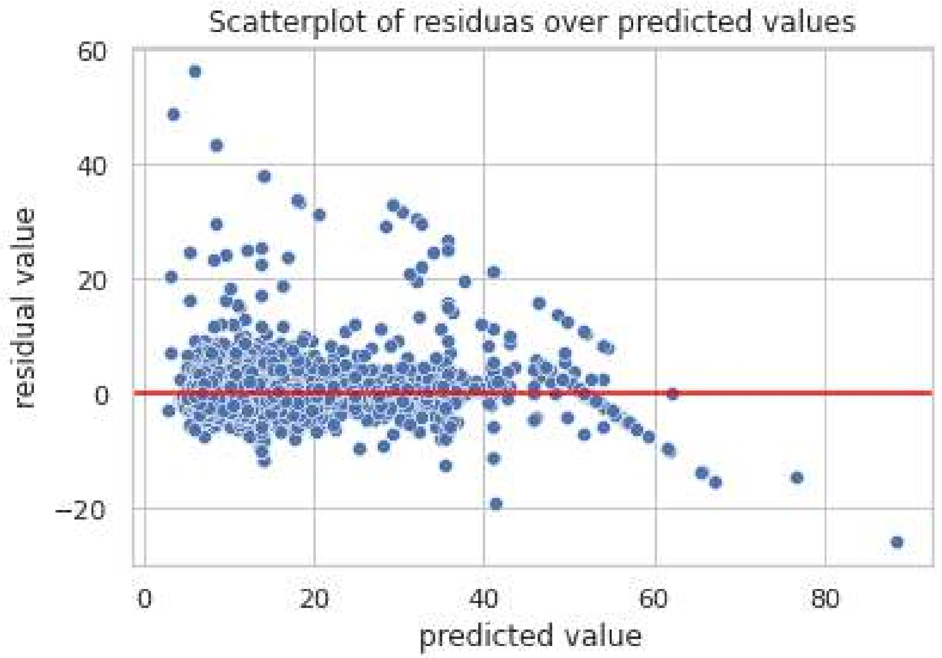
REGRESSION ANALYSIS - 8

D. Execute Model

- Visualize model results (actual,predicted, and residual for the testing set):







- The coefficients model:

	passenger_count	mean_distance	mean_duration	rush_hour	VendorID_2
0	0.030825	7.133867	2.812115	0.110233	-0.054373

- **`mean_distance` was the greatest feature in the model's final prediction.**
- This coefficient is controlling for other variables, *for every +1 change in standard deviation*, the fare amount increases by a mean of $7.13.

- Conclusion:

```
# 1. Calculate Standard Deviation of 'mean_distance' in X_train data

print(X_train["mean_distance"].std())

# 2. Divide the model coefficient by the standard deviation

print(7.133867 / X_train["mean_distance"].std())

3.574812975256415
1.9955916713344426
```

- **The fare increased by $7.13 for every 3.57 miles traveled** or an average of $2.00 per mile .
- Note that keeping some highly correlated features results in a wider confidence interval.

CLASSIFICATION ANALYSIS - 1

- Tree-based models can predict whether a customer is a generous tipper, but errors can have serious consequences:
- False Negatives: The model predicts a tip, but the customer doesn’t leave one. Drivers may become frustrated if the app frequently promises tips that don’t happen, leading to distrust.
- False Positives: The model predicts no tip, but the customer does leave one. Drivers might avoid picking up customers predicted not to tip, leaving those customers stranded and dissatisfied with the taxi service.
- Even if the model is accurate, it could unfairly discourage drivers from picking up customers who can’t afford to tip, reducing taxi accessibility and causing potential backlash. The risks and ethical concerns outweigh the benefits.

- Better Approach:
- Focus on **identifying generous tippers (those tipping 20% or more).** This strategy helps drivers increase earnings without excluding anyone. To build a better model, use data like tipping history, pickup/dropoff times and locations, estimated fares, and payment methods. Errors can have serious consequences:
- False Positives: The model predicts a tip ≥ 20%, but the tip doesn’t happen. This frustrates drivers who expected a generous tip.
- False Negatives: The model predicts a tip < 20%, but the customer tips generously. This harms customers, as drivers may pick someone else predicted to tip more.

A. Feature Engineering

- Perform feature selection, extraction, and transformation to prepare the data for modeling.

- The columns for mean_duration, mean_distance, and predicted fares were derived from the multiple linear regression analysis.
- Add a tip_percent column calculated as tip_amount / (total_amount - tip_amount).
- Create a generous column, which will serve as the target variable. This column is binary (0 = no, 1 = yes) and is based on whether the tip_percent is ≥ 20%.
- Add a day column derived from the tpep_pickup_datetime column.
- Create time-of-day columns with binary values (0 = no, 1 = yes), also based on tpep_pickup_datetime: am_rush: 06:00–10:00, daytime: 10:00–16:00, pm_rush: 16:00–20:00, nighttime: 20:00–06:00
- Add a month column derived from the tpep_pickup_datetime column.

```
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 21 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Unnamed: 0             22699 non-null  int64  
 1   VendorID               22699 non-null  int64  
 2   tpep_pickup_datetime   22699 non-null  object 
 3   tpep_dropoff_datetime  22699 non-null  object 
 4   passenger_count        22699 non-null  int64  
 5   trip_distance          22699 non-null  float64
 6   RatecodeID             22699 non-null  int64  
 7   store_and_fwd_flag     22699 non-null  object 
 8   PULocationID           22699 non-null  int64  
 9   DOLocationID           22699 non-null  int64  
 10  payment_type           22699 non-null  int64  
 11  fare_amount            22699 non-null  float64
 12  extra                  22699 non-null  float64
 13  mta_tax                22699 non-null  float64
 14  tip_amount             22699 non-null  float64
 15  tolls_amount           22699 non-null  float64
 16  improvement_surcharge  22699 non-null  float64
 17  total_amount           22699 non-null  float64
 18  mean_duration          22699 non-null  float64
 19  mean_distance          22699 non-null  float64
 20  predicted_fare         22699 non-null  float64
dtypes: float64(11), int64(7), object(3)
```

CLASSIFICATION ANALYSIS - 2


```
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 21 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             22699 non-null  int64
 1   VendorID               22699 non-null  int64
 2   tpep_pickup_datetime   22699 non-null  object
 3   tpep_dropoff_datetime  22699 non-null  object
 4   passenger_count        22699 non-null  int64
 5   trip_distance          22699 non-null  float64
 6   RatecodeID             22699 non-null  int64
 7   store_and_fwd_flag     22699 non-null  object
 8   PULocationID           22699 non-null  int64
 9   DOLocationID           22699 non-null  int64
 10  payment_type           22699 non-null  int64
 11  fare_amount            22699 non-null  float64
 12  extra                  22699 non-null  float64
 13  mta_tax                22699 non-null  float64
 14  tip_amount             22699 non-null  float64
 15  tolls_amount           22699 non-null  float64
 16  improvement_surcharge  22699 non-null  float64
 17  total_amount           22699 non-null  float64
 18  mean_duration          22699 non-null  float64
 19  mean_distance          22699 non-null  float64
 20  predicted_fare         22699 non-null  float64
dtypes: float64(11), int64(7), object(3)
```

```
Int64Index: 15265 entries, 0 to 22698
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   VendorID          15265 non-null  int64
 1   passenger_count   15265 non-null  int64
 2   RatecodeID        15265 non-null  int64
 3   PULocationID      15265 non-null  int64
 4   DOLocationID      15265 non-null  int64
 5   mean_duration     15265 non-null  float64
 6   mean_distance     15265 non-null  float64
 7   predicted_fare    15265 non-null  float64
 8   generous          15265 non-null  int64
 9   day               15265 non-null  object
 10  am_rush           15265 non-null  int64
 11  daytime           15265 non-null  int64
 12  pm_rush           15265 non-null  int64
 13  nighttime         15265 non-null  int64
 14  month             15265 non-null  object
dtypes: float64(3), int64(10), object(2)
```

```
df2["generous"].value_counts(normalize=True)

1    0.526368
0    0.473632
Name: generous, dtype: float64
```

```
1    15265
2     7267
3      121
4       46
Name: payment_type, dtype: int64
```

```
Avg. cc tip:  2.7298001965279934
Avg. cash tip:  0.0
```

```
cols_to_str = ['RatecodeID', 'PULocationID', 'DOLocationID', 'VendorID']

for col in cols_to_str:
    df1[col] = df1[col].astype('str')

df2 = pd.get_dummies(df1, drop_first=True)
```

- Drop columns: 'Unnamed: 0', 'tpep_pickup_datetime', 'tpep_dropoff_datetime', 'payment_type', 'trip_distance', 'store_and_fwd_flag', 'payment_type', 'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge', 'total_amount', 'tip_percent']

- 15,265 records were selected, focusing solely on credit card payments, with an average tip of $2.73.

- Variable encoding : the columns 'RatecodeID,' 'PULocationID,' 'DOLocationID,' and 'VendorID' are numeric but represent categorical data. Convert them to str first so the get_dummies() function can process them into binary variables.

- Check the balance of the target variable, generous columns: The dataset is almost balanced. (52.6% 1 or True vs 47.4% 0 or False)

- Choosing the Right Metric:

- False Positives: The model predicts a tip ≥ 20%, but the tip doesn't happen. This frustrates drivers who expected a generous tip.
- False Negatives: The model predicts a tip < 20%, but the customer tips generously. This harms customers, as drivers may pick someone else predicted to tip more.
- Since the stakes are balanced—supporting drivers while avoiding customer frustration—the best metric is F1 score that places equal weight on true postives and false positives, and so therefore on precision and recall.

CLASSIFICATION ANALYSIS - 3

B. Construct Modell

```
rf = RandomForestClassifier(random_state=42)

cv_params = {'max_depth': [None],
             'max_features': [1.0],
             'max_samples': [0.7],
             'min_samples_leaf': [1],
             'min_samples_split': [2],
             'n_estimators': [300]
             }

scoring = {'accuracy', 'precision', 'recall', 'f1'}

rf1 = GridSearchCV(rf, cv_params, scoring=scoring, cv=4, refit='f1')

rf1.fit(X_train, y_train)
```

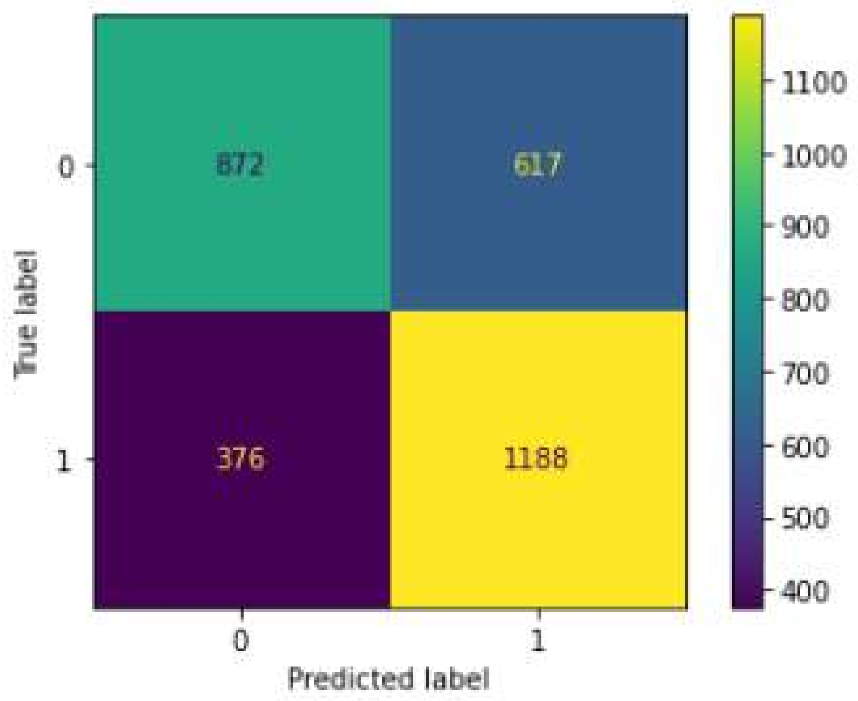
```
xgb = XGBClassifier(objective='binary:logistic', random_state=0)

cv_params= {'learning_rate': [0.1],
            'max_depth': [8],
            'min_child_weight': [2],
            'n_estimators': [100]
            }

scoring = {'accuracy', 'precision', 'recall', 'f1'}

xgb1 = GridSearchCV(xgb, cv_params, scoring=scoring, cv=4, refit='f1')

xgb1.fit(X_train, y_train)
```

	model	precision	recall	F1	accuracy
0	RF CV	0.679793	0.767111	0.720795	0.685146
0	RF test	0.658172	0.759591	0.705254	0.674746
0	XGB CV	0.689592	0.791221	0.736901	0.700622
0	XGB test	0.675690	0.797954	0.731750	0.700295

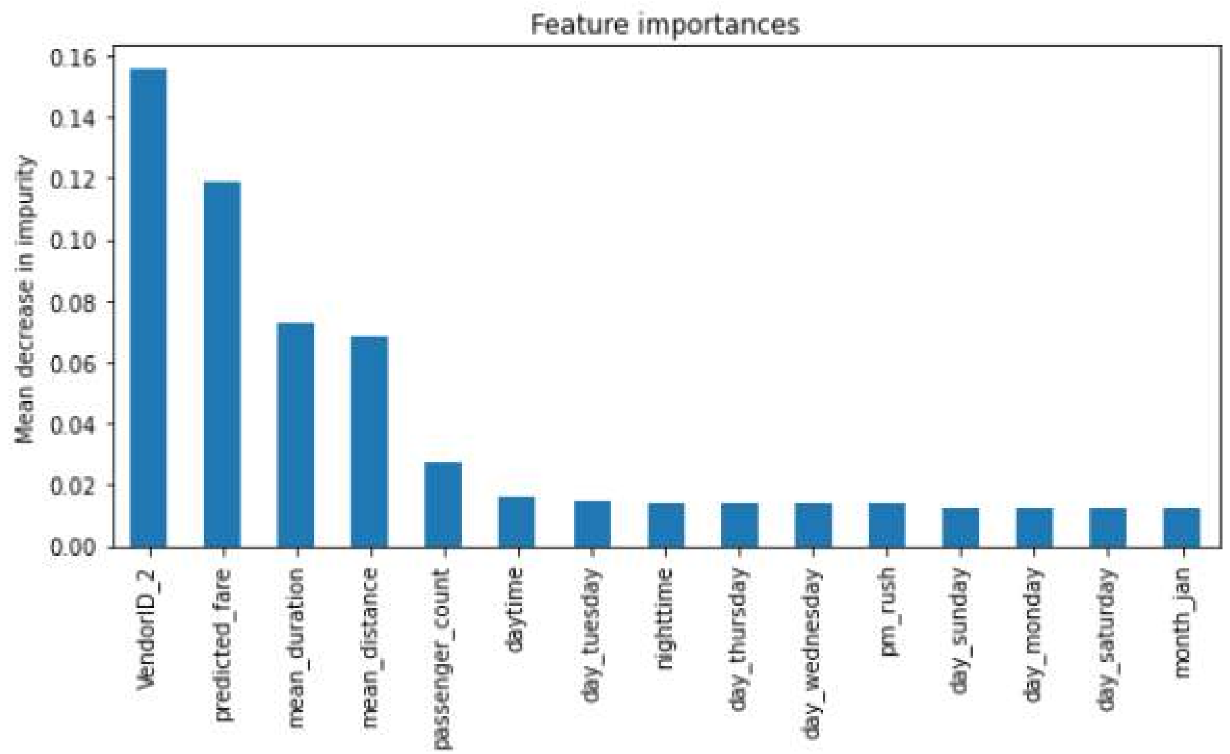
- **Gradient boosting model is the champion**, with **F1 score test 73%**, ~0.03 higher than the random forest.

- The benefit of using multiple models (Random Forest and Gradient Boosting) to predict on the test data is that you can compare models using data that was not used to train/tune hyperparameters. This helps prevent choosing a model just because it fits the training data well.

- The problem with using the final test data to choose a model is that it can bias your decision, making it harder to know how the model will perform on new data. In this case, selecting the final model is like another form of "tuning."



- **The model is nearly twice as likely to predict a false positive** (predicting a generous tip when it's actually low) **than a false negative** (predicting no generous tip when it is actually generous). This indicates that **type I errors are more common**. While it's better for drivers to be pleasantly surprised by a generous tip than disappointed by a low one, the model's overall performance remains acceptable.



- **`VendorID`, `predicted_fare`, `mean_duration`, and `mean_distance` are the most important features.** `VendorID` is the most predictive feature. This seems to indicate that one of the two vendors tends to attract more generous customers.