Activity_ Course 5 Automatidata project lab

July 21, 2023

1 Automatidata project

Course 5 - Regression Analysis: Simplify complex data relationships

The data consulting firm Automatidata has recently hired you as the newest member of their data analytics team. Their newest client, the NYC Taxi and Limousine Commission (New York City TLC), wants the Automatidata team to build a multiple linear regression model to predict taxi fares using existing data that was collected over the course of a year. The team is getting closer to completing the project, having completed an initial plan of action, initial Python coding work, EDA, and A/B testing.

The Automatidata team has reviewed the results of the A/B testing. Now it's time to work on predicting the taxi fare amounts. You've impressed your Automatidata colleagues with your hard work and attention to detail. The data team believes that you are ready to build the regression model and update the client New York City TLC about your progress.

A notebook was structured and prepared to help you in this project. Please complete the following questions.

2 Course 5 End-of-course project: Build a multiple linear regression model

In this activity, you will build a multiple linear regression model. As you've learned, multiple linear regression helps you estimate the linear relationship between one continuous dependent variable and two or more independent variables. For data science professionals, this is a useful skill because it allows you to consider more than one variable against the variable you're measuring against. This opens the door for much more thorough and flexible analysis to be completed.

Completing this activity will help you practice planning out and buidling a multiple linear regression model based on a specific business need. The structure of this activity is designed to emulate the proposals you will likely be assigned in your career as a data professional. Completing this activity will help prepare you for those career moments.

The purpose of this project is to demostrate knowledge of EDA and a multiple linear regression model

The goal is to build a multiple linear regression model and evaluate the model *This activity has three parts:*

Part 1: EDA & Checking Model Assumptions * What are some purposes of EDA before constructing a multiple linear regression model?

Part 2: Model Building and evaluation * What resources do you find yourself using as you complete this stage?

Part 3: Interpreting Model Results

- What key insights emerged from your model(s)?
- What business recommendations do you propose based on the models built?

3 Build a multiple linear regression model

4 PACE stages

Throughout these project notebooks, you'll see references to the problem-solving framework PACE. The following notebook components are labeled with the respective PACE stage: Plan, Analyze, Construct, and Execute.

4.1 PACE: Plan

Consider the questions in your PACE Strategy Document to reflect on the Plan stage.

4.1.1 Task 1. Imports and loading

Import the packages that you've learned are needed for building linear regression models.

[185]:
```
# Imports
# Packages for numerics + dataframes
### YOUR CODE HERE ###
import pandas as pd
import numpy as np

# Packages for visualization
### YOUR CODE HERE ###
import seaborn as sns
import matplotlib.pyplot as plt

# Packages for date conversions for calculating trip durations
### YOUR CODE HERE ###
import datetime

# Packages for OLS, MLR, confusion matrix
### YOUR CODE HERE ###
from statsmodels.formula.api import ols
```

```python
import statsmodels.api as sm
import sklearn.metrics as metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error,r2_score,mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
```

Note: `Pandas` is used to load the NYC TLC dataset. As shown in this cell, the dataset has been automatically loaded in for you. You do not need to download the .csv file, or provide more code, in order to access the dataset and proceed with this lab. Please continue with this activity by completing the following instructions.

[4]:
```python
# Load dataset into dataframe
data=pd.read_csv("2017_Yellow_Taxi_Trip_Data.csv") # index_col parameter␣
↪specified to avoid "Unnamed: 0" column when reading in data from csv
```

4.2 PACE: Analyze

In this stage, consider the following question where applicable to complete your code response:

- What are some purposes of EDA before constructing a multiple linear regression model?

1. Outliers and extreme data values can significantly impact linear regression equations. After visualizing data, make a plan for addressing outliers by dropping rows, substituting extreme data with average data, and/or removing data values greater than 3 standard deviations.
2. EDA activities also include identifying missing data to help the analyst make decisions on their exclusion or inclusion by substituting values with data set means, medians, and other similar methods.
3. It's important to check for things like multicollinearity between predictor variables, as well to understand their distributions, as this will help you decide what statistical inferences can be made from the model and which ones cannot.
4. Additionally, it can be useful to engineer new features by multiplying variables together or taking the difference from one variable to another. For example, in this dataset you can create a `duration` variable by subtracting `tpep_dropoff` from `tpep_pickup time`.

4.2.1 Task 2a. Explore data with EDA

Analyze and discover data, looking for correlations, missing data, outliers, and duplicates.

Start with `.shape` and `.info()`.

[5]:
```python
# Start with `.shape` and `.info()`
### YOUR CODE HERE ###
print("Shape:", data.shape)
data.info()
```

```
Shape: (22699, 18)
<class 'pandas.core.frame.DataFrame'>
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)
memory usage: 3.1+ MB
```

Check for missing data and duplicates using `.isna()` and `.drop_duplicates()`.

[6]:
```python
# Check for missing data and duplicates using .isna() and .drop_duplicates()
### YOUR CODE HERE ###
data.isna().sum()
```

[6]:
```
Unnamed: 0               0
VendorID                 0
tpep_pickup_datetime     0
tpep_dropoff_datetime    0
passenger_count          0
trip_distance            0
RatecodeID               0
store_and_fwd_flag       0
PULocationID             0
DOLocationID             0
payment_type             0
fare_amount              0
extra                    0
mta_tax                  0
tip_amount               0
```

```
tolls_amount             0
improvement_surcharge    0
total_amount             0
dtype: int64
```

Use .describe().

[7]:
```
# Use .describe()
### YOUR CODE HERE ###
data.describe()
```

[7]:
```
         Unnamed: 0      VendorID  passenger_count  trip_distance  \
count  2.269900e+04  22699.000000     22699.000000   22699.000000
mean   5.675849e+07      1.556236         1.642319       2.913313
std    3.274493e+07      0.496838         1.285231       3.653171
min    1.212700e+04      1.000000         0.000000       0.000000
25%    2.852056e+07      1.000000         1.000000       0.990000
50%    5.673150e+07      2.000000         1.000000       1.610000
75%    8.537452e+07      2.000000         2.000000       3.060000
max    1.134863e+08      2.000000         6.000000      33.960000

         RatecodeID  PULocationID  DOLocationID  payment_type   fare_amount  \
count  22699.000000  22699.000000  22699.000000  22699.000000  22699.000000
mean       1.043394    162.412353    161.527997      1.336887     13.026629
std        0.708391     66.633373     70.139691      0.496211     13.243791
min        1.000000      1.000000      1.000000      1.000000   -120.000000
25%        1.000000    114.000000    112.000000      1.000000      6.500000
50%        1.000000    162.000000    162.000000      1.000000      9.500000
75%        1.000000    233.000000    233.000000      2.000000     14.500000
max       99.000000    265.000000    265.000000      4.000000    999.990000

              extra       mta_tax    tip_amount  tolls_amount  \
count  22699.000000  22699.000000  22699.000000  22699.000000
mean       0.333275      0.497445      1.835781      0.312542
std        0.463097      0.039465      2.800626      1.399212
min       -1.000000     -0.500000      0.000000      0.000000
25%        0.000000      0.500000      0.000000      0.000000
50%        0.000000      0.500000      1.350000      0.000000
75%        0.500000      0.500000      2.450000      0.000000
max        4.500000      0.500000    200.000000     19.100000

       improvement_surcharge  total_amount
count           22699.000000  22699.000000
mean                0.299551     16.310502
std                 0.015673     16.097295
min                -0.300000   -120.300000
25%                 0.300000      8.750000
```

```
50%                  0.300000     11.800000
75%                  0.300000     17.800000
max                  0.300000   1200.290000
```

Check for missing data and duplicates using `.isna()` and `.drop_duplicates()`.

[8]:
```python
# Check for missing data and duplicates using .isna() and .drop_duplicates()
### YOUR CODE HERE ###

# Make a copy of dataframe
df = data.copy()

# Check for duplicates
print('Shape of dataframe:', df.shape)
print('Shape of dataframe with duplicates dropped:', df.drop_duplicates().shape)

# Check for missing values in dataframe
print('Total count of missing values:', df.isna().sum().sum())

# Display missing values per column in dataframe
print('Missing values per column:')
df.isna().sum()
```

```
Shape of dataframe: (22699, 18)
Shape of dataframe with duplicates dropped: (22699, 18)
Total count of missing values: 0
Missing values per column:
```

[8]:
```
Unnamed: 0               0
VendorID                 0
tpep_pickup_datetime     0
tpep_dropoff_datetime    0
passenger_count          0
trip_distance            0
RatecodeID               0
store_and_fwd_flag       0
PULocationID             0
DOLocationID             0
payment_type             0
fare_amount              0
extra                    0
mta_tax                  0
tip_amount               0
tolls_amount             0
improvement_surcharge    0
total_amount             0
dtype: int64
```

[9]:

```python
# Display descriptive stats about the data
df.describe()
```

[9]:

```
         Unnamed: 0      VendorID  passenger_count  trip_distance  \
count  2.269900e+04  22699.000000     22699.000000   22699.000000
mean   5.675849e+07      1.556236         1.642319       2.913313
std    3.274493e+07      0.496838         1.285231       3.653171
min    1.212700e+04      1.000000         0.000000       0.000000
25%    2.852056e+07      1.000000         1.000000       0.990000
50%    5.673150e+07      2.000000         1.000000       1.610000
75%    8.537452e+07      2.000000         2.000000       3.060000
max    1.134863e+08      2.000000         6.000000      33.960000

         RatecodeID  PULocationID  DOLocationID  payment_type   fare_amount  \
count  22699.000000  22699.000000  22699.000000  22699.000000  22699.000000
mean       1.043394    162.412353    161.527997      1.336887     13.026629
std        0.708391     66.633373     70.139691      0.496211     13.243791
min        1.000000      1.000000      1.000000      1.000000   -120.000000
25%        1.000000    114.000000    112.000000      1.000000      6.500000
50%        1.000000    162.000000    162.000000      1.000000      9.500000
75%        1.000000    233.000000    233.000000      2.000000     14.500000
max       99.000000    265.000000    265.000000      4.000000    999.990000

              extra       mta_tax    tip_amount  tolls_amount  \
count  22699.000000  22699.000000  22699.000000  22699.000000
mean       0.333275      0.497445      1.835781      0.312542
std        0.463097      0.039465      2.800626      1.399212
min       -1.000000     -0.500000      0.000000      0.000000
25%        0.000000      0.500000      0.000000      0.000000
50%        0.000000      0.500000      1.350000      0.000000
75%        0.500000      0.500000      2.450000      0.000000
max        4.500000      0.500000    200.000000     19.100000

       improvement_surcharge  total_amount
count           22699.000000  22699.000000
mean                0.299551     16.310502
std                 0.015673     16.097295
min                -0.300000   -120.300000
25%                 0.300000      8.750000
50%                 0.300000     11.800000
75%                 0.300000     17.800000
max                 0.300000   1200.290000
```

4.2.2 Task 2b. Convert pickup & dropoff columns to datetime

[10]:
```python
# Check the format of the data
### YOUR CODE HERE ###
print(df['tpep_pickup_datetime'][0])
print(df['tpep_dropoff_datetime'][0])
df.info()
```

```
03/25/2017 8:55:43 AM
03/25/2017 9:09:47 AM
<class 'pandas.core.frame.DataFrame'>
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)
memory usage: 3.1+ MB
```

[11]:
```python
# Convert datetime columns to datetime
### YOUR CODE HERE ###

df['tpep_pickup_datetime'] = pd.to_datetime(df.tpep_pickup_datetime, format='%m/
↪%d/%Y %I:%M:%S %p')
df['tpep_dropoff_datetime'] = pd.to_datetime(df.tpep_dropoff_datetime,␣
↪format='%m/%d/%Y %I:%M:%S %p')
df.head()
```

[11]:
```
   Unnamed: 0  VendorID tpep_pickup_datetime tpep_dropoff_datetime  \
0    24870114         2  2017-03-25 08:55:43   2017-03-25 09:09:47
```

```
1   35634249         1  2017-04-11 14:53:28   2017-04-11 15:19:58
2  106203690         1  2017-12-15 07:26:56   2017-12-15 07:34:08
3   38942136         2  2017-05-07 13:17:59   2017-05-07 13:48:14
4   30841670         2  2017-04-15 23:32:20   2017-04-15 23:49:03

   passenger_count  trip_distance  RatecodeID store_and_fwd_flag  \
0                6           3.34           1                  N
1                1           1.80           1                  N
2                1           1.00           1                  N
3                1           3.70           1                  N
4                1           4.37           1                  N

   PULocationID  DOLocationID  payment_type  fare_amount  extra  mta_tax  \
0           100           231             1         13.0    0.0      0.5
1           186            43             1         16.0    0.0      0.5
2           262           236             1          6.5    0.0      0.5
3           188            97             1         20.5    0.0      0.5
4             4           112             2         16.5    0.5      0.5

   tip_amount  tolls_amount  improvement_surcharge  total_amount
0        2.76           0.0                    0.3         16.56
1        4.00           0.0                    0.3         20.80
2        1.45           0.0                    0.3          8.75
3        6.39           0.0                    0.3         27.69
4        0.00           0.0                    0.3         17.80
```

4.2.3 Task 2c. Create duration column

Create a new column called `duration` that represents the total number of minutes that each taxi ride took.

[12]:
```python
# Create `duration` column
### YOUR CODE HERE ###
df['duration'] = (df['tpep_dropoff_datetime'] - df['tpep_pickup_datetime']).dt.
 ↪total_seconds() / 60
df.head()
```

[12]:
```
   Unnamed: 0  VendorID tpep_pickup_datetime tpep_dropoff_datetime  \
0    24870114         2  2017-03-25 08:55:43   2017-03-25 09:09:47
1    35634249         1  2017-04-11 14:53:28   2017-04-11 15:19:58
2   106203690         1  2017-12-15 07:26:56   2017-12-15 07:34:08
3    38942136         2  2017-05-07 13:17:59   2017-05-07 13:48:14
4    30841670         2  2017-04-15 23:32:20   2017-04-15 23:49:03

   passenger_count  trip_distance  RatecodeID store_and_fwd_flag  \
0                6           3.34           1                  N
```

```
1                1         1.80          1                 N
2                1         1.00          1                 N
3                1         3.70          1                 N
4                1         4.37          1                 N

   PULocationID  DOLocationID  payment_type  fare_amount  extra  mta_tax  \
0           100           231             1         13.0    0.0      0.5
1           186            43             1         16.0    0.0      0.5
2           262           236             1          6.5    0.0      0.5
3           188            97             1         20.5    0.0      0.5
4             4           112             2         16.5    0.5      0.5

   tip_amount  tolls_amount  improvement_surcharge  total_amount   duration
0        2.76           0.0                    0.3         16.56  14.066667
1        4.00           0.0                    0.3         20.80  26.500000
2        1.45           0.0                    0.3          8.75   7.200000
3        6.39           0.0                    0.3         27.69  30.250000
4        0.00           0.0                    0.3         17.80  16.716667
```

4.2.4 Outliers

Call `df.info()` to inspect the columns and decide which ones to check for outliers.

[13]:
```python
### YOUR CODE HERE ###
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
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)
memory usage: 3.3+ MB
```

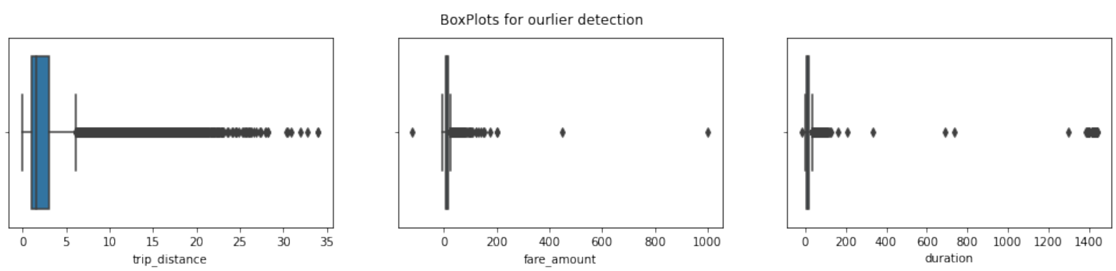
Keeping in mind that many of the features will not be used to fit your model, the most important columns to check for outliers are likely to be: * `trip_distance` * `fare_amount` * `duration`

4.2.5 Task 2d. Box plots

Plot a box plot for each feature: `trip_distance`, `fare_amount`, `duration`.

[14]:
```python
### YOUR CODE HERE ###

fig, axes = plt.subplots(1, 3, figsize=(17,3))
fig.suptitle("BoxPlots for ourlier detection")

# Boxplots
sns.boxplot(ax = axes[0],x='trip_distance', data=df)
sns.boxplot(ax=axes[1], x='fare_amount', data=df)
sns.boxplot(ax=axes[2], x='duration', data=df)
```

[14]: `<matplotlib.axes._subplots.AxesSubplot at 0x7efcf536c110>`



Questions: 1. Which variable(s) contains outliers?

2. Are the values in the `trip_distance` column unbelievable?

3. What about the lower end? Do distances, fares, and durations of 0 (or negative values) make sense?

1. All three variables contain outliers. Some are extreme, but others not so much.

2. It's 30 miles from the southern tip of Staten Island to the northern end of Manhattan and that's in a straight line. With this knowledge and the distribution of the values in this column, it's reasonable to leave these values alone and not alter them. However, the values for `fare_amount` and `duration` definitely seem to have problematic outliers on the higher end.

3. Probably not for the latter two, but for `trip_distance` it might be okay.

4.2.6 Task 2e. Imputations

`trip_distance` outliers You know from the summary statistics that there are trip distances of 0. Are these reflective of erroneous data, or are they very short trips that get rounded down?

To check, sort the column values, eliminate duplicates, and inspect the least 10 values. Are they rounded values or precise values?

[15]:
```
# Are trip distances of 0 bad data or very short trips rounded down?
### YOUR CODE HERE ###
df['trip_distance'].drop_duplicates().sort_values()[:10]
```

[15]:
```
128      0.00
2985     0.01
323      0.02
3158     0.03
1510     0.04
10146    0.05
4423     0.06
922      0.07
4623     0.08
22035    0.09
Name: trip_distance, dtype: float64
```

The distances are captured with a high degree of precision. However, it might be possible for trips to have distances of zero if a passenger summoned a taxi and then changed their mind. Besides, are there enough zero values in the data to pose a problem?

Calculate the count of rides where the `trip_distance` is zero.

[16]:
```
### YOUR CODE HERE ###
len(df[df['trip_distance']==0])
```

[16]:
```
148
```

Out of 22699 entries , almost there are 148 trips with 0 distance covered. We will leave them regardless of outliers, because since it's very low in numbers and are highly possible to have rides with 0 distance

`fare_amount` outliers

[17]:
```
### YOUR CODE HERE ###
df['fare_amount'].describe()
```

[17]:
```
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
```

Question: What do you notice about the values in the `fare_amount` column?

- The minimum value is -123, which indicates that there are negative values might might be problematic.
- The Maxium fare is nearly 999, but the majority of the data lies between 6 and 14 amount. These highly trips amount may also be problematic.
- The standard formula of `Q3 + (1.5 * IQR)` yields $26.50. That doesn't seem appropriate for the maximum fare cap. In this case, we'll use a factor of `6`, which results in a cap of $62.50.

Impute values less than $0 with `0`.

[18]:
```
# Impute values less than $0 with 0
### YOUR CODE HERE ###
df.loc[df['fare_amount'] < 0, 'fare_amount'] = 0
df['fare_amount'].min()
```

[18]:
```
0.0
```

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

[19]:
```
### YOUR CODE HERE ###
def outliers_imputer(column_list, iqr_factor):
    '''
    Impute upper-limit values in specified columns based on their interquartile␣
 ↪range.

    Arguments:
        column_list: A list of columns to iterate over
        iqr_factor: A number representing x in the formula:
                    Q3 + (x * IQR). Used to determine maximum threshold,
                    beyond which a point is considered an outlier.

    The IQR is computed for each column in column_list and values exceeding
    the upper threshold for each column are imputed with the upper threshold␣
 ↪value.
    '''
  ### YOUR CODE HERE ###
    for col in column_list:
        # Reassign minimum to zero
        ### YOUR CODE HERE ###
```

```
        df.loc[df[col] < 0, col] = 0

        # Calculate upper threshold
     ### YOUR CODE HERE ###
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)
        iqr = q3 - q1
        upper_limit = q3 + (iqr_factor * iqr)

        # Reassign values > threshold to threshold
      ### YOUR CODE HERE ###
        df.loc[df[col] > upper_limit, col] = upper_limit
        print(df[col].describe())
```

[20]:
```
outliers_imputer(['fare_amount'], 6)
```

```
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
```

duration outliers

[21]:
```
# Call .describe() for duration outliers
### YOUR CODE HERE ###
df.duration.describe()
```

[21]:
```
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
```

The `duration` column has problematic values at both the lower and upper extremities.

- **Low values:** There should be no values that represent negative time. Impute all negative durations with `0`.
- **High values:** Impute high values the same way you imputed the high-end outliers for fares: `Q3 + (6 * IQR)`.

[22]:
```python
# Impute a 0 for any negative values
### YOUR CODE HERE ###
```

[23]:
```python
# Impute the high outliers
### YOUR CODE HERE ###
outliers_imputer(['duration'], 6)
```

```
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
```

4.2.7 Task 3a. Feature engineering

Create `mean_distance` column When deployed, 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. However, you can use the statistics of trips you *do* know to generalize about ones you do not know.

In this step, create a column called `mean_distance` that captures the mean distance for each group of trips that share pickup and dropoff points.

For example, if your data were:

|Trip|Start|End|Distance| |–: |:—:|:-:| | | 1 | A | B | 1 | | 2 | C | D | 2 | | 3 | A | B |1.5 | | 4 | D | C | 3 |

The results should be:

```
A -> B: 1.25 miles
C -> D: 2 miles
D -> C: 3 miles
```

Notice that C -> D is not the same as D -> C. All trips that share a unique pair of start and end points get grouped and averaged.

Then, a new column `mean_distance` will be added where the value at each row is the average for all trips with those pickup and dropoff locations:

Trip	Start	End	Distance	mean_distance
1	A	B	1	1.25
2	C	D	2	2
3	A	B	1.5	1.25
4	D	C	3	3

Begin by creating a helper column called `pickup_dropoff`, which contains the unique combination of pickup and dropoff location IDs for each row.

One way to do this is to convert the pickup and dropoff location IDs to strings and join them, separated by a space. The space is to ensure that, for example, a trip with pickup/dropoff points of 12 & 151 gets encoded differently than a trip with points 121 & 51.

So, the new column would look like this:

Trip	Start	End	pickup_dropoff
1	A	B	'A B'
2	C	D	'C D'
3	A	B	'A B'
4	D	C	'D C'

[56]:
```
# Create `pickup_dropoff` column
### YOUR CODE HERE ###
df['pickup_dropoff'] = df['PULocationID'].astype('string') + " " +␣
↪df['DOLocationID'].astype('string')
df['pickup_dropoff'].head()
```

[56]:
```
0    100 231
1     186 43
2    262 236
3     188 97
4      4 112
Name: pickup_dropoff, dtype: string
```

Now, use a `groupby()` statement to group each row by the new `pickup_dropoff` column, compute the mean, and capture the values only in the `trip_distance` column. Assign the results to a variable named `grouped`.

[57]:
```
### YOUR CODE HERE ###
grouped = df.groupby('pickup_dropoff')[['trip_distance']].mean()
grouped
```

[57]:
```
                trip_distance
pickup_dropoff
1 1                  2.433333
10 148              15.700000
100 1               16.890000
100 100              0.253333
100 107              1.180000
…                       …
97 65                0.500000
97 66                1.400000
97 80                3.840000
```

```
97 90                   4.420000
97 97                   1.006667

[4172 rows x 1 columns]
```

`grouped` is an object of the `DataFrame` class.

1. Convert it to a dictionary using the `to_dict()` method. Assign the results to a variable called `grouped_dict`. This will result in a dictionary with a key of `trip_distance` whose values are another dictionary. The inner dictionary's keys are pickup/dropoff points and its values are mean distances. This is the information you want.

```
Example:
grouped_dict = {'trip_distance': {'A B': 1.25, 'C D': 2, 'D C': 3}
```

2. Reassign the `grouped_dict` dictionary so it contains only the inner dictionary. In other words, get rid of `trip_distance` as a key, so:

```
Example:
grouped_dict = {'A B': 1.25, 'C D': 2, 'D C': 3}
```

[59]:
```
# 1. Convert `grouped` to a dictionary
### YOUR CODE HERE ###
grouped_dict = grouped.to_dict()

# 2. Reassign to only contain the inner dictionary
### YOUR CODE HERE ###
grouped_dict = grouped_dict['trip_distance']
```

1. Create a `mean_distance` column that is a copy of the `pickup_dropoff` helper column.
2. Use the `map()` method on the `mean_distance` series. Pass `grouped_dict` as its argument. Reassign the result back to the `mean_distance` series. When you pass a dictionary to the `Series.map()` method, it will replace the data in the series where that data matches the dictionary's keys. The values that get imputed are the values of the dictionary.

```
Example:
df['mean_distance']
```

mean_distance
'A B'
'C D'
'A B'
'D C'
'E F'

```
grouped_dict = {'A B': 1.25, 'C D': 2, 'D C': 3}
df['mean_distance`] = df['mean_distance'].map(grouped_dict)
df['mean_distance']
```

mean_distance
1.25
2
1.25
3
NaN

When used this way, the `map()` `Series` method is very similar to `replace()`, however, note that `map()` will impute `NaN` for any values in the series that do not have a corresponding key in the mapping dictionary, so be careful.

[63]:
```
# 1. Create a mean_distance column that is a copy of the pickup_dropoff helper␣
↪column
### YOUR CODE HERE ###
df['mean_distance'] = pd.Series()

# 2. Map `grouped_dict` to the `mean_distance` column
### YOUR CODE HERE ###
df['mean_distance'] = df['pickup_dropoff'].map(grouped_dict)

# Confirm that it worked
### YOUR CODE HERE ###
df[(df['PULocationID'] == 100) & (df['DOLocationID'] == 231)][['mean_distance']]
```

[63]:
```
       mean_distance
0           3.521667
4909        3.521667
16636       3.521667
18134       3.521667
19761       3.521667
20581       3.521667
```

Create `mean_duration` column Repeat the process used to create the `mean_distance` column to create a `mean_duration` column.

[65]:
```
### YOUR CODE HERE ###

# Create a dictionary where keys are unique pickup_dropoffs and values are
# mean trip duration for all trips with those pickup_dropoff combos
### YOUR CODE HERE ###
grouped = df.groupby('pickup_dropoff')[['duration']].mean()
grouped_dict = grouped.to_dict()
grouped_dict = grouped_dict['duration']
df['mean_duration'] = pd.Series()
df['mean_duration'] = df['pickup_dropoff'].map(grouped_dict)
```

```
# Confirm that it worked
### YOUR CODE HERE ###
df[(df['PULocationID'] == 100) & (df['DOLocationID'] == 231)][['mean_duration']]
```

[65]:

	mean_duration
0	22.847222
4909	22.847222
16636	22.847222
18134	22.847222
19761	22.847222
20581	22.847222

Create `day` and `month` columns Create two new columns, `day` (name of day) and `month` (name of month) by extracting the relevant information from the `tpep_pickup_datetime` column.

[68]:

```
# Create 'day' col
### YOUR CODE HERE ###

df['day'] = df['tpep_pickup_datetime'].dt.day_name().str.lower()
# Create 'month' col
### YOUR CODE HERE ###
df['month'] = df['tpep_pickup_datetime'].dt.month_name().str.lower()
df[['day', 'month']].head()
```

[68]:

	day	month
0	saturday	march
1	tuesday	april
2	friday	december
3	sunday	may
4	saturday	april

Create `rush_hour` column Define rush hour as: * Any weekday (not Saturday or Sunday) AND * Either from 06:00–10:00 or from 16:00–20:00

Create a binary `rush_hour` column that contains a 1 if the ride was during rush hour and a 0 if it was not.

[77]:

```
# Create 'rush_hour' col
### YOUR CODE HERE ###
df['rush_hour'] = df['tpep_pickup_datetime'].dt.hour

# If day is Saturday or Sunday, impute 0 in `rush_hour` column
### YOUR CODE HERE ###
df.loc[df['day'].isin(['saturday', 'sunday']), 'rush_hour'] = 0
```

[78]:
```python
### YOUR CODE HERE ###
def rush_hourizer(hour):
    if 6 <= hour < 10 or 16 <= hour < 20:
        return 1
    return 0
```

[79]:
```python
# Apply the `rush_hourizer()` function to the new column
### YOUR CODE HERE ###
df['rush_hour'] = df['rush_hour'].apply(rush_hourizer)
df.head()
```

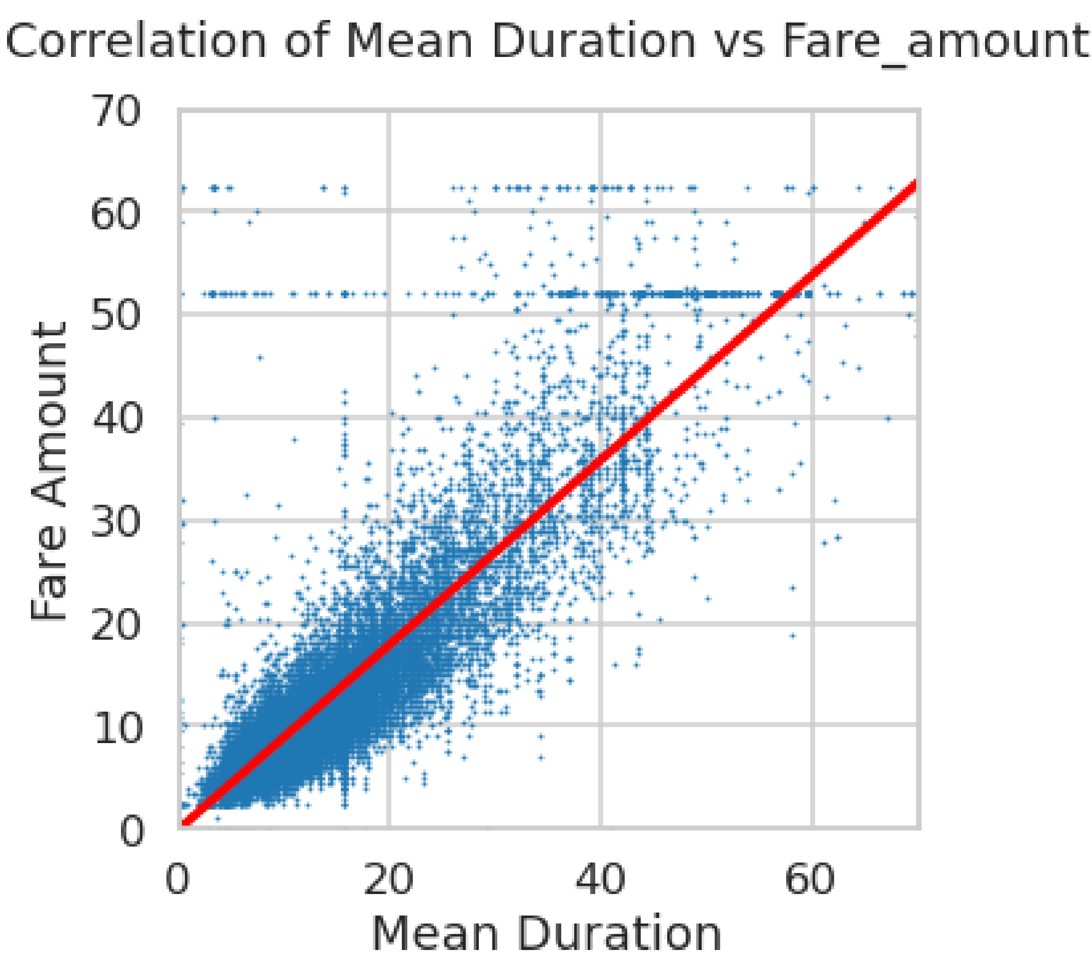
[79]:
```
   Unnamed: 0  VendorID tpep_pickup_datetime tpep_dropoff_datetime  \
0    24870114         2  2017-03-25 08:55:43   2017-03-25 09:09:47
1    35634249         1  2017-04-11 14:53:28   2017-04-11 15:19:58
2   106203690         1  2017-12-15 07:26:56   2017-12-15 07:34:08
3    38942136         2  2017-05-07 13:17:59   2017-05-07 13:48:14
4    30841670         2  2017-04-15 23:32:20   2017-04-15 23:49:03

   passenger_count  trip_distance  RatecodeID store_and_fwd_flag  \
0                6           3.34           1                  N
1                1           1.80           1                  N
2                1           1.00           1                  N
3                1           3.70           1                  N
4                1           4.37           1                  N

   PULocationID  DOLocationID  …  improvement_surcharge  total_amount  \
0           100           231  …                    0.3         16.56
1           186            43  …                    0.3         20.80
2           262           236  …                    0.3          8.75
3           188            97  …                    0.3         27.69
4             4           112  …                    0.3         17.80

    duration  pickup_dropoff  mean_distance  mean_duration       day  \
0  14.066667         100 231       3.521667      22.847222  saturday
1  26.500000          186 43       3.108889      24.470370   tuesday
2   7.200000         262 236       0.881429       7.250000    friday
3  30.250000          188 97       3.700000      30.250000    sunday
4  16.716667           4 112       4.435000      14.616667  saturday

      month  roush_hour rush_hour
0     march           8         0
1     april          14         0
2  december           7         1
3       may          13         0
4     april          23         0

[5 rows x 26 columns]
```

4.2.8 Task 4. Scatter plot

Create a scatterplot to visualize the relationship between `mean_duration` and `fare_amount`.

[118]:
```
# Create a scatterplot to visualize the relationship between variables of␣
 ↪interest
### YOUR CODE HERE ###
plt.figure(figsize=(5, 5))
sns.set_context('talk')
sns.set_style('whitegrid')
sns.regplot(x='mean_duration', y='fare_amount', data=df, line_kws={'color' :␣
 ↪'red'}, scatter_kws={'s' : 0.8})
plt.xlabel("Mean Duration")
plt.ylabel("Fare Amount")
plt.title("Correlation of Mean Duration vs Fare_amount", y=1.05)
plt.ylim(0, 70)
plt.xlim(0, 70)
```

[118]:
```
(0.0, 70.0)
```



The `mean_duration` variable correlates with the target variable. But what are the horizontal lines around fare amounts of 52 dollars and 63 dollars? What are the values and how many are there?

You know what one of the lines represents. 62 dollars and 50 cents is the maximum that was imputed for outliers, so all former outliers will now have fare amounts of $62.50. What is the other line?

Check the value of the rides in the second horizontal line in the scatter plot.

[120]:
```
### YOUR CODE HERE ###
df[df['fare_amount'] > 50]['fare_amount'].value_counts().head(3)
```

[120]:
```
52.0    514
62.5     84
59.0      9
Name: fare_amount, dtype: int64
```

Examine the first 30 of these trips.

[124]:
```
# Set pandas to display all columns
### YOUR CODE HERE ###
pd.set_option('display.max_columns', None)
df[df['fare_amount'] == 52].head(30)
```

[124]:
```
     Unnamed: 0  VendorID tpep_pickup_datetime tpep_dropoff_datetime  \
11     18600059         2  2017-03-05 19:15:30   2017-03-05 19:52:18
110    47959795         1  2017-06-03 14:24:57   2017-06-03 15:31:48
161    95729204         2  2017-11-11 20:16:16   2017-11-11 20:17:14
247   103404868         2  2017-12-06 23:37:08   2017-12-07 00:06:19
379    80479432         2  2017-09-24 23:45:45   2017-09-25 00:15:14
388    16226157         1  2017-02-28 18:30:05   2017-02-28 19:09:55
406    55253442         2  2017-06-05 12:51:58   2017-06-05 13:07:35
449    65900029         2  2017-08-03 22:47:14   2017-08-03 23:32:41
468    80904240         2  2017-09-26 13:48:26   2017-09-26 14:31:17
520    33706214         2  2017-04-23 21:34:48   2017-04-23 22:46:23
569    99259872         2  2017-11-22 21:31:32   2017-11-22 22:00:25
572    61050418         2  2017-07-18 13:29:06   2017-07-18 13:29:19
586    54444647         2  2017-06-26 13:39:12   2017-06-26 14:34:54
692    94424289         2  2017-11-07 22:15:00   2017-11-07 22:45:32
717   103094220         1  2017-12-06 05:19:50   2017-12-06 05:53:52
719    66115834         1  2017-08-04 17:53:34   2017-08-04 18:50:56
782    55934137         2  2017-06-09 09:31:25   2017-06-09 10:24:10
816    13731926         2  2017-02-21 06:11:03   2017-02-21 06:59:39
818    52277743         2  2017-06-20 08:15:18   2017-06-20 10:24:37
835     2684305         2  2017-01-10 22:29:47   2017-01-10 23:06:46
840    90860814         2  2017-10-27 21:50:00   2017-10-27 22:35:04
```

861	106575186	1	2017-12-16 06:39:59	2017-12-16 07:07:59
881	110495611	2	2017-12-30 05:25:29	2017-12-30 06:01:29
958	87017503	1	2017-10-15 22:39:12	2017-10-15 23:14:22
970	12762608	2	2017-02-17 20:39:42	2017-02-17 21:13:29
984	71264442	1	2017-08-23 18:23:26	2017-08-23 19:18:29
1082	11006300	2	2017-02-07 17:20:19	2017-02-07 17:34:41
1097	68882036	2	2017-08-14 23:01:15	2017-08-14 23:03:35
1110	74720333	1	2017-09-06 10:46:17	2017-09-06 11:44:41
1179	51937907	2	2017-06-19 06:23:13	2017-06-19 07:03:53

	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag \
11	2	18.90	2	N
110	1	18.00	2	N
161	1	0.23	2	N
247	1	18.93	2	N
379	1	17.99	2	N
388	1	18.40	2	N
406	1	4.73	2	N
449	2	18.21	2	N
468	1	17.27	2	N
520	6	18.34	2	N
569	1	18.65	2	N
572	1	0.00	2	N
586	1	17.76	2	N
692	2	16.97	2	N
717	1	20.80	2	N
719	1	21.60	2	N
782	2	18.81	2	N
816	5	16.94	2	N
818	1	17.77	2	N
835	1	18.57	2	N
840	1	22.43	2	N
861	2	17.80	2	N
881	6	18.23	2	N
958	1	21.80	2	N
970	1	19.57	2	N
984	1	16.70	2	N
1082	1	1.09	2	N
1097	5	2.12	2	N
1110	1	19.10	2	N
1179	6	19.77	2	N

	PULocationID	DOLocationID	payment_type	fare_amount	extra	mta_tax \
11	236	132	1	52.0	0.0	0.5
110	132	163	1	52.0	0.0	0.5
161	132	132	2	52.0	0.0	0.5
247	132	79	2	52.0	0.0	0.5

379	132	234	1	52.0	0.0	0.5
388	132	48	2	52.0	4.5	0.5
406	228	88	2	52.0	0.0	0.5
449	132	48	2	52.0	0.0	0.5
468	186	132	2	52.0	0.0	0.5
520	132	148	1	52.0	0.0	0.5
569	132	144	1	52.0	0.0	0.5
572	230	161	1	52.0	0.0	0.5
586	211	132	1	52.0	0.0	0.5
692	132	170	1	52.0	0.0	0.5
717	132	239	1	52.0	0.0	0.5
719	264	264	1	52.0	4.5	0.5
782	163	132	1	52.0	0.0	0.5
816	132	170	1	52.0	0.0	0.5
818	132	246	1	52.0	0.0	0.5
835	132	48	1	52.0	0.0	0.5
840	132	163	2	52.0	0.0	0.5
861	75	132	1	52.0	0.0	0.5
881	68	132	2	52.0	0.0	0.5
958	132	261	2	52.0	0.0	0.5
970	132	140	1	52.0	0.0	0.5
984	132	230	1	52.0	4.5	0.5
1082	170	48	2	52.0	4.5	0.5
1097	265	265	2	52.0	0.0	0.5
1110	239	132	1	52.0	0.0	0.5
1179	238	132	1	52.0	0.0	0.5

	tip_amount	tolls_amount	improvement_surcharge	total_amount	\
11	14.58	5.54	0.3	72.92	
110	0.00	0.00	0.3	52.80	
161	0.00	0.00	0.3	52.80	
247	0.00	0.00	0.3	52.80	
379	14.64	5.76	0.3	73.20	
388	0.00	5.54	0.3	62.84	
406	0.00	5.76	0.3	58.56	
449	0.00	5.76	0.3	58.56	
468	0.00	5.76	0.3	58.56	
520	5.00	0.00	0.3	57.80	
569	10.56	0.00	0.3	63.36	
572	11.71	5.76	0.3	70.27	
586	11.71	5.76	0.3	70.27	
692	11.71	5.76	0.3	70.27	
717	5.85	5.76	0.3	64.41	
719	12.60	5.76	0.3	75.66	
782	13.20	0.00	0.3	66.00	
816	2.00	5.54	0.3	60.34	
818	11.71	5.76	0.3	70.27	

835	13.20	0.00	0.3	66.00
840	0.00	5.76	0.3	58.56
861	6.00	5.76	0.3	64.56
881	0.00	0.00	0.3	52.80
958	0.00	0.00	0.3	52.80
970	11.67	5.54	0.3	70.01
984	42.29	0.00	0.3	99.59
1082	0.00	5.54	0.3	62.84
1097	0.00	0.00	0.3	52.80
1110	15.80	0.00	0.3	68.60
1179	17.57	5.76	0.3	76.13

	duration	pickup_dropoff	mean_distance	mean_duration	day \
11	36.800000	236 132	19.211667	40.500000	sunday
110	66.850000	132 163	19.229000	52.941667	saturday
161	0.966667	132 132	2.255862	3.021839	saturday
247	29.183333	132 79	19.431667	47.275000	wednesday
379	29.483333	132 234	17.654000	49.833333	sunday
388	39.833333	132 48	18.761905	58.246032	tuesday
406	15.616667	228 88	4.730000	15.616667	monday
449	45.450000	132 48	18.761905	58.246032	thursday
468	42.850000	186 132	17.096000	42.920000	tuesday
520	71.583333	132 148	17.994286	46.340476	sunday
569	28.883333	132 144	18.537500	37.000000	wednesday
572	0.216667	230 161	0.685484	7.965591	tuesday
586	55.700000	211 132	16.580000	61.691667	monday
692	30.533333	132 170	17.203000	37.113333	tuesday
717	34.033333	132 239	20.901250	44.862500	wednesday
719	57.366667	264 264	3.191516	15.618773	friday
782	52.750000	163 132	17.275833	52.338889	friday
816	48.600000	132 170	17.203000	37.113333	tuesday
818	88.783333	132 246	18.515000	66.316667	tuesday
835	36.983333	132 48	18.761905	58.246032	tuesday
840	45.066667	132 163	19.229000	52.941667	friday
861	28.000000	75 132	18.442500	36.204167	saturday
881	36.000000	68 132	18.785000	58.041667	saturday
958	35.166667	132 261	22.115000	51.493750	sunday
970	33.783333	132 140	19.293333	36.791667	friday
984	55.050000	132 230	18.571200	59.598000	wednesday
1082	14.366667	170 48	1.265789	14.135965	tuesday
1097	2.333333	265 265	0.753077	3.411538	monday
1110	58.400000	239 132	19.795000	50.562500	wednesday
1179	40.666667	238 132	19.470000	53.861111	monday

	month	roush_hour	rush_hour
11	march	19	0
110	june	14	0

161	november	20	0
247	december	23	0
379	september	23	0
388	february	18	1
406	june	12	0
449	august	22	0
468	september	13	0
520	april	21	0
569	november	21	0
572	july	13	0
586	june	13	0
692	november	22	0
717	december	5	0
719	august	17	1
782	june	9	1
816	february	6	1
818	june	8	1
835	january	22	0
840	october	21	0
861	december	6	0
881	december	5	0
958	october	22	0
970	february	20	0
984	august	18	1
1082	february	17	1
1097	august	23	0
1110	september	10	0
1179	june	6	1

Question: What do you notice about the first 30 trips?

==> It seems that almost all of the trips in the first 30 rows where the fare amount was $52 either begin or end at location 132, and all of them have a `RatecodeID` of 2.

There is no readily apparent reason why PULocation 132 should have so many fares of 52 dollars. They seem to occur on all different days, at different times, with both vendors, in all months. However, there are many toll amounts of $5.76 and $5.54. This would seem to indicate that location 132 is in an area that frequently requires tolls to get to and from. It's likely this is an airport.

The data dictionary says that `RatecodeID` of 2 indicates trips for JFK, which is John F. Kennedy International Airport. A quick Google search for 'new york city taxi flat rate'$52'' indicates that in 2017 (the year that this data was collected) there was indeed a flat fare for taxi trips between JFK airport (in Queens) and Manhattan.

Because `RatecodeID` is known from the data dictionary, the values for this rate code can be imputed back into the data after the model makes its predictions. This way you know that those data points will always be correct.

4.2.9 Task 5. Isolate modeling variables

Drop features that are redundant, irrelevant, or that will not be available in a deployed environment.

[125]:
```
### YOUR CODE HERE ###
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 26 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  string
 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  roush_hour             22699 non-null  int64
 25  rush_hour              22699 non-null  int64
dtypes: datetime64[ns](2), float64(11), int64(9), object(3), string(1)
memory usage: 4.5+ MB
```

[127]:
```
### YOUR CODE HERE ###
df2 = df[['VendorID' , 'passenger_count', 'fare_amount', 'mean_distance',␣
 ↪'mean_duration', 'rush_hour']]
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
```

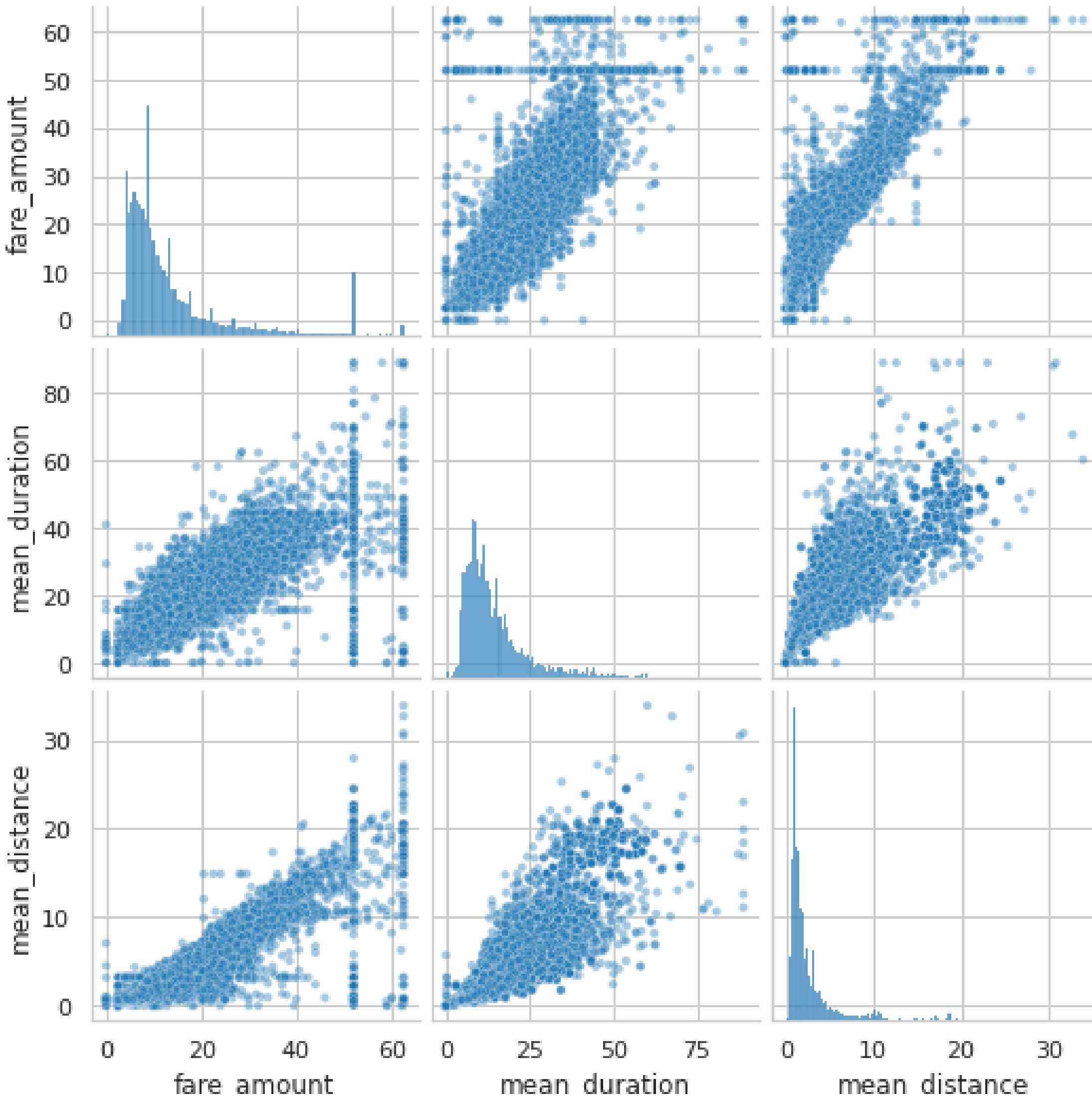
```
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   VendorID         22699 non-null  int64  
 1   passenger_count  22699 non-null  int64  
 2   fare_amount      22699 non-null  float64
 3   mean_distance    22699 non-null  float64
 4   mean_duration    22699 non-null  float64
 5   rush_hour        22699 non-null  int64  
dtypes: float64(3), int64(3)
memory usage: 1.0 MB
```

4.2.10 Task 6. Pair plot

Create a pairplot to visualize pairwise relationships between `fare_amount`, `mean_duration`, and `mean_distance`.

[132]:
```
# Create a pairplot to visualize pairwise relationships between variables in
↪the data
### YOUR CODE HERE ###
sns.set_context('notebook')
sns.pairplot(df2[['fare_amount', 'mean_duration', 'mean_distance']], 
↪plot_kws={'alpha':0.4, 'size':5})
```

[132]:
```
<seaborn.axisgrid.PairGrid at 0x7efcea8c8cd0>
```



These variables all show linear correlation with each other. Investigate this further.

4.2.11 Task 7. Identify correlations

Next, code a correlation matrix to help determine most correlated variables.

[135]:
```
# Correlation matrix to help determine most correlated variables
### YOUR CODE HERE ###
correlation_matrix = df2.corr(method='pearson')
```

Visualize a correlation heatmap of the data.

[142]:
```python
# Create correlation heatmap
### YOUR CODE HERE ###
plt.figure(figsize=(7,5))
sns.heatmap(correlation_matrix, annot=True, cmap='Reds')
plt.xticks(rotation=45)
```

[142]:
```
(array([0.5, 1.5, 2.5, 3.5, 4.5, 5.5]),
 <a list of 6 Text major ticklabel objects>)
```

Question: Which variable(s) are correlated with the target variable of `fare_amount`?

Try modeling with both variables even though they are correlated.

4.3 PACE: Construct

After analysis and deriving variables with close relationships, it is time to begin constructing the model. Consider the questions in your PACE Strategy Document to reflect on the Construct stage.

4.3.1 Task 8a. Split data into outcome variable and features

[41]:
```
### YOUR CODE HERE ###
```

Set your X and y variables. X represents the features and y represents the outcome (target) variable.

[192]:
```
# Remove the target column from the features
# X = df2.drop(columns='fare_amount')
### YOUR CODE HERE ###
X = df2.drop(['fare_amount'], axis=1)

# Set y variable
### YOUR CODE HERE ###
y = df2[['fare_amount']]

# Display first few rows
### YOUR CODE HERE ###
X.head()
```

[192]:

	VendorID	passenger_count	mean_distance	mean_duration	rush_hour
0	2	6	3.521667	22.847222	0
1	1	1	3.108889	24.470370	0
2	1	1	0.881429	7.250000	1
3	2	1	3.700000	30.250000	0
4	2	1	4.435000	14.616667	0

4.3.2 Task 8b. Pre-process data

Dummy encode categorical variables

[193]:
```
# Convert VendorID to string
### YOUR CODE HERE ###
df2['VendorID'] = df2['VendorID'].astype('category')

# Get dummies
### YOUR CODE HERE ###
X = pd.get_dummies(X, drop_first=True)
X.head()
```

[193]:

	passenger_count	mean_distance	mean_duration	rush_hour	VendorID_2
0	6	3.521667	22.847222	0	1
1	1	3.108889	24.470370	0	0
2	1	0.881429	7.250000	1	0
3	1	3.700000	30.250000	0	1
4	1	4.435000	14.616667	0	1

4.3.3 Normalize the data

Use StandardScaler() and fit_transform() to standardize the X variables. Assign the results to a variable called X_scaled.

[194]:
```
# Standardize the X variables
### YOUR CODE HERE ###
scalar = StandardScaler()
X_scaled = scalar.fit_transform(X)
X_scaled
```

[194]:
```
array([[ 3.39065627,  0.17093801,  0.83195364, -0.64959666,  0.8931955 ],
       [-0.4997803 ,  0.05495383,  0.99296921, -0.64959666, -1.11957573],
       [-0.4997803 , -0.57092814, -0.7152838 ,  1.53941679, -1.11957573],
       …,
       [-0.4997803 , -0.62633441, -0.77886169, -0.64959666,  0.8931955 ],
       [-0.4997803 , -0.23485053,  0.21719198, -0.64959666,  0.8931955 ],
       [-0.4997803 , -0.40359028, -0.50145366, -0.64959666, -1.11957573]])
```

4.3.4 Split data into training and test sets

Create training and testing sets. The test set should contain 20% of the total samples. Set random_state=0.

[195]:
```
# Create training and testing sets
#### YOUR CODE HERE ####
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=0)
```

Instantiate your model and fit it to the training data.

[196]:
```
# Fit your model to the training data
### YOUR CODE HERE ###
lr=LinearRegression()
lr.fit(X_train, y_train)
```

[196]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

4.3.5 Task 8c. Evaluate model

4.3.6 Train data

Evaluate your model performance by calculating the residual sum of squares and the explained variance score (R^2). Calculate the Mean Absolute Error, Mean Squared Error, and the Root

Mean Squared Error.

[197]:
```
# Evaluate the model performance on the training data
### YOUR CODE HERE ###
r_sq = lr.score(X_train, y_train)
print('Coefficient of determination:', r_sq)
y_pred_train = lr.predict(X_train)
print('R^2:', r2_score(y_train, y_pred_train))
print('MAE:', mean_absolute_error(y_train, y_pred_train))
print('MSE:', mean_squared_error(y_train, y_pred_train))
print('RMSE:',np.sqrt(mean_squared_error(y_train, y_pred_train)))
```

```
Coefficient of determination: 0.8398434585044773
R^2: 0.8398434585044773
MAE: 2.186666416775414
MSE: 17.88973296349268
RMSE: 4.229625629236313
```

4.3.7 Test data

Calculate the same metrics on the test data.

[198]:
```
# Evaluate the model performance on the testing data
### YOUR CODE HERE ###

r_sq_test = lr.score(X_test, y_test)
print('Coefficient of determination:', r_sq_test)
y_pred_test = lr.predict(X_test)
print('R^2:', r2_score(y_test, y_pred_test))
print('MAE:', mean_absolute_error(y_test,y_pred_test))
print('MSE:', mean_squared_error(y_test, y_pred_test))
print('RMSE:',np.sqrt(mean_squared_error(y_test, y_pred_test)))
```

```
Coefficient of determination: 0.8682583641795454
R^2: 0.8682583641795454
MAE: 2.133654984059387
MSE: 14.326454156998947
RMSE: 3.7850302716093234
```

4.4 PACE: Execute

Consider the questions in your PACE Strategy Document to reflect on the Execute stage.

4.4.1 Task 9a. Results

Use the code cell below to get `actual`,`predicted`, and `residual` for the testing set, and store them as columns in a `results` dataframe.

[199]:
```python
# Create a `results` dataframe
### YOUR CODE HERE ###

results = pd.DataFrame(data={'actual': y_test['fare_amount'],
                             'predicted': y_pred_test.ravel()})
results['residual'] = results['actual'] - results['predicted']
results.head()
```

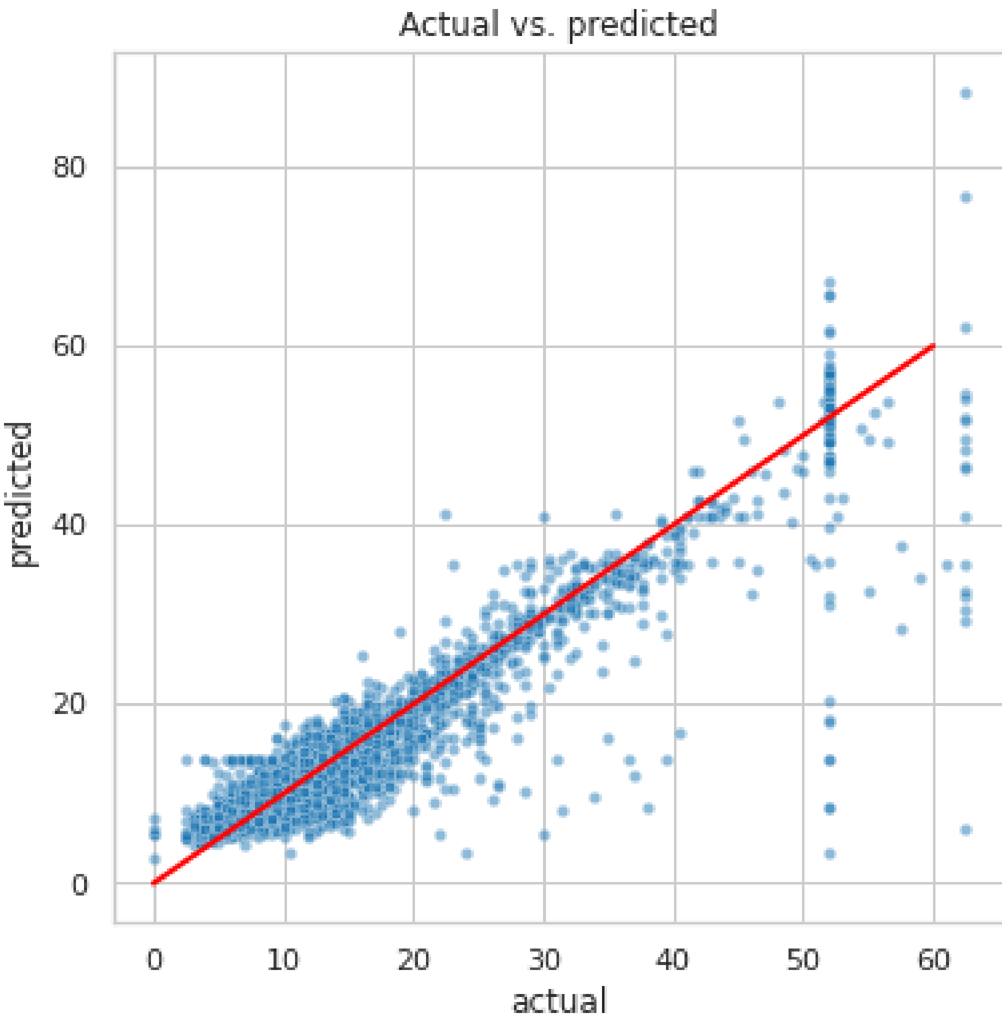
[199]:

	actual	predicted	residual
5818	14.0	12.356503	1.643497
18134	28.0	16.314595	11.685405
4655	5.5	6.726789	-1.226789
7378	15.5	16.227206	-0.727206
13914	9.5	10.536408	-1.036408

4.4.2 Task 9b. Visualize model results

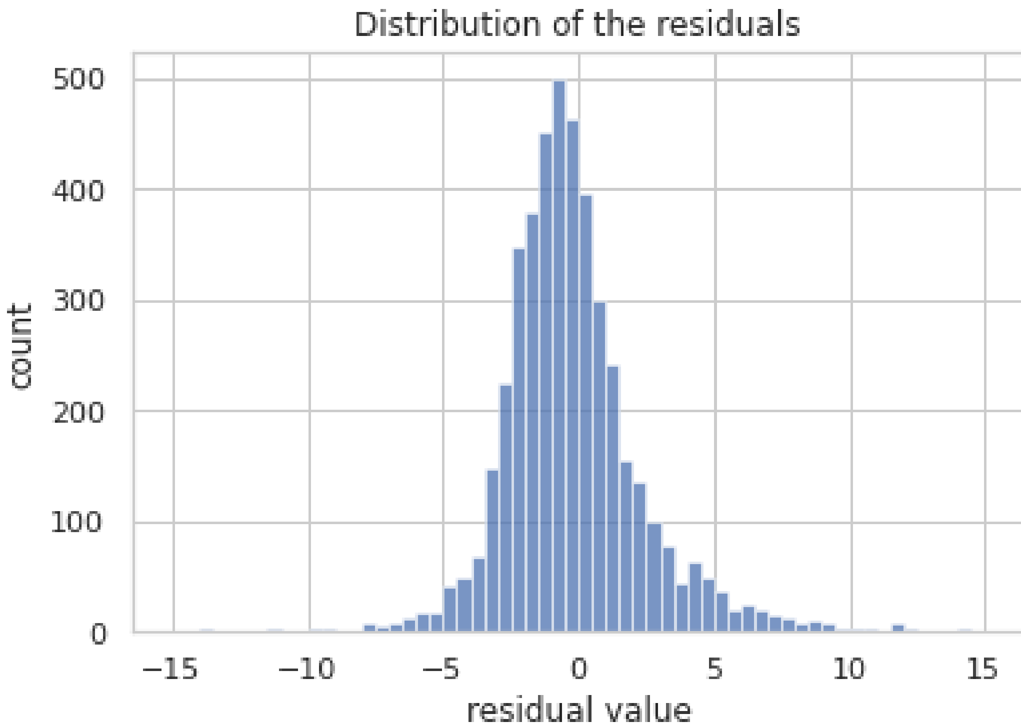
Create a scatterplot to visualize `actual` vs. `predicted`.

[200]:
```python
# Create a scatterplot to visualize `predicted` over `actual`
### YOUR CODE HERE ###
fig, ax = plt.subplots(figsize=(6, 6))
sns.set(style='whitegrid')
sns.scatterplot(x='actual',
                y='predicted',
                data=results,
                s=20,
                alpha=0.5,
                ax=ax
)
plt.plot([0,60], [0,60], c='red', linewidth=2)
plt.title('Actual vs. predicted');
```



Visualize the distribution of the `residuals` using a histogram.

[201]:
```
# Visualize the distribution of the `residuals`
### YOUR CODE HERE ###

sns.histplot(results['residual'], bins=np.arange(-15,15.5,0.5))
plt.title('Distribution of the residuals')
plt.xlabel('residual value')
plt.ylabel('count');
```



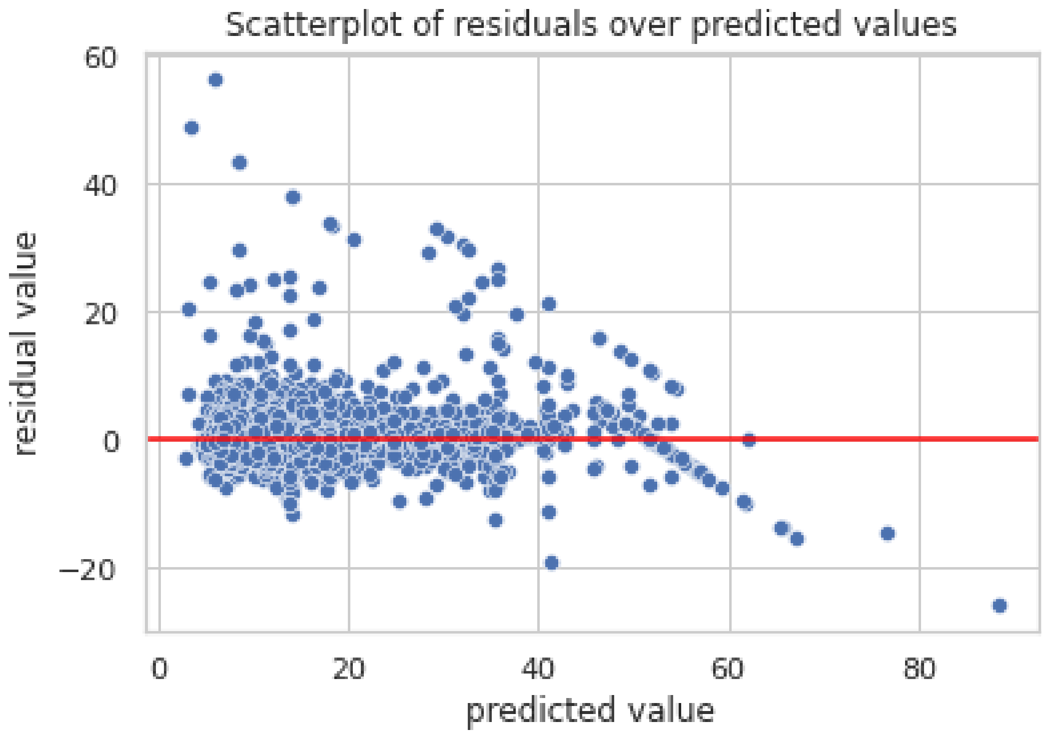
[202]:
```
# Calculate residual mean
### YOUR CODE HERE ###
results['residual'].mean()
```

[202]: -0.015442621528681361

The distribution of the residuals is normal and has a mean of -0.015. The residuals represent the variance in the outcome variable that is not explained by the model. A normal distribution around zero is good, as it demonstrates that the models errors are evenly distributed and unbiased.

Create a scatterplot of `residuals` over `predicted`.

[203]:
```
# Create a scatterplot of `residuals` over `predicted`
### YOUR CODE HERE ###
sns.scatterplot(x='predicted', y='residual', data=results)
plt.axhline(0, c='red')
plt.title('Scatterplot of residuals over predicted values')
plt.xlabel('predicted value')
plt.ylabel('residual value')
plt.show()
```



The model's residuals are evenly distributed above and below zero, with the exception of the sloping lines from the upper-left corner to the lower-right corner, which are the imputed maximum of $62.50 and the flat rate of $52 for JFK airport trips

4.4.3 Task 9c. Coefficients

Use the `coef_` attribute to get the model's coefficients. The coefficients are output in the order of the features that were used to train the model. Which feature had the greatest effect on trip fare?

[204]:
```python
# Output the model's coefficients
coefficients = pd.DataFrame(lr.coef_, columns=X.columns)
coefficients
```

[204]:

	passenger_count	mean_distance	mean_duration	rush_hour	VendorID_2
0	0.030755	7.102335	2.806779	0.110278	-0.054376

The coefficients reveal that `mean_distance` was the feature with the greatest weight in the model's final prediction. For every mile traveled, the fare amount increases by a mean of \$7. Note, however, that because some highly correlated features were not removed, the confidence interval of this assessment is wider.

4.4.4 Task 9d. Conclusion

1. What are the key takeaways from this notebook?
2. What results can be presented from this notebook?

- Multiple linear regression is a powerful tool to estimate a dependent continous variable from several independent variables.
- Exploratory data analysis is useful for selecting both numeric and categorical features for multiple linear regression.
- Fitting multiple linear regression models may require trial and error to select variables that fit an accurate model while maintaining model assumptions
- The Multiple Linear Regreession Model was built using `passenger_count`, `mean_distance`, `mean_duration`, `rush_hour`, and `VendorID` variables.
- The feature `mean_distance` has the greatest impact on the model, which means that every mile traveled will incease in a mean of $7 fare amount.
- The confidence intervals for the features is wider and not computed for the sake of prediction only.
- The R^2 score of the model is 0.83 on the traning data, which means that 83 percent of the variance in `fare_amount` can be explained by the model.
- It is important to be noted that the model includes information that there were several rides which were betweek JDK airport and Manhatten, there are fares of USD 52 in the fare_amounts, indicating that in 2017 there were flat rates of USD 52 between JDK airport and Manhatten. The most of rides include toll fee around $5.76, which is also incorporated by the model.