NYC TLC Project Part 4

November 5, 2024

1 NYC TLC Project Part 4

To build a multiple linear regression model to predict taxi fares using existing data that was collected over the course of a year.

2 Build a multiple linear regression model

In this project, we will build a multiple linear regression model.Multiple linear regression helps us 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 we're measuring against. This opens the door for much more thorough and flexible analysis to be completed.

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

Part 2: Model Building and evaluation

Part 3: Interpreting Model Results

2.0.1 Task 1. Imports and loading

Import the packages needed for building linear regression models.

[1]:
```
# Imports
# Packages for numerics + dataframes

import pandas as pd
import numpy as np

# Packages for visualization

import seaborn as sns
```

```python
import matplotlib.pyplot as plt

# Packages for date conversions for calculating trip durations

from datetime import datetime
from datetime import date
from datetime import timedelta

# Packages for OLS, MLR, confusion matrix

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

[2]:
```python
# Load dataset into dataframe
df0=pd.read_csv("2017_Yellow_Taxi_Trip_Data.csv")
```

2.0.2 Task 2a. Explore data with EDA

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

Start with .shape and .info().

[3]:
```python
# Start with `.shape` and `.info()`

print(df0.shape)
print(df0.info())
```

```
(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
None
```

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

[5]:
```python
# Check for missing data and duplicates using .isna() and .drop_duplicates()

df0.isna().sum()
```

[5]:
```
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()`.

[6]:
```python
# Use .describe()

df=df0.copy()
df.describe()
```

[6]:
```
         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
```

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

[7]:
```python
# Check the format of the data

df['tpep_dropoff_datetime'][0]
```

[7]:
```
'03/25/2017 9:09:47 AM'
```

[9]:
```python
# Convert datetime columns to datetime

print('Data type of dropoff datetime: ',df['tpep_dropoff_datetime'].dtype)
print('Data type of pickup datetime: ',df['tpep_pickup_datetime'].dtype)

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

print('Data type of dropoff datetime: ',df['tpep_dropoff_datetime'].dtype)
print('Data type of pickup datetime: ',df['tpep_pickup_datetime'].dtype)
```

```
Data type of dropoff datetime:  object
Data type of pickup datetime:  object
Data type of dropoff datetime:  datetime64[ns]
Data type of pickup datetime:  datetime64[ns]
```

[11]:
```python
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
```

2.0.4 Task 2c. Create duration column

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

[13]:
```
# Create `duration` column

df['duration'] = (df['tpep_dropoff_datetime']-df['tpep_pickup_datetime'])/np.
↪timedelta64(1,'m')
```

2.0.5 Outliers

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

[14]:
```
df.info()
```

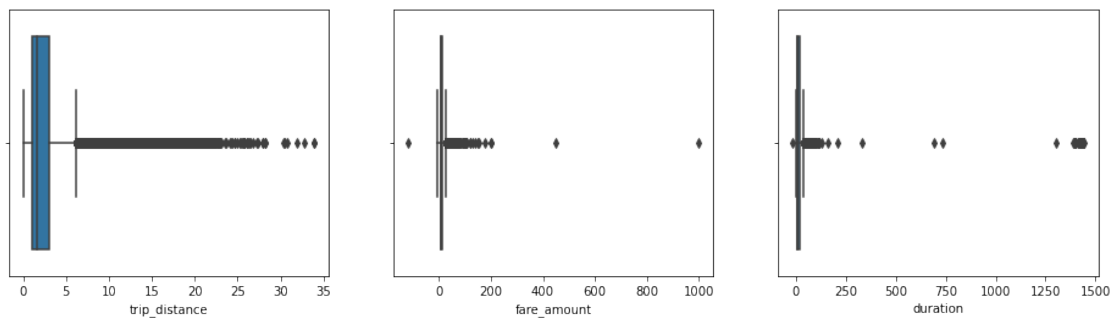
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             22699 non-null  int64
 1   VendorID               22699 non-null  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 our model, the most important columns to check for outliers are likely to be: * `trip_distance` * `fare_amount` * `duration`

2.0.6 Task 2d. Box plots

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

[15]:
```
fig, axes = plt.subplots(1,3,figsize=(16,4))
sns.boxplot(ax=axes[0], x=df['trip_distance'])
sns.boxplot(ax=axes[1], x=df['fare_amount'])
sns.boxplot(ax=axes[2], x=df['duration'])
plt.show()
```



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.

2.0.7 Task 2e. Imputations

`trip_distance` outliers From the summary statistics we know 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?

[16]:
```
# Are trip distances of 0 bad data or very short trips rounded down?

sorted(set(df['trip_distance']))[:10]
```

[16]: `[0.0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09]`

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.

[17]:
```
sum(df['trip_distance']==0)
```

[17]:
```
148
```

fare_amount outliers

[18]:
```
df['fare_amount'].describe()
```

[18]:
```
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
```

The range of values in the `fare_amount` column is large and the extremes don't make much sense.

- **Low values:** Negative values are problematic. Values of zero could be legitimate if the taxi logged a trip that was immediately canceled.
- **High values:** The maximum fare amount in this dataset is nearly \$1,000, which seems very unlikely. High values for this feature can be capped based on intuition and statistics. The interquartile range (IQR) is \$8. 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`.

[19]:
```
# Impute values less than $0 with 0

df.loc[df['fare_amount'] < 0, 'fare_amount'] = 0
df['fare_amount'].min()
```

[19]:
```
0.0
```

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

[21]:
```
def outlier_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
            df.loc[df[col] < 0, col] = 0
        ### YOUR CODE HERE ###

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

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

[22]:
```
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
```

duration outliers

[23]:
```python
# Call .describe() for duration outliers

df['duration'].describe()
```

[23]:
```
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)`.

[24]:
```python
# Impute a 0 for any negative values

df.loc[df['duration']<0,'duration'] = 0
df['duration'].min()
```

[24]:
```
0.0
```

[25]:
```python
# 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
```

2.0.8 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 we cannot train a model that uses this feature. However, we can use the statistics of trips we *do* know to generalize about ones we 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 our 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'

[26]:
```
# Create `pickup_dropoff` column

df['pickup_dropoff'] = df['PULocationID'].astype(str) + ' ' +␣
↪df['DOLocationID'].astype(str)
```

```
df['pickup_dropoff'].head(5)
```

[26]:
```
0     100 231
1      186 43
2     262 236
3      188 97
4       4 112
Name: pickup_dropoff, dtype: object
```

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`.

[27]:
```
grouped = df.groupby('pickup_dropoff').
 ↪mean(numeric_only=True)[['trip_distance']]
grouped[:5]
```

[27]:
```
                trip_distance
pickup_dropoff
1 1                  2.433333
10 148              15.700000
100 1               16.890000
100 100              0.253333
100 107              1.180000
```

`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 we 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}
```

[29]:
```
# 1. Convert `grouped` to a dictionary

grouped_dict = grouped.to_dict()

# 2. Reassign to only contain the inner dictionary

grouped_dict = grouped_dict['trip_distance']
```

[29]:
```
{'1 1': 2.433333333333333,
 '10 148': 15.7,
 '100 1': 16.89,
 '100 100': 0.25333333333333335,
 '100 107': 1.18,
 '100 113': 2.024,
 '100 114': 1.94,
 '100 12': 4.55,
 '100 125': 2.84,
 '100 13': 4.201666666666667,
 '100 132': 17.2175,
 '100 137': 1.299,
 '100 138': 10.432857142857143,
 '100 140': 2.746,
 '100 141': 2.11,
 '100 142': 1.6958333333333335,
 '100 143': 1.5825,
 '100 144': 3.0066666666666664,
 '100 148': 4.1066666666666665,
 '100 151': 3.668,
 '100 152': 4.9,
 '100 158': 1.938,
 '100 161': 0.9813888888888889,
 '100 162': 1.2163636363636363,
 '100 163': 1.2656,
 '100 164': 0.841,
 '100 166': 5.199999999999999,
 '100 170': 0.8548,
 '100 177': 12.0,
 '100 181': 9.34,
 '100 186': 0.6404761904761904,
 '100 193': 4.39,
 '100 198': 9.01,
 '100 202': 5.3,
 '100 209': 4.43,
 '100 211': 2.48,
 '100 224': 1.9500000000000002,
 '100 225': 7.5,
 '100 229': 1.7850000000000001,
 '100 230': 0.72975,
 '100 231': 3.5216666666666665,
 '100 232': 3.8449999999999998,
 '100 233': 1.2458333333333333,
 '100 234': 1.2545454545454546,
 '100 236': 3.3375,
 '100 237': 2.5566666666666666,
 '100 238': 3.3560000000000003,
```

'100 239': 2.327142857142857,
'100 243': 8.77,
'100 244': 7.9,
'100 246': 1.1746666666666667,
'100 249': 1.8066666666666666,
'100 25': 7.36,
'100 255': 6.35,
'100 256': 5.859999999999999,
'100 261': 3.8075,
'100 262': 3.8200000000000003,
'100 263': 3.4,
'100 39': 22.6,
'100 4': 2.6999999999999997,
'100 40': 7.23,
'100 41': 4.6,
'100 42': 6.779999999999999,
'100 43': 2.033333333333333,
'100 45': 3.63,
'100 48': 0.8522727272727273,
'100 49': 7.35,
'100 50': 1.1800000000000002,
'100 66': 4.7,
'100 68': 0.9942857142857143,
'100 7': 4.9,
'100 74': 4.53,
'100 75': 4.03,
'100 79': 2.608571428571428,
'100 87': 5.03,
'100 88': 5.495,
'100 90': 1.1228571428571428,
'100 95': 9.0,
'106 106': 0.02,
'106 181': 1.1,
'106 228': 1.24,
'106 231': 3.8,
'106 40': 0.8,
'107 1': 15.55,
'107 100': 1.436,
'107 107': 0.48814814814814816,
'107 113': 0.8969230769230769,
'107 114': 1.207142857142857,
'107 125': 1.8,
'107 127': 11.57,
'107 13': 3.8733333333333335,
'107 130': 12.43,
'107 132': 16.755,
'107 137': 0.6828947368421052,

'107 138': 10.385,
'107 140': 2.801428571428571,
'107 141': 2.981666666666667,
'107 142': 3.228,
'107 143': 4.3,
'107 144': 1.61625,
'107 145': 3.5300000000000002,
'107 146': 4.3,
'107 147': 8.11,
'107 148': 1.7266666666666666,
'107 152': 6.62,
'107 158': 1.7777777777777777,
'107 161': 1.7091666666666667,
'107 162': 1.5677272727272729,
'107 163': 2.4775,
'107 164': 0.748,
'107 170': 1.0014285714285716,
'107 186': 1.4310344827586208,
'107 196': 7.890000000000001,
'107 202': 5.86,
'107 209': 3.496,
'107 21': 11.5,
'107 211': 1.7342857142857144,
'107 223': 5.7,
'107 224': 0.8533333333333334,
'107 229': 1.911111111111111,
'107 23': 17.72,
'107 230': 2.1075,
'107 231': 3.278,
'107 232': 2.1418181818181816,
'107 233': 1.3471428571428572,
'107 234': 0.6842424242424242,
'107 236': 3.165,
'107 237': 2.2516666666666665,
'107 238': 4.986666666666667,
'107 244': 8.84,
'107 246': 1.577142857142857,
'107 249': 1.3976470588235295,
'107 25': 4.5,
'107 256': 3.8666666666666667,
'107 257': 9.0,
'107 26': 10.33,
'107 261': 2.19,
'107 262': 3.5725,
'107 263': 3.66875,
'107 265': 4.5,
'107 36': 5.6,

'107 37': 4.9,
'107 4': 1.16,
'107 41': 5.975,
'107 42': 7.0,
'107 43': 2.8,
'107 45': 2.1375,
'107 48': 2.5460000000000003,
'107 49': 5.0,
'107 66': 3.58,
'107 68': 1.3646666666666667,
'107 7': 5.95,
'107 74': 5.085,
'107 75': 4.930000000000001,
'107 79': 0.9903225806451613,
'107 80': 4.7,
'107 82': 7.32,
'107 87': 3.5,
'107 88': 3.501666666666667,
'107 89': 6.97,
'107 90': 0.9226666666666666,
'112 112': 0.7,
'112 223': 4.05,
'112 263': 8.0,
'112 49': 3.1,
'112 66': 4.57,
'112 80': 0.41500000000000004,
'113 100': 1.9775,
'113 106': 4.69,
'113 107': 1.0516666666666667,
'113 112': 4.5,
'113 113': 0.8346153846153845,
'113 114': 0.7644444444444445,
'113 116': 8.55,
'113 125': 1.2175,
'113 13': 2.2085714285714286,
'113 137': 1.3325,
'113 138': 10.4,
'113 14': 15.62,
'113 140': 3.3,
'113 141': 3.6333333333333333,
'113 142': 3.6,
'113 143': 4.39,
'113 144': 1.1291666666666667,
'113 146': 5.2,
'113 148': 1.1885714285714286,
'113 152': 8.0,
'113 158': 0.9530769230769232,

'113 161': 2.2449999999999997,
'113 162': 2.2125,
'113 163': 2.515,
'113 164': 1.4830769230769232,
'113 17': 4.38,
'113 170': 1.5743749999999999,
'113 181': 5.546666666666667,
'113 186': 1.4091666666666667,
'113 209': 2.8933333333333333,
'113 211': 1.0183333333333333,
'113 22': 12.0,
'113 224': 1.4649999999999999,
'113 230': 2.11,
'113 231': 1.7366666666666668,
'113 232': 2.1375,
'113 233': 2.1500000000000004,
'113 234': 0.8635714285714285,
'113 236': 3.9924999999999997,
'113 237': 3.412,
'113 238': 5.324,
'113 239': 5.054,
'113 243': 12.4,
'113 244': 9.4,
'113 246': 2.21,
'113 249': 0.7723076923076924,
'113 255': 3.7,
'113 256': 3.3,
'113 261': 2.0033333333333334,
'113 262': 4.4,
'113 263': 4.0,
'113 264': 0.0,
'113 33': 3.8266666666666667,
'113 36': 6.3,
'113 4': 1.09,
'113 41': 9.2,
'113 42': 8.34,
'113 45': 2.02,
'113 48': 2.2960000000000003,
'113 50': 2.86,
'113 66': 3.2,
'113 68': 1.1981818181818182,
'113 79': 0.7782352941176471,
'113 80': 5.31,
'113 87': 3.135,
'113 88': 2.5999999999999996,
'113 90': 0.8544444444444445,
'113 94': 12.5,

'114 100': 2.3966666666666665,
'114 107': 1.2999999999999998,
'114 112': 5.13,
'114 113': 0.61,
'114 114': 0.5481818181818182,
'114 116': 9.1,
'114 125': 0.752,
'114 13': 1.9000000000000001,
'114 137': 1.8425,
'114 14': 11.11,
'114 140': 4.1,
'114 141': 3.9333333333333336,
'114 142': 3.8766666666666665,
'114 143': 4.745,
'114 144': 0.84875,
'114 145': 6.53,
'114 148': 0.9109090909090909,
'114 151': 7.3,
'114 158': 1.356,
'114 161': 2.80375,
'114 162': 2.7075,
'114 163': 3.5,
'114 164': 1.7079999999999997,
'114 166': 7.55,
'114 169': 11.6,
'114 170': 2.0833333333333335,
'114 181': 3.77,
'114 186': 1.6784615384615384,
'114 190': 4.7,
'114 209': 1.66,
'114 211': 0.45,
'114 217': 2.4,
'114 223': 7.6,
'114 224': 5.23,
'114 225': 4.41,
'114 229': 3.0,
'114 230': 2.8775,
'114 231': 1.2021428571428572,
'114 232': 1.3399999999999999,
'114 233': 2.3375,
'114 234': 1.3515384615384616,
'114 236': 4.5649999999999995,
'114 237': 3.716666666666667,
'114 238': 5.89,
'114 239': 4.99,
'114 24': 6.673333333333333,
'114 243': 11.23,

'114 244': 8.9,
'114 246': 2.0066666666666664,
'114 249': 0.8476923076923076,
'114 255': 3.6879999999999997,
'114 257': 4.96,
'114 260': 7.08,
'114 261': 1.7333333333333334,
'114 262': 5.505,
'114 263': 4.75,
'114 36': 5.98,
'114 4': 1.35,
'114 43': 3.6,
'114 45': 1.0866666666666667,
'114 48': 3.3966666666666665,
'114 49': 3.965,
'114 50': 3.315,
'114 62': 5.7,
'114 65': 2.8,
'114 66': 3.3,
'114 68': 1.7022222222222223,
'114 69': 10.05,
'114 7': 6.4,
'114 79': 1.0336363636363635,
'114 87': 2.035,
'114 90': 1.3199999999999998,
'114 97': 3.7,
'116 116': 0.47800000000000004,
'116 119': 2.9,
'116 132': 19.05,
'116 159': 1.64,
'116 162': 6.1,
'116 166': 1.3275000000000001,
'116 186': 6.42,
'116 230': 6.38,
'116 238': 3.2975000000000003,
'116 239': 4.52,
'116 244': 1.085,
'116 41': 1.7149999999999999,
'116 42': 1.5825,
'116 68': 6.31,
'116 74': 2.08,
'116 75': 4.23,
'116 79': 9.3,
'118 118': 1.43,
'12 100': 4.0,
'12 13': 0.9,
'12 142': 5.56,

'12 144': 2.08,
'12 151': 8.3,
'12 163': 5.5,
'12 164': 5.38,
'12 170': 4.9,
'12 48': 4.67,
'123 123': 0.93,
'125 1': 14.67,
'125 100': 2.1100000000000003,
'125 106': 5.0,
'125 107': 2.1316666666666664,
'125 113': 0.7,
'125 114': 0.825,
'125 129': 8.13,
'125 13': 1.3,
'125 132': 19.88,
'125 137': 2.68,
'125 138': 10.4575,
'125 140': 4.955,
'125 141': 6.5,
'125 142': 4.8,
'125 144': 0.6583333333333333,
'125 148': 1.3885714285714283,
'125 151': 5.43,
'125 158': 0.7375,
'125 161': 2.783333333333333,
'125 162': 3.245,
'125 163': 3.4000000000000004,
'125 164': 2.615,
'125 170': 4.0275,
'125 186': 1.8719999999999999,
'125 188': 5.88,
'125 211': 0.66,
'125 227': 9.4,
'125 230': 2.71,
'125 231': 1.0077777777777779,
'125 234': 1.7242857142857144,
'125 236': 4.8,
'125 237': 3.47,
'125 238': 6.09,
'125 239': 5.05,
'125 244': 9.07,
'125 246': 2.265,
'125 249': 0.6785714285714286,
'125 255': 4.050000000000001,
'125 256': 3.1,
'125 261': 2.01,

'125 263': 5.949999999999999,
'125 42': 8.16,
'125 48': 2.74,
'125 49': 4.21,
'125 68': 1.93,
'125 75': 7.2,
'125 79': 1.5642857142857143,
'125 87': 2.0875,
'125 88': 2.505,
'125 90': 1.435,
'125 97': 4.835,
'127 243': 1.92,
'128 238': 7.3,
'129 129': 0.808,
'129 160': 6.3,
'129 164': 1.96,
'129 173': 2.1,
'129 207': 1.2,
'129 70': 1.69,
'13 100': 3.9799999999999995,
'13 107': 4.665,
'13 113': 2.75,
'13 114': 2.0975,
'13 12': 0.9,
'13 125': 0.93,
'13 13': 0.518,
'13 132': 24.505,
'13 137': 5.045,
'13 138': 15.219999999999999,
'13 14': 7.1,
'13 140': 6.955,
'13 141': 7.4399999999999995,
'13 142': 5.1,
'13 143': 5.15,
'13 144': 2.1500000000000004,
'13 148': 3.3075,
'13 158': 2.35,
'13 161': 5.9030000000000005,
'13 162': 6.3725,
'13 163': 5.172857142857143,
'13 164': 6.02,
'13 166': 7.46,
'13 17': 5.6,
'13 170': 6.166666666666667,
'13 181': 3.9,
'13 186': 3.7733333333333334,
'13 209': 1.8,

'13 211': 1.7325,
'13 224': 4.8,
'13 225': 7.51,
'13 226': 8.3,
'13 229': 6.343999999999999,
'13 230': 4.56,
'13 231': 0.9583333333333334,
'13 232': 3.13,
'13 233': 6.2,
'13 234': 3.8175,
'13 236': 8.33,
'13 237': 7.136666666666667,
'13 238': 6.7,
'13 239': 5.715,
'13 244': 10.575,
'13 246': 2.7916666666666665,
'13 249': 2.145,
'13 25': 3.0,
'13 255': 5.53,
'13 261': 0.78,
'13 262': 8.11,
'13 263': 8.07,
'13 33': 3.99,
'13 40': 2.8,
'13 45': 2.4,
'13 48': 4.154285714285714,
'13 49': 5.43,
'13 50': 4.1,
'13 54': 4.62,
'13 55': 13.26,
'13 65': 3.9600000000000004,
'13 68': 2.7,
'13 74': 10.36,
'13 79': 3.9574999999999996,
'13 85': 10.99,
'13 87': 1.2125,
'13 88': 0.9,
'13 90': 2.7,
'13 91': 7.86,
'130 230': 12.8,
'130 64': 6.03,
'131 9': 2.1,
'132 10': 3.75,
'132 100': 17.6,
'132 102': 7.7,
'132 106': 20.2,
'132 107': 17.561666666666667,

'132 11': 17.945,
'132 112': 15.809999999999999,
'132 113': 18.302,
'132 114': 21.73,
'132 117': 12.2,
'132 121': 10.47,
'132 123': 15.65,
'132 124': 5.67,
'132 125': 18.736666666666668,
'132 13': 20.86,
'132 130': 6.716666666666666,
'132 132': 2.2558620689655173,
'132 134': 6.576666666666667,
'132 137': 16.720000000000002,
'132 138': 11.68625,
'132 14': 20.065,
'132 140': 19.293333333333333,
'132 141': 19.14,
'132 142': 20.406666666666666,
'132 143': 10.905000000000001,
'132 144': 18.537499999999998,
'132 145': 15.837142857142856,
'132 148': 17.994285714285716,
'132 149': 14.32,
'132 15': 14.4,
'132 150': 14.85,
'132 151': 19.834,
'132 152': 19.1,
'132 158': 22.7,
'132 161': 18.601666666666667,
'132 162': 17.082857142857144,
'132 163': 19.229,
'132 164': 18.7575,
'132 166': 18.6,
'132 17': 10.4,
'132 170': 17.203,
'132 174': 21.17,
'132 177': 9.2,
'132 179': 15.27,
'132 181': 17.358571428571427,
'132 186': 18.375,
'132 188': 12.149999999999999,
'132 189': 12.2,
'132 19': 10.5,
'132 195': 26.54,
'132 196': 9.35,
'132 197': 6.59,

'132 198': 9.9,
'132 201': 12.94,
'132 205': 6.0,
'132 209': 21.2,
'132 211': 18.91,
'132 212': 16.85,
'132 213': 15.2,
'132 215': 4.9,
'132 216': 4.487,
'132 218': 4.5,
'132 22': 17.9,
'132 220': 30.5,
'132 222': 8.21,
'132 223': 13.25,
'132 224': 17.59,
'132 225': 11.8,
'132 226': 14.959999999999999,
'132 228': 23.875,
'132 229': 18.49,
'132 23': 30.83,
'132 230': 18.5712,
'132 231': 20.46,
'132 232': 18.3,
'132 233': 17.86,
'132 234': 17.654,
'132 236': 19.491666666666667,
'132 237': 19.540000000000003,
'132 238': 20.8375,
'132 239': 20.90125,
'132 24': 19.14,
'132 241': 20.5,
'132 243': 22.1,
'132 244': 19.9,
'132 246': 18.515,
'132 248': 17.22,
'132 249': 18.7325,
'132 25': 14.808000000000002,
'132 252': 11.3,
'132 255': 16.466666666666665,
'132 256': 17.224,
'132 257': 19.81,
'132 259': 20.96,
'132 26': 13.92,
'132 261': 22.115000000000002,
'132 262': 19.165,
'132 263': 19.21,
'132 264': 0.0,

'132 265': 14.885833333333332,
'132 28': 6.3133333333333335,
'132 33': 18.683333333333334,
'132 36': 15.886666666666665,
'132 37': 14.899999999999999,
'132 38': 7.300000000000001,
'132 39': 9.912857142857144,
'132 4': 18.59,
'132 40': 14.1,
'132 42': 17.95,
'132 43': 18.744999999999997,
'132 48': 18.761904761904763,
'132 49': 11.92,
'132 50': 18.735,
'132 51': 19.064999999999998,
'132 52': 26.86,
'132 54': 27.2,
'132 55': 17.3,
'132 61': 10.69,
'132 62': 13.229999999999999,
'132 64': 13.6,
'132 65': 15.686000000000002,
'132 66': 19.3,
'132 68': 18.7975,
'132 7': 14.783333333333333,
'132 70': 11.3,
'132 71': 11.34,
'132 72': 10.19,
'132 74': 17.25,
'132 76': 9.256666666666666,
'132 77': 9.0,
'132 79': 19.43166666666667,
'132 80': 15.606666666666667,
'132 82': 10.335,
'132 83': 11.15,
'132 85': 13.45,
'132 86': 7.8,
'132 87': 19.96,
'132 88': 20.6,
'132 89': 14.71,
'132 9': 16.51,
'132 90': 18.666666666666668,
'132 91': 13.835,
'132 92': 10.515,
'132 93': 10.265,
'132 95': 8.1,
'132 97': 16.35,

'133 133': 4.43,
'134 197': 2.2,
'135 75': 12.85,
'137 100': 1.458,
'137 107': 0.6836363636363636,
'137 112': 5.1,
'137 113': 1.338,
'137 114': 1.7600000000000002,
'137 125': 2.33,
'137 13': 5.48,
'137 132': 22.26,
'137 135': 10.36,
'137 137': 0.4633333333333333,
'137 138': 8.4,
'137 14': 12.34,
'137 140': 2.186666666666667,
'137 141': 1.83,
'137 142': 3.08,
'137 145': 2.7,
'137 148': 1.4,
'137 158': 2.66,
'137 161': 1.46875,
'137 162': 1.1652173913043478,
'137 163': 1.99,
'137 164': 0.6784615384615384,
'137 170': 0.7204347826086956,
'137 181': 7.65,
'137 186': 0.9963636363636365,
'137 209': 4.09,
'137 220': 12.3,
'137 223': 7.03,
'137 224': 0.9,
'137 229': 1.264,
'137 230': 1.5200000000000002,
'137 231': 3.3175,
'137 232': 2.5949999999999998,
'137 233': 0.886875,
'137 234': 1.0316666666666667,
'137 236': 3.0,
'137 237': 2.2125,
'137 238': 5.4,
'137 239': 4.43,
'137 243': 9.69,
'137 246': 2.11,
'137 249': 2.1574999999999998,
'137 255': 4.35,
'137 261': 5.49,

'137 262': 3.005,
'137 263': 3.5325,
'137 4': 1.6219999999999999,
'137 41': 5.125,
'137 42': 6.91,
'137 43': 2.9,
'137 45': 2.36,
'137 48': 2.1533333333333333,
'137 50': 2.7,
'137 61': 8.133333333333333,
'137 68': 1.672,
'137 7': 4.2,
'137 74': 4.68,
'137 79': 1.30875,
'137 82': 5.8,
'137 87': 4.34,
'137 88': 4.449999999999999,
'137 90': 1.3,
'138 1': 32.72,
'138 100': 9.765,
'138 106': 11.0,
'138 107': 9.463333333333333,
'138 112': 7.25,
'138 113': 11.0875,
'138 114': 11.45,
'138 116': 8.015,
'138 121': 6.7,
'138 125': 14.567499999999999,
'138 127': 10.16,
'138 129': 4.00875,
'138 13': 14.410000000000002,
'138 130': 7.21,
'138 132': 12.577142857142858,
'138 134': 6.859999999999999,
'138 137': 8.75,
'138 138': 0.9528571428571428,
'138 14': 16.18,
'138 140': 8.88125,
'138 141': 9.37,
'138 142': 10.852307692307694,
'138 143': 10.466666666666667,
'138 144': 12.341666666666667,
'138 145': 7.723333333333333,
'138 146': 4.134,
'138 148': 11.54,
'138 15': 8.100000000000001,
'138 151': 9.15,

'138 152': 8.86,
'138 158': 13.899999999999999,
'138 160': 6.68,
'138 161': 10.131388888888889,
'138 162': 9.673,
'138 163': 10.425714285714285,
'138 164': 9.65157894736842,
'138 166': 8.26,
'138 17': 9.2425,
'138 170': 8.977058823529413,
'138 171': 7.33,
'138 174': 14.1,
'138 175': 9.3,
'138 177': 10.95,
'138 178': 18.225,
'138 179': 3.9266666666666663,
'138 180': 11.2,
'138 181': 11.776666666666666,
'138 182': 10.29,
'138 186': 11.034,
'138 188': 13.36,
'138 189': 13.433333333333332,
'138 192': 4.66,
'138 196': 5.6066666666666665,
'138 197': 7.495,
'138 198': 9.96,
'138 200': 12.3,
'138 209': 12.950000000000001,
'138 210': 20.5,
'138 211': 12.796666666666667,
'138 220': 13.39,
'138 223': 3.0500000000000003,
'138 224': 9.870000000000001,
'138 225': 9.817499999999999,
'138 226': 4.58,
'138 229': 10.012,
'138 230': 10.601590909090909,
'138 231': 13.138333333333334,
'138 232': 11.485,
'138 233': 8.775555555555556,
'138 234': 10.5275,
'138 236': 8.83,
'138 237': 9.46625,
'138 238': 9.202222222222222,
'138 239': 10.172727272727274,
'138 243': 10.42,
'138 244': 10.085,

'138 246': 10.52,
'138 249': 11.162,
'138 25': 10.888333333333334,
'138 252': 5.23,
'138 255': 7.4322222222222222,
'138 256': 8.68,
'138 257': 16.4,
'138 260': 4.27,
'138 261': 16.1,
'138 262': 8.055,
'138 263': 8.690000000000001,
'138 265': 20.552,
'138 29': 21.65,
'138 33': 11.03,
'138 36': 7.7299999999999995,
'138 37': 8.565,
'138 4': 10.49,
'138 41': 8.065,
'138 42': 7.01,
'138 43': 10.445,
'138 48': 10.377692307692307,
'138 49': 9.823333333333332,
'138 50': 10.78,
'138 51': 13.8,
'138 52': 12.3,
'138 53': 5.3,
'138 56': 4.1,
'138 61': 13.31,
'138 62': 10.11,
'138 64': 12.36,
'138 65': 9.932857142857143,
'138 66': 10.6,
'138 68': 11.475000000000001,
'138 69': 7.5,
'138 7': 3.61125,
'138 70': 1.4266666666666667,
'138 74': 6.7875,
'138 75': 8.1475,
'138 79': 10.967500000000001,
'138 80': 6.99,
'138 81': 14.47,
'138 82': 3.52,
'138 83': 3.335,
'138 87': 13.8125,
'138 88': 15.393333333333333,
'138 89': 15.11,
'138 90': 10.943999999999999,

'138 92': 3.65,
'138 93': 4.29,
'138 95': 5.085,
'138 97': 11.11,
'138 98': 8.7,
'14 14': 0.38,
'140 107': 3.115,
'140 113': 4.1975,
'140 125': 8.28,
'140 13': 7.64,
'140 132': 18.7,
'140 135': 14.48,
'140 137': 2.58,
'140 138': 9.305,
'140 140': 0.5986363636363636,
'140 141': 0.7527272727272727,
'140 142': 1.7822222222222222,
'140 143': 2.32,
'140 151': 2.83,
'140 161': 1.8414285714285714,
'140 162': 1.4966666666666666,
'140 163': 1.5966666666666667,
'140 164': 2.4683333333333333,
'140 166': 4.2,
'140 170': 2.122,
'140 179': 4.22,
'140 186': 3.31625,
'140 193': 4.15,
'140 209': 6.17,
'140 211': 5.53,
'140 223': 6.4,
'140 224': 3.1,
'140 226': 3.62,
'140 229': 1.1131578947368421,
'140 230': 2.41,
'140 231': 7.45,
'140 232': 5.75,
'140 233': 1.7408333333333335,
'140 234': 3.5033333333333334,
'140 236': 1.2205714285714286,
'140 237': 0.9748837209302326,
'140 238': 2.196,
'140 239': 2.347142857142857,
'140 24': 4.36,
'140 243': 6.84,
'140 244': 7.56,
'140 246': 4.95,

'140 249': 5.073333333333333,
'140 260': 4.82,
'140 262': 0.8657692307692308,
'140 263': 0.9194117647058824,
'140 4': 3.825,
'140 43': 1.7,
'140 45': 5.0,
'140 48': 2.669166666666667,
'140 50': 3.0,
'140 52': 7.66,
'140 65': 7.12,
'140 66': 7.5,
'140 68': 4.1025,
'140 7': 4.96,
'140 74': 2.9725,
'140 75': 1.807142857142857,
'140 79': 4.2,
'140 83': 5.7,
'140 85': 11.32,
'140 87': 5.8933333333333335,
'140 88': 6.109999999999999,
'140 90': 3.685,
'140 95': 7.6,
'140 97': 9.0,
'141 100': 2.5149999999999997,
'141 107': 2.60125,
'141 112': 4.4,
'141 113': 3.75,
'141 114': 3.9,
'141 116': 6.41,
'141 13': 7.11,
'141 130': 13.8,
'141 132': 19.41,
'141 133': 11.54,
'141 137': 2.312307692307692,
'141 138': 9.285,
'141 140': 0.9331578947368421,
'141 141': 0.8211764705882354,
'141 142': 1.7133333333333332,
'141 143': 1.99,
'141 145': 2.9,
'141 148': 5.65,
'141 151': 3.125,
'141 158': 4.8,
'141 161': 1.5977777777777777,
'141 162': 1.142121212121212,
'141 163': 1.2077777777777778,

'141 164': 2.2333333333333334,
'141 166': 4.166666666666667,
'141 170': 1.7866666666666668,
'141 173': 6.45,
'141 178': 14.0,
'141 186': 2.7445454545454546,
'141 193': 2.72,
'141 196': 7.63,
'141 209': 6.66,
'141 211': 5.2,
'141 220': 10.23,
'141 224': 3.0,
'141 226': 2.48,
'141 229': 0.9359090909090909,
'141 230': 1.8555555555555554,
'141 231': 7.246666666666666,
'141 233': 1.3258333333333334,
'141 234': 2.982,
'141 236': 1.1402083333333333,
'141 237': 0.6136842105263158,
'141 238': 2.375,
'141 239': 2.0327272727272727,
'141 24': 3.9,
'141 243': 7.63,
'141 244': 7.68,
'141 246': 4.55,
'141 249': 5.66,
'141 255': 5.13,
'141 261': 6.885,
'141 262': 0.8171428571428571,
'141 263': 0.9044444444444445,
'141 4': 4.6675,
'141 42': 4.046666666666667,
'141 43': 1.2311111111111111,
'141 48': 2.526666666666667,
'141 50': 2.6775,
'141 65': 8.0,
'141 68': 3.1975,
'141 7': 3.5333333333333337,
'141 74': 3.1374999999999997,
'141 75': 1.90125,
'141 79': 3.9475,
'141 80': 5.5649999999999995,
'141 88': 7.26,
'141 90': 4.05,
'142 100': 1.6228571428571428,
'142 107': 3.2199999999999998,

'142 113': 3.2,
'142 114': 3.74,
'142 116': 4.556666666666667,
'142 125': 3.99,
'142 127': 8.965,
'142 129': 5.68,
'142 13': 5.058333333333334,
'142 132': 20.77,
'142 137': 2.985,
'142 138': 9.133333333333333,
'142 140': 2.2916666666666665,
'142 141': 1.7023076923076923,
'142 142': 0.628974358974359,
'142 143': 0.8415384615384615,
'142 144': 4.48,
'142 145': 3.6,
'142 148': 7.87,
'142 151': 2.0314285714285716,
'142 158': 2.7,
'142 161': 1.426875,
'142 162': 1.6821052631578948,
'142 163': 0.8309375,
'142 164': 2.382,
'142 166': 2.6875,
'142 17': 8.274999999999999,
'142 170': 2.3214285714285716,
'142 174': 12.6,
'142 181': 8.14,
'142 186': 1.8606666666666667,
'142 209': 7.3,
'142 211': 5.0,
'142 220': 9.0,
'142 223': 5.795,
'142 224': 3.885,
'142 225': 8.8,
'142 229': 1.6320000000000001,
'142 230': 1.0512121212121212121,
'142 231': 4.8425,
'142 233': 2.2944444444444443,
'142 234': 2.9166666666666665,
'142 236': 2.0282758620689654,
'142 237': 1.3573333333333333,
'142 238': 1.44875,
'142 239': 0.9964999999999999,
'142 24': 2.1580000000000004,
'142 243': 7.75,
'142 244': 6.058333333333334,

```
'142 246': 2.075714285714286,
'142 249': 2.982,
'142 261': 6.45,
'142 262': 2.686666666666667,
'142 263': 2.29,
'142 264': 0.4,
'142 41': 2.9244444444444446,
'142 42': 3.94,
'142 43': 1.1046153846153848,
'142 48': 0.9956756756756756,
'142 50': 1.0758333333333334,
'142 68': 1.8776470588235294,
'142 74': 3.8925,
…}
```

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 we 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

[36]:
```
# 1. Create a mean_distance column that is a copy of the pickup_dropoff helper␣
↪column
```

```python
df['mean_distance'] = df['pickup_dropoff']

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

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

[36]:

	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.

[37]:
```python
grouped = df.groupby('pickup_dropoff').mean(numeric_only=True)[['duration']]
grouped

# Create a dictionary where keys are unique pickup_dropoffs and values are
# mean trip duration for all trips with those pickup_dropoff combos
grouped_dict = grouped.to_dict()
grouped_dict = grouped_dict['duration']

df['mean_duration'] = df['pickup_dropoff']
df['mean_duration'] = df['mean_duration'].map(grouped_dict)

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

[37]:

	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.

[38]:
```
# Create 'day' col
df['day'] = df['tpep_pickup_datetime'].dt.day_name().str.lower()

# Create 'month' col
df['month'] = df['tpep_pickup_datetime'].dt.strftime('%b').str.lower()
```

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.

[40]:
```
# Create 'rush_hour' col
df['rush_hour'] = df['tpep_pickup_datetime'].dt.hour

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

[42]:
```
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
```

[43]:
```
# Apply the `rush_hourizer()` function to the new column
df.loc[(df.day!='saturday') & (df.day!='sunday'), 'rush_hour'] = df.
↪apply(rush_hourizer, axis=1)
df.head()
```

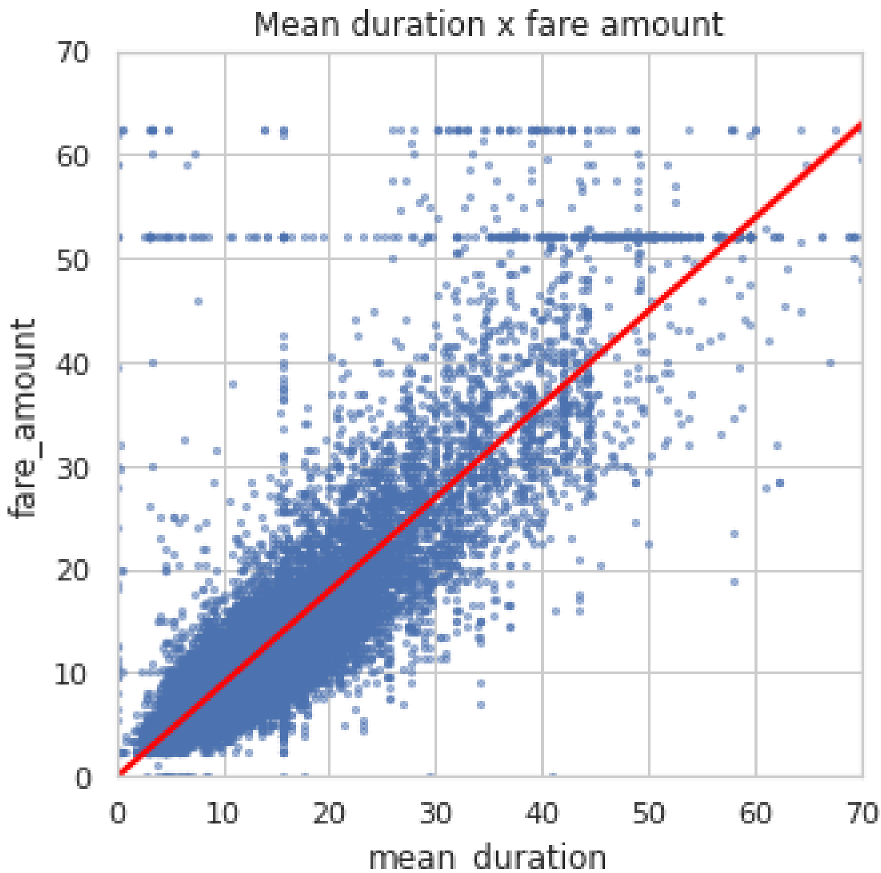
[43]:
```
   Unnamed: 0  VendorID tpep_pickup_datetime tpep_dropoff_datetime  \
0    24870114         2  2017-03-25 08:55:43   2017-03-25 09:09:47
1    35634249         1  2017-04-11 14:53:28   2017-04-11 15:19:58
2   106203690         1  2017-12-15 07:26:56   2017-12-15 07:34:08
3    38942136         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  …  tolls_amount  improvement_surcharge  \
```

```
0           100         231  …         0.0                     0.3
1           186          43  …         0.0                     0.3
2           262         236  …         0.0                     0.3
3           188          97  …         0.0                     0.3
4             4         112  …         0.0                     0.3

   total_amount   duration  pickup_dropoff  mean_distance  mean_duration  \
0         16.56  14.066667         100 231       3.521667      22.847222
1         20.80  26.500000          186 43       3.108889      24.470370
2          8.75   7.200000         262 236       0.881429       7.250000
3         27.69  30.250000          188 97       3.700000      30.250000
4         17.80  16.716667           4 112       4.435000      14.616667

        day  month rush_hour
0  saturday    mar         0
1   tuesday    apr         0
2    friday    dec         1
3    sunday    may         0
4  saturday    apr         0

[5 rows x 25 columns]
```

2.0.9 Task 4. Scatter plot

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

[45]:
```
# Create a scatterplot to visualize the relationship between variables of␣
↪interest
sns.set(style='whitegrid')
f = plt.figure()
f.set_figwidth(5)
f.set_figheight(5)
sns.regplot(x=df['mean_duration'], y=df['fare_amount'],
            scatter_kws={'alpha':0.5, 's':5},
            line_kws={'color':'red'})
plt.ylim(0, 70)
plt.xlim(0, 70)
plt.title('Mean duration x fare amount')
plt.show()
```



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?

We 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.

[46]:
```
df[df['fare_amount']>50]['fare_amount'].value_counts().head()
```

[46]:
```
52.0    514
62.5     84
59.0      9
50.5      9
57.5      8
Name: fare_amount, dtype: int64
```

Examine the first 30 of these trips.

[47]:
```python
# Set pandas to display all columns
pd.set_option('display.max_columns',None)
df[df['fare_amount']==52].head(30)
```

[47]:
```
      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	month	\
11	36.800000	236 132	19.211667	40.500000	sunday	mar	
110	66.850000	132 163	19.229000	52.941667	saturday	jun	
161	0.966667	132 132	2.255862	3.021839	saturday	nov	
247	29.183333	132 79	19.431667	47.275000	wednesday	dec	
379	29.483333	132 234	17.654000	49.833333	sunday	sep	
388	39.833333	132 48	18.761905	58.246032	tuesday	feb	
406	15.616667	228 88	4.730000	15.616667	monday	jun	
449	45.450000	132 48	18.761905	58.246032	thursday	aug	

468	42.850000	186 132	17.096000	42.920000	tuesday	sep
520	71.583333	132 148	17.994286	46.340476	sunday	apr
569	28.883333	132 144	18.537500	37.000000	wednesday	nov
572	0.216667	230 161	0.685484	7.965591	tuesday	jul
586	55.700000	211 132	16.580000	61.691667	monday	jun
692	30.533333	132 170	17.203000	37.113333	tuesday	nov
717	34.033333	132 239	20.901250	44.862500	wednesday	dec
719	57.366667	264 264	3.191516	15.618773	friday	aug
782	52.750000	163 132	17.275833	52.338889	friday	jun
816	48.600000	132 170	17.203000	37.113333	tuesday	feb
818	88.783333	132 246	18.515000	66.316667	tuesday	jun
835	36.983333	132 48	18.761905	58.246032	tuesday	jan
840	45.066667	132 163	19.229000	52.941667	friday	oct
861	28.000000	75 132	18.442500	36.204167	saturday	dec
881	36.000000	68 132	18.785000	58.041667	saturday	dec
958	35.166667	132 261	22.115000	51.493750	sunday	oct
970	33.783333	132 140	19.293333	36.791667	friday	feb
984	55.050000	132 230	18.571200	59.598000	wednesday	aug
1082	14.366667	170 48	1.265789	14.135965	tuesday	feb
1097	2.333333	265 265	0.753077	3.411538	monday	aug
1110	58.400000	239 132	19.795000	50.562500	wednesday	sep
1179	40.666667	238 132	19.470000	53.861111	monday	jun

	rush_hour
11	0
110	0
161	0
247	0
379	0
388	1
406	0
449	0
468	0
520	0
569	0
572	0
586	0
692	0
717	0
719	1
782	1
816	1
818	1
835	0
840	0
861	0
881	0

```
958            0
970            0
984            1
1082           1
1097           0
1110           0
1179           1
```

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.

2.0.10 Task 5. Isolate modeling variables

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

[48]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
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)
memory usage: 4.3+ MB
```

[49]:

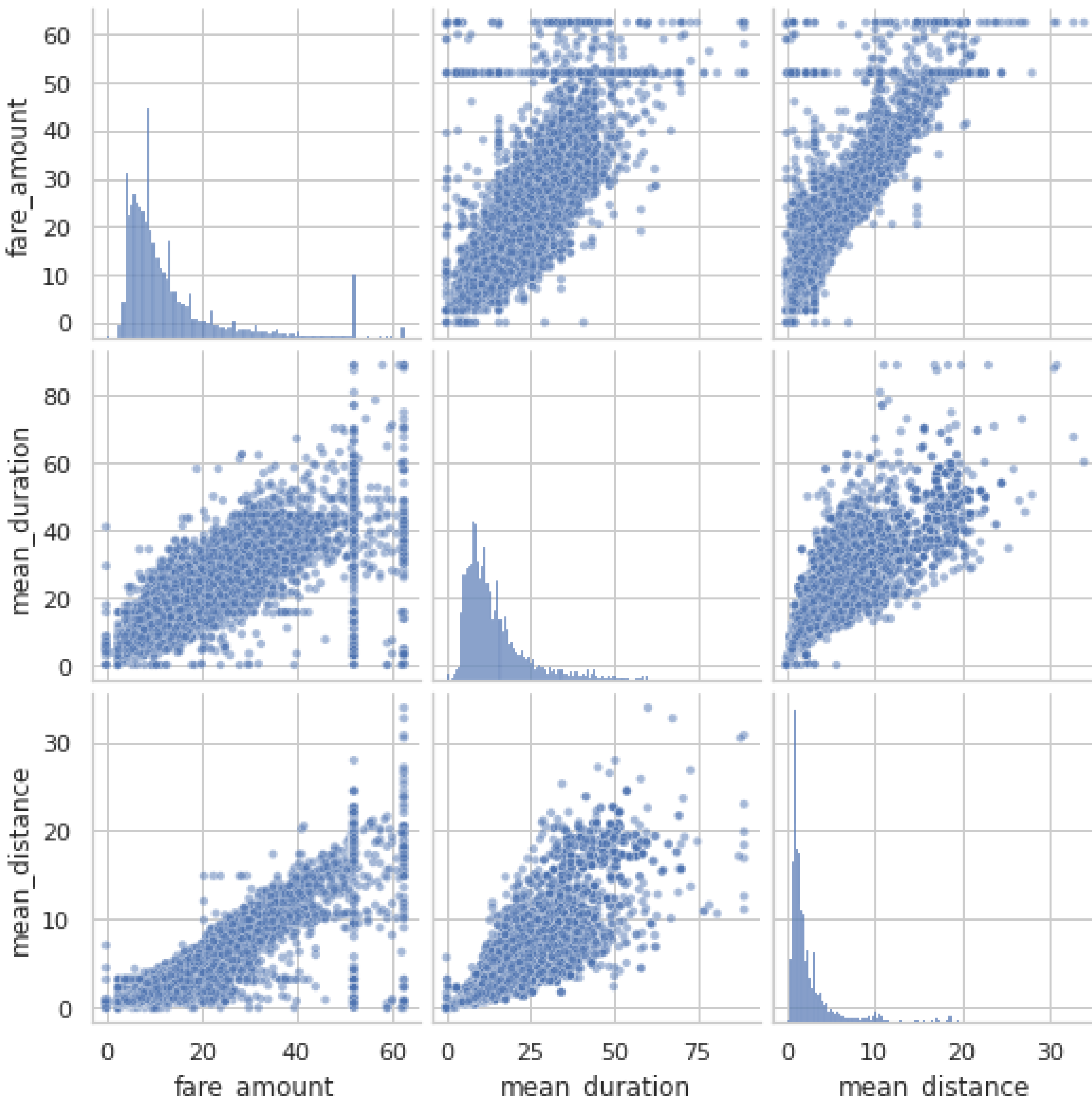
```
df2 = df.copy()

df2 = df2.drop(['Unnamed: 0', 'tpep_dropoff_datetime', 'tpep_pickup_datetime',
               'trip_distance', 'RatecodeID', 'store_and_fwd_flag',␣
 ↪'PULocationID', 'DOLocationID',
               'payment_type', 'extra', 'mta_tax', 'tip_amount',␣
 ↪'tolls_amount', 'improvement_surcharge',
               'total_amount', 'tpep_dropoff_datetime', 'tpep_pickup_datetime',␣
 ↪'duration',
               'pickup_dropoff', 'day', 'month'
               ], axis=1)

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
```

2.0.11 Task 6. Pair plot

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

[50]:
```
# Create a pairplot to visualize pairwise relationships between variables in␣
↪the data
sns.pairplot(df2[['fare_amount','mean_duration','mean_distance']],
             plot_kws={'alpha':0.5,'size':5},
             );
```



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

2.0.12 Task 7. Identify correlations

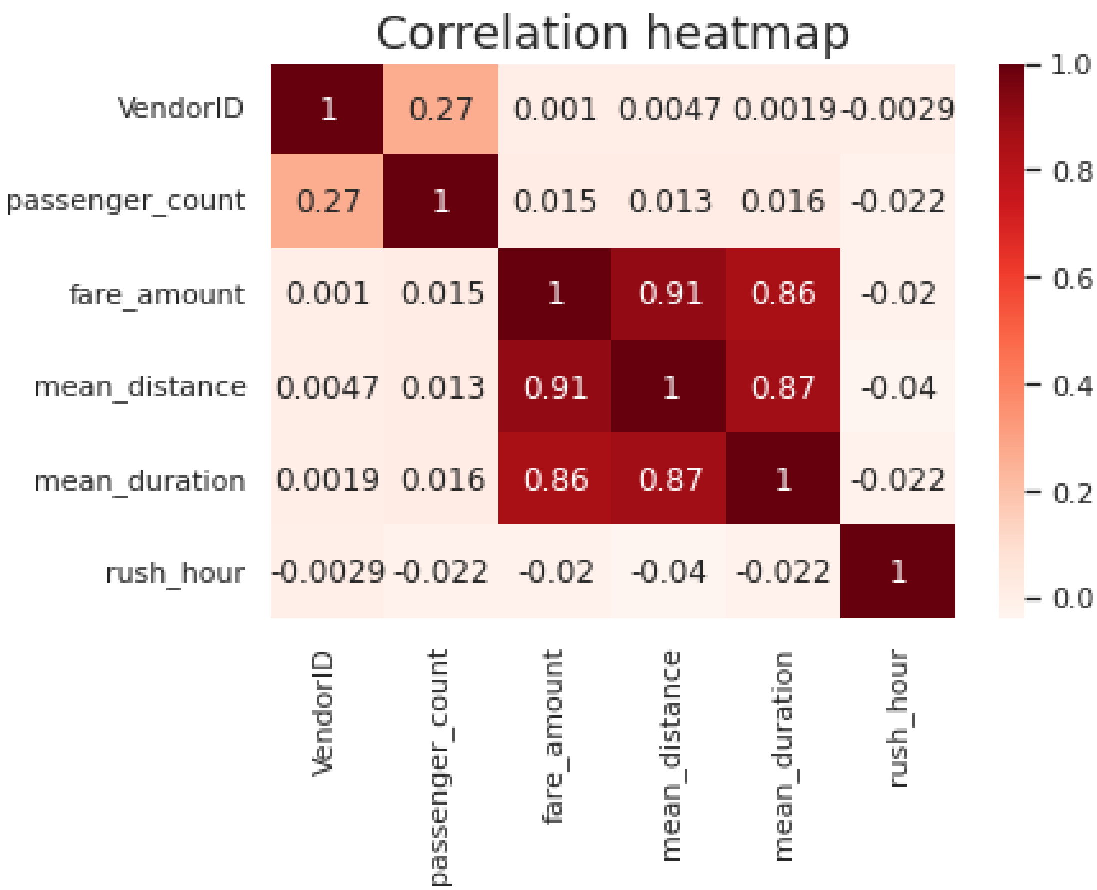
Next, code a correlation matrix to help determine most correlated variables.

[52]:
```
# Correlation matrix to help determine most correlated variables
df2.corr(method='pearson')
```

[52]:
```
                 VendorID  passenger_count  fare_amount  mean_distance  \
VendorID         1.000000         0.266463     0.001045       0.004741
passenger_count  0.266463         1.000000     0.014942       0.013428
fare_amount      0.001045         0.014942     1.000000       0.910185
mean_distance    0.004741         0.013428     0.910185       1.000000
mean_duration    0.001876         0.015852     0.859105       0.874864
rush_hour       -0.002874        -0.022035    -0.020075      -0.039725

                 mean_duration  rush_hour
VendorID              0.001876  -0.002874
passenger_count       0.015852  -0.022035
fare_amount           0.859105  -0.020075
mean_distance         0.874864  -0.039725
mean_duration         1.000000  -0.021583
rush_hour            -0.021583   1.000000
```

Visualize a correlation heatmap of the data.

[53]:
```
# Create correlation heatmap
plt.figure(figsize=(6,4))
sns.heatmap(df2.corr(method='pearson'), annot=True, cmap='Reds')
plt.title('Correlation heatmap',
          fontsize=18)
plt.show()
```



`mean_duration` and `mean_distance` are both highly correlated with the target variable of `fare_amount` They're also both correlated with each other, with a Pearson correlation of 0.87.

Highly correlated predictor variables can be bad for linear regression models when we want to be able to draw statistical inferences about the data from the model. However, correlated predictor variables can still be used to create an accurate predictor if the prediction itself is more important than using the model as a tool to learn about your data.

This model will predict `fare_amount`, which will be used as a predictor variable in machine learning models. Therefore, try modeling with both variables even though they are correlated.

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

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

[54]:
```
# Remove the target column from the features
X = df2.drop(columns='fare_amount')


# Set y variable
```

```
y = df2[['fare_amount']]
```

2.0.14 Task 8b. Pre-process data

Dummy encode categorical variables

[57]:
```
# Convert VendorID to string
X['VendorID'] = X['VendorID'].astype(str)

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

[57]:

	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

2.0.15 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`.

[58]:
```
# Create training and testing sets
x_train,x_test,y_train,y_test = train_test_split(X,y, test_size=0.2,␣
↪random_state=0)
```

2.0.16 Standardize the data

Use `StandardScaler()`, `fit()`, and `transform()` to standardize the `X_train` variables. Assign the results to a variable called `X_train_scaled`.

[59]:
```
# Standardize the X variables
scaler = StandardScaler().fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_train_scaled
```

[59]:
```
array([[-0.50301524,  0.8694684 ,  0.17616665, -0.64893329,  0.89286563],
       [-0.50301524, -0.60011281, -0.69829589,  1.54099045,  0.89286563],
       [ 0.27331093, -0.47829156, -0.57301906, -0.64893329, -1.11998936],
       …,
       [-0.50301524, -0.45121122, -0.6788917 , -0.64893329, -1.11998936],
       [-0.50301524, -0.58944763, -0.85743597,  1.54099045, -1.11998936],
```

```
        [ 1.82596329,  0.83673851,  1.13212101, -0.64893329,  0.89286563]])
```

2.0.17 Fit the model

Instantiate your model and fit it to the training data.

[60]:
```
# Fit your model to the training data
lr = LinearRegression()
lr.fit(x_train_scaled,y_train)
```

[60]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

2.0.18 Task 8c. Evaluate model

2.0.19 Train data

Evaluate the 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.

[62]:
```
# Evaluate the model performance on the training data
r_sq = lr.score(x_train_scaled, y_train)
print('Coefficient of determination:', r_sq)
y_pred_train = lr.predict(x_train_scaled)
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
```

2.0.20 Test data

Calculate the same metrics on the test data. Remember to scale the `X_test` data using the scaler that was fit to the training data. Do not refit the scaler to the testing data, just transform it. Call the results `X_test_scaled`.

[64]:
```
# Scale the X_test data
X_test_scaled = scaler.transform(x_test)
```

[65]:
```python
# Evaluate the model performance on the testing data
r_sq_test = lr.score(X_test_scaled, y_test)
print('Coefficient of determination:', r_sq_test)
y_pred_test = lr.predict(X_test_scaled)
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: 2.1336549840593864
MSE: 14.326454156998944
RMSE: 3.785030271609323
```

2.0.21 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.

[66]:
```python
# Create a `results` dataframe
results = pd.DataFrame(data={'actual': y_test['fare_amount'],
                             'predicted': y_pred_test.ravel()})
results['residual'] = results['actual'] - results['predicted']
results.head()
```

[66]:

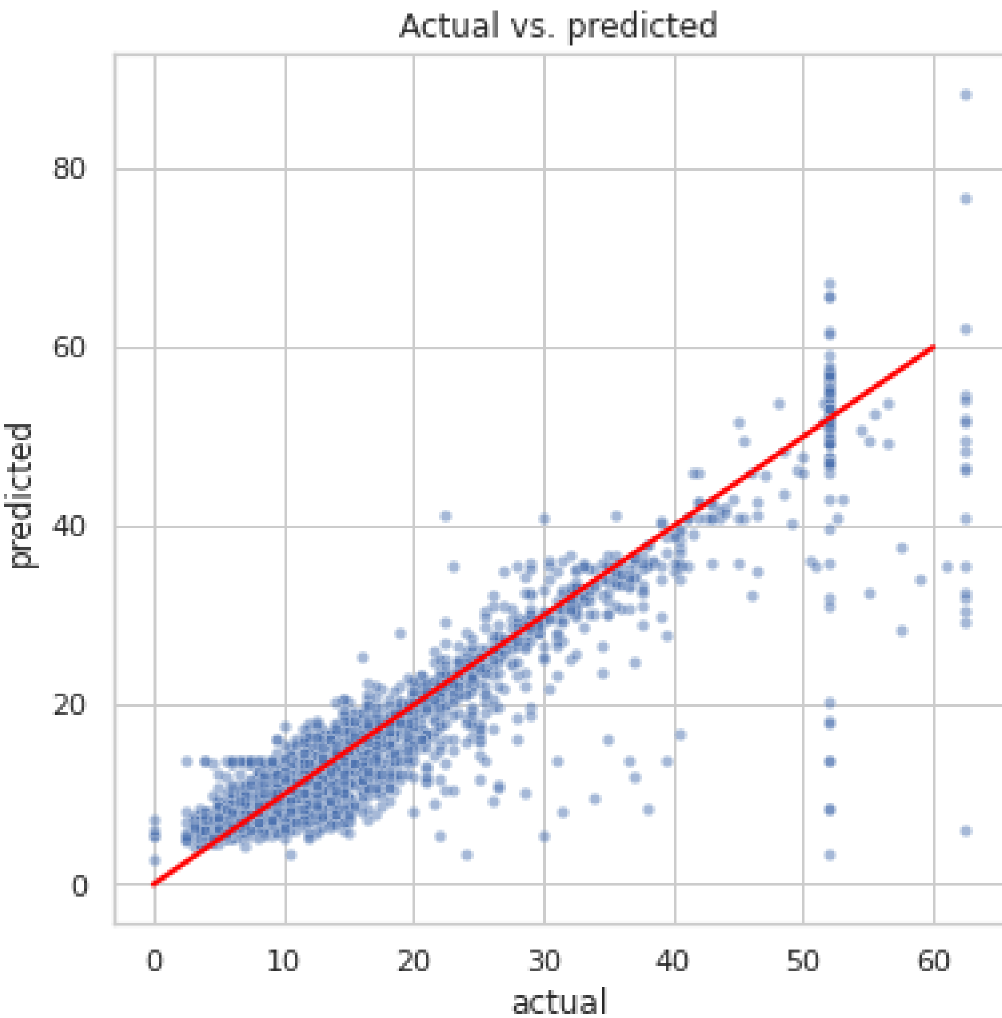
	actual	predicted	residual
5818	14.0	12.356503	1.643497
18134	28.0	16.314595	11.685405
4655	5.5	6.726789	-1.226789
7378	15.5	16.227206	-0.727206
13914	9.5	10.536408	-1.036408

2.0.22 Task 9b. Visualize model results

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

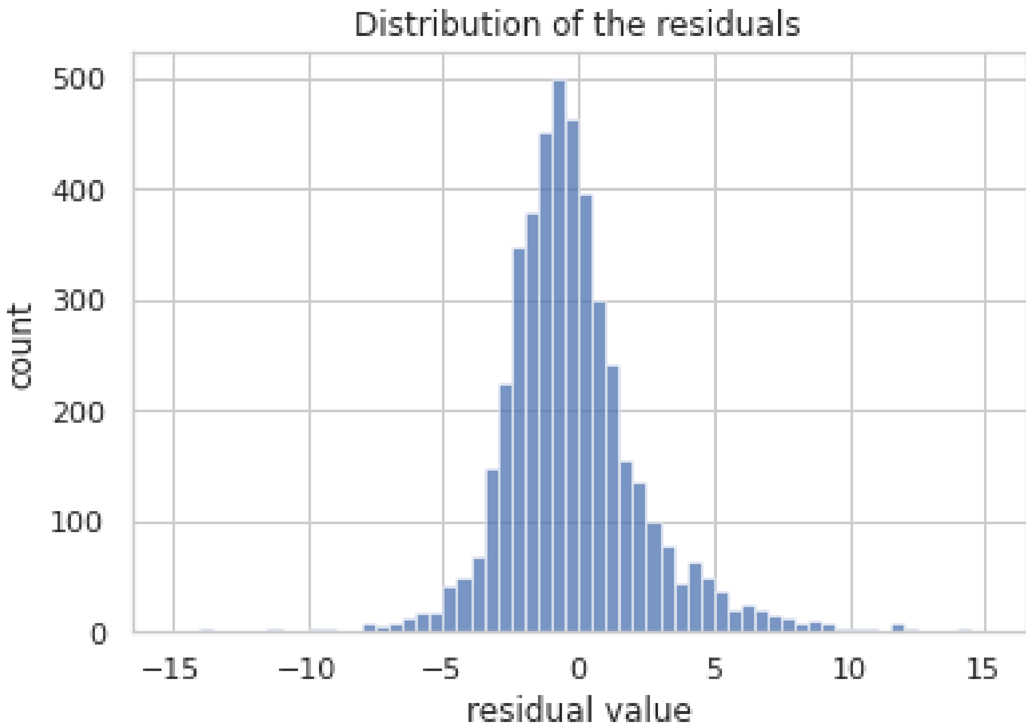
[67]:
```python
# Create a scatterplot to visualize `predicted` over `actual`
fig, ax = plt.subplots(figsize=(6, 6))
sns.set(style='whitegrid')
sns.scatterplot(x='actual',
                y='predicted',
                data=results,
                s=20,
                alpha=0.5,
                ax=ax
```

```
)
# Draw an x=y line to show what the results would be if the model were perfect
plt.plot([0,60], [0,60], c='red', linewidth=2)
plt.title('Actual vs. predicted');
```



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

[68]:
```
# Visualize the distribution of the `residuals`
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');
```

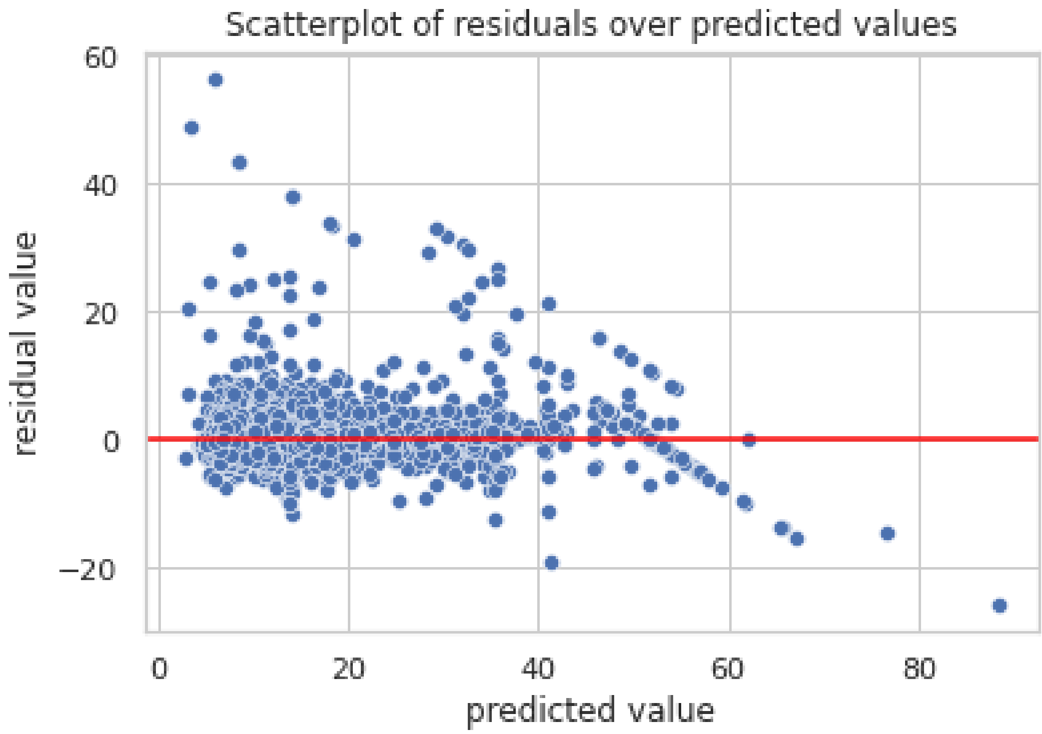


[70]:
```
# Calculate residual mean
results['residual'].mean()
```

[70]: -0.01544262152868053

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

[69]:
```
# Create a scatterplot of `residuals` over `predicted`
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()
```



2.0.23 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?

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

The coefficients reveal that `mean_distance` was the feature with the greatest weight in the model's final prediction. Be careful here! A common misinterpretation is that for every mile traveled, the fare amount increases by a mean of \$7.13. This is incorrect. Remember, the data used to train the model was standardized with `StandardScaler()`. As such, the units are no longer miles. In other words, we cannot say "for every mile traveled…", as stated above. The correct interpretation of this coefficient is: controlling for other variables, *for every +1 change in standard deviation*, the fare amount increases by a mean of \$7.13.

Note also that because some highly correlated features were not removed, the confidence interval of this assessment is wider.

So, translate this back to miles instead of standard deviation (i.e., unscale the data).

1. Calculate the standard deviation of `mean_distance` in the `X_train` data.
2. Divide the coefficient (7.133867) by the result to yield a more intuitive interpretation.

[ ]:
```python
# 1. Calculate SD 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())
```

Now we can make a more intuitive interpretation: for every 3.57 miles traveled, the fare increased by a mean of \$7.13. Or, reduced: for every 1 mile traveled, the fare increased by a mean of \$2.00.