

Activity_ Course 5 Automatidata project lab

December 8, 2025

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 building 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 demonstrate 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.

```
[34]: # Imports
# Packages for numerics + dataframes
import pandas as pd
import numpy as np

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

# 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 # For confusion matrix
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,r2_score,mean_squared_error
```

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]: # Load dataset into dataframe
df0=pd.read_csv("2017_Yellow_Taxi_Trip_Data.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?

EDA has the following purposes: - 1. Identifying outliers and investigating them. 2. Finding any missing values which can lead unbalanced data. 3. Checking for any Multicollinearity among key variables i.e. checking their distributions. 4. Feature engineering of some variables.

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]: # Start with `shape` and `info()`
df0.shape
```

```
[5]: (22699, 18)
```

```
[7]: df0.info()
```

```
<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()`.

```
[8]: # Check for missing data and duplicates using .isna() and .drop_duplicates()
df0.isna()
```

```
[8]:      Unnamed: 0  VendorID  tpep_pickup_datetime  tpep_dropoff_datetime  \
0        False    False             False             False
1        False    False             False             False
2        False    False             False             False
3        False    False             False             False
4        False    False             False             False
...
22694     ...    ...             ...             ...
22695     ...    ...             ...             ...
22696     ...    ...             ...             ...
22697     ...    ...             ...             ...
22698     ...    ...             ...             ...

      passenger_count  trip_distance  RatecodeID  store_and_fwd_flag  \
0        False        False        False        False
1        False        False        False        False
2        False        False        False        False
3        False        False        False        False
4        False        False        False        False
...
22694     ...    ...             ...             ...
22695     ...    ...             ...             ...
22696     ...    ...             ...             ...
22697     ...    ...             ...             ...
22698     ...    ...             ...             ...

      PULocationID  DOLocationID  payment_type  fare_amount  extra  mta_tax  \
0        False        False        False        False  False  False
1        False        False        False        False  False  False
2        False        False        False        False  False  False
3        False        False        False        False  False  False
```

4	False	False	False	False	False	False
...
22694	False	False	False	False	False	False
22695	False	False	False	False	False	False
22696	False	False	False	False	False	False
22697	False	False	False	False	False	False
22698	False	False	False	False	False	False
tip_amount	tolls_amount	improvement_surcharge	total_amount			
0	False	False	False	False	False	
1	False	False	False	False	False	
2	False	False	False	False	False	
3	False	False	False	False	False	
4	False	False	False	False	False	
...	
22694	False	False	False	False	False	
22695	False	False	False	False	False	
22696	False	False	False	False	False	
22697	False	False	False	False	False	
22698	False	False	False	False	False	

[22699 rows x 18 columns]

```
[9]: df0.drop_duplicates()
```

```
[9]:      Unnamed: 0  VendorID      tpep_pickup_datetime      tpep_dropoff_datetime
0          24870114        2  03/25/2017 8:55:43 AM  03/25/2017 9:09:47 AM
1          35634249        1  04/11/2017 2:53:28 PM  04/11/2017 3:19:58 PM
2          106203690       1  12/15/2017 7:26:56 AM  12/15/2017 7:34:08 AM
3          38942136        2  05/07/2017 1:17:59 PM  05/07/2017 1:48:14 PM
4          30841670        2  04/15/2017 11:32:20 PM 04/15/2017 11:49:03 PM
...
          ...      ...
22694     14873857        2  02/24/2017 5:37:23 PM  02/24/2017 5:40:39 PM
22695     66632549        2  08/06/2017 4:43:59 PM  08/06/2017 5:24:47 PM
22696     74239933        2  09/04/2017 2:54:14 PM  09/04/2017 2:58:22 PM
22697     60217333        2  07/15/2017 12:56:30 PM 07/15/2017 1:08:26 PM
22698     17208911        1  03/02/2017 1:02:49 PM  03/02/2017 1:16:09 PM

      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
...
          ...      ...
22694     3           0.61         1            N
22695     1          16.71         2            N
```

```

22696           1      0.42          1          N
22697           1      2.36          1          N
22698           1      2.10          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
...
22694           ...      186           2        4.0    1.0    0.5
22695           132        164           1       52.0    0.0    0.5
22696           107        234           2        4.5    0.0    0.5
22697           68         144           1       10.5    0.0    0.5
22698           239        236           1       11.0    0.0    0.5

      tip_amount  tolls_amount  improvement_surcharge  total_amount
0            2.76        0.00            0.3       16.56
1            4.00        0.00            0.3       20.80
2            1.45        0.00            0.3        8.75
3            6.39        0.00            0.3       27.69
4            0.00        0.00            0.3       17.80
...
22694           ...      0.00            0.3        5.80
22695           14.64        5.76            0.3       73.20
22696           0.00        0.00            0.3        5.30
22697           1.70        0.00            0.3       13.00
22698           2.35        0.00            0.3       14.15

```

[22699 rows x 18 columns]

Use `.describe()`.

```
[10]: # Use .describe()
df0.describe()
```

```
[10]:      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

```
[20]: # Check the format of the data
df0['tpep_pickup_datetime'] = pd.to_datetime(df0['tpep_pickup_datetime'])
```

```
[21]: # Convert datetime columns to datetime
df0['tpep_dropoff_datetime'] = pd.to_datetime(df0['tpep_dropoff_datetime'])
```

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.

```
[165]: # Create `duration` column
df0['duration'] = df0['tpep_dropoff_datetime'] - df0['tpep_pickup_datetime']
```

```
df0['duration_in_seconds'] = df0['duration'].dt.total_seconds()
```

4.2.4 Outliers

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

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

```
df0.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   timedelta64[ns]
 19  duration_in_seconds 22699 non-null   float64 
 20  pickup_dropoff      22699 non-null   object  
 21  mean_distance        22699 non-null   float64 
 22  mean_duration        22699 non-null   float64 
 23  day                 22699 non-null   object  
 24  month                22699 non-null   object  
 25  rush_hour            22699 non-null   int64  
dtypes: datetime64[ns](2), float64(11), int64(8), object(4), timedelta64[ns](1)
memory usage: 4.5+ 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`

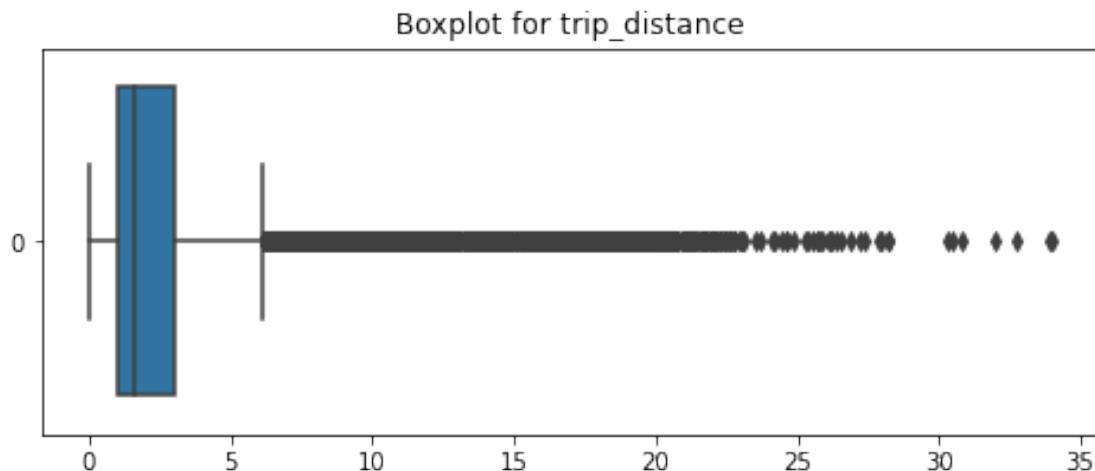
4.2.5 Task 2d. Box plots

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

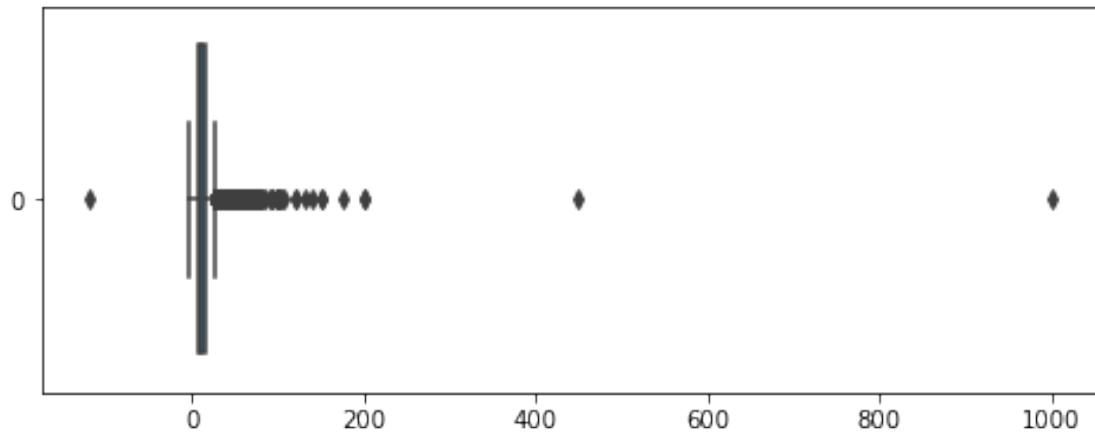
```
[47]: #Boxplot for trip duration
plt.figure(figsize = (8,3))
sns.boxplot(data=df0['trip_distance'],orient='h')
plt.title('Boxplot for trip_distance')
plt.show()

#Boxplot for fare amount
plt.figure(figsize = (8,3))
sns.boxplot(data=df0['fare_amount'],orient='h')
plt.title('Boxplot for fare_amount')
plt.show()

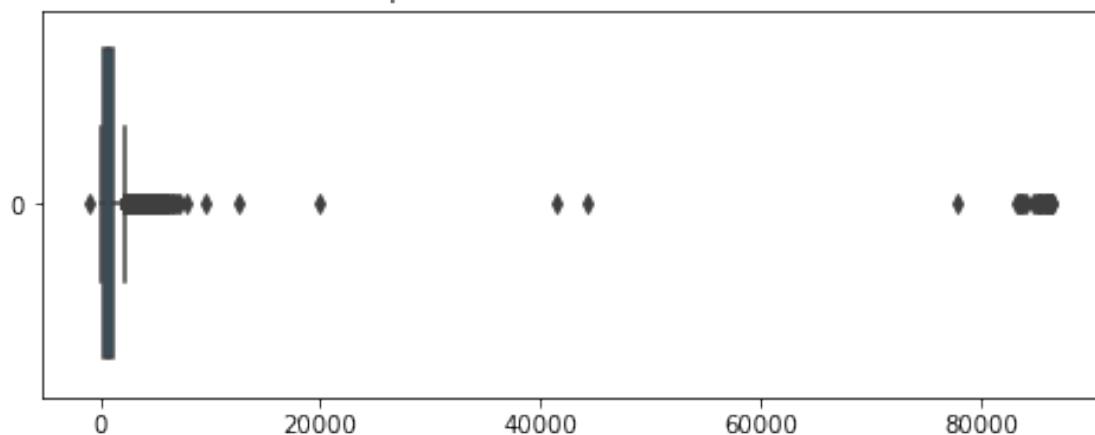
#Boxplot for duration (Duration in seconds is used because the duration column
#is a series object)
plt.figure(figsize = (8,3))
sns.boxplot(data=df0['duration_in_seconds'],orient='h')
plt.title('Boxplot for duration (in seconds)')
plt.show()
```



Boxplot for fare_amount



Boxplot for duration (in seconds)



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 of the above.
 2. There maybe some based on map based calculated distances.
 3. No they should be resetted

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?

```
[48]: # Are trip distances of 0 bad data or very short trips rounded down  
sorted(set(df0['trip_distance']))[:10]
```

```
[48]: [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.

```
[50]: sum(df0['trip_distance']==0)
```

```
[50]: 148
```

`fare_amount` outliers

```
[52]: df0['fare_amount'].describe()
```

```
[52]: 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?

Impute values less than \$0 with 0.

```
[79]: # Impute values less than $0 with 0  
#mask = df0['fare_amount']<0  
#df[mask]  
  
df0.loc[df0['fare_amount'] < 0, 'fare_amount'] = 0  
df0['fare_amount'].min()
```

```
[79]: 0.0
```

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

```
[90]: def maximum_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.

...

```
for col in column_list:  
    # Reassign minimum to zero  
    df0.loc[df0[col] < 0, col] = 0  
  
    # Calculate upper threshold  
    q1 = df0[col].quantile(0.25)  
    q3 = df0[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  
    df0.loc[df0[col] > upper_threshold, col] = upper_threshold  
    print(df0[col].describe())  
    print()
```

[91]: maximum_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

```
[94]: # Call .describe() for duration outliers  
df0['duration_in_seconds'].describe()
```

```
[94]: count    22699.000000  
mean      1020.826600  
std       3719.788923  
min     -1019.000000  
25%      399.000000  
50%      671.000000  
75%     1103.000000  
max     86373.000000  
Name: duration_in_seconds, 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: $Q3 + (6 * IQR)$.

```
[95]: # Impute a 0 for any negative values  
df0.loc[df0['duration_in_seconds'] < 0, 'duration_in_seconds'] = 0  
df0['duration_in_seconds'].min()
```

```
[95]: 0.0
```

```
[96]: # Impute the high outliers  
maximum_imputer(['duration_in_seconds'],6)
```

```
duration_in_seconds  
q3: 1103.0  
upper_threshold: 5327.0  
count    22699.000000  
mean      867.633288  
std       716.822559  
min      0.000000  
25%      399.000000  
50%      671.000000  
75%     1103.000000  
max     5327.000000  
Name: duration_in_seconds, 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'

```
[97]: # Create `pickup_dropoff` column  
df0['pickup_dropoff'] = df0['PULocationID'].astype(str) + ' ' + df0['DOLocationID'].astype(str)
```

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`.

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

```
grouped = df0.groupby('pickup_dropoff').  
    ↪mean(numeric_only=True)[['trip_distance']]
```

`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}
```

```
[101]: # 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']
```

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.

```
[102]: # 1. Create a mean_distance column that is a copy of the pickup_dropoff helper
      ↪column
df0['mean_distance'] = df0['pickup_dropoff']

# 2. Map `grouped_dict` to the `mean_distance` column
df0['mean_distance'] = df0['mean_distance'].map(grouped_dict)

# Confirm that it worked
df0[(df0['PULocationID']==100) & (df0['DOLocationID']==231)][['mean_distance']]
```

```
[102]:      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.

```
[185]: grouped = df0.groupby('pickup_dropoff').
      ↪mean(numeric_only=True)[['duration_in_seconds']]
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_in_seconds']

df0['mean_duration'] = df0['pickup_dropoff']
df0['mean_duration'] = df0['mean_duration'].map(grouped_dict)

# Confirm that it worked
```

```
df0[(df0['PULocationID']==100) & (df0['DOLocationID']==231)][['mean_duration']]
```

```
[185]:      mean_duration
0          1370.833333
4909       1370.833333
16636       1370.833333
18134       1370.833333
19761       1370.833333
20581       1370.833333
```

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.

```
[140]: # Create 'day' col
df0['day'] = df0['tpep_pickup_datetime'].dt.day_name().str.lower()

# Create 'month' col
df0['month'] = df0['tpep_pickup_datetime'].dt.strftime('%b').str.lower()

df0
```

```
[140]:      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
...
22694       ...        ...
22695       ...        ...
22696       ...        ...
22697       ...        ...
22698       ...        ...

      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
...
22694       ...        ...
22695       ...        ...
22696       ...        ...
22697       ...        ...
22698       ...        ...
```

	PULocationID	DOLocationID	...	tolls_amount	improvement_surcharge	\
0	100	231	...	0.00	0.3	
1	186	43	...	0.00	0.3	
2	262	236	...	0.00	0.3	
3	188	97	...	0.00	0.3	
4	4	112	...	0.00	0.3	
...	
22694	48	186	...	0.00	0.3	
22695	132	164	...	5.76	0.3	
22696	107	234	...	0.00	0.3	
22697	68	144	...	0.00	0.3	
22698	239	236	...	0.00	0.3	
	total_amount	duration	duration_in_seconds	pickup_dropoff		\
0	16.56	0 days 00:14:04	844.0	100 231		
1	20.80	0 days 00:26:30	1590.0	186 43		
2	8.75	0 days 00:07:12	432.0	262 236		
3	27.69	0 days 00:30:15	1815.0	188 97		
4	17.80	0 days 00:16:43	1003.0	4 112		
...	
22694	5.80	0 days 00:03:16	196.0	48 186		
22695	73.20	0 days 00:40:48	2448.0	132 164		
22696	5.30	0 days 00:04:08	248.0	107 234		
22697	13.00	0 days 00:11:56	716.0	68 144		
22698	14.15	0 days 00:13:20	800.0	239 236		
	mean_distance	mean_duration	day	month		
0	3.521667	1370.833333	saturday	mar		
1	3.108889	1468.222222	tuesday	apr		
2	0.881429	435.000000	friday	dec		
3	3.700000	1815.000000	sunday	may		
4	4.435000	877.000000	saturday	apr		
...		
22694	1.098214	515.678571	friday	feb		
22695	18.757500	3573.625000	sunday	aug		
22696	0.684242	396.545455	monday	sep		
22697	2.077500	999.000000	saturday	jul		
22698	1.476970	564.333333	thursday	mar		

[22699 rows x 25 columns]

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.

```
[143]: # Create 'rush_hour' col
df0['rush_hour'] = df0['tpep_pickup_datetime'].dt.hour

# If day is Saturday or Sunday, impute 0 in `rush_hour` column
df0.loc[df0['day'].isin(['saturday', 'sunday']), 'rush_hour'] = 0
```

```
[144]: 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
```

```
[145]: # Apply the `rush_hourizer()` function to the new column
df0.loc[(df0.day != 'saturday') & (df0.day != 'sunday'), 'rush_hour'] = df0.
    ↪apply(rush_hourizer, axis=1)
df0.head()
```

```
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  duration_in_seconds  pickup_dropoff  mean_distance \
0 0 days 00:14:04                  844.0        100 231      3.521667
1 0 days 00:26:30                  1590.0        186 43      3.108889
2 0 days 00:07:12                   432.0        262 236      0.881429
3 0 days 00:30:15                   1815.0        188 97      3.700000
4 0 days 00:16:43                  1003.0         4 112      4.435000
```

```

mean_duration      day month  rush_hour
0    1370.833333  saturday  mar        0
1    1468.222222  tuesday   apr        0
2    435.000000   friday    dec        1
3    1815.000000  sunday    may        0
4    877.000000   saturday  apr        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`.

```
[210]: df0['mean_duration'] = df0['mean_duration']/60
df0.head()
```

```

[210]: 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 ... total_amount duration \
0            100          231 ...  16.56 0 days 00:14:04
1            186          43  ...  20.80 0 days 00:26:30
2            262          236 ...   8.75 0 days 00:07:12
3            188          97  ...  27.69 0 days 00:30:15
4              4          112 ...  17.80 0 days 00:16:43

duration_in_seconds  pickup_dropoff  mean_distance  mean_duration \
0            844.0          100 231       3.521667    22.847222
1            1590.0          186 43       3.108889    24.470370
2            432.0          262 236       0.881429     7.250000
3            1815.0          188 97       3.700000    30.250000
4            1003.0          4 112       4.435000    14.616667

day  month rush_hour  mean_duration_new
```

```

0  saturday    mar      0      0.006346
1  tuesday     apr      0      0.006797
2  friday      dec      1      0.002014
3  sunday      may      0      0.008403
4  saturday    apr      0      0.004060

```

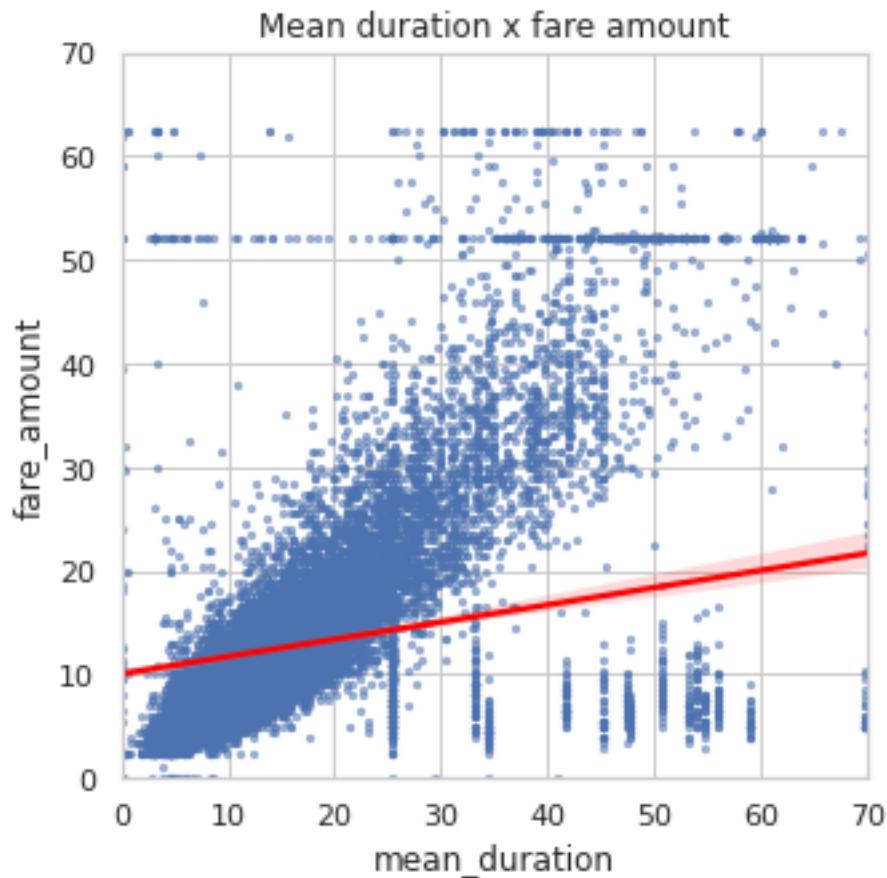
[5 rows x 27 columns]

```
[212]: # Create a scatterplot to visualize the relationship between variables of interest
```

```

sns.set(style='whitegrid')
plt.figure(figsize = (5,5))
sns.regplot(x=df0['mean_duration'], y=df0['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?

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.

```
[217]: df0[df0['fare_amount'] > 52]['fare_amount'].value_counts().head()
```

```
[217]: 62.5      84
       59.0      9
       57.5      8
       60.0      6
       55.0      6
Name: fare_amount, dtype: int64
```

Examine the first 30 of these trips.

```
[241]: # Set pandas to display all columns
pd.set_option('display.max_columns', None)
df0[df0['fare_amount'] == 52].head(30)
```

```
[241]:      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       659000029         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	duration_in_seconds	pickup_dropoff	mean_distance	\
11	0 days	00:36:48	2208.0	236 132	19.211667	
110	0 days	01:06:51	4011.0	132 163	19.229000	
161	0 days	00:00:58	58.0	132 132	2.255862	
247	0 days	00:29:11	1751.0	132 79	19.431667	
379	0 days	00:29:29	1769.0	132 234	17.654000	
388	0 days	00:39:50	2390.0	132 48	18.761905	
406	0 days	00:15:37	937.0	228 88	4.730000	
449	0 days	00:45:27	2727.0	132 48	18.761905	
468	0 days	00:42:51	2571.0	186 132	17.096000	
520	0 days	01:11:35	4295.0	132 148	17.994286	
569	0 days	00:28:53	1733.0	132 144	18.537500	
572	0 days	00:00:13	13.0	230 161	0.685484	
586	0 days	00:55:42	3342.0	211 132	16.580000	
692	0 days	00:30:32	1832.0	132 170	17.203000	
717	0 days	00:34:02	2042.0	132 239	20.901250	
719	0 days	00:57:22	3442.0	264 264	3.191516	
782	0 days	00:52:45	3165.0	163 132	17.275833	
816	0 days	00:48:36	2916.0	132 170	17.203000	
818	0 days	02:09:19	7759.0	132 246	18.515000	
835	0 days	00:36:59	2219.0	132 48	18.761905	
840	0 days	00:45:04	2704.0	132 163	19.229000	
861	0 days	00:28:00	1680.0	75 132	18.442500	
881	0 days	00:36:00	2160.0	68 132	18.785000	
958	0 days	00:35:10	2110.0	132 261	22.115000	
970	0 days	00:33:47	2027.0	132 140	19.293333	
984	0 days	00:55:03	3303.0	132 230	18.571200	
1082	0 days	00:14:22	862.0	170 48	1.265789	
1097	0 days	00:02:20	140.0	265 265	0.753077	
1110	0 days	00:58:24	3504.0	239 132	19.795000	
1179	0 days	00:40:40	2440.0	238 132	19.470000	

	mean_duration	day	month	rush_hour	mean_duration_new
11	265.147222	sunday	mar	0	0.073652
110	52.941667	saturday	jun	0	0.014706
161	3.021839	saturday	nov	0	0.000839

247	47.275000	wednesday	dec	0	0.013132
379	49.833333	sunday	sep	0	0.013843
388	61.604762	tuesday	feb	1	0.017112
406	15.616667	monday	jun	0	0.004338
449	61.604762	thursday	aug	0	0.017112
468	42.920000	tuesday	sep	0	0.011922
520	46.340476	sunday	apr	0	0.012872
569	37.000000	wednesday	nov	0	0.010278
572	7.965591	tuesday	jul	0	0.002213
586	61.691667	monday	jun	0	0.017137
692	37.113333	tuesday	nov	0	0.010309
717	44.862500	wednesday	dec	0	0.012462
719	25.329964	friday	aug	1	0.007036
782	164.759722	friday	jun	1	0.045767
816	37.113333	tuesday	feb	1	0.010309
818	86.583333	tuesday	jun	1	0.024051
835	61.604762	tuesday	jan	0	0.017112
840	52.941667	friday	oct	0	0.014706
861	36.204167	saturday	dec	0	0.010057
881	63.737500	saturday	dec	0	0.017705
958	51.493750	sunday	oct	0	0.014304
970	36.791667	friday	feb	0	0.010220
984	60.800000	wednesday	aug	1	0.016889
1082	14.135965	tuesday	feb	1	0.003927
1097	3.411538	monday	aug	0	0.000948
1110	50.562500	wednesday	sep	0	0.014045
1179	53.861111	monday	jun	1	0.014961

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

All of them have a fare amount of 52.

4.2.9 Task 5. Isolate modeling variables

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

[219]: df0.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 27 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  timedelta64[ns]
19  duration_in_seconds         22699 non-null  float64
20  pickup_dropoff                22699 non-null  object
21  mean_distance                  22699 non-null  float64
22  mean_duration                  22699 non-null  float64
23  day                           22699 non-null  object
24  month                          22699 non-null  object
25  rush_hour                     22699 non-null  int64
26  mean_duration_new             22699 non-null  float64
dtypes: datetime64[ns](2), float64(12), int64(8), object(4), timedelta64[ns](1)
memory usage: 4.7+ MB

```

```
[221]: df1 = df0.copy()

df1 = df1.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', 'mean_duration_new', 'duration_in_seconds'
                 ], axis=1)

df1.info()
```

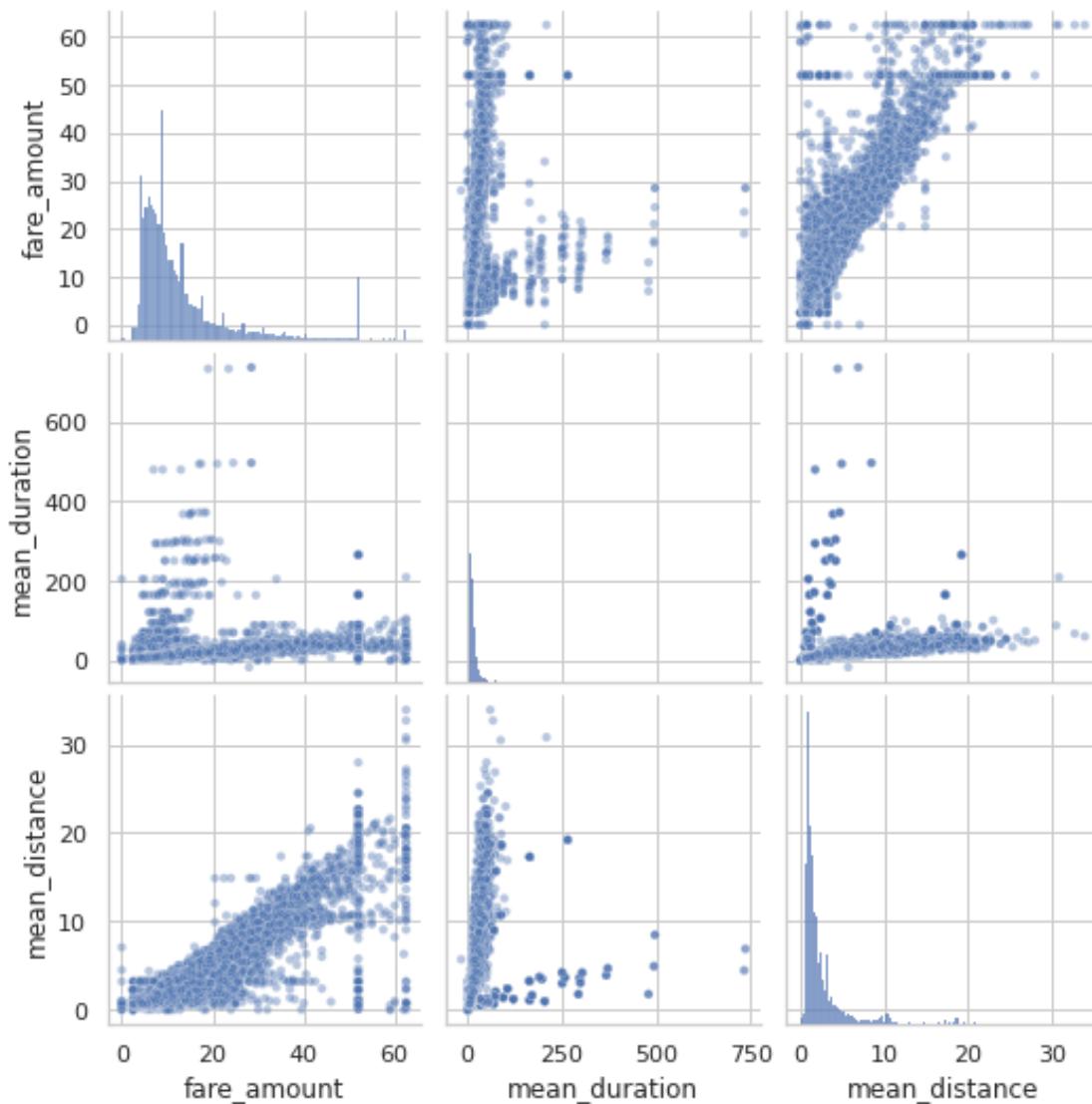
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   VendorID        22699 non-null   int64  
 1   passenger_count 22699 non-null   int64
```

```
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`.

```
[222]: # Create a pairplot to visualize pairwise relationships between variables in
       ↪the data
sns.pairplot(df1[['fare_amount', 'mean_duration', 'mean_distance']],
             plot_kws={'alpha':0.4, 'size':5},
             );
```



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.

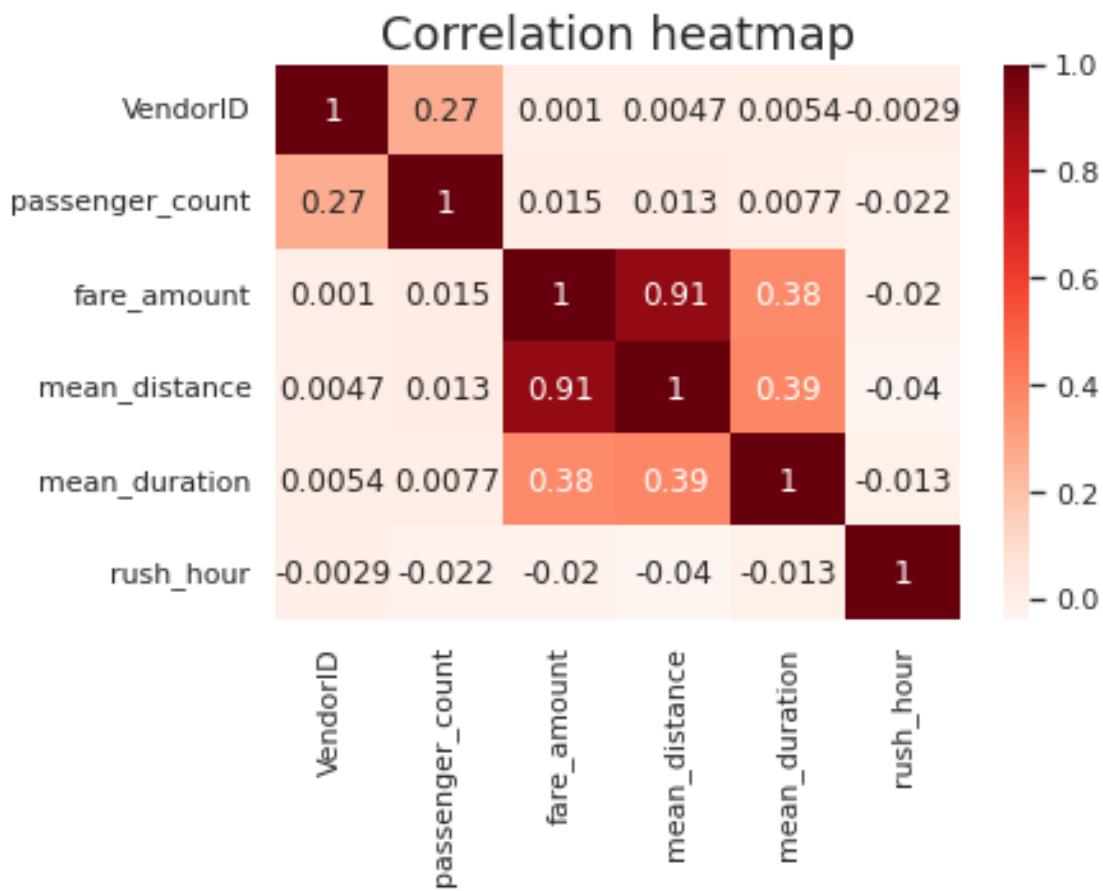
```
[224]: # Correlation matrix to help determine most correlated variables
df1.corr(method='pearson')
```

	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	0.005419	0.007674	0.379710	0.385880
rush_hour	-0.002874	-0.022035	-0.020075	-0.039725
	mean_duration	rush_hour		
VendorID	0.005419	-0.002874		
passenger_count	0.007674	-0.022035		
fare_amount	0.379710	-0.020075		
mean_distance	0.385880	-0.039725		
mean_duration	1.000000	-0.013395		
rush_hour	-0.013395	1.000000		

Visualize a correlation heatmap of the data.

```
[225]: # Create correlation heatmap
plt.figure(figsize=(6,4))
sns.heatmap(df1.corr(method='pearson'), annot=True, cmap='Reds')
plt.title('Correlation heatmap',
           fontsize=18)
plt.show()
```



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

[226]: `df1.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
```

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

[227]: `# Remove the target column from the features`
`X = df1.drop(columns='fare_amount')`

```
# Set y variable
y = df1[['fare_amount']]

# Display first few rows
X.head()
```

[227]:

	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

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

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

```
[228]:   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 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`.

```
[229]: # 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)
```

4.3.4 Standardize the data

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

```
[230]: # Standardize the X variables
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
print('X_train scaled:', X_train_scaled)
```

```
X_train scaled: [[-0.50301524  0.8694684 -0.03111281 -0.64893329  0.89286563]
[-0.50301524 -0.60011281 -0.39518468  1.54099045  0.89286563]
[ 0.27331093 -0.47829156 -0.34302719 -0.64893329 -1.11998936]
...
[-0.50301524 -0.45121122 -0.38710598 -0.64893329 -1.11998936]
[-0.50301524 -0.58944763 -0.46144072  1.54099045 -1.11998936]
[ 1.82596329  0.83673851  0.36688719 -0.64893329  0.89286563]]
```

4.3.5 Fit the model

Instantiate your model and fit it to the training data.

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

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

4.3.6 Task 8c. Evaluate model

4.3.7 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.

```
[232]: # 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: 0.82408171846456
R^2: 0.82408171846456
MAE: 2.505249536544882
MSE: 19.650343661757503
RMSE: 4.432870814918647
```

4.3.8 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`.

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

```
[234]: # 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: 0.8525112490537972
 R²: 0.8525112490537972
 MAE: 2.435591005059153
 MSE: 16.038899289086707
 RMSE: 4.0048594593427005

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.

```
[235]: # 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()
```

	actual	predicted	residual
5818	14.0	12.549215	1.450785
18134	28.0	14.416191	13.583809
4655	5.5	7.236760	-1.736760
7378	15.5	17.006963	-1.506963
13914	9.5	10.333522	-0.833522

4.4.2 Task 9b. Visualize model results

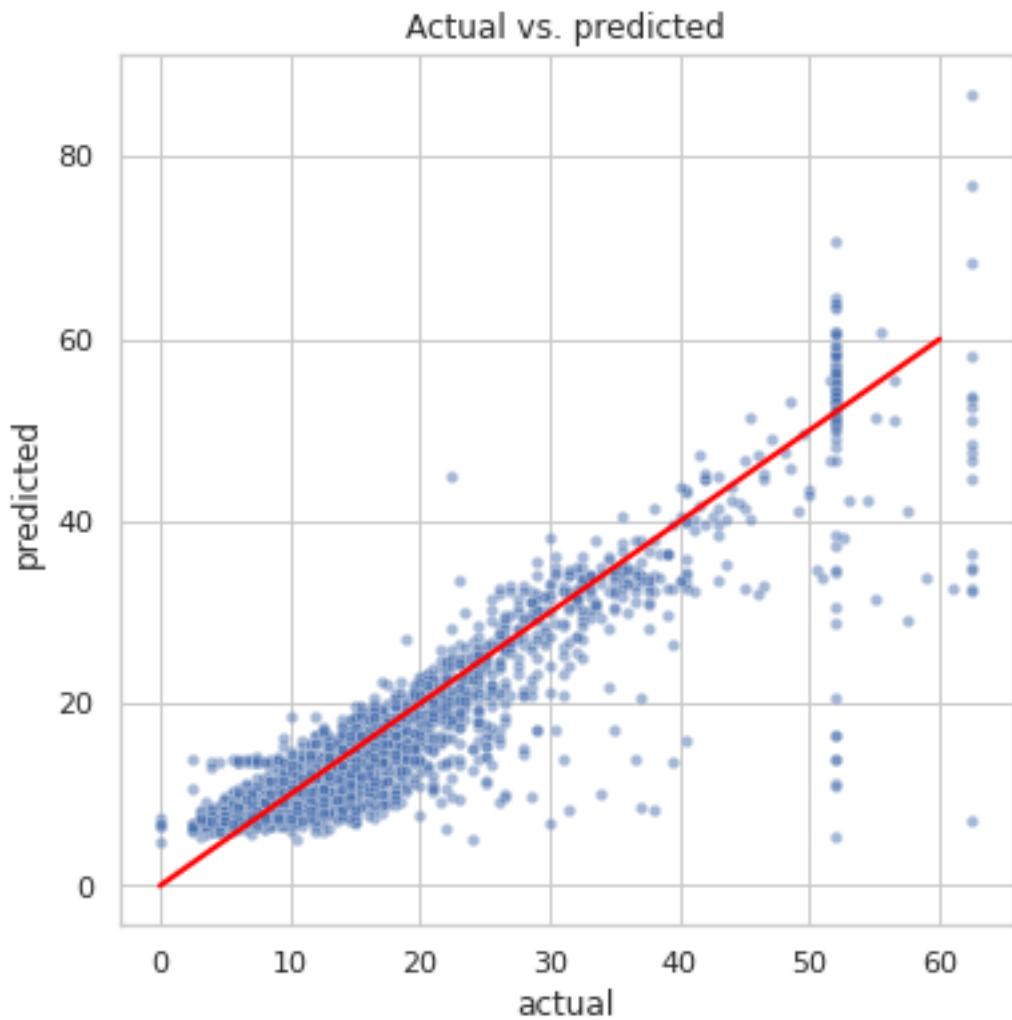
Create a scatterplot to visualize `actual` vs. `predicted`.

```
[236]: # 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,
```

```

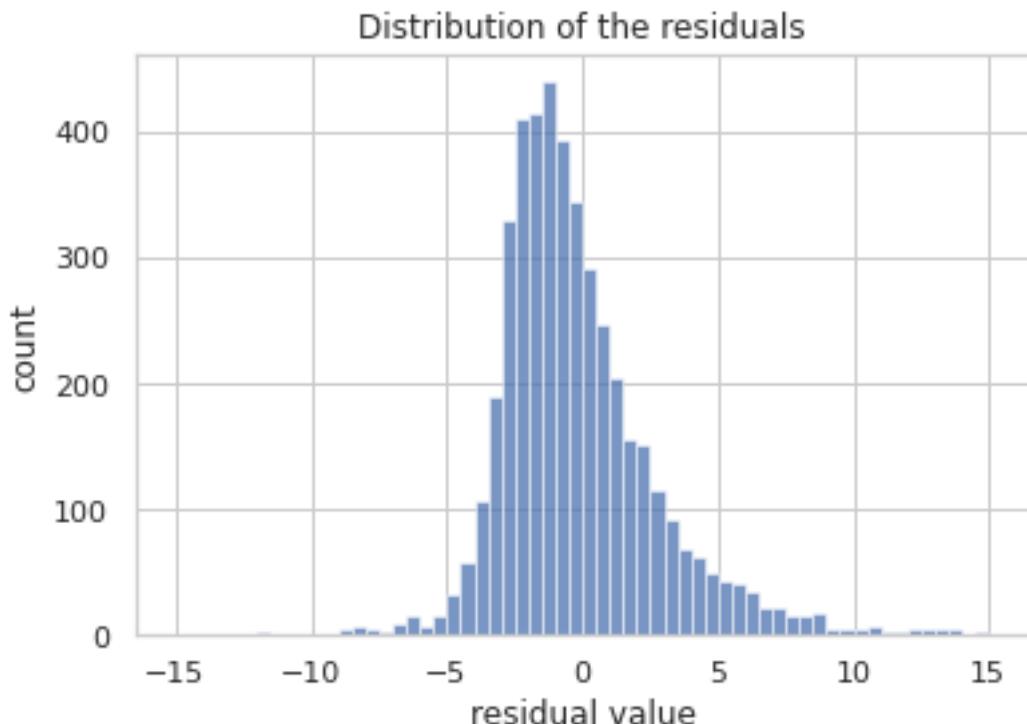
        ax=ax
)
# Draw an x=y line to show what the results would be if the model were perfect
plt.plot([0,60], [0,60], c='red', linewidth=2)
plt.title('Actual vs. predicted');

```



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

```
[237]: # 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');
```

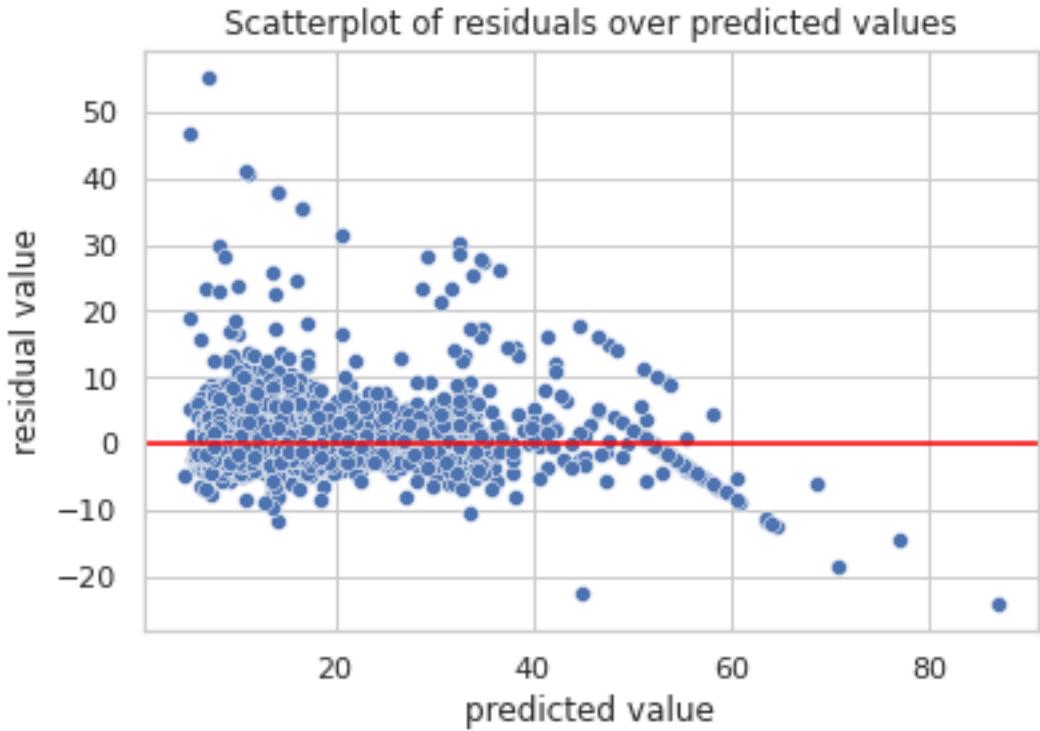


```
[238]: # Calculate residual mean  
results['residual'].mean()
```

```
[238]: 0.009147887686377778
```

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

```
[239]: # 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()
```



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?

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

```
[240]:   passenger_count  mean_distance  mean_duration  rush_hour  VendorID_2
0           0.040686        9.458984       0.349533      0.148466     -0.06373
```

What do these coefficients mean? How should they be interpreted?

For every +1 increase in standard deviation, the fare amount increases by \$9.45. Being maximum, mean distance had the greatest effect on trip fare. Same for the other mean and distinct values.

4.4.4 Task 9d. Conclusion

1. What are the key takeaways from this notebook?
2. What results can be presented from this notebook?
1. Key Takeaways The model built to predict the key variable fare amount.

2. Key Presentations All visualizations

Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.