# Activity_ Course 5 Automatidata project lab

March 13, 2024

## 1 Automatidata project

**Course 5 - Regression Analysis: Simplify complex data relationships**

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

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

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

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

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

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

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

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

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

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

**Part 3:** Interpreting Model Results

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

# 3 Build a multiple linear regression model

# 4 PACE stages

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

## 4.1 PACE: Plan

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

### 4.1.1 Task 1. Imports and loading

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

[1]:
```
# Imports
# Packages for numerics + dataframes
### YOUR CODE HERE ###
import pandas as pd
import numpy as np

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

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

# Packages for OLS, MLR, confusion matrix
```

```
### YOUR CODE HERE ###
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.

[2]:
```
# 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?

==> ENTER YOUR RESPONSE HERE

#### 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()`.

[3]:
```
# Start with `.shape` and `.info()`
### YOUR CODE HERE ###
df = df0.copy()
print(df.shape)
df.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
```

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

[4]:
```
# Check for missing data and duplicates using .isna() and .drop_duplicates()
### YOUR CODE HERE ###
print ("Missing values: ", df.isna().any(axis = 1).sum())
print('Shape of dataframe with duplicates dropped:', df.drop_duplicates().shape)
```

```
Missing values:  0
Shape of dataframe with duplicates dropped: (22699, 18)
```

Use .describe().

[5]:
```
# Use .describe()
### YOUR CODE HERE ###
df.describe()
```

[5]:

| | 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

[6]:
```python
# Check the format of the data
### YOUR CODE HERE ###
df.dtypes
```

[6]:
```
Unnamed: 0                 int64
VendorID                   int64
tpep_pickup_datetime      object
tpep_dropoff_datetime     object
passenger_count            int64
trip_distance            float64
RatecodeID                 int64
store_and_fwd_flag        object
PULocationID               int64
DOLocationID               int64
payment_type               int64
fare_amount              float64
extra                    float64
mta_tax                  float64
tip_amount               float64
tolls_amount             float64
improvement_surcharge    float64
total_amount             float64
dtype: object
```

[7]:
```
# Convert datetime columns to datetime
### YOUR CODE HERE ###
df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])
df['tpep_dropoff_datetime'] = pd.to_datetime(df['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.

[8]:
```
# Create `duration` column
### YOUR CODE HERE ###
df['duration'] = (df['tpep_dropoff_datetime'] - df['tpep_pickup_datetime'])/np.
↪timedelta64(1,'m')
```

#### 4.2.4 Outliers

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

[9]:
```
### YOUR CODE HERE ###
df.info()
```

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

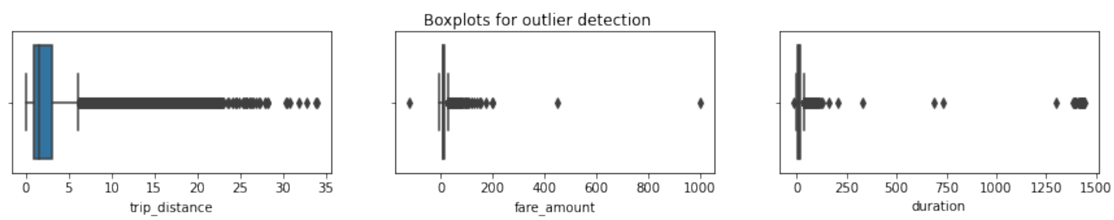
```
dtypes: datetime64[ns](2), float64(9), int64(7), object(1)
memory usage: 3.3+ MB
```

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

### 4.2.5 Task 2d. Box plots

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

[10]:
```
### YOUR CODE HERE ###
fig, axes = plt.subplots(1, 3, figsize=(15, 2))
fig.suptitle('Boxplots for outlier detection')
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()
```




**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?

==> ENTER YOUR RESPONSE HERE All three variables contain outliers. Some are extreme, but others not so much.

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.

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

### 4.2.6 Task 2e. Imputations

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

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

[11]:
```
# Are trip distances of 0 bad data or very short trips rounded down?
### YOUR CODE HERE ###
sorted(set(df['trip_distance']))[:10]
```

[11]:
```
[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.

[12]:
```
### YOUR CODE HERE ###
sum(df['trip_distance']==0)
```

[12]:
```
148
```

**`fare_amount` outliers**

[13]:
```
### YOUR CODE HERE ###
df['fare_amount'].describe()
```

[13]:
```
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? 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`.

[14]:
```
# Impute values less than $0 with 0
### YOUR CODE HERE ###
df.loc[df['fare_amount'] < 0, 'fare_amount'] = 0
```

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

[19]:
```
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.
    '''
    for col in column_list:
        # Reassign minimum to zero
        df.loc[df[col] < 0, col] = 0

        # Calculate upper threshold
        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
        df.loc[df[col] > upper_threshold, col] = upper_threshold
        print(df[col].describe())
        print()
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**

[20]:
```
# Call .describe() for duration outliers
### YOUR CODE HERE ###
df['duration'].describe()
```

[20]:
```
count    22699.000000
mean        17.013777
std         61.996482
min        -16.983333
25%          6.650000
50%         11.183333
75%         18.383333
max       1439.550000
Name: duration, dtype: float64
```

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

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

[22]:
```
# Impute a 0 for any negative values
### YOUR CODE HERE ###
df.loc[df['duration'] < 0, 'duration'] = 0
```

[24]:
```
# Impute the high outliers
### YOUR CODE HERE ###
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
```

### 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' |

[25]:
```
# Create `pickup_dropoff` column
### YOUR CODE HERE ###
```

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

[30]:
```
### YOUR CODE HERE ###
grouped = df.groupby(['pickup_dropoff']).mean()[['trip_distance']]
grouped[:5]
```

[30]:
```
                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 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}
```

[35]:
```
# 1. Convert `grouped` to a dictionary
### YOUR CODE HERE ###
grouped_dict = grouped.to_dict()

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

1. Create a `mean_distance` column that is a copy of the `pickup_dropoff` helper column.

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

```
Example:
```

```
df['mean_distance']
```

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

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

| mean_distance |
|---|
| 1.25 |
| 2 |
| 1.25 |
| 3 |
| NaN |

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

[37]:
```
# 1. Create a mean_distance column that is a copy of the pickup_dropoff helper␣
↪column
### YOUR CODE HERE ###
df['mean_distance'] = df['pickup_dropoff']

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

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

[37]:
```
       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.

[38]:
```
grouped = df.groupby('pickup_dropoff').mean(numeric_only=True)[['duration']]

# 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']]
```

[38]:

| | 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.

[64]:
```
# Create 'day' col
### YOUR CODE HERE ###
df['day'] = df['tpep_pickup_datetime'].dt.day_name().str.lower()

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

**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.

[65]:
```
# 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
```

[66]:
```
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
```

[67]:
```
# 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()
```

[67]:
```
   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     march         0
1   tuesday     april         0
2    friday  december         1
3    sunday       may         0
```

```
4  saturday     april        0

[5 rows x 25 columns]
```

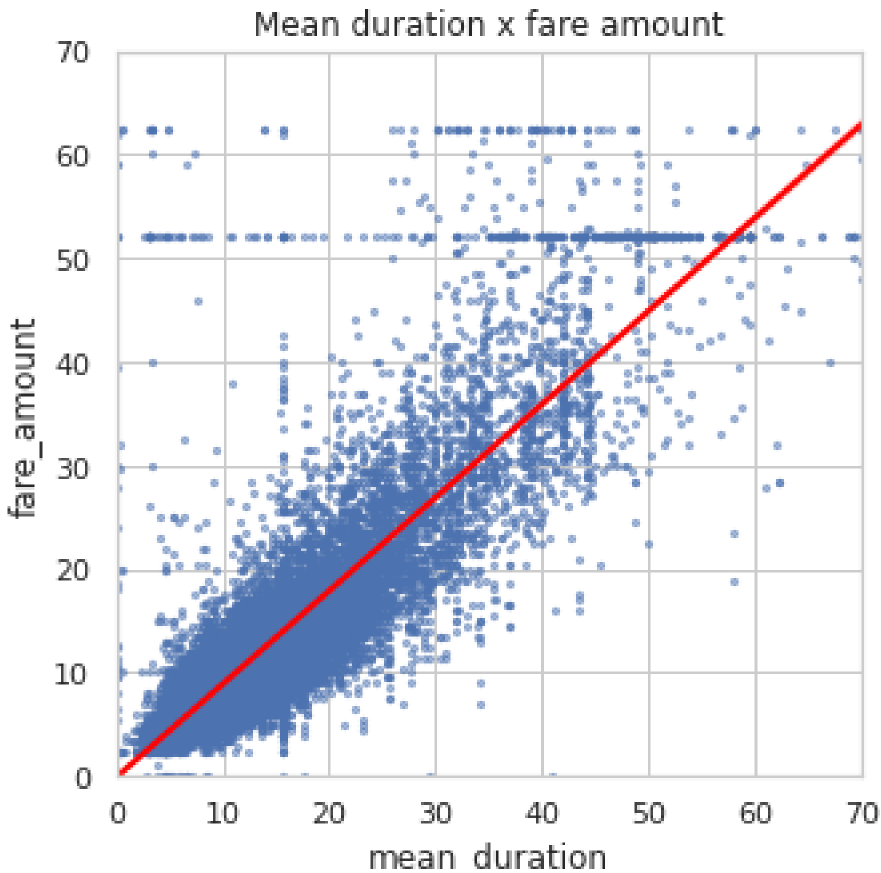
### 4.2.8 Task 4. Scatter plot

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

[69]:

```
# Create a scatterplot to visualize the relationship between variables of␣
↪interest
### YOUR CODE HERE ###
#sns.scatterplot(x = 'mean_duration', y = 'fare_amount', data = df)

###Exempler method
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?

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.

[71]:
```
### YOUR CODE HERE ###
df[df['fare_amount'] > 50]['fare_amount'].value_counts().head()
```

[71]:
```
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.

[72]:
```
# Set pandas to display all columns
### YOUR CODE HERE ###
pd.set_option('display.max_columns', None)
df[df['fare_amount']==52].head(30)
```

[72]:
```
      Unnamed: 0  VendorID tpep_pickup_datetime tpep_dropoff_datetime  \
11      18600059         2  2017-03-05 19:15:30   2017-03-05 19:52:18
110     47959795         1  2017-06-03 14:24:57   2017-06-03 15:31:48
161     95729204         2  2017-11-11 20:16:16   2017-11-11 20:17:14
247    103404868         2  2017-12-06 23:37:08   2017-12-07 00:06:19
379     80479432         2  2017-09-24 23:45:45   2017-09-25 00:15:14
388     16226157         1  2017-02-28 18:30:05   2017-02-28 19:09:55
406     55253442         2  2017-06-05 12:51:58   2017-06-05 13:07:35
449     65900029         2  2017-08-03 22:47:14   2017-08-03 23:32:41
468     80904240         2  2017-09-26 13:48:26   2017-09-26 14:31:17
520     33706214         2  2017-04-23 21:34:48   2017-04-23 22:46:23
569     99259872         2  2017-11-22 21:31:32   2017-11-22 22:00:25
572     61050418         2  2017-07-18 13:29:06   2017-07-18 13:29:19
586     54444647         2  2017-06-26 13:39:12   2017-06-26 14:34:54
692     94424289         2  2017-11-07 22:15:00   2017-11-07 22:45:32
717    103094220         1  2017-12-06 05:19:50   2017-12-06 05:53:52
719     66115834         1  2017-08-04 17:53:34   2017-08-04 18:50:56
782     55934137         2  2017-06-09 09:31:25   2017-06-09 10:24:10
816     13731926         2  2017-02-21 06:11:03   2017-02-21 06:59:39
818     52277743         2  2017-06-20 08:15:18   2017-06-20 10:24:37
835      2684305         2  2017-01-10 22:29:47   2017-01-10 23:06:46
840     90860814         2  2017-10-27 21:50:00   2017-10-27 22:35:04
861    106575186         1  2017-12-16 06:39:59   2017-12-16 07:07:59
881    110495611         2  2017-12-30 05:25:29   2017-12-30 06:01:29
958     87017503         1  2017-10-15 22:39:12   2017-10-15 23:14:22
970     12762608         2  2017-02-17 20:39:42   2017-02-17 21:13:29
984     71264442         1  2017-08-23 18:23:26   2017-08-23 19:18:29
1082    11006300         2  2017-02-07 17:20:19   2017-02-07 17:34:41
1097    68882036         2  2017-08-14 23:01:15   2017-08-14 23:03:35
1110    74720333         1  2017-09-06 10:46:17   2017-09-06 11:44:41
1179    51937907         2  2017-06-19 06:23:13   2017-06-19 07:03:53

      passenger_count  trip_distance  RatecodeID store_and_fwd_flag  \
11                  2          18.90           2                  N
110                 1          18.00           2                  N
161                 1           0.23           2                  N
247                 1          18.93           2                  N
379                 1          17.99           2                  N
388                 1          18.40           2                  N
406                 1           4.73           2                  N
449                 2          18.21           2                  N
468                 1          17.27           2                  N
```

| | | | | |
|---|---|---|---|---|
| 520 | 6 | 18.34 | 2 | N |
| 569 | 1 | 18.65 | 2 | N |
| 572 | 1 | 0.00 | 2 | N |
| 586 | 1 | 17.76 | 2 | N |
| 692 | 2 | 16.97 | 2 | N |
| 717 | 1 | 20.80 | 2 | N |
| 719 | 1 | 21.60 | 2 | N |
| 782 | 2 | 18.81 | 2 | N |
| 816 | 5 | 16.94 | 2 | N |
| 818 | 1 | 17.77 | 2 | N |
| 835 | 1 | 18.57 | 2 | N |
| 840 | 1 | 22.43 | 2 | N |
| 861 | 2 | 17.80 | 2 | N |
| 881 | 6 | 18.23 | 2 | N |
| 958 | 1 | 21.80 | 2 | N |
| 970 | 1 | 19.57 | 2 | N |
| 984 | 1 | 16.70 | 2 | N |
| 1082 | 1 | 1.09 | 2 | N |
| 1097 | 5 | 2.12 | 2 | N |
| 1110 | 1 | 19.10 | 2 | N |
| 1179 | 6 | 19.77 | 2 | N |

| | PULocationID | DOLocationID | payment_type | fare_amount | extra | mta_tax | \ |
|---|---|---|---|---|---|---|---|
| 11 | 236 | 132 | 1 | 52.0 | 0.0 | 0.5 | |
| 110 | 132 | 163 | 1 | 52.0 | 0.0 | 0.5 | |
| 161 | 132 | 132 | 2 | 52.0 | 0.0 | 0.5 | |
| 247 | 132 | 79 | 2 | 52.0 | 0.0 | 0.5 | |
| 379 | 132 | 234 | 1 | 52.0 | 0.0 | 0.5 | |
| 388 | 132 | 48 | 2 | 52.0 | 4.5 | 0.5 | |
| 406 | 228 | 88 | 2 | 52.0 | 0.0 | 0.5 | |
| 449 | 132 | 48 | 2 | 52.0 | 0.0 | 0.5 | |
| 468 | 186 | 132 | 2 | 52.0 | 0.0 | 0.5 | |
| 520 | 132 | 148 | 1 | 52.0 | 0.0 | 0.5 | |
| 569 | 132 | 144 | 1 | 52.0 | 0.0 | 0.5 | |
| 572 | 230 | 161 | 1 | 52.0 | 0.0 | 0.5 | |
| 586 | 211 | 132 | 1 | 52.0 | 0.0 | 0.5 | |
| 692 | 132 | 170 | 1 | 52.0 | 0.0 | 0.5 | |
| 717 | 132 | 239 | 1 | 52.0 | 0.0 | 0.5 | |
| 719 | 264 | 264 | 1 | 52.0 | 4.5 | 0.5 | |
| 782 | 163 | 132 | 1 | 52.0 | 0.0 | 0.5 | |
| 816 | 132 | 170 | 1 | 52.0 | 0.0 | 0.5 | |
| 818 | 132 | 246 | 1 | 52.0 | 0.0 | 0.5 | |
| 835 | 132 | 48 | 1 | 52.0 | 0.0 | 0.5 | |
| 840 | 132 | 163 | 2 | 52.0 | 0.0 | 0.5 | |
| 861 | 75 | 132 | 1 | 52.0 | 0.0 | 0.5 | |
| 881 | 68 | 132 | 2 | 52.0 | 0.0 | 0.5 | |
| 958 | 132 | 261 | 2 | 52.0 | 0.0 | 0.5 | |

```
970           132          140             1        52.0   0.0      0.5
984           132          230             1        52.0   4.5      0.5
1082          170           48             2        52.0   4.5      0.5
1097          265          265             2        52.0   0.0      0.5
1110          239          132             1        52.0   0.0      0.5
1179          238          132             1        52.0   0.0      0.5

      tip_amount  tolls_amount  improvement_surcharge  total_amount  \
11         14.58          5.54                    0.3         72.92
110         0.00          0.00                    0.3         52.80
161         0.00          0.00                    0.3         52.80
247         0.00          0.00                    0.3         52.80
379        14.64          5.76                    0.3         73.20
388         0.00          5.54                    0.3         62.84
406         0.00          5.76                    0.3         58.56
449         0.00          5.76                    0.3         58.56
468         0.00          5.76                    0.3         58.56
520         5.00          0.00                    0.3         57.80
569        10.56          0.00                    0.3         63.36
572        11.71          5.76                    0.3         70.27
586        11.71          5.76                    0.3         70.27
692        11.71          5.76                    0.3         70.27
717         5.85          5.76                    0.3         64.41
719        12.60          5.76                    0.3         75.66
782        13.20          0.00                    0.3         66.00
816         2.00          5.54                    0.3         60.34
818        11.71          5.76                    0.3         70.27
835        13.20          0.00                    0.3         66.00
840         0.00          5.76                    0.3         58.56
861         6.00          5.76                    0.3         64.56
881         0.00          0.00                    0.3         52.80
958         0.00          0.00                    0.3         52.80
970        11.67          5.54                    0.3         70.01
984        42.29          0.00                    0.3         99.59
1082        0.00          5.54                    0.3         62.84
1097        0.00          0.00                    0.3         52.80
1110       15.80          0.00                    0.3         68.60
1179       17.57          5.76                    0.3         76.13

       duration pickup_dropoff  mean_distance  mean_duration        day  \
11    36.800000        236 132      19.211667      40.500000     sunday
110   66.850000        132 163      19.229000      52.941667   saturday
161    0.966667        132 132       2.255862       3.021839   saturday
247   29.183333         132 79      19.431667      47.275000  wednesday
379   29.483333        132 234      17.654000      49.833333     sunday
388   39.833333         132 48      18.761905      58.246032    tuesday
406   15.616667         228 88       4.730000      15.616667     monday
```

| | | | | | |
|---|---|---|---|---|---|
| 449 | 45.450000 | 132 48 | 18.761905 | 58.246032 | thursday |
| 468 | 42.850000 | 186 132 | 17.096000 | 42.920000 | tuesday |
| 520 | 71.583333 | 132 148 | 17.994286 | 46.340476 | sunday |
| 569 | 28.883333 | 132 144 | 18.537500 | 37.000000 | wednesday |
| 572 | 0.216667 | 230 161 | 0.685484 | 7.965591 | tuesday |
| 586 | 55.700000 | 211 132 | 16.580000 | 61.691667 | monday |
| 692 | 30.533333 | 132 170 | 17.203000 | 37.113333 | tuesday |
| 717 | 34.033333 | 132 239 | 20.901250 | 44.862500 | wednesday |
| 719 | 57.366667 | 264 264 | 3.191516 | 15.618773 | friday |
| 782 | 52.750000 | 163 132 | 17.275833 | 52.338889 | friday |
| 816 | 48.600000 | 132 170 | 17.203000 | 37.113333 | tuesday |
| 818 | 88.783333 | 132 246 | 18.515000 | 66.316667 | tuesday |
| 835 | 36.983333 | 132 48 | 18.761905 | 58.246032 | tuesday |
| 840 | 45.066667 | 132 163 | 19.229000 | 52.941667 | friday |
| 861 | 28.000000 | 75 132 | 18.442500 | 36.204167 | saturday |
| 881 | 36.000000 | 68 132 | 18.785000 | 58.041667 | saturday |
| 958 | 35.166667 | 132 261 | 22.115000 | 51.493750 | sunday |
| 970 | 33.783333 | 132 140 | 19.293333 | 36.791667 | friday |
| 984 | 55.050000 | 132 230 | 18.571200 | 59.598000 | wednesday |
| 1082 | 14.366667 | 170 48 | 1.265789 | 14.135965 | tuesday |
| 1097 | 2.333333 | 265 265 | 0.753077 | 3.411538 | monday |
| 1110 | 58.400000 | 239 132 | 19.795000 | 50.562500 | wednesday |
| 1179 | 40.666667 | 238 132 | 19.470000 | 53.861111 | monday |

| | month | rush_hour |
|---|---|---|
| 11 | march | 0 |
| 110 | june | 0 |
| 161 | november | 0 |
| 247 | december | 0 |
| 379 | september | 0 |
| 388 | february | 1 |
| 406 | june | 0 |
| 449 | august | 0 |
| 468 | september | 0 |
| 520 | april | 0 |
| 569 | november | 0 |
| 572 | july | 0 |
| 586 | june | 0 |
| 692 | november | 0 |
| 717 | december | 0 |
| 719 | august | 1 |
| 782 | june | 1 |
| 816 | february | 1 |
| 818 | june | 1 |
| 835 | january | 0 |
| 840 | october | 0 |
| 861 | december | 0 |

```
881     december          0
958      october          0
970     february          0
984       august          1
1082    february          1
1097      august          0
1110   september          0
1179        june          1
```

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

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

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

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

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

### 4.2.9 Task 5. Isolate modeling variables

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

[73]:
```
### YOUR CODE HERE ###
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
```

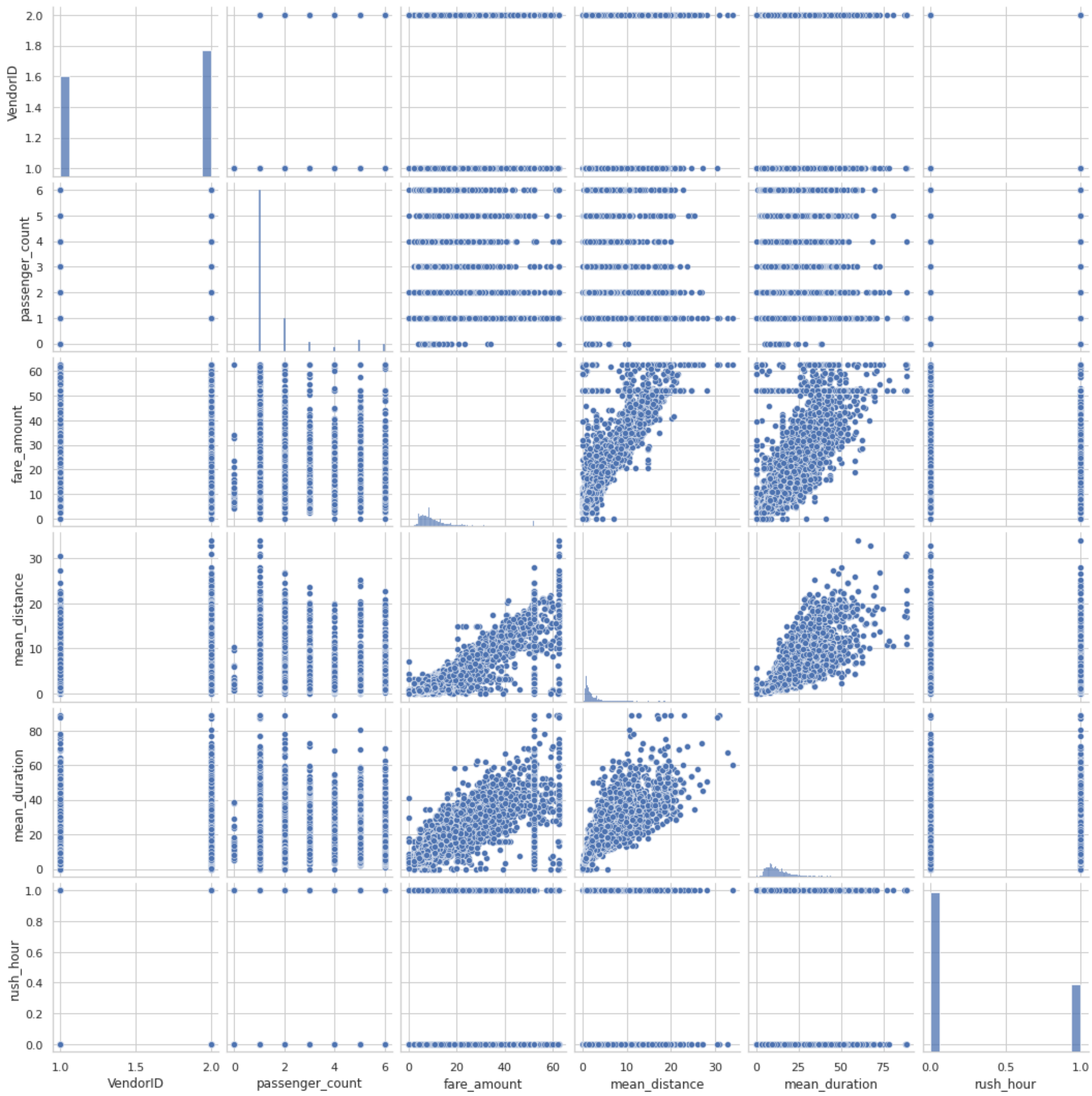
[ ]:
```
### YOUR CODE HERE ###
```

#### 4.2.10 Task 6. Pair plot

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

[75]:
```
# Create a pairplot to visualize pairwise relationships between variables in
↪the data
### YOUR CODE HERE ###
sns.pairplot(df2)
```

[75]:
```
<seaborn.axisgrid.PairGrid at 0x7f26d6c74f10>
```




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.

[76]:
```
# Correlation matrix to help determine most correlated variables
### YOUR CODE HERE ###
df2.corr(method='pearson')
```

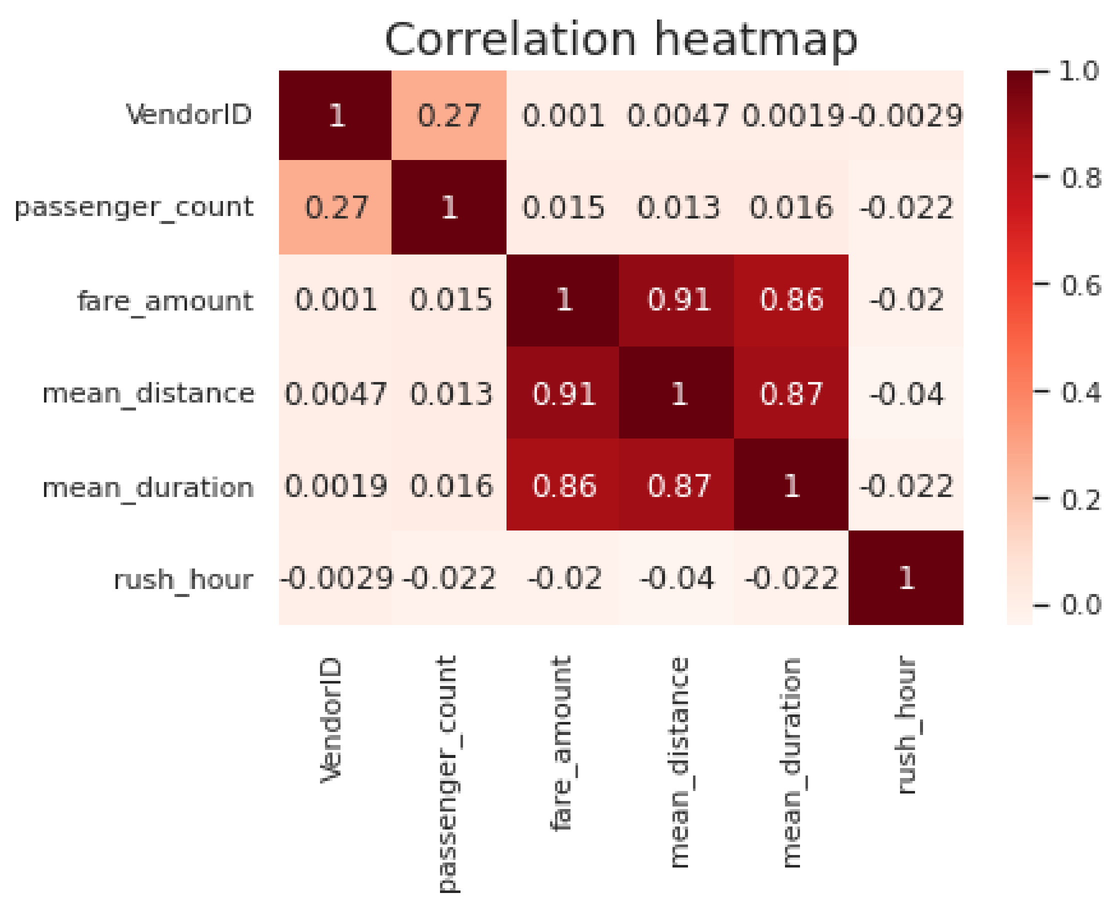
[76]:
```
                 VendorID  passenger_count  fare_amount  mean_distance  \
VendorID         1.000000         0.266463     0.001045       0.004741
passenger_count  0.266463         1.000000     0.014942       0.013428
```

```
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.

[77]:
```python
# Create correlation heatmap
### YOUR CODE HERE ###
plt.figure(figsize=(6,4))
sns.heatmap(df2.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`? 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.

Recall that highly correlated predictor variables can be bad for linear regression models when you 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.

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

[78]:
```
### YOUR CODE HERE ###
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
```

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

[79]:
```
# Remove the target column from the features
X = df2.drop(columns=['fare_amount'])

# Set y variable
y = df2[['fare_amount']]

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

[79]:

| | 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

[80]:
```
# Convert VendorID to string
### YOUR CODE HERE ###
X['VendorID'] = X['VendorID'].astype(str)

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

[80]:

| | 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`.

[81]:
```
# Create training and testing sets
#### YOUR CODE HERE ####
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=0)
```

### 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`.

[82]:
```
# Standardize the X variables
### YOUR CODE HERE ###
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   0.17616665 -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]]
```

### 4.3.5 Fit the model

Instantiate your model and fit it to the training data.

[83]:
```
# Fit your model to the training data
### YOUR CODE HERE ###
lr=LinearRegression()
lr.fit(X_train_scaled, y_train)
```

[83]:
```
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.

[86]:
```
# Evaluate the model performance on the training data
### YOUR CODE HERE ###
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
```

### 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`.

[84]:
```python
# Scale the X_test data
### YOUR CODE HERE ###
X_test_scaled = scaler.transform(X_test)
```

[85]:
```python
# Evaluate the model performance on the testing data
### YOUR CODE HERE ###
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
```

## 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.

[87]:
```python
# Create a `results` dataframe
### YOUR CODE HERE ###
results = pd.DataFrame(data={'actual': y_test['fare_amount'],
                             'predicted': y_pred_test.ravel()})
results['residual'] = results['actual'] - results['predicted']
results.head()
```

[87]:

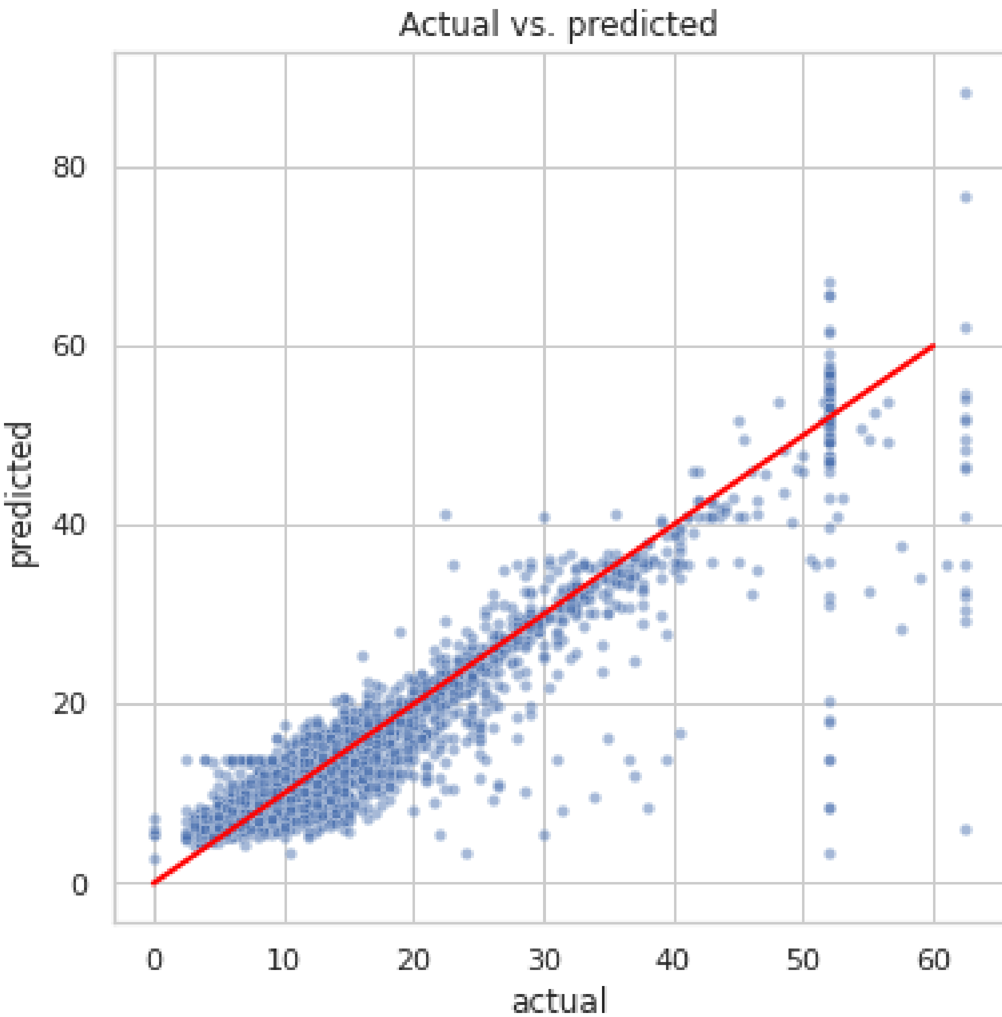
| | actual | predicted | residual |
|---|---|---|---|
| 5818 | 14.0 | 12.356503 | 1.643497 |
| 18134 | 28.0 | 16.314595 | 11.685405 |
| 4655 | 5.5 | 6.726789 | -1.226789 |
| 7378 | 15.5 | 16.227206 | -0.727206 |

```
13914      9.5  10.536408  -1.036408
```

#### 4.4.2 Task 9b. Visualize model results

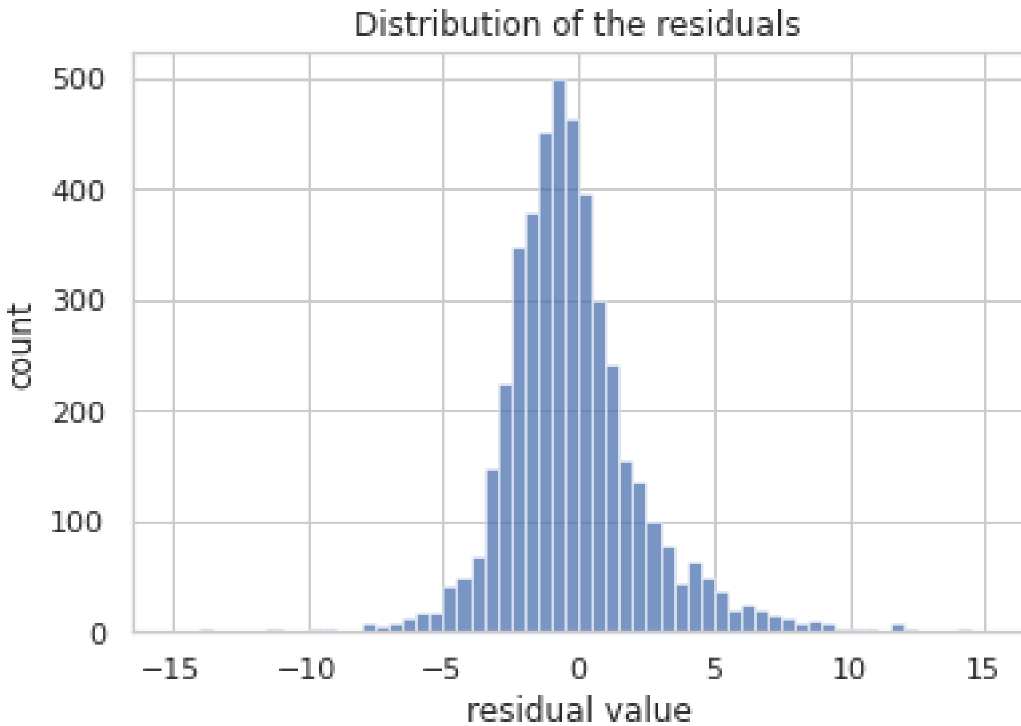
Create a scatterplot to visualize `actual` vs. `predicted`.

[88]:
```
# Create a scatterplot to visualize `predicted` over `actual`
### YOUR CODE HERE ###
fig, ax = plt.subplots(figsize=(6, 6))
sns.set(style='whitegrid')
sns.scatterplot(x='actual',
                y='predicted',
                data=results,
                s=20,
                alpha=0.5,
                ax=ax
)
# 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.

[89]:
```
# Visualize the distribution of the `residuals`
### YOUR CODE HERE ###
sns.histplot(results['residual'], bins=np.arange(-15,15.5,0.5))
plt.title('Distribution of the residuals')
plt.xlabel('residual value')
plt.ylabel('count');
```

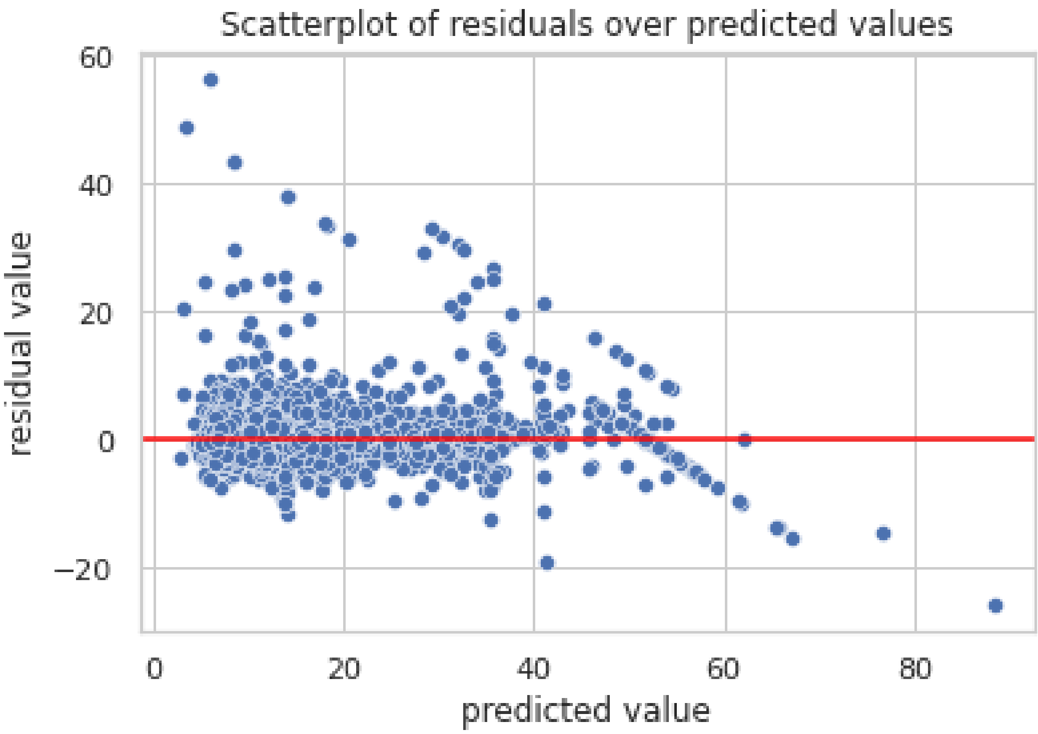



[90]:
```
# Calculate residual mean
### YOUR CODE HERE ###
results['residual'].mean()
```

[90]: -0.01544262152868053

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

[91]:
```
# Create a scatterplot of `residuals` over `predicted`
### YOUR CODE HERE ###
sns.scatterplot(x='predicted', y='residual', data=results)
plt.axhline(0, c='red')
plt.title('Scatterplot of residuals over predicted values')
plt.xlabel('predicted value')
plt.ylabel('residual value')
plt.show()
```




### 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?

[92]:
```
# Output the model's coefficients
coefficients = pd.DataFrame(lr.coef_, columns=X.columns)
coefficients
```

[92]:

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

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

==> ENTER YOUR RESPONSE HERE 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, you 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).

Calculate the standard deviation of mean_distance in the X_train data.

Divide the coefficient (7.133867) by the result to yield a more intuitive interpretation.

[93]:
```
# 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())
```

```
3.574812975256415
1.9955916713344426
```

[ ]:
```
Now you 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.
```

#### 4.4.4 Task 9d. Conclusion

1. What are the key takeaways from this notebook? Multiple linear regression is a powerful tool to estimate a dependent continous variable from several independent variables. Exploratory data analysis is useful for selecting both numeric and categorical features for multiple linear regression. Fitting multiple linear regression models may require trial and error to select variables that fit an accurate model while maintaining model assumptions (or not, depending on your use case).

2. What results can be presented from this notebook? You can discuss meeting linear regression assumptions, and you can present the MAE and RMSE scores obtained from the model.

==> ENTER YOUR RESPONSE HERE

**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.