Automatidata Project

Kasra Hashemi

Jul 10, 2024

CONTENTS

As a junior data analyst at the fictional data consulting company Automatidata, I was given a project to analyze a dataset from the New York City Taxi and Limousine Commission project (New York City TLC). The aim of this project was to help NYC TLC make data-driven decisions to improve their operations.

CHAPTER ONE

EDA

You are the newest data professional in a fictional data consulting firm: Automatidata. The team is still early into the project, having only just completed an initial plan of action and some early Python coding work.

Luana Rodriquez, the senior data analyst at Automatidata, is pleased with the work you have already completed and requests your assistance with some EDA and data visualization work for the New York City Taxi and Limousine Commission project (New York City TLC) to get a general understanding of what taxi ridership looks like. The management team is asking for a Python notebook showing data structuring and cleaning, as well as any matplotlib/seaborn visualizations plotted to help understand the data. At the very least, include a box plot of the ride durations and some time series plots, like a breakdown by quarter or month.

Additionally, the management team has recently asked all EDA to include Tableau visualizations. For this taxi data, create a Tableau dashboard showing a New York City map of taxi/limo trips by month. Make sure it is easy to understand to someone who isn't data savvy, and remember that the assistant director at the New York City TLC is a person with visual impairments.

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

1.1 First steps to examine the structure of the dataset

```
# Import packages and libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Load dataset into dataframe
df = pd.read_csv('2017_Yellow_Taxi_Trip_Data.csv')
```

```
df.head()
```

```
   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
```

(continues on next page)

(continued from previous page)

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

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

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

```python
print(df.size)
print()
print(df.shape)
```

```
408582

(22699, 18)
```

```python
df.describe()
```

```
         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   
```

(continues on next page)

(continued from previous page)

```
              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
```

```
df.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
```

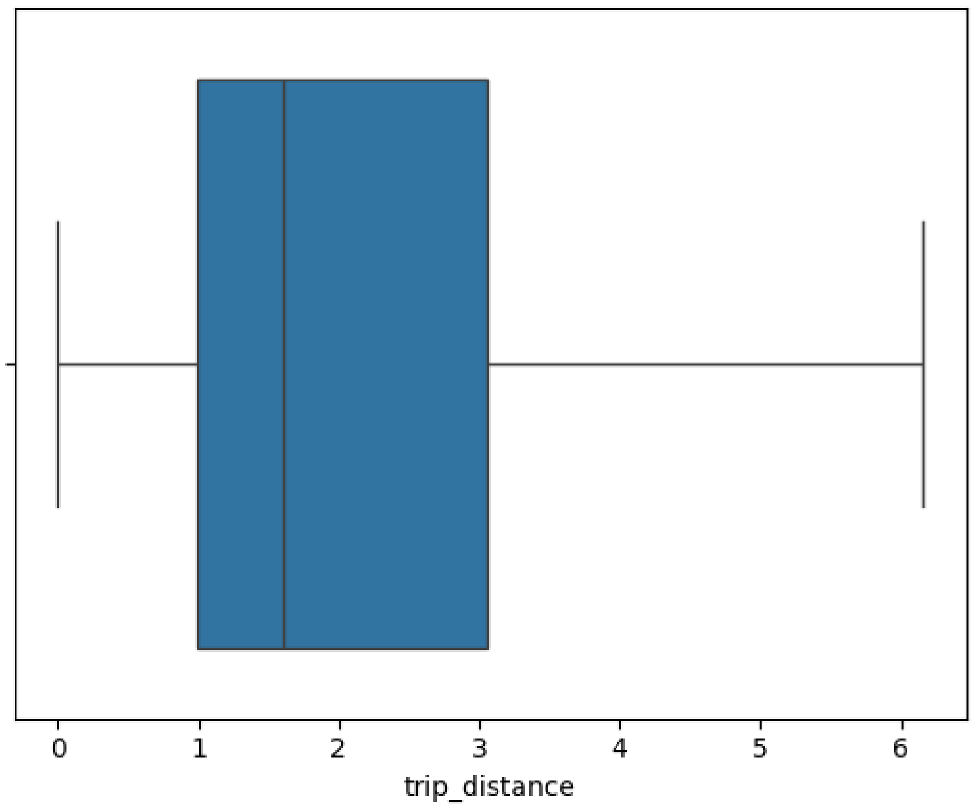
1.2 Data Visualization

trip distance

```
# Create box plot of trip_distance
sns.boxplot(x=df['trip_distance'], orient='v', showfliers=False)
```

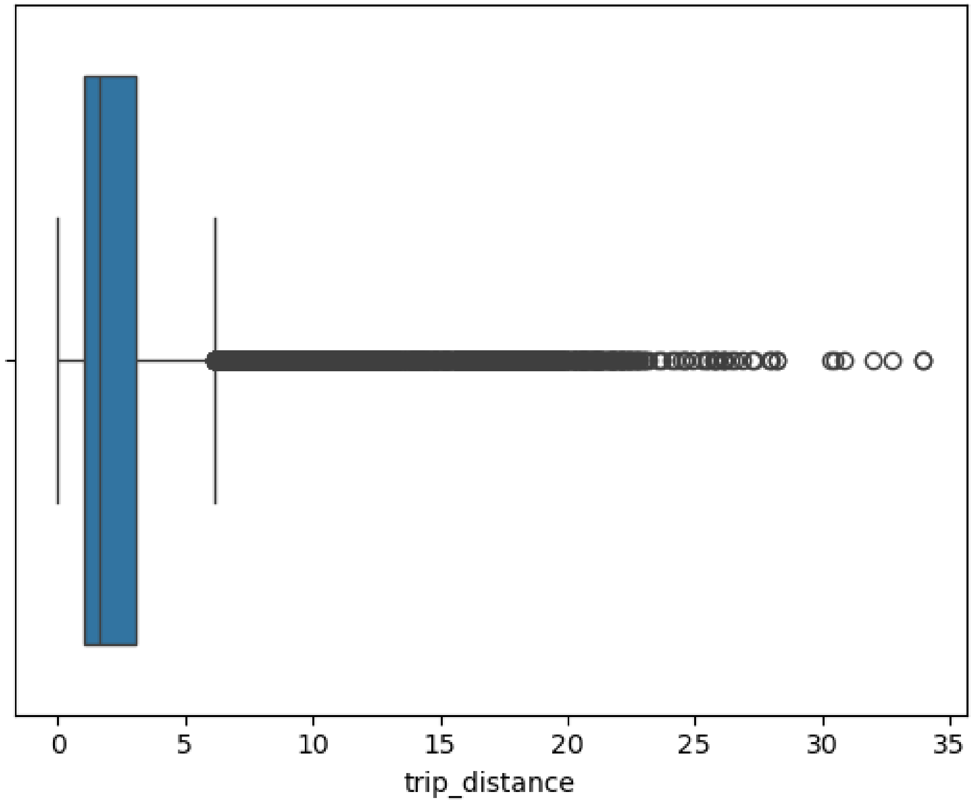
```
/opt/anaconda3/envs/General/lib/python3.12/site-packages/seaborn/_base.py:1608: 
↪UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```

```
<Axes: xlabel='trip_distance'>
```



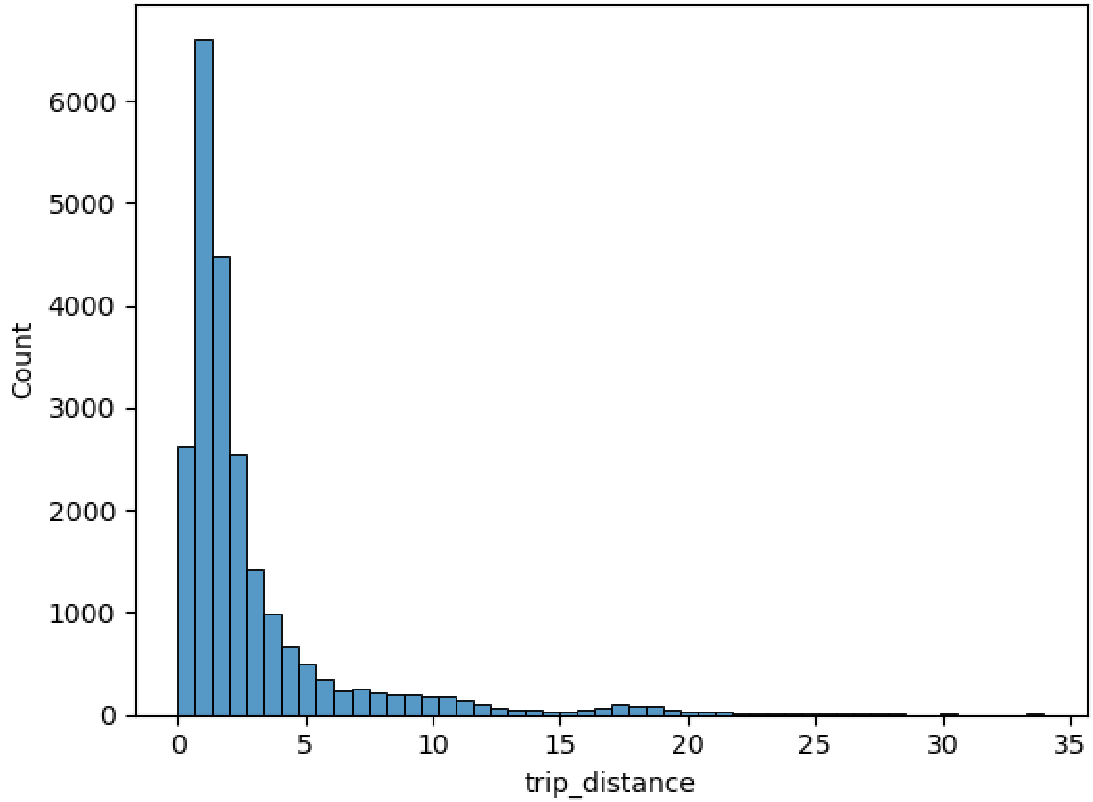

```
sns.boxplot(x=df['trip_distance'])
```

```
<Axes: xlabel='trip_distance'>
```




```
sns.histplot(df['trip_distance'], bins=50)
```

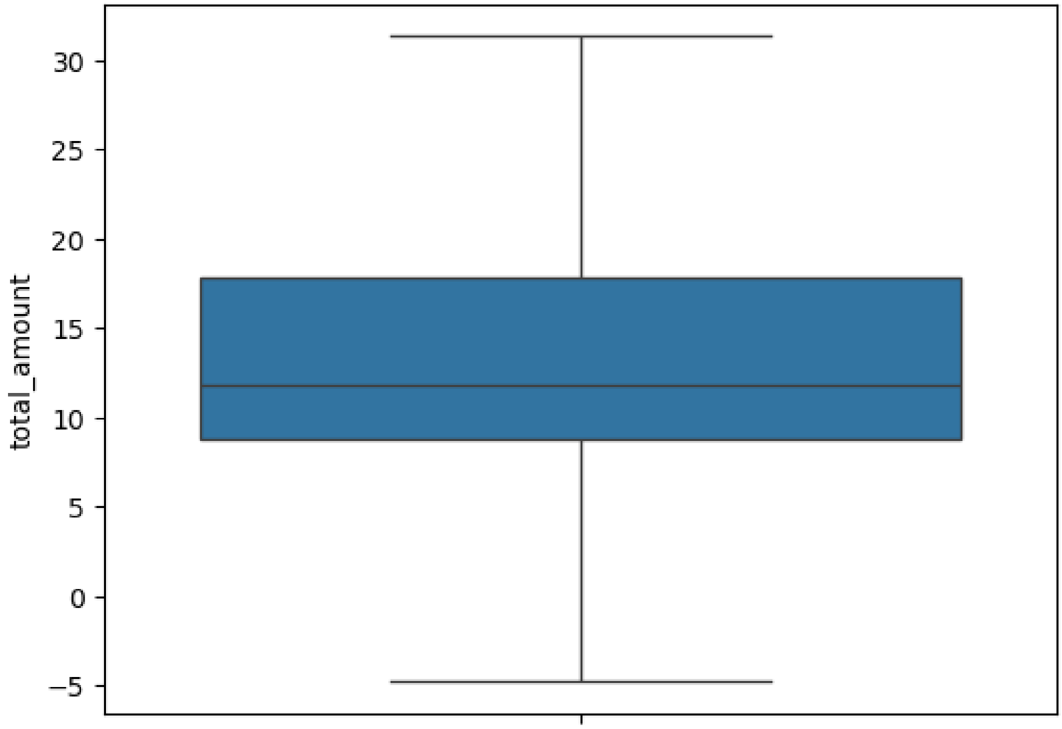
```
<Axes: xlabel='trip_distance', ylabel='Count'>
```



total amount

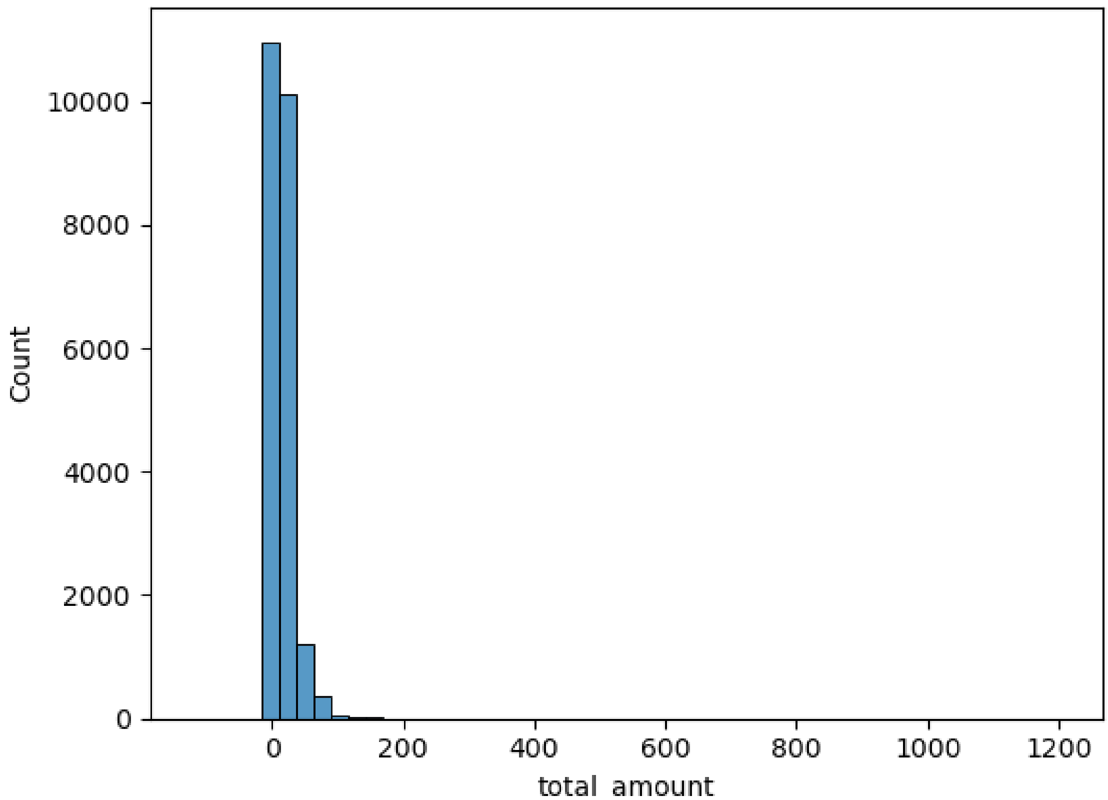
```
sns.boxplot(df['total_amount'], showfliers=False)
```

```
<Axes: ylabel='total_amount'>
```




```
sns.histplot(df['total_amount'], bins=50)
```

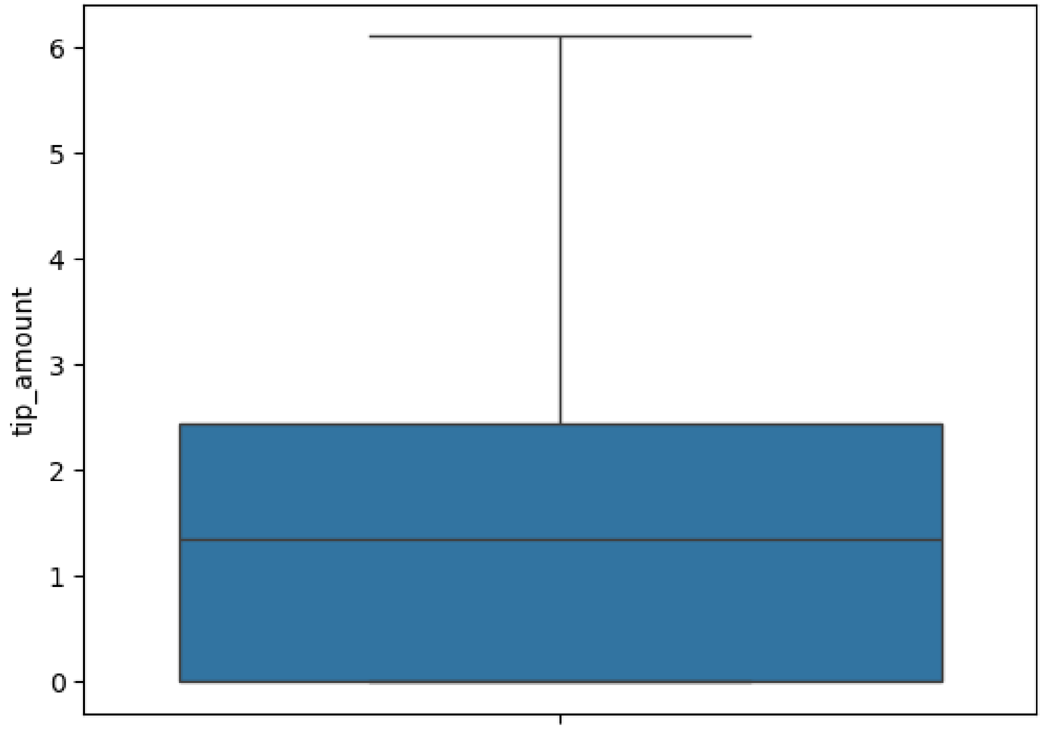
```
<Axes: xlabel='total_amount', ylabel='Count'>
```



tip amount

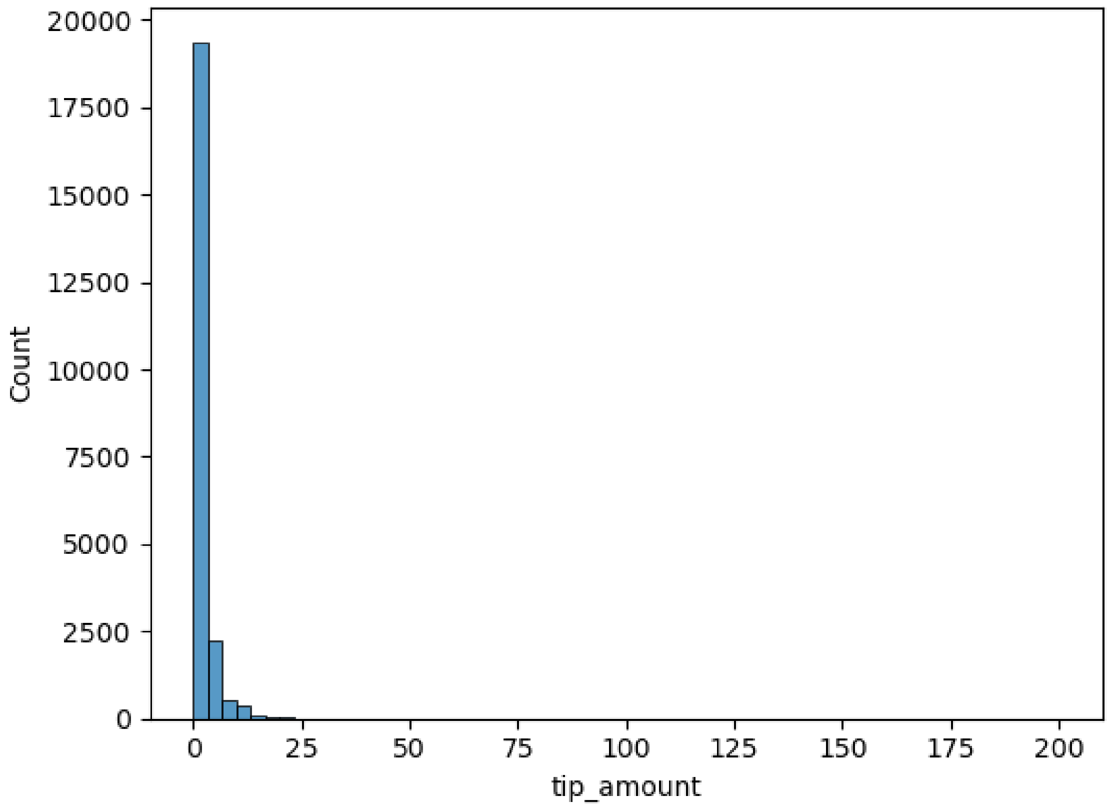
```
sns.boxplot(df['tip_amount'], showfliers=False)
```

```
<Axes: ylabel='tip_amount'>
```




```
sns.histplot(df['tip_amount'], bins=60)
```

```
<Axes: xlabel='tip_amount', ylabel='Count'>
```

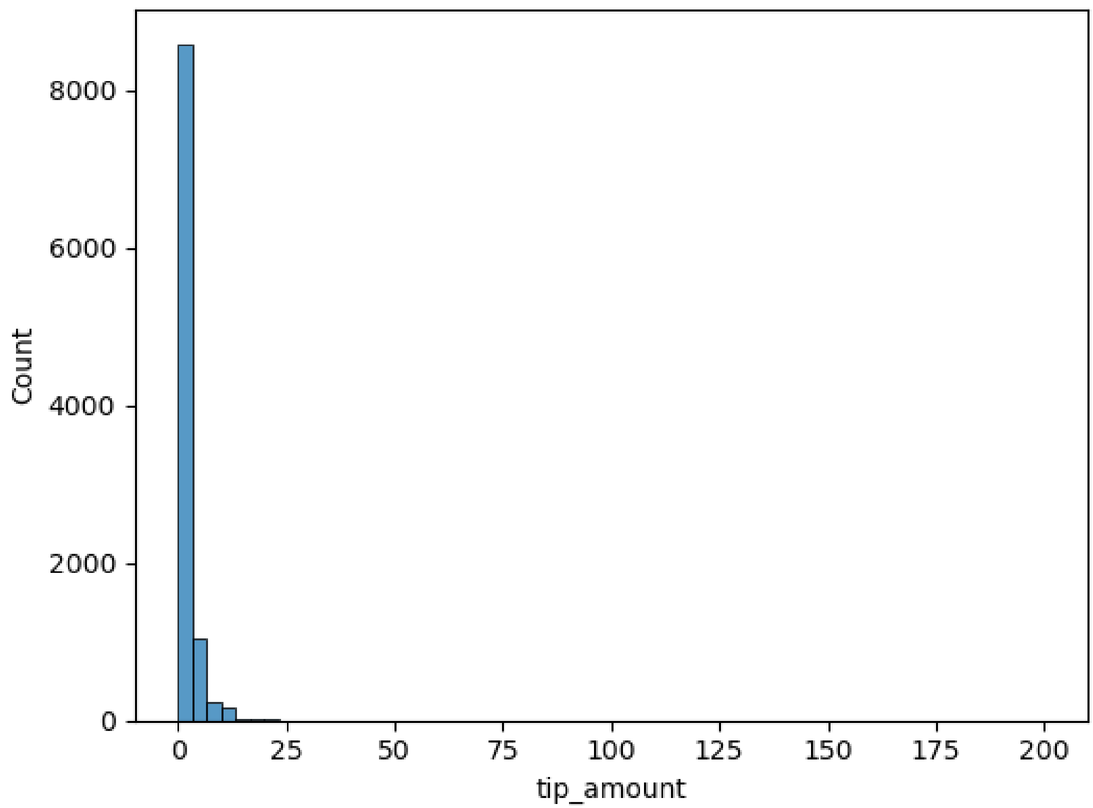


tip_amount by vendor

```
# Create histogram of tip_amount by vendor
mask1 = df['VendorID']==1
df_mask1 = df[mask1]
mask2 = df['VendorID']==2
df_mask2 = df[mask2]
```

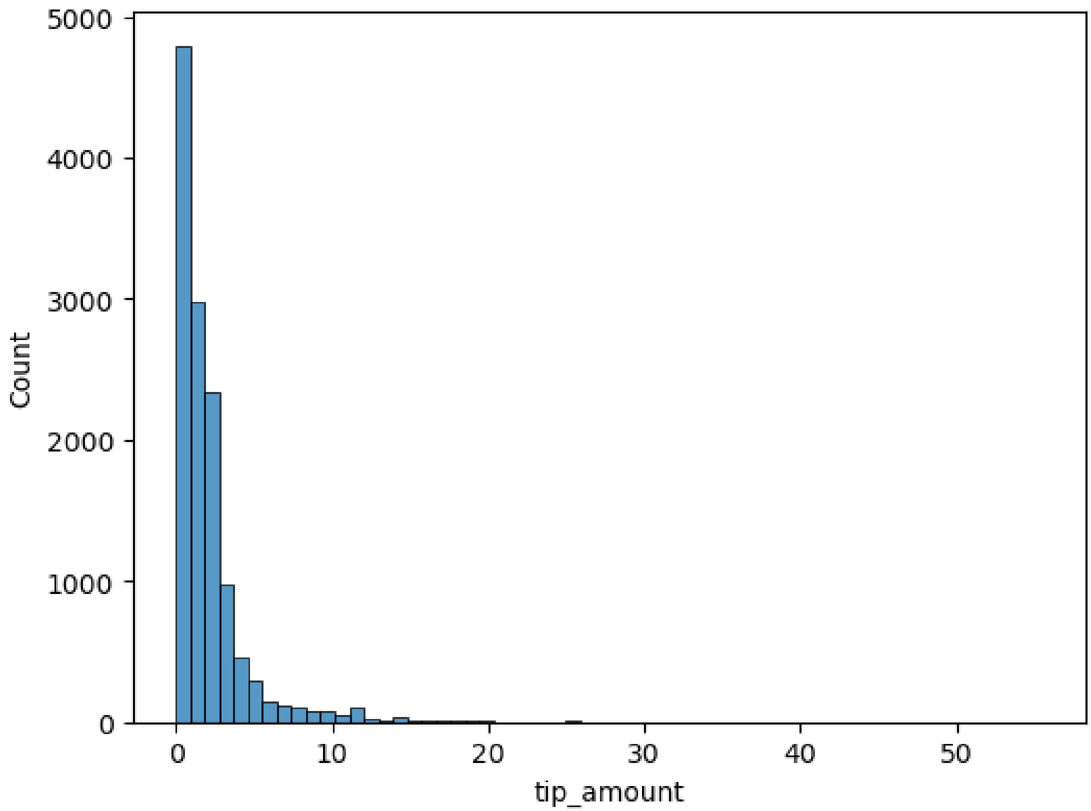
```
sns.histplot(df_mask1['tip_amount'], bins=60)
```

```
<Axes: xlabel='tip_amount', ylabel='Count'>
```




```
sns.histplot(df_mask2['tip_amount'], bins=60)
```

```
<Axes: xlabel='tip_amount', ylabel='Count'>
```

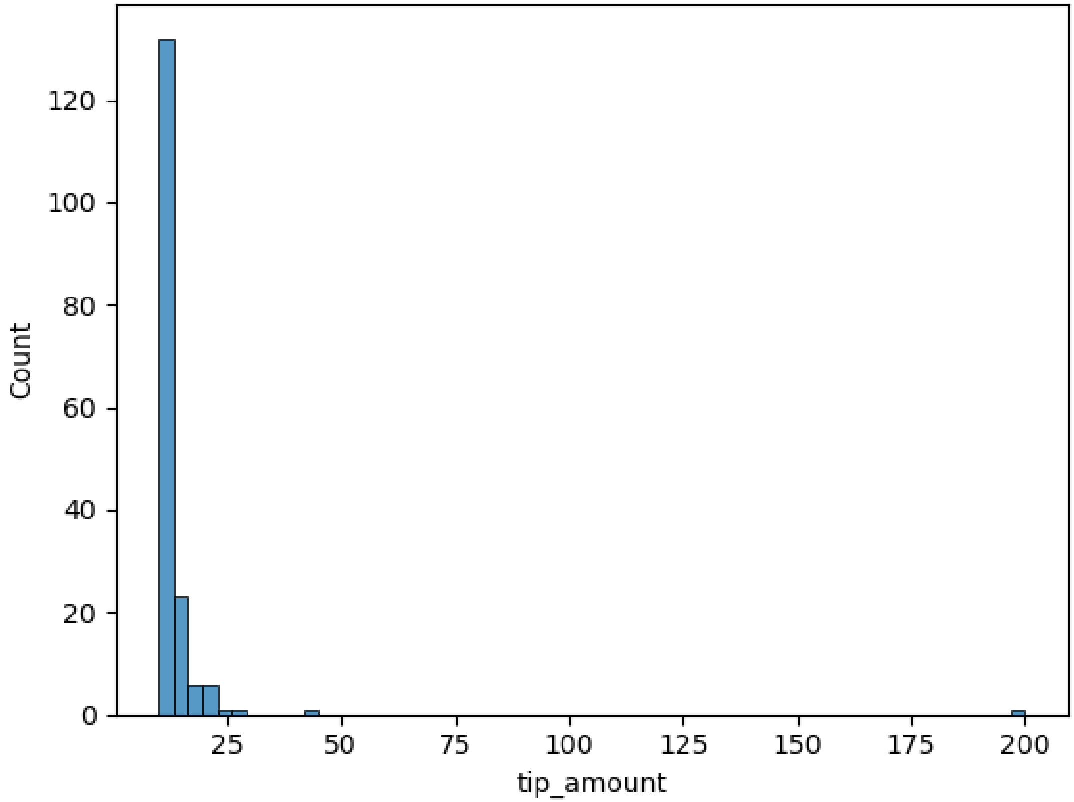


Next, zoom in on the upper end of the range of tips to check whether vendor one gets noticeably more of the most generous tips.

```
# Create histogram of tip_amount by vendor for tips > $10
tips1 = df_mask1[df_mask1['tip_amount']>10]
tips2 = df_mask2[df_mask2['tip_amount']>10]
```

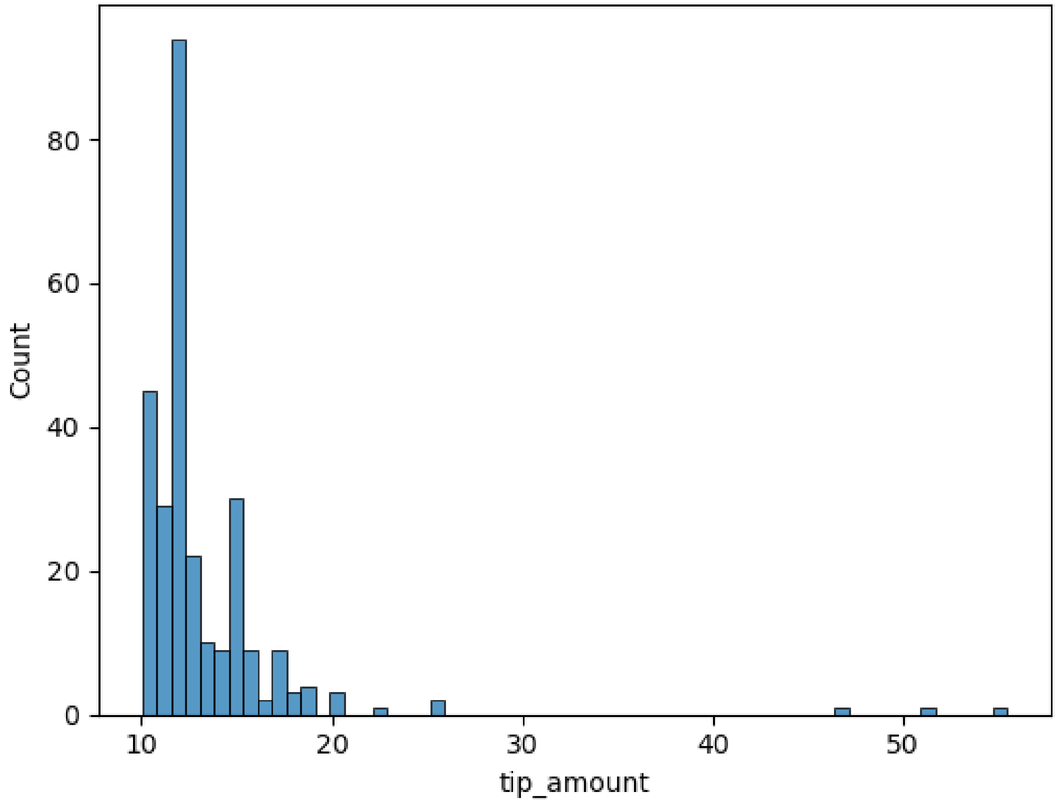
```
sns.histplot(tips1['tip_amount'], bins=60)
```

```
<Axes: xlabel='tip_amount', ylabel='Count'>
```




```
sns.histplot(tips2['tip_amount'], bins=60)
```

```
<Axes: xlabel='tip_amount', ylabel='Count'>
```



Mean tips by passenger count

Examine the unique values in the `passenger_count` column.

```
passengers = df['passenger_count'].unique()
passengers
```

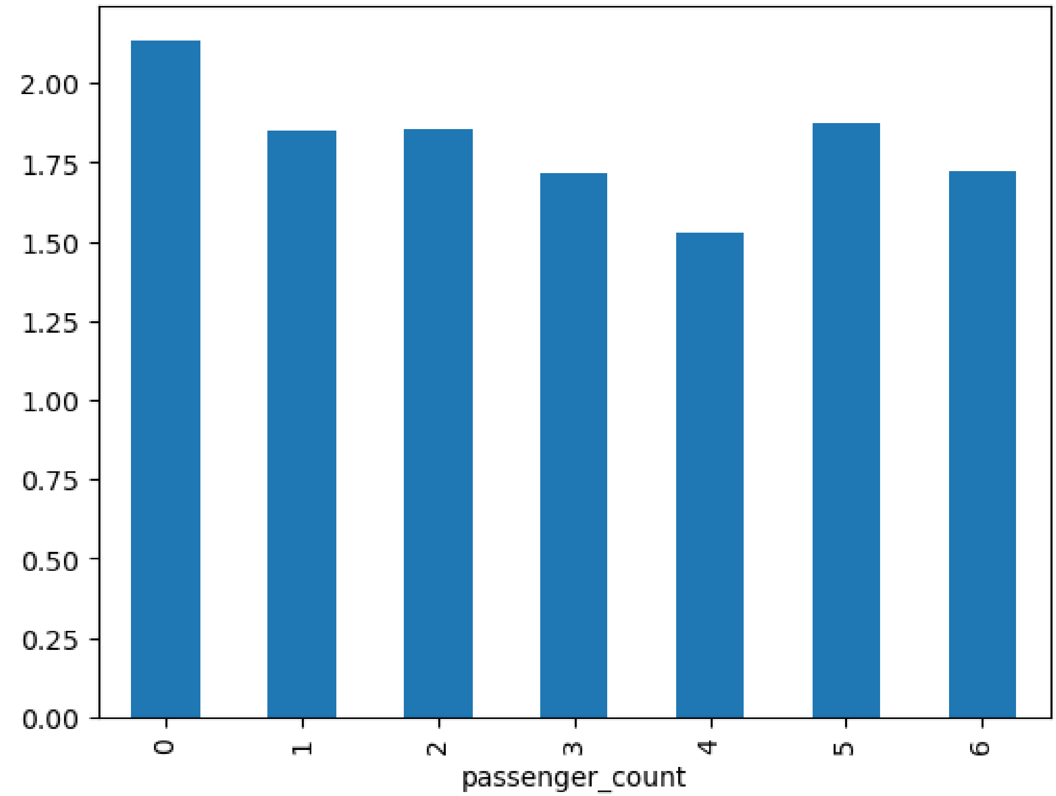
```
array([6, 1, 2, 4, 5, 3, 0])
```

```
# Calculate mean tips by passenger_count
mean_tips = df.groupby('passenger_count')['tip_amount'].mean()
mean_tips
```

```
passenger_count
0    2.135758
1    1.848920
2    1.856378
3    1.716768
4    1.530264
5    1.873185
6    1.720260
Name: tip_amount, dtype: float64
```

```
# Create bar plot for mean tips by passenger count
mean_tips.plot.bar()
```

```
<Axes: xlabel='passenger_count'>
```



Create month and day columns

```
import datetime as dt
```

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

```
# Create a month column
df['Month'] = df['tpep_pickup_datetime'].dt.month_name()
# Create a day column
df['Day'] = df['tpep_pickup_datetime'].dt.day_name()
```

```
# Create a month column
df['Month'] = df['tpep_pickup_datetime'].dt.month

# Create a day column
df['Day'] = df['tpep_pickup_datetime'].dt.day
```

```
df
```

```
       Unnamed: 0  VendorID tpep_pickup_datetime tpep_dropoff_datetime  \
```

(continues on next page)

(continued from previous page)

```
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   14873857         2  2017-02-24 17:37:23   2017-02-24 17:40:39
22695   66632549         2  2017-08-06 16:43:59   2017-08-06 17:24:47
22696   74239933         2  2017-09-04 14:54:14   2017-09-04 14:58:22
22697   60217333         2  2017-07-15 12:56:30   2017-07-15 13:08:26
22698   17208911         1  2017-03-02 13:02:49   2017-03-02 13:16:09

       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            48           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  Month  \
0            2.76          0.00                    0.3         16.56      3
1            4.00          0.00                    0.3         20.80      4
2            1.45          0.00                    0.3          8.75     12
3            6.39          0.00                    0.3         27.69      5
4            0.00          0.00                    0.3         17.80      4
...           ...           ...                    ...           ...    ...
22694        0.00          0.00                    0.3          5.80      2
22695       14.64          5.76                    0.3         73.20      8
22696        0.00          0.00                    0.3          5.30      9
22697        1.70          0.00                    0.3         13.00      7
22698        2.35          0.00                    0.3         14.15      3

       Day
```

(continues on next page)

(continued from previous page)

```
0        25
1        11
2        15
3         7
4        15
...     ...
22694    24
22695     6
22696     4
22697    15
22698     2

[22699 rows x 20 columns]
```

Plot total ride count by month

Begin by calculating total ride count by month.

```
# Get total number of rides for each month
monthly_rides = df['Month'].value_counts()
monthly_rides_df = monthly_rides.to_frame().reset_index()
monthly_rides_df.columns = ['Month', 'Count']
monthly_rides_df.sort_values(by='Month',inplace=True)
```

Reorder the results to put the months in calendar order.

```
# Reorder the monthly ride list so months go in order
monthly_rides_df
```

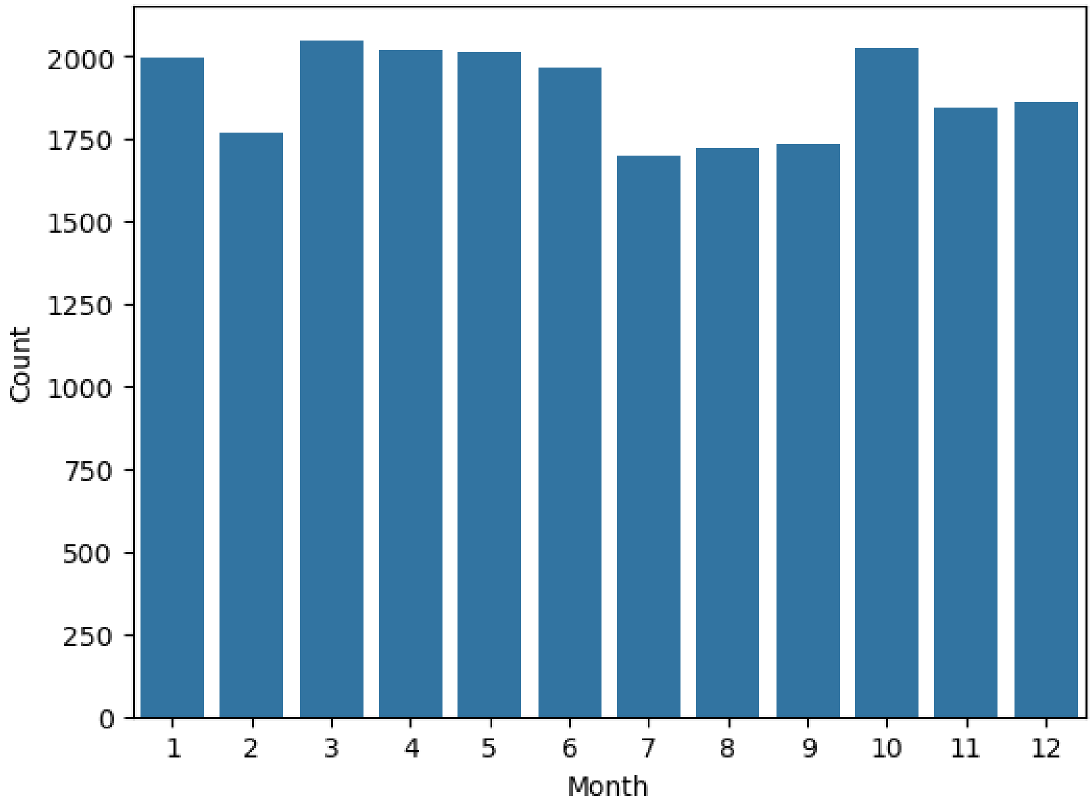
```
    Month  Count
4       1   1997
8       2   1769
0       3   2049
2       4   2019
3       5   2013
5       6   1964
11      7   1697
10      8   1724
9       9   1734
1      10   2027
7      11   1843
6      12   1863
```

```
# Show the index
monthly_rides_df.index
```

```
Index([4, 8, 0, 2, 3, 5, 11, 10, 9, 1, 7, 6], dtype='int64')
```

```
# Create a bar plot of total rides per month
sns.barplot(x='Month', y='Count', data=monthly_rides_df)
```

```
<Axes: xlabel='Month', ylabel='Count'>
```



Plot total ride count by day

Repeat the above process, but now calculate the total rides by day of the week.

```
daily_rides = df['Day'].value_counts()
daily_rides_df = daily_rides.to_frame().reset_index()
daily_rides_df.columns = ['Day', 'Count']
daily_rides_df.sort_values(by='Day',inplace=True)
```

```
# Repeat the above process, this time for rides by day
daily_rides_df
```

```
    Day  Count
20    1    721
16    2    739
0     3    810
25    4    703
14    5    746
21    6    718
5     7    788
13    8    752
22    9    714
11   10    760
4    11    788
```

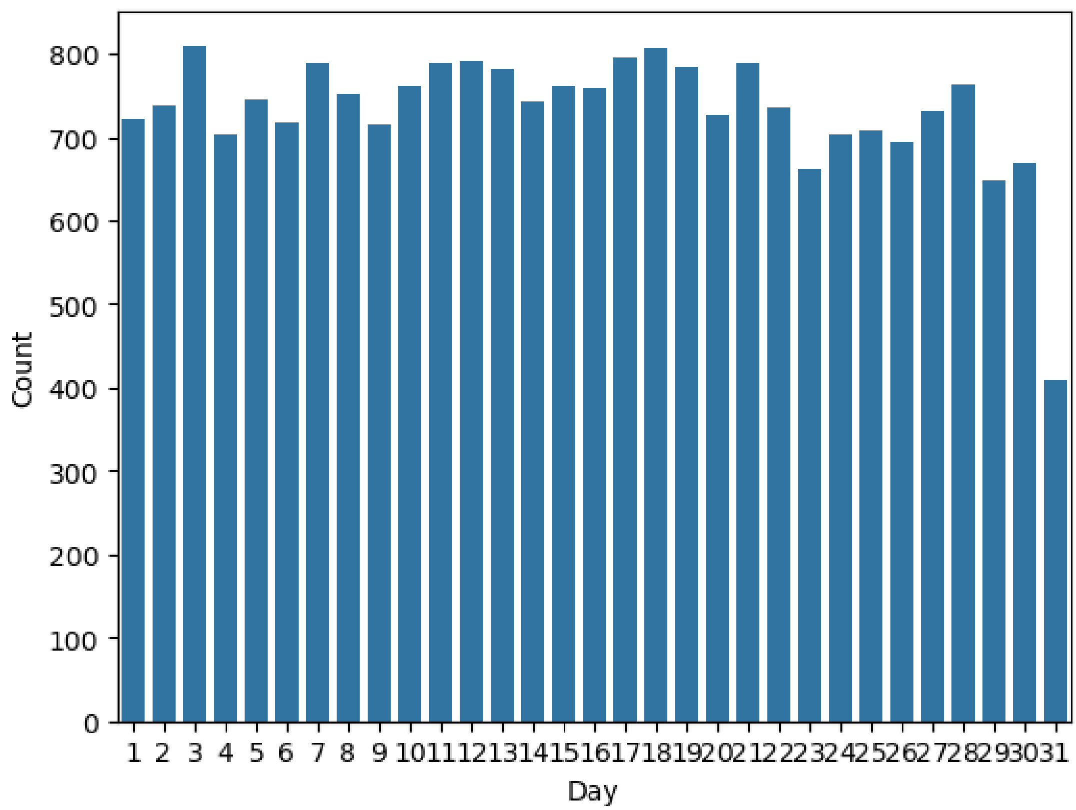
(continues on next page)

(continued from previous page)

```
3    12    791
8    13    782
15   14    742
10   15    761
12   16    759
2    17    795
1    18    806
7    19    784
19   20    727
6    21    788
17   22    735
28   23    663
24   24    703
23   25    708
26   26    695
18   27    732
9    28    763
29   29    649
27   30    669
30   31    408
```

```
# Create bar plot for ride count by day
sns.barplot(x='Day', y='Count', data=daily_rides_df)
```

```
<Axes: xlabel='Day', ylabel='Count'>
```



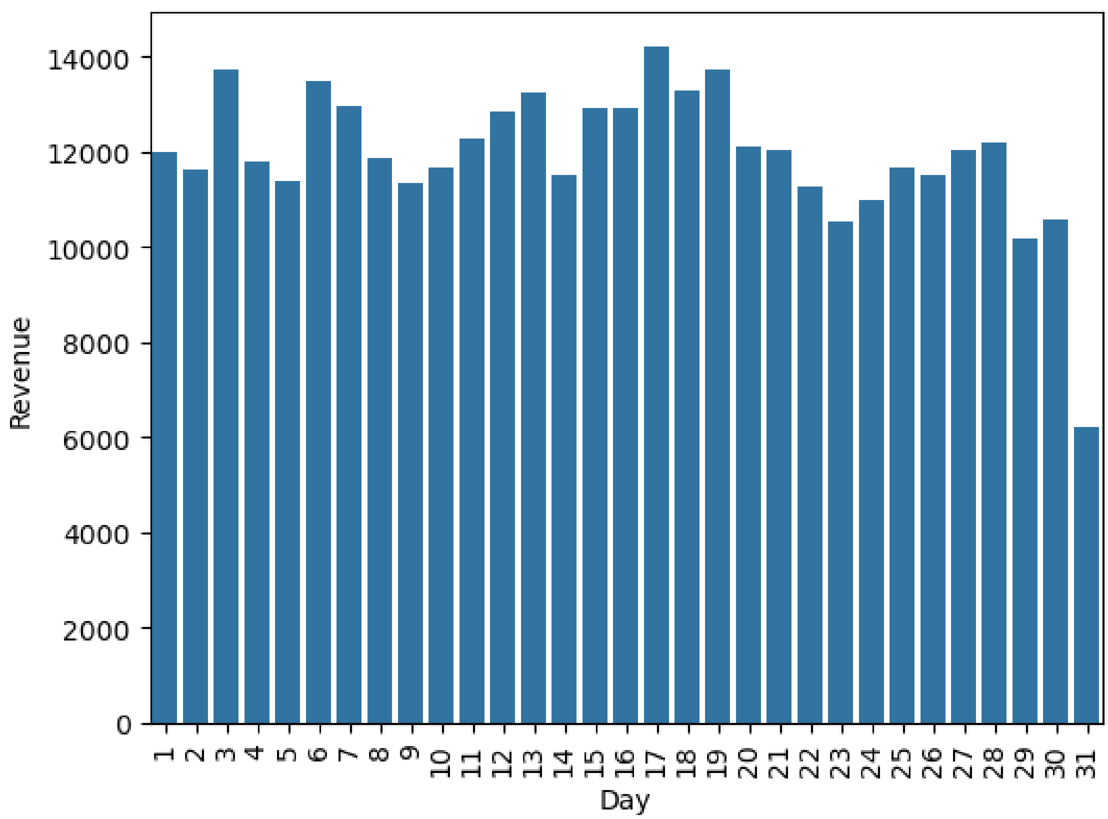
Plot total revenue by day of the week

Repeat the above process, but now calculate the total revenue by day of the week.

```
# Repeat the process, this time for total revenue by day
revenue = df.groupby('Day')['total_amount'].sum()
revenue = list(revenue)
```

```
daily_rides_df['Revenue'] = revenue
```

```
# Create bar plot of total revenue by day
sns.barplot(x='Day',y='Revenue',data=daily_rides_df)
plt.xticks(rotation=90)
plt.show()
```

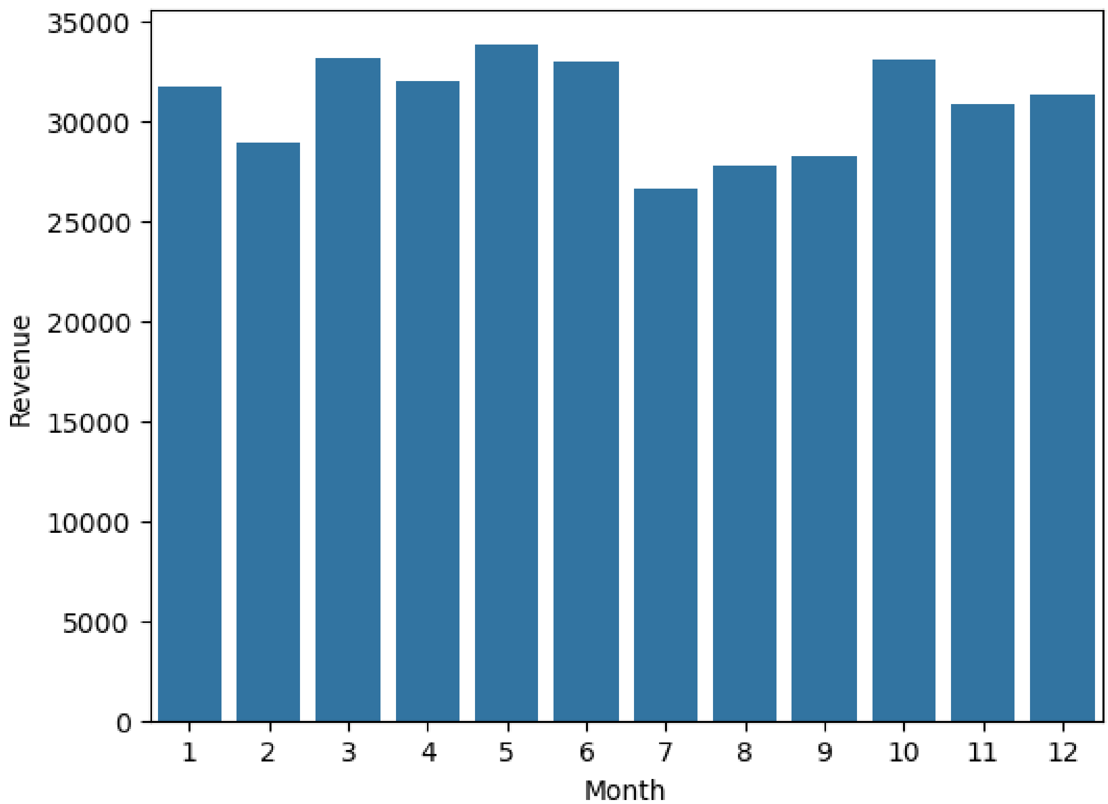


Plot total revenue by month

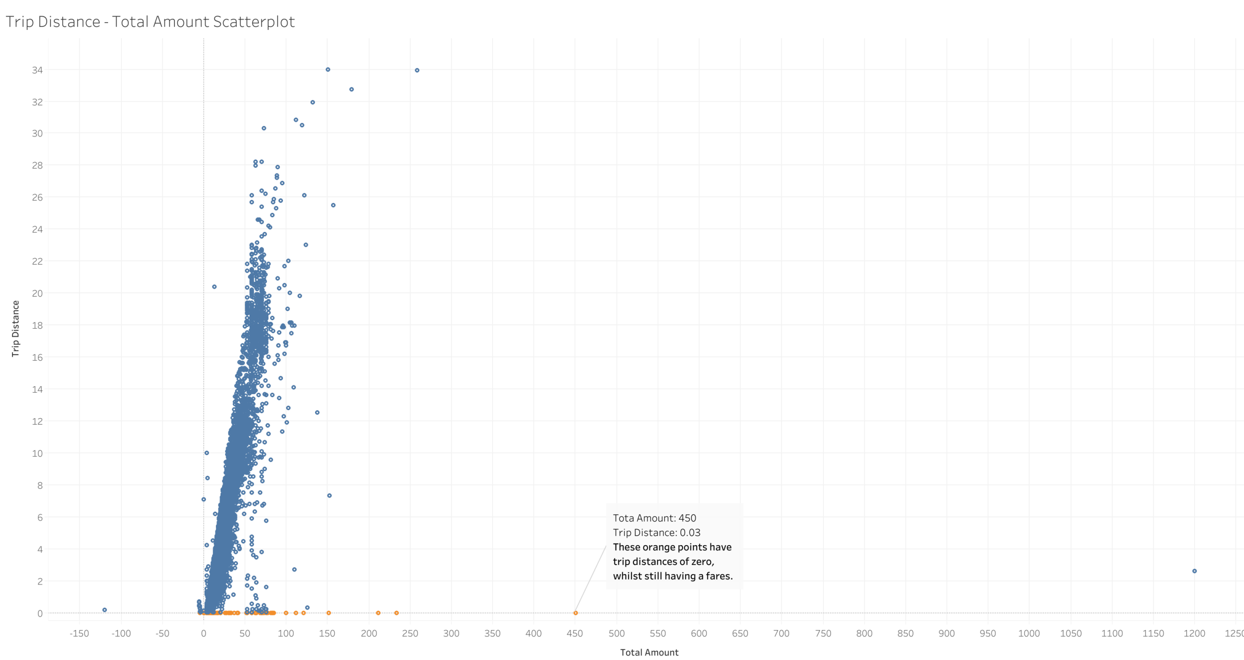
```
# Repeat the process, this time for total revenue by month
revenue_month = df.groupby('Month')['total_amount'].sum()
revenue_month = list(revenue_month)
```

```
monthly_rides_df['Revenue'] = revenue_month
```

```
# Create a bar plot of total revenue by month
sns.barplot(x='Month', y='Revenue', data=monthly_rides_df)
plt.show()
```



Scatter plot from Tableau



CHAPTER TWO

STATISTICAL ANALYSIS

You are a data professional in a data consulting firm, called Automatidata. The current project for their newest client, the New York City Taxi & Limousine Commission (New York City TLC) is reaching its midpoint, having completed a project proposal, Python coding work, and exploratory data analysis.

You receive a new email from Uli King, Automatidata's project manager. Uli tells your team about a new request from the New York City TLC: to analyze the relationship between fare amount and payment type. A follow-up email from Luana includes your specific assignment: to conduct an A/B test.

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

```
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Load dataset into dataframe
taxi = pd.read_csv("2017_Yellow_Taxi_Trip_Data.csv", index_col = 0)
```

```
taxi.head()
```

```
           VendorID    tpep_pickup_datetime   tpep_dropoff_datetime  \
24870114          2   03/25/2017 8:55:43 AM   03/25/2017 9:09:47 AM
35634249          1   04/11/2017 2:53:28 PM   04/11/2017 3:19:58 PM
106203690         1   12/15/2017 7:26:56 AM   12/15/2017 7:34:08 AM
38942136          2   05/07/2017 1:17:59 PM   05/07/2017 1:48:14 PM
30841670          2  04/15/2017 11:32:20 PM  04/15/2017 11:49:03 PM

           passenger_count  trip_distance  RatecodeID store_and_fwd_flag  \
24870114                 6           3.34           1                  N
35634249                 1           1.80           1                  N
106203690                1           1.00           1                  N
38942136                 1           3.70           1                  N
30841670                 1           4.37           1                  N

           PULocationID  DOLocationID  payment_type  fare_amount  extra  \
24870114            100           231             1         13.0    0.0
35634249            186            43             1         16.0    0.0
106203690           262           236             1          6.5    0.0
38942136            188            97             1         20.5    0.0
30841670              4           112             2         16.5    0.5
```

(continues on next page)

(continued from previous page)

```
           mta_tax  tip_amount  tolls_amount  improvement_surcharge  \
24870114       0.5        2.76           0.0                    0.3
35634249       0.5        4.00           0.0                    0.3
106203690      0.5        1.45           0.0                    0.3
38942136       0.5        6.39           0.0                    0.3
30841670       0.5        0.00           0.0                    0.3

           total_amount
24870114          16.56
35634249          20.80
106203690          8.75
38942136          27.69
30841670          17.80
```

```
taxi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 22699 entries, 24870114 to 17208911
Data columns (total 17 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   VendorID               22699 non-null  int64
 1   tpep_pickup_datetime   22699 non-null  object
 2   tpep_dropoff_datetime  22699 non-null  object
 3   passenger_count        22699 non-null  int64
 4   trip_distance          22699 non-null  float64
 5   RatecodeID             22699 non-null  int64
 6   store_and_fwd_flag     22699 non-null  object
 7   PULocationID           22699 non-null  int64
 8   DOLocationID           22699 non-null  int64
 9   payment_type           22699 non-null  int64
 10  fare_amount            22699 non-null  float64
 11  extra                  22699 non-null  float64
 12  mta_tax                22699 non-null  float64
 13  tip_amount             22699 non-null  float64
 14  tolls_amount           22699 non-null  float64
 15  improvement_surcharge  22699 non-null  float64
 16  total_amount           22699 non-null  float64
dtypes: float64(8), int64(6), object(3)
memory usage: 3.1+ MB
```

```
taxi['fare_amount'].groupby(by=taxi['payment_type']).mean()
```

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

Before you conduct your hypothesis test, consider the following questions where applicable to complete your code response:

1. Recall the difference between the null hypothesis and the alternative hypotheses. Consider your hypotheses for this project as listed below.

H_0: There is no difference in the average fare amount between customers who use credit cards and customers who use cash.

H_A: There is a difference in the average fare amount between customers who use credit cards and customers who use cash.

Your goal in this step is to conduct a two-sample t-test. Recall the steps for conducting a hypothesis test:

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

Note: For the purpose of this exercise, your hypothesis test is the main component of your A/B test.

You choose 5% as the significance level and proceed with a two-sample t-test.

```
credit_card = taxi[taxi['payment_type'] == 1]
```

```
cash = taxi[taxi['payment_type'] == 2]
```

```
stats.ttest_ind(credit_card['fare_amount'], cash['fare_amount'], equal_var=False)
```

```
TtestResult(statistic=6.866800855655372, pvalue=6.797387473030518e-12, df=16675.
↪48547403633)
```

The p-value is lower than the significance level and thus we reject the null hypothesis.

---

1. The key business insight is that encouraging customers to pay with credit cards can generate more revenue for taxi cab drivers.
2. This project requires an assumption that passengers were forced to pay one way or the other, and that once informed of this requirement, they always complied with it. The data was not collected this way; so, an assumption had to be made to randomly group data entries to perform an A/B test. This dataset does not account for other likely explanations. For example, riders might not carry lots of cash, so it's easier to pay for longer/farther trips with a credit card. In other words, it's far more likely that fare amount determines payment type, rather than vice versa.

CHAPTER

THREE

REGRESSION ANALYSIS

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.

```
# Imports
# Packages for numerics + dataframes
import numpy as np
import pandas as pd

# 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 timedelta
from datetime import date

# 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
```

```
sns.set_theme(context='notebook',
              style='dark' ,
              palette='deep')
```

3.1 EDA

```
# Load dataset into dataframe
df0=pd.read_csv("2017_Yellow_Taxi_Trip_Data.csv")
```

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

```
(22699, 18)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Unnamed: 0             22699 non-null  int64  
 1   VendorID               22699 non-null  int64  
 2   tpep_pickup_datetime   22699 non-null  object 
 3   tpep_dropoff_datetime  22699 non-null  object 
 4   passenger_count        22699 non-null  int64  
 5   trip_distance          22699 non-null  float64
 6   RatecodeID             22699 non-null  int64  
 7   store_and_fwd_flag     22699 non-null  object 
 8   PULocationID           22699 non-null  int64  
 9   DOLocationID           22699 non-null  int64  
 10  payment_type           22699 non-null  int64  
 11  fare_amount            22699 non-null  float64
 12  extra                  22699 non-null  float64
 13  mta_tax                22699 non-null  float64
 14  tip_amount             22699 non-null  float64
 15  tolls_amount           22699 non-null  float64
 16  improvement_surcharge  22699 non-null  float64
 17  total_amount           22699 non-null  float64
dtypes: float64(8), int64(7), object(3)
memory usage: 3.1+ MB
None
```

```
# Check for missing data and duplicates using .isna() and .drop_duplicates()
print('Shape of df0 with no duplicates for comparison with full df0:' , df0.drop_
↪duplicates().shape)
print()
print(df0.isna().sum())
```

```
Shape of df0 with no duplicates for comparison with full df0: (22699, 18)

Unnamed: 0               0
VendorID                 0
tpep_pickup_datetime     0
tpep_dropoff_datetime    0
passenger_count          0
```

(continues on next page)

(continued from previous page)

```
trip_distance            0
RatecodeID               0
store_and_fwd_flag       0
PULocationID             0
DOLocationID             0
payment_type             0
fare_amount              0
extra                    0
mta_tax                  0
tip_amount               0
tolls_amount             0
improvement_surcharge    0
total_amount             0
dtype: int64
```

```
df0.describe()
```

```
         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
```

(continues on next page)

(continued from previous page)

```
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
```

Convert pickup & dropoff columns to datetime

```
print(df0['tpep_pickup_datetime'][0:5])
print()
print('_' * 50)
print()
print(df0['tpep_dropoff_datetime'][0:5])
```

```
0     03/25/2017 8:55:43 AM
1     04/11/2017 2:53:28 PM
2     12/15/2017 7:26:56 AM
3     05/07/2017 1:17:59 PM
4    04/15/2017 11:32:20 PM
Name: tpep_pickup_datetime, dtype: object

__________________________________________________

0     03/25/2017 9:09:47 AM
1     04/11/2017 3:19:58 PM
2     12/15/2017 7:34:08 AM
3     05/07/2017 1:48:14 PM
4    04/15/2017 11:49:03 PM
Name: tpep_dropoff_datetime, dtype: object
```

```
# Convert datetime columns to datetime
df1 = df0.copy()

df1['tpep_pickup_datetime'] = pd.to_datetime(df1['tpep_pickup_datetime'])
df1['tpep_dropoff_datetime'] = pd.to_datetime(df1['tpep_dropoff_datetime'])
```

```
print(df1['tpep_pickup_datetime'][0:5])
print()
print('_' * 50)
print()
print(df1['tpep_dropoff_datetime'][0:5])
print()
print('_' * 50)
print()
print(df1['tpep_pickup_datetime'][0:5].dt.month)
print()
print('_' * 50)
print()
print(df1['tpep_dropoff_datetime'][0:5].dt.minute)
```

```
0   2017-03-25 08:55:43
1   2017-04-11 14:53:28
2   2017-12-15 07:26:56
3   2017-05-07 13:17:59
4   2017-04-15 23:32:20
Name: tpep_pickup_datetime, dtype: datetime64[ns]


__________________________________________________

0   2017-03-25 09:09:47
1   2017-04-11 15:19:58
2   2017-12-15 07:34:08
3   2017-05-07 13:48:14
4   2017-04-15 23:49:03
Name: tpep_dropoff_datetime, dtype: datetime64[ns]


__________________________________________________

0     3
1     4
2    12
3     5
4     4
Name: tpep_pickup_datetime, dtype: int32


__________________________________________________

0     9
1    19
2    34
3    48
4    49
Name: tpep_dropoff_datetime, dtype: int32
```

Create duration column

```
# Create `duration` column
df1['duration'] = df1['tpep_dropoff_datetime'] - df1['tpep_pickup_datetime']
df1['duration'].dtype
```

```
dtype('<m8[ns]')
```

```
df1['duration'] = df1['duration'].dt.total_seconds() / 60
```

```
df1['duration']
```

```
0        14.066667
1        26.500000
2         7.200000
3        30.250000
```

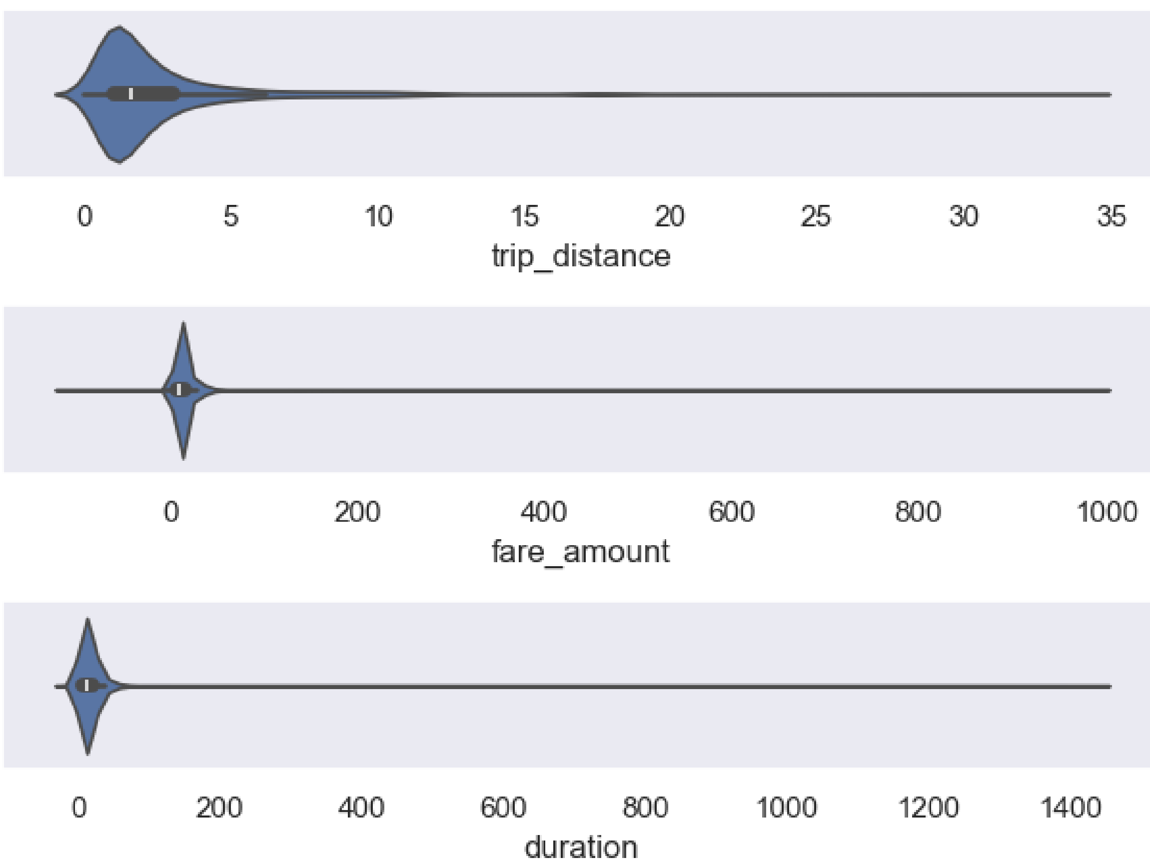
(continues on next page)

(continued from previous page)

```
4        16.716667
           ...
22694     3.266667
22695    40.800000
22696     4.133333
22697    11.933333
22698    13.333333
Name: duration, Length: 22699, dtype: float64
```

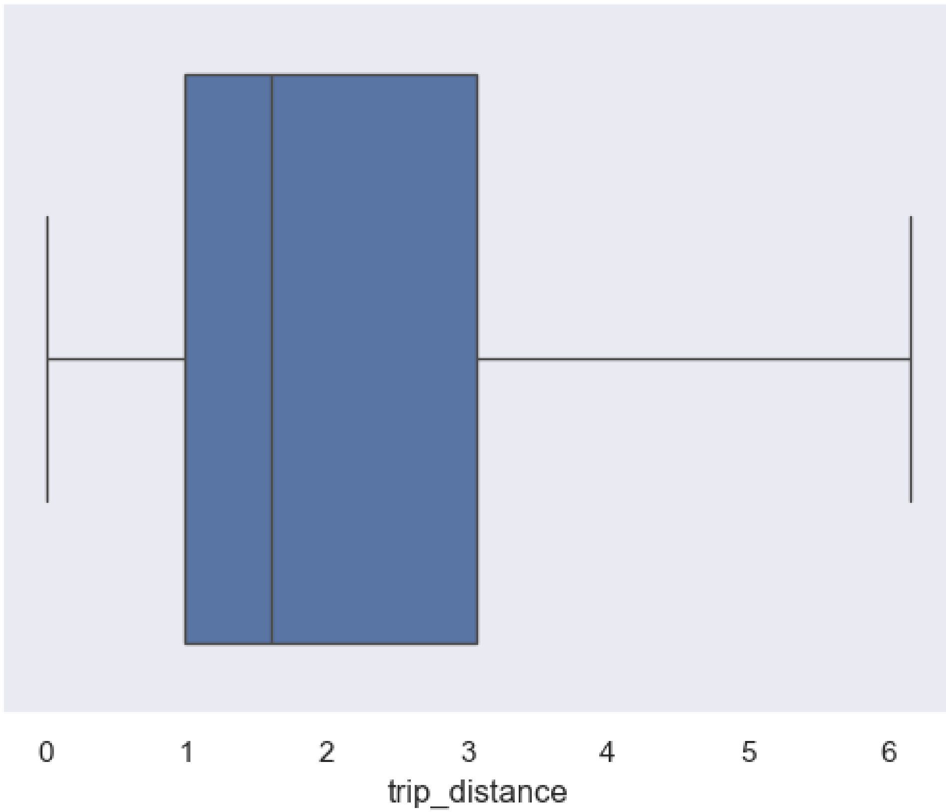
Check for Outliers

```
fig , axes = plt.subplots(3 , 1)

sns.violinplot(data=df1 , x='trip_distance' , ax=axes[0])
sns.violinplot(data=df1 , x='fare_amount' , ax=axes[1])
sns.violinplot(data=df1 , x='duration' , ax=axes[2])
plt.tight_layout()
plt.show()
```



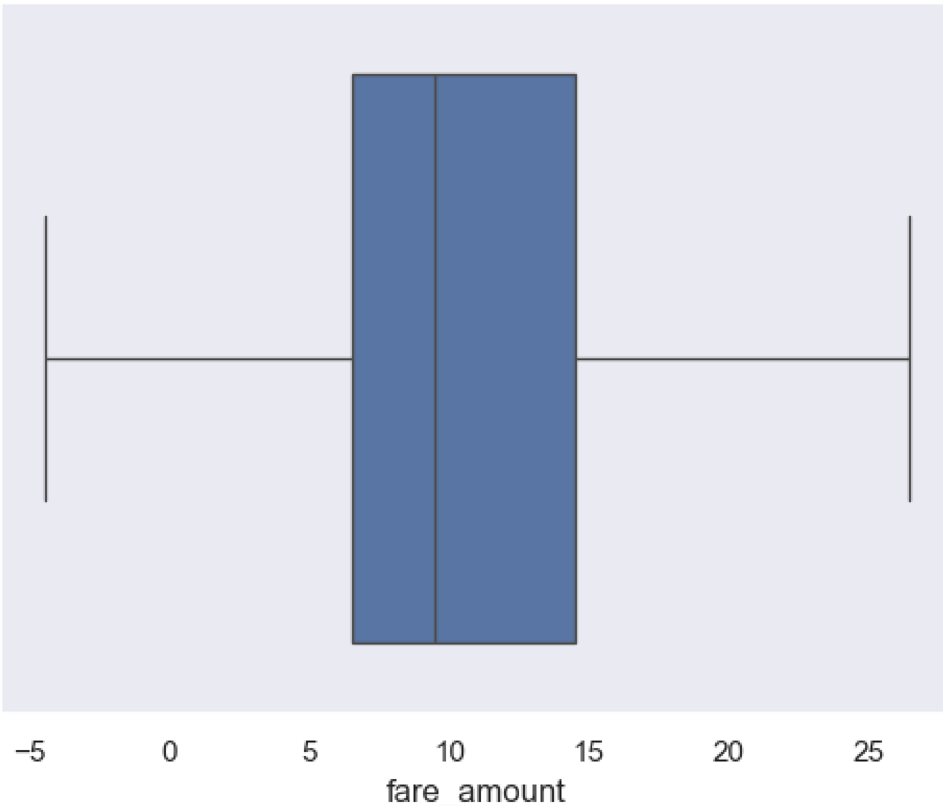

```
sns.boxplot(data=df1 , x='trip_distance' , showfliers=False)
```

```
<Axes: xlabel='trip_distance'>
```



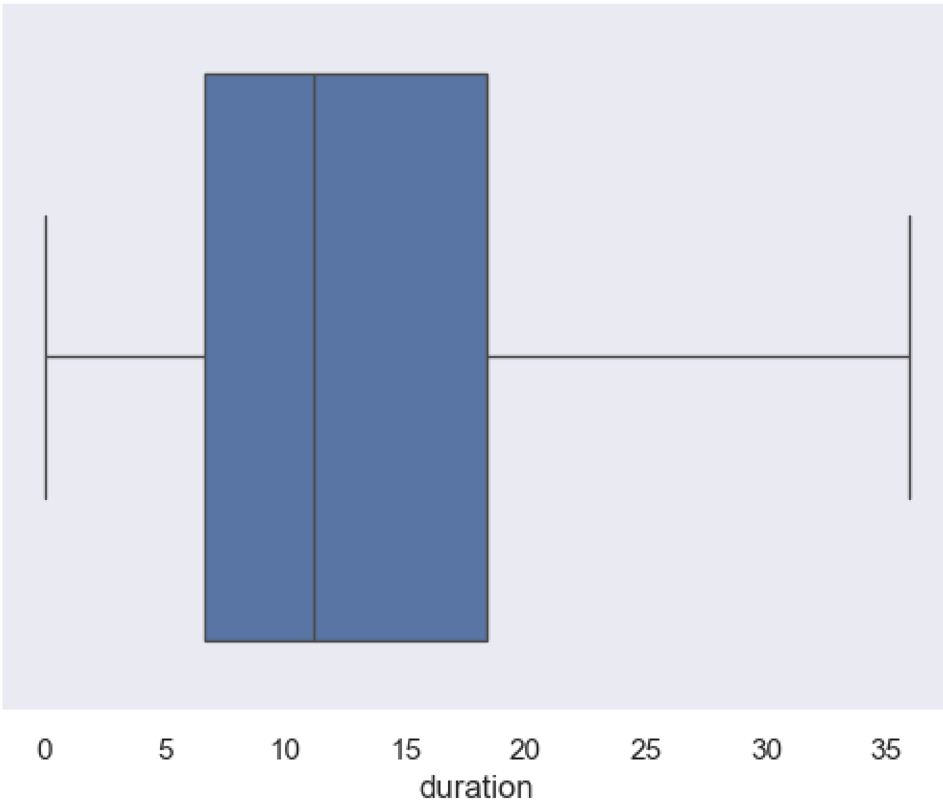

```
sns.boxplot(data=df1 , x='fare_amount' , showfliers=False)
```

```
<Axes: xlabel='fare_amount'>
```




```
sns.boxplot(data=df1 , x='duration' , showfliers=False)
```

```
<Axes: xlabel='duration'>
```




```
# Are trip distances of 0 bad data or very short trips rounded down?
df1['trip_distance'].sort_values(ascending=True)[:150]
```

```
22026    0.00
795      0.00
6908     0.00
13561    0.00
12238    0.00
         ...
7383     0.00
7281     0.00
9155     0.00
5501     0.01
19644    0.01
Name: trip_distance, Length: 150, dtype: float64
```

```
sum(df1['trip_distance'] == 0)
```

```
148
```

```
df1['fare_amount'].describe()
```

```
count    22699.000000
mean        13.026629
std         13.243791
```

(continues on next page)

(continued from previous page)

```
min       -120.000000
25%          6.500000
50%          9.500000
75%         14.500000
max        999.990000
Name: fare_amount, dtype: float64
```

```
# Impute values less than $0 with 0
df1['fare_amount'][df1['fare_amount'] < 0]
```

```
314       -2.5
1646      -2.5
4423      -3.0
5448      -3.5
5758      -2.5
8204      -3.5
10281     -2.5
11204     -4.5
12944   -120.0
14714     -4.0
17602     -4.0
18565     -3.0
20317     -3.5
20698     -4.5
Name: fare_amount, dtype: float64
```

```
df1.loc[df1['fare_amount'] < 0 , 'fare_amount'] = 0
df1['fare_amount'][df1['fare_amount'] < 0]
```

```
Series([], Name: fare_amount, dtype: float64)
```

```
df1['fare_amount'].describe()
```

```
count    22699.000000
mean        13.033832
std         13.212462
min          0.000000
25%          6.500000
50%          9.500000
75%         14.500000
max        999.990000
Name: fare_amount, dtype: float64
```

```
iqr = df1['fare_amount'].quantile(0.75) - df1['fare_amount'].quantile(0.25)
print(iqr)
maximum = df1['fare_amount'].quantile(0.75) + (6 * iqr)
print(maximum)
```

```
8.0
62.5
```

```
for fare in df1['fare_amount']:
        df1.loc[df1['fare_amount'] > maximum , 'fare_amount'] = maximum
```

```
print(df1['fare_amount'].min())
print(df1['fare_amount'].max())
```

```
0.0
62.5
```

```
# Call .describe() for duration outliers
df1['duration'].describe()
```

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

```
# Impute a 0 for any negative values
df1.loc[df1['duration'] < 0 , 'duration'] = 0
```

```
# Impute the high outliers
iqr = df1['duration'].quantile(0.75) - df1['duration'].quantile(0.25)
print(iqr)
maximum = df1['duration'].quantile(0.75) + (6 * iqr)
print(maximum)
```

```
11.733333333333333
88.78333333333333
```

```
for fare in df1['duration']:
        df1.loc[df1['duration'] > maximum , 'duration'] = maximum
```

```
print(df1['duration'].min())
print(df1['duration'].max())
```

```
0.0
88.78333333333333
```

3.2 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'

```
# Create `pickup_dropoff` column
df1['pickup_dropoff'] = df1['PULocationID'].astype(str) + ' ' + df1['DOLocationID'].
↪astype(str)
df1['pickup_dropoff'].head(2)
```

```
0    100 231
1     186 43
Name: pickup_dropoff, dtype: object
```

```
grouped = df1.groupby('pickup_dropoff').mean(numeric_only=True)[['trip_distance']]
grouped[:5]
```

```
                trip_distance
pickup_dropoff
1 1                  2.433333
10 148              15.700000
100 1               16.890000
100 100              0.253333
100 107              1.180000
```

```
# 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
df1['mean_distance'] = df1['pickup_dropoff']
# 2. Map `grouped_dict` to the `mean_distance` column
df1['mean_distance'] = df1['mean_distance'].map(grouped_dict)

# Confirm that it worked
df1[(df1['PULocationID'] == 100) & (df1['DOLocationID'] == 202)]
```

```
      Unnamed: 0  VendorID tpep_pickup_datetime tpep_dropoff_datetime  \
9304   112330762         1  2017-01-18 15:51:53   2017-01-18 16:31:14

      passenger_count  trip_distance  RatecodeID store_and_fwd_flag  \
9304                1            5.3           1                  N

      PULocationID  DOLocationID  ...  fare_amount  extra  mta_tax  \
9304           100           202  ...         26.0    0.0      0.5

      tip_amount  tolls_amount  improvement_surcharge  total_amount  duration  \
9304         2.0           0.0                    0.3          28.8     39.35

      pickup_dropoff mean_distance
9304         100 202           5.3

[1 rows x 21 columns]
```

```
grouped = df1.groupby('pickup_dropoff').mean(numeric_only=True)[['duration']]
grouped

# Create a dictionary where keys are unique pickup_dropoffs and values are
# mean trip duration for all trips with those pickup_dropoff combos
```

(continues on next page)

(continued from previous page)

```
grouped_dict = grouped.to_dict()
grouped_dict = grouped_dict['duration']

df1['mean_duration'] = df1['pickup_dropoff']
df1['mean_duration'] = df1['mean_duration'].map(grouped_dict)

# Confirm that it worked
df1[(df1['PULocationID']==100) & (df1['DOLocationID']==231)][['mean_duration']]
```

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

```
# Create 'day' col
df1['day'] = df1['tpep_pickup_datetime'].dt.day_name()

# Create 'month' col
df1['month'] = df1['tpep_pickup_datetime'].dt.strftime('%b')
```

```
df1
```

```
       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    14873857         2  2017-02-24 17:37:23   2017-02-24 17:40:39
22695    66632549         2  2017-08-06 16:43:59   2017-08-06 17:24:47
22696    74239933         2  2017-09-04 14:54:14   2017-09-04 14:58:22
22697    60217333         2  2017-07-15 12:56:30   2017-07-15 13:08:26
22698    17208911         1  2017-03-02 13:02:49   2017-03-02 13:16:09

       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
```

(continues on next page)

(continued from previous page)

```
       PULocationID  DOLocationID  ...  tip_amount  tolls_amount  \
0               100           231  ...        2.76          0.00   
1               186            43  ...        4.00          0.00   
2               262           236  ...        1.45          0.00   
3               188            97  ...        6.39          0.00   
4                 4           112  ...        0.00          0.00   
...             ...           ...  ...         ...           ...   
22694            48           186  ...        0.00          0.00   
22695           132           164  ...       14.64          5.76   
22696           107           234  ...        0.00          0.00   
22697            68           144  ...        1.70          0.00   
22698           239           236  ...        2.35          0.00   

       improvement_surcharge  total_amount   duration  pickup_dropoff  \
0                        0.3         16.56  14.066667         100 231   
1                        0.3         20.80  26.500000          186 43   
2                        0.3          8.75   7.200000         262 236   
3                        0.3         27.69  30.250000          188 97   
4                        0.3         17.80  16.716667           4 112   
...                      ...           ...        ...             ...   
22694                    0.3          5.80   3.266667          48 186   
22695                    0.3         73.20  40.800000         132 164   
22696                    0.3          5.30   4.133333         107 234   
22697                    0.3         13.00  11.933333          68 144   
22698                    0.3         14.15  13.333333         239 236   

       mean_distance  mean_duration       day month  
0           3.521667      22.847222  Saturday   Mar  
1           3.108889      24.470370   Tuesday   Apr  
2           0.881429       7.250000    Friday   Dec  
3           3.700000      30.250000    Sunday   May  
4           4.435000      14.616667  Saturday   Apr  
...              ...            ...       ...   ...  
22694       1.098214       8.594643    Friday   Feb  
22695      18.757500      59.560417    Sunday   Aug  
22696       0.684242       6.609091    Monday   Sep  
22697       2.077500      16.650000  Saturday   Jul  
22698       1.476970       9.405556  Thursday   Mar  

[22699 rows x 24 columns]
```

- 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.

```
# Create 'rush_hour' col
df1['rush_hour'] = df1['tpep_pickup_datetime'].dt.hour

# If day is Saturday or Sunday, impute 0 in `rush_hour` column
df1.loc[(df1['day'] == 'Saturday') | (df1['day'] == 'Sunday') , 'rush_hour'] = 0
```

```
df1.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  ...  tolls_amount  improvement_surcharge  \
0           100           231  ...           0.0                    0.3
1           186            43  ...           0.0                    0.3
2           262           236  ...           0.0                    0.3
3           188            97  ...           0.0                    0.3
4             4           112  ...           0.0                    0.3

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

        day  month rush_hour
0  Saturday    Mar         0
1   Tuesday    Apr        14
2    Friday    Dec         7
3    Sunday    May         0
4  Saturday    Apr         0

[5 rows x 25 columns]
```

```
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
```

```
# Apply the `rush_hourizer()` function to the new column
df1.loc[(df1['day'] != 'Saturday') & (df1.day != 'Sunday') , 'rush_hour'] = df1.
↪apply(rush_hourizer , axis=1)
```

```
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70633/1505293423.py:2:
↪FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an
↪error in a future version of pandas. Value '[0 1 1 ... 1 0 0]' has dtype incompatible
↪with int32, please explicitly cast to a compatible dtype first.
  df1.loc[(df1['day'] != 'Saturday') & (df1.day != 'Sunday') , 'rush_hour'] = df1.
↪apply(rush_hourizer , axis=1)
```

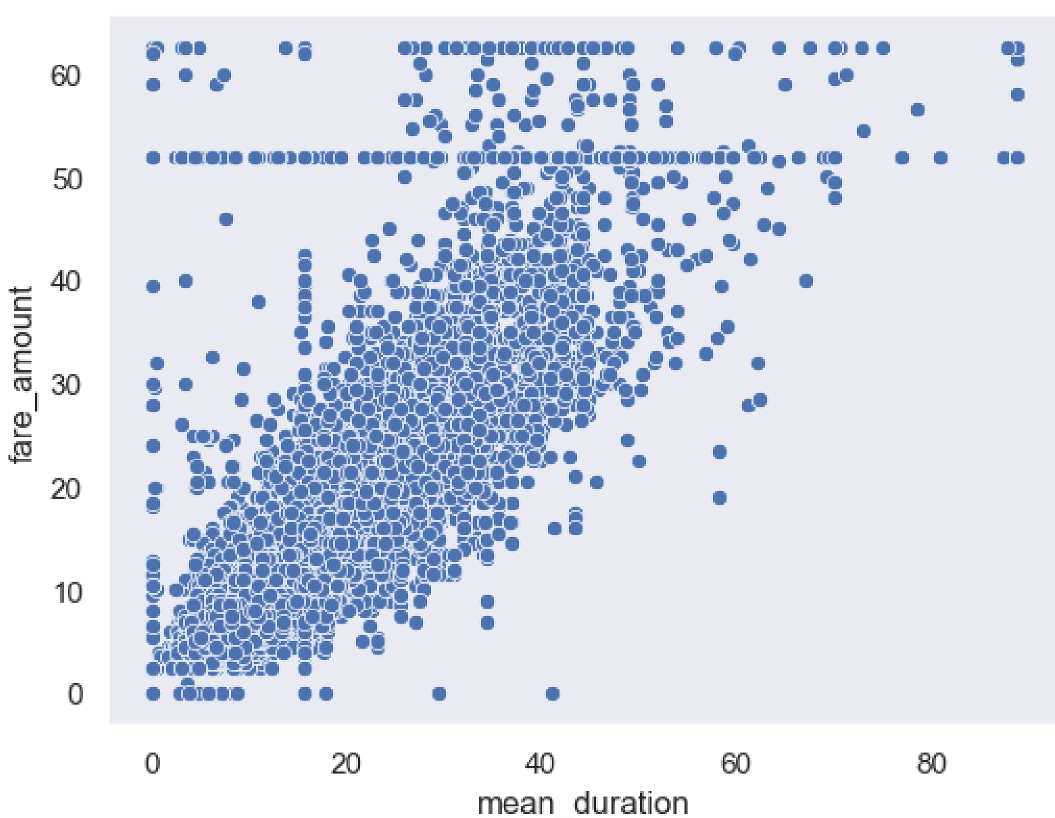
```
df1['rush_hour'].describe()
```

```
count    22699.000000
mean         0.296753
std          0.456837
min          0.000000
25%          0.000000
50%          0.000000
75%          1.000000
max          1.000000
Name: rush_hour, dtype: float64
```

Scatter Plot

```
# Create a scatterplot to visualize the relationship between variables of interest
sns.scatterplot(data = df1 , x='mean_duration' , y='fare_amount', markers='.')
```

```
<Axes: xlabel='mean_duration', ylabel='fare_amount'>
```



```
df1[df1['fare_amount'] > 50]['fare_amount'].value_counts()
```

```
fare_amount
52.0    514
62.5     84
59.0      9
50.5      9
57.5      8
51.0      7
60.0      6
55.0      6
51.5      6
53.0      4
52.5      4
61.0      3
62.0      3
55.5      3
56.0      3
56.5      3
58.5      2
59.5      2
61.5      2
57.0      2
54.0      2
58.0      1
54.7      1
54.5      1
Name: count, dtype: int64
```

```
# Set pandas to display all columns
pd.set_option('display.max_columns' , None)
df1[df1['fare_amount'] == 52].head(30)
```

```
      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
```

(continues on next page)

(continued from previous page)

```
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
```

(continues on next page)

(continued from previous page)

```
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
```

(continues on next page)

(continued from previous page)

```
1082       0.00          5.54                    0.3        62.84
1097       0.00          0.00                    0.3        52.80
1110      15.80          0.00                    0.3        68.60
1179      17.57          5.76                    0.3        76.13

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

      rush_hour
11            0
110           0
161           0
247           0
379           0
388           1
406           0
449           0
468           0
520           0
569           0
572           0
586           0
692           0
```

(continues on next page)

(continued from previous page)

```
717          0
719          1
782          1
816          1
818          1
835          0
840          0
861          0
881          0
958          0
970          0
984          1
1082         1
1097         0
1110         0
1179         1
```

Isolate modeling variables

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

```
df1.info()
```

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

(continues on next page)

(continued from previous page)

```
 23  month                   22699 non-null  object
 24  rush_hour               22699 non-null  int64
dtypes: datetime64[ns](2), float64(11), int64(8), object(4)
memory usage: 4.3+ MB
```

```
df1.describe()
```

```
         Unnamed: 0      VendorID           tpep_pickup_datetime  \
count  2.269900e+04  22699.000000                          22699
mean   5.675849e+07      1.556236  2017-06-29 07:32:48.973126656
min    1.212700e+04      1.000000            2017-01-01 00:08:25
25%    2.852056e+07      1.000000     2017-03-30 03:09:38.500000
50%    5.673150e+07      2.000000            2017-06-23 12:35:57
75%    8.537452e+07      2.000000            2017-10-02 10:34:34
max    1.134863e+08      2.000000            2017-12-31 23:45:30
std    3.274493e+07      0.496838                            NaN

              tpep_dropoff_datetime  passenger_count  trip_distance  \
count                         22699     22699.000000   22699.000000
mean   2017-06-29 07:49:49.799726848         1.642319       2.913313
min              2017-01-01 00:17:20         0.000000       0.000000
25%       2017-03-30 03:11:20.500000         1.000000       0.990000
50%              2017-06-23 12:55:11         1.000000       1.610000
75%              2017-10-02 10:53:47         2.000000       3.060000
max              2017-12-31 23:49:24         6.000000      33.960000
std                              NaN         1.285231       3.653171

         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     12.897913
min        1.000000      1.000000      1.000000      1.000000      0.000000
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     62.500000
std        0.708391     66.633373     70.139691      0.496211     10.541137

              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
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
std        0.463097      0.039465      2.800626      1.399212

       improvement_surcharge  total_amount      duration  mean_distance  \
count           22699.000000  22699.000000  22699.000000   22699.000000
mean                0.299551     16.310502     14.460555       2.913313
min                -0.300000   -120.300000      0.000000       0.000000
```

(continues on next page)

(continued from previous page)

```
25%                 0.300000      8.750000      6.650000       1.010000
50%                 0.300000     11.800000     11.183333       1.620000
75%                 0.300000     17.800000     18.383333       3.115625
max                 0.300000   1200.290000     88.783333      33.920000
std                 0.015673     16.097295     11.947043       3.558993

       mean_duration     rush_hour
count   22699.000000  22699.000000
mean       14.460555      0.296753
min         0.000000      0.000000
25%         8.031481      0.000000
50%        11.556667      0.000000
75%        17.321667      1.000000
max        88.783333      1.000000
std        10.080913      0.456837
```

```
df2 = df1.copy()
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 , inplace=True)
df2.head()
```

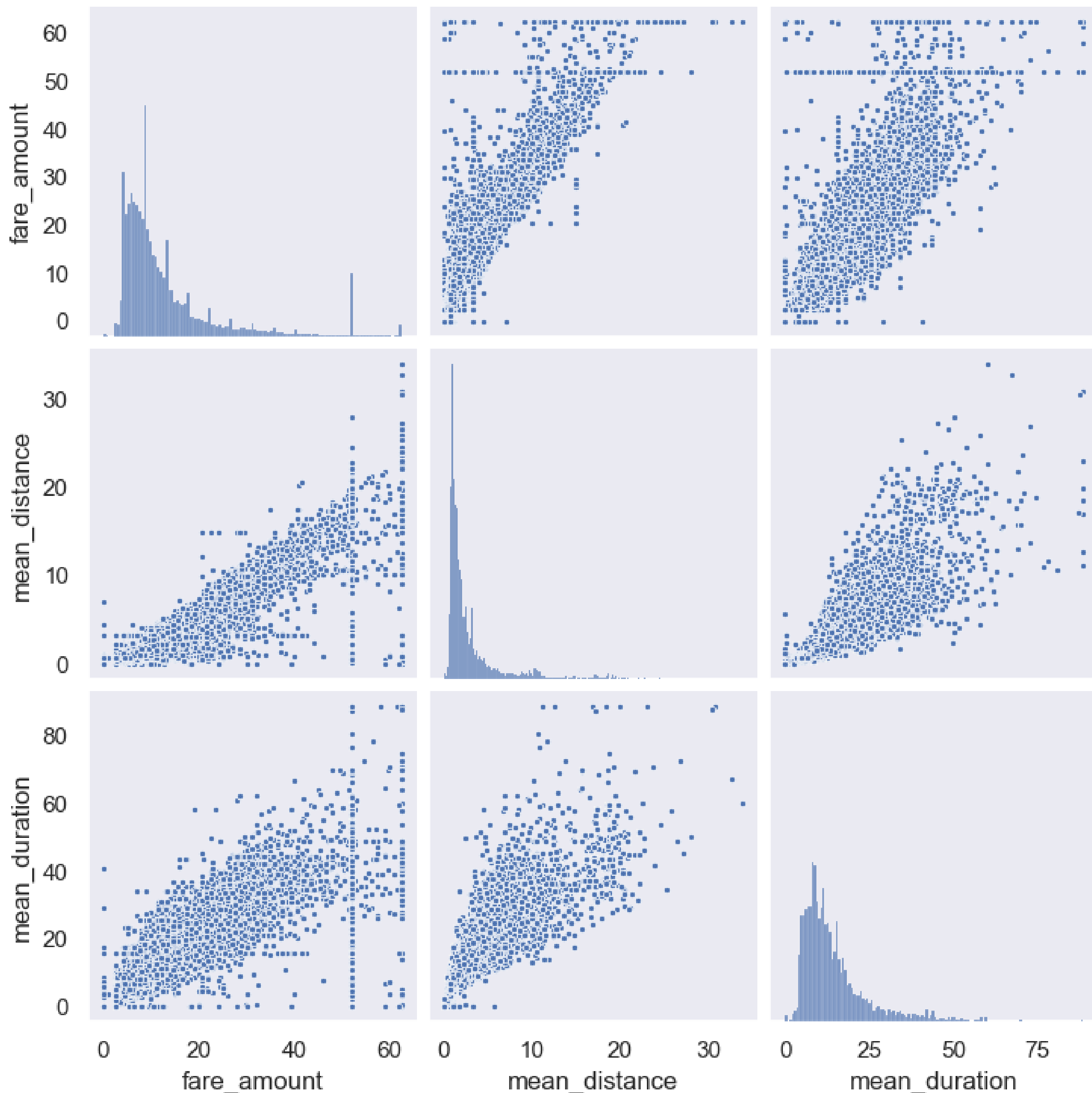
```
   VendorID  passenger_count  fare_amount  mean_distance  mean_duration  \
0         2                6         13.0       3.521667      22.847222
1         1                1         16.0       3.108889      24.470370
2         1                1          6.5       0.881429       7.250000
3         2                1         20.5       3.700000      30.250000
4         2                1         16.5       4.435000      14.616667

   rush_hour
0          0
1          0
2          1
3          0
4          0
```

Pair plot

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

```
# Create a pairplot to visualize pairwise relationships between variables in the data
sns.pairplot(df2[['fare_amount' , 'mean_distance' , 'mean_duration']] , markers='.' )
```

```
<seaborn.axisgrid.PairGrid at 0x13cae7710>
```



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

Identify correlations

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

```
# Correlation matrix to help determine most correlated variables
df2.corr()
```

```
                 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
```

(continues on next page)

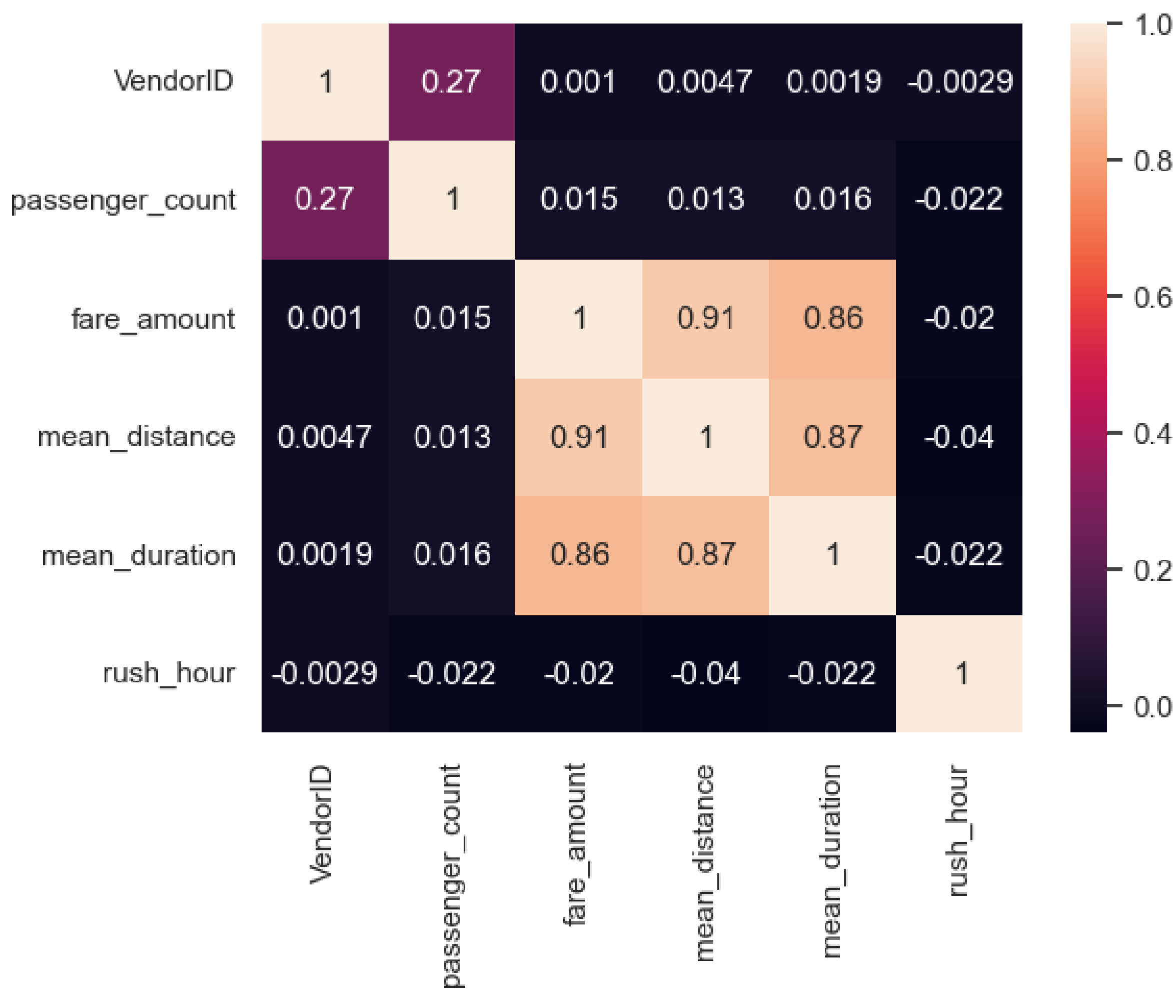
(continued from previous page)

```
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.

```
# Create correlation heatmap
sns.heatmap(df2.corr() , annot=True)
```

```
<Axes: >
```



Split data into outcome variable and features

```
X = df2.drop(columns='fare_amount')
y = df2[['fare_amount']]
```

```
print(X.head())
print()
print(y.head())
```

```
   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

   fare_amount
0         13.0
1         16.0
2          6.5
3         20.5
4         16.5
```

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

Pre-process data

Dummy encode categorical variables

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

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

```
X.head()
```

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

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`.

```
# 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)
```

Standardize the data

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

```
# Standardize the X variables
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
```

```
X_train_scaled
```

```
array([[-0.50301524,  0.8694684 ,  0.17616665, -0.64893329,  0.89286563],
       [-0.50301524, -0.60011281, -0.69829589,  1.54099045,  0.89286563],
       [ 0.27331093, -0.47829156, -0.57301906, -0.64893329, -1.11998936],
       ...,
       [-0.50301524, -0.45121122, -0.6788917 , -0.64893329, -1.11998936],
       [-0.50301524, -0.58944763, -0.85743597,  1.54099045, -1.11998936],
       [ 1.82596329,  0.83673851,  1.13212101, -0.64893329,  0.89286563]])
```

3.3 Model Estimation

Instantiate your model and fit it to the training data.

```
# Fit your model to the training data
model = LinearRegression()

model.fit(X_train_scaled , y_train)
```

```
LinearRegression()
```

Evaluate model

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.

```
y_pred_train = model.predict(X_train_scaled)
```

```
# Evaluate the model performance on the training data
print('R-squared' , model.score(X_train_scaled, y_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)))
```

```
R-squared 0.8398434585044773
MAE 2.1866664167754157
MSE 17.88973296349268
RMSE 4.229625629236313
```

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`.

```
X_test_scaled = scaler.transform(X_test)
y_pred_test = model.predict(X_test_scaled)
```

```
# Scale the X_test data
model.fit(X_test_scaled , y_test)
```

```
LinearRegression()
```

```
# Evaluate the model performance on the testing data
y_pred_test = model.predict(X_test_scaled)
print('R-squared' , model.score(X_test_scaled , y_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)))
```

```
R-squared 0.8688418048100106
MAE 2.1126188601915166
MSE 14.263006975751834
RMSE 3.7766396407059855
```

3.4 Results

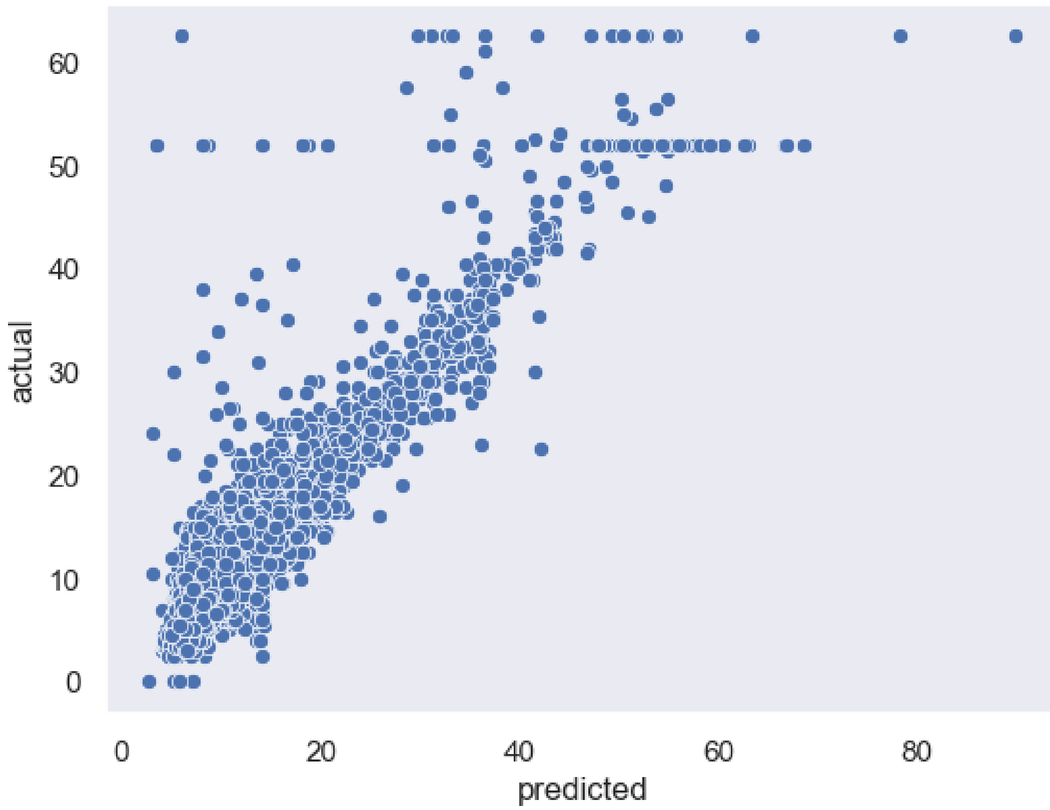
```
# Create a `results` dataframe
results = pd.DataFrame(data={'actual':y_test['fare_amount'],
                      'predicted':y_pred_test.ravel()})
results['residual'] = results['actual'] - results['predicted']
```

```
results.head()
```

```
       actual  predicted   residual
5818     14.0  12.346025   1.653975
18134    28.0  16.365952  11.634048
4655      5.5   6.587892  -1.087892
7378     15.5  16.309853  -0.809853
13914     9.5  10.291103  -0.791103
```

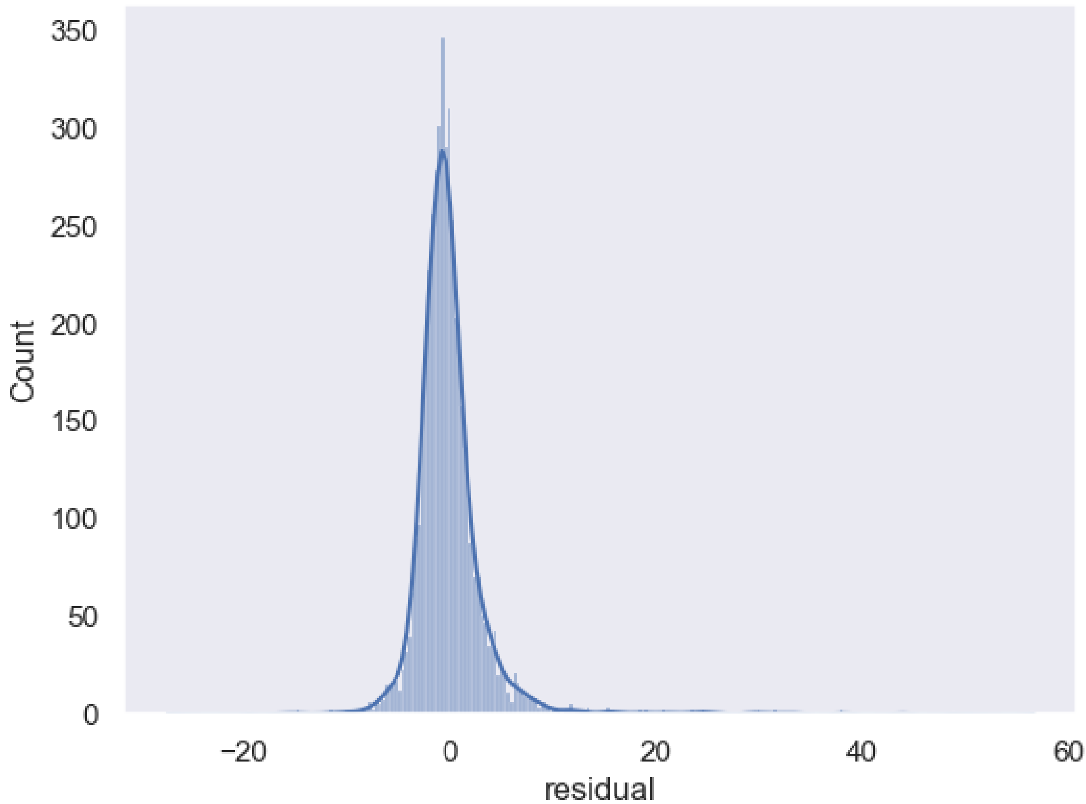
```
# Create a scatterplot to visualize `predicted` over `actual`
sns.scatterplot(data=results , x='predicted' , y='actual')
```

```
<Axes: xlabel='predicted', ylabel='actual'>
```




```
# Visualize the distribution of the `residuals`
sns.histplot(data=results, x='residual', kde=True)
```

```
<Axes: xlabel='residual', ylabel='Count'>
```

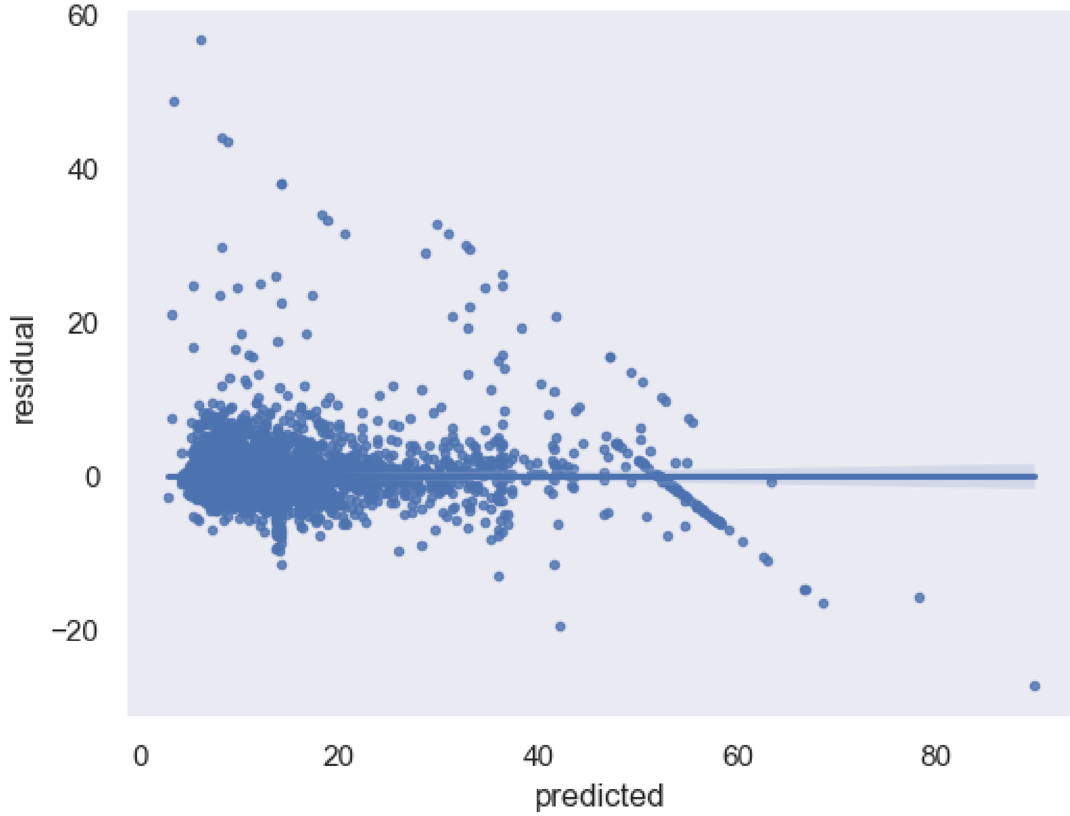



```python
# Calculate residual mean
results['residual'].mean()
```

```
7.762757641784354e-16
```

```python
# Create a scatterplot of `residuals` over `predicted`
sns.regplot(data=results , x='predicted' , y='residual' , marker='.')
```

```
<Axes: xlabel='predicted', ylabel='residual'>
```




```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   passenger_count  22699 non-null  int64
 1   mean_distance    22699 non-null  float64
 2   mean_duration    22699 non-null  float64
 3   rush_hour        22699 non-null  int64
 4   VendorID_2       22699 non-null  bool
dtypes: bool(1), float64(2), int64(2)
memory usage: 731.6 KB
```

```
# 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  12.346025   1.653975
18134    28.0  16.365952  11.634048
```

(continues on next page)

(continued from previous page)

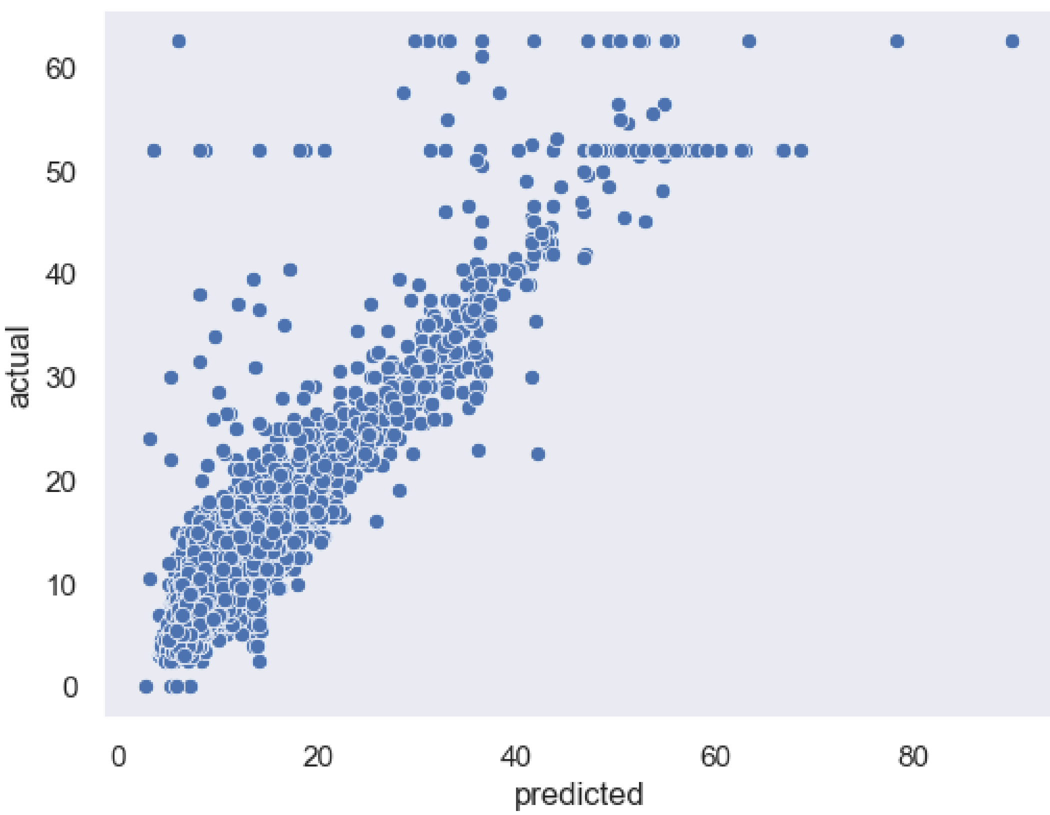
```
4655      5.5   6.587892  -1.087892
7378     15.5  16.309853  -0.809853
13914     9.5  10.291103  -0.791103
```

3.4.1 Visualize Model Results

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

```
# Create a scatterplot to visualize `predicted` over `actual`
sns.scatterplot(data=results , x='predicted' , y='actual')
```

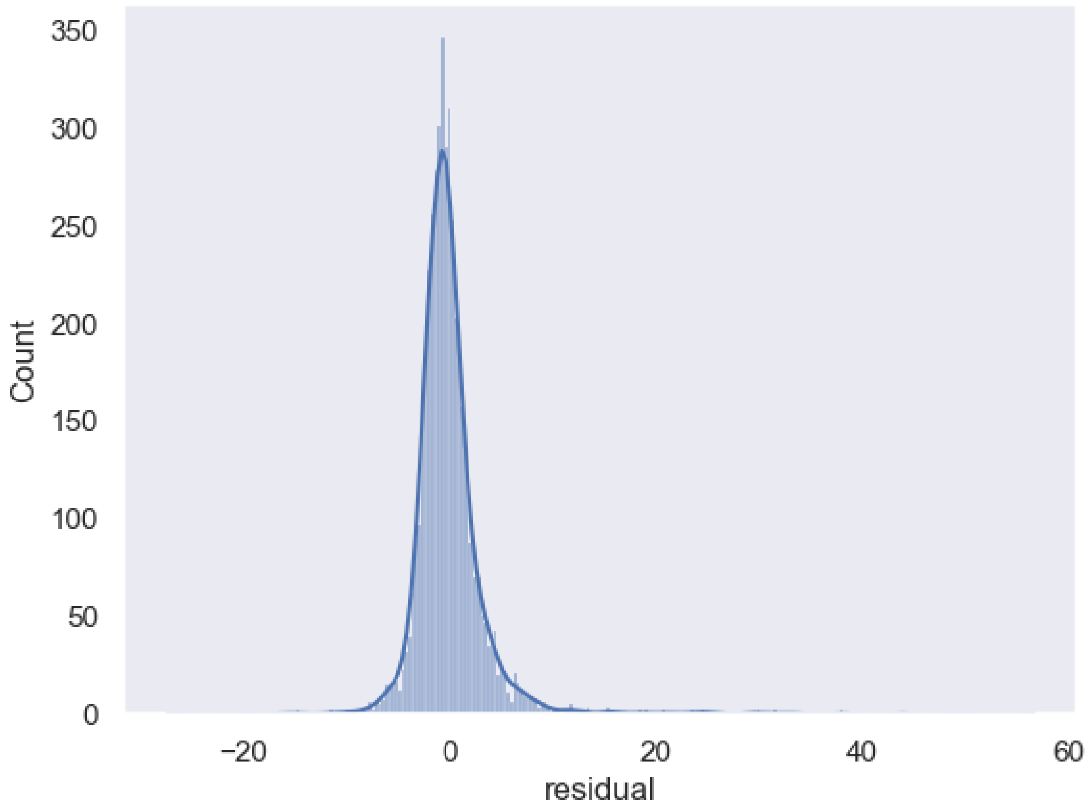
```
<Axes: xlabel='predicted', ylabel='actual'>
```



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

```
# Visualize the distribution of the `residuals`
sns.histplot(data=results, x='residual', kde=True)
```

```
<Axes: xlabel='residual', ylabel='Count'>
```



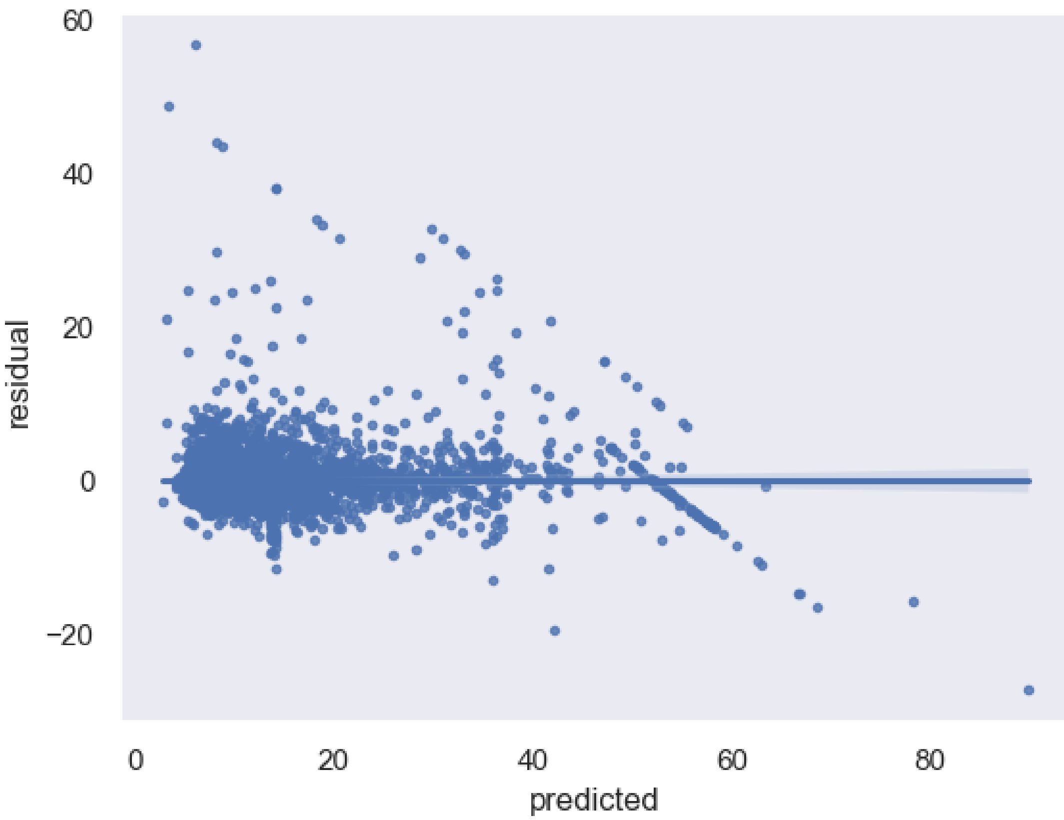

```
# Calculate residual mean
results['residual'].mean()
```

```
7.762757641784354e-16
```

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

```
# Create a scatterplot of `residuals` over `predicted`
sns.regplot(data=results , x='predicted' , y='residual' , marker='.')
```

```
<Axes: xlabel='predicted', ylabel='residual'>
```



3.4.2 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?

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   passenger_count  22699 non-null  int64  
 1   mean_distance    22699 non-null  float64
 2   mean_duration    22699 non-null  float64
 3   rush_hour        22699 non-null  int64  
 4   VendorID_2       22699 non-null  bool   
dtypes: bool(1), float64(2), int64(2)
memory usage: 731.6 KB
```

```
# Output the model's coefficients
model.coef_
```

```
array([[0.02196379, 7.32092374, 2.84139502, 0.22159236, 0.03474851]])
```

```
len(model.coef_.flatten())
```

```
5
```

```
coefficients = pd.DataFrame(columns=X.columns)
```

```
coefficients.shape
```

```
(0, 5)
```

```
len(coefficients)
```

```
0
```

```
coefficients.loc[1] = model.coef_.flatten()
```

```
coefficients
```

```
   passenger_count  mean_distance  mean_duration  rush_hour  VendorID_2
1         0.021964       7.320924       2.841395   0.221592    0.034749
```

CHAPTER FOUR

MACHINE LEARNING

You are a data professional in a data analytics firm called Automatidata. Their client, the New York City Taxi & Limousine Commission (New York City TLC), was impressed with the work you have done and has requested that you build a machine learning model to predict if a customer will not leave a tip. They want to use the model in an app that will alert taxi drivers to customers who are unlikely to tip, since drivers depend on tips.

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

```python
# Import packages and libraries
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

from xgboost import XGBClassifier,plot_importance

from sklearn import metrics
from sklearn.model_selection import GridSearchCV,train_test_split
from sklearn.ensemble import RandomForestClassifier
```

```python
# This lets us see all of the columns, preventing Juptyer from redacting them.
pd.set_option('display.max_columns', None)
```

```python
# Load dataset into dataframe
df0 = pd.read_csv('2017_Yellow_Taxi_Trip_Data.csv')

# Import predicted fares and mean distance and duration from previous course
nyc_preds_means = pd.read_csv('nyc_preds_means.csv')
df0.head(10)
```

```
   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
5    23345809         2   03/25/2017 8:34:11 PM   03/25/2017 8:42:11 PM
6    37660487         2   05/03/2017 7:04:09 PM   05/03/2017 8:03:47 PM
7    69059411         2   08/15/2017 5:41:06 PM   08/15/2017 6:03:05 PM
8     8433159         2   02/04/2017 4:17:07 PM   02/04/2017 4:29:14 PM
9    95294817         1   11/10/2017 3:20:29 PM   11/10/2017 3:40:55 PM
```

(continues on next page)

(continued from previous page)

```
   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   
5                6           2.30           1                  N   
6                1          12.83           1                  N   
7                1           2.98           1                  N   
8                1           1.20           1                  N   
9                1           1.60           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   
5           161           236             1          9.0    0.5      0.5   
6            79           241             1         47.5    1.0      0.5   
7           237           114             1         16.0    1.0      0.5   
8           234           249             2          9.0    0.0      0.5   
9           239           237             1         13.0    0.0      0.5   

   tip_amount  tolls_amount  improvement_surcharge  total_amount  
0        2.76           0.0                    0.3         16.56  
1        4.00           0.0                    0.3         20.80  
2        1.45           0.0                    0.3          8.75  
3        6.39           0.0                    0.3         27.69  
4        0.00           0.0                    0.3         17.80  
5        2.06           0.0                    0.3         12.36  
6        9.86           0.0                    0.3         59.16  
7        1.78           0.0                    0.3         19.58  
8        0.00           0.0                    0.3          9.80  
9        2.75           0.0                    0.3         16.55  
```

```
nyc_preds_means.head(10)
```

```
   mean_duration  mean_distance  predicted_fare
0      22.847222       3.521667       16.434245
1      24.470370       3.108889       16.052218
2       7.250000       0.881429        7.053706
3      30.250000       3.700000       18.731650
4      14.616667       4.435000       15.845642
5      11.855376       2.052258       10.441351
6      59.633333      12.830000       45.374542
7      26.437500       4.022500       18.555128
8       7.873457       1.019259        7.151511
9      10.541111       1.580000        9.122755
```

Join the two dataframes

Join the two dataframes using a method of your choice.

```
# Merge datasets
### YOUR CODE HERE ###
df0 = df0.merge(nyc_preds_means,left_index=True,right_index=True)
```

```
df0
```

```
       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            48           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
...           ...           ...                    ...           ...
```

(continues on next page)

(continued from previous page)

```
22694       0.00        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

       mean_duration  mean_distance  predicted_fare
0          22.847222       3.521667       16.434245
1          24.470370       3.108889       16.052218
2           7.250000       0.881429        7.053706
3          30.250000       3.700000       18.731650
4          14.616667       4.435000       15.845642
...              ...            ...             ...
22694       8.594643       1.098214        7.799138
22695      59.560417      18.757500       52.000000
22696       6.609091       0.684242        6.130896
22697      16.650000       2.077500       11.707049
22698       9.405556       1.476970        8.600969

[22699 rows x 21 columns]
```

4.1 Feature engineering

```
df0.info()
```

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

(continues on next page)

(continued from previous page)

```
 19  mean_distance          22699 non-null  float64
 20  predicted_fare         22699 non-null  float64
dtypes: float64(11), int64(7), object(3)
memory usage: 3.6+ MB
```

```
df0.head()
```

```
   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

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

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

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

   mean_duration  mean_distance  predicted_fare
0      22.847222       3.521667       16.434245
1      24.470370       3.108889       16.052218
2       7.250000       0.881429        7.053706
3      30.250000       3.700000       18.731650
4      14.616667       4.435000       15.845642
```

```
# Subset the data to isolate only customers who paid by credit card
df1 = df0[df0['payment_type'] == 1]
```

4.1.1 Target

Notice that there isn't a column that indicates tip percent, which is what you need to create the target variable. You'll have to engineer it.

Add a `tip_percent` column to the dataframe by performing the following calculation:

$$tip\ percent = \frac{tip\ amount}{total\ amount - tip\ amount}$$

Round the result to three places beyond the decimal. **This is an important step.** It affects how many customers are labeled as generous tippers. In fact, without performing this step, approximately 1,800 people who do tip 20% would be labeled as not generous.

To understand why, you must consider how floats work. Computers make their calculations using floating-point arithmetic (hence the word "float"). Floating-point arithmetic is a system that allows computers to express both very large numbers and very small numbers with a high degree of precision, encoded in binary. However, precision is limited by the number of bits used to represent a number, which is generally 32 or 64, depending on the capabilities of your operating system.

This comes with limitations in that sometimes calculations that should result in clean, precise values end up being encoded as very long decimals. Take, for example, the following calculation:

```
# Run this cell
1.1 + 2.2
```

```
3.3000000000000003
```

Notice the three that is 16 places to the right of the decimal. As a consequence, if you were to then have a step in your code that identifies values 3.3, this would not be included in the result. Therefore, whenever you perform a calculation to compute a number that is then used to make an important decision or filtration, round the number. How many degrees of precision you round to is your decision, which should be based on your use case.

Refer to this guide for more information related to floating-point arithmetic.
Refer to this guide for more information related to fixed-point arithmetic, which is an alternative to floating-point arithmetic used in certain cases.

```
df1.head(1)
```

```
   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

   passenger_count  trip_distance  RatecodeID store_and_fwd_flag  \
0                6           3.34           1                  N

   PULocationID  DOLocationID  payment_type  fare_amount  extra  mta_tax  \
0           100           231             1         13.0    0.0      0.5

   tip_amount  tolls_amount  improvement_surcharge  total_amount  \
0        2.76           0.0                    0.3         16.56

   mean_duration  mean_distance  predicted_fare
0      22.847222       3.521667       16.434245
```

```
round((100 * (df1['tip_amount'] / (df1['tip_amount'] + df1['total_amount']))),2)
```

```
0        14.29
1        16.13
2        14.22
3        18.75
5        14.29
         ...
22692    14.26
22693    14.29
22695    16.67
22697    11.56
22698    14.24
Length: 15265, dtype: float64
```

```
# Create tip % col
df1['tip_percent'] = round(df1['tip_amount'] / (df1['total_amount'] - df1['tip_amount']),
↪ 3)
```

```
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/1025802444.py:2:␣
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['tip_percent'] = round(df1['tip_amount'] / (df1['total_amount'] - df1['tip_amount
↪']), 3)
```

Now create another column called `generous`. This will be the target variable. The column should be a binary indicator of whether or not a customer tipped 20% (0=no, 1=yes).

1. Begin by making the `generous` column a copy of the `tip_percent` column.
2. Reassign the column by converting it to Boolean (True/False).
3. Reassign the column by converting Boolean to binary (1/0).

```
# Create 'generous' col (target)
df1['generous'] = df1['tip_percent']
df1['generous'] = (df1['generous'] >= 0.2)
df1['generous'] = df1['generous'].astype(int)
```

```
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/53023181.py:2:␣
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['generous'] = df1['tip_percent']
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/53023181.py:3:␣
↪SettingWithCopyWarning:
```

(continues on next page)

(continued from previous page)

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['generous'] = (df1['generous'] >= 0.2)
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/53023181.py:4:␣
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['generous'] = df1['generous'].astype(int)
```

4.1.2 Create day column

Next, you're going to be working with the pickup and dropoff columns.

Convert the `tpep_pickup_datetime` and `tpep_dropoff_datetime` columns to datetime.

```
# Convert pickup and dropoff cols to datetime
df1['tpep_pickup_datetime'] = pd.to_datetime(df1['tpep_pickup_datetime'], format='%m/%d/
↪%Y %I:%M:%S %p')
df1['tpep_dropoff_datetime'] = pd.to_datetime(df1['tpep_dropoff_datetime'], format='%m/
↪%d/%Y %I:%M:%S %p')
```

```
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/3830057672.py:2:␣
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['tpep_pickup_datetime'] = pd.to_datetime(df1['tpep_pickup_datetime'], format='%m/
↪%d/%Y %I:%M:%S %p')
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/3830057672.py:3:␣
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['tpep_dropoff_datetime'] = pd.to_datetime(df1['tpep_dropoff_datetime'], format='%m/
↪%d/%Y %I:%M:%S %p')
```

Create a day column that contains only the day of the week when each passenger was picked up. Then, convert the values to lowercase.

```
# Create a 'day' col
df1['day'] = df1['tpep_pickup_datetime'].dt.day_name().str.lower()
```

```
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/918047604.py:2:␣
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['day'] = df1['tpep_pickup_datetime'].dt.day_name().str.lower()
```

4.1.3 Create time of day columns

Next, engineer four new columns that represent time of day bins. Each column should contain binary values (0=no, 1=yes) that indicate whether a trip began (picked up) during the following times:

`am_rush` = [06:00–10:00)
`daytime` = [10:00–16:00)
`pm_rush` = [16:00–20:00)
`nighttime` = [20:00–06:00)

To do this, first create the four columns. For now, each new column should be identical and contain the same information: the hour (only) from the `tpep_pickup_datetime` column.

```
# Create 'am_rush' col
df1['am_rush'] = df1['tpep_pickup_datetime'].dt.hour

# Create 'daytime' col
df1['daytime'] = df1['tpep_pickup_datetime'].dt.hour

# Create 'pm_rush' col
df1['pm_rush'] = df1['tpep_pickup_datetime'].dt.hour

# Create 'nighttime' col
df1['nighttime'] = df1['tpep_pickup_datetime'].dt.hour
```

```
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/986325379.py:2:␣
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['am_rush'] = df1['tpep_pickup_datetime'].dt.hour
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/986325379.py:5:␣
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['daytime'] = df1['tpep_pickup_datetime'].dt.hour
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/986325379.py:8:␣
↪SettingWithCopyWarning:
```

(continues on next page)

(continued from previous page)

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['pm_rush'] = df1['tpep_pickup_datetime'].dt.hour
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/986325379.py:11:␣
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['nighttime'] = df1['tpep_pickup_datetime'].dt.hour
```

You'll need to write four functions to convert each new column to binary (0/1). Begin with `am_rush`. Complete the function so if the hour is between [06:00–10:00), it returns 1, otherwise, it returns 0.

```
# Define 'am_rush()' conversion function [06:00-10:00)
def am_rush(hour):
    if 6 <= hour['am_rush'] < 10:
        val = 1
    else:
        val = 0
    return val
```

Now, apply the `am_rush()` function to the `am_rush` series to perform the conversion. Print the first five values of the column to make sure it did what you expected it to do.

Note: Be careful! If you run this cell twice, the function will be reapplied and the values will all be changed to 0.

```
# Apply 'am_rush' function to the 'am_rush' series
df1['am_rush'] = df1.apply(am_rush, axis=1)
df1['am_rush'].head()
```

```
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/227023620.py:2:␣
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['am_rush'] = df1.apply(am_rush, axis=1)
```

```
0    1
1    0
2    1
3    0
5    0
Name: am_rush, dtype: int64
```

Now, apply the `am_rush()` function to the `am_rush` series to perform the conversion. Print the first five values of the column to make sure it did what you expected it to do.

Note: Be careful! If you run this cell twice, the function will be reapplied and the values will all be changed to 0.

```
# Define 'daytime()' conversion function [10:00-16:00)
def daytime(hour):
    if 10 <= hour['daytime'] < 16:
        val = 1
    else:
        val = 0
    return val
```

```
# Apply 'daytime' function to the 'daytime' series
df1['daytime'] = df1.apply(daytime, axis=1)
```

```
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/1985778239.py:2:
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['daytime'] = df1.apply(daytime, axis=1)
```

```
# Define 'pm_rush()' conversion function [16:00-20:00)
def pm_rush(hour):
    if 16 <= hour['pm_rush'] < 20:
        val = 1
    else:
        val = 0
    return val
```

```
# Apply 'pm_rush' function to the 'pm_rush' series
df1['pm_rush'] = df1.apply(pm_rush, axis=1)
```

```
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/3651034778.py:2:
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['pm_rush'] = df1.apply(pm_rush, axis=1)
```

```
# Define 'nighttime()' conversion function [20:00-06:00)
def nighttime(hour):
    if 20 <= hour['nighttime'] < 24:
        val = 1
    elif 0 <= hour['nighttime'] < 6:
        val = 1
    else:
        val = 0
    return val
```

```
# Apply 'nighttime' function to the 'nighttime' series
df1['nighttime'] = df1.apply(nighttime, axis=1)
```

```
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/230608774.py:2:
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['nighttime'] = df1.apply(nighttime, axis=1)
```

4.1.4 Create month column

Now, create a month column that contains only the abbreviated name of the month when each passenger was picked up, then convert the result to lowercase.

```
# Create 'month' col
df1['month'] = df1['tpep_pickup_datetime'].dt.strftime('%b').str.lower()
```

```
/var/folders/9f/pv1nlhw528d_5zttzbkb_h5m0000gn/T/ipykernel_70614/2202272965.py:2:
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  df1['month'] = df1['tpep_pickup_datetime'].dt.strftime('%b').str.lower()
```

Examine the first five rows of your dataframe.

```
df1.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
5    23345809         2  2017-03-25 20:34:11   2017-03-25 20:42:11

   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
5                6           2.30           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
```

(continues on next page)

(continued from previous page)

```
3          188            97             1          20.5    0.0      0.5
5          161           236             1           9.0    0.5      0.5

   tip_amount  tolls_amount  improvement_surcharge  total_amount  \
0        2.76           0.0                    0.3         16.56
1        4.00           0.0                    0.3         20.80
2        1.45           0.0                    0.3          8.75
3        6.39           0.0                    0.3         27.69
5        2.06           0.0                    0.3         12.36

   mean_duration  mean_distance  predicted_fare  tip_percent  generous  \
0      22.847222       3.521667       16.434245        0.200         1
1      24.470370       3.108889       16.052218        0.238         1
2       7.250000       0.881429        7.053706        0.199         0
3      30.250000       3.700000       18.731650        0.300         1
5      11.855376       2.052258       10.441351        0.200         1

        day  am_rush  daytime  pm_rush  nighttime month
0  saturday        1        0        0          0   mar
1   tuesday        0        1        0          0   apr
2    friday        1        0        0          0   dec
3    sunday        0        1        0          0   may
5  saturday        0        0        0          1   mar
```

4.1.5 Drop columns

Drop redundant and irrelevant columns as well as those that would not be available when the model is deployed. This includes information like payment type, trip distance, tip amount, tip percentage, total amount, toll amount, etc. The target variable (`generous`) must remain in the data because it will get isolated as the `y` data for modeling.

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 15265 entries, 0 to 22698
Data columns (total 29 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             15265 non-null  int64
 1   VendorID               15265 non-null  int64
 2   tpep_pickup_datetime   15265 non-null  datetime64[ns]
 3   tpep_dropoff_datetime  15265 non-null  datetime64[ns]
 4   passenger_count        15265 non-null  int64
 5   trip_distance          15265 non-null  float64
 6   RatecodeID             15265 non-null  int64
 7   store_and_fwd_flag     15265 non-null  object
 8   PULocationID           15265 non-null  int64
 9   DOLocationID           15265 non-null  int64
 10  payment_type           15265 non-null  int64
 11  fare_amount            15265 non-null  float64
 12  extra                  15265 non-null  float64
 13  mta_tax                15265 non-null  float64
```

(continues on next page)

(continued from previous page)

```
 14  tip_amount              15265 non-null  float64
 15  tolls_amount            15265 non-null  float64
 16  improvement_surcharge   15265 non-null  float64
 17  total_amount            15265 non-null  float64
 18  mean_duration           15265 non-null  float64
 19  mean_distance           15265 non-null  float64
 20  predicted_fare          15265 non-null  float64
 21  tip_percent             15262 non-null  float64
 22  generous                15265 non-null  int64
 23  day                     15265 non-null  object
 24  am_rush                 15265 non-null  int64
 25  daytime                 15265 non-null  int64
 26  pm_rush                 15265 non-null  int64
 27  nighttime               15265 non-null  int64
 28  month                   15265 non-null  object
dtypes: datetime64[ns](2), float64(12), int64(12), object(3)
memory usage: 3.5+ MB
```

```
# Drop columns
drop_cols = ['Unnamed: 0', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
             'payment_type', 'trip_distance', 'store_and_fwd_flag', 'payment_type',
             'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
             'improvement_surcharge', 'total_amount', 'tip_percent']

df1 = df1.drop(drop_cols, axis=1)
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 15265 entries, 0 to 22698
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   VendorID         15265 non-null  int64
 1   passenger_count  15265 non-null  int64
 2   RatecodeID       15265 non-null  int64
 3   PULocationID     15265 non-null  int64
 4   DOLocationID     15265 non-null  int64
 5   mean_duration    15265 non-null  float64
 6   mean_distance    15265 non-null  float64
 7   predicted_fare   15265 non-null  float64
 8   generous         15265 non-null  int64
 9   day              15265 non-null  object
 10  am_rush          15265 non-null  int64
 11  daytime          15265 non-null  int64
 12  pm_rush          15265 non-null  int64
 13  nighttime        15265 non-null  int64
 14  month            15265 non-null  object
dtypes: float64(3), int64(10), object(2)
memory usage: 1.9+ MB
```

4.1.6 Variable encoding

Many of the columns are categorical and will need to be dummied (converted to binary). Some of these columns are numeric, but they actually encode categorical information, such as `RatecodeID` and the pickup and dropoff locations. To make these columns recognizable to the `get_dummies()` function as categorical variables, you'll first need to convert them to `type(str)`.

1. Define a variable called `cols_to_str`, which is a list of the numeric columns that contain categorical information and must be converted to string: `RatecodeID`, `PULocationID`, `DOLocationID`.
2. Write a for loop that converts each column in `cols_to_str` to string.

```
# 1. Define list of cols to convert to string
cols_to_str = ['RatecodeID', 'PULocationID', 'DOLocationID', 'VendorID']

# 2. Convert each column to string
for col in cols_to_str:
    df1[col] = df1[col].astype('str')
```

Now convert all the categorical columns to binary.

1. Call `get_dummies()` on the dataframe and assign the results back to a new dataframe called `df2`.

```
# Convert categoricals to binary
df2 = pd.get_dummies(df1, drop_first=True)
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 15265 entries, 0 to 22698
Columns: 347 entries, passenger_count to month_sep
dtypes: bool(338), float64(3), int64(6)
memory usage: 6.1 MB
```

Evaluation metric

Before modeling, you must decide on an evaluation metric.

1. Examine the class balance of your target variable.

```
# Get class balance of 'generous' col
df2['generous'].value_counts(normalize=True)
```

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

A little over half of the customers in this dataset were "generous" (tipped 20%). The dataset is very nearly balanced.

To determine a metric, consider the cost of both kinds of model error:

- False positives (the model predicts a tip 20%, but the customer does not give one)
- False negatives (the model predicts a tip < 20%, but the customer gives more)

False positives are worse for cab drivers, because they would pick up a customer expecting a good tip and then not receive one, frustrating the driver.

False negatives are worse for customers, because a cab driver would likely pick up a different customer who was predicted to tip more—even when the original customer would have tipped generously.

The stakes are relatively even. You want to help taxi drivers make more money, but you don't want this to anger customers. Your metric should weigh both precision and recall equally. Which metric is this?

F1 score is the metric that places equal weight on true postives and false positives, and so therefore on precision and recall.

4.2 Modeling

4.2.1 Split the data

Now you're ready to model. The only remaining step is to split the data into features/target variable and training/testing data.

1. Define a variable `y` that isolates the target variable (`generous`).
2. Define a variable `X` that isolates the features.
3. Split the data into training and testing sets. Put 20% of the samples into the test set, stratify the data, and set the random state.

```
# Isolate target variable (y)
y = df2['generous']

# Isolate the features (X)
X = df2.drop('generous', axis=1)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2,␣
↪random_state=42)
```

4.2.2 Random forest

Begin with using `GridSearchCV` to tune a random forest model.

1. Instantiate the random forest classifier `rf` and set the random state.
2. Create a dictionary `cv_params` of any of the following hyperparameters and their corresponding values to tune. The more you tune, the better your model will fit the data, but the longer it will take.

- `max_depth`
- `max_features`
- `max_samples`
- `min_samples_leaf`
- `min_samples_split`
- `n_estimators`

3. Define a set `scoring` of scoring metrics for GridSearch to capture (precision, recall, F1 score, and accuracy).
4. Instantiate the `GridSearchCV` object `rf1`. Pass to it as arguments:

- estimator=`rf`

- param_grid=`cv_params`
- scoring=`scoring`
- cv: define the number of you cross-validation folds you want (`cv=_`)
- refit: indicate which evaluation metric you want to use to select the model (`refit=_`)

Note: `refit` should be set to `'f1'`.

```
# 1. Instantiate the random forest classifier
rf = RandomForestClassifier(random_state=42)

# 2. Create a dictionary of hyperparameters to tune
# Note that this example only contains 1 value for each parameter for simplicity,
# but you should assign a dictionary with ranges of values
cv_params = {'max_depth': [None],
             'max_features': [1.0],
             'max_samples': [0.7],
             'min_samples_leaf': [1],
             'min_samples_split': [2],
             'n_estimators': [300]
             }

# 3. Define a set of scoring metrics to capture
scoring = ['accuracy', 'precision', 'recall', 'f1']

# 4. Instantiate the GridSearchCV object
rf1 = GridSearchCV(rf, cv_params, scoring=scoring, cv=4, refit='f1')
```

Now fit the model to the training data.

Note: *Depending on how many options you include in your search grid and the number of cross-validation folds you select, this could take a very long time—even hours. If you use 4-fold validation and include only one possible value for each hyperparameter and grow 300 trees to full depth, it should take about 5 minutes. If you add another value for GridSearch to check for, say,* `min_samples_split` *(so all hyperparameters now have 1 value except for* `min_samples_split`*, which has 2 possibilities), it would double the time to ~10 minutes. Each additional parameter would approximately double the time.*

```
%%time
rf1.fit(X_train , y_train)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
File <timed eval>:1

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/sklearn/base.py:1473, in _
↪fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1466     estimator._validate_params()
   1468 with config_context(
   1469     skip_parameter_validation=(
   1470         prefer_skip_nested_validation or global_skip_validation
   1471     )
   1472 ):
-> 1473     return fit_method(estimator, *args, **kwargs)
```

(continues on next page)

(continued from previous page)

```
File /opt/anaconda3/envs/General/lib/python3.12/site-packages/sklearn/model_selection/_
↪search.py:1018, in BaseSearchCV.fit(self, X, y, **params)
   1012     results = self._format_results(
   1013         all_candidate_params, n_splits, all_out, all_more_results
   1014     )
   1016     return results
-> 1018 self._run_search(evaluate_candidates)
   1020 # multimetric is determined here because in the case of a callable
   1021 # self.scoring the return type is only known after calling
   1022 first_test_score = all_out[0]["test_scores"]

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/sklearn/model_selection/_
↪search.py:1572, in GridSearchCV._run_search(self, evaluate_candidates)
   1570 def _run_search(self, evaluate_candidates):
   1571     """Search all candidates in param_grid"""
-> 1572     evaluate_candidates(ParameterGrid(self.param_grid))

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/sklearn/model_selection/_
↪search.py:964, in BaseSearchCV.fit.<locals>.evaluate_candidates(candidate_params, cv,␣
↪more_results)
    956 if self.verbose > 0:
    957     print(
    958         "Fitting {0} folds for each of {1} candidates,"
    959         " totalling {2} fits".format(
    960             n_splits, n_candidates, n_candidates * n_splits
    961         )
    962     )
--> 964 out = parallel(
    965     delayed(_fit_and_score)(
    966         clone(base_estimator),
    967         X,
    968         y,
    969         train=train,
    970         test=test,
    971         parameters=parameters,
    972         split_progress=(split_idx, n_splits),
    973         candidate_progress=(cand_idx, n_candidates),
    974         **fit_and_score_kwargs,
    975     )
    976     for (cand_idx, parameters), (split_idx, (train, test)) in product(
    977         enumerate(candidate_params),
    978         enumerate(cv.split(X, y, **routed_params.splitter.split)),
    979     )
    980 )
    982 if len(out) < 1:
    983     raise ValueError(
    984         "No fits were performed. "
    985         "Was the CV iterator empty? "
    986         "Were there no candidates?"
    987     )

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/sklearn/utils/parallel.
```

(continues on next page)

(continued from previous page)

```
↪py:74, in Parallel.__call__(self, iterable)
     69 config = get_config()
     70 iterable_with_config = (
     71     (_with_config(delayed_func, config), args, kwargs)
     72     for delayed_func, args, kwargs in iterable
     73 )
---> 74 return super().__call__(iterable_with_config)

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/joblib/parallel.py:1918,␣
↪in Parallel.__call__(self, iterable)
   1916     output = self._get_sequential_output(iterable)
   1917     next(output)
-> 1918     return output if self.return_generator else list(output)
   1920 # Let's create an ID that uniquely identifies the current call. If the
   1921 # call is interrupted early and that the same instance is immediately
   1922 # re-used, this id will be used to prevent workers that were
   1923 # concurrently finalizing a task from the previous call to run the
   1924 # callback.
   1925 with self._lock:

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/joblib/parallel.py:1847,␣
↪in Parallel._get_sequential_output(self, iterable)
   1845 self.n_dispatched_batches += 1
   1846 self.n_dispatched_tasks += 1
-> 1847 res = func(*args, **kwargs)
   1848 self.n_completed_tasks += 1
   1849 self.print_progress()

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/sklearn/utils/parallel.
↪py:136, in _FuncWrapper.__call__(self, *args, **kwargs)
    134     config = {}
    135 with config_context(**config):
--> 136     return self.function(*args, **kwargs)

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/sklearn/model_selection/_
↪validation.py:888, in _fit_and_score(estimator, X, y, scorer, train, test, verbose,␣
↪parameters, fit_params, score_params, return_train_score, return_parameters, return_n_
↪test_samples, return_times, return_estimator, split_progress, candidate_progress,␣
↪error_score)
    886         estimator.fit(X_train, **fit_params)
    887     else:
--> 888         estimator.fit(X_train, y_train, **fit_params)
    890 except Exception:
    891     # Note fit time as time until error
    892     fit_time = time.time() - start_time

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/sklearn/base.py:1473, in _
↪fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1466     estimator._validate_params()
   1468 with config_context(
   1469     skip_parameter_validation=(
   1470         prefer_skip_nested_validation or global_skip_validation
```

(continues on next page)

(continued from previous page)

```
   1471     )
   1472 ):
-> 1473     return fit_method(estimator, *args, **kwargs)

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/sklearn/ensemble/_forest.py:489, in BaseForest.fit(self, X, y, sample_weight)
    478 trees = [
    479     self._make_estimator(append=False, random_state=random_state)
    480     for i in range(n_more_estimators)
    481 ]
    483 # Parallel loop: we prefer the threading backend as the Cython code
    484 # for fitting the trees is internally releasing the Python GIL
    485 # making threading more efficient than multiprocessing in
    486 # that case. However, for joblib 0.12+ we respect any
    487 # parallel_backend contexts set at a higher level,
    488 # since correctness does not rely on using threads.
--> 489 trees = Parallel(
    490     n_jobs=self.n_jobs,
    491     verbose=self.verbose,
    492     prefer="threads",
    493 )(
    494     delayed(_parallel_build_trees)(
    495         t,
    496         self.bootstrap,
    497         X,
    498         y,
    499         sample_weight,
    500         i,
    501         len(trees),
    502         verbose=self.verbose,
    503         class_weight=self.class_weight,
    504         n_samples_bootstrap=n_samples_bootstrap,
    505         missing_values_in_feature_mask=missing_values_in_feature_mask,
    506     )
    507     for i, t in enumerate(trees)
    508 )
    510 # Collect newly grown trees
    511 self.estimators_.extend(trees)

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/sklearn/utils/parallel.py:74, in Parallel.__call__(self, iterable)
     69 config = get_config()
     70 iterable_with_config = (
     71     (_with_config(delayed_func, config), args, kwargs)
     72     for delayed_func, args, kwargs in iterable
     73 )
---> 74 return super().__call__(iterable_with_config)

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/joblib/parallel.py:1918, in Parallel.__call__(self, iterable)
   1916     output = self._get_sequential_output(iterable)
   1917     next(output)
```

(continues on next page)

(continued from previous page)

```
-> 1918     return output if self.return_generator else list(output)
   1920 # Let's create an ID that uniquely identifies the current call. If the
   1921 # call is interrupted early and that the same instance is immediately
   1922 # re-used, this id will be used to prevent workers that were
   1923 # concurrently finalizing a task from the previous call to run the
   1924 # callback.
   1925 with self._lock:

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/joblib/parallel.py:1847,␣
↪in Parallel._get_sequential_output(self, iterable)
   1845 self.n_dispatched_batches += 1
   1846 self.n_dispatched_tasks += 1
-> 1847 res = func(*args, **kwargs)
   1848 self.n_completed_tasks += 1
   1849 self.print_progress()

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/sklearn/utils/parallel.
↪py:136, in _FuncWrapper.__call__(self, *args, **kwargs)
    134     config = {}
    135 with config_context(**config):
--> 136     return self.function(*args, **kwargs)

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/sklearn/ensemble/_forest.
↪py:192, in _parallel_build_trees(tree, bootstrap, X, y, sample_weight, tree_idx, n_
↪trees, verbose, class_weight, n_samples_bootstrap, missing_values_in_feature_mask)
    189     elif class_weight == "balanced_subsample":
    190         curr_sample_weight *= compute_sample_weight("balanced", y,␣
↪indices=indices)
--> 192     tree._fit(
    193         X,
    194         y,
    195         sample_weight=curr_sample_weight,
    196         check_input=False,
    197         missing_values_in_feature_mask=missing_values_in_feature_mask,
    198     )
    199 else:
    200     tree._fit(
    201         X,
    202         y,
   (...)
    205         missing_values_in_feature_mask=missing_values_in_feature_mask,
    206     )

File /opt/anaconda3/envs/General/lib/python3.12/site-packages/sklearn/tree/_classes.
↪py:472, in BaseDecisionTree._fit(self, X, y, sample_weight, check_input, missing_
↪values_in_feature_mask)
    461 else:
    462     builder = BestFirstTreeBuilder(
    463         splitter,
    464         min_samples_split,
   (...)
    469         self.min_impurity_decrease,
```

(continues on next page)

(continued from previous page)

```
    470     )
--> 472 builder.build(self.tree_, X, y, sample_weight, missing_values_in_feature_mask)
    474 if self.n_outputs_ == 1 and is_classifier(self):
    475     self.n_classes_ = self.n_classes_[0]

KeyboardInterrupt:
```

```
import pickle

# Define a path to the folder where you want to save the model
path = '/home/jovyan/work/'
```

```
def write_pickle(path, model_object, save_name:str):
    '''
    save_name is a string.
    '''
    with open(path + save_name + '.pickle', 'wb') as to_write:
        pickle.dump(model_object, to_write)
```

```
def read_pickle(path, saved_model_name:str):
    '''
    saved_model_name is a string.
    '''
    with open(path + saved_model_name + '.pickle', 'rb') as to_read:
        model = pickle.load(to_read)

        return model
```

Examine the best average score across all the validation folds.

```
# Examine best score
rf1.best_score_
```

```
0.7133701077670043
```

Examine the best combination of hyperparameters.

```
rf1.best_params_
```

```
{'max_depth': None,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 300}
```

Use the `make_results()` function to output all of the scores of your model. Note that it accepts three arguments.

```
def make_results(model_name:str, model_object, metric:str):
    '''
    Arguments:
```

(continues on next page)

(continued from previous page)

```
    model_name (string): what you want the model to be called in the output table
    model_object: a fit GridSearchCV object
    metric (string): precision, recall, f1, or accuracy

    Returns a pandas df with the F1, recall, precision, and accuracy scores
    for the model with the best mean 'metric' score across all validation folds.
    '''

    # Create dictionary that maps input metric to actual metric name in GridSearchCV
    metric_dict = {'precision': 'mean_test_precision',
                   'recall': 'mean_test_recall',
                   'f1': 'mean_test_f1',
                   'accuracy': 'mean_test_accuracy',
                   }

    # Get all the results from the CV and put them in a df
    cv_results = pd.DataFrame(model_object.cv_results_)

    # Isolate the row of the df with the max(metric) score
    best_estimator_results = cv_results.iloc[cv_results[metric_dict[metric]].idxmax(), :]

    # Extract Accuracy, precision, recall, and f1 score from that row
    f1 = best_estimator_results.mean_test_f1
    recall = best_estimator_results.mean_test_recall
    precision = best_estimator_results.mean_test_precision
    accuracy = best_estimator_results.mean_test_accuracy

    # Create table of results
    table = pd.DataFrame({'model': [model_name],
                          'precision': [precision],
                          'recall': [recall],
                          'F1': [f1],
                          'accuracy': [accuracy],
                          },
                         )

    return table
```

```
# Call 'make_results()' on the GridSearch object
results = make_results('RF CV', rf1, 'f1')
results
```

```
   model  precision    recall       F1  accuracy
0  RF CV   0.675349  0.756223  0.71337  0.680314
```

This is an acceptable model across the board. Typically scores of 0.65 or better are considered acceptable, but this is always dependent on your use case. Optional: try to improve the scores. It's worth trying, especially to practice searching over different hyperparameters.

```
# Get scores on test data
rf_preds = rf1.best_estimator_.predict(X_test)
```

Use the below `get_test_scores()` function you will use to output the scores of the model on the test data.

```
def get_test_scores(model_name:str, preds, y_test_data):
    '''
    Generate a table of test scores.

    In:
    model_name (string): Your choice: how the model will be named in the output table
    preds: numpy array of test predictions
    y_test_data: numpy array of y_test data

    Out:
    table: a pandas df of precision, recall, f1, and accuracy scores for your model
    '''
    accuracy = metrics.accuracy_score(y_test_data, preds)
    precision = metrics.precision_score(y_test_data, preds)
    recall = metrics.recall_score(y_test_data, preds)
    f1 = metrics.f1_score(y_test_data, preds)

    table = pd.DataFrame({'model': [model_name],
                          'precision': [precision],
                          'recall': [recall],
                          'F1': [f1],
                          'accuracy': [accuracy]
                          })

    return table
```

1. Use the `get_test_scores()` function to generate the scores on the test data. Assign the results to `rf_test_scores`.
2. Call `rf_test_scores` to output the results.

```
# Get scores on test data
rf_test_scores = get_test_scores('RF test', rf_preds, y_test)
results = pd.concat([results, rf_test_scores], axis=0)
results
```

```
     model  precision    recall        F1  accuracy
0    RF CV   0.675349  0.756223  0.713370  0.680314
0  RF test   0.674419  0.775980  0.721644  0.684900
```

4.2.3 XGBoost

Try to improve your scores using an XGBoost model.

1. Instantiate the XGBoost classifier `xgb` and set `objective='binary:logistic'`. Also set the random state.
2. Create a dictionary `cv_params` of the following hyperparameters and their corresponding values to tune:

- `max_depth`
- `min_child_weight`
- `learning_rate`

- n_estimators

3. Define a set scoring of scoring metrics for grid search to capture (precision, recall, F1 score, and accuracy).
4. Instantiate the GridSearchCV object xgb1. Pass to it as arguments:

- estimator=xgb
- param_grid=cv_params
- scoring=scoring
- cv: define the number of cross-validation folds you want (cv=_)
- refit: indicate which evaluation metric you want to use to select the model (refit='f1')

```
# 1. Instantiate the XGBoost classifier
xgb = XGBClassifier(objective='binary:logistic', random_state=0)

# 2. Create a dictionary of hyperparameters to tune
# Note that this example only contains 1 value for each parameter for simplicity,
# but you should assign a dictionary with ranges of values
cv_params = {'learning_rate': [0.1],
             'max_depth': [8],
             'min_child_weight': [2],
             'n_estimators': [500]
             }

# 3. Define a set of scoring metrics to capture
scoring = ['accuracy', 'precision', 'recall', 'f1']

# 4. Instantiate the GridSearchCV object
xgb1 = GridSearchCV(xgb, cv_params, scoring=scoring, cv=4, refit='f1')
```

Now fit the model to the X_train and y_train data.

```
%%time
xgb1.fit(X_train, y_train)
```

```
CPU times: user 25.3 s, sys: 8.48 s, total: 33.8 s
Wall time: 16.2 s
```

```
GridSearchCV(cv=4,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, device=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None,
                                     feature_types=None, gamma=None,
                                     grow_policy=None, importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=None,...
                                     max_delta_step=None, max_depth=None,
                                     max_leaves=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
```

(continues on next page)

(continued from previous page)

```
                                    multi_strategy=None, n_estimators=None,
                                    n_jobs=None, num_parallel_tree=None,
                                    random_state=0, ...),
             param_grid={'learning_rate': [0.1], 'max_depth': [8],
                         'min_child_weight': [2], 'n_estimators': [500]},
             refit='f1', scoring=['accuracy', 'precision', 'recall', 'f1'])
```

```
# Examine best score
xgb1.best_score_
```

```
0.6955124635485909
```

```
# Examine best parameters
xgb1.best_params_
```

```
{'learning_rate': 0.1,
 'max_depth': 8,
 'min_child_weight': 2,
 'n_estimators': 500}
```

```
# Call 'make_results()' on the GridSearch object
xgb1_cv_results = make_results('XGB CV', xgb1, 'f1')
results = pd.concat([results, xgb1_cv_results], axis=0)
results
```

```
     model  precision    recall        F1  accuracy
0    RF CV   0.675349  0.756223  0.713370  0.680314
0  RF test   0.674419  0.775980  0.721644  0.684900
0   XGB CV   0.669726  0.723553  0.695512  0.666557
```

Use your model to predict on the test data. Assign the results to a variable called `xgb_preds`.

```
# Get scores on test data
xgb_preds = xgb1.best_estimator_.predict(X_test)
```

1. Use the `get_test_scores()` function to generate the scores on the test data. Assign the results to `xgb_test_scores`.
2. Call `xgb_test_scores` to output the results.

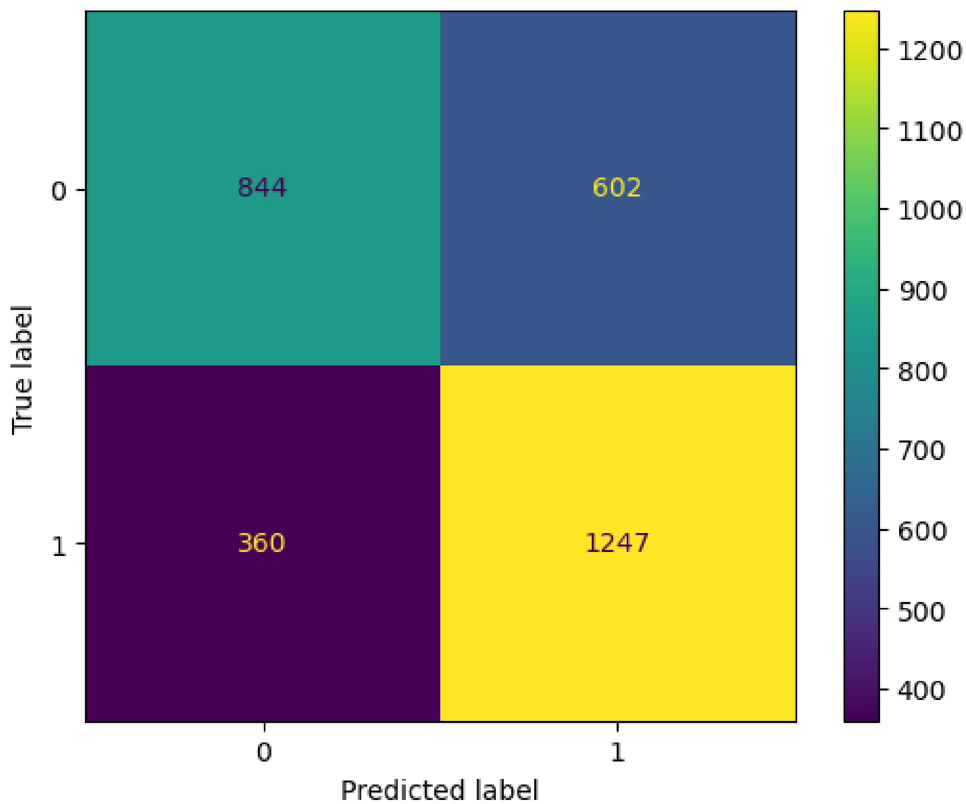
```
# Get scores on test data
xgb_test_scores = get_test_scores('XGB test', xgb_preds, y_test)
results = pd.concat([results, xgb_test_scores], axis=0)
results
```

```
      model  precision    recall        F1  accuracy
0     RF CV   0.675349  0.756223  0.713370  0.680314
0   RF test   0.674419  0.775980  0.721644  0.684900
0    XGB CV   0.669726  0.723553  0.695512  0.666557
0  XGB test   0.677219  0.745488  0.709716  0.679004
```

Plot a confusion matrix of the champion model's predictions on the test data.

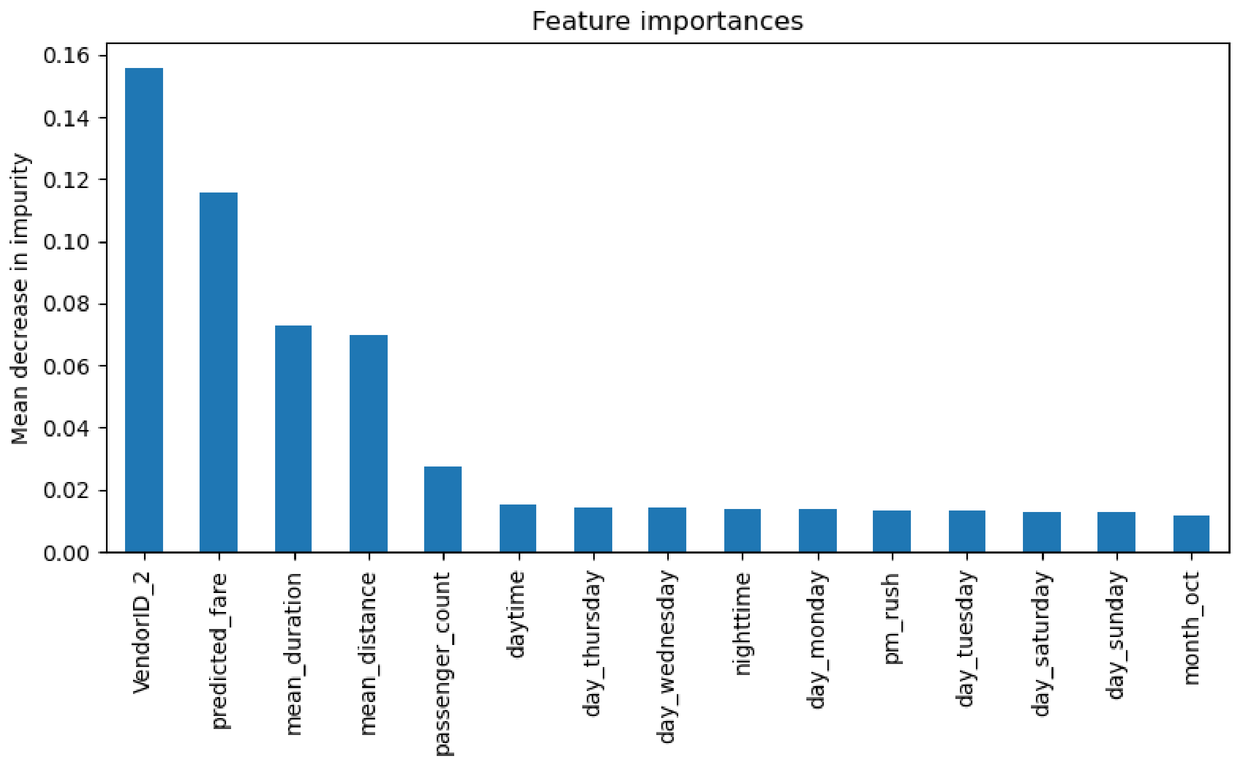
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, RocCurveDisplay
```

```
# Generate array of values for confusion matrix
cm = confusion_matrix(y_test, rf_preds, labels=rf1.classes_)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=rf1.classes_,
                              )
disp.plot(values_format='');
```



Use the `feature_importances_` attribute of the best estimator object to inspect the features of your final model. You can then sort them and plot the most important ones.

```
importances = rf1.best_estimator_.feature_importances_
rf_importances = pd.Series(importances, index=X_test.columns)
rf_importances = rf_importances.sort_values(ascending=False)[:15]

fig, ax = plt.subplots(figsize=(8,5))
rf_importances.plot.bar(ax=ax)
ax.set_title('Feature importances')
ax.set_ylabel('Mean decrease in impurity')
fig.tight_layout();
```


Feature importances
Mean decrease in impurity
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
VendorID_2
predicted_fare
mean_duration
mean_distance
passenger_count
daytime
day_thursday
day_wednesday
nighttime
day_monday
pm_rush
day_tuesday
day_saturday
day_sunday
month_oct