Automatidata Project

Ismi Ana Sulasiyah
annaismi17@gmail.com

Google Advanced Data Analytics Certification

Automatidata Overview

Automatidata

Background TLC

Since 1971, the New York City Taxi and Limousine Commission (TLC) has been regulating and overseeing the licensing of New York City's taxi cabs, for-hire vehicles, commuter vans, and paratransit vehicles.

Background Automatidata

Automatidata, a fictional data consulting firm. Automatidata’s focus is to help clients transform their unused and stored data into useful solutions.

Project Goal

TLC has approached the data consulting firm Automatidata to develop an app that enables TLC riders to estimate the taxi fares in advance of their ride.

Project Journey

Project Proposal

Project proposal that will create milestones for the tasks within the TLC project.

Understand the Data

Build a dataframe and organize the data for the process of exploratory data analysis

EDA

Conduct EDA on data for the TLC project and create visuals using tableau for an executive summary

Statistical Tests

Conduct hypothesis testing on the data for the TLC data

Regression Model

Determine the type of regression model that is needed and develop one using the TLC data

Machine Learning Models

Create a machine learning model for the TLC data

Project Proposal **Steps**

01

Gather information from the notes from the last executive meeting of Automatidata

02

Classify tasks using the PACE workflow

03

Organize tasks into milestones

04

Create a project proposal for the executive team’s approval

Project Proposal

Gather information from the notes from the last executive meeting of Automatidata

The TLC has been collecting New York City-based data on taxi and rideshare trips for several years now. They've contracted our team to build a regression model that predicts ride durations based on distance, time of day, season, and additional variables as we find necessary.

Task: Please draft a plan of action for the team. Include questions we need to answer before we get started on the project, important details to consider at the beginning of the project, and action items we'll need throughout the duration of the project.

PACE Workflow: Plan Stage

What are you trying to solve or accomplish? And, what do you anticipate the impact of this work will be on the larger needs of the client?

I am trying to solve the estimation of taxi fares based on relevant variables that I already identify.

What questions need to be asked or answered?

I considered this following questions: What is the condition of the provided dataset? What variables will be the most useful? Are there trends within the data that can provide insight? What steps can I take to reduce the impact of bias?

Who is your audience for this project?

The New York City Taxi and Limousine Commission.

What resources are required to complete this project?

I will need the project dataset, Python notebook, and input from stakeholders.

What are the deliverables that will need to be created over the course of this project?

The deliverables include a dataset scrubbed for exploratory data analysis, visualizations, statistical model, regression analysis and/or machine learning model.

Automatidata Project Proposal

Overview: The New York City Taxi and Limousine Commission seeks a way to utilize the data collected from the New York City area to predict the fare amount for taxi cab rides.

Milestones	Tasks	Deliverables/Reports	Milestone Estimate
1	Establish structure for project workflow (PACE) Plan	• Global-level project document	1 - 2 days
1a	Write a project proposal Plan		
2	Compile summary information about the data Analyze	• Data files ready for EDA	2 - 3 weeks
2a	Begin exploring the data Analyze		
3	Data exploration and cleaning Plan and Analyze	• EDA report	1 week
3a	Visualization building Construct and Analyze	• Tableau dashboard/visualizations	

Automatidata Project Proposal

Overview: The New York City Taxi and Limousine Commission seeks a way to utilize the data collected from the New York City area to predict the fare amount for taxi cab rides.

Milestones	Tasks	Deliverables/Reports	Milestone Estimate
4	**Compute descriptive statistics Analyze**	• Analysis of testing results between two important variables	1 week
4a	**Conduct hypothesis testing Analyze and Construct**	• Review testing results	
5	**Build a regression model Analyze and Construct**	• Model report	2 - 3 weeks
5a	**Build a machine learning model Construct**		
6	**Evaluate the model Execute**	• Determine the success of the model • Final model	1 week
6a	**Communicate final insights with stakeholders Execute**	• Report to all stakeholders	

Understand the Data Steps

01

Load New York City TLC data with Python

02

Build a Dataframe for the TLC dataset and Examine data type of each column

03

Gather descriptive statistics

04

Create an executive summary for Automatidata

Inspect and Analyze Data

1 PACE: Plan

Task 1. Understand the situation

- How can you best prepare to understand and organize the provided taxi cab information?

Begin by exploring the dataset and consider reviewing the Data Dictionary. One can prepare to understand the information by reading the taxi cab data fields and understanding the impact of each one. Reviewing the fact sheet could also provide helpful background information. However, the primary goal is to get the data into Python, inspect it, and provide DeShawn with initial observations. The next step would be to learn more about the data and check for any anomalies.

2 PACE: Analyze

Task 2a. Build Dataframe

Create a pandas dataframe for data learning, future exploratory data analysis (EDA) and statistical activities.

In [7]:
```
#Import libraries and packages listed above
### YOUR CODE HERE ###
import pandas as pd
import numpy as np

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

```
done
```

Code the following,

- import pandas as `pd`. pandas is used for buidling dataframes.
- import numpy as `np`. numpy is imported with pandas
- `df = pd.read_csv('Datasets\NYC taxi data.csv')`

Inspect and Analyze Data

2

PACE: Analyze

Task 2b. Understand the data - Inspect the data

View and inspect summary information about the dataframe by coding the following:

1. df.head(10)
2. df.info()
3. df.describe()

In [9]:
```
#==> ENTER YOUR CODE HERE
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
```

In [8]:
```
#==> ENTER YOUR CODE HERE
df.head(10)
```

Out[8]:

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance
0	24870114	2	03/25/2017 8:55:43 AM	03/25/2017 9:09:47 AM	6	3.34
1	35634249	1	04/11/2017 2:53:28 PM	04/11/2017 3:19:58 PM	1	1.80
2	106203690	1	12/15/2017 7:26:56 AM	12/15/2017 7:34:08 AM	1	1.00
3	38942136	2	05/07/2017 1:17:59 PM	05/07/2017 1:48:14 PM	1	3.70
4	30841670	2	04/15/2017 11:32:20 PM	04/15/2017 11:49:03 PM	1	4.37
5	23345809	2	03/25/2017 8:34:11 PM	03/25/2017 8:42:11 PM	6	2.30
6	37660487	2	05/03/2017 7:04:09 PM	05/03/2017 8:03:47 PM	1	12.83
7	69059411	2	08/15/2017 5:41:06 PM	08/15/2017 6:03:05 PM	1	2.98
8	8433159	2	02/04/2017 4:17:07 PM	02/04/2017 4:29:14 PM	1	1.20
9	95294817	1	11/10/2017 3:20:29 PM	11/10/2017 3:40:55 PM	1	1.60

Inspect and Analyze Data

2

PACE: Analyze

Task 2b. Understand the data - Inspect the data

In [10]:
```
#==> ENTER YOUR CODE HERE
df.describe()
```

Out[10]:

	Unnamed: 0	VendorID	passenger_count	trip_distance	RatecodeID	PULocationID	DOLocationID	payment_type	fare_amount	extra
count	2.269900e+04	22699.000000	22699.000000	22699.000000	22699.000000	22699.000000	22699.000000	22699.000000	22699.000000	22699.000000
mean	5.675849e+07	1.556236	1.642319	2.913313	1.043394	162.412353	161.527997	1.336887	13.026629	0.333275
std	3.274493e+07	0.496838	1.285231	3.653171	0.708391	66.633373	70.139691	0.496211	13.243791	0.463097
min	1.212700e+04	1.000000	0.000000	0.000000	1.000000	1.000000	1.000000	1.000000	-120.000000	-1.000000
25%	2.852056e+07	1.000000	1.000000	0.990000	1.000000	114.000000	112.000000	1.000000	6.500000	0.000000
50%	5.673150e+07	2.000000	1.000000	1.610000	1.000000	162.000000	162.000000	1.000000	9.500000	0.000000
75%	8.537452e+07	2.000000	2.000000	3.060000	1.000000	233.000000	233.000000	2.000000	14.500000	0.500000
max	1.134863e+08	2.000000	6.000000	33.960000	99.000000	265.000000	265.000000	4.000000	999.990000	4.500000

Q1: When reviewing the df.info() output, what do you notice about the different variables? Are there any null values? Are all of the variables numeric? Does anything else stand out? **All the variables are non-numeric type. Two of which are datetime and the values are non-null.**

Q2: When reviewing the df.describe() output, what do you notice about the distributions of each variable? Are there any questionable values? **Regarding fare amount, the distribution is worth considering. The maximum fare amount is a much larger value (1000) than the 25-75 percent range of values. Also, it's questionable how there are negative values for fare amount. Regarding trip distance, most rides are between 1-3 miles, but the maximum is over 33 miles.**

Inspect and Analyze Data

2

PACE: Analyze

Task 2c. Understand the data - Investigate the variables

Sort and interpret the data table for two variables: trip_distance and total_amount.

In [11]:
```
# ==> ENTER YOUR CODE HERE
df_sort = df.sort_values(by='trip_distance', ascending=False)
df_sort.head(10)
# Sort the data by trip distance from maximum to minimum value
```

Out[11]:

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance
9280	51810714	2	06/18/2017 11:33:25 PM	06/19/2017 12:12:38 AM	2	33.96
13861	40523668	2	05/19/2017 8:20:21 AM	05/19/2017 9:20:30 AM	1	33.92
6064	49894023	2	06/13/2017 12:30:22 PM	06/13/2017 1:37:51 PM	1	32.72
10291	76319330	2	09/11/2017 11:41:04 AM	09/11/2017 12:18:58 PM	1	31.95
29	94052446	2	11/06/2017 8:30:50 PM	11/07/2017 12:00:00 AM	1	30.83
18130	90375786	1	10/26/2017 2:45:01 PM	10/26/2017 4:12:49 PM	1	30.50
5792	68023798	2	08/11/2017 2:14:01 PM	08/11/2017 3:17:31 PM	1	30.33
15350	77309977	2	09/14/2017 1:44:44 PM	09/14/2017 2:34:29 PM	1	28.23
10302	43431843	1	05/15/2017 8:11:34 AM	05/15/2017 9:03:16 AM	1	28.20
2592	51094874	2	06/16/2017 6:51:20 PM	06/16/2017 7:41:42 PM	1	27.97

Q1: Sort your first variable (trip_distance) from maximum to minimum value, do the values seem normal? **The values align with our earlier data discovery, the longest rides are approximately 33 miles.**

Inspect and Analyze Data

2

PACE: Analyze

Task 2c. Understand the data - Investigate the variables

Sort and interpret the data table for two variables: trip_distance and total_amount.

In [12]:
```
#==> ENTER YOUR CODE HERE
sorted_tamount = df.sort_values(by='total_amount', ascending=False)['total_amount']
sorted_tamount.head(20)
# Sort the data by total amount and print the top 20 values
```

Out[12]:
```
8476     1200.29
20312     450.30
13861     258.21
12511     233.74
15474     211.80
6064      179.06
16379     157.06
3582      152.30
11269     151.82
9280      150.30
1928      137.80
10291     131.80
6708      126.00
11608     123.30
908       121.56
7281      120.96
18130     119.31
13621     115.94
13359     111.95
29        111.38
Name: total_amount, dtype: float64
```

Q2: Sort by your second variable (total_amount), are any values unusual?

Yes, the first two values are significantly higher than the others.

Inspect and Analyze Data

2

PACE: Analyze

Task 2c. Understand the data - Investigate the variables

Sort and interpret the data table for two variables: trip_distance and total_amount.

In [14]:
```
# ==> ENTER YOUR CODE HERE
df_sort = df.sort_values(by='fare_amount', ascending=False)
df_sort.head(10)
# Sort the data by trip distance from maximum to minimum value
```

Out[14]:

ckup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_type	fare_amount
017 5:50:10 AM	02/06/2017 5:51:08 AM	1	2.60	5	N	226	226	1	999.99
017 9:40:46 AM	12/19/2017 9:40:55 AM	2	0.00	5	N	265	265	2	450.00
017 8:20:21 AM	05/19/2017 9:20:30 AM	1	33.92	5	N	229	265	1	200.01
017 8:55:01 PM	06/06/2017 8:55:06 PM	1	0.00	5	N	265	265	1	200.00
017 6:24:24 PM	12/17/2017 6:24:42 PM	1	0.00	5	N	265	265	1	175.00
/2017 11:53:01 PM	01/01/2017 11:53:42 PM	1	7.30	5	N	1	1	1	152.00
/2017 11:33:25 PM	06/19/2017 12:12:38 AM	2	33.96	5	N	132	265	2	150.00
/2017 10:41:11 AM	11/30/2017 11:31:45 AM	1	25.50	5	N	132	265	2	140.00
/2017 11:41:04 AM	09/11/2017 12:18:58 PM	1	31.95	4	N	138	265	2	131.00
/2017 12:51:17 AM	06/19/2017 12:52:12 AM	2	0.00	5	N	265	265	1	120.00

Q3: Are the resulting rows similar for both sorts? Why or why not? **The most expensive rides are not necessarily the longest ones.**

Dataset Summarize: What can you summarize for DeShawn and the data team?
=> After looking at the dataset, the two variables could help to build a predictive model for taxi ride fares are total_amount and trip_distance. Because those variables show a picture of a taxi cab ride.

New York City TLC Project Preliminary Data Summary

Understand the Data: Executive summary report

OVERVIEW

The NYC Taxi & Limousine Commission has contracted with Automatidata to build a regression model that predicts taxi cab fares. In this part of the project, the Automatidata data team performed a preliminary inspection of the data supplied by the NYC Taxi and Limousine Commission in order to inform the team of key data variable descriptions, and ensure the information provided is suitable for generating clear and meaningful insights.

PROJECT STATUS

- Explored dataset to find any unusual values.
- Considered which variables are most useful to build predictive models (in this case: total_amount and trip_distance, which work together to depict a taxi cab ride).
- Considered potential interactions between the two chosen variables.
- Examined which components of the provided data will provide relevant insights.
- Built the groundwork for future exploratory data analysis, visualizations, and models.

NEXT STEPS

1. Conduct a complete exploratory data analysis.
2. Perform any data cleaning and data analysis steps to understand unusual variables (e.g., outliers).
3. Use descriptive statistics to learn more about the data.
4. Create and run a regression model.

KEY INSIGHTS

- This dataset includes variables that should be helpful for building prediction model(s) on taxi cab ride fares.
- The identified unusual values are trips that are a short distance but have high charges associated with them, as shown in the total_amount variable. Reference screenshots:

Total_amount variable

trip_distance	fare_amount
2.60	999.99
0.00	450.00
33.92	200.01
0.00	200.00
0.00	175.00
7.30	152.00
33.96	150.00
25.50	140.00
31.95	131.00
0.00	120.00

The total_amount variable indicates the necessity of further analyzing outlier variables.

EDA Steps

1 EDA and Cleaning

Create a Jupyter Notebook of full EDA

2 Build Visualization

Create a Tableau visualization showing two important variables

3 Share your results with the Automatidata team

Write an executive summary of results and include a visualization

Exploratory Data Analysis

1

PACE: Plan

Task 1. Identify any outliers:

- What methods are best for identifying outliers?
  - ➢ Use numpy functions to investigate the mean() and median() of the data and understand range of data values
  - ➢ Use a boxplot to visualize the distribution of the data
  - ➢ Use histograms to visualize the distribution of the data

- How do you make the decision to keep or exclude outliers from any future models?
  There are three main options for dealing with outliers: keeping them as they are, deleting them, or reassigning them. Whether to keep outliers as they are, delete them, or reassign values, these are the general guidelines that help me making a decision of outliers:
  - ➢ **Delete them**: When I'm sure the outliers are mistakes, typos, or errors and the dataset will be used for modeling or machine learning, then I'm more likely to decide to delete outliers.
  - ➢ **Reassign them**: If the dataset is small and/or the data will be used for modeling or machine learning, I'm more likely to choose a path of deriving new values to replace the outlier values.
  - ➢ **Leave them**: For a dataset that I plan to do EDA/analysis on and nothing else, or for a dataset I'm preparing for a model that is resistant to outliers, it is most likely that I'm going to leave them in.

Exploratory Data Analysis

2 PACE: Analyze

Task 2a. Data exploration and Cleaning

In [3]:
```python
# Import packages and libraries
#==> ENTER YOUR CODE HERE
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import datetime as dt
import seaborn as sns
```

In [4]:
```python
# Load dataset into dataframe
df = pd.read_csv('2017_Yellow_Taxi_Trip_Data.csv')
```

In [5]:
```python
df.head()
```

Out[5]:

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance
0	24870114	2	03/25/2017 8:55:43 AM	03/25/2017 9:09:47 AM	6	3.34
1	35634249	1	04/11/2017 2:53:28 PM	04/11/2017 3:19:58 PM	1	1.80
2	106203690	1	12/15/2017 7:26:56 AM	12/15/2017 7:34:08 AM	1	1.00
3	38942136	2	05/07/2017 1:17:59 PM	05/07/2017 1:48:14 PM	1	3.70
4	30841670	2	04/15/2017 11:32:20 PM	04/15/2017 11:49:03 PM	1	4.37

In [6]:
```python
df.size
```

Out[6]: 408582

In [8]:
```python
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
```

Note: There is no missing data according to the results from the info() function.

Exploratory Data Analysis

3

PACE: Construct

Task 3. Data visualization

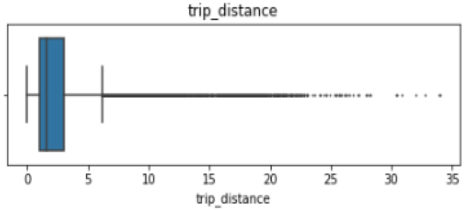
Perform a check for outliers on relevant columns such as trip_distance and total_amount from previous step. Some of the best ways to identify the presence of outliers in data are box plots and histograms.

Note: Convert your date columns to datetime in order to derive total trip duration.

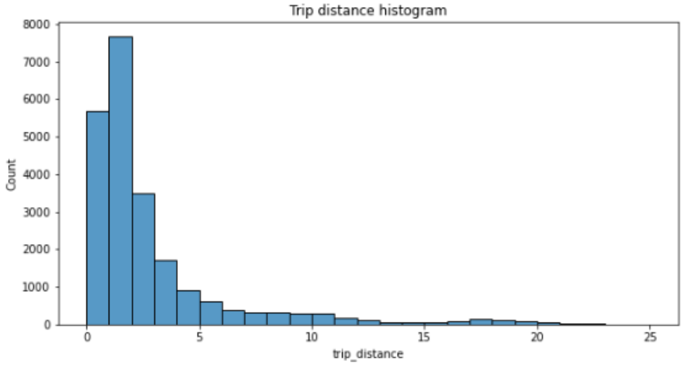
In [9]:
```
# Convert data columns to datetime
#==> ENTER YOUR CODE HERE
df['tpep_pickup_datetime']=pd.to_datetime(df['tpep_pickup_datetime'])
df['tpep_dropoff_datetime']=pd.to_datetime(df['tpep_dropoff_datetime'])
```

trip distance

In [10]:
```
# Create box plot of trip_distance
#==> ENTER YOUR CODE HERE
plt.figure(figsize=(7,2))
plt.title('trip_distance')
sns.boxplot(data=None, x=df['trip_distance'], fliersize=1);
```



In [11]:
```
# Create histogram of trip_distance
#==> ENTER YOUR CODE HERE
plt.figure(figsize=(10,5))
sns.histplot(df['trip_distance'], bins=range(0,26,1))
plt.title('Trip distance histogram');
```



Note: The majority of trips were journeys of less than two miles. The number of trips falls away steeply as the distance traveled increases beyond two miles.

Exploratory Data Analysis

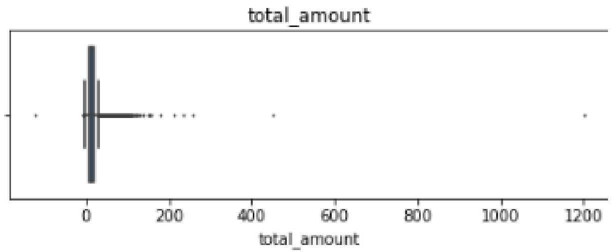
3 PACE: Construct

Task 3. Data visualization

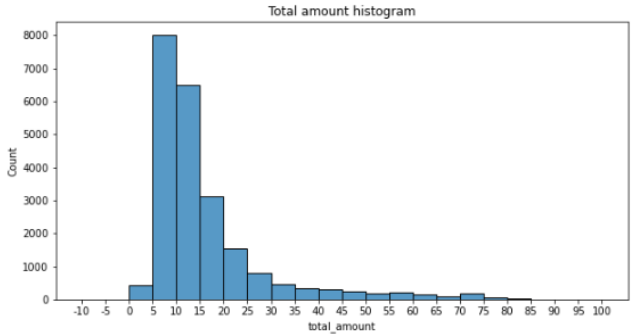
Perform a check for outliers on relevant columns such as trip_distance and total_amount from previous step. Some of the best ways to identify the presence of outliers in data are box plots and histograms.

total amount

In [12]:
```
# Create box plot of total_amount
#==> ENTER YOUR CODE HERE
plt.figure(figsize=(7,2))
plt.title('total_amount')
sns.boxplot(x=df['total_amount'], fliersize=1);
```



In [13]:
```
# Create histogram of total_amount
#==> ENTER YOUR CODE HERE
plt.figure(figsize=(10,5))
ax = sns.histplot(df['total_amount'], bins=range(-10,101,5))
ax.set_xticks(range(-10,101,5))
ax.set_xticklabels(range(-10,101,5))
plt.title('Total amount histogram');
```



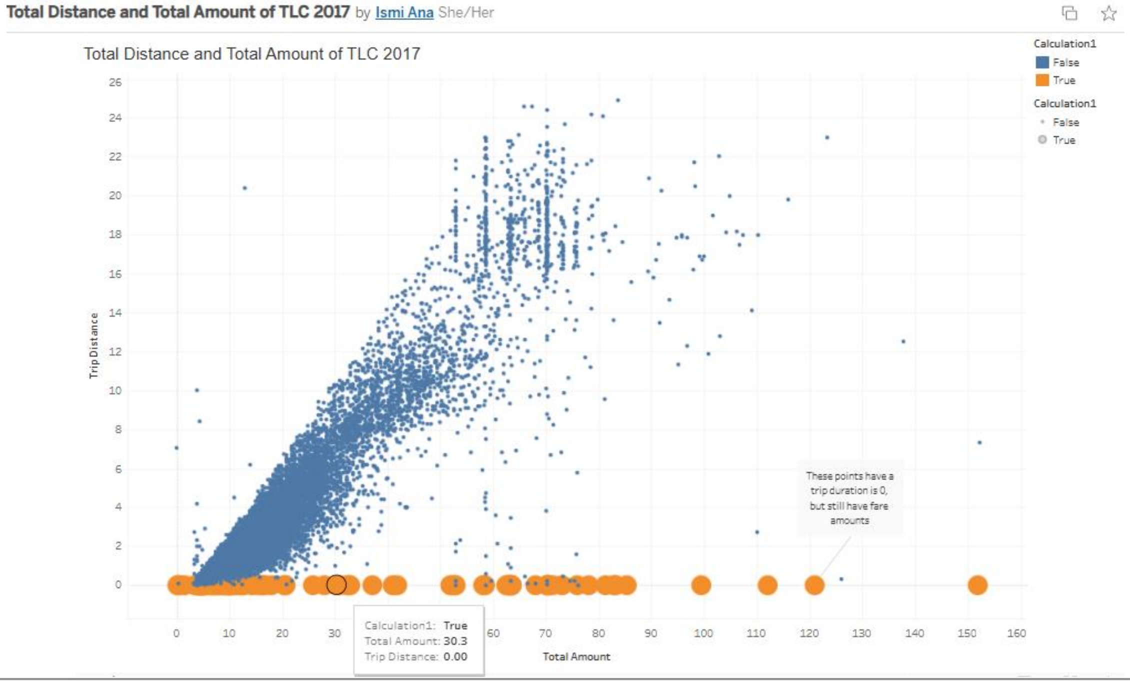
Note: The total cost of each trip also has a distribution that skews right, with most costs falling in the $5-15 range.

Exploratory Data Analysis

PACE: Construct

3

Task 3. Data visualization
create a scatterplot showing the relationship between trip_distance and total_amount.



Exploratory Data Analysis of New York City TLC Data

EDA : Executive summary report

Overview

The NYC Taxi & Limousine Commission has contracted with Automatidata to build a regression model that predicts taxi cab ride fares. In this part of the project, the data needs to be analyzed, explored, cleaned and structured prior to any modeling.

Problem

After running initial exploratory data analysis (EDA) on a sample of the data provided by New York City TLC, it is clear that some of the data will prove an obstacle for accurate ride fare prediction. Namely, trips that have a total cost entered, but a total distance of “0.” At this point, our analysis indicates these to be anomalies or outliers that need to be factored into the algorithm or removed completely.

Solution

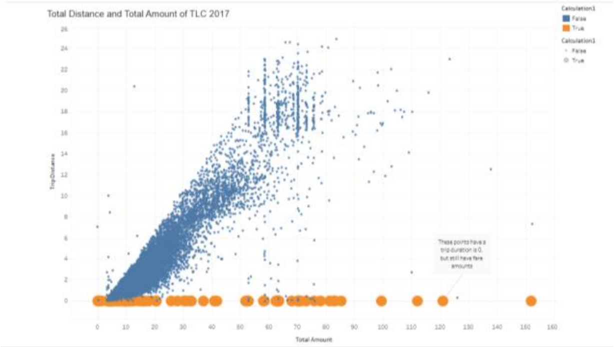
After analysis, we recommend removing outliers with a total distanced recorded of 0.

Details

As a result of the conducted exploratory data analysis, the Automatidata data team considered trip distance and total amount as key variables to depict a taxi cab ride. The provided scatter plot shows the relationship between the two variables. This scatter plot was created in Tableau to enhance the provided visualization.

Keys to success

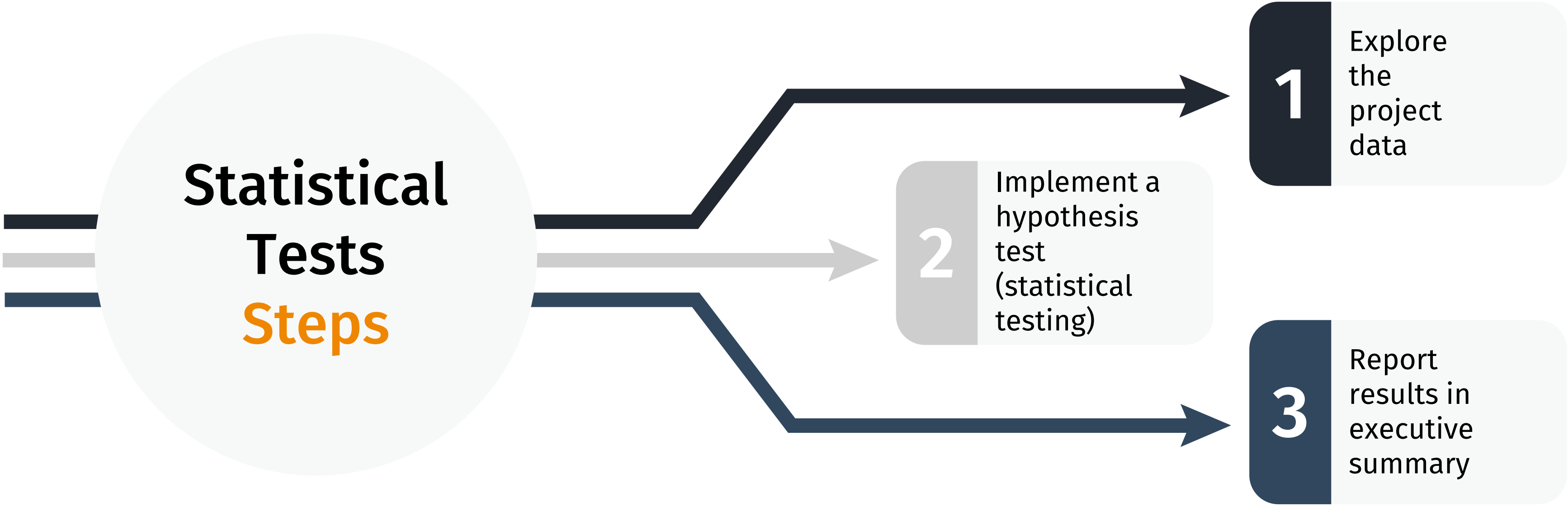
- ❑ Ensuring with New York City TLC that the sample provided is an accurate reflection of their data as a whole.
- ❑ Plan for handling other outliers, such as low trip distance paired with high costs.



Graph displaying New York City TLC data plotting variables for total distance and total amount.

Next Steps

- ➢ Determine any unusual data points that could pose a problem for future analysis in predicting trip fares. For example, locations that have longer durations.
- ➢ Determine the variables that have the largest impact on trip fares.
- ➢ Filter down to consider the most relevant variables for running regression, statistical analysis, and parameter tuning.


Statistical
Tests
Steps
1
Explore
the
project
data
2
Implement a
hypothesis
test
(statistical
testing)
3
Report
results in
executive
summary

Statistical Analysis

The Purpose

to demonstrate knowledge of how to prepare, create, and analyze A/B tests. Your A/B test results should aim to find ways to generate more revenue for taxi cab drivers

Note

For the purpose of this exercise, assume that the sample data comes from an experiment in which customers are randomly selected and divided into two groups: 1) customers who are required to pay with credit card, 2) customers who are required to pay with cash. Without this assumption, I cannot draw causal conclusions about how payment method affects fare amount.

The Goal

is to apply descriptive statistics and hypothesis testing in Python. The goal for this A/B test is to sample data and analyze whether there is a relationship between payment type and fare amount.

Statistical Analysis : Conduct an A/B test

1

PACE: Plan

Task 1. Imports and data loading

- What is your research question for this data project?

The research question for this data project: “Is there a relationship between total fare amount and payment type?”

```
In [1]: # Import packages and libraries needed to compute descriptive statistics and conduct a hypothesis test.
        import pandas as pd
        import numpy as np
        from scipy import stats
```

```
In [2]: # Load dataset into dataframe
        taxi_data = pd.read_csv("2017_Yellow_Taxi_Trip_Data.csv", index_col = 0)
```

Statistical Analysis : Conduct an A/B test

2

PACE: Analyze and Construct

Task 2. Data exploration

- Data professionals use descriptive statistics for Exploratory Data Analysis. How can computing descriptive statistics help me to learn more about data in this analysis stage?

In general, descriptive statistics are useful because they let you quickly explore and understand large amounts of data. In this case, computing descriptive statistics helps you quickly compare the average total fare amount among different payment types.

In [4]:
```
# descriptive stats code for EDA
taxi_data.describe(include='all')
```

Out[4]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID
count	22699.000000	22699	22699	22699.000000	22699.000000	22699.000000	22699	22699.000000
unique	NaN	22687	22688	NaN	NaN	NaN	2	NaN
top	NaN	07/03/2017 3:45:19 PM	10/18/2017 8:07:45 PM	NaN	NaN	NaN	N	NaN
freq	NaN	2	2	NaN	NaN	NaN	22600	NaN
mean	1.556236	NaN	NaN	1.642319	2.913313	1.043394	NaN	162.412353
std	0.496838	NaN	NaN	1.285231	3.653171	0.708391	NaN	66.633373
min	1.000000	NaN	NaN	0.000000	0.000000	1.000000	NaN	1.000000
25%	1.000000	NaN	NaN	1.000000	0.990000	1.000000	NaN	114.000000
50%	2.000000	NaN	NaN	1.000000	1.610000	1.000000	NaN	162.000000
75%	2.000000	NaN	NaN	2.000000	3.060000	1.000000	NaN	233.000000
max	2.000000	NaN	NaN	6.000000	33.960000	99.000000	NaN	265.000000

In [5]:
```
taxi_data.groupby('payment_type')['fare_amount'].mean()
```

Out[5]:
```
payment_type
1    13.429748
2    12.213546
3    12.186116
4     9.913043
Name: fare_amount, dtype: float64
```

Note. In the dataset, payment_type is encoded in integers:

1. Credit card
2. Cash
3. No charge
4. Dispute
5. Unknown

Based on the averages shown, it appears that customers who pay in credit card tend to pay a larger fare amount than customers who pay in cash. However, this difference might arise from random sampling, rather than being a true difference in fare amount. To assess whether the difference is statistically significant, I conduct a hypothesis test.

Statistical Analysis : Conduct an A/B test

2

PACE: Analyze and Construct

Task 3. Hypothesis testing

- **Null hypothesis:** There is no difference in average fare between customers who use credit cards and customers who use cash.
- **Alternative hypothesis:** There is a difference in average fare between customers who use credit cards and customers who use cash

```
In [17]: #hypothesis test, A/B test
         #significance level 5% and two-sample t-test

         credit_card = taxi_data[taxi_data['payment_type'] == 1]['fare_amount']
         cash = taxi_data[taxi_data['payment_type'] == 2]['fare_amount']
         stats.ttest_ind(a=credit_card, b=cash, equal_var=False)

Out[17]: Ttest_indResult(statistic=6.866800855655372, pvalue=6.797387473030518e-12)

In [20]: tstat, pvalue = stats.ttest_ind(a=credit_card, b=cash, equal_var=False)
         print(f"is pvalue < significance level:", pvalue < 0.05)

         is pvalue < significance level: True
```

<u>Recall the steps for conducting a hypothesis test:</u>

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

Since the p-value is significantly smaller than the significance level of 5%, **we reject null hypothesis.**

Notice the 'e-12' = 6.797387473030518^{-12}

=> I conclude that there is a statistically significant difference in the average fare amount between customers who use credit cards and customers who use cash

Statistical Analysis : Conduct an A/B test

4

PACE: Execute

Task 4. Communicate insights with stakeholders

- **What business insight(s) can you draw from the result of your hypothesis test?**

The key business insight is that encouraging customers to pay with credit cards can generate more revenue for taxi cab drivers.

- **Consider why this A/B test project might not be realistic, and what assumptions had to be made for this educational project.**

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.

Statistical Review and A/B Testing for New York City TLC Project

Statistical Analysis : Executive summary report

Overview

The purpose of this project is to predict taxi cab fares before each ride. At this point, this project's focus is to find ways to generate more revenue for New York City taxi cab drivers. This part of the project examines the relationship between total fare amount and payment type.

Problem

Taxi cab drivers receive varying amount of tips. While examining the relationship between total fare amount and payment type, this project seeks to discover if customers who pay in credit card tend to pay a larger total fare amount than customers who pay in cash.

Solution

The Automatidata team ran an A/B test to analyze the relationship between credit card payment and total fare amount. The key business insight is that encouraging customers to pay with credit cards will likely generate more revenue for taxi drivers.

Details

Steps conducted in the A/B test

1. Collected sample data from an experiment in which customers are randomly selected and divided into two groups:
   a. Customers who are required to pay with credit card.
   b. Customers who are required to pay with cash. This enables us to draw causal conclusions about how payment method affects fare amount.
2. Computed descriptive statistics to better understand the average total fare amount for each payment method available to the customer.
3. Conducted a two-sample t-test to determine if there is a statistically significant difference in average total fare between customers who use credit cards and customers who use cash.

A/B test results

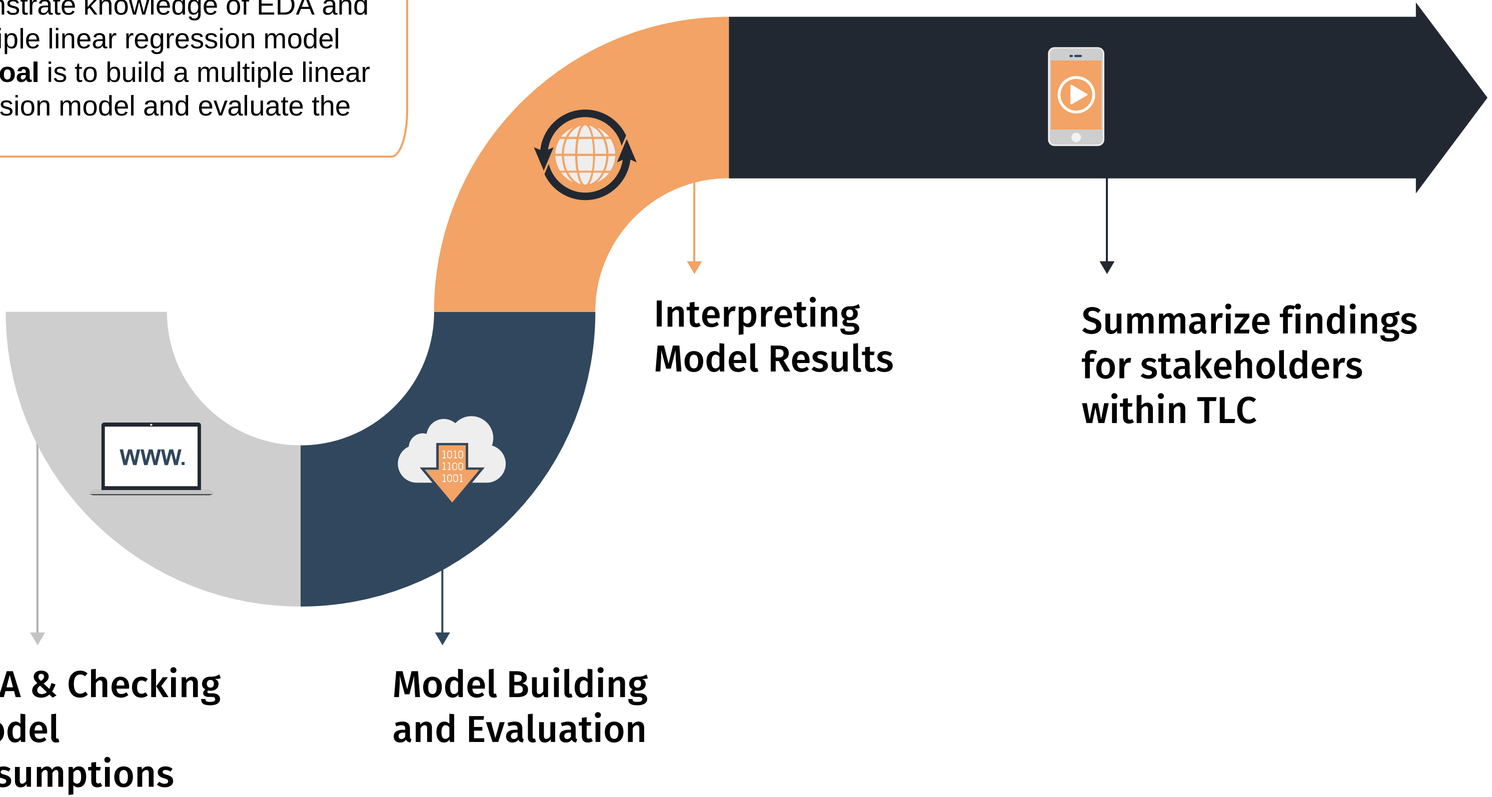
There is a statistically significant difference in the average total fare between customers who use credit cards and customers who use cash. Customers who used credit cards showed a higher total amount compared to cash.

Next Steps

The Automatidata data team recommends that the New York City TLC encourages customers to pay with credit cards, and create strategies to promote credit card payments. For example, the New York City TLC can install signs that read "Credit card payments are preferred" in their cabs, and implement a protocol that requires cab drivers to verbally inform customers that credit card payments are preferred.

Regression Model

- **The Purpose** of this project is to demonstrate knowledge of EDA and a multiple linear regression model
- **The Goal** is to build a multiple linear regression model and evaluate the model



Build a Multiple Linear Regression Model

1

PACE: Plan

Task 1. Imports and data loading

```
In [1]: # 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 date
        from datetime import timedelta

        # Packages for OLS, MLR, confusion matrix
        from sklearn.preprocessing import StandardScaler
        from sklearn.model_selection import train_test_split
        import sklearn.metrics as metrics # For confusion matrix
        from sklearn.linear_model import LinearRegression
        from sklearn.metrics import mean_absolute_error,r2_score,mean_squared_error
```

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

2

PACE: Analyze

Task 2. Data exploration

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

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

Build a Multiple Linear Regression Model

2 PACE: Analyze

Task 2a. Explore data with EDA

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

```
In [9]: # Start with `.shape` and `.info()`
        # Keep `df0` as the original dataframe and create a copy (df) where changes will go
        df = df0.copy()

        print("row, column :", df.shape)
        df.info()
```

```
row, column : (22699, 18)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             22699 non-null  int64
 1   VendorID               22699 non-null  int64
 2   tpep_pickup_datetime   22699 non-null  object
 3   tpep_dropoff_datetime  22699 non-null  object
 4   passenger_count        22699 non-null  int64
 5   trip_distance          22699 non-null  float64
 6   RatecodeID             22699 non-null  int64
 7   store_and_fwd_flag     22699 non-null  object
 8   PULocationID           22699 non-null  int64
 9   DOLocationID           22699 non-null  int64
 10  payment_type           22699 non-null  int64
 11  fare_amount            22699 non-null  float64
 12  extra                  22699 non-null  float64
 13  mta_tax                22699 non-null  float64
 14  tip_amount             22699 non-null  float64
 15  tolls_amount           22699 non-null  float64
 16  improvement_surcharge  22699 non-null  float64
 17  total_amount           22699 non-null  float64
dtypes: float64(8), int64(7), object(3)
memory usage: 3.1+ MB
```

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

```
In [10]: # Check for duplicates
         print('Shape of dataframe:', df.shape)
         print('Shape of dataframe with duplicates dropped:', df.drop_duplicates().shape)

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

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

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

Out[10]: Unnamed: 0               0
         VendorID                 0
         tpep_pickup_datetime     0
         tpep_dropoff_datetime    0
         passenger_count          0
         trip_distance            0
         RatecodeID               0
         store_and_fwd_flag       0
         PULocationID             0
         DOLocationID             0
         payment_type             0
         fare_amount              0
         extra                    0
         mta_tax                  0
         tip_amount               0
         tolls_amount             0
         improvement_surcharge    0
         total_amount             0
         dtype: int64
```

Build a Multiple Linear Regression Model

2

PACE: Analyze

Task 2a. Explore data with EDA

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

In [11]:
```
# Use .describe()
df.describe()
```

Out[11]:

	Unnamed: 0	VendorID	passenger_count	trip_distance	RatecodeID	PULocationID	DOLocationID	payment_type	fare_amount	extra
count	2.269900e+04	22699.000000	22699.000000	22699.000000	22699.000000	22699.000000	22699.000000	22699.000000	22699.000000	22699.000000
mean	5.675849e+07	1.556236	1.642319	2.913313	1.043394	162.412353	161.527997	1.336887	13.026629	0.333275
std	3.274493e+07	0.496838	1.285231	3.653171	0.708391	66.633373	70.139691	0.496211	13.243791	0.463097
min	1.212700e+04	1.000000	0.000000	0.000000	1.000000	1.000000	1.000000	1.000000	-120.000000	-1.000000
25%	2.852056e+07	1.000000	1.000000	0.990000	1.000000	114.000000	112.000000	1.000000	6.500000	0.000000
50%	5.673150e+07	2.000000	1.000000	1.610000	1.000000	162.000000	162.000000	1.000000	9.500000	0.000000
75%	8.537452e+07	2.000000	2.000000	3.060000	1.000000	233.000000	233.000000	2.000000	14.500000	0.500000
max	1.134863e+08	2.000000	6.000000	33.960000	99.000000	265.000000	265.000000	4.000000	999.990000	4.500000

Some things stand out from this table of summary statistics. For instance, there are clearly some outliers in several variables, like tip_amount ($200) and total_amount ($1,200). Also, a number of the variables, such as mta_tax, seem to be almost constant throughout the data, which would imply that they would not be expected to be very predictive.

Build a Multiple Linear Regression Model

PACE: Analyze

2

Task 2b. Convert pickup & dropoff columns to datetime

```
In [12]: # Check the format of the data
         df['tpep_dropoff_datetime'][0]

Out[12]: '03/25/2017 9:09:47 AM'
```

```
In [13]: # Convert datetime columns to datetime
         # Display data types of `tpep_pickup_datetime`, `tpep_dropoff_datetime`
         print('Data type of tpep_pickup_datetime:', df['tpep_pickup_datetime'].dtype)
         print('Data type of tpep_dropoff_datetime:', df['tpep_dropoff_datetime'].dtype)

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

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

         # Display data types of `tpep_pickup_datetime`, `tpep_dropoff_datetime`
         print('Data type of tpep_pickup_datetime:', df['tpep_pickup_datetime'].dtype)
         print('Data type of tpep_dropoff_datetime:', df['tpep_dropoff_datetime'].dtype)

         df.head(3)

Data type of tpep_pickup_datetime: object
Data type of tpep_dropoff_datetime: object
Data type of tpep_pickup_datetime: datetime64[ns]
Data type of tpep_dropoff_datetime: datetime64[ns]
```

Out[13]:

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocatio
0	24870114	2	2017-03-25 08:55:43	2017-03-25 09:09:47	6	3.34	1	N	100	
1	35634249	1	2017-04-11 14:53:28	2017-04-11 15:19:58	1	1.80	1	N	186	
2	106203690	1	2017-12-15 07:26:56	2017-12-15 07:34:08	1	1.00	1	N	262	

Build a Multiple Linear Regression Model

2

PACE: Analyze

Task 2c. Create duration column

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

```
In [14]: # Create `duration` column
         df['duration'] = (df['tpep_dropoff_datetime'] - df['tpep_pickup_datetime']) /np.timedelta64(1,'m')
```

Outliers

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

```
In [16]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             22699 non-null  int64
 1   VendorID               22699 non-null  int64
 2   tpep_pickup_datetime   22699 non-null  datetime64[ns]
 3   tpep_dropoff_datetime  22699 non-null  datetime64[ns]
 4   passenger_count        22699 non-null  int64
 5   trip_distance          22699 non-null  float64
 6   RatecodeID             22699 non-null  int64
 7   store_and_fwd_flag     22699 non-null  object
 8   PULocationID           22699 non-null  int64
 9   DOLocationID           22699 non-null  int64
 10  payment_type           22699 non-null  int64
 11  fare_amount            22699 non-null  float64
 12  extra                  22699 non-null  float64
 13  mta_tax                22699 non-null  float64
 14  tip_amount             22699 non-null  float64
 15  tolls_amount           22699 non-null  float64
 16  improvement_surcharge  22699 non-null  float64
 17  total_amount           22699 non-null  float64
 18  duration               22699 non-null  float64
dtypes: datetime64[ns](2), float64(9), int64(7), object(1)
memory usage: 3.3+ MB
```

Many of the features will not be used to fit the model, the most important columns to check for outliers are likely to be:

- trip_distance
- fare_amount
- duration

Build a Multiple Linear Regression Model

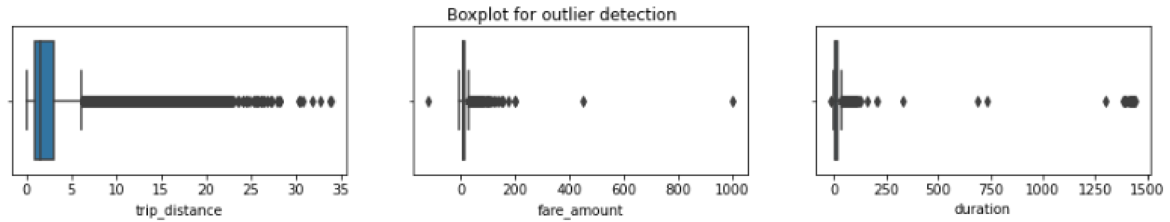
2

PACE: Analyze

Task 2d. Box plots

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

In [17]:
```
fig, axes = plt.subplots(1, 3, figsize=(15, 2))
fig.suptitle('Boxplot for outlier detection')
sns.boxplot(ax=axes[0], x=df['trip_distance'])
sns.boxplot(ax=axes[1], x=df['fare_amount'])
sns.boxplot(ax=axes[2], x=df['duration'])
plt.show()
```



1. **Which variable(s) contains outliers?**

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

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

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

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

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

Build a Multiple Linear Regression Model

2

PACE: Analyze

Task 2e. Imputations

trip_distance outliers

From the summary statistics that there are trip distances of 0. To check, sort the column values, eliminate duplicates, and inspect the least 10 values. Are they rounded values or precise values?

```
In [22]: # Are trip distances of 0 bad data or very short trips rounded down?
         sorted(set(df['trip_distance']))[:10]

Out[22]: [0.0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09]
```

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

Calculate the count of rides where the trip_distance is zero.

```
In [23]: sum(df['trip_distance']== 0)

Out[23]: 148
```

148 out of ~23,000 rides is relatively insignificant. I could impute it with a value of 0.01, but it's unlikely to have much of an effect on the model. Therefore, the trip_distance column will remain untouched with regard to outliers.

Build a Multiple Linear Regression Model

2

PACE: Analyze

Task 2e. Imputations

fare_amount outliers

```
In [24]: df['fare_amount'].describe()

Out[24]: count    22699.000000
         mean        13.026629
         std         13.243791
         min       -120.000000
         25%          6.500000
         50%          9.500000
         75%         14.500000
         max        999.990000
         Name: fare_amount, dtype: float64
```

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

- **Low values**: Negative values are problematic. Values of zero could be legitimate if the taxi logged a trip that was immediately canceled.
- **High values**: The maximum fare amount in this dataset is nearly $1,000, which seems very unlikely. High values for this feature can be capped based on intuition and statistics. The interquartile range (IQR) is $8. The standard formula of Q3 + (1.5 * IQR) yields $26.50. That doesn't seem appropriate for the maximum fare cap. In this case, we'll use a factor of 6, which results in a cap of $62.50.

Impute values less than $0 with `0`.

```
In [25]: # Impute values less than $0 with 0
         df.loc[df['fare_amount'] < 0, 'fare_amount'] = 0
         df['fare_amount'].min()

Out[25]: 0.0
```

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

```
In [26]: def outlier_imputer(column_list, iqr_factor):
             '''
             Impute upper-limit values in specified columns based on their interquartile range.

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

             The IQR is computed for each column in column_list and values exceeding
             the upper threshold for each column are imputed with the upper threshold value.
             '''
             for col in column_list:
                 # Reassign minimum to zero
                 df.loc[df[col] < 0, col] = 0

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

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

```
In [27]: outlier_imputer(['fare_amount'], 6)

         fare_amount
         q3: 14.5
         upper_threshold: 62.5
         count    22699.000000
         mean        12.897913
         std         10.541137
         min          0.000000
         25%          6.500000
         50%          9.500000
         75%         14.500000
         max         62.500000
         Name: fare_amount, dtype: float64
```

Build a Multiple Linear Regression Model

2

PACE: Analyze

Task 2e. Imputations

duration outliers

```
In [28]: # Call .describe() for duration outliers
         df['duration'].describe()

Out[28]: count    22699.000000
         mean        17.013777
         std         61.996482
         min        -16.983333
         25%          6.650000
         50%         11.183333
         75%         18.383333
         max       1439.550000
         Name: duration, dtype: float64
```

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

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

```
In [29]: # Impute a 0 for any negative values
         df.loc[df['duration'] < 0, 'duration'] = 0
         df['duration'].min()

Out[29]: 0.0

In [30]: # Impute the high outliers
         outlier_imputer(['duration'], 6)

         duration
         q3: 18.383333333333333
         upper_threshold: 88.78333333333333
         count    22699.000000
         mean        14.460555
         std         11.947043
         min          0.000000
         25%          6.650000
         50%         11.183333
         75%         18.383333
         max         88.783333
         Name: duration, dtype: float64
```

Build a Multiple Linear Regression Model

2

PACE: Analyze

Task 3a. Feature engineering

When deployed, the model will not know the duration of a trip until after the trip occurs, so I cannot train a model that uses this feature. Instead, I can use the statistics of trips that I know to generalize this feature.

- Create a column called mean_distance that captures the mean distance for each group of trips that share pickup and dropoff points.

In [34]:
```
# 1. Create a mean_distance column that is a copy of the pickup_dropoff helper column
df['mean_distance'] = df['pickup_dropoff']

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

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

Out[34]:

	mean_distance
0	3.521667
4909	3.521667
16636	3.521667
18134	3.521667
19761	3.521667
20581	3.521667

Build a Multiple Linear Regression Model

2

PACE: Analyze

Task 3a. Feature engineering

Create mean_duration column

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

In [35]:
```
grouped = df.groupby('pickup_dropoff').mean(numeric_only=True)[['duration']]
grouped

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

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

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

Out[35]:

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

Create day and month columns

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

In [36]:
```
# Create 'day' col
df['day'] = df['tpep_pickup_datetime'].dt.day_name().str.lower()

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

Build a Multiple Linear Regression Model

2

PACE: Analyze

Task 3a. Feature engineering

Create rush_hour column

Define rush hour as:

- Any weekday (not Saturday or Sunday) AND
- Either from 06:00–10:00 or from 16:00–20:00

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

```
In [37]: # Create 'rush_hour' col
df['rush_hour'] = df['tpep_pickup_datetime'].dt.hour

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

```
In [38]: 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
```

```
In [39]: # Apply the `rush_hourizer()` function to the new column
df.loc[(df.day != 'saturday') & (df.day != 'sunday'), 'rush_hour'] = df.apply(rush_hourizer, axis=1)
df.head()
```

Out[39]:

D	DOLocationID	...	tolls_amount	improvement_surcharge	total_amount	duration	pickup_dropoff	mean_distance	mean_duration	day	month	rush_hour
0	231	...	0.0	0.3	16.56	14.066667	100 231	3.521667	22.847222	saturday	mar	0
6	43	...	0.0	0.3	20.80	26.500000	186 43	3.108889	24.470370	tuesday	apr	0
2	236	...	0.0	0.3	8.75	7.200000	262 236	0.881429	7.250000	friday	dec	1
8	97	...	0.0	0.3	27.69	30.250000	188 97	3.700000	30.250000	sunday	may	0
4	112	...	0.0	0.3	17.80	16.716667	4 112	4.435000	14.616667	saturday	apr	0

Build a Multiple Linear Regression Model

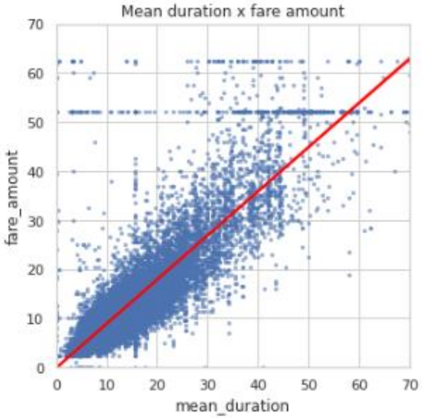
2

PACE: Analyze

Task 4. Scatter plot

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

In [40]:
```
# Create a scatterplot to visualize the relationship between variables of interest
sns.set(style='whitegrid')
f = plt.figure()
f.set_figwidth(5)
f.set_figheight(5)
sns.regplot(x=df['mean_duration'], y=df['fare_amount'],
            scatter_kws={'alpha':0.5, 's':5},
            line_kws={'color':'red'})
plt.ylim(0, 70)
plt.xlim(0, 70)
plt.title('Mean duration x fare amount')
plt.show()
```



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

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

In [41]:
```
df[df['fare_amount'] > 50]['fare_amount'].value_counts().head()
```

Out[41]:
```
52.0    514
62.5     84
59.0      9
50.5      9
57.5      8
Name: fare_amount, dtype: int64
```

There are 514 trips whose fares were $52.

Build a Multiple Linear Regression Model

2

PACE: Analyze

Task 4. Scatter plot

Examine the first 30 of these trips (with fare_amount 52).

In [42]:
```python
# Set pandas to display all columns
pd.set_option('display.max_columns', None)
df[df['fare_amount']==52].head(30)
```

Out[42]:

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLoc
11	18600059	2	2017-03-05 19:15:30	2017-03-05 19:52:18	2	18.90	2	N	236	
110	47959795	1	2017-06-03 14:24:57	2017-06-03 15:31:48	1	18.00	2	N	132	
161	95729204	2	2017-11-11 20:16:16	2017-11-11 20:17:14	1	0.23	2	N	132	
247	103404868	2	2017-12-06 23:37:08	2017-12-07 00:06:19	1	18.93	2	N	132	
379	80479432	2	2017-09-24 23:45:45	2017-09-25 00:15:14	1	17.99	2	N	132	
388	16226157	1	2017-02-28 18:30:05	2017-02-28 19:09:55	1	18.40	2	N	132	

- It seems that almost all of the trips in the first 30 rows where the fare amount was $52 either begin or end at location 132, and all of them have a RatecodeID of 2.
- There is no readily apparent reason why PULocation 132 should have so many fares of 52 dollars. They seem to occur on all different days, at different times, with both vendors, in all months. However, there are many toll amounts of $5.76 and $5.54. This would seem to indicate that location 132 is in an area that frequently requires tolls to get to and from. It's likely this is an airport.
- The data dictionary says that RatecodeID of 2 indicates trips for JFK, which is John F. Kennedy International Airport. A quick Google search for "new york city taxi flat rate $52" indicates that in 2017 (the year that this data was collected) there was indeed a flat fare for taxi trips between JFK airport (in Queens) and Manhattan.
- Because RatecodeID is known from the data dictionary, the values for this rate code can be imputed back into the data after the model makes its predictions. This way you know that those data points will always be correct.

Build a Multiple Linear Regression Model

2

PACE: Analyze

Task 5. Isolate modeling variables

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

In [43]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 25 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             22699 non-null  int64
 1   VendorID               22699 non-null  int64
 2   tpep_pickup_datetime   22699 non-null  datetime64[ns]
 3   tpep_dropoff_datetime  22699 non-null  datetime64[ns]
 4   passenger_count        22699 non-null  int64
 5   trip_distance          22699 non-null  float64
 6   RatecodeID             22699 non-null  int64
 7   store_and_fwd_flag     22699 non-null  object
 8   PULocationID           22699 non-null  int64
 9   DOLocationID           22699 non-null  int64
 10  payment_type           22699 non-null  int64
 11  fare_amount            22699 non-null  float64
 12  extra                  22699 non-null  float64
 13  mta_tax                22699 non-null  float64
 14  tip_amount             22699 non-null  float64
 15  tolls_amount           22699 non-null  float64
 16  improvement_surcharge  22699 non-null  float64
 17  total_amount           22699 non-null  float64
 18  duration               22699 non-null  float64
 19  pickup_dropoff         22699 non-null  object
 20  mean_distance          22699 non-null  float64
 21  mean_duration          22699 non-null  float64
 22  day                    22699 non-null  object
 23  month                  22699 non-null  object
 24  rush_hour              22699 non-null  int64
dtypes: datetime64[ns](2), float64(11), int64(8), object(4)
memory usage: 4.3+ MB
```

In [44]:
```
df2 = df.copy()

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

df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   VendorID         22699 non-null  int64
 1   passenger_count  22699 non-null  int64
 2   fare_amount      22699 non-null  float64
 3   mean_distance    22699 non-null  float64
 4   mean_duration    22699 non-null  float64
 5   rush_hour        22699 non-null  int64
dtypes: float64(3), int64(3)
memory usage: 1.0 MB
```

Build a Multiple Linear Regression Model

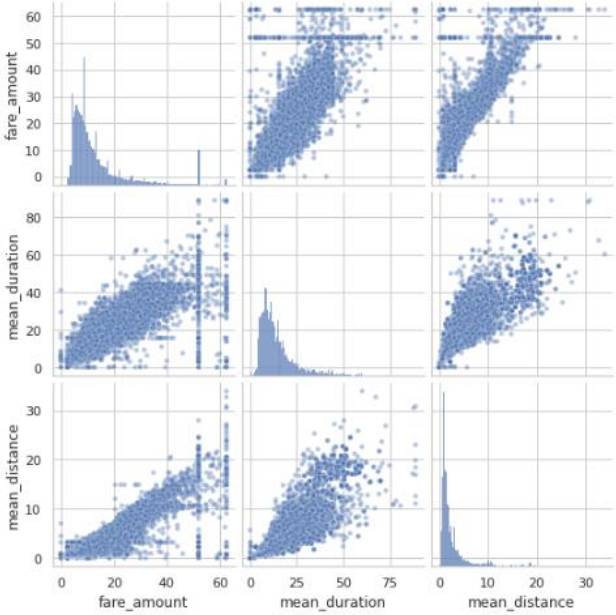
2

PACE: Analyze

Task 6. Pair Plot

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

```
In [45]: # Create a pairplot to visualize pairwise relationships between variables in the data
         sns.pairplot(df2[['fare_amount', 'mean_duration', 'mean_distance']],
                      plot_kws={'alpha':0.4, 'size':5}, );
```



These variables all show linear correlation with each other.

Build a Multiple Linear Regression Model

2

PACE: Analyze

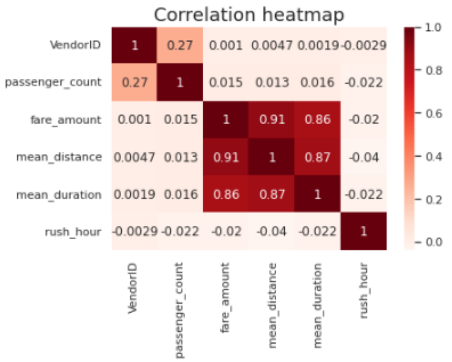
Task 7. Identify correlations

```
In [46]: # Correlation matrix to help determine most correlated variables
df2.corr(method='pearson')
```

Out[46]:

	VendorID	passenger_count	fare_amount	mean_distance	mean_duration	rush_hour
VendorID	1.000000	0.266463	0.001045	0.004741	0.001876	-0.002874
passenger_count	0.266463	1.000000	0.014942	0.013428	0.015852	-0.022035
fare_amount	0.001045	0.014942	1.000000	0.910185	0.859105	-0.020075
mean_distance	0.004741	0.013428	0.910185	1.000000	0.874864	-0.039725
mean_duration	0.001876	0.015852	0.859105	0.874864	1.000000	-0.021583
rush_hour	-0.002874	-0.022035	-0.020075	-0.039725	-0.021583	1.000000

```
In [47]: # Create correlation heatmap
plt.figure(figsize=(6,4))
sns.heatmap(df2.corr(method='pearson'), annot=True, cmap='Reds')
plt.title('Correlation heatmap',
          fontsize=18)
plt.show()
```



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

- mean_duration and mean_distance are both highly correlated with the target variable of fare_amount They're also both correlated with each other, with a Pearson correlation of 0.87.
- Recall that highly correlated predictor variables can be bad for linear regression models when you want to be able to draw statistical inferences about the data from the model. However, correlated predictor variables can still be used to create an accurate predictor if the prediction itself is more important than using the model as a tool to learn about your data.
- This model will predict fare_amount, which will be used as a predictor variable in machine learning models. Therefore, try modeling with both variables even though they are correlated.

Build a Multiple Linear Regression Model

3

PACE: Construct

Task 8a. Split data into outcome variable and features

After analysis and deriving variables with close relationships, it is time to begin constructing the model.

```
In [53]: df2.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   VendorID         22699 non-null  int64
 1   passenger_count  22699 non-null  int64
 2   fare_amount      22699 non-null  float64
 3   mean_distance    22699 non-null  float64
 4   mean_duration    22699 non-null  float64
 5   rush_hour        22699 non-null  int64
dtypes: float64(3), int64(3)
memory usage: 1.0 MB
```

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

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

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

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

Out[54]:

	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

Build a Multiple Linear Regression Model

3

PACE: Construct

Task 8b. Pre-process data

Dummy encode categorical variables

```
In [55]: # Convert VendorID to string
         X['VendorID'] = X['VendorID'].astype(str)

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

Out[55]:

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

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.

```
In [56]: # Create training and testing sets
         X_train, X_test, y_train, y_test =
         train_test_split(X, y, test_size=0.2, random_state=0)
```

Standardize the data

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

```
In [57]: # Standardize the X variables
         scaler = StandardScaler().fit(X_train)
         X_train_scaled = scaler.transform(X_train)
         print('X_train scaled:', X_train_scaled)

         X_train scaled: [[-0.50301524  0.8694684   0.17616665 -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]]
```

Fit the model

```
In [59]: # Fit your model to the training data
         lr=LinearRegression()
         lr.fit(X_train_scaled, y_train)

Out[59]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Build a Multiple Linear Regression Model

3

PACE: Construct

Task 8c. 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.

```
In [60]: # Evaluate the model performance on the training data
r_sq = lr.score(X_train_scaled, y_train)
print('Coefficient of determination:', r_sq)
y_pred_train = lr.predict(X_train_scaled)
print('R^2:', r2_score(y_train, y_pred_train))
print('MAE:', mean_absolute_error(y_train, y_pred_train))
print('MSE:', mean_squared_error(y_train, y_pred_train))
print('RMSE:',np.sqrt(mean_squared_error(y_train, y_pred_train)))
```

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

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.

```
In [61]: # Scale the X_test data
X_test_scaled = scaler.transform(X_test)
```

```
In [62]: # Evaluate the model performance on the testing data
r_sq_test = lr.score(X_test_scaled, y_test)
print('Coefficient of determination:', r_sq_test)
y_pred_test = lr.predict(X_test_scaled)
print('R^2:', r2_score(y_test, y_pred_test))
print('MAE:', mean_absolute_error(y_test,y_pred_test))
print('MSE:', mean_squared_error(y_test, y_pred_test))
print('RMSE:',np.sqrt(mean_squared_error(y_test, y_pred_test)))
```

```
Coefficient of determination: 0.8682583641795454
R^2: 0.8682583641795454
MAE: 2.1336549840593864
MSE: 14.326454156998944
RMSE: 3.785030271609323
```

Build a Multiple Linear Regression Model

4

PACE: Execute

Task 9a. Results

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

In [63]:
```
# 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()
```

Out[63]:

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

Build a Multiple Linear Regression Model

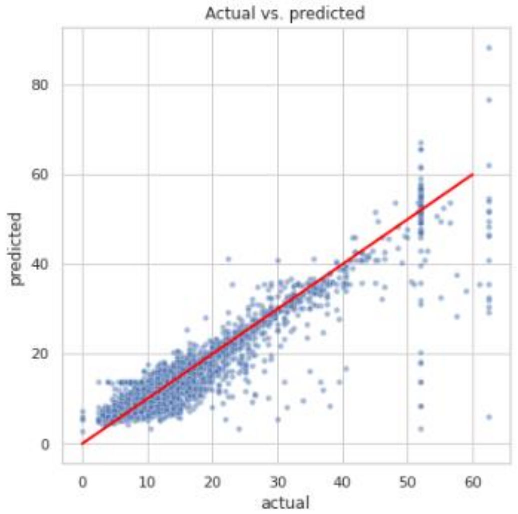
4

PACE: Execute

Task 9b. Visualize model results

Create a scatterplot to visualize actual vs. predicted

```
In [64]: # Create a scatterplot to visualize `predicted` over `actual`
fig, ax = plt.subplots(figsize=(6, 6))
sns.set(style='whitegrid')
sns.scatterplot(x='actual',
                y='predicted',
                data=results,
                s=20,
                alpha=0.5,
                ax=ax
)
# Draw an x=y line to show what the results would be if the model were perfect
plt.plot([0,60], [0,60], c='red', linewidth=2)
plt.title('Actual vs. predicted');
```



Build a Multiple Linear Regression Model

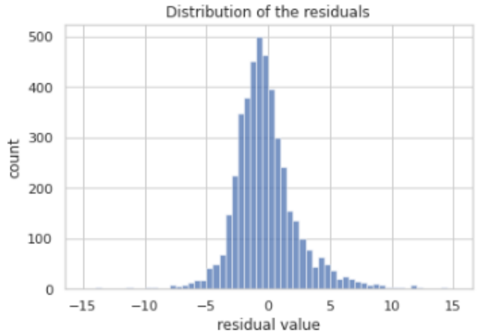
4

PACE: Execute

Task 9b. Visualize model results

Visualize the distribution of the residuals using a histogram.

```
In [65]: # Visualize the distribution of the `residuals`
         sns.histplot(results['residual'], bins=np.arange(-15,15.5,0.5))
         plt.title('Distribution of the residuals')
         plt.xlabel('residual value')
         plt.ylabel('count');
```

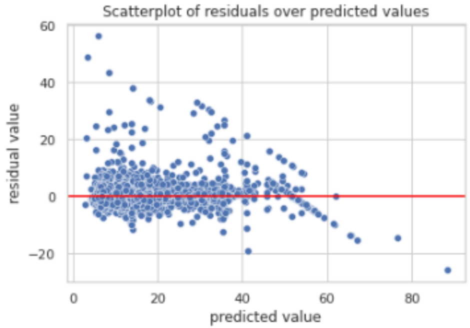



```
In [66]: # Calculate residual mean
         results['residual'].mean()

Out[66]: -0.01544262152868053
```

Create a scatterplot of residuals over predicted.

```
In [67]: # Create a scatterplot of `residuals` over `predicted`
         sns.scatterplot(x='predicted', y='residual', data=results)
         plt.axhline(0, c='red')
         plt.title('Scatterplot of residuals over predicted values')
         plt.xlabel('predicted value')
         plt.ylabel('residual value')
         plt.show()
```



Build a Multiple Linear Regression Model

4

PACE: Execute

Task 9c. Coefficients

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

```
In [68]: coefficients = pd.DataFrame(lr.coef_, columns=X.columns)
         coefficients
```

Out[68]:

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

The coefficients reveal that mean_distance was the feature with the greatest weight in the model's final prediction. A common misinterpretation is that for every mile traveled, the fare amount increases by a mean of $7.13. This is incorrect. Remember, the data used to train the model was standardized with StandardScaler(). As such, the units are no longer miles. **The correct interpretation** of this coefficient is: controlling for other variables, for every +1 change in standard deviation, the fare amount increases by a mean of $7.13.
(**Note:** because some highly correlated features were not removed, the confidence interval of this assessment is wider.)

```
In [69]: # 1. Calculate SD of `mean_distance` in X_train data
         print(X_train['mean_distance'].std())

         # 2. Divide the model coefficient by the standard deviation
         print(7.133867 / X_train['mean_distance'].std())

3.574812975256415
1.9955916713344426
```

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

Build a Multiple Linear Regression Model

4

PACE: Execute

Task 9d. Conclusion

What are the key takeaways from this Regression Model part?

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

I can discuss meeting linear regression assumptions, and present the MAE and RMSE scores obtained from the model.

Regression Assumptions After Modeling

Executive summary report for the New York City Taxi and Limousine Commission

ISSUE / PROBLEM

The New York City Taxi & Limousine Commission contracted Automatidata to predict taxi cab fares. In this part of the project, the Automatidata data team created the deliverable for the original ask from their client: a regression model.

RESPONSE

The Automatidata data team chose to create a multiple linear regression (MLR) model based on the type and distribution of data provided. The MLR model showed a successful model that estimates taxi cab fares prior to the ride.

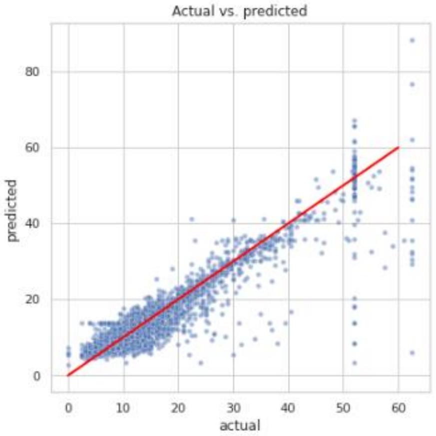
The model performance is high on both training and test sets, suggesting that the model is not over-biased and that the model is not overfit. The model performed better on the test data.

IMPACT

Imputing outliers optimized the model, specifically in regards to the variables of: fare amount and duration.

The linear regression model provides a sound framework for predicting the estimated fare amount for taxi rides.

In order to showcase the efficacy of the linear regression model, the Automatidata data team included a scatter plot comparing the predicted and actual fare amount. This model can be used to predict the fare amount of taxi cab rides with reasonable confidence. The provided notebook exhibits further analysis on the model residuals.



The scatter plot shows a linear regression model plot illustrating predicted and actual fare amount for taxi cab rides.

Model metrics:

Net model tuning resulted in:

- ✓ R^2 0.87, meaning that 86.8% of the variance is described by the model.
- ✓ MAE 2.1
- ✓ MSE: 14.36
- ✓ RMSE 3.8

KEY INSIGHTS

- The feature with the greatest effect on fare amount was ride duration, which was not unexpected. The model revealed a mean increase of $7 for each additional minute, however, this is not a reliable benchmark due to high correlation between some features.
- Request additional data from under-represented itineraries.
- The New York City Taxi and Limousine commission can use these findings to create an app that allows users (TLC riders) to see the estimated fare before their ride begins.
- The model provides a generally strong and reliable fare prediction that can be used in downstream modeling efforts.

Machine Learning Models

Ethical Considerations

Consider the ethical implications of the request

Feature Engineering

Perform feature selection, extraction, and transformation to prepare the data for modeling

Modeling

Build the models, evaluate them, and advise on next steps

Executive Summary

Summarize findings for Automatidata and the stakeholders at TLC

Machine Learning Models

1

PACE: Plan

The purpose of this model is to find ways to generate more revenue for taxi cab drivers.

The goal of this model is to predict whether or not a customer is a generous tipper.

What are you being asked to do?

Predict if a customer will not leave a tip.

Do the benefits of such a model outweigh the potential problems?

It's not good to disincentivize drivers from picking up customers. It could also cause a customer backlash. The problems seem to outweigh the benefits.

Would you proceed with the request to build this model? Why or why not?

No. Effectively limiting equal access to taxis is ethically problematic, and carries a lot of risk.

Can the objective be modified to make it less problematic?

We can build a model that predicts the most generous customers. This could accomplish the goal of helping taxi drivers increase their earnings from tips while preventing the wrongful exclusion of certain people from using taxis.

Machine Learning Models

1

PACE: Plan

Ethical Implications

What are the ethical implications of the model? What are the consequences of the model making errors?

When the model predicts a false negative (i.e., when the model says a customer will give a tip, but they actually won't)?

Drivers who didn't receive tips will probably be upset that the app told them a customer would leave a tip. If it happened often, drivers might not trust the app.

When the model predicts a false positive (i.e., when the model says a customer will not give a tip, but they actually will)?

Drivers are unlikely to pick up people who are predicted to not leave tips. Customers will have difficulty finding a taxi that will pick them up, and might get angry at the taxi company.

Even when the model is correct, people who can't afford to tip will find it more difficult to get taxis, which limits the accessibility of taxi service to those who pay extra.

Machine Learning **Models**

1

PACE: Plan

Modify the modeling by **predicted** people who are particularly generous—those who **will tip 20% or more** (*instead of predicting people who won't tip at all*).

What would be the target variable?

The target variable would be a binary variable (1 or 0) that indicates whether or not the customer is expected to tip ≥ 20%.

What features do you need to make this prediction?

- Ideally, we'd have behavioral history for each customer, so we could know how much they tipped on previous taxi rides.
- We'd also want times, dates, and locations of both pickups and dropoffs, estimated fares, and payment method.

What metric should you use to evaluate your model?

- This is a supervised learning, classification task.
- We could use accuracy, precision, recall, F-score, area under the ROC curve, or a number of other metrics.
- We need to know the class balance of the target variable.

Machine Learning Models

1

PACE: Plan

Task 1. Imports and data loading

In [1]:
```
# Import packages and libraries
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score, precision_score, recall_score,\
f1_score, confusion_matrix, ConfusionMatrixDisplay, RocCurveDisplay

from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

# This is the function that helps plot feature importance
from xgboost import plot_importance
```

In [2]:
```
# RUN THIS CELL TO SEE ALL COLUMNS
# This lets us see all of the columns, preventing Juptyer from redacting them.
pd.set_option('display.max_columns', None)
```

Begin by reading in the data. There are two dataframes: one containing the original data, the other containing the mean durations, mean distances, and predicted fares from the previous course's project called nyc_preds_means.csv.

Machine Learning Models

1

PACE: Plan

Task 1. Imports and data loading

In [3]:
```
# RUN THE CELL BELOW TO IMPORT YOUR DATA.

# 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')
```

Inspect the first few rows of `df0`.

In [4]:
```
# Inspect the first few rows of df0
df0.head()
```

Out[4]:

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID
0	24870114	2	03/25/2017 8:55:43 AM	03/25/2017 9:09:47 AM	6	3.34	1
1	35634249	1	04/11/2017 2:53:28 PM	04/11/2017 3:19:58 PM	1	1.80	1
2	106203690	1	12/15/2017 7:26:56 AM	12/15/2017 7:34:08 AM	1	1.00	1
3	38942136	2	05/07/2017 1:17:59 PM	05/07/2017 1:48:14 PM	1	3.70	1
4	30841670	2	04/15/2017 11:32:20 PM	04/15/2017 11:49:03 PM	1	4.37	1

Inspect the first few rows of `nyc_preds_means`.

In [5]:
```
# Inspect the first few rows of `nyc_preds_means`
nyc_preds_means.head()
```

Out[5]:

	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

- Join the two dataframes

In [6]:
```
# Merge datasets
df0 = df0.merge(nyc_preds_means,
                left_index=True,
                right_index=True)

df0.head()
```

Out[6]:

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocatic
0	24870114	2	03/25/2017 8:55:43 AM	03/25/2017 9:09:47 AM	6	3.34	1	N	100	
1	35634249	1	04/11/2017 2:53:28 PM	04/11/2017 3:19:58 PM	1	1.80	1	N	186	
2	106203690	1	12/15/2017 7:26:56 AM	12/15/2017 7:34:08 AM	1	1.00	1	N	262	
3	38942136	2	05/07/2017 1:17:59 PM	05/07/2017 1:48:14 PM	1	3.70	1	N	188	
4	30841670	2	04/15/2017 11:32:20 PM	04/15/2017 11:49:03 PM	1	4.37	1	N	4	

Machine Learning Models

2 PACE: Analyze

Task 2. Feature engineering

Call info() on the new combined dataframe from previous stage

```
In [7]: 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
 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
```

From EDA, that customers who pay cash generally have a tip amount of $0. To meet the modeling objective, I need to sample the data to select only the customers who pay with credit card.

- Copy df0 and assign the result to a variable called df1. Then, use a Boolean mask to filter df1 so it contains only customers who paid with credit card.

```
In [9]: # Subset the data to isolate only customers who paid by credit card
df1 = df0[df0['payment_type']==1]
```

Machine Learning Models

2

PACE: Analyze

Task 2. Feature engineering

I need to create the target variable cause there isn't a column that indicates tip percent. I'll have to engineer it by Add a tip_percent column to the dataframe.

```
In [11]: # Create tip % col
         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).

```
In [12]: # Create 'generous' col (target)
         df1['generous'] = df1['tip_percent']
         df1['generous'] = (df1['generous'] >= 0.2)
         df1['generous'] = df1['generous'].astype(int)
```

Machine Learning Models

2

PACE: Analyze

Task 2. Create day column

```
In [14]: # 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')
         # Create a 'day' col hat contains only the day of the week when each passenger was picked up.
         df1['day'] = df1['tpep_pickup_datetime'].dt.day_name().str.lower()
```

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)

```
In [15]: # 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
```

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.

Machine Learning Models

2 PACE: Analyze

Task 2. Create day column

Write four functions to convert each new column to binary (0/1).

In [17]:
```
# 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
# Apply 'am_rush' function to the 'am_rush' series
df1['am_rush'] = df1.apply(am_rush, axis=1)
df1['am_rush'].head()
```

Out[17]:
```
0    1
1    0
2    1
3    0
5    0
Name: am_rush, dtype: int64
```

In [18]:
```
# 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)
```

In [19]:
```
# Apply 'daytime()' function to the 'daytime' series
df1['daytime'] = df1.apply(daytime, axis=1)
```

In [22]:
```
# 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
```

In [23]:
```
# Apply 'nighttime' function to the 'nighttime' series
df1['nighttime'] = df1.apply(nighttime, axis=1)
```

In [20]:
```
# 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
```

In [21]:
```
# Apply 'pm_rush()' function to the 'pm_rush' series
df1['pm_rush'] = df1.apply(pm_rush, axis=1)
```

Machine Learning Models

2 PACE: Analyze

Task 2. Create month column

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.

In [24]:
```
# Create 'month' col
df1['month'] = df1['tpep_pickup_datetime'].dt.strftime('%b').str.lower()
```

Examine the first five rows of your dataframe.

In [25]:
```
df1.head()
```

Out[25]:

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocatic
0	24870114	2	2017-03-25 08:55:43	2017-03-25 09:09:47	6	3.34	1	N	100	
1	35634249	1	2017-04-11 14:53:28	2017-04-11 15:19:58	1	1.80	1	N	186	
2	106203690	1	2017-12-15 07:26:56	2017-12-15 07:34:08	1	1.00	1	N	262	
3	38942136	2	2017-05-07 13:17:59	2017-05-07 13:48:14	1	3.70	1	N	188	
5	23345809	2	2017-03-25 20:34:11	2017-03-25 20:42:11	6	2.30	1	N	161	

Machine Learning Models

2

PACE: Analyze

Task 2. 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.

```
In [27]: # 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
```

Machine Learning Models

2

PACE: Analyze

Task 2. 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.

In [28]:
```
# 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')
```

In [29]:
```
# 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
```

Machine Learning Models

2

PACE: Analyze

Task 2. Evaluation metric (Examine the class balance of your target variable)

```
In [30]: # Get class balance of 'generous' col
         df2['generous'].value_counts(normalize=True)

Out[30]: 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 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** (the model predicts a tip < 20%, but the customer gives more)

  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.

Machine Learning Models

3

PACE: Construct

Task 3. Modeling

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.

In [31]:
```
# 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)
```

Machine Learning Models

3

PACE: Construct

Task 3. Modeling

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'.

Machine Learning Models

PACE: Construct

3

Task 3. Modeling

Random Forest

In [32]:

```
# 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 list 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')
```

Machine Learning Models

3

PACE: Construct

Task 3. Modeling

Random Forest

Now fit the model to the training data.

```
In [32]: %%time
         rf1.fit(X_train, y_train)

CPU times: user 4min 36s, sys: 118 ms, total: 4min 36s
Wall time: 4min 36s

Out[32]:
GridSearchCV

GridSearchCV(cv=4, estimator=RandomForestClassifier(random_state=42),
             param_grid={'max_depth': [None], 'max_features': [1.0],
                         'max_samples': [0.7], 'min_samples_leaf': [1],
                         'min_samples_split': [2], 'n_estimators': [300]},
             refit='f1', scoring=['accuracy', 'precision', 'recall', 'f1'])

estimator: RandomForestClassifier
RandomForestClassifier(random_state=42)

RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Machine Learning Models

3

PACE: Construct

Task 3. Modeling

Random Forest

Use pickle to save my models and read them back in. This can be particularly helpful when performing a search over many possible hyperparameter values.

```
In [33]: import pickle

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

```
In [34]: 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)
```

```
In [35]: 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.

```
In [36]: # Examine best score
         rf1.best_score_

Out[36]: 0.7130669698017492
```

Examine the best combination of hyperparameters.

```
In [37]: rf1.best_params_

Out[37]: {'max_depth': None,
          'max_features': 1.0,
          'max_samples': 0.7,
          'min_samples_leaf': 1,
          'min_samples_split': 2,
          'n_estimators': 300}
```

Machine Learning Models

3

PACE: Construct

Task 3. Modeling
Random Forest

```
In [38]: def make_results(model_name:str, model_object, metric:str):
    '''
    Arguments:
    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
```

RF CV Results

Call `make_results()` on the GridSearch object.

```
In [39]: results = make_results('RF CV', rf1, 'f1')
         results
```

Out[39]:

	model	precision	recall	F1	accuracy
0	RF CV	0.674915	0.756067	0.713067	0.679905

This results produce an acceptable model across the board. Typically scores of 0.65 or better are considered acceptable.

Machine Learning Models

3

PACE: Construct

Task 3. Modeling

Random Forest

Try to improve the scores by use this model to predict on the test data. Assign the results to a variable called rf_preds.

```
In [40]: # 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.

```
In [41]: 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 = accuracy_score(y_test_data, preds)
    precision = precision_score(y_test_data, preds)
    recall = recall_score(y_test_data, preds)
    f1 = f1_score(y_test_data, preds)

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

    return table
```

RF Test Results

```
In [42]: # 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
```

Out[42]:

	model	precision	recall	F1	accuracy
0	RF CV	0.674915	0.756067	0.713067	0.679905
0	RF test	0.670436	0.774736	0.718822	0.680970

How do your test results compare to your validation results? **All scores increased by at most ~0.02.**

Machine Learning Models

3

PACE: Construct

Task 3. Modeling

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')

Machine Learning Models

3 PACE: Construct

Task 3. Modeling

XGBoost

In [43]:
```
# 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,
# 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 list 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.

In [44]:
```
%%time
xgb1.fit(X_train, y_train)
```

```
CPU times: user 23.3 s, sys: 116 ms, total: 23.4 s
Wall time: 12.2 s
```

Out[44]:
```
GridSearchCV
estimator: XGBClassifier
XGBClassifier
```

Get the best score from this model.

In [45]:
```
# Examine best score
xgb1.best_score_
```

Out[45]: `0.6949068999567092`

And the best parameters.

In [46]:
```
# Examine best parameters
xgb1.best_params_
```

Out[46]:
```
{'learning_rate': 0.1,
 'max_depth': 8,
 'min_child_weight': 2,
 'n_estimators': 500}
```

Machine Learning Models

3

PACE: Construct

Task 3. Modeling

XGBoost (XGB CV Results)

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

In [47]:
```
# 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
```

Out[47]:

	model	precision	recall	F1	accuracy
0	RF CV	0.674915	0.756067	0.713067	0.679905
0	RF test	0.670436	0.774736	0.718822	0.680970
0	XGB CV	0.670451	0.721375	0.694907	0.666639

XGB Test Results

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.

In [49]:
```
# 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
```

Out[49]:

	model	precision	recall	F1	accuracy
0	RF CV	0.674915	0.756067	0.713067	0.679905
0	RF test	0.670436	0.774736	0.718822	0.680970
0	XGB CV	0.670451	0.721375	0.694907	0.666639
0	XGB test	0.672278	0.745488	0.706993	0.674746

Compare these scores to the random forest test scores

The F1 score is ~0.014 lower than the random forest model. Both models are acceptable, but the random forest model is the champion.

Machine Learning Models

3

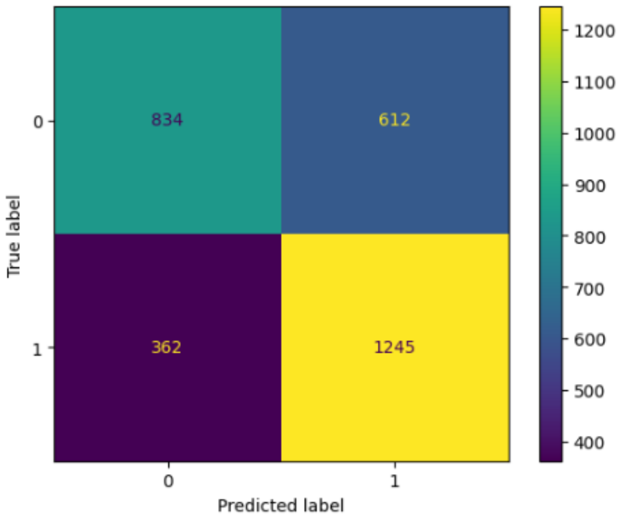
PACE: Construct

Task 3. Modeling

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

```
In [50]: # 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='');
```



What type of errors are more common for my model?

The model is almost twice as likely to predict a false positive than it is to predict a false negative. Therefore, type I errors are more common. This is less desirable, because it's better for a driver to be pleasantly surprised by a generous tip when they weren't expecting one than to be disappointed by a low tip when they were expecting a generous one. However, the overall performance of this model is satisfactory.

Machine Learning Models

3

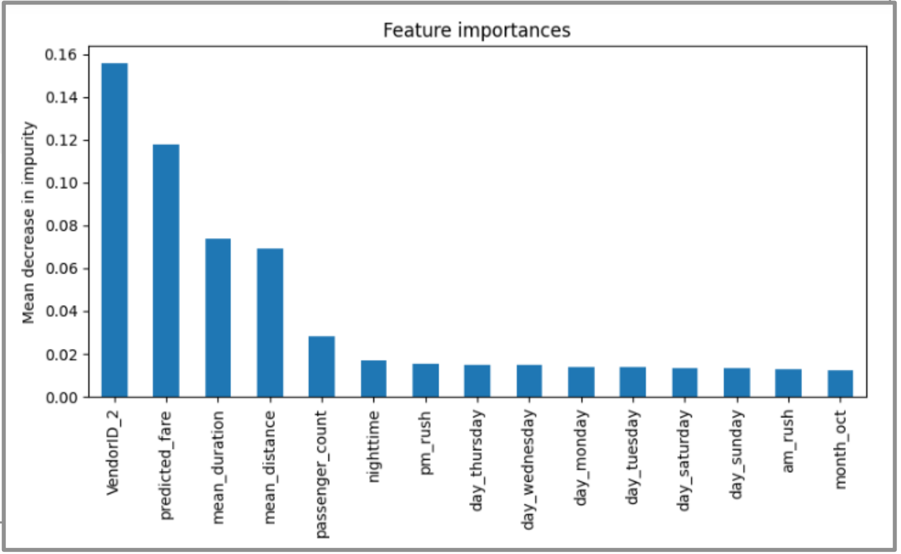
PACE: Construct

Task 3. Modeling

Feature importance (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.)

```
In [51]: 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();
```



Machine Learning Models

4

PACE: Execute

Task 4. Conclusion

1. **Would you recommend using this model? Why or why not?**

   Yes, this is model performs acceptably. Its F1 score was 0.7235 and it had an overall accuracy of 0.6865. It correctly identified ~78% of the actual responders in the test set, which is 48% better than a random guess. It may be worthwhile to test the model with a select group of taxi drivers to get feedback.

2. **What was your highest scoring model doing? Can you explain how it was making predictions?**

   Unfortunately, random forest is not the most transparent machine learning algorithm. We know that VendorID, predicted_fare, mean_duration, and mean_distance are the most important features, but we don't know how they influence tipping. This would require further exploration. It is interesting that VendorID is the most predictive feature. This seems to indicate that one of the two vendors tends to attract more generous customers. It may be worth performing statistical tests on the different vendors to examine this further.

Machine Learning Models

4

PACE: Execute

Task 4. Conclusion

3. **Are there new features that you can engineer that might improve model performance?**
 There are almost always additional features that can be engineered, but hopefully the most obvious ones were generated during the first round of modeling. In our case, we could try creating three new columns that indicate if the trip distance is short, medium, or far. We could also engineer a column that gives a ratio that represents (the amount of money from the fare amount to the nearest higher multiple of $5) / fare amount. For example, if the fare were $12, the value in this column would be 0.25, because $12 to the nearest higher multiple of $5 ($15) is $3, and $3 divided by $12 is 0.25. The intuition for this feature is that people might be likely to simply round up their tip, so journeys with fares with values just under a multiple of $5 may have lower tip percentages than those with fare values just over a multiple of $5. We could also do the same thing for fares to the nearest $10

$$round5\_ratio = \frac{amount\ of\ money\ from\ the\ fare\ amount\ to\ the\ nearest\ higher\ multiple\ of\ \$5}{fare\ amount}$$

4. **What features would you want to have that would likely improve the performance of your model?**
 It would probably be very helpful to have past tipping behavior for each customer. It would also be valuable to have accurate tip values for customers who pay with cash. It would be helpful to have a lot more data. With enough data, we could create a unique feature for each pickup/dropoff combination.

Machine Learning Model Outcomes

Executive summary report for the New York City Taxi and Limousine Commission

Overview

New York City Taxi & Limousine Commission has contracted the Automatidata data team to build a machine learning model to predict whether a NYC TLC taxi cab rider will be a generous tipper.

Problem

After rejecting the initial modeling objective (predicting non-tippers) out of ethical concern, it was decided to predict "generous" tippers—those who tip ≥ 20%. This decision was made to balance the sometimes competing interests of taxi drivers and potential passengers.

Solution

The data team used two different modeling architectures and compared their results. Both models performed acceptably, with a random forest architecture yielding slightly better predictions. As a result, the team would recommend beta testing with taxi drivers to gain further feedback.

Details

Behind the data

- The data team's assumption was that a trip's itinerary, predicted fare amount, and time of day may have a strong enough relationship with tip amount that we could accurately predict generous tipping.
- After the data team built the identified models and performed the testing, it is clear that these factors do indeed help predict tipping. The model's F_1 score was 0.7235.

Results Summary

The resulting algorithm is usable to predict riders who might be generous tippers, with reasonably strong precision, recall, F_1, and overall accuracy scores. Refer to the "next steps" section for suggestions.

	model	precision	recall	F1	accuracy
0	RF CV	0.674915	0.756067	0.713067	0.679905
0	RF test	0.670436	0.774736	0.718822	0.680970
0	XGB CV	0.670451	0.721375	0.694907	0.666639
0	XGB test	0.672278	0.745488	0.706993	0.674746

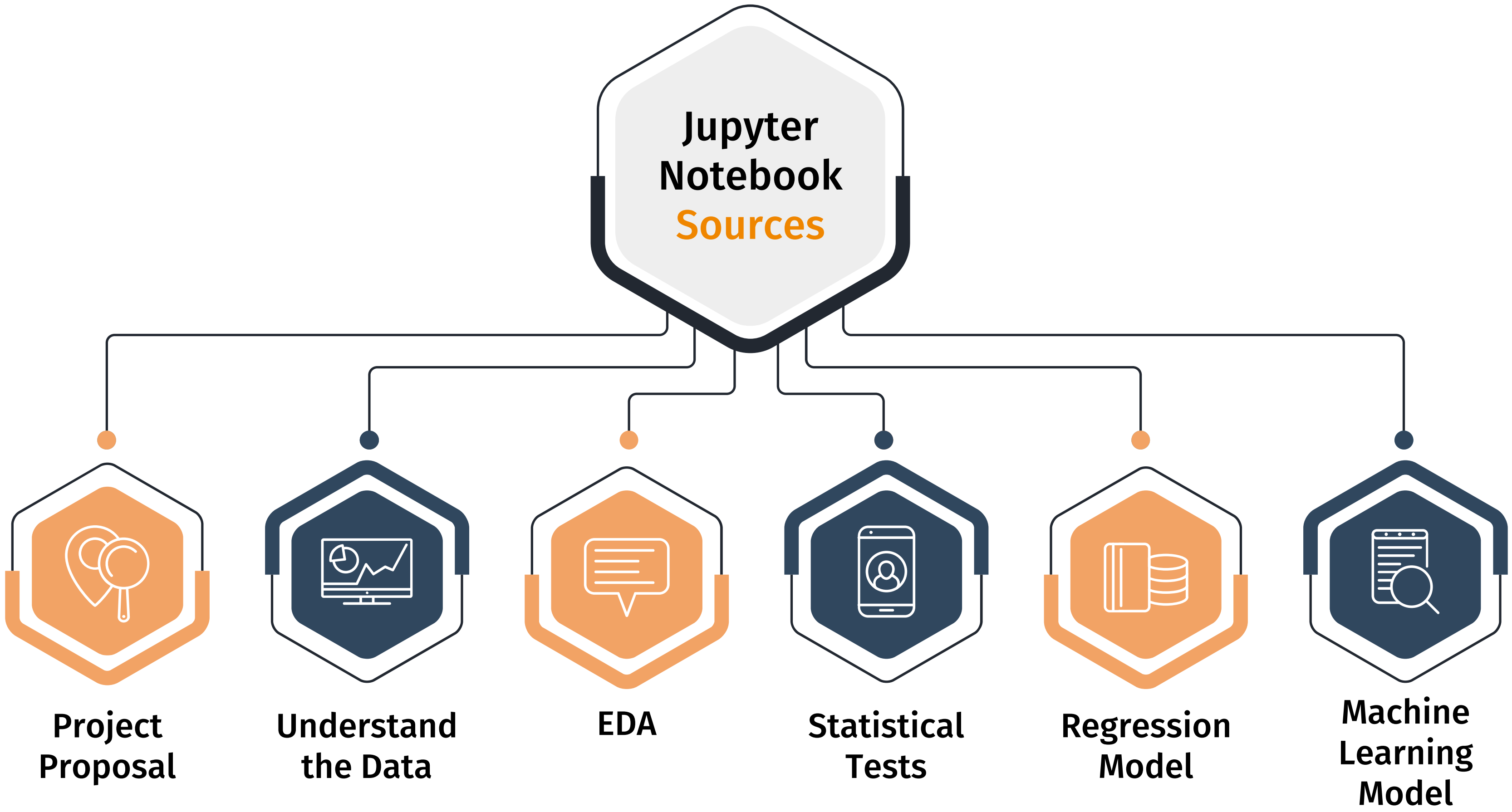
F1 scores for random forest and XGboost models

Future model suggestions

- Collect/add more granular driver and user-level data, including past tipping behavior.
- Cluster with K-means and analyze the clusters to derive insights from the data

Next Steps

As a next step, the Automatidata data team can consult the New York City Taxi and Limousine commission to share the model results and recommend that the model could be used as an indicator of tip amount. However, additional data would be needed to realize significant improvement to the model.


Jupyter Notebook Sources
Project Proposal
Understand the Data
EDA
Statistical Tests
Regression Model
Machine Learning Model

Thank You!

Project

Automatidata: **Predict** the fare amount for taxi cab rides

Date

17 July 2025

Google

Google Advanced Data Analytics Certification

GitHub Portfolio

Author

Ismi Ana Sulasiyah

annaismi17@gmail.com

LinkedIn Ismi