How Does a Bike-Share Navigate Speedy Success?

![drawing]

Scenario

You are a `junior data analyst` working in the marketing analyst team at `Cyclistic`, a bike-share company in Chicago. The director of marketing believes the company's future `success` depends on `maximizing the number of annual memberships`. Therefore, your team `wants to understand how casual riders and annual members use Cyclistic bikes differently`. From these insights, your team will design a new marketing strategy `to convert casual riders into annual members`. But first, Cyclistic executives must approve your recommendations, so they must be backed up with compelling data insights and professional data visualizations.

Characters and teams

- `Cyclistic`: A `bike-share program` that features more than 5,800 bicycles and 600 docking stations. Cyclistic sets itself apart by also offering reclining bikes, hand tricycles, and cargo bikes, making bike-share more inclusive to people with disabilities and riders who can't use a standard two-wheeled bike. The majority of riders opt for traditional bikes; about 8% of riders use the assistive options. Cyclistic users are more likely to ride for leisure, but about 30% use them to commute to work each day.
- `Lily Moreno`: The `director` of marketing and your `manager`. Moreno is `responsible for the development of campaigns and initiatives to promote the bike-share program`. These may include email, social media, and other channels.
- `Cyclistic marketing analytics team`: A `team of data analysts` who are `responsible` for collecting, analyzing, and reporting data that helps guide Cyclistic marketing strategy. You joined this team six months ago and have been busy learning about Cyclistic's mission and business goals — as well as how you, as a junior data analyst, can help Cyclistic achieve them.
- `Cyclistic executive team`: The notoriously detail-oriented executive team will `decide whether to approve the recommended marketing program`.

About the company

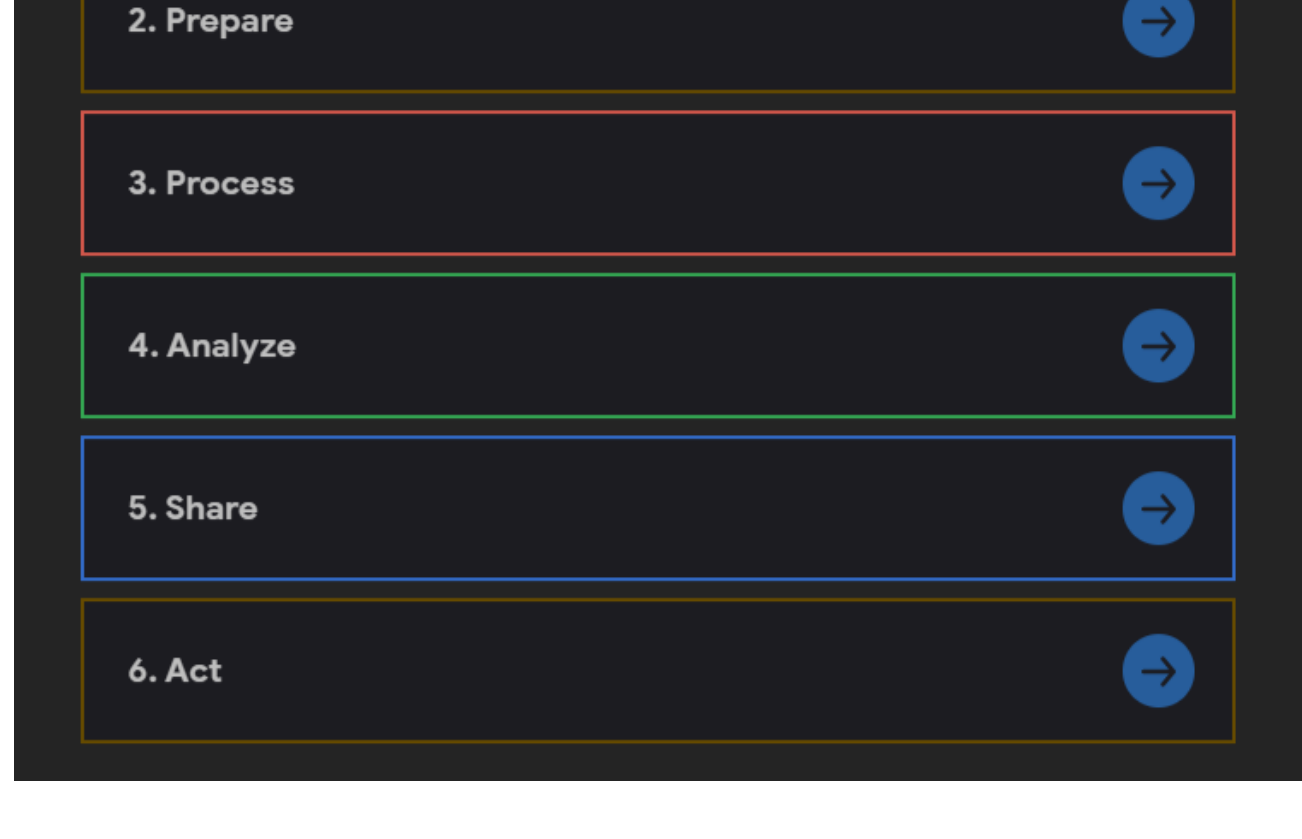
In 2016, Cyclistic launched a successful bike-share offering. Since then, the program has grown to a fleet of 5,824 bicycles that are geotracked and locked into a network of 692 stations across Chicago. The bikes can be unlocked from one station and returned to any other station in the system anytime. Until now, Cyclistic's marketing strategy relied on building general awareness and appealing to broad consumer segments. One approach that helped make these things possible was the flexibility of its pricing plans: `single-ride passes, full-day passes, and annual memberships`. Customers who purchase single-ride or full-day passes are referred to as casual riders. Customers who purchase annual memberships are Cyclistic members.

Cyclistic's finance analysts have `concluded that annual members are much more profitable than casual riders`. Although the pricing flexibility helps Cyclistic attract more customers, `Moreno believes that maximizing the number of annual members will be key to future growth`. Rather than creating a marketing campaign that targets all-new customers, `Moreno believes there is a very good chance to convert casual riders into members`. She notes that casual riders are already aware of the Cyclistic program and have chosen Cyclistic for their mobility needs.

Moreno has set a clear goal:

- Design marketing strategies aimed at `converting casual riders into annual members`. In order to do that, however, the marketing analyst team needs to better understand `how annual members and casual riders differ`, `why casual riders would buy a membership`, and `how digital media could affect their marketing tactics`. Moreno and her team are interested in analyzing the Cyclistic historical bike trip data to identify trends.

Roadmap For Data Analysis Process



Three questions will guide the future marketing program:

1. How do annual members and casual riders use Cyclistic bikes differently?
2. Why would casual riders buy Cyclistic annual memberships?
3. How can Cyclistic use digital media to influence casual riders to become members?

You will produce a report with the following deliverables:

1. A clear statement of the business task
2. A description of all data sources used
3. Documentation of any cleaning or manipulation of data
4. A summary of your analysis
5. Supporting visualizations and key findings
6. Your top three recommendations based on your analysis

Ask

Guiding questions

- what is the problem you are trying to solve?
- How can your insights drive business decisions?

Key tasks

- Identify the business task
- Consider key stakeholders

Moreno has assigned you the first question to answer:

How do annual members and casual riders use Cyclistic bikes differently

So our problem which we are trying to solve is

- How do annual members and casual riders use Cyclistic bikes differently?

If we answer this question we can help Moreno weather her claim is valid or not. `Because Moreno believes there is a very good chance to convert casual riders into members` she also believes that maximizing the number of annual members will be key to future growth.

Key stakeholders

includes Cyclistic executive team, Director of Marketing (Lily Moreno), Marketing Analytics team.

Now the business task is clear what we suppose to do and the we knows who are the stakeholders.

Let's go to the next step of data analysis process.

Prepare

You will use Cyclistic's historical trip data to analyze and identify trends. Download the previous 12 months of Cyclistic trip data here. Download from the 202004-divvy-tripdata.zip to 202103-divvy-tripdata.zip all the 12 zip files.

(Note: The datasets have a different name because Cyclistic is a fictional company. For the purposes of this case study, the datasets are appropriate and will enable you to answer the business questions. The data has been made available by Motivate International Inc. under this license.) This is public data that you can use to explore how different customer types are using Cyclistic bikes. But note that data-privacy issues prohibit you from using riders' personally identifiable information. This means that you won't be able to connect pass purchases to credit card numbers to determine if casual riders live in the Cyclistic service area or if they have purchased multiple single passes.

Now, prepare your data for analysis using the following Case Study Roadmap as a guide:

Prepare

Guiding question

- Where is your data located?
- How is the data organized?
- Are there issues with bias or credibility in this data?
- Does your data ROCCC?
- How are you addressing licensing, privacy, security, and accessibility?
- How did you verify the data's integrity?
- How does it help you answer your question?
- Are there any problems with the data?

Key tasks

- Download data and store it appropriately.
- Identify how it's organized.
- Sort and filter the data.
- Determine the credibility of the data.

Deliverable

- A description of all data sources used

Key tasks:

- The data has been downloaded from Motivate International Incand store appropriately in my local computer and google drive.
- identify how the data is organized.

Let's do this

In [1]:
```
import pandas as pd
y20_4 = pd.read_csv("202004-divvy-tripdata.csv")
y20_4.columns
```

Out[1]:
```
Index(['ride_id', 'rideable_type', 'started_at', 'ended_at',
       'start_station_name', 'start_station_id', 'end_station_name',
       'end_station_id', 'start_lat', 'start_lng', 'end_lat', 'end_lng',
       'member_casual'],
      dtype='object')
```

In [2]:
```
import pandas as pd
y21_3 = pd.read_csv("202103-divvy-tripdata.csv")
y21_3.columns
```

Out[2]:
```
Index(['ride_id', 'rideable_type', 'started_at', 'ended_at',
       'start_station_name', 'start_station_id', 'end_station_name',
       'end_station_id', 'start_lat', 'start_lng', 'end_lat', 'end_lng',
       'member_casual'],
      dtype='object')
```

Let's check few record from the dataset.

In [3]:
```
y21_3.head
```

Out[3]:

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_station_name
0	CFA86D4455AA1030	classic_bike	2021-03-16 08:32:30	2021-03-16 08:36:34	Humboldt Blvd & Armitage Ave	15651	Stave St & Armitage A
1	30D9DC61227D1AF3	classic_bike	2021-03-28 01:26:28	2021-03-28 01:36:55	Humboldt Blvd & Armitage Ave	15651	Central Park Ave & Bloomingdale A
2	846D87A15682A284	classic_bike	2021-03-11 21:17:29	2021-03-11 21:33:53	Shields Ave & 28th Pl	15443	Halsted St & 35
3	994D05AA75A168F2	classic_bike	2021-03-11 13:26:42	2021-03-11 13:55:41	Winthrop Ave & Lawrence Ave	TA1308000021	Broadway & Sheridan
4	DF7464FBE92D8308	classic_bike	2021-03-21 09:09:37	2021-03-21 09:27:33	Glenwood Ave & Touhy Ave	525	Chicago Ave & Sheridan

All trip data is in comma-delimited (.CSV) format with 13 columns, including:

```
'ride_id', 'rideable_type', 'started_at', 'ended_at',
    'start_station_name', 'start_station_id', 'end_station_name',
    'end_station_id', 'start_lat', 'start_lng', 'end_lat', 'end_lng',
    'member_casual'
```

Determine the credibility of the data.

Due to the fact that this is a case study using public data, and the data is recommended by well know organization Google so we are going to assume the data is credible.

Let's move on to the next step of data analysis process.

Process

Guiding questions

- What tools are you choosing and why?
- Have you ensured your data's integrity?
- What steps have you taken to ensure that your data is clean?
- How can you verify that your data is clean and ready to analyze?
- Have you documented your cleaning process so you can review and share those results?

Key tasks

- Check the data for errors.
- Choose your tools.
- Transform the data so you can work with it effectively.
- Document the cleaning process.

Deliverable

- Documentation of any cleaning or manipulation of data

Follow these steps:

- Follow these instructions for either Excel (a) or Google Sheets (b)
- Create a column called "ride_length." Calculate the length of each ride by subtracting the column "started_at" from the column "ended_at" (for example, =ended_at-started_at).
- Create a column called "day_of_week," and calculate the day of the week that each ride that 1 Sunday and 7 = Saturday.
- Proceed to the analyze step.

Key tasks

1. Check the data for errors.

The code chunk below will import 12 individual .csv files as data frames, each representing 1 of the last 12 months of trip data. Some parsing errors persist, however, this represents less 0.25% of the data set, so this is still a representative sample.

In [4]:
```
import pandas as pd      # pandas is mainly used for data analysis
# importing data from various file formats such as comma-separated values,
JSON, SQL, Microsoft Excel.
```

In [5]:
```
y20_4 = pd.read_csv("202004-divvy-tripdata.csv")      # the variable y20_
represent year_2020_April
y20_5 = pd.read_csv("202005-divvy-tripdata.csv")
y20_6 = pd.read_csv("202006-divvy-tripdata.csv")
y20_7 = pd.read_csv("202007-divvy-tripdata.csv")
y20_8 = pd.read_csv("202008-divvy-tripdata.csv")
y20_9 = pd.read_csv("202009-divvy-tripdata.csv")
y20_10 = pd.read_csv("202010-divvy-tripdata.csv")
y20_11 = pd.read_csv("202011-divvy-tripdata.csv")
y20_12 = pd.read_csv("202012-divvy-tripdata.csv")
y21_1 = pd.read_csv("202101-divvy-tripdata.csv")
y21_2 = pd.read_csv("202102-divvy-tripdata.csv")
y21_3 = pd.read_csv("202103-divvy-tripdata.csv")
```

1. Choose your tools

For this analysis, we will be using `SQL` and `Python` for it's easy statistical analysis tools and data visualizations.

1. Transform the data so you can work with it effectively.

Let's check the data type.

In [6]:
```
print("year 2020 April:\n",y20_4.dtypes)
print()
print("year 2021 March:\n",y21_2.dtypes)
```

```
year 2020 April:
 ride_id                object
rideable_type          object
started_at             object
ended_at               object
start_station_name     object
start_station_id        int64
end_station_name       object
end_station_id        float64
start_lat             float64
start_lng             float64
end_lat               float64
end_lng               float64
member_casual          object
dtype: object

year 2021 March:
 ride_id                object
rideable_type          object
started_at             object
ended_at               object
start_station_name     object
start_station_id       object
end_station_name       object
end_station_id         object
start_lat             float64
start_lng             float64
end_lat               float64
end_lng               float64
member_casual          object
dtype: object
```

In [7]:
```
print(y20_4['end_station_id'].dtypes)
```

```
float64
```

In [8]:
```
print(y21_1['end_station_id'].dtypes)
```

```
object
```

In [9]:
```
print(y21_1['start_station_id'].dtypes)
```

```
object
```

In [10]:
```
print(y20_4['start_station_id'].dtypes)
```

```
int64
```

In [11]:
```
print(y20_7['start_station_id'].dtypes)
```

```
float64
```

We can see that the the start_station_id and the end_station_id data type have some probem in some dataset it data type is float in some it's datatype is object or int, and we need to make sure that the data type in all the datasets are the same and then we will make one large datast to perform our analysis.

Let's convert the end_station_id and start_station_id to int type in all the datasets to make it unique data type.

In [13]:
```
print(y20_4['end_station_id'].dtypes)
print(y20_4['start_station_id'].dtypes)
y20_4[['end_station_id', 'start_station_id']] = y20_4[['end_station_id',
'start_station_id']].astype(int)
```

```
float64
int64
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-4ace45e85ff4> in <module>
      1 print(y20_4['end_station_id'].dtypes)
      2 print(y20_4['start_station_id'].dtypes)
----> 3 y20_4[['end_station_id', 'start_station_id']] = y20_4[['end_station_id', 'star
t_station_id']].astype(int)

~\anaconda3\lib\site-packages\pandas\core\generic.py in astype(self, dtype, copy, erro
rs)
   5875         else:
   5876             # else, only a single dtype is given
-> 5877             new_data = self._mgr.astype(dtype=dtype, copy=copy, errors=errors)
   5878             return self._constructor(new_data).__finalize__(self, method="asty
pe")
   5879

~\anaconda3\lib\site-packages\pandas\core\internals\managers.py in astype(self, dtype,
 copy, errors)
    629         self, dtype, copy: bool = False, errors: str = "raise"
    630     ) -> "BlockManager":
--> 631         return self.apply("astype", dtype=dtype, copy=copy, errors=errors)
    632
    633     def convert(

~\anaconda3\lib\site-packages\pandas\core\internals\managers.py in apply(self, f, alig
n_keys, ignore_failures, **kwargs)
    425                     applied = b.apply(f, **kwargs)
    426                 else:
--> 427                     applied = getattr(b, f)(**kwargs)
    428             except (TypeError, NotImplementedError):
    429                 if not ignore_failures:

~\anaconda3\lib\site-packages\pandas\core\internals\blocks.py in astype(self, dtype, c
opy, errors)
    671             vals1d = values.ravel()
    672             try:
--> 673                 values = astype_nansafe(vals1d, dtype, copy=True)
    674             except (ValueError, TypeError):
    675                 # e.g. astype_nansafe can fail on object-dtype of strings

~\anaconda3\lib\site-packages\pandas\core\dtypes\cast.py in astype_nansafe(arr, dtype,
 copy, skipna)
   1066
   1067         if not np.isfinite(arr).all():
-> 1068             raise ValueError("Cannot convert non-finite values (NA or inf) to
 integer")
   1069
   1070     elif is_object_dtype(arr):

ValueError: Cannot convert non-finite values (NA or inf) to integer
```

In [14]:
```
print(y20_5['end_station_id'].dtypes)
print(y20_5['start_station_id'].dtypes)
y20_5[['end_station_id', 'start_station_id']] = y20_5[['end_station_id',
'start_station_id']].astype(int)
```

```
float64
int64
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-14-965775f3e58d> in <module>
      1 print(y20_5['end_station_id'].dtypes)
      2 print(y20_5['start_station_id'].dtypes)
----> 3 y20_5[['end_station_id', 'start_station_id']] = y20_5[['end_station_id', 'star
t_station_id']].astype(int)

~\anaconda3\lib\site-packages\pandas\core\generic.py in astype(self, dtype, copy, erro
rs)
   5875         else:
   5876             # else, only a single dtype is given
-> 5877             new_data = self._mgr.astype(dtype=dtype, copy=copy, errors=errors)
   5878             return self._constructor(new_data).__finalize__(self, method="asty
pe")
   5879

~\anaconda3\lib\site-packages\pandas\core\internals\managers.py in astype(self, dtype,
 copy, errors)
    629         self, dtype, copy: bool = False, errors: str = "raise"
    630     ) -> "BlockManager":
--> 631         return self.apply("astype", dtype=dtype, copy=copy, errors=errors)
    632
    633     def convert(

~\anaconda3\lib\site-packages\pandas\core\internals\managers.py in apply(self, f, alig
n_keys, ignore_failures, **kwargs)
    425                     applied = b.apply(f, **kwargs)
    426                 else:
--> 427                     applied = getattr(b, f)(**kwargs)
    428             except (TypeError, NotImplementedError):
    429                 if not ignore_failures:

~\anaconda3\lib\site-packages\pandas\core\internals\blocks.py in astype(self, dtype, c
opy, errors)
    671             vals1d = values.ravel()
    672             try:
--> 673                 values = astype_nansafe(vals1d, dtype, copy=True)
    674             except (ValueError, TypeError):
    675                 # e.g. astype_nansafe can fail on object-dtype of strings

~\anaconda3\lib\site-packages\pandas\core\dtypes\cast.py in astype_nansafe(arr, dtype,
 copy, skipna)
   1066
   1067         if not np.isfinite(arr).all():
-> 1068             raise ValueError("Cannot convert non-finite values (NA or inf) to
 integer")
   1069
   1070     elif is_object_dtype(arr):

ValueError: Cannot convert non-finite values (NA or inf) to integer
```

You can see that we have problem

`ValueError: Cannot convert non-finite values (NA or inf) to integer`

It seems that we have some missing values in our dataset to solve this problem here I am going to simply dorop the missing values from our datasets.

let's do this.

In [15]:
```
y20_5.isnull().sum()
```

Out[15]:
```
ride_id                 0
rideable_type           0
started_at              0
ended_at                0
start_station_name      0
start_station_id        0
end_station_name      321
end_station_id        321
start_lat               0
start_lng               0
end_lat               321
end_lng               321
member_casual           0
dtype: int64
```

We can see that we have 321 missing values in 4 different columns let's drop this.

In [16]:
```
y20_5.dropna(inplace=True) # to drop all the missing vlaues from our
dataset, inplace=True  do operation inplace and return None.
```

Now let's check our dataset.

In [17]:
```
y20_5.isnull().sum()
```

Out[17]:
```
ride_id               0
rideable_type         0
started_at            0
ended_at              0
start_station_name    0
start_station_id      0
end_station_name      0
end_station_id        0
start_lat             0
start_lng             0
end_lat               0
end_lng               0
member_casual         0
dtype: int64
```

`Note` : there are different way you can handle missing values but for now we just droping all the missing values.

Now let's convert the datatype to int.

In [18]:
```
y20_5[['end_station_id', 'start_station_id']] = y20_5[['end_station_id',
'start_station_id']].astype(int)
```

In [19]:
```
print(y20_5['end_station_id'].dtypes)
print(y20_5['start_station_id'].dtypes)
```

```
int32
int32
```

Let's do this for all the dataset.

In [20]:
```
y20_6.dropna(inplace=True)
y20_7.dropna(inplace=True)
y20_8.dropna(inplace=True)
y20_9.dropna(inplace=True)
y20_10.dropna(inplace=True)
y20_11.dropna(inplace=True)
y20_12.dropna(inplace=True)
y21_1.dropna(inplace=True)
y21_2.dropna(inplace=True)
y21_3.dropna(inplace=True)
```

you may use another method to handle the missing values you are free to do. you may create a function to drop all the missing values instead of doing manually.

Now let's change the dataype of start_station_id and end_station_id to int.

In [21]:
```
# we change the datatype of y20_4 and y20_5 already so we don't need to do
it.
y20_6[['end_station_id', 'start_station_id']] = y20_6[['end_station_id',
'start_station_id']].astype(int)
y20_7[['end_station_id', 'start_station_id']] = y20_7[['end_station_id',
'start_station_id']].astype(int)
y20_8[['end_station_id', 'start_station_id']] = y20_8[['end_station_id',
'start_station_id']].astype(int)
y20_9[['end_station_id', 'start_station_id']] = y20_9[['end_station_id',
'start_station_id']].astype(int)
y20_10[['end_station_id', 'start_station_id']] = y20_10[['end_station_id',
'start_station_id']].astype(int)
y20_11[['end_station_id', 'start_station_id']] = y20_11[['end_station_id',
'start_station_id']].astype(int)
y20_12[['end_station_id', 'start_station_id']] = y20_12[['end_station_id',
'start_station_id']].astype(int)
y21_1[['end_station_id', 'start_station_id']] = y21_1[['end_station_id',
'start_station_id']].astype(int)
y21_2[['end_station_id', 'start_station_id']] = y21_2[['end_station_id',
'start_station_id']].astype(int)
y21_3[['end_station_id', 'start_station_id']] = y21_3[['end_station_id',
'start_station_id']].astype(int)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-21-9cf38a3f75cc> in <module>
      6 y20_10[['end_station_id', 'start_station_id']] = y20_10[['end_station_id', 'st
art_station_id']].astype(int)
      7 y20_11[['end_station_id', 'start_station_id']] = y20_11[['end_station_id', 'st
art_station_id']].astype(int)
----> 8 y20_12[['end_station_id', 'start_station_id']] = y20_12[['end_station_id', 'st
art_station_id']].astype(int)
      9 y21_1[['end_station_id', 'start_station_id']] = y21_1[['end_station_id', 'star
t_station_id']].astype(int)
     10 y21_2[['end_station_id', 'start_station_id']] = y21_2[['end_station_id', 'star
t_station_id']].astype(int)

~\anaconda3\lib\site-packages\pandas\core\generic.py in astype(self, dtype, copy, erro
rs)
   5875         else:
   5876             # else, only a single dtype is given
-> 5877             new_data = self._mgr.astype(dtype=dtype, copy=copy, errors=errors)
   5878             return self._constructor(new_data).__finalize__(self, method="asty
pe")
   5879

~\anaconda3\lib\site-packages\pandas\core\internals\managers.py in astype(self, dtype,
 copy, errors)
    629         self, dtype, copy: bool = False, errors: str = "raise"
    630     ) -> "BlockManager":
--> 631         return self.apply("astype", dtype=dtype, copy=copy, errors=errors)
    632
    633     def convert(

~\anaconda3\lib\site-packages\pandas\core\internals\managers.py in apply(self, f, alig
n_keys, ignore_failures, **kwargs)
    425                     applied = b.apply(f, **kwargs)
    426                 else:
--> 427                     applied = getattr(b, f)(**kwargs)
    428             except (TypeError, NotImplementedError):
    429                 if not ignore_failures:

~\anaconda3\lib\site-packages\pandas\core\internals\blocks.py in astype(self, dtype, c
opy, errors)
    671             vals1d = values.ravel()
    672             try:
--> 673                 values = astype_nansafe(vals1d, dtype, copy=True)
    674             except (ValueError, TypeError):
    675                 # e.g. astype_nansafe can fail on object-dtype of strings

~\anaconda3\lib\site-packages\pandas\core\dtypes\cast.py in astype_nansafe(arr, dtype,
 copy, skipna)
   1072         # work around NumPy brokenness, #1987
   1073         if np.issubdtype(dtype.type, np.integer):
-> 1074             return lib.astype_intsafe(arr.ravel(), dtype).reshape(arr.shape)
   1075
   1076         # if we have a datetime/timedelta array of objects

pandas\_libs\lib.pyx in pandas._libs.lib.astype_intsafe()

ValueError: invalid literal for int() with base 10: 'TA1306000003'
```

We face a problem let's solve this problem.

`ValueError: invalid literal for int() with base 10: 'TA1306000003'`

from the error message we can see that the probem is in `y20_12`.

In [ ]:
```
print(y20_12['start_station_id'].dtypes)
print(y21_1['start_station_id'].dtypes)
print(y21_2['start_station_id'].dtypes)
print(y21_3['start_station_id'].dtypes)
```

Data type inconsistencies exist `start/end_station_id(12,01,02,03)`.

The start_station_id and end_station_id fields in 4 of the .csv files imported as the 'character' data type when they should have been numbers (y2020_12, y2021_01, y2021_02, y2021_03).

We will analyze these 4 files speratly.

You can use some technique to combine these datasets but I will analyze separte.

so we don't need to change the data type of this.

we will do for the remaining.

In [22]:
```
y20_6[['end_station_id', 'start_station_id']] = y20_6[['end_station_id',
'start_station_id']].astype(int)
y20_7[['end_station_id', 'start_station_id']] = y20_7[['end_station_id',
'start_station_id']].astype(int)
y20_8[['end_station_id', 'start_station_id']] = y20_8[['end_station_id',
'start_station_id']].astype(int)
y20_9[['end_station_id', 'start_station_id']] = y20_9[['end_station_id',
'start_station_id']].astype(int)
y20_10[['end_station_id', 'start_station_id']] = y20_10[['end_station_id',
'start_station_id']].astype(int)
y20_11[['end_station_id', 'start_station_id']] = y20_11[['end_station_id',
'start_station_id']].astype(int)
```

Now let's combine y20_4 to y20_11 into one dataset, and y20_12 to y21_3 to another dataset.

In [23]:
```
# use this syntx
# bigdata = pd.concat([data1, data2], ignore_index=True, sort=False)
tripdata1 = y20_4.append([y20_5,y20_6,y20_7,y20_8,y20_9,y20_10,y20_11],
ignore_index=True)
tripdata2 = y20_12.append([y21_1,y21_2,y21_3], ignore_index=True)
```

In [24]:
```
tripdata1.head(2)
```

Out[24]:

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_station_nam
0	A847FADBBC638E45	docked_bike	2020-04-26 17:45:14	2020-04-26 18:12:03	Eckhart Park	86	Lincoln Ave Diversey Pkw
1	5405B80E996FF60D	docked_bike	2020-04-17 17:08:54	2020-04-17 17:17:03	Drake Ave & Fullerton Ave	503	Kosciuszko Pa

Create a column called "ride_length." Calculate the length of each ride by subtracting the column "started_at" from the column "ended_at" (for example, =ended_at-started_at).

Let's first check the datatype of started_at and ended_at column.

In [25]:
```
print("started_at dataType:",tripdata1['started_at'].dtype)
print("ended_at dataType:",tripdata1['ended_at'].dtype)
```

```
started_at dataType: object
ended_at dataType: object
```

we can see that both are object let's perform the `ended_at - started_at` operation to create a new coloumn `ride_length`.

In [26]:
```
tripdata1['ride_length'] = tripdata1['ended_at'] - tripdata1['started_at']
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\anaconda3\lib\site-packages\pandas\core\ops\array_ops.py in _na_arithmetic_op(left,
 right, op, is_cmp)
    141     try:
--> 142         result = expressions.evaluate(op, left, right)
    143     except TypeError:

~\anaconda3\lib\site-packages\pandas\core\computation\expressions.py in evaluate(op,
 a, b, use_numexpr)
    234             # error: "None" not callable
--> 235             return _evaluate(op, op_str, a, b)  # type: ignore[misc]
    236     return _evaluate_standard(op, op_str, a, b)

~\anaconda3\lib\site-packages\pandas\core\computation\expressions.py in _evaluate_nume
xpr(op, op_str, a, b)
    119     if result is None:
--> 120         result = _evaluate_standard(op, op_str, a, b)
    121

~\anaconda3\lib\site-packages\pandas\core\computation\expressions.py in _evaluate_stan
dard(op, op_str, a, b)
     68     with np.errstate(all="ignore"):
---> 69         return op(a, b)
     70

TypeError: unsupported operand type(s) for -: 'str' and 'str'

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
<ipython-input-26-0a8ed9b343e6> in <module>
----> 1 tripdata1['ride_length'] = tripdata1['ended_at'] - tripdata1['started_at']

~\anaconda3\lib\site-packages\pandas\core\ops\common.py in new_method(self, other)
     63         other = item_from_zerodim(other)
     64
---> 65         return method(self, other)
     66
```

From the error we can see that they type is str, we need to convert it to date data type.

Let's do that.

```
In [27]: tripdata1['ended_at'] = pd.to_datetime(tripdata1['ended_at'])
         tripdata1['started_at'] = pd.to_datetime(tripdata1['started_at'])
```

Now let's check the datatype.

```
In [28]: print(tripdata1['started_at'].dtype)
         print(tripdata1['ended_at'].dtype)
```

Now we are ready to create a new column.

```
In [29]: tripdata1['ride_length'] = tripdata1['ended_at'] - tripdata1['started_at']
```

```
In [30]: tripdata1['ride_length'].head
```

```
In [31]: #to get date only
         print(tripdata1['started_at'].dt.date)
```

```
In [32]: #to get time only
         print(tripdata1['started_at'].dt.time)
```

```
In [33]: #to get hour and minute
         print(tripdata1['started_at'].dt.strftime('%H:%M'))
```

Create a column called "day_of_week," and calculate the day of the week that each ride started specify 1 Sunday and 7 = Saturday.

```
In [34]: tripdata1['started_at'].dt.day_name()
```

```
In [76]: tripdata1['day_of_week'] = tripdata1['started_at'].dt.day_name
```

Oh this give us names instead of number let's use another method. we will use the `dayofweek` method.

```
tripdata1['day_of_week'] = tripdata1['started_at'].dt.dayofweek
tripdata1['day_of_week'].unique()
```

We can see that we got 0 as Monday and 6 as Saturday. But we want Monday = 1 and Saturday = 7, let's solve this issue. For a problem there are different type of solution but we will solve it in a simple way let's add 1 to `tripdata1[day_of_week]` column.

```
tripdata1['day_of_week'] = tripdata1['day_of_week'] + 1
tripdata1['day_of_week'].unique()
```

Now let's do for the second tripdataset2 also the same operations.

```
In [37]: tripdata2['ended_at'] = pd.to_datetime(tripdata2['ended_at'])
         tripdata2['started_at'] = pd.to_datetime(tripdata2['started_at'])
         tripdata2['ride_length'] = tripdata2['ended_at'] - tripdata2['started_at']

         tripdata2['day_of_week'] = tripdata2['started_at'].dt.dayofweek + 1
         tripdata2['day_of_week'].unique()
```

So now our data is ready for analysis if we face some issue we can clean it during the analysis.

Analyze

Now that your data is stored appropriately and has been prepared for analysis, start putting it to work. Use the following Case Study Roadmap as a guide:

- How should you organize your data to perform analysis on it?
- Has your data been properly formatted?
- What surprises did you discover in the data?
- What trends or relationships have you found in the data?
- How do these insights answer your business questions or solve the problems?

Key tasks

- Aggregate your data so it's useful and accessible
- Organize and format your data
- Perform calculations
- Identify trends and relationships

Deliverable

- A summary of your analysis

please refer to this if you are using spreadsheet,sql or R programing language for further instruction of analysis. Here.

Key tasks:

- Aggregate your data so it's useful and accessible.

We laready combine the 12 individual data frames into one large data frame for analysis.

Next, we add columns to list the date, month, day, and year of each ride for additional aggregation capabilities.

```
In [38]: tripdata1.head(2)
```

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_station_name
0	A847FADBBC638E45	docked_bike	2020-04-26 17:45:14	2020-04-26 18:12:03	Eckhart Park	86	Lincoln Ave & Diversey Pkw
1	5405B80E996FF60D	docked_bike	2020-04-17 17:08:54	2020-04-17 17:17:03	Drake Ave & Fullerton Ave	503	Kosciuszko Pa

```
In [39]: tripdata1[tripdata1['ride_length']> '40 days 00:00:00']
```

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_station_
17516	FF4F7C2895AF4127	docked_bike	2020-04-19 20:50:53	2020-05-30 15:30:55	Wabash Ave & Wacker Pl	194	HUBBARD S CHECKING WH

let's create another column by name time_diff and represent it in minutes.

```
In [40]: import numpy as np
         tripdata1["time_diff"] = tripdata1["ride_length"]/np.timedelta64(1,'m')
         #Convert to minutes
         tripdata2["time_diff"] = tripdata2["ride_length"]/np.timedelta64(1,'m')
         #Convert to minutes
```

```
In [41]: tripdata1['time_diff'].head(3)
```

```
In [42]: tripdata2['time_diff'].head(3)
```

Next, we add columns to list the date, month, day, and year of each ride for additional aggregation capabilities.

first we will add `year`

```
In [43]: # get a year from corresponding
         # started_at column value
         tripdata1['year'] = pd.DatetimeIndex(tripdata1['started_at']).year
         tripdata1['year'].unique()
```

Let's use another method to do this for the second data type.

```
In [44]: tripdata2['year'] = tripdata2['started_at'].dt.year
```

```
In [45]: tripdata2['year'].unique()
```

Now let's add `month`.

```
In [46]: # get a Month from corresponding
         # started_at column value
         tripdata1['Month'] = pd.DatetimeIndex(tripdata1['started_at']).month
         tripdata1['Month'].unique()
```

from the output we can see that the dataset is of 8 months.

Now let's do for the second datasets.

```
In [47]: # get a Month from corresponding
         # started_at column value
         tripdata2['Month'] = tripdata2['started_at'].dt.month
         tripdata2['Month'].unique()
```

from the ouput we can see this is the second datasets of 4 month data.

Now let's add `days of month`.

```
In [48]: # get a day from corresponding
         # started_at column value
         tripdata1['day'] = pd.DatetimeIndex(tripdata1['started_at']).day
         tripdata1['day'].unique()
```

Now let's do for the second datasets using another method.

```
In [49]: # get a day from corresponding
         # started_at column value
         tripdata2['day'] = tripdata2['started_at'].dt.day
         tripdata2['day'].unique()
```

Now we add extra columns for additional aggregation capabilities. let's check the dataset.

```
In [50]: tripdata1.head()
```

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_station_nam
0	A847FADBBC638E45	docked_bike	2020-04-26 17:45:14	2020-04-26 18:12:03	Eckhart Park	86	Lincoln Ave Diversey Pkw
1	5405B80E996FF60D	docked_bike	2020-04-17 17:08:54	2020-04-17 17:17:03	Drake Ave & Fullerton Ave	503	Kosciuszko Pa
2	5DD24A79A4E006F4	docked_bike	2020-04-01 17:54:13	2020-04-01 18:08:36	McClurg Ct & Erie St	142	Indiana Ave Roosevelt
3	2A59BBDF5CDBA725	docked_bike	2020-04-07 12:50:19	2020-04-07 13:02:31	California Ave & Division St	216	Wood St Augusta Bl
4	27AD306C119C6158	docked_bike	2020-04-18 10:22:59	2020-04-18 11:15:54	Rush St & Hubbard St	125	Sheridan Rd Lawrence A

Now We also remove any unnecessary columns (latitude/longitude fields).

```
In [51]: tripdata1.drop(columns=
         ['start_lat','end_lat','start_lng','end_lng'],inplace=True)
```

```
In [52]: tripdata2.drop(columns=
         ['start_lat','end_lat','start_lng','end_lng'],inplace=True)
```

```
In [53]: tripdata1.head(2)
```

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_station_nam
0	A847FADBBC638E45	docked_bike	2020-04-26 17:45:14	2020-04-26 18:12:03	Eckhart Park	86	Lincoln Ave Diversey Pkw
1	5405B80E996FF60D	docked_bike	2020-04-17 17:08:54	2020-04-17 17:17:03	Drake Ave & Fullerton Ave	503	Kosciuszko Pa

The `time_diff` column is not meaningfull let's rename it to `ride_time`.

```
In [54]: tripdata1.rename(columns={'time_diff': 'ride_time'}, inplace=True)
```

```
In [55]: tripdata2.rename(columns={'time_diff': 'ride_time'}, inplace=True)
```

```
In [56]: tripdata1.head(2)
```

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_station_nam
0	A847FADBBC638E45	docked_bike	2020-04-26 17:45:14	2020-04-26 18:12:03	Eckhart Park	86	Lincoln Ave Diversey Pkw
1	5405B80E996FF60D	docked_bike	2020-04-17 17:08:54	2020-04-17 17:17:03	Drake Ave & Fullerton Ave	503	Kosciuszko Pa

Now it looks nice

Let's next remove the NA rows

Let's check for null values.

```
In [57]: tripdata1.isnull().values.any()
Out[57]: True
```

```
In [58]: tripdata2.isnull().values.any()
Out[58]: False
```

So we don't have any null vlaues because we remove at the begining that why we don't have any null values

```
In [59]: tripdata1[tripdata1['ride_length'] < '0 0 00:00:00']
```

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_stati
2422	7C1E92200AEFF70E	docked_bike	2020-04-27 17:20:30	2020-04-27 17:20:18	St. Clair St & Erie St	211	Cl Washi
2504	671BB1F73F4CD303	docked_bike	2020-04-20 16:51:18	2020-04-20 16:51:05	Lake Shore Dr & Belmont Ave	334	Bu
3136	502B972C6B1FCAE6	docked_bike	2020-04-12 19:09:54	2020-04-12 19:09:40	Clark St & Schiller St	301	C
3542	68E70FDF06F0A439	docked_bike	2020-04-29 17:37:11	2020-04-29 17:37:07	Dearborn St & Adams St	37	Califor
8212	6EB323BCC83A9D1D	docked_bike	2020-04-05 15:46:12	2020-04-05 15:46:11	Francisco Ave & Foster Ave	471	Dan Clyl
...	...	...	...	...	...	...	
2847397	2421864442787859	docked_bike	2020-11-08 16:47:16	2020-11-08 16:47:08	Theater on the Lake	177	Wellii
2847415	90432BCC4C8CBE7B	docked_bike	2020-11-08 12:17:19	2020-11-08 12:16:48	Clark St & Armitage Ave	94	Ashla
2847821	80E99DF91A4790FF	docked_bike	2020-11-22 17:27:58	2020-11-22 17:27:51	Desplaines St & Randolph St	96	C M
2847904	C30F761FFA6954A5	docked_bike	2020-11-10 09:20:05	2020-11-10 09:19:50	Pine Grove Ave & Waveland Ave	232	Stre
2848582	BD4D6E7A0B52E96E	docked_bike	2020-11-14 10:25:00	2020-11-14 10:24:26	Lake Shore Dr & Diversey Pkwy	329	Millen

10082 rows × 15 columns

```
In [60]: tripdata2[tripdata2['ride_length'] < '0 days 00:00:00']
```

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_statio
184	43802DDD3A14F821	docked_bike	2020-12-02 08:14:17	2020-12-02 08:14:12	Albany Ave & Bloomingdale Ave	15655	Sheffie Wellin
322	CDA12952E2FC2D99	docked_bike	2020-12-15 11:39:19	2020-11-25 10:16:59	Damen Ave & Thomas St (Augusta Blvd)	TA1307000070	Morgan S
481	934AEDB1B6C094EC	docked_bike	2020-12-15 11:46:37	2020-11-25 13:16:56	Throop St & Taylor St	13139	Halsted S
832	F9338303AF8415BA	docked_bike	2020-12-15 12:12:41	2020-11-25 13:52:21	Burnham Harbor	15545	Indian Roos
890	7787906127D4A8FE	electric_bike	2020-12-15 12:00:46	2020-11-25 16:54:15	Albany Ave & Bloomingdale Ave	15655	Dame Co
...	...	...	...	...	...	...	
112715	7074FF42B83EC6AD	docked_bike	2020-12-15 11:50:14	2020-11-25 13:42:44	Peoria St & Jackson Blvd	13158	Clin Ma
113268	7BD7B9F52607DAB3	electric_bike	2020-12-15 11:53:44	2020-11-25 14:50:00	Pine Grove Ave & Irving Park Rd	TA1308000022	Broadway
113705	70F458C5AAE4C49F	docked_bike	2020-12-15 12:02:17	2020-11-25 18:03:18	Pine Grove Ave & Irving Park Rd	TA1308000022	Pine Grov Irving
279727	F79335E3A77A57B5	electric_bike	2021-03-29 15:41:21	2021-03-29 15:41:20	Ashland Ave & Belle Plaine Ave	13249	Ashlan Belle Pl
353457	5D2797A8FFA71B49	classic_bike	2021-03-13 18:02:58	2021-03-13 18:02:57	Dayton St & North Ave	13058	Dayton St

372 rows × 15 columns

Since we don't have negative time we will drop all the rows which ride_length is negative.

With the help of the following code we can drop rows based on conditional expression.

```
df = df.drop(some labels)
df = df.drop(df[].index)
```

With the help of this let's drop all the rows whihc ride_lenght is less than 0.

```
In [61]: tripdata1.drop(tripdata1[tripdata1.ride_length < "0 days 00:00:00"].index,
         inplace=True)
         tripdata2.drop(tripdata2[tripdata2.ride_length < "0 days 00:00:00"].index,
         inplace=True)
```

Zero length also dosent' make any sense but we will keep it for now.

```
In [62]: tripdata1[tripdata1['ride_length'] < '0 days 00:00:00']
```

ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_station_name	end_statio

We can see that we successfully drop all the rows with the negative ride_lenght.

```
In [63]: tripdata1.head(2)
```

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_station_nam
0	A847FADBBC638E45	docked_bike	2020-04-26 17:45:14	2020-04-26 18:12:03	Eckhart Park	86	Lincoln Ave Diversey Pkw
1	5405B80E996FF60D	docked_bike	2020-04-17 17:08:54	2020-04-17 17:17:03	Drake Ave & Fullerton Ave	503	Kosciuszko Pa

It seems like our data is now free of null values and it's look clean but if we face some problem we will resolve it.

Perform calculation

From now on we will look at only first 8 months data we will ignore the 4 month data.

Let's first look at the structure of the columns.

```
In [64]: tripdata1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2838601 entries, 0 to 2848682
Data columns (total 15 columns):
 #   Column              Dtype
---  ------              -----
 0   ride_id             object
 1   rideable_type       object
 2   started_at          datetime64[ns]
 3   ended_at            datetime64[ns]
 4   start_station_name  object
 5   start_station_id    int64
 6   end_station_name    object
 7   end_station_id      float64
 8   member_casual       object
 9   ride_length         timedelta64[ns]
 10  day_of_week         int64
 11  ride_time           float64
 12  year                int64
 13  Month               int64
 14  day                 int64
dtypes: datetime64[ns](2), float64(2), int64(5), object(5), timedelta64[ns](1)
memory usage: 346.5+ MB
```

Now let's look at a statistical summary of the aggregated and transformed data frame.

```
In [65]: tripdata1.describe(include='all')
```

```
<ipython-input-65-6139dd9a4696>:1: FutureWarning: Treating datetime data as categorica
l rather than numeric in `.describe` is deprecated and will be removed in a future ver
sion of pandas. Specify `datetime_is_numeric=True` to silence this warning and adopt t
he future behavior now.
  tripdata1.describe(include='all')
<ipython-input-65-6139dd9a4696>:1: FutureWarning: Treating datetime data as categorica
l rather than numeric in `.describe` is deprecated and will be removed in a future ver
sion of pandas. Specify `datetime_is_numeric=True` to silence this warning and adopt t
he future behavior now.
  tripdata1.describe(include='all')
```

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id
count	2838601	2838601	2838601	2838601	2838601	2.838601e+06
unique	2838601	2	2454431	2440090	685	NaN
top	07B025CD8E5F9ABE	docked_bike	2020-08-25 17:18:20	2020-07-11 15:35:03	Streeter Dr & Grand Ave	NaN
freq	1	2512332	8	13	32069	NaN
first	NaN	NaN	2020-04-01 00:00:30	2020-04-01 00:10:45	NaN	NaN
last	NaN	NaN	2020-11-30 23:56:22	2020-12-01 17:28:22	NaN	NaN
mean	NaN	NaN	NaN	NaN	NaN	2.247137e+02
std	NaN	NaN	NaN	NaN	NaN	1.603175e+02
min	NaN	NaN	NaN	NaN	NaN	2.000000e+00
25%	NaN	NaN	NaN	NaN	NaN	9.600000e+01
50%	NaN	NaN	NaN	NaN	NaN	1.960000e+02
75%	NaN	NaN	NaN	NaN	NaN	3.120000e+02
max	NaN	NaN	NaN	NaN	NaN	7.320000e+02

Let's next focus on the average time of each ride.

```
In [67]: tripdata1['ride_time'].describe()
```

```
In [69]: tripdata1['ride_time'].mean()
```

We see that on average, each ride is close to 30 minutes. We'll then break that down by casual riders versus members.

Looking at casual riders versus members, we can see that the average casual ride is about 46 minutes compared to the members' average ride of 16 minutes.

```
In [71]: tripdata1.groupby('member_casual').mean()['ride_time']
```

The median rides are 22 minutes and 12 minutes respectively.

```
In [72]: tripdata1.groupby('member_casual').median()['ride_time']
```

What is the most common day for renting bikes?

```
In [83]: tripdata1.groupby('member_casual')['day_name'].agg(pd.Series.mode)
```

Using the statistical mode, we see that the most common day for renting bikes is Saturday.

If this is needed as a DataFrame, use

```
In [89]: tripdata1.groupby('member_casual')
         ['day_name'].agg(pd.Series.mode).to_frame()
```

	day_name
member_casual	
casual	Saturday
member	Saturday

```
In [123]: average_day_ride = pd.DataFrame(tripdata1.groupby('day_of_week').mean()
          ['ride_time'])
          x = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
          "Saturday"]
          average_day_ride['Day_name'] = x
          average_day_ride.rename(columns={'ride_time': 'average_ride_time'},
          inplace=True)
          average_day_ride
```

	average_ride_time	Day_name
day_of_week		
1	27.324941	Sunday
2	24.985572	Monday
3	25.280675	Tuesday
4	26.685197	Wednesday
5	28.139172	Thursday
6	33.943831	Friday
7	36.048899	Saturday

```
In [90]: tripdata1.groupby('member_casual').mean()['day_of_week']
```

Also, we can take a look at the average ride time by day for members and casual riders with duration again in minutes. Regardless of day of the week, casual users ride longer than members, with both groups riding longer on weekends.

```
In [210]: #tripdata1.groupby('day_of_week','member_casual').mean()
          grop_by_member_and_day = tripdata1.groupby(['day_name',
          'member_casual']).agg({'ride_time':['mean']})
          grop_by_member_and_day
```

		ride_time
		mean
day_name	**member_casual**	
Friday	casual	44.195253
	member	16.137315
Monday	casual	46.420619
	member	15.603183
Saturday	casual	47.930144
	member	18.276854
Sunday	casual	52.308475
	member	18.755118
Thursday	casual	45.007104
	member	15.572239
Tuesday	casual	41.939300
	member	15.524317
Wednesday	casual	42.340339
	member	15.593903

```
In [211]: index = grop_by_member_and_day.index
          day = []
          mem = []
          for i in index:
              day.append(i[0])
              mem.append(i[1])
```

```
In [229]: df0 = grop_by_member_and_day['ride_time']
          df = df0['mean']
```

```
df.reset_index(drop=True, inplace=True)
df1 = pd.DataFrame(df)
df1['weekdays'] = day
df1['member_casual'] = mem
df1.rename(columns={'mean': 'average_ride_time'}, inplace=True)
df1
```

	average_ride_time	weekdays	member_casual
0	44.195253	Friday	casual
1	16.137315	Friday	member
2	46.420619	Monday	casual
3	15.603183	Monday	member
4	47.930144	Saturday	casual
5	18.276854	Saturday	member
6	52.308475	Sunday	casual
7	18.755118	Sunday	member
8	45.007104	Thursday	casual
9	15.572239	Thursday	member
10	41.939300	Tuesday	casual
11	15.524317	Tuesday	member
12	42.340339	Wednesday	casual
13	15.593903	Wednesday	member

```
In [238]: df1 = grop_by_member_and_day ... ('ride_time', 'day', columnsember_casual')
          df1.reset_index(drop=True, inplace=True)
```

```
df1.columns
```

```
In [184]: df2 = tripdata1.groupby(['day_name','member_casual']).size().to_frame()
          df2.reset_index(drop=True, inplace=True)
          df2[...] = df2[0]
          df2['day'] = day
          df2['member_casual'] = mem
          del df2[0]
          df2
```

Out[184…

	number_of_ride	day	member_casual
0	183369	Friday	casual
1	245311	Friday	member
2	126329	Monday	casual
3	205800	Monday	member
4	288345	Saturday	casual
5	257413	Saturday	member
6	225945	Sunday	casual
7	212433	Sunday	member
8	144841	Thursday	casual
9	238799	Thursday	member
10	121448	Tuesday	casual
11	217624	Tuesday	member
12	134345	Wednesday	casual
13	236599	Wednesday	member

The last column represent the number of ride perday as casual and memebers.

Identify trends and relationships.

Here are some key observations using the simple analyses above:

- Casual riders average longer rides than members. This makes sense as members are likely using the bike rental service for particular commutes (i.e. work or school) whereas casual riders are just that--casual (perhaps they are sightseeing).
- The number of rides for both types of users starts off slow on Mondays, gradually increases to a peak on Saturdays with a small drop off on Sundays.

Share

Now that you have performed your analysis and gained some insights into your data, create visualizations to share your findings. Moreno has reminded you that they should be sophisticated and polished in order to effectively communicate to the executive team. Use the following Case Study Roadmap as a guide:

Guiding questions

- Were you able to answer the question of how annual members and casual riders use Cyclistic bikes differently?
- What story does your data tell?
- How do your findings relate to your original question?
- Who is your audience? What is the best way to communicate with them?
- Can data visualization help you share your findings?
- Is your presentation accessible to your audience?

Key tasks

- Determine the best way to share your findings.
- Create effective data visualizations.
- Create effective data visualizations
- Present your findings
- Ensure your work is accessible to your audience

Deliverable

- Supporting visualizations and key findings

Follow these steps:

- Take out a piece of paper and a pen and sketch some ideas for how you will visualize the data.
- Once you choose a visual form, open your tool of choice to create your visualization. Use a presentation software, such as PowerPoint or Google Slides; your spreadsheet program; Tableau; or R.
- Create your data visualization, remembering that contrast should be used to draw your audience's attention to the most important insights. Use artistic principles including size, color, and shape.
- Ensure clear meaning through the proper use of common elements, such as headlines, subtitles, and labels.
- Refine your data visualization by applying deep attention to detail.

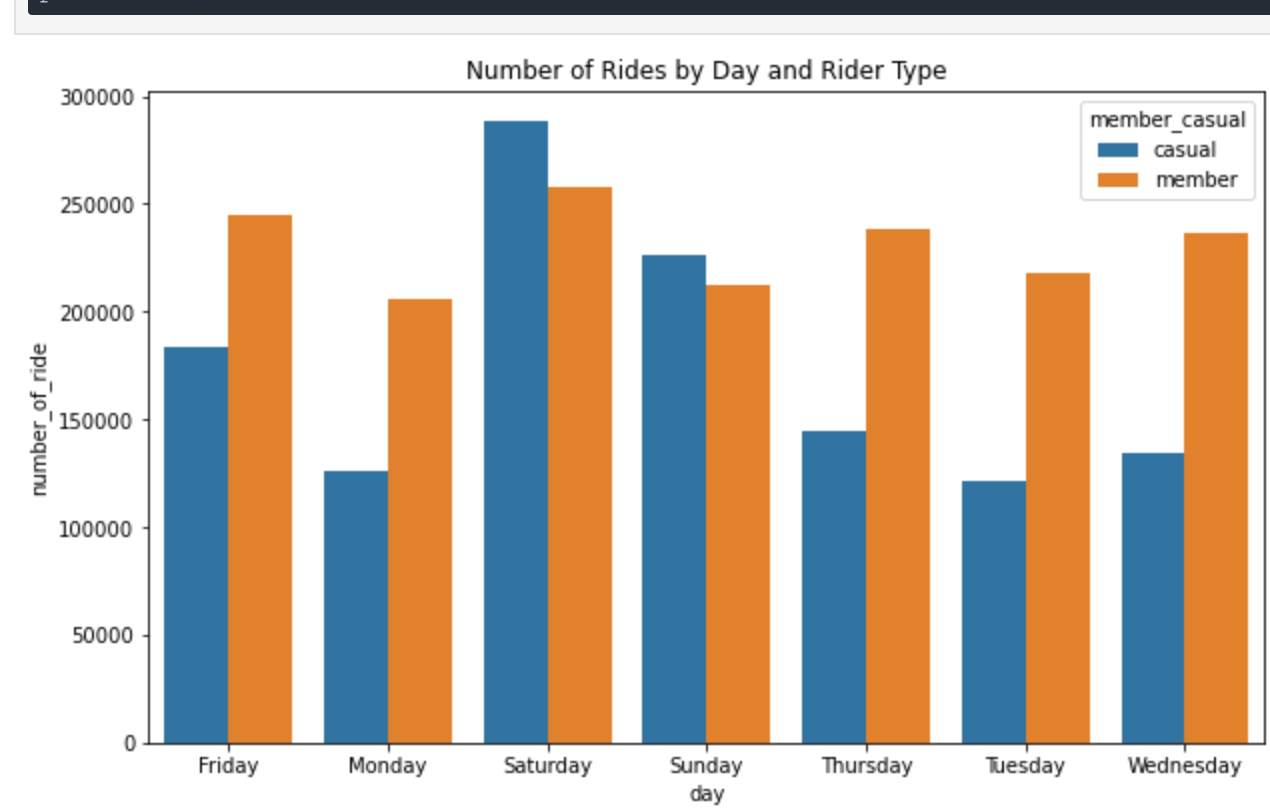
Key tasks

- determine the best way to share your findings.

With the help of df1 and df2 let's do some visualization.

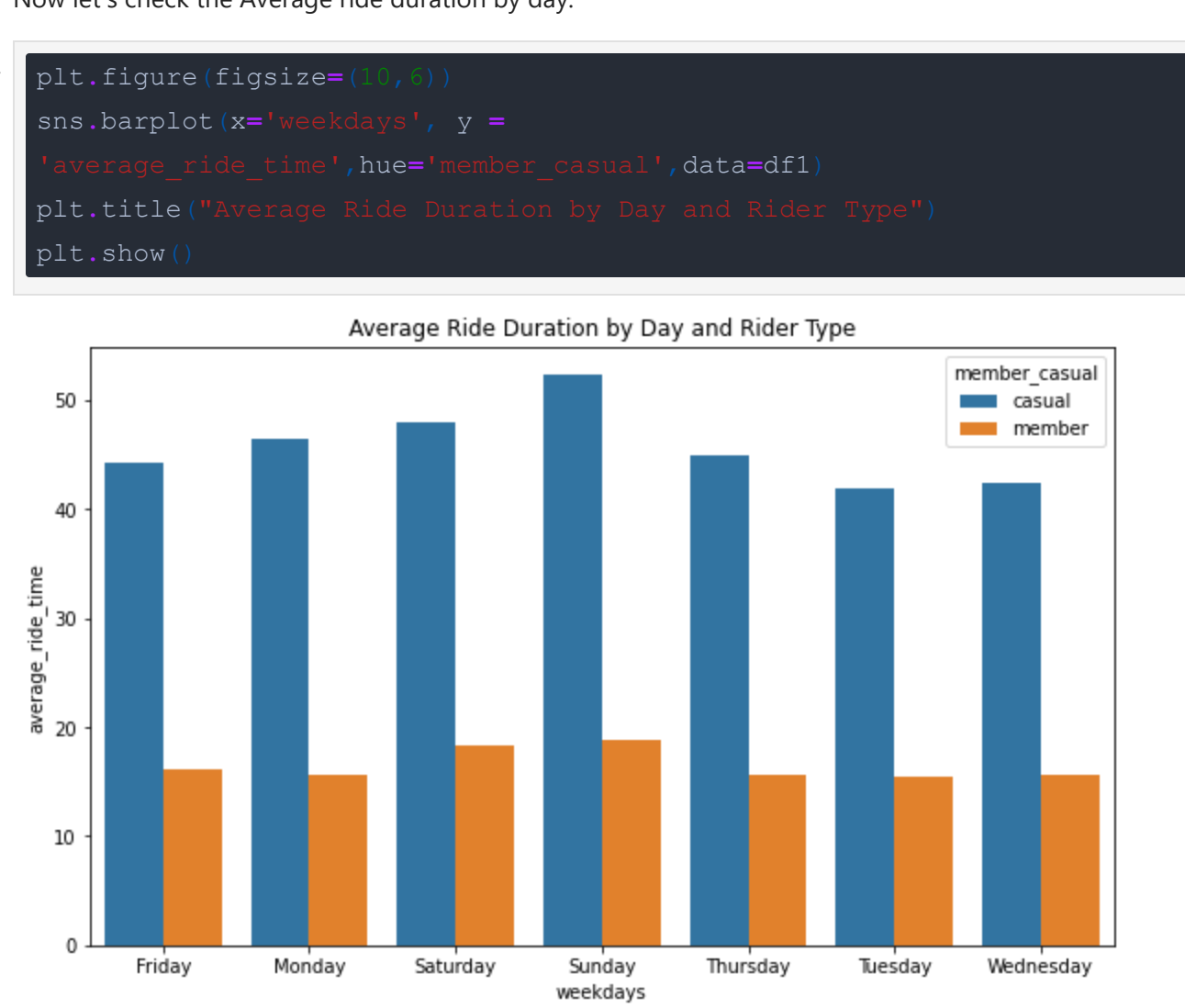
In [185…
```
import seaborn as sns      # visualization library for statistical graphics plotting
import matplotlib.pyplot as plt    # import matplotlib.pyplot as plt
%matplotlib inline
```

In [197…
```
plt.figure(figsize=(10,6))
sns.barplot(x='day' , y = 'number_of_ride',hue='member_casual',data=df2)
plt.title("Number of Rides by Day and Rider Type")
plt.show()
```



Now let's check the Average ride duration by day.

In [233…
```
plt.figure(figsize=(10,6))
sns.barplot(x='weekdays', y =
'average_ride_time',hue='member_casual',data=df1)
plt.title("Average Ride Duration by Day and Rider Type")
plt.show()
```



Let's plot Number of rides by bike type and rider type

In [243…
```
df = tripdata1.groupby(['rideable_type', 'member_casual']).agg('count')
df
```

Out[243…

rideable_type	member_casual	ride_id	started_at	ended_at	start_station_name	start_station_id	end_station_n…
docked_bike	casual	1087520	1087520	1087520	1087520	1087520	108…
	member	1424812	1424812	1424812	1424812	1424812	142…
electric_bike	casual	137102	137102	137102	137102	137102	13…
	member	189167	189167	189167	189167	189167	18…

In [260…
```
x = tripdata1.groupby(['rideable_type',
'member_casual']).size().to_frame()
x.index
```

Out[260…
```
MultiIndex([( 'docked_bike', 'casual'),
            ( 'docked_bike', 'member'),
            ('electric_bike', 'casual'),
            ('electric_bike', 'member')],
           names=['rideable_type', 'member_casual'])
```

In [261…
```
index = x.index
bike_type = []
member_casual = []

for i in index:
    bike_type.append(i[0])
    member_casual.append(i[1])
```

In [264…
```
```

Out[264…
```
['docked_bike', 'docked_bike', 'electric_bike', 'electric_bike']
```

In [275…
```
x.reset_index(drop=True,inplace=True)
bike_df = x
bike_df['bike_type'] = bike_type
bike_df['member_casual'] = member_casual
bike_df['number_of_ride'] = bike_df[0]
bike_df.drop(columns=0)
```

Out[275…

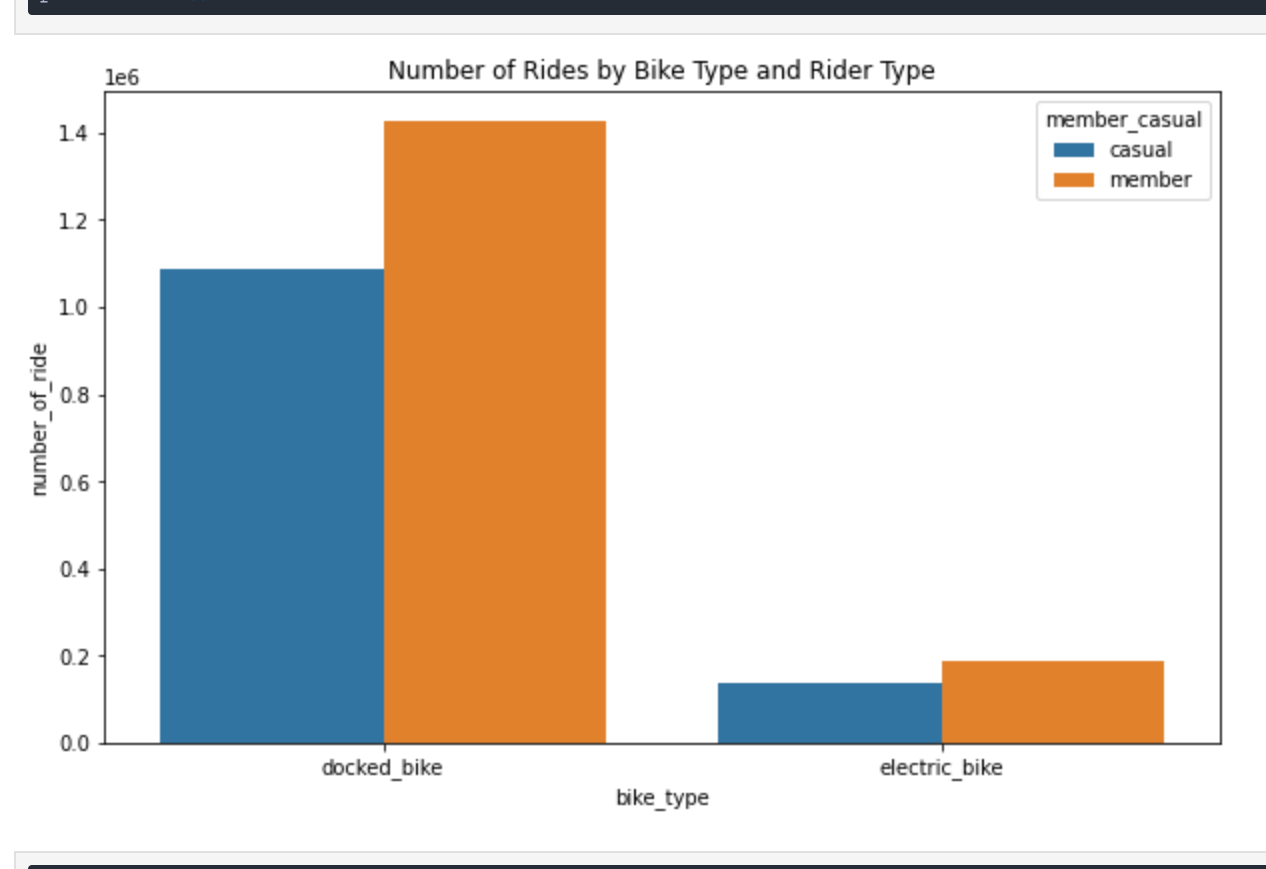
	bike_type	member_casual	number_of_ride
0	docked_bike	casual	1087520
1	docked_bike	member	1424812
2	electric_bike	casual	137102
3	electric_bike	member	189167

In [276…
```
bike_df
```

Out[276…

	0	bike_type	member_casual	number_of_ride
0	1087520	docked_bike	casual	1087520
1	1424812	docked_bike	member	1424812
2	137102	electric_bike	casual	137102
3	189167	electric_bike	member	189167

In [278…
```
plt.figure(figsize=(10,6))
sns.barplot(x='bike_type', y =
'number_of_ride',hue='member_casual',data=bike_df)
plt.title("Number of Rides by Bike Type and Rider Type")
plt.show()
```



In [ ]:
```
```

Now let's plot Number of rides by day and bike type.

First we need to prepare our data.

In [280…
```
tripdata1.groupby(['rideable_type', 'day_name']).size().to_frame()
```

Out[280…

rideable_type	day_name	0
docked_bike	Friday	376188
	Monday	292734
	Saturday	489948
	Sunday	394651
	Thursday	334877
	Tuesday	298500
	Wednesday	325434
electric_bike	Friday	52492
	Monday	39395
	Saturday	55810
	Sunday	43727
	Thursday	48763
	Tuesday	40572
	Wednesday	45510

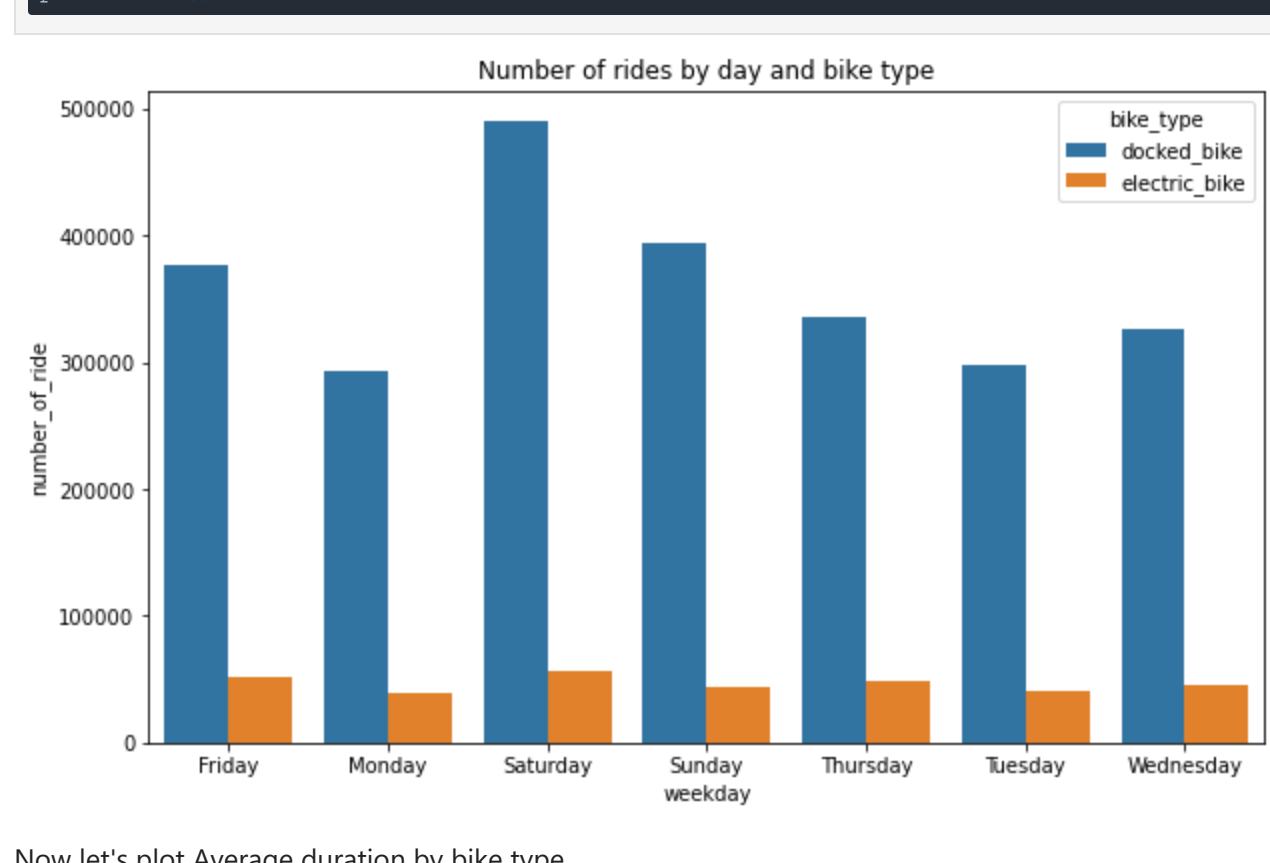
In [283…
```
x = tripdata1.groupby(['rideable_type', 'day_name']).size().to_frame()
index = x.index
bike_type = []
weekdays = []

for i in index:
    bike_type.append(i[0])
    weekdays.append(i[1])

x.reset_index(drop=True,inplace=True)
bike_type_df = x
bike_type_df['bike_type'] = bike_type
bike_type_df['weekday'] = weekdays
bike_type_df['number_of_ride'] = bike_type_df[0]
bike_type_df.drop(columns=0)
```

Out[283…

	bike_type	weekday	number_of_ride
0	docked_bike	Friday	376188
1	docked_bike	Monday	292734
2	docked_bike	Saturday	489948
3	docked_bike	Sunday	394651
4	docked_bike	Thursday	334877
5	docked_bike	Tuesday	298500
6	docked_bike	Wednesday	325434
7	electric_bike	Friday	52492
8	electric_bike	Monday	39395
9	electric_bike	Saturday	55810
10	electric_bike	Sunday	43727
11	electric_bike	Thursday	48763
12	electric_bike	Tuesday	40572
13	electric_bike	Wednesday	45510

Now our data is ready let's plot the graph.

In [287…
```
plt.figure(figsize=(10,6))
sns.barplot(x='weekday', y =
'number_of_ride',hue='bike_type',data=bike_type_df)
plt.title("Number of rides by day and bike type")
plt.show()
```



Now let's plot Average duration by bike type.

In [299…
```
x = tripdata1.groupby(['day_name', 'rideable_type']).agg({'ride_time':
['mean']})
index = x.index
bike_type = []
weekdays = []

for i in index:
    bike_type.append(i[0])
    weekdays.append(i[1])

y = x['ride_time']
y.reset_index(drop=True,inplace=True)

bike_type_df = y
bike_type_df['bike_type'] = bike_type
bike_type_df['weekday'] = weekdays
```

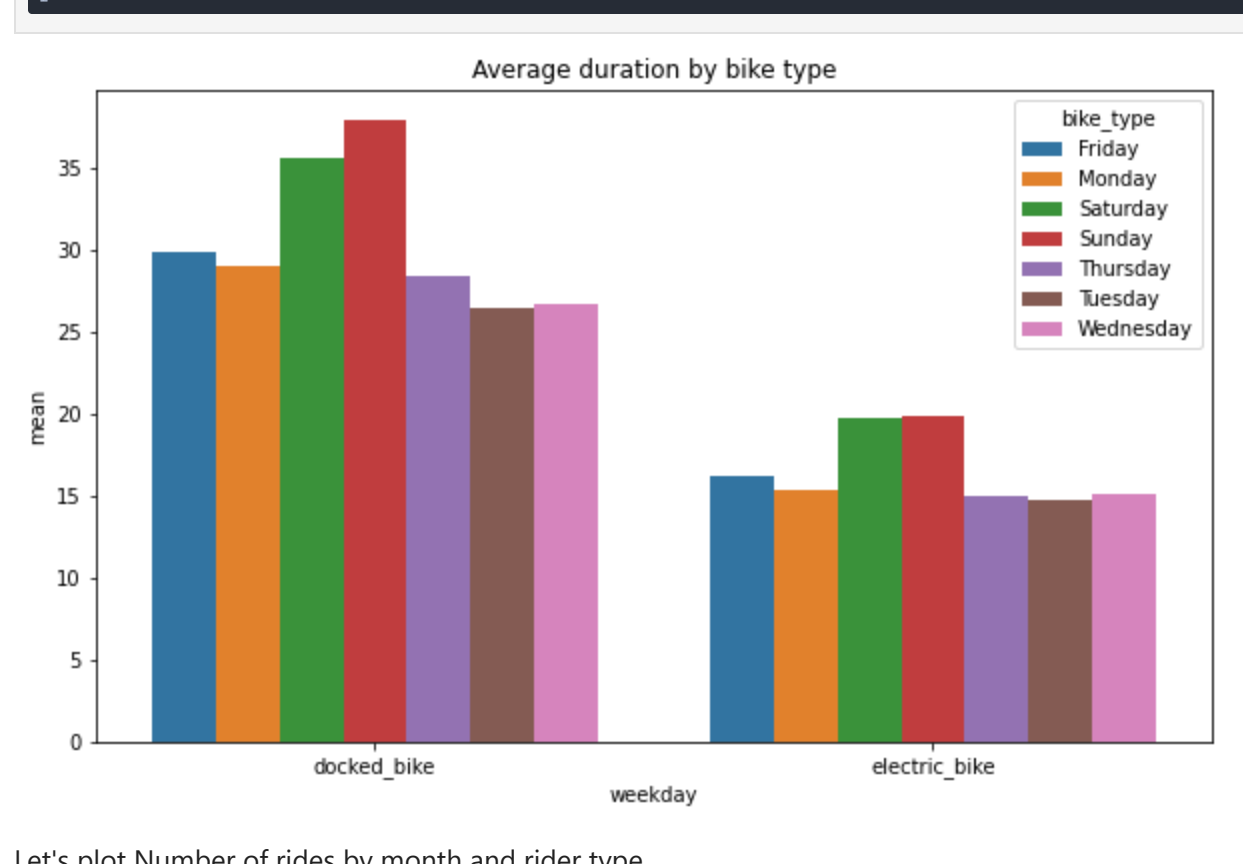
```
<ipython-input-299-8ice5cc38e97>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  bike_type_df['bike_type'] = bike_type
<ipython-input-299-8ice5cc38e97>:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  bike_type_df['weekday'] = weekdays
```

In [300…
```
bike_type_df
```

Out[300…

	mean	bike_type	weekday
0	29.814207	Friday	docked_bike
1	16.134905	Friday	electric_bike
2	28.931834	Monday	docked_bike
3	15.384537	Monday	electric_bike
4	35.560684	Saturday	docked_bike
5	19.749711	Saturday	electric_bike
6	37.847801	Sunday	docked_bike
7	19.813203	Sunday	electric_bike
8	28.384532	Thursday	docked_bike
9	15.015175	Thursday	electric_bike
10	26.378330	Tuesday	docked_bike
11	14.738653	Tuesday	electric_bike
12	26.699327	Wednesday	docked_bike
13	15.136145	Wednesday	electric_bike

In [302…
```
plt.figure(figsize=(10,6))
sns.barplot(x='weekday', y = 'mean',hue='bike_type',data=bike_type_df)
plt.title("Average duration by bike type")
plt.show()
```



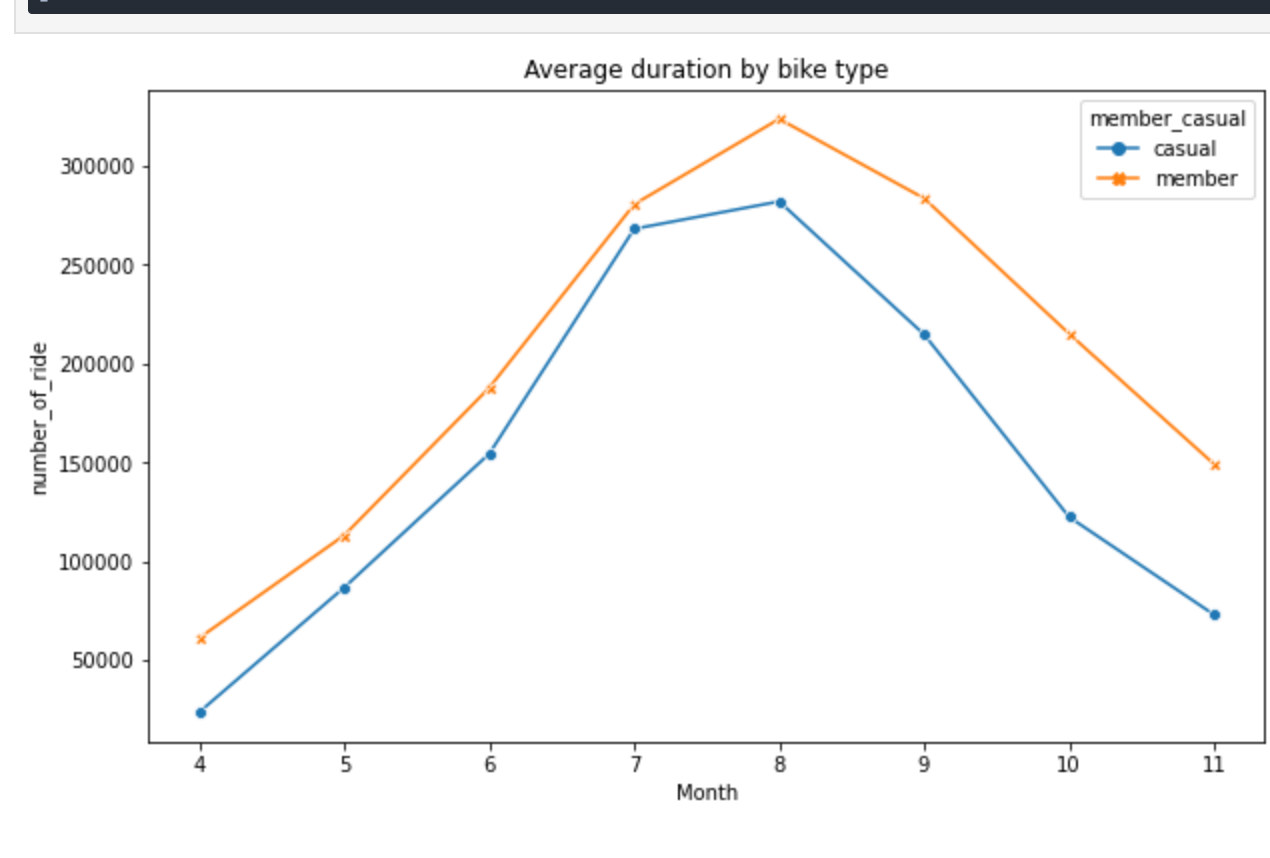
Let's plot Number of rides by month and rider type.

In [324…
```
x = tripdata1.groupby(['member_casual', 'Month']).size().to_frame()
index = x.index
member_casual = []
month = []

for i in index:
    member_casual.append(i[0])
    month.append(i[1])

x.reset_index(drop=True,inplace=True)
bike_type_df = x
bike_type_df['member_casual'] = member_casual
bike_type_df['Month'] = month
bike_type_df['number_of_ride'] = bike_type_df[0]
bike_type_df.drop(columns=0)
```

Out[324…

	member_casual	Month	number_of_ride
0	casual	4	23610
1	casual	5	86699
2	casual	6	154342
3	casual	7	268125
4	casual	8	281987
5	casual	9	214681
6	casual	10	122328
7	casual	11	72850
8	member	4	61115
9	member	5	113083
10	member	6	187727
11	member	7	280556
12	member	8	323759
13	member	9	283582
14	member	10	215072
15	member	11	149085

In [328…
```
plt.figure(figsize=(10,6))
sns.lineplot(x='Month', y =
'number_of_ride',hue='member_casual',data=bike_type_df,style="member_casual",
 dashes=False)
plt.title("Average duration by bike type")
plt.show()
```



Present your findings.

Here is a summary of the key observations from above:

- On average, each ride is about 30 minutes:
  - Casual users ride for 46 minutes on average.
  - Members ride for 16 minutes on average.
- Regardless of being a member or not, the most popular day to rent a bike is Saturday.
- Bike rentals start off at a low on Mondays, peak on Saturdays with a slight drop off on Sundays.
- Members rent bikes on a more consistent basis throughout the entire week, whereas casual rentals are low Monday through Thursday and peak towards the weekend
- On any day of the week, casual users ride longer than members
- The docked bike option is far more popular than both classic bikes and electric bikes, both in terms of number of rentals and average ride duration on each type of bike.
- Bike rentals follow a seasonal pattern for both types of users
- Since Chicago experiences inclement weather, lowest usage is in the winter with rentals starting to ramp up in the spring. Peak usage is in the summer (August) before it starts to decline again during the Fall.

Act

Now that you have finished creating your visualizations, act on your findings. Prepare the deliverables Morena asked you to create, including the three top recommendations based on your analysis. Use the following Case Study Roadmap as a guide:

- What is your final conclusion based on your analysis?
- How could your team and business apply your insights?
- What next steps would you or your stakeholders take based on your findings?
- Is there additional data you could use to expand on your findings?
- How can you apply your insights?

Key tasks

- Practice presenting your case study to a friend or family member.
- Share next steps with your stakeholders
- Determine if more data could give you new insights

Deliverable

- Your top three recommendations based on your analysis

First, let's revisit the key business task:

`discover how casual riders and Cyclistic members use their rental bikes differently?`

It appears that members rent bikes for regular commuting---perhaps to work or school. This is based on the observation that the number of rides is fairly consistent day over day as is the average ride duration.

Conversely, casual user rentals are lower Monday through Thursday and trend upwards starting on Friday and peaking on the weekend.

This seems to indicate using the bikes for leisure activities such as touring the city, sightseeing, etc. Additional data that supports this is the longer average ride duration.

My three recommendations for the new marketing strategy are as follows:

1. Offer a weekend-only membership at a different price point than the full annual membership to entice casual users towards a full annual membership. They can only unlock bikes on Friday, Saturday, or Sunday.
2. Create a "See our City" campaign targeted to casual users that includes 52 suggested routes that will cover all of the major sights in Chicago---one for each weekend. Casual riders could rent as they go, but they could see their city in one year (and save money!) for the price of an annual membership.
3. Be sure that any campaigns are out to the market in the summer when ridership is at its annual peak.

`Note:` it needs to be improve because my analysis is not completed yet.

reference for some help!: https://www.kaggle.com/codyfreeman/code

In [ ]:
```
```