Bike-Share Case Study

This report provides the results and step-by-step explanation of the data analysis performed for a bike-sharing case study. The data belongs to a bike-sharing company that has two kinds of users: annual members and casual riders. The goal of the study was to identify how annual members and casual riders use the bikes differently in order to help the stake-holders decide whether or not to target converting casual riders into annual members in the next marketing campaign. Python was used to perform the analysis using data collected from January-November 2023 and downloaded from https://divvy-tripdata.s3.amazonaws.com/index.html.

Cleaning:

The code that was used to perform the data exploration can be found in the Jupyter Notebook cleaning.ipynb. The main functions are explained below:

- *read_data*:
  Here the .csv file for the bike rides of each month are read and stored into a dictionary. Each element in the dictionary has a key (the name of the month) and a value (the Panada data-frame that holds the data entries). This simplifies the access of the entries for each corresponding month, by using the month as the key (e.g. original_data[“May”] retrieves the data-frame that holds the entries from May), while maintaining the segregation between months (data from different months are not fused together). In Figure 1 we can see the first and last 5 entries of bike rides from May:

	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
0	0D9FA920C3062031	electric_bike	2023-05-07 19:53:48	2023-05-07 19:58:32	Southport Ave & Belmont Ave	13229	NaN	NaN	41.939408	-87.663831	41.930000	-87.650000	member
1	92485E5FB5888ACD	electric_bike	2023-05-06 18:54:08	2023-05-06 19:03:35	Southport Ave & Belmont Ave	13229	NaN	NaN	41.939482	-87.663848	41.940000	-87.690000	member
2	FB144B3FC8300187	electric_bike	2023-05-21 00:40:21	2023-05-21 00:44:36	Halsted St & 21st St	13162	NaN	NaN	41.853793	-87.646719	41.860000	-87.650000	member
3	DDEB93BC2CE9AA77	classic_bike	2023-05-10 16:47:01	2023-05-10 16:59:52	Carpenter St & Huron St	13196	Damen Ave & Cortland St	13133	41.894556	-87.653449	41.915983	-87.677335	member
4	C07B70172FC92F59	classic_bike	2023-05-09 18:30:34	2023-05-09 18:39:28	Southport Ave & Clark St	TA1308000047	Southport Ave & Belmont Ave	13229	41.957081	-87.664199	41.939478	-87.663748	member
...	...	...	...	...	...	...	...	...	...	...	...	...	...
604822	48BDA26F34445546	electric_bike	2023-05-18 10:26:43	2023-05-18 10:48:00	Clark St & Elmdale Ave	KA1504000148	NaN	NaN	41.990876	-87.669721	42.000000	-87.660000	member
604823	573025E5EDE10DE1	electric_bike	2023-05-17 14:32:48	2023-05-17 14:45:37	State St & 33rd St	13216	NaN	NaN	41.834734	-87.625798	41.830000	-87.620000	member
604824	D88D48898C6FB63E	electric_bike	2023-05-17 07:59:29	2023-05-17 08:04:54	Columbus Dr & Randolph St	13263	NaN	NaN	41.884422	-87.619393	41.880000	-87.630000	member
604825	4692DCD2F87497F5	electric_bike	2023-05-18 08:34:48	2023-05-18 08:38:40	Public Rack - Karlov Ave & Lawrence Ave	1127.0	NaN	NaN	41.970000	-87.730000	41.970000	-87.740000	member
604826	6ACB7E383473D019	electric_bike	2023-05-29 21:16:58	2023-05-29 21:24:35	State St & 33rd St	13216	NaN	NaN	41.834715	-87.625764	41.840000	-87.650000	member

Figure 1: First and last 5 entries of bike rides from May (original_data[“May”])

From the entries we can see that the data consists of 13 columns: 1) ride id, 2) type of bike, 3-4) date and time for the start and end of the ride, 5-12) the name, id, latitude and longitude of the start and end stations, and 13) whether the rider was a casual rider or a member.

- *count_entries*:
After the csv files are read into the data-frames, the method *count_entries* is used to collect further information about the dataset. It finds the number of entries per file as well as the number of columns. This is done in order to check that the data format is consistent across the different files. Next, the method calculates the total number of bike rides in the entire dataset. There is an option within the method to remove duplicates. Therefore, the method is first called with the remove duplicates option deactivated, in order to collect preliminary information about the dataset. And then the method is called again with the remove duplicates option activated. The results are shown in Figure 2.

BikeRides_Original

Month	No Of Entries	No Of Cols
January	190301	13
February	190445	13
March	258678	13
April	426590	13
May	604827	13
June	719618	13
July	767650	13
August	771693	13
September	666371	13
October	537113	13
November	362518	13
Total:	5495804	
Average:	499618	

BikeRides_without_Duplicates

Month	No Of Entries	No Of Cols
January	190301	13
February	190445	13
March	258678	13
April	426590	13
May	604827	13
June	719618	13
July	767650	13
August	771693	13
September	666371	13
October	537113	13
November	362518	13
Total:	5495804	
Average:	499618	

Figure 2: No of entries and columns before and after removing duplicates

The table in Figure 2 on the left is the result of running the method without removing duplicates, and the table on the right is the result after removing duplicates. We can see that all the files have the same number of columns, which is a good preliminary indicator of the consistency of data across the months. In total the dataset contains almost 5.5 Million entries, with an average of approximately 500,000 entries per month. From January to March the number of rides is relatively lower than the average, which is expected as these are cold months. This is confirmed by the peak in the number of rides during the Summer months June to August. The number of entries before and after removing duplicates is identical, therefore the original dataset did not have any duplicates.

- *check_NAN*:
Given that a brief look at the entries from May already showed a couple of NaN values

(Figure 1 *end_station_name*, and *end_station_id*), the method *check_NAN* calculates the percentage of NaN values for each column and month. The results are shown in Figure 3. As we can see the columns *start_station_name*, *start_station_id*, *end_station_name*, *end_station_id* in every month have 13-17% null values. The columns *end_lat* and *end_long* have less than 1% null values.

NaN_Percentages

Month	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
January	0	0	0	0	14 %	14 %	14 %	14 %	0	0	< 1%	< 1%	0
February	0	0	0	0	13 %	13 %	14 %	14 %	0	0	< 1%	< 1%	0
March	0	0	0	0	13 %	13 %	14 %	14 %	0	0	< 1%	< 1%	0
April	0	0	0	0	14 %	14 %	16 %	16 %	0	0	< 1%	< 1%	0
May	0	0	0	0	14 %	14 %	15 %	15 %	0	0	< 1%	< 1%	0
June	0	0	0	0	16 %	16 %	17 %	17 %	0	0	< 1%	< 1%	0
July	0	0	0	0	16 %	16 %	16 %	16 %	0	0	< 1%	< 1%	0
August	0	0	0	0	15 %	15 %	16 %	16 %	0	0	< 1%	< 1%	0
September	0	0	0	0	15 %	15 %	16 %	16 %	0	0	< 1%	< 1%	0
October	0	0	0	0	15 %	15 %	16 %	16 %	0	0	< 1%	< 1%	0
November	0	0	0	0	15 %	15 %	15 %	15 %	0	0	< 1%	< 1%	0

Figure 3: Percentage of null values for each column across the various months

In order to explore the dataset a bit further, the number of unique (distinct) values for each column in original_data[“May”] is calculated and shown in Figure 4.

```
Column Name:          NUnique
ride_id               604827
rideable_type              3
started_at            503683
ended_at              505259
start_station_name      1287
start_station_id        1250
end_station_name        1254
end_station_id          1210
start_lat             188591
start_lng             185410
end_lat                 4759
end_lng                 4762
member_casual              2
```

Figure 4: No. of unique values for every column in original_data[“May”]

- *ride_id_unique* = 604827: as expected, has as many unique values as the number of entries in the dataframe.
- *rideable_type_unique* = 3: these 3 unique values are: [electric_bike, classic_bike, docked_bike].
- *started(ended)_at_unique* = 503683, 505259: since these are datetimes (yy-mm-dd hh:mm:ss), one may have expected that they would have as many distinct values as the number of entries, since it seems unlikely for more than one rider to have

rented a bike at the exact same time down to the second. However, the number of unique values in these columns is less than the number of entries by 17%. To ensure that these are not duplicate entries but with different *ride_id*, one of these incidents has been retrieved and is shown in Figure 9. By looking at the values, it is clear that they are indeed different entries but with the exact same start time.

	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
727	041763A703C94783	electric_bike	2023-05-28 14:59:58	2023-05-28 15:12:47	Kedzie Ave & Milwaukee Ave	13085	Kilpatrick Ave & Parker Ave	358	41.929673	-87.708045	41.930731	-87.744106	casual
2710	5AEC034DB275854E	electric_bike	2023-05-28 14:59:58	2023-05-28 15:26:56	Broadway & Belmont Ave	13277	NaN	NaN	41.940170	-87.645626	41.960000	-87.640000	casual
400665	A99D22D37DC92962	electric_bike	2023-05-28 14:59:58	2023-05-28 15:09:35	NaN	NaN	MTV Hubbard St	021320	41.880000	-87.660000	41.889779	-87.680341	member
557920	79A55702C0B6D246	docked_bike	2023-05-28 14:59:58	2023-05-28 16:24:12	Streeter Dr & Grand Ave	13022	Field Museum	13029	41.892278	-87.612043	41.865312	-87.617867	casual

Figure 5: Entries from original_data[“May”] that have the exact same *started_at*

- *start(end)_station_name(id)_unique* = 1287, 1250, 1254, 1210: since there is a limited number of stations, it is expected that these columns have a smaller number of unique values than the number of entries. However, one would have expected the number of unique station names and station ids to be the same, whereas the ids are less than the names by a small fraction. Which could either by accounted for by the null values or could mean that there are stations that have the different names but the same id.
- *start(end)_lat(long)_unique* = 188591, 185419, 4759, 4762: the start latitude and longitude numbers seem to be as expected, which is less than the total number of entries, but more than the number of stations (since the exact location where a bike is docked can vary within the station especially that the values are given to the 6^{th} decimal place). However, there is a large difference between the number of values in the start (∼188000) and the numbers in the end (∼4800) latitude and longitude. This difference cannot be accounted for by the null values in the end columns, since these were less than 1%. If these values are true, that would mean that users rode there bikes from many start points, but ended up in a much smaller set of points. Which cannot be the case since that would have been reflected in the number of end stations.
- *member_casual_unique* = 2: these 2 unique values are: [member, casual].

• *clean_data*:
After exploring the dataset, we can see that the extractable information can be divided into information about the:

1. rider (casual/member)
2. bike (electric/classic/docked)
3. ride (start-end: time, date, location)

Since, the goal of the analysis is to find out whether to target converting casual riders into members or not, it seems that all the information is relevant to the analysis, with the exception of the ride location. The geographical location would have been important if for example the goal of the analysis was to find out whether more stations should be added and where to do so. Therefore, since the locations seem to be irrelevant, contain null values and discrepancies, these columns will be dropped. The column *ride_id* also does not provide any valuable information for the current analysis. Thus, the first task performed by the method *clean_data* is to drop all columns related to geographical location and ride id, which leaves: *rideable_type*, *started_at*, *ended_at*, *member_casual*.

Next, is data formatting. We have already looked at the columns *rideable_type* and *member_casual*, and ensured that they have the expected values. As for the columns *started_at* and *ended_at*, we need to make sure that the *ended_at* time always comes after *started_at* time. In order to compare the values, they are first converted to a numerical datetime format, since they were stored as strings of characters. Next, by comparing the values and filtering them, we find that there are indeed cases where the *ended_at* time is actually before the *started_at* time. These cases for the month of May are shown in Figure 6.

	rideable_type	started_at	ended_at	member_casual
8308	classic_bike	2023-05-29 17:34:21	2023-05-29 17:34:09	member
38552	electric_bike	2023-05-29 16:57:34	2023-05-29 16:57:27	casual
103546	electric_bike	2023-05-26 15:39:47	2023-05-26 15:38:17	member
103547	electric_bike	2023-05-26 15:38:53	2023-05-26 15:38:17	member
209340	classic_bike	2023-05-07 15:54:58	2023-05-07 15:54:47	casual
211708	classic_bike	2023-05-23 17:39:38	2023-05-23 17:39:35	casual
216859	classic_bike	2023-05-13 18:08:15	2023-05-13 18:08:09	member
336480	electric_bike	2023-05-29 11:31:41	2023-05-29 11:31:33	member
417351	classic_bike	2023-05-27 05:31:51	2023-05-27 05:31:37	member
456170	electric_bike	2023-05-30 07:40:55	2023-05-30 07:39:58	member

Figure 6: Entries in May when the ended_at time is before the started_at time

Looking at the entries in Figure 6, we can see that these are cases when the *ended_at* time is before the *started_at* time by just a few seconds. It can be that in these incidents the start and end time were switched due to some glitch, perhaps the bike rental time being only a few seconds (shorter than the server response time). In the entire dataset of approximately 5.5 Million entries, there is a total of 262 entries that have this issue. Since, the dataset is large, we can simply drop these entries.

Finally, after dropping the columns related to location, and the entries with the switched times, the method *count_entries* is called once again and used to compare the cleaned dataset to the original one. The result is shown in Figure 7. As we can see the number of columns has changed from 13 to 4, and the number of entries is slightly less due to removing the entries where the *ended_at* time is before the *started_at* time.

BikeRides_Original

Month	No Of Entries	No Of Cols
January	190301	13
February	190445	13
March	258678	13
April	426590	13
May	604827	13
June	719618	13
July	767650	13
August	771693	13
September	666371	13
October	537113	13
November	362518	13
Total:	5495804	
Average:	499618	

BikeRides_Cleaned

Month	No Of Entries	No Of Cols
January	190301	4
February	190444	4
March	258678	4
April	426586	4
May	604817	4
June	719611	4
July	767620	4
August	771633	4
September	666321	4
October	537077	4
November	362454	4
Total:	5495542	
Average:	499594	

Figure 7: No of entries and columns before and after cleaning

Analysis:

- *prepare_data*:
  The method *prepare_data* adds two new columns: *ride_length*: the difference between the columns *ended_at* and *started_at* times, and *day_of_week*: extracted from the date in *started_at*.

- *statistics*:
  The method *statistics* calculates for each month the average and maximum ride length, as well as the mode of the day of the week; i.e. the day of the week that occurred the most. The results are shown in Figure 8. The mean ride length varies between 13 - 22 minutes across the months, whereas the maximum ride length is 68 days.

Mean_Max_Mode

Month	Mean Ride Length	Max Ride Length	Mode of Day of Week
January	0:13:00	23 days 8:03:44	Tuesday
February	0:13:31	13 days 2:25:46	Tuesday
March	0:13:04	11 days 16:08:04	Wednesday
April	0:17:12	12 days 18:35:29	Saturday
May	0:19:02	20 days 6:50:31	Tuesday
June	0:19:59	20 days 11:05:58	Friday
July	0:21:44	35 days 17:41:24	Saturday
August	0:22:25	68 days 9:29:04	Wednesday
September	0:17:52	1 day 1:07:46	Saturday
October	0:15:41	1 day 0:59:57	Tuesday
November	0:13:49	1 day 1:00:25	Thursday
Across whole year	0:17:01	68 days 9:29:04	Tuesday

Figure 8: The mean and max ride length, and mode of day of the week

Figure 9: The mean and max ride length, and mode of day of the week for members and casual riders separately