Case Study 1: Cyclistic Bike Share

MS

2022-08-04

1 Introduction

This case study has been prepared as a partial fulfillment for the Capstone project, the final course in Google Data Analytics offered by Google at the Coursera platform.

2 Stage Ask: Business Task Statement

The purpose of this study is to help define a new marketing strategy for *Cyclistic*, a Chicago bike-sharing company. My manager is a director of marketing and the idea for defining a new strategy is to convert *casual riders* into *annual members*.

Casual riders are customers who purchase single or full-day passes, *annual members* are customers who purchase an annual membership in a program.

Cyclistic has been growing since 2016, and has relied on building general awareness and appealing to a broad clientele. Cyclistic has been on a success path since then, however, defining a new marketing strategy might help in growing even more. Cyclistic's finance analysts have concluded that out of two types of riders, annual members are much more profitable than casual riders. Therefore, maximizing the number of annual members might be key to Cyclistic future growth. The easiest way to accomplish this is to convert the casual riders, who have already chosen Cyclistic for their mobility needs, into annual members.

In order to determine a new marketing strategy that would assume converting casual riders into annual members we should answer the following questions:

1. How do annual members and casual riders differ?
2. Why would casual riders buy a membership?
3. How could digital media affect their marketing tactics?

This study addresses the first question and investigates how annual members and casual riders use Cyclistic differently.

3 Stage Prepare: Data Source

The original data is located at [1]. The link is provided by the *Coursera Google Data Analytics* course in the final course - the Capstone project.

3.1 Cyclistic - in real Divvy

Although the name of the company is fictional, the data belongs to the Divvy company which is a real bike share system across Chicago, IL, and Evanston, IL. Divvy is a program of the Chicago Department of Transportation (CDOT) and the Divvy websitecan be found at [2].

The Divvy's data can be found at [3], which eventually points to the same location as provided by the Coursera course.

3.2 Original Data Organization

The original data is organized as .zip files and each archive contains one .csv file. They provide data from 2013 till May 2022 (as of June 14, 2022). Early .zip files contain data quarterly, newer archives contain monthly data.

According to [3]

> "The data has been processed to remove trips that are taken by staff as they service and inspect the system; and any trips that were below 60 seconds in length (potentially false starts or users trying to re-dock a bike to ensure it was secure)."

The trips were anonymized. A single trip record includes trip start (day and time), trip start and end stations (identifier, name, longitude, latitude), rider type, and member type.

3.3 Data Selection

For this analysis, I have chosen data June 2021 - May 2022 (one archive per month). There are a dozen .zip files, each containing a .csv file.

3.4 Data Structure

Each observation consists of 13 attributes that describe a ride identifier, ride type, start station (id, name, longitude, latitude), end station (id, name, longitude, latitude), and membership type.

The number of rows in a single file is within range (inclusive) [103,770; 822,410]. However, there exist a substantial amount of missing values and if missing values are excluded then the number of rows [80,128; 692,321]. Missing values might be recoverable as only name stations are missing (longitude and latitude are not missing), however, it requires further investigation.

- Ride identifier - inconsistent, some are a collection of letters and numbers, shorter (about 3-5 characters), longer (around 12 characters), some are only numbers
- Ride type - according to data: electric_bike, classic_bike, docked_bike
- Start station - the number of stations identified by name is within range [689;1024] and by identifiers [689, 999]
- End station - the number of stations identified by name is within range [689, 1032] and if identifiers are taken into account [689, 1006].
- Membership type - the data (June 2021 - May 2022) shows two types: member and casual; the website [1] claims that there are three: Member, Single Ride, and Day Pass

3.5 Data Credibility

The data credibility is high. The website exists. However, more steps might be required such as phone calls to verify identity of the company or emails.

The data is offered by a program of the Chicago Department of Transportation (CDOT) so confirming its credibility should not be an issue.

3.6 Data Bias

Without further investigation it is hard to tell whether the data is biased. Potential bias might include logging data from preferred stations, although it might also mean that some stations are simply more popular. The data has been anonymized. It might make determining the existence of bias difficult.

CDOT is a governmental institution and should follow guidelines regarding collecting data in an unbiased manner.

3.7 Potential Issues

I have identified several data issues that require further investigation.

- Missing values - roughly 15-29% rows contain missing values. They seem to be names of the stations so it should be relatively easy to recover or disregard them by creating a map based on longitude and latitude.
- Discrepancies in stations identified by name and by id. It seems that there are less station identifiers than their names. That might mean that one name may have more than one identifier. I can use coordinates to identify stations, and map them to station names, if necessary.

3.8 License

The Divvy license for data use is available at [4].

In brief, Divvy allows for data download, mapping, and visualization. More specifically, it permits to

> “access, reproduce, analyze, copy, modify, distribute […] and use the Data for any lawful purpose.”

The license does not allow the user to connect the Bikeshare’s website and the Data, as well as any attempt to identify riders or their personal information.

3.9 Research Methodology

I used:

- Coursera Google Data Analytics instructions
- Divvy website [2] - to gather info about the data source, data and the license.
- RStudio v. 2022.02.2+485 “Prairie Trillium” Release for Windows Mozilla/5.0 - used for the preliminary analysis.

4 Stage: Process

I could not use the software that was used during the course:

- BigQuery allows only for uploading files less than 100MB; several csv files exceed that limit. I could not use Google Drive because BigQuery does not accept the URL format of the link generated by Google Drive a URL share link.
- It was difficult to import csv files to Google Sheets and MS Excel due to their sizes.

I decided to use SQLite [5] with R for performance reasons. It occurred that SQLite has many limitations and surprising assumptions and I would not recommend SQLite for using SQLite for this project.

In fact, during this project I used SQLite, bash, R, MS Excel, and Tableau. I ended up with writing bash scripts, R scripts, MS Excel spreadsheets, and Tableau viz.

4.1 Data Cleaning

In this stage I interacted with SQLite:

- directly, via command line,
- via R, or
- by running bash scripts with SQL queries.

I have imported 12 csv files from June 2021 to May 2022 to SQLite and created 12 tables. (db.R)

Tables' names are in format: b_yyyymm, e.g., b_202106 corresponds to 202106-divvy-tripdata.csv.

I have also found out in the Analysis stage the need for data cleaning.

The tables schema is as follows:

```
CREATE TABLE `b_202106` (
  `ride_id` TEXT,
  `rideable_type` TEXT,
  `started_at` TEXT,
  `ended_at` TEXT,
  `start_station_name` TEXT,
  `start_station_id` TEXT,
  `end_station_name` TEXT,
  `end_station_id` TEXT,
  `start_lat` REAL,
  `start_lng` REAL,
  `end_lat` REAL,
  `end_lng` REAL,
  `member_casual` TEXT
);
```

The initial number of rows, final number of rows after the cleaning phase and the number of rows removed are summarized in Table 4.1.

Table 4.1: Row counts after import from csv files to SQLite tables.

Table	Row_count	Final_row_count	Removed_rows
b_202106	729595	608863	120732
b_202107	822410	692412	129998
b_202108	804352	674493	129859
b_202109	756147	621284	134863
b_202110	631226	478072	153154

Table	Row_count	Final_row_count	Removed_rows
b_202111	359978	255911	104067
b_202112	247540	176392	71148
b_202201	103770	80135	23635
b_202202	115609	89188	26421
b_202203	284042	216011	68031
b_202204	371249	272585	98664
b_202205	634858	502611	132247

During the cleaning phase, I checked for:

- null values in all fields – none found,
- zero-length values in the fields – addressed existing found,
- padded spaces – none found,
- duplicates – no duplicates found
- ranges in all fields:
    - addressed negative trip durations or too short trip durations (<= 5min),
    - addressed unidentifiable trip end locations
    - duplicates in bike stations ids and stations names
    - discovered a database issue with the member_casual field (‘0D’ - Carriage Return) and proposed a hack to workaround this issue

The final database is summarized in Table 4.1.

Table 4.2: Total rows counts before and after removal.

Initial total row count	Final total row count	Total rows removed	Total rows removed %
5860776	4667957	1192819	20%

4.1.1 No Null Values

Script: clean.R

There are no null values.

However, this is one of the SQLite surprises. Apparently, SQLite stores data as characters and has only five fundamental data types: integer, float, blob, string and NULL. The import of csv into SQLite tables has not created NULL values, however, there were zero-length string values.

4.1.2 Zero-length Field - Exist

Script: clean.R

There are zero-length values in fields: start_station_name, start_station_id, end_station_name and end_station_id. Column ‘total_rows’ indicates the total number of rows in a corresponding SQL table.

```
         total_rows ride_id rideable_type started_at ended_at start_station_name start_station_i
d
b_202106     729595       0             0          0        0              80093             8009
3
b_202107     822410       0             0          0        0              87263             8726
2
b_202108     804352       0             0          0        0              88458             8845
8
b_202109     756147       0             0          0        0              93113             9311
1
b_202110     631226       0             0          0        0             108210            10821
0
b_202111     359978       0             0          0        0              75290             7529
0
b_202112     247540       0             0          0        0              51063             5106
3
b_202201     103770       0             0          0        0              16260             1626
0
b_202202     115609       0             0          0        0              18580             1858
0
b_202203     284042       0             0          0        0              47246             4724
6
b_202204     371249       0             0          0        0              70887             7088
7
b_202205     634858       0             0          0        0              86704             8670
4
         end_station_name end_station_id member_casual
b_202106            86387          86387             0
b_202107            93158          93158             0
b_202108            94115          94115             0
b_202109            99261          99261             0
b_202110           114834         114834             0
b_202111            79187          79187             0
b_202112            53498          53498             0
b_202201            17927          17927             0
b_202202            20355          20355             0
b_202203            51157          51157             0
b_202204            75288          75288             0
b_202205            93171          93171             0
```

4.1.3 Identical Start and End Coordinates

Script: get_loc.sh

There are trips that have identical start coordinates and end coordinates. For instance for coordinates: 41.8|-87.59|41.8|-87.59 there are 408 entries.

```
table    count
b_202106 408
b_202107 418
b_202108 491
b_202109 1101
b_202110 3915
b_202111 4233
b_202112 1695
b_202201 331
b_202202 879
b_202203 1781
b_202204 4294
b_202205 2314
```

I left them, as the riders might return bikes to the location that was a start location.

4.1.4 No Need to Trim Padded Spaces

Script: find_trim.sh

I compared the trimmed fields to no trimmed fields. Zero results were returned.

4.1.5 Duplicates Check

Script: find_dup.sh

I have not found rows with duplicated ride_id.

I have found duplicates in stations ids and station names - more in 4.1.6.3.1.2.

4.1.6 Range Check

4.1.6.1 ride_id

Script: minmax_ride_id.sh

The ride_id identifiers are 16 characters long in all databases.

4.1.6.2 rideable_type

Script: bike_types.sh

There are three types in rideable_type.

- classic_bike,
- docked_bike,
- electric_bike.

No need to upper or lower-case them. All tables have those types present.

4.1.6.3 Date Check

Script: dates.sh

Date ranges are as expected. There are no trips that start in previous or next months.

There are, however, “negative” trips, i.e., trips that ended before they started. It affects 7 tables, and the range of record numbers with negative duration trips ranges from 1 to 53 per table. The negative duration ranges from 1sec to 3482 sec. I decided to remove those records. It might be worth to inform Divvy about negative duration trips entries so they might check their tracking devices to eliminate such a behavior in the future.

```
table|count|min_duration[sec]|max_duration[sec]
b_202106|5|-1.0|-306.0
b_202107|13|-1.0|-12.0
b_202108|29|-1.0|-167.0
b_202109|36|-1.0|-423.0
b_202110|0||
b_202111|53|-1010.0|-3482.0
b_202112|0||
b_202201|0||
b_202202|0||
b_202203|2|-7.0|-356.0
b_202204|0||
b_202205|1|-12.0|-12.0
```

4.1.6.3.1 Coordinates Check

Scripts: clean.R, minmax_lat_lng.sh, del_missing_ends.sh, det_stations.sh, stations.R, and update_stations.R

There are no zero-length start coordinates, however, there are zero-length end coordinates as indicated below. That is again an SQLite curiosity that shows that SQLite although coordinates are REAL (see Section 4.1 for the schema), they can be also treated as strings by SQLite.

```
table       total_rows start_lat start_lng end_lat end_lng
b_202106     729590    0         0     717     717
b_202107     822397    0         0     731     731
b_202108     804323    0         0     706     706
b_202109     756111    0         0     595     595
b_202110     631226    0         0     484     484
b_202111     359925    0         0     191     191
b_202112     247540    0         0     144     144
b_202201     103770    0         0      86      86
b_202202     115609    0         0      77      77
b_202203     284040    0         0     266     266
b_202204     371249    0         0     317     317
b_202205     634857    0         0     722     722
```

4.1.6.3.1.1 Unindentifiable End Locations

Queries will regard routes that start and end at known locations. Each location can be identified by one of the fields:

- _station_name (start/end)
- _station_id (start/end)
- geolocation (latitude, longitude)

If all of them are missing, the route cannot be identified.

The end geolocations are missing in all tables and the count of missing geolocations ranges from 77 rows to 722, as shown in the above table. Start locations are not missing.

The script minmax_lat_lng.sh checks whether the rows miss all of the fields that allow for identifying the endpoint of the trip, and counts them in tables.

```
table_name rows_count
b_202106    717
b_202107    731
b_202108    706
b_202109    595
b_202110    484
b_202111    191
b_202112    144
b_202201    86
b_202202    77
b_202203    266
b_202204    317
b_202205    722
```

I decided to remove all the rows with the missing endpoints (script del_missing_ends.sh).

4.1.6.3.1.2 Duplicates in Stations Ids and Stations Names

There are duplicate station ids which have different names, and have slightly different geolocations. That is why I treat the name of the station as its identifier, and not the station_id as station_id is not unique. It was a reason why I needed to rerun the scripts with the names as ids. And adapt them to use station names instead of station_id as identifiers.

The below stations have more than one identifiers assigned (v_stations is a view created by joining start stations and end stations across all tables describing bike trips):

```
select count(*), name from
  (select  name, id from v_stations where length(name) != 0 group by name, id) t
    group by t.name having count(*) > 1;

2|Bissell St & Armitage Ave - Charging
2|California Ave & Cortez St
2|Calumet Ave & 51st St
2|Central Park Ave & Ogden Ave
2|Christiana Ave & Lawrence Ave
2|Halsted St & 111th St
2|Jeffery Blvd & 67th St
2|Lake Park Ave & 47th St
2|Lakefront Trail & Bryn Mawr Ave
2|Loomis St & 89th St
2|Prairie Ave & Garfield Blvd
2|WEST CHI-WATSON
2|Wabash Ave & 87th St
```

Below are counted identifiers that have more than one station names assigned:

```
sqlite> select count(*), id from (select  id, name from v_stations
          where length(id) != 0 group by id, name) t
            group by t.id having count(*) > 1;

2|13099
2|13221
2|13300
2|20215
2|351
2|444
2|514
2|515
2|517
2|518
2|519
2|520
2|534
2|535
2|536
2|543
2|545
2|546
2|549
2|553
2|554
2|559
2|560
2|561
2|562
2|564
2|567
2|569
2|570
2|571
2|572
2|574
2|586
2|587
2|595
2|596
2|599
2|605
2|620
2|623
2|639
2|642
2|644
2|646
2|650
2|657
2|658
2|661
```

```
2|662
2|665
2|Hubbard Bike-checking (LBS-WH-TEST)
2|LF-005
3|TA1306000029
2|TA1307000041
2|TA1309000039
2|TA1309000042
2|TA1309000049
2|WL-008
2|chargingstx1
```

I identified a few stations by manually comparing ids, lat and lng with the auxilary files names_ids_stations.txt and empty_name_stations.txt generated by the script det_stations.sh.

```
station_id    station_name
13221         Wood St & Milwaukee Ave
20215         Hegewisch Metra Station
WL-008        Clinton St & Roosevelt Rd
```

I saved 781 files by identifying the end_stations based on lat and lng. For this I used script det_stations.sh. I also used stations.R and update_stations.R. The remaining stations that I identified in scripts det_stations.sh and stations.R are Unidentifiable at this point as they require more sophisticated analysis with lat and lng. The same lat and lng can fall into several name stations and choosing the right one requires more effort and time. For this project I decided to delete the rows that represent the trips which cannot be fully identified.

I updated auxilary tables and views in my SQLite database appropriately. It resulted in the total number of rows as follows:

```
table     rows_count    rows_removed
b_202106    608863      -120727
b_202107    692412      -129985
b_202108    674493      -129830
b_202109    621284      -134827
b_202110    478072      -153154
b_202111    255911      -104014
b_202112    176392      -71148
b_202201    80135       -23635
b_202202    89188       -26421
b_202203    216011      -68029
b_202204    272585      -98664
b_202205    502611      -132246
```

4.1.6.3.2 member_casual field check

Script: member_type.sh

Each table contains twp different values for the member_casual field. There are neither zero nor NULL values.

```
table       count  member_type
b_202106    304268    casual
b_202106    304595    member
b_202107    369495    casual
b_202107    322917    member
b_202108    341550    casual
b_202108    332943    member
b_202109    293023    casual
b_202109    328261    member
b_202110    189180    casual
b_202110    288892    member
b_202111    69986   casual
b_202111    185925    member
b_202112    45091   casual
b_202112    131301    member
b_202201    12609   casual
b_202201    67526   member
b_202202    15150   casual
b_202202    74038   member
b_202203    67175   casual
b_202203    148836    member
b_202204    91914   casual
b_202204    180671    member
b_202205    220285    casual
b_202205    282326    member
```

During the analysis stage I discovered that this field causes issues with SQLite. If you check in the database with the hex function that shows the content of the field as the hexadecimal ASCII codes, the last character is '0D' which is CR (carriage return). This character is not printable, however, it confuses SQLite regarding queries with count(*). If the field is included at the beginning of the SELECT query and eventually, followed by COUNT()*, the count is not printed as the CR is interpreted. It is probably an SQLite bug, which I intend to report. In order to overcome this issue:

1. I used the field as the last in the query and the output was produced,
2. I used only the first letter of the field, i.e., SUBSTR(field, 1,1) (it requires an extra computational effort; the better solution would be to overwrite the field excluding trailing '0D' character, probably with the TRIM function)

4.1.7 Removed Trip Durations <= 5 minutes

Script: analysis.sh

I created the view v_durations in my db that has precomputed durations in seconds, minutes, and hours in one big view (script analysis.sh). SQLite does not have a date type and the relevant data-related values need to be calculated at during the query time. For performance reasons, I decided to create a view with precomputed values.

There were 4,667,275 records combined from all cleaned tripdata with min, avg and max minute duration: 0, 20, 55944. Although, the Divvy website claims the trips < 60 seconds were removed, I have found trips with 0-minute duration.

```
select count(*), min(mins), avg(mins), max(mins) from v_durations;
4667275   0.0   20.215567756346   55944.0
```

There were 868,413 trips at most 5min.

```
rider_type   trips <= 5min
--------------------------
casual       197426
member       670987
--------------------------
total        868413
```

I decided to remove trips where duration is <= 5min, as too short trips to take them into account during the analysis stage. I created a v_5min_up_durations view which contained trips with durations greater than 5min.

The final number of records for the analysis purposes is shown at the beginning of Table 4.2.

4.2 Tools to Use for the Analysis Stage

- Bash, R, command line to interact with SQLite
- MS Excel for viz, and tabularization
- Tableau for viz

5 Stage: Analysis

Scripts, files: analysis.sh, analysis.xslx, stations.xslx, ids_stations_coord.xlsx

In order to find differences between casual and member riders I searched for answers to the following questions:

1. What is the typical duration of the ride by a casual or a member rider?
2. When do they ride? Days of the week, time of the day?
3. Where do they ride? What are 10 top popular stations?
4. Which bikes do they ride?

There are 4,667,275 trips from June, 2021 to May, 2022 out of which 52% were performed by member riders and 48% were ridden by casual riders.

The key findings:

1. Majority of bike trips for both casual and member riders are under 1h long (~90%).
2. Casuals use more bikes on weekends (Saturday, Sunday) compared to work days. Members use bikes evenly over the week; slightly less on Sunday in comparison to other days. Afternoon and evening trips, i.e., from 12pm to 00am, dominate trips of Casuals regardless of the month. Afternoon and evening trips dominate Members trips. The difference between the number of morning and afternoon trips is less than 5% for Members over the year, whereas for Casuals it ranges between 16% and 27%.
3. Six of the ten top start stations are common among casual and member riders, Five of the ten top end stations are common among casual and member riders.
4. Casuals and members use more classic bikes than electric bikes. Casuals have the choice to use also ‘docked’ bikes.

5.1 Main Differences Between Casuals and Members

The top three main differences include:

1. Weekdays for riding - weekends vs. workdays

- Casuals ride more on weekends than workdays
- Members ride more on workdays than weekends

2. Trip duration Members trips are short, Casuals do short trips but also longer ones

- 62% of all member trips <15min, 1% trips >= 1h
- 40% of all casual trips <15min, 10% trips >=1h

3. Time of the day Members did more morning trips than Casuals (28% vs 18%). This might indicate a commuting pattern.

6 Stage: Share

Software: MS Powerpoint

I created a presentation to communicate my findings to the executive team of Cyclistic.

7 Stage Act: Recommendations

In order to convert Casuals to Members, based on findings in this project, I would recommend:

1. A marketing campaign to advertise benefits of riding during workdays as combined benefits: environment friendly, workout, pleasure, etc.
2. Encouraging Casuals to use bikes in the mornings in their routines for commuting, workout, and/or pleasure.
3. Offering free/discounted daily passes/discounts for workdays for Casuals.
4. Offering free/discounted daily passes for weekends - it might encourage Casuals to become Members.
5. Offering “transitional passes” such as 2-/3-day passes for workdays.

8 Next Steps

Given timeframe for this project, it was preliminary analysis that has given some insight into differences regarding bikes usage by Members and Casuals.

- Find out why Casuals ride less in the mornings.
- More analysis is required to obtain a better understanding and profiling a member and casual user.
- Future explorations might include searching for more sophisticated relationships that include more features like timing, start and end stations, etc.

9 Contact

Scripts and other info are available upon request. Please contact me via Kaggle.

References

[1] Divvy, “Cyclistic data.” Coursera Google Data Analytics, Jun. 2022. Available: https://divvy-tripdata.s3.amazonaws.com/index.html (https://divvy-tripdata.s3.amazonaws.com/index.html)
[2] Divvy, “The Divvy Website.” Jun. 2022. Available: https://divvybikes.com/ (https://divvybikes.com/)
[3] Divvy, “Divvy Data.” Jun. 2022. Available: https://ride.divvybikes.com/system-data (https://ride.divvybikes.com/system-data)
[4] Divvy, “Divvy Data License Agreement.” Jun. 2022. Available: https://ride.divvybikes.com/data-license-agreement (https://ride.divvybikes.com/data-license-agreement)
[5] SQLite, “SQLite.” Jul. 2022. Available: https://www.sqlite.org (https://www.sqlite.org)