Citi Bike Data Profiling and Cleaning

1. Shell Command

This section includes the complete workflow for:

- Upload the Citi Bike raw dataset to the Dataproc cluster
- Steps to organize the dataset inside HDFS

Authenticate Google Cloud account and Environment Setup:

```
gcloud auth login
gcloud config set project hpc-dataproc-19b8
gcloud config set compute/zone us-central1-f
gcloud auth list
```

Transfer Monthly Zip Files to Dataproc cluster:

```
gcloud compute scp 202401-citibike-tripdata.zip
hx2487_nyu_edu@nyu-dataproc-m:~/citibike_data --zone=us-central1-f
gcloud compute scp 202403-citibike-tripdata.zip
hx2487_nyu_edu@nyu-dataproc-m:~/citibike_data --zone=us-central1-f
.........
```

The Citi Bike raw data is stored in monthly zip archives. I use gcloud compute scp in the local environment to upload these zip files one by one to Dataproc and place them in the citibike_data folder.

Unzip Monthly Archives on the cluster:

```
cd citibike_data
for f in *.zip; do ym=${f:0:6}; unzip "$f" -d "$ym"/; done
```

After logging in to nyu-dataproc-m, navigate to the directory where the zip files are stored and use a simple Bash loop to batch extract all zip files.

Create HDFS Directory Structure:

```
hadoop fs -mkdir -p citibike/raw/2024
hadoop fs -mkdir -p citibike/raw/2025
hadoop fs -mkdir -p citibike/raw/2024/{01..12}
hadoop fs -mkdir -p citibike/raw/2025/{01..10}
```

To efficiently manage Citi Bike data within HDFS, I employ a year- and month-partitioned directory layout. Moreover, this raw data was originally partitioned in the same way.

Load CSV Files into HDFS:

```
for d in 2024*/ 2025*/; do
    ym=${d%/}
    year=${ym:0:4}
    month=${ym:4:2}
    hadoop fs -put "$ym"/*.csv "citibike/raw/$year/$month/"
done
```

Using this Bash loop, the CSV files will be automatically uploaded to the corresponding year and month directory in HDFS

Check directory structures and CSV files are correct:

```
hadoop fs -ls -R citibike/raw
```

2. Data Profiling

I first collected all the monthly zip files for 2024 and 2025 and then uploaded them to the cluster using gcloud compute scp, as mentioned in Section 1, Shell Command. Since uploading all the files at once failed several times, I can only upload them one month at a time.

Then, I implemented a Python script called "check_columns_order.py" that takes an absolute folder path as input and checks whether all CSV files in that folder use the same column order. The results confirm that all files follow a consistent column order. This step can ensure that the next MapReduce can safely parse the data.

2.1 Implement a MapReduce Data Profiling

Mapper Function: I designed the mapper to output (columnName, rawValue) pairs for each of the 13 columns in every CSV file. These include 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.

Reducer Function: Here is the detailed description of different column profiling.

- ride_id: I scan all ride_id values, count how many records exist, and detect how many entries are empty. The goal is to check the completeness of the data and identify the missing identifiers.

- Ridetable_type and member_casual: For these small categorial fields, I only count the total record, missing values, and the frequency of each category. The goal is to create a simple distribution summary that helps identify imbalances or unexpected cases.

- Start_station_name and end_station_name: Same ideas with ride_id

- Start_station_id and end_station_id: I count the total number, missing entries, and values that do not match a numeric pattern. The goal is to detect formatting issues.

- started_at and ended_at: This function validates the timestamp format and tracks the minimum and maximum time observed. It also counts missing values, invalid formats, and timestamps outside the expected year range.

- start_lat , start_lng, end_lat, and end_lng. For latitude and longitude, I count the missing values, values that cannot be parsed into numbers, and out-of-bounds numbers that fall outside NYC's geographic bounds. The goal is to identify the invalid and misplaced coordinates efficiently.

2.2 Compile MapReduce Program

After implementing the MapReduce function, I compiled the Java files using:

```
javac -classpath `hadoop classpath` *.java
jar cvf citibike.jar *.class
```

2.3 Running the MapReduce Program

My first test was a "full dataset profiling run", but this test failed with "Halting due to Out of Memory Error".

```
hadoop jar citibike.jar DataProfiling citibike/raw
```

```
citibike/output_profile_all
```

Two different issues mainly caused the failure. Firstly, I used two HashSet objects inside the Reducer, which caused the job to crash during the reduce phase because the reducer attempted to hold a large number of values in memory simultaneously. Although I fixed the analysis logic and removed the HashSet, the same failure appeared again. This time, the issue was caused mainly by the size of the dataset itself. Running the full 15GB dataset in a single job reached the cluster's limit.

To avoid the memory error in the reducer function, I reran data profiling separately for each year.

```
hadoop jar citibike.jar DataProfiling citibike/raw/2024
citibike/output_profile_2024

hadoop jar citibike.jar DataProfiling citibike/raw/2025
citibike/output_profile_2025
```

This change immediately resolved the memory error issue, as it reduced the original workload by almost half, allowing the reducer to process the data without running out of memory.

2.4 Output Analytics

Both jobs completed successfully without memory error issues, and the reducer produced complete profiling summaries for every column.

Output for 2024 Dataset:

```
end_lat     Total=44303209, missing=122304(0.28%), invalid=0(0.00%),
outOfRange=179(0.00%)
end_lng     Total=44303209, missing=122304(0.28%), invalid=0(0.00%),
outOfRange=179(0.00%)
end_station_id    Total=44303209, missing=122573(0.28%),
invalid_format=8351(0.02%)
end_station_name  Total=44303209, missing=114545(0.26%)
ended_at    Total=44303209, missing=0(0.00%), invalid=0(0.00%),
miss_precision=0(0.00%), out_of_range=0(0.00%),  min_time=2024-01-01
00:00:08.272, max_time=2024-12-31 23:59:48.482, range=[2024-01-01
00:00:00.000 to 2025-11-01 00:00:00.000]
member_casual     Total=44303209, missing=0(0.00%), unique=2,
values={casual=8556931, member=35746278}
ride_id     Total=44303209, missing=0(0.00%)
```

```
rideable_type     Total=44303209, missing=0(0.00%), unique=2,
values={classic_bike=15007957, electric_bike=29295252}
start_lat   Total=44303209, missing=29835(0.07%), invalid=0(0.00%),
outOfRange=0(0.00%)
start_lng   Total=44303209, missing=29835(0.07%), invalid=0(0.00%),
outOfRange=0(0.00%)
start_station_id  Total=44303209, missing=29835(0.07%),
invalid_format=3703(0.01%)
start_station_name      Total=44303209, missing=29835(0.07%)
started_at  Total=44303209, missing=0(0.00%), invalid=0(0.00%),
miss_precision=0(0.00%), out_of_range=410(0.00%),  min_time=2023-12-31
02:36:55.648, max_time=2024-12-31 23:57:46.390, range=[2024-01-01
00:00:00.000 to 2025-11-01 00:00:00.000]
```

The 2024 dataset contains 44303209 trip records. Several columns have no data quality issues at all, which include ended_at, member_casual, ride_id, and riderable_type. All of them are stable and valid data for these attributes. The other columns only have some minor issues, such as a small percentage of missing values, formatting mismatches, and a few out-of-range data. For example, end_lat and end_lng have 0.28% of missing, and 179 records are out-of-range. We can easily notice that the proportion of problem data across all columns is extremely low, usually below 0.3%. This shows that the 2024 dataset is very clean and reliable.

Output for 2025 Dataset:

```
end_lat     Total=40262430, missing=115952(0.29%), invalid=0(0.00%),
outOfRange=34(0.00%)
end_lng     Total=40262430, missing=115952(0.29%), invalid=0(0.00%),
outOfRange=34(0.00%)
end_station_id    Total=40262430, missing=116113(0.29%),
invalid_format=22675(0.06%)
end_station_name  Total=40262430, missing=109432(0.27%)
ended_at    Total=40262430, missing=0(0.00%), invalid=0(0.00%),
miss_precision=0(0.00%), out_of_range=0(0.00%),  min_time=2025-01-01
00:00:04.075, max_time=2025-10-31 23:59:59.794, range=[2024-01-01
00:00:00.000 to 2025-11-01 00:00:00.000]
member_casual     Total=40262430, missing=0(0.00%), unique=2,
values={casual=7262402, member=33000028}
ride_id     Total=40262430, missing=0(0.00%)
rideable_type     Total=40262430, missing=0(0.00%), unique=2,
values={electric_bike=28269491, classic_bike=11992939}
```

```
start_lat   Total=40262430, missing=17737(0.04%), invalid=0(0.00%),
outOfRange=16(0.00%)
start_lng   Total=40262430, missing=17737(0.04%), invalid=0(0.00%),
outOfRange=16(0.00%)
start_station_id  Total=40262430, missing=17737(0.04%),
invalid_format=19759(0.05%)
start_station_name      Total=40262430, missing=17737(0.04%)
started_at  Total=40262430, missing=0(0.00%), invalid=0(0.00%),
miss_precision=0(0.00%), out_of_range=0(0.00%),  min_time=2024-12-30
23:41:25.635, max_time=2025-10-31 23:58:16.325, range=[2024-01-01
00:00:00.000 to 2025-11-01 00:00:00.000]
```

The 2025 dataset contains 40262430 trip records, and its overall quality is on the same level as the 2024 dataset. The columns without any issues include ended_at, member_casual, ride_id, rideable_type, and started_at. We can see that the 2025 dataset has one more column without any issues. This difference comes from how the dataset is designed. The 2024 dataset includes a small number of trips that started before January 1, 2024, which makes those timestamps in the outside of the expected range. However, the 2025 dataset can include trips that started a little before January 1, 2025 and they are still within our expected time range (2024-01-01 00:00:00.000 to 2025-11-01 00:00:00.000). That's why the started_at column in 2025 does not show the same issue. The remaining columns exhibit similar data issues to those in the 2024 dataset.

Overall, both the 2024 and 2025 datasets exhibit very high data quality, and all data issue ratios remain below 0.3%. These insights directly help me take the next step of data cleaning, and these minor data issues will be corrected to produce a fully clean and analysis-ready dataset.

3. Data Cleaning

This section removes rows with missing or invalid values from the dataset, and the cleaning rules are directly based on the issues detected during the data profiling stage. This step ensures that all remaining data meets the expected formats, stays within valid geographic boundaries, satisfies the timestamp constraints, and contains no missing or invalid data. In this section, I only used the driver and mapper because the task is only to filter out the problematic values. Since there is no need to aggregate or merge values by column, a reducer function is unnecessary. I can directly use the mapper output as the cleaned results.

3.1 Compile MapReduce Program

After implementing the MapReduce function, I compiled the Java files using:

```
javac -classpath `hadoop classpath` *.java
jar cvf citibike.jar *.class
```

3.2 Running the MapReduce Program

```
hadoop jar citibike.jar DataCleaning citibike/raw/2024 citibike/clean/2024
hadoop jar citibike.jar DataCleaning citibike/raw/2025 citibike/clean/2025
```

3.3 Reran Data Profiling

```
hadoop jar citibike.jar DataProfiling citibike/clean/2024
citibike/clean/output_2024

hadoop jar citibike.jar DataProfiling citibike/clean/2025
citibike/clean/output_2024
```

3.4 Output Analytics

Output for 2024 Dataset after data cleaning:

```
end_lat     Total=44153597, missing=0(0.00%), invalid=0(0.00%),
outOfRange=0(0.00%)
end_lng     Total=44153597, missing=0(0.00%), invalid=0(0.00%),
outOfRange=0(0.00%)
end_station_id    Total=44153597, missing=0(0.00%), invalid_format=0(0.00%)
end_station_name  Total=44153597, missing=0(0.00%)
ended_at    Total=44153597, missing=0(0.00%), invalid=0(0.00%),
miss_precision=0(0.00%), out_of_range=0(0.00%),  min_time=2024-01-01
00:03:03.428, max_time=2024-12-31 23:59:48.482, range=[2024-01-01
00:00:00.000 to 2025-11-01 00:00:00.000]
member_casual     Total=44153597, missing=0(0.00%), unique=2,
values={casual=8495469, member=35658128}
ride_id     Total=44153597, missing=0(0.00%)
rideable_type     Total=44153597, missing=0(0.00%), unique=2,
values={classic_bike=14996175, electric_bike=29157422}
start_lat   Total=44153597, missing=0(0.00%), invalid=0(0.00%),
outOfRange=0(0.00%)
start_lng   Total=44153597, missing=0(0.00%), invalid=0(0.00%),
outOfRange=0(0.00%)
```

```
start_station_id  Total=44153597, missing=0(0.00%), invalid_format=0(0.00%)
start_station_name      Total=44153597, missing=0(0.00%)
started_at  Total=44153597, missing=0(0.00%), invalid=0(0.00%),
miss_precision=0(0.00%), out_of_range=0(0.00%),  min_time=2024-01-01
00:00:03.208, max_time=2024-12-31 23:57:46.390, range=[2024-01-01
00:00:00.000 to 2025-11-01 00:00:00.000]
```

Output for 2025 Dataset after data cleaning:

```
end_lat     Total=40090877, missing=0(0.00%), invalid=0(0.00%),
outOfRange=0(0.00%)
end_lng     Total=40090877, missing=0(0.00%), invalid=0(0.00%),
outOfRange=0(0.00%)
end_station_id    Total=40090877, missing=0(0.00%), invalid_format=0(0.00%)
end_station_name  Total=40090877, missing=0(0.00%)
ended_at    Total=40090877, missing=0(0.00%), invalid=0(0.00%),
miss_precision=0(0.00%), out_of_range=0(0.00%),  min_time=2025-01-01
00:00:04.075, max_time=2025-10-31 23:59:59.794, range=[2024-01-01
00:00:00.000 to 2025-11-01 00:00:00.000]
member_casual     Total=40090877, missing=0(0.00%), unique=2,
values={casual=7208456, member=32882421}
ride_id     Total=40090877, missing=0(0.00%)
rideable_type     Total=40090877, missing=0(0.00%), unique=2,
values={electric_bike=28120395, classic_bike=11970482}
start_lat   Total=40090877, missing=0(0.00%), invalid=0(0.00%),
outOfRange=0(0.00%)
start_lng   Total=40090877, missing=0(0.00%), invalid=0(0.00%),
outOfRange=0(0.00%)
start_station_id  Total=40090877, missing=0(0.00%), invalid_format=0(0.00%)
start_station_name      Total=40090877, missing=0(0.00%)
started_at  Total=40090877, missing=0(0.00%), invalid=0(0.00%),
miss_precision=0(0.00%), out_of_range=0(0.00%),  min_time=2024-12-31
12:54:23.643, max_time=2025-10-31 23:58:16.325, range=[2024-01-01
00:00:00.000 to 2025-11-01 00:00:00.000]
```

After running the cleaning job, the data profiling program was applied again on the cleaned datasets. We can notice that: 0% missing values across all columns, no invalid values, no out-of-range values, all coordinates fall within NYC bounds. This validates that the cleaning logic successfully removes all problematic data. Overall, this data cleaning effectively cleans two large raw datasets into fully validated datasets.