Dataset Stewarship Documentation

Hazem Haffouz

Cyclistic bike-share Data Preparing and Processing Report:

Initial Data Acquisition:

Data Was provided raw, however annonimized by the stakeholders, sourced from four different quarters:

- Divvy_Trips_2019_Q2
- Divvy_Trips-2019-Q3
- Divvy_Trips-2019-Q4
- Divvy_Trips-2020-Q1

Each dataset had its own set of inconsistencies in column names, orders, and numbers.

Schema Information:

The data covers trips made by cyclists and is divided into quarterly tables, Each table contains variables like ride_id, started_at, ended_at, bike_id, trip_duration, start_station_id, end_station_name, etc. Different tables might have slightly different column names and data types.

ROCCC Assessment:

- **Relevance**: The dataset is partialy relevant for analyzing cyclist behavior, trip duration, station performance, and other ride-related metrics.
- **Originality**: The dataset is a primary source of information and is assumed to be coming directly from the service that provides these bike rentals.
- **Comprehensive**: With a schema that includes trip identifiers, time stamps, station IDs, and user information, the dataset is fairly comprehensive. However, it lacks some demographic details and bike technical specifications that could make it even more comprehensive.
- **Consistency**: The dataset shows some inconsistency in column naming across different quarterly tables. Some variables also show inconsistent data types.

- **Credibility**: The credibility of the dataset is assumed to be high as it is sourced from a presumably reliable data warehouse.

Integrity, Vitality, and Validity

- **Integrity**: Data integrity has been partially compromised due to missing values, inconsistencies in naming conventions, and some illogical data points like negative trip durations.
- **Vitality**: The dataset is vital for any behavioral analytics related to the cyclists, trip efficiency, and operational efficiency of the cycling service.
- **Validity**: The validity of the data is subject to the cleaning and normalization processes. Post-cleaning, the data should be highly valid for analytical purposes.

Solution:

Given the massive size and complexity of the data, Google Cloud's BigQuery was chosen for processing and querying. SQL queries were executed to summarize the structure of each dataset after uploading it to Google Cloud.

Suggestions

Data Enrichment(Future Project): Adding weather data or public holiday data to provide more context for the rides.

Advanced Analytics: With the cleaned dataset, time-series analysis or and converstion Analysis.

Stakeholder Communication: Regular updates should be given to stakeholders, especially about data inconsistencies and missing data, to improve data quality in the future.

Automation: Automating the cleaning and transformation steps for efficiency and consistency.

Data Governance: Implement better data governance practices to improve data quality and compliance.

Limitations:

Limited Feature:

- **Type of Bikes**: The dataset does not provide information on the type of bikes being used (e.g., electric, standard, tandem). This limits the ability to analyze user preferences and the demand for different bike types.
- **Geographical Coordinates**: The absence of geographical coordinates for the start and end stations limits spatial analyses that could provide insights into optimal bike station placements, route planning, or even identifying high-traffic areas.

- Accounts and User Behavior*: The dataset lacks more granular data on user accounts, which restricts the scope of user behavior analysis. Features like frequency of rides per user, loyalty, or even user ratings are not available.

High Volume of Nulls:

- Missing data can be particularly limiting when trying to conduct a comprehensive analysis. Fields with a high number of nulls (e.g., gender, birthyear) not only reduce the statistical power of tests but also can introduce bias.

Lack of Contextual Data **(Future Project)**:

- Contextual data such as weather conditions, time of day, and traffic conditions can significantly impact bike rentals. The absence of this information limits the ability to conduct more nuanced analyses, like how weather affects rental duration or how time of day correlates with user types.

Data Source Integrity:

- The dataset does not provide metadata or a data dictionary that describes the source of each column, the data collection methods, or the data reliability.

Lack of Historical User Data*:

- Information on user history (e.g., past rentals, loyalty program status, etc.) is not available. This limits the ability to perform analyses like customer lifetime value or repeat rental behavior.

***Note that User Data generally considered sensitive and thus has been anonymized, Howeve there are other methods of anonymization other than deletion, like encrypting the generic user account number without PII**

Data Pre-processing

Datasets had incompatible formats and inconsistent column names and orders.

Solution

SQL queries were executed to convert incompatible columns and align all necessary columns. A new dataset was created as a union of all initial datasets.

Data Summaries and Exploration

Before analysis, a deep understanding of the data was required. Errors, null values, and inconsistencies had to be identified.

Solution

SQL queries were run to summarize each row, perform comparisons, correlations, and identify relationships among the variables.

Data Cleaning

Inconsistencies were found in categorical data (like user types), timings, and durations. Also, a specific station (HQ QR) was identified as an outlier.

Solution

SQL queries were used to normalize naming conventions and filter out erroneous data related to the HQ QR station.

Data Transformation

Many rows lacked crucial data and additional metrics were required for in-depth analysis.

Solution

New columns were created based on existing ones to capture additional metrics such as seasons, weekdays, holidays, and hour of the day.

Adding public holiday:

I added liststed public holidays from 2019-March-01 to 2020-March-01 in Chicago, to create a category for each ride in holiday_or column, using SQL during the Process phase

Data Validation

Gender and age data initially seemed important but had many null values.

Solution

SQL queries were run to calculate the rate of missing data and it was decided to ignore these columns.

Data Export

After exhaustive validation, preparation, investigation, cleaning, formatting, anonymization, and normalization, the final cleaned dataset was **exported to R for further analysis**.

Dataset Life cycle

Original : Divvy_Trips_2019_Q2 , Divvy_Trips-2019-Q3 , Divvy_Trips-2019-Q4 , Divvy_Trips-2020-Q1

Exported from SQL csv: Cleaned_Trips.csv

Renamed to and loaded to R : Final Trip Table.csv

After Analysis with R: rides.csv

SQL Queries:

A comprehensive list of all SQL queries used in this project is attached at the end of this document for transparency and reproducibility.

Comments on with each Query is added with the code.

- **Data Cleaning, Preparation, and Validation Documentation**

Data Type Query

The first query is designed to fetch information about the data types of the columns in each of the quarterly tables. This helps in understanding the schema and ensuring compatibility when performing joins or unions.

data type query:

```
SELECT column_name, data_type, 'Trips-2019-Q1' as table_name

FROM `myproject-391418.cyclistride.INFORMATION_SCHEMA.COLUMNS`

WHERE table_name='Trips-2019-Q1'

UNION ALL

SELECT column_name, data_type, 'Trips-2019-Q2' as table_name

FROM `myproject-391418.cyclistride.INFORMATION_SCHEMA.COLUMNS`

WHERE table_name='Trips-2019-Q2'

UNION ALL

SELECT column_name, data_type, 'Trips-2019-Q3' as table_name
```

```
FROM `myproject-391418.cyclistride.INFORMATION_SCHEMA.COLUMNS`

WHERE table_name='Trips-2019-Q3'

UNION ALL

SELECT column_name, data_type, 'Trips-2020-Q1' as table_name

FROM `myproject-391418.cyclistride.INFORMATION_SCHEMA.COLUMNS`

WHERE table_name='Trips-2020-Q1'
```

Joining Query

The main aim here is to create a comprehensive table that includes data from all the quarterly tables. Notably, data types for some of the columns like ride_id are cast to STRING to maintain uniformity.

Joining Query

```
--Joining datasets

--there is inconsistency of naming and organisation of the columns, rigerous checking is needed

--NOTE: After checking the types of data points (running Schema Info query), albeit there is no issue but the rental ID, not only it prevented the joining between all datasets, but rather usually IDs preferably be STRING, Another point, is that it seems there was an overhaul of naming ride IDs starting from 2020 so we will CAST their types to STRING.

CREATE TABLE `myproject-391418.cyclistride.Trips-perYear` AS

SELECT

  CAST(`_01___Rental_Details_Rental_ID` AS STRING) AS ride_id,

  `_01___Rental_Details_Local_Start_Time` AS started_at,

  `_01___Rental_Details_Local_End_Time` AS ended_at,

  `_01___Rental_Details_Bike_ID` AS bike_id,

  `_01___Rental_Details_Duration_In_Seconds_Uncapped` AS trip_duration,

  `_03___Rental_Start_Station_ID` AS start_station_id,

  `_03___Rental_Start_Station_Name` AS start_station_name,

  `_02___Rental_End_Station_ID` AS end_station_id,

  `_02___Rental_End_Station_Name` AS end_station_name,

  `User_Type` AS user_type,
  `Member_Gender` AS gender,

  `_05___Member_Details_Member_Birthday_Year` AS birthyear,

FROM `myproject-391418.cyclistride.Trips-2019-Q2`

UNION ALL

SELECT

  CAST(`trip_id` AS STRING) AS ride_id,

  `start_time` AS started_at,

  `end_time` AS ended_at,

  `bikeid` AS bike_id,

  `tripduration` AS trip_duration,

  `from_station_id` AS start_station_id,

  `from_station_name` AS start_station_name,

  `to_station_id` AS end_station_id,

  `to_station_name` AS end_station_name,

  `usertype` AS user_type,

  `gender` AS gender,

  `birthyear` AS birthyear,

FROM `myproject-391418.cyclistride.Trips-2019-Q3`

UNION ALL

SELECT

  CAST(`trip_id` AS STRING) AS ride_id,

  `start_time` AS started_at,

  `end_time` AS ended_at,

  `bikeid` AS bike_id,

  `tripduration` AS trip_duration,

  `from_station_id` AS start_station_id,

  `from_station_name` AS start_station_name,

  `to_station_id` AS end_station_id,
  `to_station_name` AS end_station_name,
  `usertype` AS user_type,
  `gender` AS gender,
  `birthyear` AS birthyear,
FROM `myproject-391418.cyclistride.Trips-2019-Q4`
UNION ALL
SELECT
  `ride_id` AS ride_id,
  `started_at` AS started_at,
  `ended_at` AS ended_at,
  NULL AS bike_id,
  TIMESTAMP_DIFF(`ended_at`, `started_at`, SECOND) AS trip_duration,
  `start_station_id` AS start_station_id,
  `start_station_name` AS start_station_name,
  `end_station_id` AS end_station_id,
  `end_station_name` AS end_station_name,
  `member_casual` AS user_type,
  NULL AS gender,
  NULL AS birthyear
FROM `myproject-391418.cyclistride.Trips-2020-Q1`;
```

data Information query

```
SELECT column_name, data_type AS table_name
FROM `myproject-391418.cyclistride.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name='Trips-perYear'
```

Missing Data Query

This query identifies the number of missing values in each column. It's crucial for assessing the quality of the dataset.

Missing data Query

```
--Checking how much missing data points

SELECT

    COUNT(*) AS total_rows,

    COUNTIF(`ride_id` IS NULL) AS missing_ride_id,

    COUNTIF(`started_at` IS NULL) AS missing_started_at,

    COUNTIF(`ended_at` IS NULL) AS missing_ended_at,

    COUNTIF(`bike_id` IS NULL) AS missing_bike_id,

    COUNTIF(`trip_duration` IS NULL) AS missing_trip_duration,

    COUNTIF(`start_station_id` IS NULL) AS missing_start_station_id,

    COUNTIF(`start_station_name` IS NULL) AS missing_start_station_name,

    COUNTIF(`end_station_id` IS NULL) AS missing_end_station_id,

    COUNTIF(`end_station_name` IS NULL) AS missing_end_station_name,

    COUNTIF(`user_type` IS NULL) AS missing_user_type,

    COUNTIF(`gender` IS NULL) AS missing_gender,

    COUNTIF(`birthyear` IS NULL) AS missing_birthyear

FROM `myproject-391418.cyclistride.Trips-perYear`;

--After investigating the number of missing data, it appears that there 1 missing station at start and end, we will investigate it separatly,

--As for the missing genders and birthyear, albeit they may be relavent to our analysis, and would offer us more insights to user behaviors, however this missing data represent ar und 25%, which is highly unreliabale at a first glance, However, we can speculate that those missing data are associated with casual users, in this case, a further inspection is needed,however we may exclude these data points totally.

--as for missing bike ids nearly a further investigation is needed.
```

Duplicates Query

The purpose is to find any duplicate ride_ids. A unique ride_id is crucial for the integrity of the dataset.

number of Duplicates of rider-ID

```
SELECT ride_id, COUNT(*) AS count

FROM `myproject-391418.cyclistride.Trips-perYear`

GROUP BY ride_id

HAVING count > 1;

--We conclude that there is no duplicate ride-ID

--However that doesnt mean that all rides are valid, we need to check other variables
```

Inconsistencies with Duration

This query aims to identify negative or illogical trip durations. Such data points are either errors or outliers and need to be handled appropriately.

number of inconsistencies with duration

```
--checking how many illogical or invalid trip durations data points, example negative durations,

--Also checking for 0 time interval

--Invertly we check for any error with trip start time and trip end time

-- Counting rows with negative or illogical durations

-- Counting rows with negative or illogical durations

SELECT 'negative_or_illogical_duration' AS issue_type, COUNT(*) AS count

FROM `myproject-391418.cyclistride.Trips-perYear`

WHERE trip_duration < 0

UNION ALL

-- Counting rows where end time is earlier than start time

SELECT 'end_time_before_start_time' AS issue_type, COUNT(*) AS count

FROM `myproject-391418.cyclistride.Trips-perYear`

WHERE ended_at < started_at

UNION ALL
```

```
-- Counting rows where trip duration is equal to 0

SELECT 'time_interval_equal_zero' AS issue_type, COUNT(*) AS count

FROM `myproject-391418.cyclistride.Trips-perYear`

WHERE trip_duration = 0 OR ended_at = started_at;

--we need to check the entire rows associated with these inconsistencies in a separate query
```

Investigating Issues Related to Trip Durations

This query digs deeper into the rows that have inconsistencies in their trip durations to provide a clear picture of the issue.

investigating issues related to trip durations

```
-- Query rows with time-related inconsistencies

SELECT *

FROM `myproject-391418.cyclistride.Trips-perYear`

WHERE trip_duration < 0

  OR ended_at < started_at

  OR trip_duration = 0

  OR ended_at = started_at;

--After we made our query, we can see that all negative and zero trip duration, start and end at "HQ QR" station name with 675 station ID, of casual user type,and a null bike ID

--Let's query all generated rides that are linked to this station before we conclude anything

--A clear vision is formulating here that this station may be irrelavent and these rides are related to the technical department, it can be maintenace team, or all together a technical error,

--This should be Reported to the stakeholders, and connect with the maitenance team to further investigate about the station 675.

--AS for the errors with start time that actually start after end time, it appears that the trip duration is valid, however this issue seems to be rare, we may conclude that its a minor error, since the duration is valid.
```

Investigating null bike ID

```
-- Query rows with NULL bike_id and excluding rows with the station name "HQ QR"

SELECT DISTINCT ride_id, *
```

```
FROM `myproject-391418.cyclistride.Trips-perYear`

WHERE bike_id IS NULL

AND (start_station_name != 'HQ QR' AND end_station_name != 'HQ QR');

--after checking the rows with null bike id data points, and corss referencing it with other errrors we can safely conclude that the rows woth null bike ID except the ones are associated with HQ QR, are safe to work with,

--now finally, we will move to checking the missing gender and birthyear relatively to the user type, to confirm if our theeory about mostly casual riders are least likely or if NONE have filled their personal info,
```

Inconsistencies

This query checks for inconsistencies in our data, which is crucial for analysis.

Stations and ID inconsistencies

```
-- Check for Inconsistencies in Starting Stations

SELECT 'start_station' AS station_type,

    CAST(start_station_id AS STRING) AS station_id,

    COUNT(DISTINCT start_station_name) AS distinct_names

FROM `myproject-391418.cyclistride.Trips-perYear`

GROUP BY start_station_id

HAVING distinct_names > 1

UNION ALL

-- Check for Inconsistencies in Ending Stations

SELECT 'end_station' AS station_type,

    CAST(end_station_id AS STRING) AS station_id,

    COUNT(DISTINCT end_station_name) AS distinct_names

FROM `myproject-391418.cyclistride.Trips-perYear`

GROUP BY end_station_id

HAVING distinct_names > 1

UNION ALL

-- Check for Starting Stations with No Name
```

```
SELECT 'start_station_no_name' AS station_type,

      CAST(start_station_id AS STRING) AS station_id,

      COUNT(*) AS distinct_names

FROM `myproject-391418.cyclistride.Trips-perYear`

WHERE start_station_name IS NULL

GROUP BY start_station_id

UNION ALL

-- Check for Ending Stations with No Name

SELECT 'end_station_no_name' AS station_type,

      CAST(end_station_id AS STRING) AS station_id,

      COUNT(*) AS distinct_names

FROM `myproject-391418.cyclistride.Trips-perYear`

WHERE end_station_name IS NULL

GROUP BY end_station_id

UNION ALL

-- Check for Starting Station Names with No ID

SELECT 'start_station_name_no_id' AS station_type,

      start_station_name AS station_name,

      COUNT(*) AS count

FROM `myproject-391418.cyclistride.Trips-perYear`

WHERE start_station_id IS NULL

GROUP BY start_station_name

UNION ALL

-- Check for Ending Station Names with No ID

SELECT 'end_station_name_no_id' AS station_type,

      end_station_name AS station_name,

      COUNT(*) AS count

FROM `myproject-391418.cyclistride.Trips-perYear`
```

```
WHERE end_station_id IS NULL

GROUP BY end_station_name;
```

--It appears that there are 50 station-ID are associated with 2 distinct station names and 1 end station with no name that have a null station ID and 1 station id have a null name, we will investigate each one of them closely, We can Conclude that the null station ID is also associated with the null station name.

Identifiying issues with station ID and missing names

```
-- Check for Inconsistencies in Station IDs

(SELECT 'start_station_id' AS Check_Type,

      CAST(start_station_id AS STRING) AS Value,

      COUNT(DISTINCT start_station_name) AS Count,

      STRING_AGG(DISTINCT start_station_name, ',') AS Names

 FROM `myproject-391418.cyclistride.Trips-perYear`

 GROUP BY start_station_id

 HAVING Count > 1)

UNION ALL

(SELECT 'end_station_id' AS Check_Type,

      CAST(end_station_id AS STRING) AS Value,

      COUNT(DISTINCT end_station_name) AS Count,

      STRING_AGG(DISTINCT end_station_name, ',') AS Names

 FROM `myproject-391418.cyclistride.Trips-perYear`

 GROUP BY end_station_id

 HAVING Count > 1)
```

--we conclude that the stations with 2 distinct station names are the same naming station but have a (temp) or(*) this may include that these stations in some period of times were moved or closed temporarly due to technical circumnstances or constructions.

--As for the stations with null name and id, this may be related to another type of data issues, we will come to that in the next query,

Investigating HQ QR station name

```
-- Query rows with the station name "HQ OR" in either the start_station_name or end_station_name

SELECT DISTINCT ride_id, *

FROM `myproject-391418.cyclistride.Trips-perYear`

WHERE start_station_name = 'HQ QR' OR end_station_name = 'HQ QR';

--In total there is 3768 ride associated with HQ QR station,

--Now we will investigate the null bike ID, excluding the ones associated with HQ QR.
```

Identifying issues with data

```
-- Check for Inconsistencies in Station IDs

(SELECT 'start_station_id' AS Check_Type,

      CAST(start_station_id AS STRING) AS Value,

      COUNT(DISTINCT start_station_name) AS Count,

      STRING_AGG(DISTINCT start_station_name, ',') AS Names

 FROM `myproject-391418.cyclistride.Trips-perYear`

 GROUP BY start_station_id

 HAVING Count > 1)

UNION ALL

(SELECT 'end_station_id' AS Check_Type,

      CAST(end_station_id AS STRING) AS Value,

      COUNT(DISTINCT end_station_name) AS Count,

      STRING_AGG(DISTINCT end_station_name, ',') AS Names

 FROM `myproject-391418.cyclistride.Trips-perYear`

 GROUP BY end_station_id

 HAVING Count > 1)

UNION ALL

-- Check for Unique User Types

(SELECT 'user_type' AS Check_Type,

      user_type AS Value,
       COUNT(*) AS Count,

       CAST(NULL AS STRING) AS Names

 FROM `myproject-391418.cyclistride.Trips-perYear`

 GROUP BY user_type)

UNION ALL

-- Check for Duplicate Ride IDs

(SELECT 'duplicate_ride_id' AS Check_Type,

       CAST(ride_id AS STRING) AS Value,

       COUNT(*) AS Count,

       CAST(NULL AS STRING) AS Names

 FROM `myproject-391418.cyclistride.Trips-perYear`

 GROUP BY ride_id

 HAVING Count > 1)

UNION ALL

-- Check for Negative or Zero Trip Durations

(SELECT 'inconsistent_trip_duration' AS Check_Type,

       CASE

         WHEN trip_duration <= 0 THEN 'negative_or_zero'

         WHEN ended_at <= started_at THEN 'invalid_time'

         ELSE 'other'

       END AS Value,

       COUNT(*) AS Count,

       CAST(NULL AS STRING) AS Names

 FROM `myproject-391418.cyclistride.Trips-perYear`

 WHERE trip_duration <= 0 OR ended_at <= started_at

 GROUP BY Value);
```

User Type Inconsistencies

This query reveals that there are more than two types of users, which is not expected. It indicates a need for data normalization.

Identifying user type inconcistencies

```
--The user-type data is Discrete and catagorical type of data, we expect to be only 2, "casual" or "member", so we need to check if there is more than that and what are they,

SELECT user_type, COUNT(*) AS count

FROM `myproject-391418.cyclistride.Trips-perYear`

GROUP BY user_type;

--after inspection, we got 4 in total,in this case we need to inspect each pre-joining of the 4 datasets

--After inspecting each Qurater year dataset (4 in total) it appears that there is an inconsistency in naming convention for the Q2 of 2019 dataset,

--In the 2019-Q2 dataset, there is 2 types "customer" and "subscriber", IN THIS CASE, WE NEED TO INVESTIGATE THIS FURTHER WITH THE STAKEHOLDER,

--After investigation and confirming that "customer" is the "Casual" users type, and "Subscriber" is "member", Normalization required
```

Counting Members and Non-Members

This query checks the count of different user types to ensure that the data is balanced and to confirm the effectiveness of previous cleaning operations.

Counting members and non members, and verifying if our count is correct:

```
-- Count non-members excluding 'HQ QR'

SELECT 'non_members' AS description, COUNT(*) AS value

FROM `myproject-391418.cyclistride.Trips-perYear`

WHERE (user_type = 'casual' OR user_type = 'Customer') AND (start_station_name != 'HQ QR' AND end_station_name != 'HQ QR')

UNION ALL

-- Count members excluding 'HQ QR'

SELECT 'members' AS description, COUNT(*) AS value
```

```
FROM `myproject-391418.cyclistride.Trips-perYear`

WHERE (user_type = 'member' OR user_type = 'Subscriber') AND (start_station_name != 'HQ QR' AND end_station_name != 'HQ QR')

UNION ALL

-- Count total rows in the entire dataset

SELECT 'total_rows_in_entire_dataset' AS description, COUNT(*) AS value

FROM `myproject-391418.cyclistride.Trips-perYear`

UNION ALL

-- Count total rows excluding 'HQ QR'

SELECT 'total_rows_excluding_HQ_QR' AS description, COUNT(*) AS value

FROM `myproject-391418.cyclistride.Trips-perYear`

WHERE start_station_name != 'HQ QR' AND end_station_name != 'HQ QR'

UNION ALL

-- Summation of members and non-members excluding 'HQ QR'

SELECT 'sum_members_and_non_members' AS description, COUNT(*) AS value

FROM `myproject-391418.cyclistride.Trips-perYear`

WHERE (user_type IN ('casual', 'Customer', 'member', 'Subscriber')) AND (start_station_name != 'HQ QR' AND end_station_name != 'HQ QR');
```

Creating Final Table

The final table, Cleaned_Trips, is a cleaned and normalized version of the original data. New columns like is_weekend and is_holiday have been added to enrich the dataset for further analysis.

Descriptive Query

This query is for generating basic statistics like minimum, maximum, average, and standard deviation for ride lengths, segmented by user type.

creating final table

```
-- Create the cleaned table with weekend and holiday flags
-- We conducted User Type Standardization or Normalization

-- Date and Time Extraction

-- Weekend/Weekday Classification

-- Data Cleaning

-- Now this table is ready for an analysis

CREATE TABLE `myproject-391418.cyclistride.Cleaned_Trips` AS

SELECT

 ride_id,

 CASE

  WHEN user_type = 'Subscriber' THEN 'member'

  WHEN user_type = 'Customer' THEN 'casual'

  ELSE user_type

 END AS user_type,

 start_station_id,

 end_station_id,

 trip_duration,

 CAST(started_at AS DATE) AS start_date,

 EXTRACT(TIME FROM started_at) AS start_time,

 CAST(ended_at AS DATE) AS end_date,

 EXTRACT(TIME FROM ended_at) AS end_time,

 CASE

  WHEN EXTRACT(DAYOFWEEK FROM started_at) IN (1, 7) THEN 'weekend'

  ELSE 'weekday'

 END AS is_weekend,

 CASE

  WHEN CAST(started_at AS DATE) IN ('2019-07-04', '2019-09-02', '2019-10-14', '2019-11-11', '2019-11-28', '2019-12-25', '2020-
01-01', '2020-01-20', '2020-02-17') THEN 'holiday'

  ELSE 'not_holiday'
 END AS is_holiday
FROM
 `myproject-391418.cyclistride.Trips-perYear`
WHERE
 start_station_name != 'HQ QR' AND end_station_name != 'HQ QR';
```

Descriptive Query

```
-- Descriptive statistics for ride lengths
SELECT
  user_type,
  COUNT(*) AS total_rides,
  MIN(trip_duration) AS min_duration,
  MAX(trip_duration) AS max_duration,
  AVG(trip_duration) AS avg_duration,
  STDDEV(trip_duration) AS stddev_duration
FROM
  `myproject-391418.cyclistride.Cleaned_Trips`
GROUP BY
  user_type
UNION ALL
-- Totals for all users
SELECT
  'total' AS user_type,
  COUNT(*) AS total_rides,
  MIN(trip_duration) AS min_duration,
  MAX(trip_duration) AS max_duration,
  AVG(trip_duration) AS avg_duration,
  STDDEV(trip_duration) AS stddev_duration
FROM
  `myproject-391418.cyclistride.Cleaned_Trips`;
```

Conclusions and Recommendations

The journey from inconsistent, massive raw datasets to a well-structured, cleaned, and enriched dataset was exhaustive but rewarding. It is recommended to continuously validate and update this dataset to adapt to new data structures and ensure its reliability for future analyses.