1. **Data source**

The data comes from a reliable source - the Divvy system, operated by Lyft - and describes the bike rides of users of the city's bike-sharing app. Divvy is owned by the Chicago Department of Transportation, an entity affiliated with the city.

The data is shared under an open license - Lyft has permission from the Department of Transportation to share the data.

I decided to use the data, which covers one year from January to December 2022 and is stored in 12 CSV files, one for each month.

Link to the data

2. **Stages of the project**

I started the project by preprocessing the data, analyzing it, and then visualizing it. Finally, there is a section where I created some machine learning models to impute missing values in two of the columns.

3. **Tools used**

I used SQL in BigQuery to preprocess and analyze the data, while I utilized Python in Google Colab notebook to create visualizations and impute missing data. In the part of the project where I use Python, I established a connection to the data source, located in the Google Cloud Storage bucket containing the aforementioned collection.

4. **The steps I took to ensure that the data is ready for analysis**

I checked the data for missing values, consistency, duplicates, and removed unnecessary information.

All data preprocessing steps are described in this document.

Data description

The dataset consists of 5.667 million rows and contains the following information for each registered ride:

ride_id - Unique identifier of the ride in alphanumeric format with a length of 16 characters.

rideable_type - The type of bike rented, which is a categorical variable, with possible values: electric_bike, classic_bike or docked_bike

started_at, ended_at - The start and end date and time of each ride, formatted in datetime format and UTC time zone

start_station_name, start_station_id, end_station_name, end_station_id - Names and identifiers of start and end stations, represented as categorical variables. Station names are represented as their literal names, while station identifiers include a set of values that do not have a uniform format across all data rows

start_lat, start_lng, end_lat, end_lng - The geographic coordinates of the start and end point of each trip, represented as floating point numbers

member_casual - A categorical variable indicating whether the bicycle was rented by an occasional user or a subscriber to the service

Data preprocessing

I created a table containing data for the entire year 2022 from twelve individual files (2022_01, 2022_02, etc.).

```
CREATE TABLE hazel-math-393216.cyclistic_2022.data_2022 AS
SELECT *
FROM (
    SELECT * FROM hazel-math-393216.cyclistic_2022.2022_01
    UNION ALL
    SELECT * FROM hazel-math-393216.cyclistic_2022.2022_02
    UNION ALL
    SELECT * FROM hazel-math-393216.cyclistic_2022.2022_03
    UNION ALL
    SELECT * FROM hazel-math-393216.cyclistic_2022.2022_04
    UNION ALL
    SELECT * FROM hazel-math-393216.cyclistic_2022.2022_05
    UNION ALL
    SELECT * FROM hazel-math-393216.cyclistic_2022.2022_06
    UNION ALL
    SELECT * FROM hazel-math-393216.cyclistic_2022.2022_07
    UNION ALL
    SELECT * FROM hazel-math-393216.cyclistic_2022.2022_08
    UNION ALL
    SELECT * FROM hazel-math-393216.cyclistic_2022.2022_09
    UNION ALL
    SELECT * FROM hazel-math-393216.cyclistic_2022.2022_10
    UNION ALL
    SELECT * FROM hazel-math-393216.cyclistic_2022.2022_11
    UNION ALL
    SELECT * FROM hazel-math-393216.cyclistic_2022.2022_12
);
```

The created dataset consists of 5667717 rows.

I checked if the data includes entries for 2022 only.

```
SELECT
  MAX(started_at) AS max_started_at,
  MAX(ended_at) AS  max_ended_at,
  MIN(started_at) AS min_started_at,
  MIN(ended_at) AS min_ended_at

FROM
  `hazel-math-393216.cyclistic_2022.data_2022`
```

Result:

Row	max_started_at	max_ended_at	min_started_at	min_ended_at
1	2022-12-31 23:59:26 UTC	2023-01-02 04:56:45 UTC	2022-01-01 00:00:05 UTC	2022-01-01 00:01:48 UTC

It can be noticed that *max_ended_at* has a value for 2023, presumably someone rented a bike before the end of the year and returned it on the second of January, but overall, the result indicates that the dataset contains rides for 2022.

Since the columns corresponding to the station id are redundant for this analysis, I removed them. Because this table is not in a relationship with any other table, I also removed the primary key, the *ride_id* column, which is the unique identifier of each ride.

```
ALTER TABLE `hazel-math-393216.cyclistic_2022.data_2022`
DROP COLUMN ride_id,
DROP COLUMN start_station_id,
DROP COLUMN end_station_id;
```

Using the following query, I checked the unique values of the categorical columns.

```
SELECT DISTINCT rideable_type
FROM `hazel-math-393216.cyclistic_2022.data_2022`;
```

Result:

Row	rideable_type
1	electric_bike
2	classic_bike
3	docked_bike

The *rideable_type* column, which stores information about the type of bicycle rented, contains three unique values.

The unique values of the *member_casual* column:

```
SELECT DISTINCT member_casual
FROM `hazel-math-393216.cyclistic_2022.data_2022`;
```

Result:

Row	member_casual
1	member
2	casual

The column specifying the type of user, contains two unique values.

The *start_station_name* column contains 1675 unique station names, while the *end_station_name* column contains 1693 names:

```
SELECT DISTINCT start_station_name
FROM `hazel-math-393216.cyclistic_2022.data_2022`;

SELECT DISTINCT end_station_name
FROM `hazel-math-393216.cyclistic_2022.data_2022`;
```

I then checked to see if there were rows with missing values in the set.

```
SELECT
  COUNT(*) AS number_of_missing_values
FROM
  `hazel-math-393216.cyclistic_2022.data_2022`
WHERE
      rideable_type IS NULL OR
      started_at IS NULL OR
      ended_at IS NULL OR
      start_station_name IS NULL OR
      end_station_name IS NULL OR
      start_lat IS NULL OR
      start_lng IS NULL OR
      end_lat IS NULL OR
      end_lng IS NULL OR
      member_casual IS NULL;
```

Result:

Row	number_of_missing_
1	1298357

The number of rows with missing values is significant, as it is about 20-25% of all rows.

I decided to use a function that counts the missing values for each column to determine in which ones they occur.

```
SELECT
  COUNTIF(rideable_type IS NULL) AS rideable_type_missing,
  COUNTIF(started_at IS NULL) AS started_at_missing,
  COUNTIF(ended_at IS NULL) AS ended_at_missing,
  COUNTIF(start_station_name IS NULL) AS start_station_name_missing,
  COUNTIF(end_station_name IS NULL) AS end_station_name_missing,
  COUNTIF(start_lat IS NULL) AS start_lat_missing,
  COUNTIF(start_lng IS NULL) AS start_lng_missing,
  COUNTIF(end_lat IS NULL) AS end_lat_missing,
  COUNTIF(end_lng IS NULL) AS end_lng_missing,
  COUNTIF(member_casual IS NULL) AS member_casual_missing
```

```
FROM
  `hazel-math-393216.cyclistic_2022.data_2022`;
```

Result:

Row	rideable_type_missin	started_at_missing	ended_at_missing	start_station_name_	end_station_name_m	start_lat_missing	start_lng_missing	end_lat_missing	end_lng_missing	member_casual_mis
1	0	0	0	833064	892742	0	0	5858	5858	0

The missing values appear almost exclusively in the columns with the names of the start and end stations of the trips.

I will try later to perform imputation of the start and end station names using machine learning models (the coordinates are stored to 6-8 decimal places, so with high precision), so I kept all the rows with missing entries in the columns with station names.

Since the columns with coordinates are missing values for a small number of rows, I removed them.

```
DELETE FROM `hazel-math-393216.cyclistic_2022.data_2022`
WHERE end_lat IS NULL OR end_lng IS NULL;
```

I used the `SELECT DISTINCT *` statement to check for duplicates in the dataset. The result displayed the same number of rows as the `SELECT *` statement, leading to the conclusion that there are no duplicates in the dataset.

I then conducted a consistency analysis of the dataset.

I started by checking whether the columns storing station coordinate information contain data that is outside the range in which such information is given (-90 to 90 degrees for latitude and -180 to 180 degrees for longitude).

```
SELECT *
FROM `hazel-math-393216.cyclistic_2022.data_2022`
WHERE start_lat < -90 OR start_lat > 90;

SELECT *
FROM `hazel-math-393216.cyclistic_2022.data_2022`
```

```
WHERE start_lng < -180 OR start_lng > 180;

SELECT *
FROM `hazel-math-393216.cyclistic_2022.data_2022`
WHERE end_lat < -90 OR end_lat > 90;

SELECT *
FROM `hazel-math-393216.cyclistic_2022.data_2022`
WHERE end_lng < -180 OR end_lng > 180;
```

None of the queries returned a result, so this means that there are no such inconsistencies in the dataset.

I checked to see if there are rows where the start date and time are later than or equal to the end date.

```
SELECT *
FROM `hazel-math-393216.cyclistic_2022.data_2022`
WHERE started_at >= ended_at;
```

Result:

Row	ride_id	rideable_type	started_at	ended_at	st
1	6A871510E302E8CD	electric_bike	2022-06-23 19:22:57 UTC	2022-06-23 19:21:46 UTC	n
2	AF62B881CEBBC69F	electric_bike	2022-11-06 01:51:53 UTC	2022-11-06 01:18:03 UTC	n
3	F017ABAD58857045	electric_bike	2022-06-07 19:13:27 UTC	2022-06-07 17:07:57 UTC	n
4	8C81C06131425300	classic_bike	2022-06-23 08:49:45 UTC	2022-06-23 08:49:45 UTC	P
5	B36B146DFBF98441	electric_bike	2022-09-09 10:40:41 UTC	2022-09-09 10:40:41 UTC	G
6	85330FA7C427181F	electric_bike	2022-08-02 15:39:09 UTC	2022-08-02 15:39:09 UTC	K

Results per page: 50 1 – 50 of 531

There are more than 500 of such rows, I removed them all:

```
DELETE FROM `hazel-math-393216.cyclistic_2022.data_2022`
WHERE started_at >= ended_at;
```

I created a new column containing the length of each trip in minutes.

```
CREATE OR REPLACE TABLE `hazel-math-393216.cyclistic_2022.data_2022` AS
SELECT
 *,
 TIMESTAMP_DIFF(ended_at, started_at, SECOND) / 60.0 AS ride_length_minutes
FROM
 `hazel-math-393216.cyclistic_2022.data_2022`;
```

I created a *day_of_week column* that contained information about the day of the week on which a particular ride took place (1 for Sunday, 2 for Monday, etc.).

```
ALTER TABLE `hazel-math-393216.cyclistic_2022.data_2022`
ADD COLUMN day_of_week INT64;

UPDATE `hazel-math-393216.cyclistic_2022.data_2022`
SET day_of_week = EXTRACT(DAYOFWEEK FROM started_at)
WHERE started_at IS NOT NULL;
```

I assigned to the numbers corresponding to the days of the week their proper names in the *day_of_week_name* column:

```
ALTER TABLE `hazel-math-393216.cyclistic_2022.data_2022`
ADD COLUMN day_of_week_name STRING;

UPDATE `hazel-math-393216.cyclistic_2022.data_2022`
SET day_of_week_name =
CASE
  WHEN day_of_week = 1 THEN 'Sunday'
  WHEN day_of_week = 2 THEN 'Monday'
  WHEN day_of_week = 3 THEN 'Tuesday'
  WHEN day_of_week = 4 THEN 'Wednesday'
  WHEN day_of_week = 5 THEN 'Thursday'
  WHEN day_of_week = 6 THEN 'Friday'
  WHEN day_of_week = 7 THEN 'Saturday'
  ELSE NULL
END
WHERE day_of_week IS NOT NULL;
```

I removed the *day_of_week_name* column, and assigned the values from it to the *day_of_week* column, to shorten the column name.

```
ALTER TABLE `hazel-math-393216.cyclistic_2022.data_2022`
DROP COLUMN day_of_week;

ALTER TABLE `hazel-math-393216.cyclistic_2022.data_2022`
ADD COLUMN day_of_week STRING;

UPDATE `hazel-math-393216.cyclistic_2022.data_2022`
SET day_of_week = day_of_week_name
WHERE day_of_week_name IS NOT NULL;

ALTER TABLE `hazel-math-393216.cyclistic_2022.data_2022`
DROP COLUMN day_of_week_name;
```

Data analysis

I started by checking the average length of the trip.

```
SELECT
 AVG(ride_length_minutes) AS average_ride_length_minutes
FROM
 `hazel-math-393216.cyclistic_2022.data_2022`
```

Result:

Row	average_ride_length_
1	16.33139904041...

The average ride time is above 16 minutes.

Shortest and longest travel time:

```
SELECT
 MAX(ride_length_minutes) AS max_ride_length,
 MIN(ride_length_minutes) AS min_ride_length
FROM
 `hazel-math-393216.cyclistic_2022.data_2022`;
```

Result:

Row	max_ride_length	min_ride_length
1	34354.06666666...	0.016666666666...

The maximum result is clearly an outlier; the minimum result is also extremely low.

Since the `APPROX_QUANTILES` function returns imprecise results, I decided to calculate the positions of the 1st and 99th percentiles by multiplying the number of rows in the dataset by 1% and 99%. Then, I checked the values at these positions.

```
SELECT ride_length_minutes
FROM `hazel-math-393216.cyclistic_2022.data_2022`
ORDER BY ride_length_minutes ASC
LIMIT 1 OFFSET 56671;
```

Result:

Row	ride_length_minutes
1	0.35

The value of the 1st percentile is 0.35 minutes, which is approximately 20 seconds.

99th percentile:

```
SELECT ride_length_minutes
FROM `hazel-math-393216.cyclistic_2022.data_2022`
ORDER BY ride_length_minutes ASC
LIMIT 1 OFFSET 5610513;
```

Result:

Row	ride_length_minutes
1	107.15

For the 99th percentile, the travel time is over 107 minutes.

I removed rows with travel times less than one minute and longer than the value of the 99th percentile.

```
DELETE FROM `hazel-math-393216.cyclistic_2022.data_2022`
WHERE ride_length_minutes < 1 OR ride_length_minutes > (107.15)
```

Result:

This statement removed 171,363 rows from data_2022.

Over 170,000 extreme values were removed from the dataset.

I checked the average travel time after removing the extreme values.

```
SELECT
 AVG(ride_length_minutes) AS average_ride_length_minutes
FROM
 `hazel-math-393216.cyclistic_2022.data_2022`
```

Result:

Row	average_ride_length_
1	14.84208054453...

The average travel time has shortened to 14.8 minutes.

Next, I checked the average travel time for each of the two user types separately, grouping the results by the *member_casual* column.

```
SELECT
 member_casual,
 AVG(ride_length_minutes) AS ride_length_minutes
FROM
 `hazel-math-393216.cyclistic_2022.data_2022`
GROUP BY
 Member_casual;
```

Result:

Row	member_casual	ride_length_minutes
1	casual	18.83304787015...
2	member	12.12188645729...

The average rental time for casual users is longer, nearly 19 minutes, compared to subscribers, for whom the average travel time is just over 12 minutes.

I checked the total number of rides for each day of the week.

```
SELECT
 day_of_week,
 COUNT(*) AS dow_ride_count
FROM
 `hazel-math-393216.cyclistic_2022.data_2022`
GROUP BY day_of_week
ORDER BY
 CASE day_of_week
   WHEN 'Monday' THEN 1
   WHEN 'Tuesday' THEN 2
   WHEN 'Wednesday' THEN 3
   WHEN 'Thursday' THEN 4
   WHEN 'Friday' THEN 5
   WHEN 'Saturday' THEN 6
   WHEN 'Sunday' THEN 7
 END;
```

Result:

Row	day_of_week	dow_ride_count
1	Monday	727964
2	Tuesday	760231
3	Wednesday	776103
4	Thursday	817457
5	Friday	777409
6	Saturday	883185
7	Sunday	747616

The values do not vary significantly for individual days, with the highest count for Saturday and the lowest for Monday.

Number of rides for each day of the week based on user type:

```
SELECT
  day_of_week,
  SUM(CASE WHEN member_casual = 'member' THEN 1 ELSE 0 END) AS member_ride_count,
  SUM(CASE WHEN member_casual = 'casual' THEN 1 ELSE 0 END) AS casual_ride_count
FROM
  `hazel-math-393216.cyclistic_2022.data_2022`
GROUP BY
  day_of_week
ORDER BY
 CASE day_of_week
   WHEN 'Monday' THEN 1
   WHEN 'Tuesday' THEN 2
   WHEN 'Wednesday' THEN 3
   WHEN 'Thursday' THEN 4
   WHEN 'Friday' THEN 5
   WHEN 'Saturday' THEN 6
   WHEN 'Sunday' THEN 7
 END;
```

Result:

Row	day_of_week	member_ride_count	casual_ride_count
1	Monday	462278	265686
2	Tuesday	506690	253541
3	Wednesday	511680	264423
4	Thursday	519642	297815
5	Friday	455794	321615
6	Saturday	431609	451576
7	Sunday	377058	370558

Here, a certain trend is evident - casual users ride more on weekends, while subscribers use the service more from Monday to Friday.

The percentage distribution of rides by user type:

```
SELECT
  member_casual,
  COUNT(*) AS ride_count,
  ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM
`hazel-math-393216.cyclistic_2022.data_2022`), 2) AS percentage
FROM
  `hazel-math-393216.cyclistic_2022.data_2022`
GROUP BY
  member_casual;
```

Result:

Row	member_casual	ride_count	percentage
1	member	3264751	59.47
2	casual	2225214	40.53

Approximately 60% of rides are taken by subscribers, while the remaining portion is attributed to casual users.

Number of rides and the percentage of all rides for each bike type:

```
SELECT
  rideable_type,
  COUNT(*) AS ride_count,
  ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM
`hazel-math-393216.cyclistic_2022.data_2022`), 2) AS percentage
FROM
  `hazel-math-393216.cyclistic_2022.data_2022`
GROUP BY
  rideable_type;
```

Result:

Row	rideable_type	ride_count	percentage
1	electric_bike	2797418	50.96
2	docked_bike	156937	2.86
3	classic_bike	2535610	46.19

Number of rides and the percentage of all rides for each bike type and user type:

```
SELECT
  member_casual,
  rideable_type,
  COUNT(*) AS ride_count,
  ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM
`hazel-math-393216.cyclistic_2022.data_2022`), 2) AS percentage
FROM
 `hazel-math-393216.cyclistic_2022.data_2022`
GROUP BY
 member_casual,
 rideable_type
ORDER BY
 member_casual;
```

Result:

Row	member_casual	rideable_type	ride_count	percentage
1	casual	electric_bike	1212645	22.09
2	casual	docked_bike	156937	2.86
3	casual	classic_bike	855632	15.59
4	member	electric_bike	1584773	28.87
5	member	classic_bike	1679978	30.6

I checked the top 10 most popular starting and ending stations.

```
SELECT
 start_station_name,
 COUNT(*) AS ride_count
FROM
 `hazel-math-393216.cyclistic_2022.data_2022`
WHERE
 start_station_name IS NOT NULL
GROUP BY
 start_station_name
ORDER BY
 ride_count DESC
LIMIT 10;
```

Result:

Row	start_station_name	ride_count
1	Streeter Dr & Grand Ave	71306
2	DuSable Lake Shore Dr & Monr...	39297
3	DuSable Lake Shore Dr & North...	38501
4	Michigan Ave & Oak St	37953
5	Wells St & Concord Ln	36745
6	Clark St & Elm St	34239
7	Kingsbury St & Kinzie St	33046
8	Millennium Park	32752
9	Theater on the Lake	31913
10	Wells St & Elm St	30815

Top 10 most popular ending stations:

```
SELECT
 end_station_name,
 COUNT(*) AS ride_count
FROM
 `hazel-math-393216.cyclistic_2022.data_2022`
WHERE
 end_station_name IS NOT NULL
GROUP BY
 end_station_name
ORDER BY
 ride_count DESC
LIMIT 10
```

Result:

Row	end_station_name	ride_count
1	Streeter Dr & Grand Ave	71545
2	DuSable Lake Shore Dr & North...	40729
3	Michigan Ave & Oak St	38332
4	DuSable Lake Shore Dr & Monr...	38213
5	Wells St & Concord Ln	36752
6	Clark St & Elm St	33754
7	Millennium Park	33372
8	Theater on the Lake	31888
9	Kingsbury St & Kinzie St	31792
10	Wells St & Elm St	29786