POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Systems and Methods for Big and Unstructured Data Project

Author: **Laura Colazzo**

Person Code: **10705522**

Academic Year: 2023-2024

Contents

1 | Introduction

This chapter presents an introduction to the project work delivered for the Systems and Methods for Big and Unstructured Data course at Politecnico di Milano.

1.1. Objective

The project specifications required searching for a dataset publicly available online to be analyzed using a NoSQL technology, among the ones studied in the course. The set of technologies that were comprehensively covered from a practical point of view includes Neo4j, MongoDB, and Elasticsearch. Therefore, the objective was to put into practice the knowledge acquired during the course on one of these Database Management Systems (DBMSs), employing its corresponding querying language to extract valuable insights from the gathered data. More specifically, the request was to formulate and execute ten of these queries.

1.2. Dataset Overview

The dataset selected for the analysis belongs to a bike-sharing company based in Chicago and was found on Kaggle at this link. The web page contains a reference to a more complete source, where data updated to November 2023 about rides performed by users of the bike-sharing system is available, divided by month.

The analysis focuses on the last quarter of 2022, to generate useful insights on the bike-renting operations registered in the last part of the year.

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
A50255C1E17942AB	classic_bike	2022-10-14 17:13:30	2022-10-14 17:19:39	Noble St & Milwaukee Ave	13290	Larrabee St & Division St	KA1504000079	41.900680	-87.662600	41.903486	-87.643353	member
DB692A70BD2DD4E3	electric_bike	2022-10-01 16:29:26	2022-10-01 16:49:06	Damen Ave & Charleston St	13288	Damen Ave & Cullerton St	13089	41.920037	-87.677937	41.854967	-87.675700	casual
3C02727AAF60F873	electric_bike	2022-10-19 18:55:40	2022-10-19 19:03:30	Hoyne Ave & Balmoral Ave	655	Western Ave & Leland Ave	TA1307000140	41.979879	-87.681902	41.966400	-87.688704	member
47E653FDC2D99236	electric_bike	2022-10-31 07:52:36	2022-10-31 07:58:49	Rush St & Cedar St	KA1504000133	Orleans St & Chestnut St (NEXT Apts)	620	41.902274	-87.627692	41.898203	-87.637536	member
8B5407BE535159BF	classic_bike	2022-10-13 18:41:03	2022-10-13 19:26:18	900 W Harrison St	13028	Adler Planetarium	13431	41.874754	-87.649807	41.866095	-87.607267	casual

Figure 1.1: First five rows of the dataset

Each row of the dataset includes information about a specific ride, providing its ID, the date and time of its start and end, and the details of the start and end stations, together with their geospatial coordinates. Moreover, information about the type of bike used and the type of user who initiated the ride is reported. A small portion of the dataset rows can be observed in Figure 1.1. In the following sections, we will discuss the transformations this dataset underwent, the NoSQL technology adopted, and we will describe its non-relational schema.

1.3. Methodology

In approaching the dataset analysis, an organizational perspective was adopted, aiming to capture pertinent information to support decision-making inside the company. Indeed, data exploration focused on key aspects of the business, including the differences in the utilization of the service by the different categories of users, the change in utilization of the service on special days like Christmas, Thanksgiving, New Year's Eve, or Black Friday, compared to the ordinary days. Additionally, insights on the normal usage patterns of the system were gathered to get a comprehensive understanding of the business under analysis and to possibly identify some anomalies in the service.

The results of the study could suggest new strategic paths to be explored by the bike-sharing company to increase business performance.

The project started by conducting an in-depth examination of the dataset's features and its structure. This preliminary phase was crucial to identify interesting aspects to be further investigated in the subsequent analysis phases. Once succeeded in this task, queries were extensively formulated in natural language and, only once having cleaned the dataset, queries were translated into the designated query language.

1.4. DBMS Technology

The exploratory analysis of the dataset is conducted in MongoDB.

1.4.1. MongoDB Overview

MongoDB is an open-source, document-oriented database system, that stores data in the form of JSON-like documents. Its distributed architecture is designed to be fast, scalable, and performant when managing large volumes of data. Moreover, the technology is particularly suitable for handling web data due to its compatibility with JSON format.

One of the key features of MongoDB is the fact that it allows flexibility in document structures, allowing adaptability to evolving data needs. Additionally, it provides robust support for aggregation pipelines and geospatial data management, enabling complex data manipulations. Also, being MongoDB a documental database, retrieving data is extremely efficient, since it does not involve costly join operations, like in relational databases.

1.4.2. Technology Selection Criteria

The rationale behind the choice of MongoDB as DBMS for the exploratory analysis of the bike-sharing dataset lies in the need to perform a historical aggregated analysis of data about rides done with the bike-sharing system, but also in some properties of the dataset itself.

First of all, being the initial number of samples in the dataset more than 1 million, we can reasonably assume that the company needs to efficiently deal with large volumes of data, also considering that this is just a very small portion of the amount of data actually available for analytical purposes. Also, considering that the volume of data increases every month, a high-scalable DBMS technology like MongoDB is necessary to store and manage this data effectively.

```
RangeIndex: 1078226 entries, 0 to 1078225
Data columns (total 13 columns):
 #   Column              Non-Null Count    Dtype
---  ------              --------------    -----
 0   ride_id             1078226 non-null  object
 1   rideable_type       1078226 non-null  object
 2   started_at          1078226 non-null  object
 3   ended_at            1078226 non-null  object
 4   start_station_name  905631 non-null   object
 5   start_station_id    905631 non-null   object
 6   end_station_name    896192 non-null   object
 7   end_station_id      896192 non-null   object
 8   start_lat           1078226 non-null  float64
 9   start_lng           1078226 non-null  float64
 10  end_lat             1077393 non-null  float64
 11  end_lng             1077393 non-null  float64
 12  member_casual       1078226 non-null  object
dtypes: float64(4), object(9)
memory usage: 106.9+ MB
```

Figure 1.2: Description of the dataset's structure

Moreover, by looking at the structure of the dataset (Figure 1.2) one may notice that it perfectly resembles the result of a join between several tables of a relational database. This suggests that if the data was analyzed using relational technology, joins would have been needed each time a query was formulated to achieve the data granularity required for extracting valuable pieces of information. By leveraging a documental database, instead, we can design the data model such that it is already a stand-alone document, with all

the information needed to perform the query. This makes querying very efficient on such a large data source.

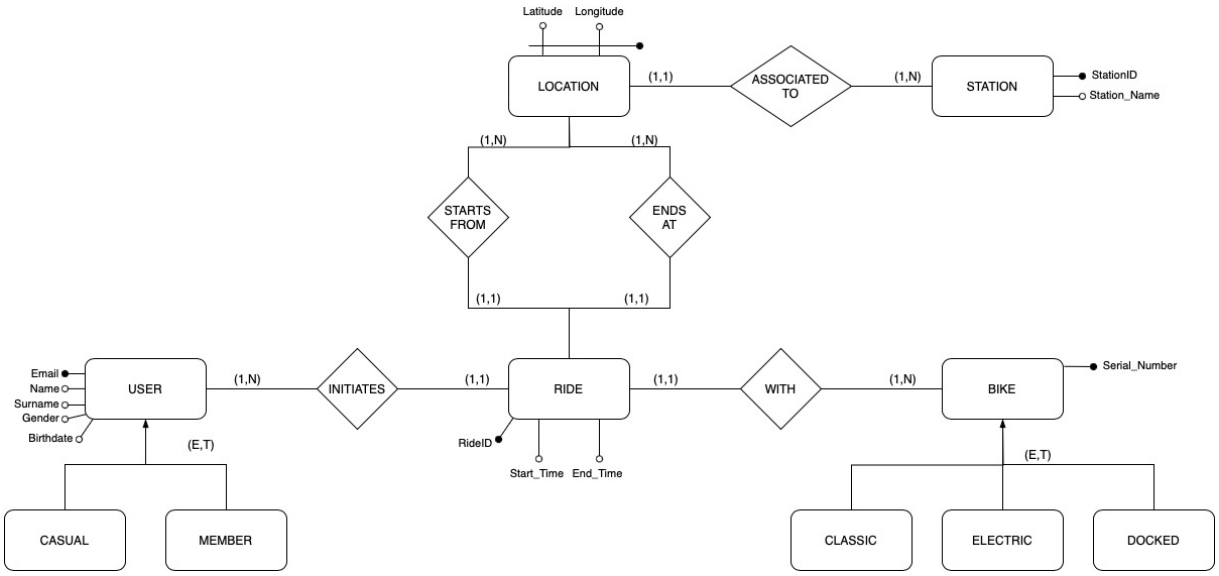

Figure 1.3: Inferred ER model

To better understand this concept, it might be useful to look at the ER model derived from the dataset under analysis (Figure 1.3). The ER reconstruction was performed inferring some pieces of information that were missing in the fields of the dataset, but which we can reasonably assume the bike-sharing system gathers for business purposes. An example is given by sensitive attributes of the User entity, which were presumably truncated due to privacy concerns. Another example is given by the Bike entity, which is missing the serial number. Despite their absence in the dataset, these details were included in the ER model to ensure a comprehensive representation. It is to underline the fact that rides start and end at a specific location, identified by its geospatial coordinates. These coordinates are associated with a single station, which may be assigned to multiple latitude and longitude coordinates of locations nearby. This property of the dataset was taken into account when building the ER model.

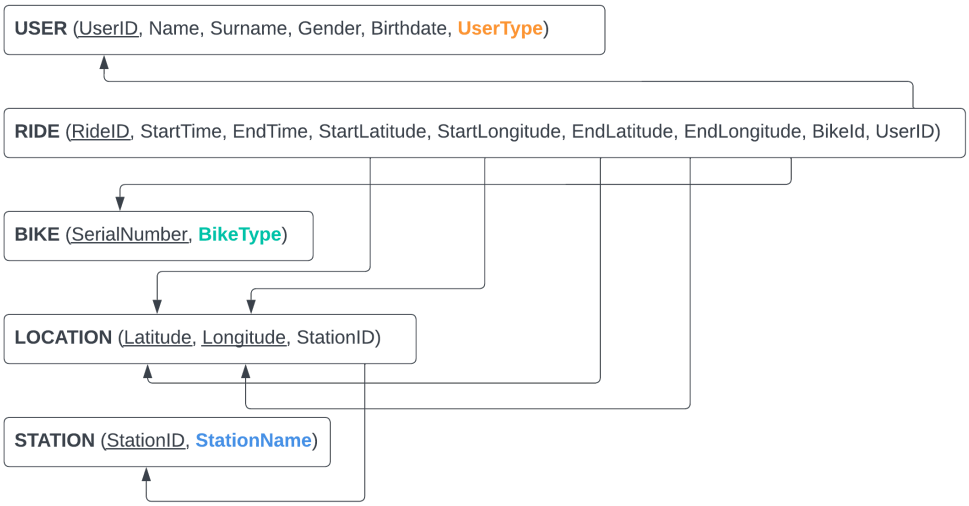

Figure 1.4: Inferred logical schema

It might also be interesting to translate the conceptual model, described by the ER diagram, into its logical counterpart to examine the structure of the tables derived from this translation. Figure 1.4 highlights in different colors the attributes that are part of the dataset under analysis. This is to show that they actually reside in tables different from the Ride one, requiring join operations operations between the Ride table and others to extract valuable data, like the type of bike adopted, the station name, or the user category.

The other reason for considering MongoDB as DBMS technology, as previously mentioned, comes from the need to perform historical analysis over the data. Given that this operation often involves aggregating data over time, MongoDB's robust aggregation pipeline framework makes it particularly suitable for handling such complex historical analyses seamlessly.

Another, although minor, feature that made MongoDB particularly attractive for the task was its efficient management of geospatial data. Indeed, it offers query operators for proximity and spatial relationships. Given that the dataset contained the latitude and longitude of the starting and ending locations of each ride, it seemed advantageous to leverage this feature.

One might consider Neo4j to be more appropriate for handling ride paths. However, since no information about the path itself was provided, such as intermediate stops or distances covered, constructing a graph model did not seem beneficial or practical for representing the data at hand. Moreover, given that Neo4j is not optimized for running aggregation queries over the entire graph, performing historical analysis over such a model did not

appear suitable for the intended analysis.

As far as Elasticsearch is concerned, while the technology is particularly optimized for running real-time analytics and performing full-text search operations on extensive datasets, the capabilities of such a database, primarily designed to be integrated into a search engine, did not fit well the nature of an analysis envisioned for the purpose of the project.

2 | Data Wrangling

In this chapter we shift the focus to Data Wrangling. The term refers to the process of transforming raw data in a cleaner format, to enhance its quality and usability for analytical purposes. This process was developed within a Python Notebook and relied on the functionalities offered by the Pandas library to perform data cleaning, transformation, aggregation, and statistical analysis. The result of this process was stored in a CSV file to be uploaded in MongoDB and it can be observed in Figures 2.1.

2.1. Collecting Data from Multiple Sources

In the introductory chapter we briefly mentioned the fact that the source from which data was collected provides *monthly* information about bike rides. Given that for the purpose of the analysis we wanted to focus on the last quarter of 2022, the corresponding three CSV files were downloaded separately.

Then, once having turned each CSV file into a Pandas DataFrames and once having checked that column names were actually coherent among the three sources, a merge operation was performed, by appending them on top of each other along the vertical axis. This operation allowed us to end up with a single DataFrame, with all the information needed to perform some analysis on rides taken in October, November, and December 2022.

2.2. Dealing with Missing Values

From a closer look at the datatet it emerged the presence of null values within fields corresponding to station names and IDs, but also related to the spatial coordinates of rides' endpoints. Further investigations revealed that missing values represented approximately 5% of the overall number of values inside the dataset. Although data was to be injected in MongoDB and this NoSQL technology in particular admits documents within the same collection to have a different structure, it did not appear meaningful in this case to have documents where essential fields like the end station's name were missing. This is because

the objective of the queries was to extract valuable and complete information concerning rides and therefore, having the above-mentioned fields missing from documents could have altered the effectiveness of queries aiming at identifying genuine patterns inside data. As a direct consequence, incomplete rows were discarded.

2.3. Addressing Discrepancies in the Assignment of ID Values

Data inspection also revealed another issue that required further data-cleaning steps. The problem concerned the mapping between station names and IDs, which was expected to be one-to-one. However, it turned out that multiple names were assigned to a single station ID, revealing that IDs were not unique. To address this issue two alternative solutions were considered:

1. Solve the name clash by choosing a single name for each station ID, by removing the rows with the discarded station names. However, the logic to adopt for the choice of a single name to be associated with an ID was to be made randomly, since no additional information was available on the way the mapping was originally done.
2. Drop columns related to station IDs, at the cost of keeping the discussed inconsistencies, which nonetheless affected only a small portion of the dataset.

In the end, the second strategy was the one adopted, because the station ID fields were not useful for queries that needed to be run in MongoDB afterward. Indeed, queries only reasoned in terms of station names, making their ID not relevant for the analysis. Therefore, to fully embrace the schema-on-read philosophy of MongoDB, the data model was designed to include only the information required by queries, thus the ID fields were discarded from the dataset.

2.4. Data Transformation and Augmentation

Some minor transformations were applied to the fields of the dataset. In particular, the names of some columns were modified to make them self-explanatory (e.g. `rideable_type` to `bike_type`, or `member_casual` to `user_category`).

Other transformations affected values within `bike_type` column, which contained categorical values such as "classic_bike," "electric_bike," and "docked_bike." During this process, the "bike" string at the end of each value was truncated.

Additionally, a new column was added to store the length of each ride. Values were computed starting from the `start_time` and `end_time` fields. However, during this process, a cleaning step became necessary due to the identification of rows with negative or null ride duration, which were removed from the dataset to ensure data accuracy and consistency. Then, ride duration values were mapped to their corresponding representation in milliseconds.

The last step of transformations applied involved the embedding of latitude and longitude values of each ride's endpoint into an array of coordinates. This transition shifted from an initial flat representation of four values (`start_lat`, `start_lng`, `end_lat`,`end_lng`) to an aggregate one, made by a couple of 2D arrays of coordinates.

	_id	bike_type	start_time	end_time	start_station_name	end_station_name	user_category	ride_length_in_ms	start_coordinates	end_coordinates
0	A50255C1E17942AB	classic	2022-10-14 17:13:30	2022-10-14 17:19:39	Noble St & Milwaukee Ave	Larrabee St & Division St	member	369000	[41.90068, -87.6626]	[41.90348607004, -87.6433534936]
1	DB692A70BD2DD4E3	electric	2022-10-01 16:29:26	2022-10-01 16:49:06	Damen Ave & Charleston St	Damen Ave & Cullerton St	casual	1180000	[41.920037, -87.67793683333333]	[41.854966518753926, -87.67569959163664]
2	3C02727AAF60F873	electric	2022-10-19 18:55:40	2022-10-19 19:03:30	Hoyne Ave & Balmoral Ave	Western Ave & Leland Ave	member	470000	[41.979878902, -87.681902051]	[41.966399801840986, -87.68870428204536]
3	47E653FDC2D99236	electric	2022-10-31 07:52:36	2022-10-31 07:58:49	Rush St & Cedar St	Orleans St & Chestnut St (NEXT Apts)	member	373000	[41.902273666666666, -87.6276925]	[41.898203, -87.637536]
4	8B5407BE535159BF	classic	2022-10-13 18:41:03	2022-10-13 19:26:18	900 W Harrison St	Adler Planetarium	casual	2715000	[41.874754, -87.649807]	[41.866095, -87.607267]

Figure 2.1: First five rows of the final dataset

3 | Dataset Description

In this chapter we will go into the details of the dataset under analysis, after the transformations introduced by the data cleaning step. Additionally, the non-relational schema will be presented, including data types for each field.

3.1. Data Injection

The previously generated CSV file was imported into MongoDB, enabling the execution of queries on the dataset. The procedure was performed using MongoDB Compass, which is a Graphical User Interface (GUI) that allows a more convenient form of interaction with MongoDB.

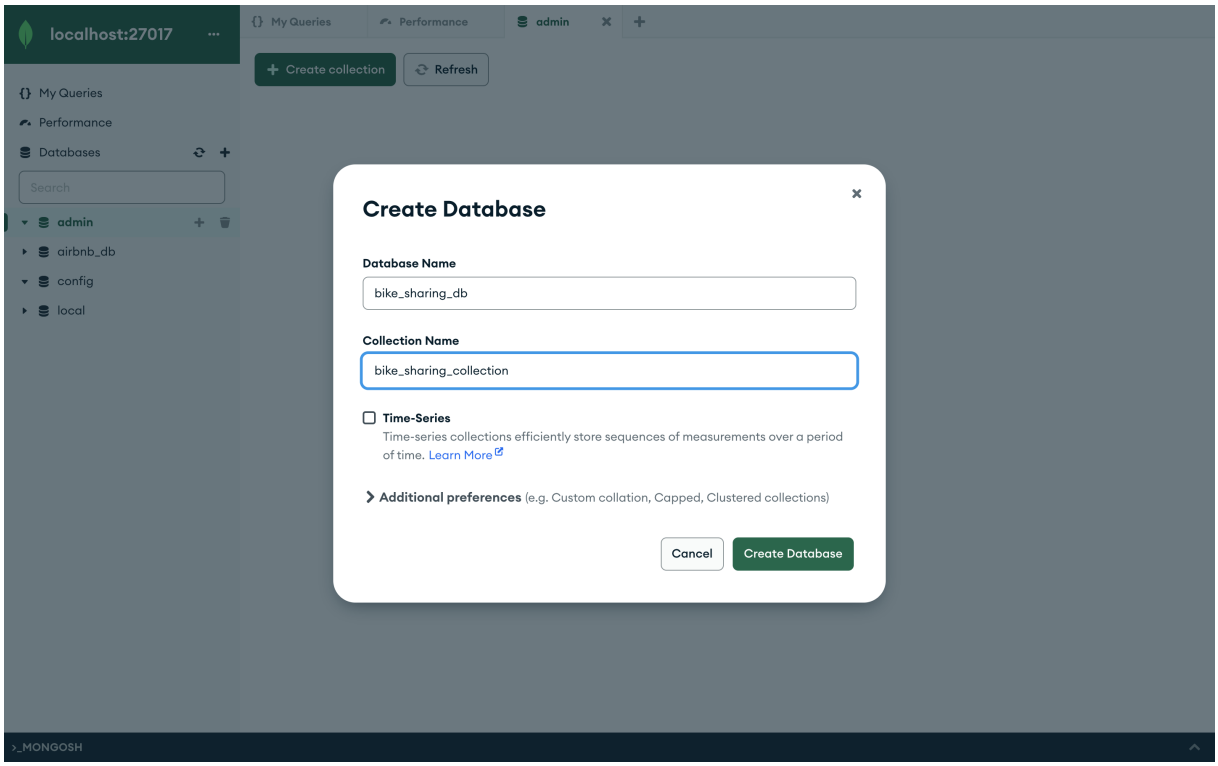

Figure 3.1: Definition of a new database instance and of a new collection of documents

The first step (Figure 3.1) was to create a new database instance, which was named `bike_sharing_db`. Then a collection of documents named `bike_sharing_collection` was added to the instance, specifying the above-mentioned CSV file as the source of data. The collection was designed to store documents representing rides taken with bikes of the bike-sharing system under analysis.

3.2. Schema Definition

The final steps of the data import process required defining the schema of documents that were about to be uploaded inside the specified collection. This procedure required the selection of fields to be included inside each document, together with their corresponding data types. Figure 3.2 and Figure 3.3 show the details of this procedure.

The result was a collection of documents structured as follows:

```
{
  "_id": String,
  "bike_type": String,
  "start_time": Date,
  "end_time": Date
  "start_station_name": String,
  "end_station_name": String,
  "user_category": String,
  "ride_length_in_ms": Integer,
  "start_coordinates": Array [Double, Double],
  "end_coordinates": Array [Double, Double]
}
```

- **_id** is the ID of each ride. It is an alphanumeric string derived from the original dataset. It identifies documents inside the collection in a unique way.
- **bike_type** is a string representing the type of bike used for the specific ride. Alternatives include classic, electric, or docked bikes, where "docked bike" stands for a category of bikes that are typically stationed at a specific location. Users can only rent these kinds of bikes from specific docking stations and need to return them to any available docking station after use.
- **start_time** represents the date and time of the start of the bike rental.
- **end_time** represents the date and time of the end of the bike rental.
- **user_category** is a string that specifies whether a user is a casual or a member user. The difference lies in the fact that the latter type owns a formal subscription

to the service.

- **ride_length_in_ms** is an integer value that holds the information about the length of a ride in milliseconds. From a preliminary analysis done in the previous phase of the project, it emerged that values for this field range from a few seconds up to 5 days. Notice that only the former bound was considered anomalous, assuming that renting a bike for several days in a row was allowed by the system.
- **start_coordinates** is an array of floating point numbers representing the latitude and longitude of a ride's starting point respectively.
- **end_coordinates** is an array of floating point numbers representing the latitude and longitude of a ride's starting point respectively.

Below we provide an example of a document that belongs to the `bike_sharing_collection`.

```
{
  "_id": "A50255C1E17942AB",
  "bike_type": "classic",
  "start_time": {
    "$date": "2022-10-14T15:13:30.000Z"
  },
  "end_time": {
    "$date": "2022-10-14T15:19:39.000Z"
  },
  "start_station_name": "Noble St & Milwaukee Ave",
  "end_station_name": "Larrabee St & Division St",
  "user_category": "member",
  "ride_length_in_ms": 369000,
  "start_coordinates": [41.90068,-87.6626],
  "end_coordinates": [41.90348607004,-87.6433534936]
}
```

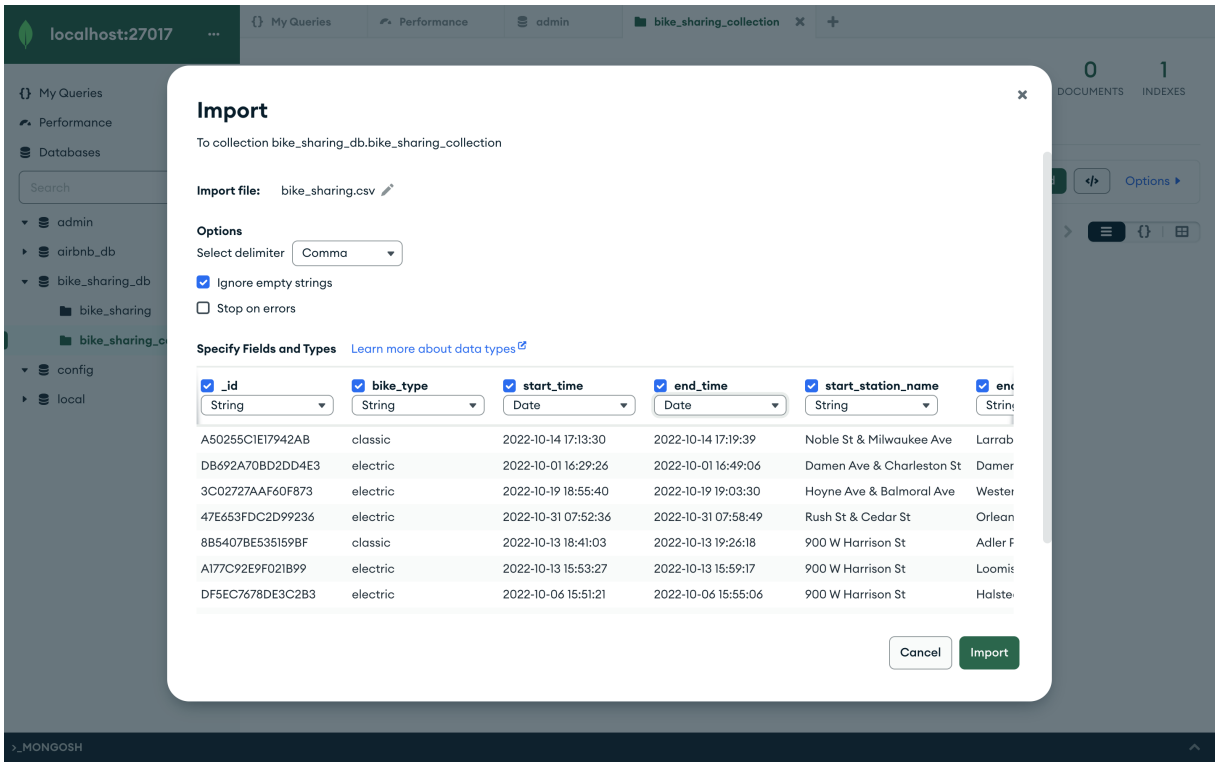

Figure 3.2: Schema definition procedure - Part 1

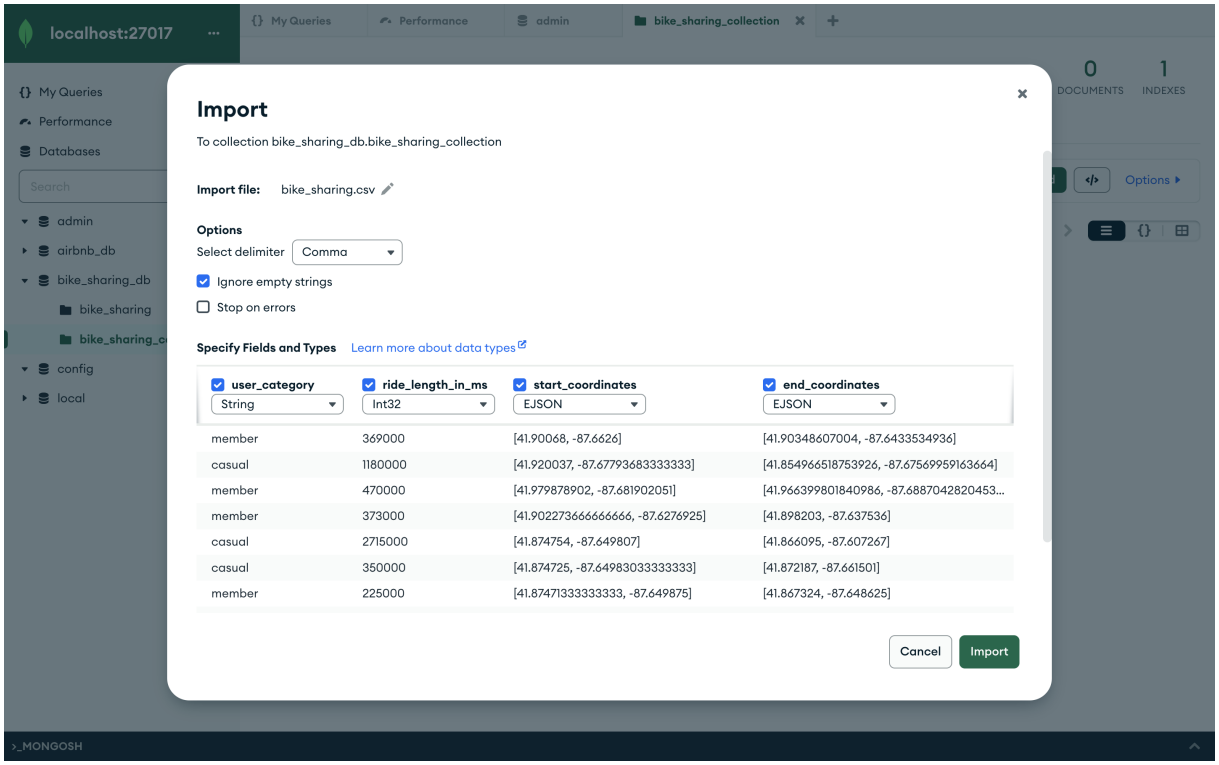

Figure 3.3: Schema definition procedure - Part 2

4 | Queries

This chapter is dedicated to the presentation of the queries performed in MongoDB to extract key information from the bike-sharing dataset. For some query results a visualization was also generated in a dedicated Python Notebook, by using the `pymongo` library to connect to the MongoDB database instance and execute queries. Then, after having retrieved the results, plots were created using `matplotlib`, `plotly`, and `seaborn` libraries.

4.1. Query 1: Rides Count per User and Bike Category

The purpose of the query is to identify, for each user category, the count of rides performed with each category of bikes.

```
db.bike_sharing_collection.aggregate([
    {
        "$group":{
            "_id":{
                "user_cat":"$user_category",
                "bike_type":"$bike_type"},
            "rides_count_per_user_per_bike":{"$sum":1}
        }
    },
    {
        "$sort":{"_id.user_cat":1}
    }
])
```

```
{
  _id: {
    user_cat: 'casual',
    bike_type: 'docked'
  },
  rides_count_per_user_per_bike: 20062
}
{
  _id: {
    user_cat: 'member',
    bike_type: 'electric'
  },
  rides_count_per_user_per_bike: 224986
}
```


Figure 4.1: Partial result of Query 1

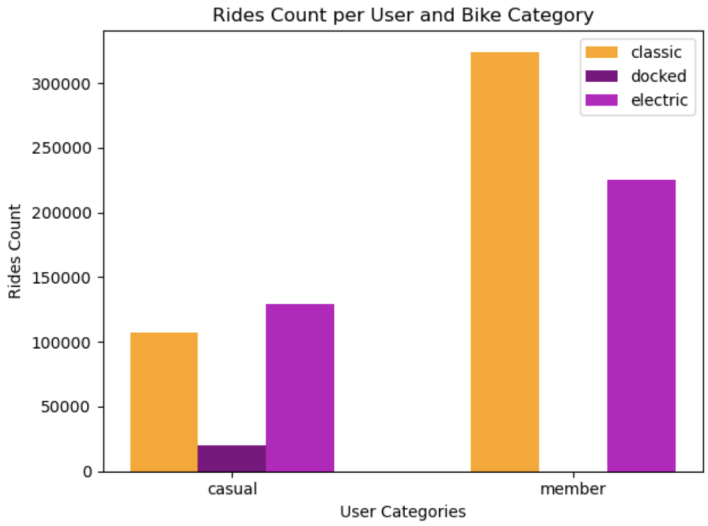

Figure 4.2: Visualization of the result of Query 1

Figure 4.1 shows the partial result obtained after running the query in MongoDB. By looking and the entire collection of documents returned, we can understand for each user category which type of bike is the most popular within that category. It turns out that electric bikes are the preferred choice among casual users, whereas member users choose classic bikes most of the time.

This result may suggest the introduction of a new marketing strategy to encourage member users to use electric bikes more often.

4.2. Query 2: Number of Rides and Average Ride Length per Day of the Week and User Category

The query aims at identifying, for each day of the week, the count and average length in minutes of rides performed by both categories of users.

```
db.bike_sharing_collection.aggregate([
    {
        "$group":{
             "_id":{
                 "day_of_week":{"$dayOfWeek":"$start_time"},
                 "user_cat":"$user_category"},
             "avg_rides_length_in_mins":{
                    "$avg":{
                        "$divide": ["$ride_length_in_ms",60000]
                    }
                },
                "rides_count":{"$sum":1}
        }
    },
    {
        "$project":{
            "_id":1,
            "avg_rides_length_in_mins":{
                "$ceil":"$avg_rides_length_in_mins"
            },
            "rides_count":1
        }
    },
    {
        "$sort":{
            "_id.day_of_week":1,
            "_id.user_cat":1
        }
    }
])
```

Figure 4.3 shows a portion of the result produced by the query after its execution. Each document in the collection returned carries information related to the average ride length in minutes, referred to a specific day of the week and a certain user category. Notice that according to the convention followed by the `$dayOfWeek` operator of MongoDB, the identifiers of the days of the week are to be interpreted as 1 for 'Sunday' and 7 for

>_MONGOSH

```
{
  _id: {
    day_of_week: 2,
    user_cat: 'casual'
  },
  rides_count: 30546,
  avg_rides_length_in_mins: 19
}
{
  _id: {
    day_of_week: 2,
    user_cat: 'member'
  },
  rides_count: 84876,
  avg_rides_length_in_mins: 11
}
{
  _id: {
    day_of_week: 3,
    user_cat: 'casual'
  },
  rides_count: 26522,
  avg_rides_length_in_mins: 16
}
```

Figure 4.3: Partial result of Query 2

'Saturday'.

Figure 4.4: Visualization of average rides duration by day of week

What emerges from the query result is that casual users take on average longer rides, but member users access the service more frequently. Moreover, it is interesting to underline that his trend is consistent across the week. Assuming that the cost for a ride is proportional to its duration and that a fixed price for unlocking the bike is paid by non-member users, this explains the reason why the number of accesses to the service is higher for member users. Non-members instead might opt to extend their ride duration to avoid incurring in additional unlocking fees for multiple short rides.

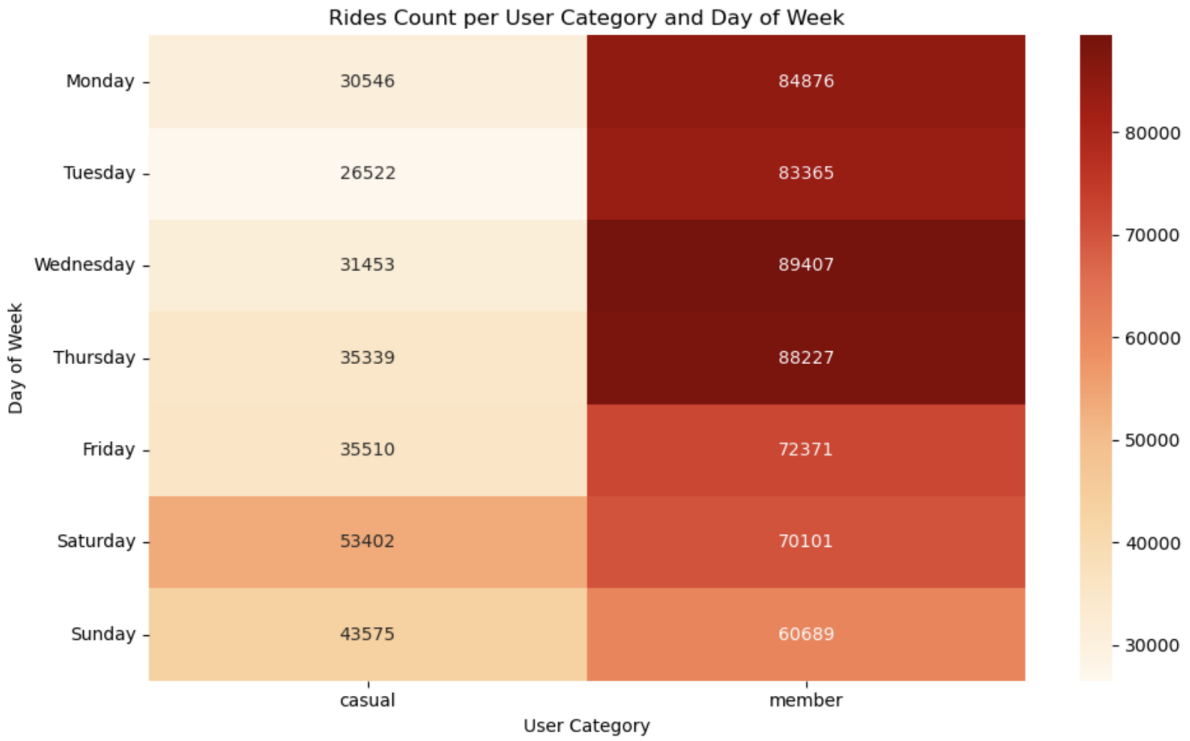

Figure 4.5: Visualization of rides count by day of week

Another possible scenario may be the one in which the company charges casual users with extra costs if they exceed a duration threshold, to avoid having a large number of bikes occupied for extended periods. In this case, given that non-registered users ride for longer periods of time, the company may want to start a campaign to encourage non-member users to subscribe to the bike-sharing system, to manage rides' duration more effectively.

However, having access to the company fairs would be helpful to more accurately interpret these results. Indeed, understanding how the pricing model is structured for both members and non-members, including any potential thresholds, additional fees for extended rides, or differences in pricing strategies, would provide clearer insights into users' behavior.

4.3. Query 3: Hourly Distribution of Morning Rides by Day of the Week

The query aims at performing an analysis of the hourly distribution of rides started between 6 A.M. and 11 A.M., for each day of the week.

```
db.bike_sharing_collection.aggregate([
    {
        "$project":{
            "_id":1,
            "day_of_week":{"$dayOfWeek":"$start_time"},
            "hour":{"$hour":"$start_time"}
        }
    },
    {
        "$match":{
            "$and":[
                {"hour":{"$gte":6}},
                {"hour":{"$lte":11}}
            ]
        }
    },
    {
        "$group": {
            "_id": {
                "day_of_week":"$day_of_week",
                "hour":"$hour",
            },
            "rides_count":{"$sum": 1}
        }
    },
    {
        "$sort":{
            "_id.day_of_week":1,
            "_id.hour":1
        }
    }
])
```

Figure 4.6 displays the partial result produced by the execution of the query. Each document returned contains the number of rides started in a specific time interval (e.g. rides started after 7 A.M. and before 8 A.M.), on a particular day of the week (e.g. on Monday). Notice that according to the convention followed by the `$dayOfWeek` operator

>_MONGOSH

```
{
  _id: {
    day_of_week: 1,
    hour: 6
  },
  rides_count: 2503
}
{
  _id: {
    day_of_week: 1,
    hour: 7
  },
  rides_count: 4055
}
{
  _id: {
    day_of_week: 1,
    hour: 8
  },
  rides_count: 5932
}
{
  _id: {
    day_of_week: 1,
    hour: 9
  },
  rides_count: 7840
}
```

Figure 4.6: Partial result of Query 3

of MongoDB, the identifiers of the days of the week are to be interpreted as 1 for 'Sunday' and 7 for 'Saturday'.

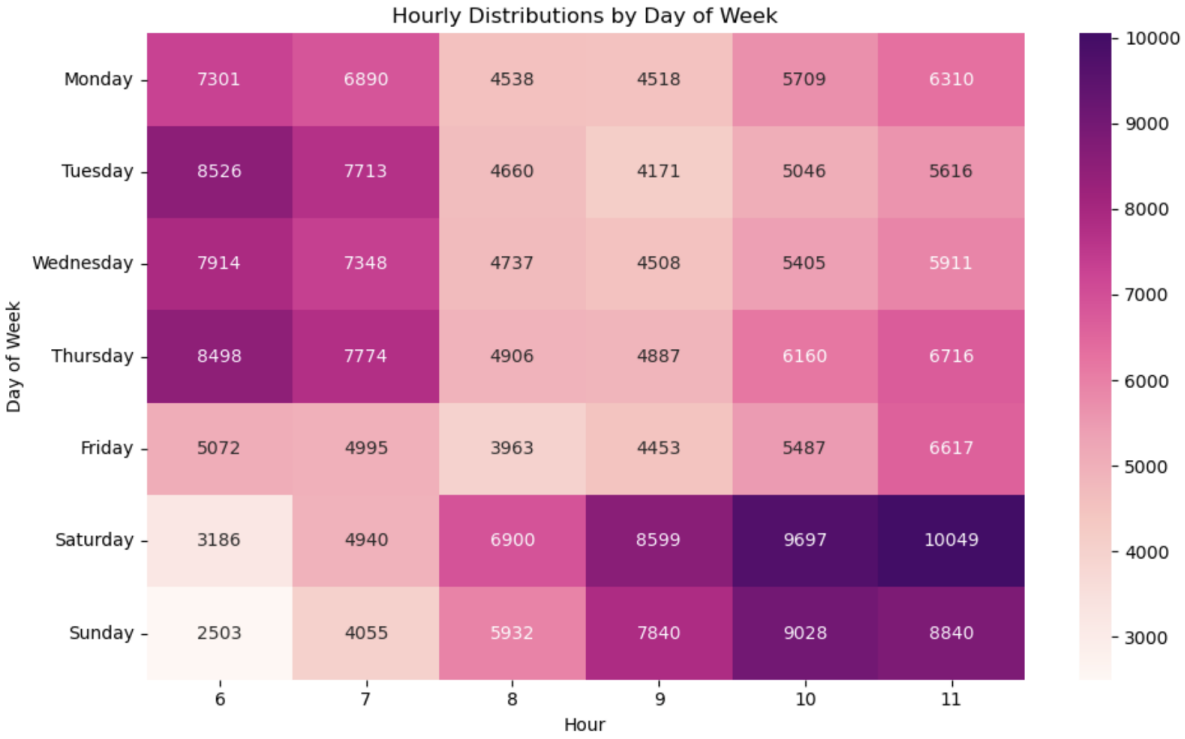

Figure 4.7: Heatmap of the result of Query 3

The query highlights the fact that weekday rush hours typically occur between 6 A.M. and 7 A.M. Conversely, weekends show a pick or rides between 10 A.M. and 11 A.M. This result is reasonable as it is perfectly aligned with the intuitive idea according to which people tend to have slow mornings during weekends, delaying of a couple of hours their activities outside, since they are not limited by strict working hours.

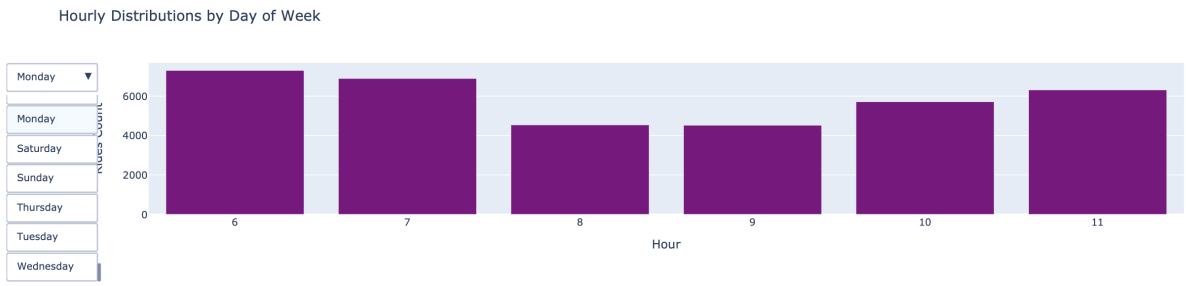

Figure 4.8: Interactive bar plot of the result of Query 3

4.4. Query 4: Count of December Rides by Week

The query aims at counting the number of rides performed in each week of December. The result should include the start and end date of the week, together with the number of days the week comprises.

```
db.bike_sharing_collection.aggregate([
    {
        "$match":{
            "$and":[
                {"start_time":{"$gte":ISODate("2022-12-01T00:00:00Z")}},
                {"start_time":{"$lte":ISODate("2022-12-31T23:59:59Z")}}
            ]}
    },
    {
        "$group": {
            "_id": {"week_num":{"$week":"$start_time"}},
            "rides_count":{"$sum":1},
            "week_start":{"$min":"$start_time"},
            "week_end":{"$max":"$start_time"}
        }
    },
    {
        "$project":{
            "_id":1,
            "rides_count":1,
            "number_of_days":{
                "$add":[
                    1,
                    {"$subtract":[
                        {"$dayOfYear":"$week_end"},
                        {"$dayOfYear":"$week_start"}]}]
            },
            "week_start":{
                "$dateToString":{
                    "format":"%Y-%m-%d","date":"$week_start"}},
            "week_end":{
                "$dateToString":{
                    "format":"%Y-%m-%d","date":"$week_end"}}
        }
    },
    {
        "$sort":{"_id.week_num":1}
    }])
```

```
>_MONGOSH
{
  _id: {
    week_num: 48
  },
  rides_count: 19133,
  number_of_days: 3,
  week_start: '2022-12-01',
  week_end: '2022-12-03'
}
{
  _id: {
    week_num: 49
  },
  rides_count: 44760,
  number_of_days: 7,
  week_start: '2022-12-04',
  week_end: '2022-12-10'
}
```

Figure 4.9: Partial result of Query 4

Figure 4.9 shows a subset of the query result. Each document produced is identified by the week number (the value refers to the yearly week count) and shows the count of rides performed in that week, the number of days inside the week, and its start and end dates.

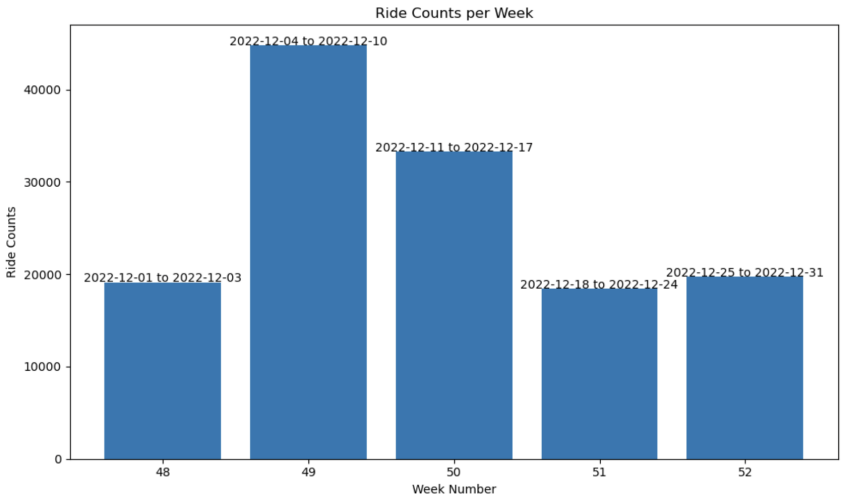

Figure 4.10: Visualization of the result of Query 4

From the outcome registered we can first observe that the first week is actually made by

just three days, thus it is reasonable that the count of rides is smaller than the following weeks. Another consideration that can be made concerns the last two weeks of the year, where the number of rides gets halved with respect to the count registered in the two intermediate weeks of the month. This decline coincides with Christmas and New Year's Eve periods, typically characterized by festive celebrations. This might have induced a reduced mobility, possibly indicating that fewer individuals commuted or travelled during that time.

4.5. Query 5: Number of Rides with very Short Duration

The query needs to identify the number of documents inside the collection representing rides with length shorter than one minute.

```
db.bike_sharing_collection.aggregate([
    {
        "$match":{
            "ride_length_in_ms":{"$lt":60000}
        }
    },
    {
        "$group":{
            "_id": true,
            "outlier_rides":{"$sum":1}
        }
    }
])
```

Figure 4.11: Result of Query 5

Figure 4.11 shows the result of Query 5. The document contains the number count of documents representing rides with a length smaller than one minute.

When considering the count of rides meeting this criterion against the total rows in the cleaned dataset (805,383), it emerges that outlier rows represent 1.9% of the dataset. This outcome could potentially highlight some anomalies within the app or it could be the result of user dissatisfaction. Hence, it would be beneficial for the company to conduct a deeper investigation into the underlying causes.

4.6. Query 6: Top 20 Most Popular Routes

The aims at retrieving the name of the start and end stations of the most popular routes, i.e. the ones with the highest count of journeys.

```
db.bike_sharing_collection.aggregate([
    {
        "$group": {
            "_id":{
                "start_station":"$start_station_name",
                "end_station":"$end_station_name"
            },
            "rides_count": {"$sum":1}
        }
    },
    {
        "$sort":{
            "rides_count":-1
        }
    },
    {
        "$limit":20
    }
])
```

Figure 4.12 shows a portion of the ranking produced as a result of the execution of Query 6. Each document in the collection contains the name of the start and end stations, together with the number of journeys registered for that particular route.

It is interesting to note that round-trip routes tend to hold close positions within the ranking. Another interesting property observed is that some routes have the same starting and ending stations. It might be that users of the system get engaged in leisure rides, finishing at their initial starting point. Alternatively, these routes may correspond to tourist tours designed to explore local attractions.

```
>_MONGOSH
< {
    _id: {
      start_station: 'Ellis Ave & 60th St',
      end_station: 'University Ave & 57th St'
    },
    rides_count: 2786
  }
  {
    _id: {
      start_station: 'University Ave & 57th St',
      end_station: 'Ellis Ave & 60th St'
    },
    rides_count: 2512
  }
  {
    _id: {
      start_station: 'Ellis Ave & 60th St',
      end_station: 'Ellis Ave & 55th St'
    },
    rides_count: 2448
  }
```

Figure 4.12: Partial result of Query 6

4.7. Query 7: Stations in Order of Popularity in the Mornings by Day of the Week

The objective of the query is to sort station names from the most to the least visited between 6 A.M. and 9 A.M. for each day of the week.

```
db.bike_sharing_collection.aggregate([
    {
        "$project":{
            "_id":1,
            "day_of_week":{"$dayOfWeek":"$start_time"},
            "start_station_name":1,
            "hour":{"$hour":"$start_time"}
        }
    },
    {
        "$match":{
            "$and":[{"hour":{"$gte":6}},{"hour":{"$lte":9}}]
        }
  },
    {
        "$group": {
            "_id":{
                "day_of_week":"$day_of_week",
                "start_station_name":"$start_station_name"
            },
            "num_of_rides_started_at_station":{"$sum":1}
        }
    },
    {
        "$sort":{
            "_id.day_of_week":1,
            "num_of_rides_started_at_station":-1
        }
    },
    {
        "$group":{
            "_id":"$_id.day_of_week",
            "stations_ranking":{
                "$push":{
                    "start_station_name":"$_id.start_station_name",
                    "num_of_rides_started_at_station":
                        "$num_of_rides_started_at_station"}}}}
])
```

```
>_MONGOSH
{
  _id: 2,
  stations_ranking: [
    {
      start_station_name: 'Kingsbury St & Kinzie St',
      num_of_rides_started_at_station: 275
    },
    {
      start_station_name: 'Clinton St & Madison St',
      num_of_rides_started_at_station: 255
    },
    {
      start_station_name: 'Ellis Ave & 60th St',
      num_of_rides_started_at_station: 226
    },
    {
      start_station_name: 'Clinton St & Washington Blvd',
      num_of_rides_started_at_station: 205
    },
```

Figure 4.13: Partial result of Query 7

Figure 4.13 shows the partial result of the execution of Query 7. Each document returned is identified by the number of day of the week it corresponds to. According to the convention followed by the `$dayOfWeek` operator of MongoDB, the identifiers of the days of the week are to be interpreted as 1 for 'Sunday' and 7 for 'Saturday'. Each document also contains an array of all the stations ordered from the one where the majority of rides is initiated in the morning on that day of the week, to the least popular in this terms.

This ranking could be interesting for the company to relocate bikes overnight closer to the station that are the most frequented the following morning. The choice of not truncating the list of stations for each day of the week to the top n-most popular on that day stems from the necessity to allocate to each station a number of bikes proportional to its popularity on the subsequent day. Thus, it is useful to know for *every* station in the dataset its corresponding popularity, by day of the week.

4.8. Query 8: Rides Count on the fourth Week of November by Day

The query aims at retrieving the count of rides performed during the fourth week of November, divided by day of the week. This week is particularly interesting to analyze, due to the occurrence of Thanksgiving on Thursday of that week. Moreover, examining the day following this festivity, known as Black Friday, adds further interest to the analysis.

```
db.bike_sharing_collection.aggregate([
    {
        "$match":{
            "$and":[
                {"start_time":{"$gte":ISODate("2022-11-01T00:00:00Z")}},
                {"end_time":{"$lte":ISODate("2022-11-30T23:59:59Z")}}]}
    },
    {
        "$group":{
            "_id":{
                "week_num":{"$week":"$start_time"},
                "day_of_week":{"$dayOfWeek":"$start_time"}
            },
            "rides_count":{"$sum":1},
            "date":{"$first":"$start_time"}
        }
    },
    {
        "$match":{
            "$expr":{"$eq":[
                "$_id.week_num",
                {"$add":[{"$week":ISODate("2022-11-01T00:00:00Z")},3]}]}
        }
    },
    {
        "$project":{
            "_id":1,
            "rides_count":1,
            "date":{
                "$dateToString":{
                    "format":"%Y-%m-%d", "date":"$date"}}
        }
    },
    {"$sort":{"date":1}}])
```

```
    day_of_week: 3
  },
  rides_count: 7435,
  date: '2022-11-22'
}
{
  _id: {
    week_num: 47,
    day_of_week: 4
  },
  rides_count: 8012,
  date: '2022-11-23'
}
{
  _id: {
    week_num: 47,
    day_of_week: 5
  },
  rides_count: 3851,
  date: '2022-11-24'
}
{
  _id: {
    week_num: 47,
    day_of_week: 6
  },
  rides_count: 5835,
  date: '2022-11-25'
```


Figure 4.14: Partial result of Query 8

Figure 4.14 shows the partial result produced by Query 8. Each document returned represents a day of week 47, which is the week of the year on which both Thanksgiving and Black Friday occurred in 2022. These festivities, occurring on Thursday and Friday respectively, are not bound to specific calendar dates, but are traditionally observed during the fourth week of November. Thus, the query was written to align with this logic. Days of week 47 are placed in separate documents, together with the full date and the count of rides performed on that day. Also notice that according to the convention followed by the `$dayOfWeek` operator of MongoDB, the identifiers of the days of the week are to be interpreted as 1 for 'Sunday' and 7 for 'Saturday'. As a consequence, Thanksgiving is on day 5 and Black Friday is on day 6.

The outcome suggests that as a general trend, users tend to access the service less often during both festivities. This conclusion is drawn from the observation that the count of rides performed on those days is lower than the weekly average of 5,933 rides per day. This confirms the pattern already identified within the dataset, indicating a consistent decrease in the number of rides recorded during special days.

4.9. Query 9: Stations Ordered from the Busiest to the least Busy

The objective of this query is to sort stations from the busiest, i.e. the one that counts the highest number of pick-ups and drop-offs, to the least busy.

```
db.bike_sharing_collection.aggregate([
    {
        "$facet":{
            "pickups_pipeline":[
                {
                    "$group":{
                        "_id":"$start_station_name",
                        "num_rides":{"$sum":1}
                    }
                }
            ],
            "dropoffs_pipeline":[
                {
                    "$group":{
                        "_id":"$end_station_name",
                        "num_rides":{"$sum":1}
                    }}
            ]}
    },
    {
        "$project":{
            "counts_per_station":
                {"$concatArrays":[
                        "$pickups_pipeline",
                        "$dropoffs_pipeline"
                    ]}
        }
    },
    {"$unwind":"$counts_per_station"},
    {
        "$group":{
            "_id":"$counts_per_station._id",
            "total_pickups_dropoffs":{
                "$sum":"$counts_per_station.num_rides"}}
  },
    {"$sort":{"total_pickups_dropoffs":-1}}
])
```

```
>_MONGOSH
< {
    _id: 'Streeter Dr & Grand Ave',
    total_pickups_dropoffs: 17073
  }
  {
    _id: 'Ellis Ave & 60th St',
    total_pickups_dropoffs: 15472
  }
  {
    _id: 'University Ave & 57th St',
    total_pickups_dropoffs: 14547
  }
  {
    _id: 'Kingsbury St & Kinzie St',
    total_pickups_dropoffs: 12547
  }
```

Figure 4.15: Partial result of Query 9

Figure 4.15 shows the partial result produced by Query 9. The query required the construction of two pipelines to be executed in parallel: one retrieving the aggregate count of rides over the start station field of the documents and the other counting the number of rides ended at a given station name. Results were then combined in order to sum for each station name the total count of pick-ups and drop-offs registered.

The information retrieved could be useful for the company to understand which are the busiest stations to strategically place advertisements in those places. In doing so the visibility of our promotions and services could be enhanced.

4.10. Query 10: Hourly Distribution of Rides starting from Millennium Park

This query aims at collecting the hourly distribution of rides starting at most 1 kilometer away from Millennium Park in Chicago.

The aggregation pipeline designed for this query relies on the `$geoNear` aggregation stage to filter documents based and the geospatial information of rides' starting points. This stage gives as output documents in order from the nearest to the farthest from a specified point, based on some distance criteria. According to the official documentation, the `$geoNear` stage must be the first one in the pipeline and additionally, an index must be defined over the coordinate field on which the distance is computed. For this reason, an index was defined over the **start_coordinates** field of documents inside the collection.

```
db.bike_sharing_collection.createIndex(
    {"start_coordinates":"2dsphere"})
```

```
db.bike_sharing_collection.aggregate([
    {
        "$geoNear":{
            "near":{
                "type":"Point",
                //Millennium Park coordinates
                "coordinates":[41.882702,-87.619392]
            },
            "spherical":true,
            "distanceField":"distance",
            "maxDistance":1000,
        }
    },
    {
        "$group":{
            "_id":{"hour":{"$hour":"$start_time"}},
            "rides_count":{"$sum":1}
        }
    },
    {
        "$sort":{"_id.hour":1}}
])
```

Figure 4.16 shows the partial result produced by Query 10. The query computes for each document referred to a ride the distance from the Millennium Park coordinates, which

>_MONGOSH

```
{
  _id: {
    hour: 14
  },
  rides_count: 14774
}
{
  _id: {
    hour: 15
  },
  rides_count: 15432
}
{
  _id: {
    hour: 16
  },
  rides_count: 12341
}
{
  _id: {
    hour: 17
  },
  rides_count: 7925
}
```

Figure 4.16: Partial result of Query 10

were found online (source: link). Then it only keeps rides started in the range of one kilometer from the park and only at the end it retrieves the hourly distribution of the filtered rides in the area.

The peek of rides initiated close to the parks is registered between 1 P.M. and 2 A.M.

List of Figures