Case Study:

Cyclistic Marketing Strategy

Case Background:

As a junior data analyst on the marketing analyst team at Cyclistic (a Chicago-based bike-sharing firm), I am responsible for determining how casual users and annual members use Cyclistic bikes differently. Casual riders are customers who buy single-ride or full day passes, whereas annual members pay a yearly fee for unrestricted riding access. According to the marketing director, the company's future success is dependent on increasing the number of yearly memberships by turning casual riders into annual members. My team will develop a new marketing plan based on this concept, subject to executive approval.

The purpose of this project will be to find and provide meaningful insights to guide any decision-making behind Cyclistic's new marketing approach.

Scope of Work

Deliverable {

Tasks

}

1. Define and discuss the project

- Creating a work scope document.
- Defining project goals and metrics for success by discussing the project with stakeholders and establishing expectations.
- Developing a problem statement.

2. Extract and prepare the data for exploration

- Determining the location and organization of data.
- Addressing concerns about licensing, privacy, security, and accessibility.
- Validating the accuracy of data.
- Identifying how the data contributes to the answers we seek.
- Data filtering and sorting.
- Determining the dependability, origins, completeness, present relevance, and trustworthiness of data.

3. Process the data for analysis

- Investigate and employ the appropriate data manipulation technologies for processing and analysis.
- Examine the data for dirt, particularly data that is obsolete, duplicated, incomplete, inconsistent, or wrong.
- Wrestle and clean the data until we are able to use unbiased and representative data analytics.

4. Perform a descriptive analysis.

- Calculations, data aggregations, and appropriate graphics can be used to identify trends and linkages.
- Data must be organized and formatted using useful aggregate tables.
- Give a summary of the findings.

5. Share key findings with stakeholders

- Create an effective and accessible presentation that uses compelling storytelling to address the original business problem.

6. Act on key findings

- Provide **three recommendations** for the marketing campaign to move forward with the analysis based on the study.

Deliverable: Define and discuss the project

Business Background

Business Model:

- Product: Across Chicago, bike-sharing geo-tracked and network-locked bikes.
- Customer types and revenue model: Casual-members (single-ride and full-day purchasers) and annual members (annual subscribers).
- Competitive advantages: Pricing flexibility and bicycle variety includes covering a wide range of consumer categories.

Product Background:

- There are 5,824 bicycles and 692 docking facilities.
- Casual riders for their mobility needs have picked eight Cyclistic.
- Percentage of riders choose an assistance bike.
- More than half of all riders choose classic motorcycles.
- Every day, 30% of users cycle to work.
- Riders are more inclined to ride for fun.

Discussing Goals and Expectations

Stakeholders	Expectations	Project/Business Goals
1. **Lily Moreno, Director of Marketing**	Evidence to support her argument and marketing suggestions.	Convert a substantial number of casual riders into annual members.
2. **Marketing analytics team**	Discovering the contrasts and motivations of various client types.	Produce data-driven and actionable solutions that help businesses make decisions.
3. **Cyclistic Executive Team**	Insights that are compelling, relevant, and easy to inform data-driven marketing decisions	Putting strategic measures in place to enhance corporate growth

Problem Statement

Cyclistic is facing an uncertain future and can no longer rely only on its old marketing techniques of developing general awareness and catering to a range of client demands. In the sake of corporate expansion, the director of marketing feels that Cyclistic should capitalize on the high profit margins of annual subscribers by marketing to existing casual consumers and enticing them to become yearly members. If such method is viable, a well-executed marketing campaign may result in more sustainable long-term income. To that end, we must examine how and why Cyclistic casual riders and members vary in order to consider any evidence, opportunities, and challenges to any future marketing approach.

Deliverable: Prepare the data for exploration

Preprocessing Scope

- Determining whether the data fulfills analytical criteria (ROCCC - Reliable, Original, Comprehensive, Current, and Credible).
- Identifying how the data contributes to the answers to the questions posed.
- Addressing ethical issues (security, accessibility, privacy, and licensing).
- Choosing a dataset, describing the data's location and organization.
- Data filtering and sorting.
- Ensuring the data's accuracy.

Data Source and Organization

The data we will be utilizing was taken from (**https://divvy-tripdata.s3.amazonaws.com/index.html**) and a helper dataset was downloaded from (**https://data.cityofchicago.org/Transportation/Divvy-Bicycle-Stations/bbyy-e7gq/data**) to filter out any unclean data in our primary dataset. Motivate International Inc. provides this data under this license, and this fictional organization (Cyclistic) and the open source data are exclusively utilized for this case study.

The data we have access to be largely a storehouse of quantitative measures collected throughout time. Each data point represents a single bike travel between parking stations. At first look, this data does not appear to be adequate for completely understanding how casuals and members utilize Cyclistic bikes differently. This information offers an overview of what they could be doing differently, but not why.

It is recommended to conduct a study of members and casuals to obtain qualitative responses to questions such as "What do you use Cyclistic bikes for?" and "Is there anything that makes you want to avoid using Cyclistic? If so, what would it be? ".

However assuming that CEOs are hesitant to spend further money and effort to collecting new data, we shall do our best with the data we have now.

We will look at a 12-month period, beginning in July 2020 and ending in June 2021. Each month has its own comma-separated value file with the same titles. Each record represents a bike trip in the bike-sharing program, and it includes several features such as a unique hash ID that serves as the table's primary key to identify each bike trip, the type of bike used, the type of customer (casual or member), information about the starting and ending docking station (name, ID, latitude, and longitude), and the Date Time for when the bike was picked up and dropped off.

Data Quality Assessment

- **Comprehensiveness:** ✓

This data set is sufficient for our investigation and is mainly free of human error.

- **Current:** ✓

The time span we are looking at provides us with an up-to-date picture as of the time this study was written.

- **Vetted:** ✓

According to our first studies, the majority of reported bike journeys are correct and devoid of significant inaccuracies.

- **Original:** ✓

Without any third-party influence, the data was sourced by a first-party group motivate international Inc.

Reliable: ✗

Reliability issues	Description
Duplicates	Although there are hundreds of duplicate main keys, each instance is distinct.
Validity	Minimal formatting and range limits were applied, resulting in difficulties such as end date times beginning before start date times, stations with numerous ids, ids belonging to several stations, and formatting changes over time.
Incomplete	One or more missing values are present in 10% of the records.

Ethical Concerns

- **Licensing:** These licensing restrictions limit the scope of this dataset.
- **Privacy:** De-identification provided sufficient anonymization, and the data is free of any personally identifying information.
- **Security:** The data is stored in a cloud repository controlled by authorized Motivate staff and secured by a reputable worldwide cloud services provider.
- **Credibility:** The good faith of the data supplier the is Motivate International Inc., may be relied on to guarantee that the data is factually represented. At first examination, there appears to be no indication of purposeful deception.
- **Accessibility:** All of Cyclistic's bike ride records are freely accessible to the public.

Data Integrity

The time-series nature of the data present some inconsistencies ,however the consistency and general accuracy of the data are not adjusted.

Filtering/Sorting

Between July 2020 and June 2021, the data has already been filtered. Any filtering will be used to eliminate major outliers and occurrences with mistakes.

Data Purpose

To uncover patterns and useful data into how different user types utilize Cyclistic bikes.

Assumptions:

We presume that the data collecting procedure was done correctly. Furthermore, we proceed with our research assuming that the data is free of obvious flaws, bias, and credibility difficulties. We also presume that the original repository was never accessed or updated without permission.

Data preprocessing, a subset of data preparation, refers to any sort of processing performed on raw data in order to prepare it for further processing.

Preprocessing Coding Log

Setup: library modules and helper functions

```
import numpy as np                          # efficient data types
import csv                                  # csv handling
import os                                   # data file path handling
import glob                                 # pathnames matching
import pandas as pd                         # data manipulation and analysis
import warnings                             # eliminate markdown warnings
import geopandas as gpd                     # geographical visualizations
import matplotlib.pyplot as plt             # plotting visuals
import seaborn as sns                       # visualization module
from shapely.geometry import Point          # plotting gps coordinates
import difflib                              # comparing strings
import re                                   # pattern matching


# global settings
sns.set(style="white")
%matplotlib inline
warnings.simplefilter('ignore')
pd.set_option('display.max_rows', 100)

def cols_mem_usage(*args: str) -> None:
    """Print dataframe column's memory usage in megabytes."""
    df_mem_usage = df.memory_usage(deep = True)
    for col_name in list(args):
        print(f"{col_name} total memory usage: {df_mem_usage[col_name]/1_000_000}MB

"""
pandas is inadvertantly converting integers into floats (i.e. 1 -> 1.0) on csv read
this is mainly to handle comparisons that should hold true like '1' == '1.0'
while accounting for strings without digits
"""
def try_int(x: str) -> str:
    """Attempt to convert a string into an integer, and back into a string to remov
    try:
        return str(int(x))
    except ValueError:
        return x
```

Data preview

Let's take a look at a single row in one of the csv files we downloaded to see if there are any extraneous fields we should remove or investigate.

```
def get_csv_peek(fp: str) -> pd.DataFrame:
    """Look at csv's first record as a dataframe"""
    with open(fp) as csvfile:
        reader = csv.reader(csvfile)
        column_headers = next(reader)
        first_row = next(reader)
        return pd.DataFrame(data = [first_row], columns = column_headers)

get_csv_peek("uncleaned_data/202007-divvy-tripdata.csv")
```

	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
0	762198876D69004D	docked_bike	2020-07-09 15:22:02	2020-07-09 15:25:52	Ritchie Ct & Banks St	180	Wells St & Evergreen Ave	291	41.906866	-87.626217	41.906724	-87.63483

end_lat	end_lng	member_casual
41.906724	-87.63483	member

Checking dataset size

```
def get_files_total_size() -> float:
    """Find the total dataset size by summing each individual file's size"""
    data_folder_path = os.getcwd() + "\\uncleaned_data\\"
    files = list(os.listdir(data_folder_path))
    total_size = 0
    for f in files:
        total_size += os.stat(data_folder_path + f).st_size

    return round(total_size/1_000_000, 2)

print(f"Total dataset size: {get_files_total_size()} Megabytes")
```

And by checking the dataset size we concluded that the size is 820 MB.

Deliverable: Process the data for analysis

Processing Scope

- Transform and clean the data so that it may be used successfully in analysis.
- Investigate and employ the appropriate data manipulation technologies for processing and analysis.
- Look for filthy data, namely data that is obsolete, duplicated, incomplete, inconsistent, or erroneous.

Choosing the right tools for the job

Given the tiny size of our combined dataset, I will be using Python 3.0 and many library modules for data processing and visualization. For effective usage of data structures and operations, I shall mostly rely on the Pandas software library. Because the full merged dataset fits easily in RAM, most simple procedures will be completed in a matter of seconds. If I run into any performance concerns, I will use a cloud cluster querying solution like Big Query.

Checking for dirty data

Merging our dataset into a single dataframe

```
def get_merged_df(data_folder_path: str) -> pd.DataFrame:
    """Merge data files into a single dataframe."""
    files = list(os.listdir(data_folder_path))
    file_paths = [data_folder_path + f for f in files]
    #load all files
    df_per_file = (pd.read_csv(fp,
                               low_memory = False,
                               parse_dates = ['started_at', 'ended_at'],
                               infer_datetime_format = True,
                               memory_map = True)
                   for fp in file_paths)
    return pd.concat(df_per_file, ignore_index = True)

df = get_merged_df(os.getcwd() + "\\uncleaned_data\\")
```

Loading our helper dataset

In addition to our core dataset, we'll use data from the city of Chicago to get the most up-to-date information about active Cyclistic docking stations. This information is available at https://data.cityofchicago.org/Transportation/Divvy-Bicycle-Stations/bbyy-e7gq/data.

And by using this data-set we will get a clean look of how data will be formatted.

```
stations_df = pd.read_csv("Divvy_Bicycle_Stations.csv", usecols = ['ID', 'Station Name', 'Latitude', 'Longitude'], index_col = 'Station Name')
stations_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 708 entries, Kedzie Ave & Chicago Ave to Pulaski Rd & 21st St
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   ID         708 non-null    int64
 1   Latitude   708 non-null    float64
 2   Longitude  708 non-null    float64
dtypes: float64(2), int64(1)
memory usage: 22.1+ KB
```

At First Glance

Examining our metadata

First, we examine the data in order to validate it before diving into data processing.

```
# get information from our dataframe (number of records, memory use and data types)
df.info(memory_usage = 'deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4460151 entries, 0 to 4460150
Data columns (total 13 columns):
 #   Column              Dtype
---  ------              -----
 0   ride_id             object
 1   rideable_type       object
 2   started_at          datetime64[ns]
 3   ended_at            datetime64[ns]
 4   start_station_name  object
 5   start_station_id    object
 6   end_station_name    object
 7   end_station_id      object
 8   start_lat           float64
 9   start_lng           float64
 10  end_lat             float64
 11  end_lng             float64
 12  member_casual       object
dtypes: datetime64[ns](2), float64(4), object(7)
memory usage: 2.1 GB
```

The memory size of our merged dataset has increased dramatically compared to the combined CSV's total memory size of 820 megabytes. This is partially due to the expensive memory use of object data types, which is a symptom of columns housing multiple data types.

Based on our helper dataset, we noticed that station IDs should only consist of integers, however our primary dataset does not, so we will need to address that once we apply our preprocessing transformations.

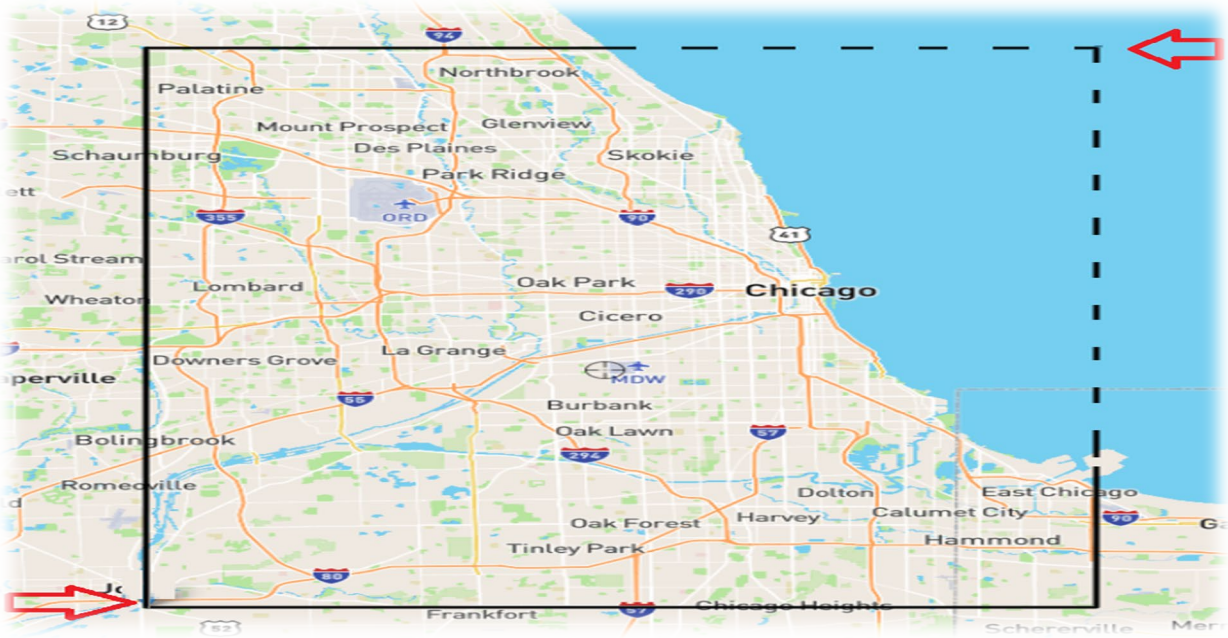
```
# get descriptive statistics under each numeric column
df.describe().apply(lambda s: s.apply('{0:.3f}'.format))
```

	start_lat	start_lng	end_lat	end_lng
count	4460151.000	4460151.000	4454865.000	4454865.000
mean	41.903	-87.644	41.903	-87.645
std	0.044	0.026	0.044	0.026
min	41.640	-87.870	41.510	-88.070
25%	41.882	-87.659	41.882	-87.659
50%	41.899	-87.641	41.900	-87.641
75%	41.930	-87.627	41.930	-87.628
max	42.080	-87.520	42.160	-87.440

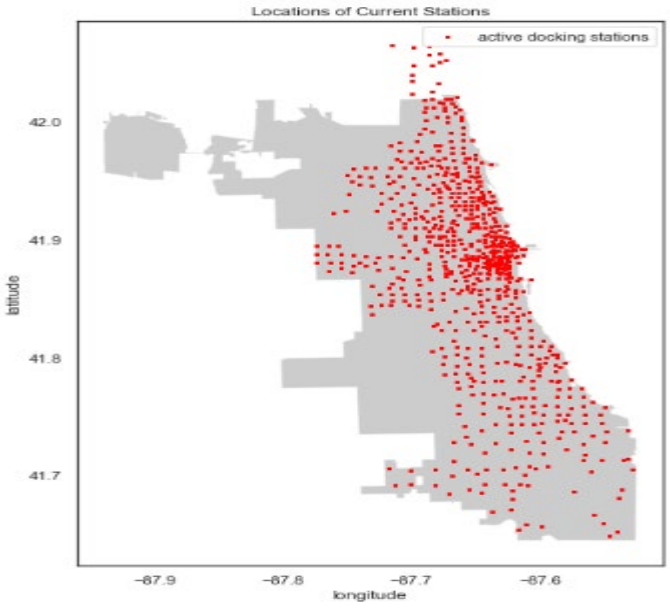
Validating GPS columns

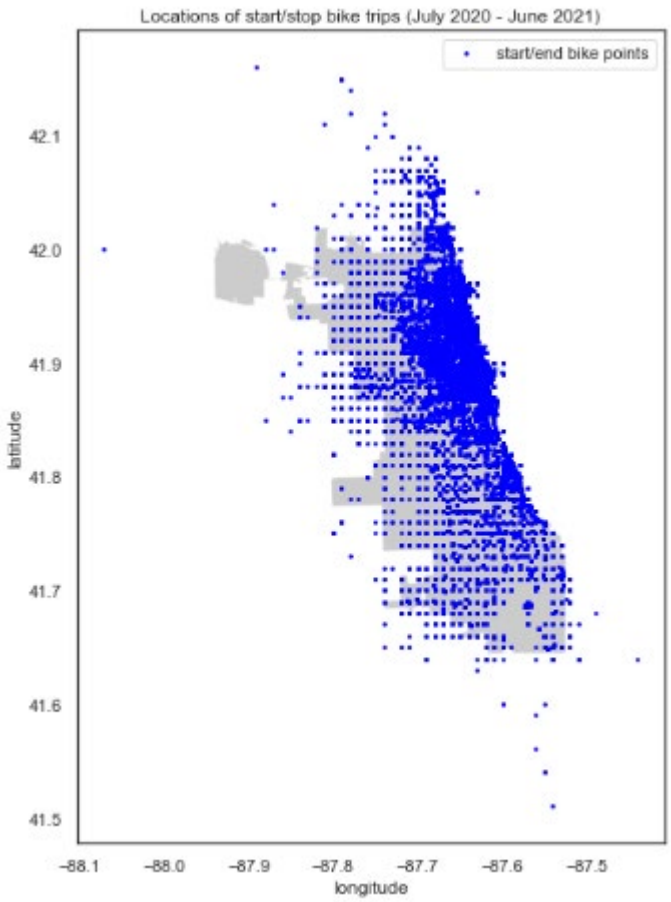
We want to verify that the longitude and latitude coordinates are accurate.

This bounding box has a range of (41.510, -88.070) to (42.160, -87.440).



```
def validate_coordinates() -> None:
    """Display data scatter points vs active station locations on a map of chicago"
    longitudes = pd.concat([df['start_lng'], df['end_lng']])
    latitudes = pd.concat([df['start_lat'], df['end_lat']])
    chicago_map = gpd.read_file('geo_export_392b318d-fb6e-4db5-b8e4-c5e35746342e.sh

    fig, (ax1, ax2) = plt.subplots(1,2, figsize=(15,15))

    official_geometry = [Point(xy) for xy in zip(stations_df['Longitude'], stations
    main_geometry = [Point(xy) for xy in zip(longitudes, latitudes)]

    main_gdf = gpd.GeoDataFrame(crs="EPSG:4326", geometry = main_geometry)
    official_gdf = gpd.GeoDataFrame(crs="EPSG:4326", geometry = official_geometry)

    chicago_map.plot(ax = ax1, alpha = 0.4, color = 'grey')
    main_gdf.plot(ax = ax1, markersize = 3, color = "blue", marker = "o", label = "
    chicago_map.plot(ax = ax2, alpha = 0.4, color = 'grey')
    official_gdf.plot(ax = ax2, markersize = 3, color = "red", marker = "s", label

    ax1.set(xlabel = "longitude", ylabel = "latitude")
    ax1.set_title('Locations of start/stop bike trips (July 2020 - June 2021)')
    ax2.set(xlabel = "longitude", ylabel = "latitude")
    ax2.set_title('Locations of Current Stations')
    ax1.legend(loc = 'upper right')
    ax2.legend(loc = 'upper right')

validate_coordinates()
```





Checking for duplicate values

```
# checking the number of unique values per column
print("Unique values per column")
for col in df.columns:
    print(f"{col}: {df[col].nunique()}")
```

```
Unique values per column
ride_id: 4459942
rideable_type: 3
started_at: 3813396
ended_at: 3801221
start_station_name: 712
start_station_id: 1360
end_station_name: 713
end_station_id: 1360
start_lat: 322245
start_lng: 310159
end_lat: 372054
end_lng: 344901
member_casual: 2
```

- "ride id," apparently the intended primary key of this database, plainly has duplicates because the number of unique values exceeds the capacity of the data frame.

- Because they both have a limited amount of unique values, "rideable type" and "member casual" might benefit from a more efficient data type.

- Because there are more station IDs than station names, many station names are connected with several station IDs.

Checking for entity participation constraints

We should anticipate a one-to-one link with IDs having a single related station based on our helper data collection. If it doesn't, we should be concerned about the consequences of such contradiction.

```
def get_inconsistent_ids_series() -> pd.Series:
    """Get a pandas series of stations sharing the same ID and their value counts""
    start_ids_sr = df.groupby(['start_station_id','start_station_name']).size().ren
    end_ids_sr = df.groupby(['end_station_id','end_station_name']).size().rename_ax

    vc1 = start_ids_sr.reset_index()['id'].value_counts()
    vc2 = end_ids_sr.reset_index()['id'].value_counts()

    start_ids_with_many_names = vc1[vc1 > 1].index
    end_ids_with_many_names = vc2[vc2 > 1].index

    start_mask = start_ids_sr.index.get_level_values(0).isin(start_ids_with_many_na
    start_ids_sr[start_mask]
    end_mask = end_ids_sr.index.get_level_values(0).isin(end_ids_with_many_names)
    end_ids_sr[end_mask]

    return pd.concat([start_ids_sr[start_mask], end_ids_sr[end_mask]], axis = 0)

inconsistent_ids_series = get_inconsistent_ids_series()
print("Number of station IDs with more than one name associated with it:", len(inco
inconsistent_ids_series.head(6)
```

```
Number of station IDs with more than one name associated with it: 70
id     station_name
19.0   Throop (Loomis) St & Taylor St     2410
       Throop St & Taylor St              1061
26.0   McClurg Ct & Illinois St          10359
       New St & Illinois St               1269
317.0  Wood St & Taylor St                2047
       Wood St & Taylor St (Temp)         1275
dtype: int64
```

Because of the time series structure of our data collection and the development of station names and locations as a result of events such as relocations, renovations, and renamings, these statistics illustrate that IDs can be associated with many station names.

Transforming and cleaning the data

Resolving duplicate data

```
def get_duplicates(field: str) -> pd.DataFrame:
    """Display rows with duplicate ride_ids"""
    vc = df[field].value_counts()
    duplicate_index = vc[vc > 1].index.to_list()
    n_duplicates = len(duplicate_index)
    print(f"Number of duplicates for {field}: {n_duplicates}")
    if n_duplicates > 0:
        print("Sample rows:")
        mask = df[field].isin(duplicate_index)
        return df[mask].sort_values(by = field).head(4)
    return

get_duplicates('ride_id')
```

```
Number of duplicates for ride_id: 209
```

	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
2378453	021A73F8C18B932D	docked_bike	2020-12-15 12:15:58	2020-11-25 16:48:02	Clark St & Winnemac Ave	TA1309000035	Ravenswood Ave & Berteau Ave	TA1309000018	41.973347	-87.667855	41.957921
2201528	021A73F8C18B932D	docked_bike	2020-11-25 16:35:39	2020-11-25 16:48:02	Clark St & Winnemac Ave	325.0	Ravenswood Ave & Berteau Ave	314.0	41.973347	-87.667855	41.957921
2169467	0334987B57662109	docked_bike	2020-11-25 16:15:04	2020-11-25 16:22:04	Broadway & Berwyn Ave	294.0	Lakefront Trail & Bryn Mawr Ave	459.0	41.978353	-87.659753	41.984037
2392329	0334987B57662109	docked_bike	2020-12-15 11:56:33	2020-11-25 16:22:04	Broadway & Berwyn Ave	13109	Lakefront Trail & Bryn Mawr Ave	KA1504000152	41.978353	-87.659753	41.984037

end_lng	member_casual
-87.673567	member
-87.673567	member
-87.652310	member
-87.652310	member

Although none of these rows precisely match, they do have a few unusual similarities:

- The names of the starting and ending stations are identical.
- Bike rides conclude at the same time but begin at widely different times. On one side, ride lengths are 'negative,' with numerous bike excursions commencing on 2020-12-15 and ending on 2020-11-25.
- Station identifiers do not match; formatting on one side is based on a mix of letters and integers.

We could investigate further to determine if these 'duplicates' are distinct and were registered incorrectly. Errors only show up on 209 rows out of 5 million. For that, we will omit the offending rows (duplicate rows and rides with negative ride durations) as cleaning dirty data data remains a priority.

```
mask = (df['ended_at'] - df['started_at']) < pd.Timedelta("0 days")
print("Number of rows with negative ride durations:", len(df.loc[mask]))
df = df.loc[~mask]
```

```
Number of rows with negative ride durations: 9872
```

```
get_duplicates('ride_id')
```

In addition, after checking we notice that we removed duplicates.

Let us convert rideable type and member casual to fit their respective data types:

```
print("Before:")
cols_mem_usage('rideable_type','member_casual')

# let's convert these low unique count columns into categories
df['rideable_type'] = df['rideable_type'].astype('category')
df['member_casual'] = df['member_casual'].astype('category')

print("After:")
cols_mem_usage('rideable_type','member_casual')
```

Cleaning station ids

Now we need to enforce consistency in all sections especially station id column by converting them into their integer equivalent.

```
# checking the number of non digit values in start_station_id and end_station_id
mask = (df['start_station_id'].astype(str).str.contains(pat = '[^0-9.]', regex = True)) & (df['start_station_id'].notnull())
print(f"unique values with characters in start_station_id: {df.loc[mask, 'start_station_id'].nunique()}")

mask = (df['end_station_id'].astype(str).str.contains(pat = '[^0-9.]', regex = True)) & (df['end_station_id'].notnull())
print(f"unique values with characters in end_station_id: {df.loc[mask, 'end_station_id'].nunique()}")
```

This does not seem to be an isolated issue. For that, the data is somehow formatted consistently between integers and non-numeric strings. Therefore, we will group stations by their ids and check if they are connected to multiple ones.

```
def get_inconsistent_stations_index() -> pd.Index:
    """Get a pandas index of stations IDs (start and end) that share multiple names"""
    start_station_ids_df = df.groupby(['start_station_name', 'start_station_id']).size()
    end_station_ids_df = df.groupby(['end_station_name', 'end_station_id']).size()
    vc1 = start_station_ids_df.reset_index()['start_station_name'].value_counts()
    vc2 = end_station_ids_df.reset_index()['end_station_name'].value_counts()
    starts = vc1[vc1 > 1].index
    ends = vc2[vc2 > 1].index
    return starts.union(ends)

inconsistent_stations = get_inconsistent_stations_index()

print("Number of inconsistent stations with more than one id:",len(inconsistent_stations))
```

We discovered a few broken participation limitations. We may utilize our helper dataset to guarantee that each station has exactly one ID. However, mismatches are possible due to station closures, renamings, and other factors. We must first address that issue.

```
mask = inconsistent_stations.isin(stations_df.index)
mismatched_names = set(inconsistent_stations[~mask])
mismatched_names
```

Let's see if we can bring the mismatches in our dataset up to date and impose some consistency.

```
def get_mismatches_and_suggested_replacements_df() -> pd.DataFrame:
    """Find any mismatches to our helper active stations set and find out if they have any close relations to other IDs/names"""
    start_station_ids_df = df.groupby(['start_station_name', 'start_station_id']).size()
    end_station_ids_df = df.groupby(['end_station_name', 'end_station_id']).size()

    start_station_ids_df.index.set_names(['station_name', 'associated_id'], inplace=True)
    end_station_ids_df.index.set_names(['station_name', 'associated_id'], inplace=True)

    # tabulate total id frequency under each name
    station_ids_count_df = pd.concat([start_station_ids_df.rename("start"), end_station_ids_df.rename("end")], axis=1)
    station_ids_count_df["count"] = station_ids_count_df['start'] + station_ids_count_df['end']
    station_ids_count_df = station_ids_count_df.drop(['start', 'end'], axis = 1)

    clean_station_names = list(stations_df.index)
    global mismatches_dict

    for mismatch in mismatches_dict:
        associated_name = None
        valid_associated_id = None
        # lists ids from most frequent to least frequent
        uncleaned_ids = station_ids_count_df.loc[mismatch].index.tolist()

        # try to find a valid id that lines up with the official set
        for id in uncleaned_ids:
            if re.match(r"[+-]?([0-9]*[.])?[0-9]+", str(id)) and stations_df.loc[stations_df["ID"] == int(id)].index.size > 0:
                associated_name = stations_df.loc[stations_df["ID"] == int(id)].index[0]
                valid_associated_id = id
                break

        # get the three closest name matches from the official set
        closest_names = difflib.get_close_matches(mismatch, clean_station_names, 3)
        change_id_to = None
        change_name_to = None

        # try to find an official name that lines up with the mismatch's based on id matching
        for name in closest_names:
            official_id = stations_df.loc[stations_df.index == name]["ID"][0]
            if str(official_id) in uncleaned_ids:
                change_id_to = official_id
                change_name_to = name
                break
```

```
        """
        if no closest names matched with their id, but an id associated with the mismatch did match up,
        check to see if the associated name has enough similarities with our mismatch.
        if it does, we can say with some certainty that we've found our match.
        """
        if change_name_to is None and len(uncleaned_ids) > 0 and associated_name is not None:
            # check that the uncleaned name is largely within the official associated id name
            text = mismatch.split(" ")
            keywords = []
            for word in text:
                if len(word) > 3: # disregard small keywords like ave, st
                    keywords.append(word)
            n_keywords = len(keywords)
            n_matches = 0
            for keyword in keywords:
                if keyword in associated_name:
                    n_matches += 1

            if n_matches == n_keywords:
                change_name_to = associated_name
                change_id_to = valid_associated_id

        """
        by this point, if no matches were made, then neither an associated id or a
        similar name is on the official list. In which case, we'll just assign the
        most common id and leave the name as is.
        """
        if change_id_to is None and change_name_to is None:
            # update to most common associated id since no match was found
            change_id_to = uncleaned_ids[0]
            # keep the name the same
            change_name_to = mismatch

        mismatches_dict[mismatch] = {
            'uncleaned_ids': uncleaned_ids,
            'valid_associated_id': valid_associated_id,
            'associated_id_official_name': associated_name,
            'change_name_to': change_name_to,
            'change_id_to': change_id_to,
        }

    return pd.DataFrame.from_dict(mismatches_dict, orient = 'index')

mismatches_dict = dict.fromkeys(mismatched_names)

get_mismatches_and_suggested_replacements_df()
```

	uncleaned_ids	valid_associated_id	associated_id_official_name	change_name_to	change_id_to
Halsted St & 63rd St	[388.0, KA1503000055]	388.0	Halsted & 63rd - Kennedy-King Vaccination Site	Halsted & 63rd - Kennedy-King Vaccination Site	388.0
Lake Shore Dr & Diversey Pkwy	[329.0, TA1309000039]	329.0	DuSable Lake Shore Dr & Diversey Pkwy	DuSable Lake Shore Dr & Diversey Pkwy	329.0
HUBBARD ST BIKE CHECKING (LBS-WH-TEST)	[671.0, Hubbard Bike-checking (LBS-WH-TEST)]	NaN	None	HUBBARD ST BIKE CHECKING (LBS-WH-TEST)	671.0
Lake Shore Dr & North Blvd	[268.0, LF-005]	268.0	DuSable Lake Shore Dr & North Blvd	DuSable Lake Shore Dr & North Blvd	268.0
Western Ave & 28th St	[446.0, KA1504000168]	446.0	Western & 28th - Velasquez Institute Vaccinati...	Western & 28th - Velasquez Institute Vaccinati...	446.0
Lake Shore Dr & Belmont Ave	[334.0, TA1309000049]	334.0	DuSable Lake Shore Dr & Belmont Ave	DuSable Lake Shore Dr & Belmont Ave	334.0
Malcolm X College	[631.0, 631]	631.0	Malcolm X College Vaccination Site	Malcolm X College Vaccination Site	631.0
Chicago Ave & Dempster St	[625.0, E011]	625.0	Dodge Ave & Main St	Chicago Ave & Dempster St	625.0
Lake Shore Dr & Wellington Ave	[157.0, TA1307000041]	157.0	DuSable Lake Shore Dr & Wellington Ave	DuSable Lake Shore Dr & Wellington Ave	157.0
Marshfield Ave & Cortland St	[58.0, TA1305000039]	58.0	Elston Ave & Cortland St	Marshfield Ave & Cortland St	58.0
Lake Shore Dr & Ohio St	[99.0, TA1306000029]	99.0	DuSable Lake Shore Dr & Ohio St	DuSable Lake Shore Dr & Ohio St	99.0
Lake Shore Dr & Monroe St	[76.0, 13300]	76.0	DuSable Lake Shore Dr & Monroe St	DuSable Lake Shore Dr & Monroe St	76.0
Broadway & Wilson Ave	[293.0, 13074]	293.0	Broadway & Wilson - Truman College Vaccination...	Broadway & Wilson - Truman College Vaccination...	293.0

The majority of them appear to make sense, with the ids accurately connected to renamed station names.

We will personally search up two suspiciously distinct stations to check correctness.

Marshfield Ave & Cortland St -> Elston Ave & Cortland St

Chicago Ave & Dempster St -> Dodge Ave & Main St

These original stations were closed and replaced with a new docking station at a different but close-by location. Let us keep the original station and its associated name, id, and coordinates to maintain consistency.

Let us see if we can look into the remaining station absent from the official stations' data set:

HUBBARD ST BIKE CHECKING (LBS-WH-TEST)

It appears that this station is a testing docking station, which I am surmising based on the absence of any Google results and lack of associations with the official active stations set. On lookup, we have also found another invalid station that acts as Cyclistic's base; Base - 2132 W Hubbard Warehouse.

Since these stations are presuming unavailable to customers and only used for testing, we will remove any rows that contain these docking stations as either a start or endpoint.

```
mask = (df['start_station_name'] == 'HUBBARD ST BIKE CHECKING (LBS-WH-TEST)') | (df['end_station_name'] == 'HUBBARD ST BIKE CHECKING (LBS-WH-TEST)')
df = df.loc[~mask]
mask = (df['start_station_name'] == 'Base - 2132 W Hubbard Warehouse') | (df['end_station_name'] == 'Base - 2132 W Hubbard Warehouse')
df = df.loc[~mask]
```

Moving on to the mismatch cleansing:

```
def cleanse_mismatches() -> None:
    """
    For every mismatched station name, we'll apply the changes we've brought in the table above
    by treating the associated station's unclean ids, and then moving on to the name
    """

    unchanging_names = ['Marshfield Ave & Cortland St', 'Chicago Ave & Dempster St']
    for uncleaned_name in mismatches_dict:
        print(f"Cleaning station {uncleaned_name}...")
        cleaned_name = mismatches_dict[uncleaned_name]['change_name_to']
        cleaned_id = str(mismatches_dict[uncleaned_name]['change_id_to'])

        # let's clean the bad ids first, since we need the unclean name to work with
        for uncleaned_id in mismatches_dict[uncleaned_name]['uncleaned_ids']:
            mask = (df['start_station_id'] == uncleaned_id) & (df['start_station_name'] == uncleaned_name)
            df.loc[mask, 'start_station_id'] = cleaned_id
            mask = (df['end_station_id'] == uncleaned_id) & (df['end_station_name'] == uncleaned_name)
            df.loc[mask, 'end_station_id'] = cleaned_id

        # now we treat the names
        if uncleaned_name not in unchanging_names:
            mask = (df['start_station_name'] == uncleaned_name)
            df.loc[mask, 'start_station_name'] = cleaned_name
            mask = (df['end_station_name'] == uncleaned_name)
            df.loc[mask, 'end_station_name'] = cleaned_name

cleanse_mismatches()
```

Now that the mismatches are taken care of, we can take care of cleansing ID inconsistencies for the rest of the dataset.

```
inconsistent_stations = get_inconsistent_stations_index()
print("Number of inconsistent stations with more than one id:",len(inconsistent_stations))
```

Number of inconsistent stations with more than one id: 656

```python
def cleanse_id_inconsistencies() -> None:
    """cleansing ID inconsistencies for the rest of the dataset with the helper dataset"""

    start_station_ids_df = df.groupby(['start_station_name', 'start_station_id']).size()
    end_station_ids_df = df.groupby(['end_station_name', 'end_station_id']).size()

    vc1 = start_station_ids_df.reset_index()['start_station_name'].value_counts()
    vc2 = end_station_ids_df.reset_index()['end_station_name'].value_counts()

    vc1.rename('station_name', inplace = True)
    vc2.rename('station_name', inplace = True)

    start_station_ids_df.index.set_names(['station_name', 'associated_id'], inplace=True)
    end_station_ids_df.index.set_names(['station_name', 'associated_id'], inplace=True)

    station_ids_count_df = pd.concat([start_station_ids_df.rename("start"), end_station_ids_df.rename("end")], axis=1)
    station_ids_count_df["count"] = station_ids_count_df['start'] + station_ids_count_df['end']
    station_ids_count_df = station_ids_count_df.drop(['start', 'end'], axis = 1)

    consistent_stations = pd.concat([vc1[vc1 == 1], vc2[vc2 == 1]], axis = 0)
    # look at stations within the official stations dataset where 1 id is associated with it
    mask = consistent_stations.index.isin(stations_df.index.to_list())
    consistent_stations = consistent_stations[mask].index.to_list()

    """
    cleaning consistent stations that do not match the official id
    """
    for name in consistent_stations:
        mask = station_ids_count_df.index.get_level_values(0) == name
        current_id = station_ids_count_df[mask].index.get_level_values(1)[0]
        mask = stations_df.index == name
        official_id = stations_df.loc[mask, 'ID'][0]

        if try_int(current_id) != str(official_id):
            mask = (df['start_station_id'] == current_id) & (df['start_station_name'] == name)
            df.loc[mask, 'start_station_id'] = str(official_id)
            mask = (df['end_station_id'] == current_id) & (df['end_station_name'] == name)
            df.loc[mask, 'end_station_id'] = str(official_id)

    mask = stations_df.index.isin(inconsistent_stations)
    inconsistent_official_stations = stations_df[mask].index.to_list()
```

```
    cleaning consistent stations that do not match the official id
    """
    for name in consistent_stations:
        mask = station_ids_count_df.index.get_level_values(0) == name
        current_id = station_ids_count_df[mask].index.get_level_values(1)[0]
        mask = stations_df.index == name
        official_id = stations_df.loc[mask, 'ID'][0]

        if try_int(current_id) != str(official_id):
            mask = (df['start_station_id'] == current_id) & (df['start_station_name'] == name)
            df.loc[mask, 'start_station_id'] = str(official_id)
            mask = (df['end_station_id'] == current_id) & (df['end_station_name'] == name)
            df.loc[mask, 'end_station_id'] = str(official_id)

    mask = stations_df.index.isin(inconsistent_stations)
    inconsistent_official_stations = stations_df[mask].index.to_list()

    """
    cleaning inconsistent stations by enforcing a single integer id
    """
    for name in inconsistent_official_stations:
        mask = station_ids_count_df.index.get_level_values(0) == name
        inconsistent_ids = station_ids_count_df.loc[mask].index.get_level_values(1).to_list()
        mask = stations_df.index == name
        official_id = stations_df.loc[mask, 'ID'][0]
        for id in inconsistent_ids:
            if try_int(current_id) != str(official_id):
                mask = (df['start_station_id'] == id) & (df['start_station_name'] == name)
                df.loc[mask, 'start_station_id'] = str(official_id)
                mask = (df['end_station_id'] == id) & (df['end_station_name'] == name)
                df.loc[mask, 'end_station_id'] = str(official_id)

cleanse_id_inconsistencies()
```

```
inconsistent_stations = get_inconsistent_stations_index()
print("Number of inconsistent stations with more than one id:",len(inconsistent_stations))
```

Number of stations with several IDs that are inconsistent: 0

All stations now use the same connected station ID.

Let's examine if any remaining station IDs were not processed for any reason.

```
# for stations outside of the official set
mask = (df['start_station_id'].str.contains('[A-Za-z]', na=False)) | (df['end_station_id'].str.contains('[A-Za-z]', na=False))
bad_station_ids_df = df.loc[mask]
print("Bad station id records remaining:", len(bad_station_ids_df))
bad_station_ids_df.head(4)
```

Despite this the bad station id records remaining are 26 near to 30

	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
3754953	40E0B4BC2C2F46C8	electric_bike	2021-06-23 11:21:04	2021-06-23 11:21:10	WEST CHI-WATSON	DIVVY 001	WEST CHI-WATSON	DIVVY 001	41.894792	-87.730906	41.894792
3754954	77328DAA369CF5DC	electric_bike	2021-06-23 16:18:20	2021-06-23 16:18:26	WEST CHI-WATSON	DIVVY 001	WEST CHI-WATSON	DIVVY 001	41.894710	-87.730827	41.894739
3754955	740E905067040E77	electric_bike	2021-06-23 10:14:56	2021-06-23 10:15:01	WEST CHI-WATSON	DIVVY 001	WEST CHI-WATSON	DIVVY 001	41.894739	-87.730896	41.894755
3754956	D19B4E87A1C9190D	electric_bike	2021-06-23 12:10:54	2021-06-23 12:11:02	WEST CHI-WATSON	DIVVY 001	WEST CHI-WATSON	DIVVY 001	41.894768	-87.730897	41.894777

Another set of rows has a single inactive station that is not included in official records. Based on the ride time and end station, these all appear to be test rides. Let's examine whether there are any valid bike trips with the same station name.

```
mask = (df['start_station_name'] == 'WEST CHI-WATSON') | (df['end_station_name'] == 'WEST CHI-WATSON')
print("Records with WEST CHI-WATSON as the station name:", len(df.loc[mask]))
```

```
Records with WEST CHI-WATSON as the station name: 26
```

Those are the same records as the above, so it's safe to remove them

```
df = df.loc[~mask]
```

Treating missing values

Before we can convert our station ids into the proper data types, we'll need to deal with missing values (i.e. NaNs).

Let's see if we can find any patterns:

```
df.isna().sum()
```

The majority of missing values are related to missing station IDs (name and id). We could either remove these rows or fill in the station names and IDs using inference using GPS locations. Other approaches for dealing with missing values are out of the question since we cannot employ forward, backward, or average fills without introducing considerable bias into our dataset.

Only 10% of our core dataset has missing values. We can still make effective use of the remaining dataset without jeopardizing our conclusions because the remaining 90% will appropriately represent customer behaviors from both Casuals and Members. For that drop rows with missing station ids or station names.

```
mask = (df['start_station_name'].isna()) | (df['start_station_id'].isna())
df = df.loc[~mask]
mask = (df['end_station_name'].isna()) | (df['end_station_id'].isna())
df = df.loc[~mask]
```

Converting station IDs

Now that we've cleaned all the dirty data, we can convert the station ID columns into more efficient formats

```
print("Before:")
cols_mem_usage('start_station_id','end_station_id')

df['start_station_id'] = df['start_station_id'].astype(float).astype(np.uint64)
df['end_station_id'] = df['end_station_id'].astype(float).astype(np.uint64)

print("After:")
cols_mem_usage('start_station_id','end_station_id')
```

Adding practical columns for analysis that would assist us in the analysis phase.

```
df['ride_length'] = (df['ended_at'] - df['started_at'])/np.timedelta64(1, 'm') # turn it into minutes
df['start_day_of_week'] = df['started_at'].dt.dayofweek
df['end_day_of_week'] = df['started_at'].dt.dayofweek
```

```
def haversine(lat1, lon1, lat2, lon2):
    """
    haversine formula determining the great-circle distance between two points on a sphere in miles
    """
    lat1, lon1, lat2, lon2 = np.radians([lat1, lon1, lat2, lon2])

    a = np.sin((lat2-lat1)/2.0)**2 + \
        np.cos(lat1) * np.cos(lat2) * np.sin((lon2-lon1)/2.0)**2

    return 3956 * 2 * np.arcsin(np.sqrt(a))

df['distance'] = haversine(df['start_lat'],df['start_lng'],df['end_lat'],df['end_lng'])
```

Let's output our data frame to a CSV file for further analysis in tableau.

```
df.to_csv('processed_cyclistic_data.csv', index = False)
```

Deliverable: Conduct a descriptive analysis

Analysis scope

- Calculations, data aggregations, and appropriate graphics can be used to identify patterns and linkages.
- Organize and arrange the information (including useful aggregate tables)
- Provide a summary of the findings.

Aggregates & Formatting

We'll use aggregates on a number of columns to see how members and casuals act differently.

Average Ride Length (minutes)

```
df.groupby('member_casual')['ride_length'].describe().unstack(1).apply(lambda s: "%.2f" %s)['mean']
```

Casuals spend 2.7 more time than members on Cyclistic bike journeys, which will most likely show in the average bike trip distance.

Average Distance Length

```
df.groupby('member_casual')['distance'].describe().unstack(1).apply(lambda s: "%.2f" %s)[['mean','std']]
```

It also does not! However, we must exercise caution in making conclusions here because we don't know much about the bike excursions other than customers picking up their bikes at point A and leaving them off at point B. What occurs in the interim remains a mystery. We may suppose that casuals are taking longer than members due to probable diversions, pit stops, and varying speeds involved in their bike excursions. In this dataset, there is no conclusive explanation as to why, but we might infer that casuals take their time because they ride Cyclistic bikes for fun, whereas members are more interested with commuting swiftly.

Busiest Days of the Week

```
df.groupby('member_casual')['start_day_of_week'].describe().unstack(1).apply(lambda s: "%.2f" %s)[['mean','std']]
```

```
      member_casual
mean  casual          3.55
      member          2.98
std   casual          2.00
      member          1.95
dtype: object
```

```
df.groupby('member_casual')['start_day_of_week'].value_counts()
```

```
member_casual  start_day_of_week
casual         5                   398635
               6                   333070
               4                   250285
               2                   189843
               3                   187573
               0                   185656
               1                   179388
member         2                   353763
               5                   339480
               4                   338864
               3                   330274
               1                   330065
               0                   305907
               6                   292243
Name: start_day_of_week, dtype: int64
```

Casuals, as expected, spend the majority of their bike journeys on weekends, whereas members utilize their bikes more equally throughout the week.

Aggregate measures can only go so far, so let's look at some images for further hints.

Relevant Visuals

I made some interactive visualizations in Tableau that you can see here on tableau public.

Identifying Trends and Relationships

Both User Types:

Rarely ride During the winter months, I ride my bicycle.

They pick up and put off their motorcycles at around the same distance.

Casuals:

During the summer, they become increasingly numerous and concentrate along the bay area.

They are most active in the afternoons and late evenings.

Year after year, have increasing ride lengths. Prefer docked bikes for extended bike rides.

During the winter, have a little presence.

Members:

Maintain consistent average riding durations throughout the year.

Are busy at all hours of the day, but especially in the mornings and afternoons; have excursions scattered across Chicago's downtown area

Are more consistent with their bike outings throughout the year

Findings Summary: Inconclusive

Our research uncovered some significant behavioral differences between Casuals and Members. These distinctions suggest some differences in fundamental values between the two groups, but at this point, the data we've gathered can only be used to make broad generalizations about the entire population of bike trips on both sides and is insufficient to draw any conclusions about your average member or casual. As a result, without collecting more relevant data and repeating the data analysis procedure, our findings cannot adequately address the question that prompted this effort.

Speculations

A. Casuals primarily use Cyclistic bikes for leisure

We make that assumption based on the fact that casuals:

- Saturdays and Sundays should be ridden twice as much as any other day of the week.
- They spend the majority of their bike outings around parks and bodies of water.
- Spend much more time on average on each bike excursion, implying that they spend time performing leisurely activities in between docking stations.
- Cyclistic bikes are rarely used in the morning (6 am-noon)
- Do not use Cyclistic bikes frequently enough to justify paying for an annual membership.

B. Members get the most out of their Cyclistic bikes by utilizing them for leisure and commuting on a regular basis.

We came to this conclusion based on the fact that members:

- Are driven by the financial advantages of an annual membership pass.
- On a routine workday, use Cyclistic bikes frequently during rush hour. Have a wide geographical distribution in the downtown area, especially in heavily inhabited regions.
- rely on motorbikes weekly and year-round, with no evident preference for any one day of the week.

Deliverable: Share Key Findings with Stakeholders

Deliverable: Act on Key Findings

Next Steps: Recommended Paths

A) Continuing with Data Extraction and Analysis

To validate our hypotheses and reveal any other major behavioral distinctions, we must poll a considerable portion of our user community to determine what genuinely distinguishes each user group's behavior. Choosing qualitative data that provides insights such as views and motives (what do people use Cyclistic bikes for?) adds much-needed context to our preliminary findings. Finding out what can encourage consumers in particular can help us harness consumer demands towards a more successful marketing plan. Furthermore, gathering additional quantitative data on user demographics would give greater insight into your average casual or member user, such as income, age, and weight.

The goal here is to obtain a strong grasp of the primary barriers and possibilities that may or may not interfere with Cyclistic's conversion plan. For example, if Casuals do not have adequate discretionary money, convincing them to raise their fiscal responsibility would be extremely difficult.

Finally, a successful conversion is dependent on a plan that gives enough incentive to casual users to make the transition.

B) Continuing with our Initial Findings

Unfortunately, the safest solution typically involves a bigger time and resource investment. If Cyclistic leaders are prepared to incur some risk, the findings of this inquiry can be utilized to kick-start and form the basis of a few proposals. Due diligence is required to analyze the viability of any new marketing strategy.

The following are the top three recommendations for the future:

1. Investigate new strategies to communicate the benefits of riding more regularly.

If Cyclistic can persuade casual users to expand their bike habits, they may be prepared to increase their commitment. This incentivization might come from a variety of ways, one of which could be a successful marketing effort informing people of the major advantages of riding. As previously said, this method is dangerous without a thorough grasp of casual users.

Persuading someone to change their habits has always been a difficult undertaking. This plan would have to assess how Cyclistic may actually alter Casual user patterns while also instilling a desire to cycle year-round. As a result, it may be beneficial to concentrate the emphasis of the marketing approach to a subgroup of Casuals who are especially vulnerable to persuasion and have relevant personal objectives that are reachable with Cyclistic.

2. Investigate models that reward higher-priced offers with greater benefits.

Casuals and Members currently receive comparable levels of service. They both have the same privileges for every available bike type on a first-come, first-served basis. Naturally, some individuals will be irritated by this system whenever they arrive at a station and are unable to use their preferred bike type or are inconvenienced by a bicycle scarcity during peak riding season.

Casuals will bear the brunt of the hardship if this difficulty is minimized for Members. Members, for example, may have the advantage of knowing what bikes are available in real-time and reserving them ahead of time, removing Casuals' access to the first-come, first-served experience. As a result, they must accept the inconvenience, upgrade their commitment, or switch to a competitive offering. To be sure, this is dangerous in the sense that it might result in harsh reaction and a high churn rate. Cyclistic, on the other hand, might implement a technique that gradually implements these modifications over time, exposing consumers to insignificant and reasonable variances in the Cyclistic experience.

3. As an alternative to conversion, consider new service and price models.

Cyclistic's future should not be restricted to a single specific plan for achieving long-term commercial success. We should consider all choices for innovating and improving the present Cyclistic experience, even if they are extreme. A three-tiered pricing structure, in instance, might maximize income by providing a service that fits between single-use permits and an annual membership that enables unlimited trips. This solution would strive to strike a compromise between the two, identifying the sweet spot for casuals who cannot justify the economics of a pricey yearly payment or the constraints of single-use passes. A weekend pass, for example, might be a nice compromise for people who only use Cyclistic for pleasure on weekends. However, it is critical to investigate how this method may inadvertently damage current annual members and whether there is a way to mitigate any unexpected impacts.

These benefits would not necessarily convert a big proportion of Casuals, since I doubt they would give enough rationale for them to pay more for the same quantity of bike use with additional benefits. Regardless, this method might be used with the three-tiered pricing model, presenting varying degrees of rewards for each service offering and encouraging most consumers to select the service that offers the greatest experience at the best price.