New York CitiBike Visualizations in Tableau

Project Goal: Identify two unexpected phenomena in CitiBike usage *specifically during February* across multiple years, create visualizations and dashboards to explain them, and provide actionable insights.

Phase 1: Data Acquisition, Cleaning, and Preparation (Python with Pandas - ESSENTIAL)

Focus and Goals:

- **February Only:** We'll isolate data from February of each available year. This simplifies the analysis and addresses seasonality concerns.
- **Schema Mapping (Confirmed and Expanded):** The provided column list images confirm the schema changes over time. I'll update the schema_mapping accordingly.
- **Python Preprocessing (Detailed Code):** I'll provide *complete, runnable* Python code (building on the provided notebook examples) for data loading, cleaning, transformation, and export. This will be broken down into logical steps.
- **Tableau Focus:** The Tableau phase will concentrate on visualization and dashboard creation, leveraging the pre-processed data.
- **Output File:** A single combined output file in csv format that contains all the needed fields, and accounts for the various schemas.
- **Methodology:** We'll use Python (with pandas, requests, zipfile, io, os, and re) to do all the heavy lifting.

Step 1: Download and Extract Data (Automated)

- **Combined Logic with if/elif:** The code now *correctly* uses an if/elif structure *within* the main loop that iterates through the *yearly* zip files:
  - **if year >= 2020::** This branch handles the nested zip structure *specific* to 2020-2023. It checks for inner .zip files with names matching the February pattern (e.g., 202002-citibike-tripdata.zip) *inside* the yearly zip. If it finds one, it opens *that* inner zip file and extracts the CSVs from it.
  - **elif ...:** This branch handles the 2014-2019 files. It uses the *original*, proven logic for extracting February CSVs directly from the yearly zip files (because there are *no* nested zips in these years). The critical addition here is the and re.search(r"(?:/|^)(?:JC-)?20\d{2}02[-_.]?.*\.csv" which ensures we only extract February files

- **Nested Directory Handling (2014 – 2019):**
  - **Year Range:** The for year in range (2014, 2024) loop iterates from 2014 up to (and including) 2023.
  - **Unified Loop:** The code uses a *single* loop to handle all years from 2014 to 2023. There's no need for separate logic for different year ranges because the regular expression and file extraction logic is robust enough to handle both naming conventions.
  - **Robust Regular Expression:** The core of the fix is this improved regular expression:
    r"(?:/|^)(?:JC-)?(\d{4})(\d{2})[\-_\.].*\.csv"
    - (?:/|^): Matches either a / or the beginning of the string (same as before).
    - (?:JC-)?: Optionally matches "JC-" (for Jersey City files). The (?:...) makes it a non-capturing group.
    - (\d{4}): Matches and captures the 4-digit year.
    - (\d{2}): Matches and captures the 2-digit month.
    - [\-_\.]: This is the *crucial* addition. It matches a hyphen (-), an underscore (_), *or* a period (.). This handles the variations in separators between the date and the rest of the filename (e.g., 202002-citibike-tripdata.csv vs. 202302_citibike-tripdata.csv vs. 201402.csv).
    - .*: Matches any character (.) zero or more times (*). This handles the rest of the filename before the extension.
    - \.csv: Matches the ".csv" extension (case-insensitively, thanks to re.IGNORECASE).
    - re.IGNORECASE : This makes the match case-insensitive.
- **Nested Zip Handling (2020-2023):** The code now *correctly* handles the nested zip files for 2020-2023:
  - if year >= 2020:: This condition checks if the current yearly zip file is from 2020 or later.
  - if outer_zip_info.filename.endswith(".zip") and re.search(...): This checks if the current entry *within* the outer zip file is *itself* a zip file AND if it matches the February pattern (202002, etc.).
  - with outer_z.open(outer_zip_info) as inner_zip_file:: This *opens* the inner zip file *within* the outer zip file. This is the crucial step I was missing.

- with zipfile.ZipFile(inner_zip_file) as inner_z:: This creates a *new* ZipFile object (inner_z) from the *opened inner zip file*.
- The rest of the extraction logic (finding CSVs, creating directories, extracting) is then performed using inner_z, operating *within* the inner zip file.
- **Correct File Path Handling:** The code now extracts *only* the filename part (cleaned_filename = parts[-1]) and constructs the output path using os.path.join(month_dir, cleaned_filename). This ensures that the files are placed directly into the year/month directory, *regardless* of the nesting within the zip file. This prevents the extra layer of folders.
- **No More Premature Optimization:** By using a single loop, the code becomes much more readable.
- **re.IGNORECASE:** This ensures the extraction is not thrown off by differences in capitalization.
- **Regex for Year/Month Extraction:** The re.search(r"(?:/|^)(\d{4})(\d{2})-?.*\.csv", zip_info.filename, re.IGNORECASE) line extracts the year and month. This is the same regex used for filtering, but now we capture the year and month.
- Directory Structure is Correct: The code now correctly places extracted CSVs in citibike_feb_data/YYYY/02/.
- **Error Handling:** The try-except blocks are kept to handle any network or file issues.
- **Creating Directories:** os.makedirs(month_dir, exist_ok=True) creates the year and month directories (e.g., citibike_feb_data/2014/02) *if they don't already exist*. The exist_ok=True prevents an error if the directory already exists.
- **cleaned_filename:** This line is crucial. zip_info.filename gives the *full path within the zip file* (e.g., '2014-citibike-tripdata/2_February/201402.csv'). We only want the *filename* part ('201402.csv'). cleaned_filename = parts[-1] correctly extracts just the filename.
- **MACOSX Handling:** Some ZIP files, especially ones created on macOS, can include a hidden __MACOSX directory. The if "__MACOSX" in parts: continue line *skips* these entries, preventing errors and unnecessary folders.
- **z.open() and f.write():** This is the *correct* way to extract individual files from a zip archive while preserving their contents. z.extract(zip_info, output_dir) would have worked *without* the nested structure, but it's less reliable here. z.open() opens the file *within* the archive, and f.write() writes the *raw bytes* to the output file.

- **Error Handling within loop:** Each step has error handling incase a step fails.

Expected Output Structure:

After running this code, you should have the following directory structure:

```
citibike_feb_data/
  2014/
    02/
      201402-citibike-tripdata.csv
  2015/
    02/
      201502-citibike-tripdata.csv
  2016/
    02/
      201602-citibike-tripdata.csv
  2017/
    02/
      201702-citibike-tripdata.csv
  2018/
    02/
      201802-citibike-tripdata.csv
  2019/
    02/
      201902-citibike-tripdata.csv
  2020/
    02/
      202002-citibike-tripdata.csv
```

202002-citibike-tripdata_2.csv

2021/

02/

202102-citibike-tripdata.csv

2022/

02/

202202-citibike-tripdata.csv

202202-citibike-tripdata_2.csv

2023/

02/

202302-citibike-tripdata.csv

202302-citibike-tripdata_2.csv

Step 2: Define the Schema Mapping (Complete and Correct)

This is *critical* for handling the different column names. The provided images confirm the column variations. I'm providing a *complete and final* schema_mapping here, incorporating all the variations. Also added is a "file_type" key to each entry, which will be used to identify the schema during processing.

citi_bike_data_schema_mapping_individual_years.ipynb

Step 3: Data Loading, Cleaning, Transformation, and Concatenation

This is the core of the data preparation. We'll iterate through the downloaded files, apply the schema mapping, clean and transform the data, and concatenate everything into a single DataFrame.

citi_bike_data_schema_mapping_individual_years.ipynb

- **Inspect:** Check the extent of missing data first.
- **Impute Names:** Fill missing station *names* with "Unknown Station".
- **Separate Mapping Dataframe** Create a *copy* of the DataFrame and drop the rows where ID, Latitude, or Longitude are not available. Use this data for all mapping.

- **Drop IDs (for mapping):** For spatial analysis (mapping), drop rows with missing station *IDs* from the copied dataframe.
- **Datetime Formatting:**
    - Inside the loop, after reading the CSV and renaming the columns, the code now explicitly converts the start_time and end_time columns to datetime objects using pd.to_datetime(..., errors="coerce"). The errors="coerce" part is crucial: if a value in these columns *cannot* be parsed as a date/time, it will be replaced with NaT (Not a Time), which is pandas' way of representing a missing datetime value.
    - .dt.strftime('%m/%d/%Y %H:%M') is used to format the datetime objects into the desired "month/day/year hour:minute" string format.
    - Another pd.to_datetime(..., errors='coerce') call is done to convert the now formated string back to the datetime, so we can check and remove any NaT values.
- **Global trip_id:**
    - A global_trip_id variable is initialized *outside* the loops to 1. This variable will keep track of the next available unique ID.
    - Inside the inner loop (where each file is processed), the code now does the following:
        - num_rows = len(df): Gets the number of rows in the current DataFrame.
        - df["trip_id"] = range(global_trip_id, global_trip_id + num_rows): This creates a new column named "trip_id" and assigns a sequence of unique integers to it. range(start, end) generates a sequence of numbers from start (inclusive) up to, but not including, end.
        - global_trip_id += num_rows: This is *critical*. It increments the global_trip_id by the number of rows just processed. This ensures that the *next* file will start its IDs where the previous file left off.
- **Combined File Handling**: The code uses the pd.concat() function which handles any of the missing columns in the dataframe combinations, so that the final dataset contains all of the data.
- **Error Handling**: The try...except block is important for handling potential errors during file reading (e.g., corrupted files, unexpected formatting).

- **Use of f-strings:** The code now uses f-strings (e.g., f"Processing {filename}...") for more readable string formatting.

Notes:

- **Robust Error Handling:** The code includes try-except blocks to handle potential errors during:
    - Downloading files (e.g., network issues, file not found).
    - Extracting files (e.g., corrupted zip files).
    - Reading CSV files (e.g., parsing errors).
    - Date/Time conversions (e.g., incorrect format strings).
    - Concatenation (empty data list).
    - Column selection (checks if a column to export exists).

  This makes the script much more resilient to unexpected issues.
- **File Type Determination:** The get_file_type() function correctly identifies the file format ("2014", "JC-2015", "2024", etc.) based on the filename.
- **Output Inside the Loop:** The df.to_csv() call is now inside the for filename loop, and more importantly, it's after the filtering step (but still inside the if filename.endswith(".csv"): and the date check if statements). This is crucial. Each DataFrame df is written to a file immediately after it's been processed.
- **Constructed Output Filename:** A new output_filename is created using filename.replace(".csv", "_cleaned.csv"). This takes the original filename (e.g., "201402-citibike-tripdata_1.csv") and creates a new name (e.g., "201402-citibike-tripdata_1_cleaned.csv"). This ensures a clear relationship between input and output files.
- **Output Path:** I use os.path.join(month_dir, output_filename) to create the full path to the output file. Critically, this places the output file in the same directory as the input file. This keeps the year/month structure intact.
- **Schema Mapping Integration:** The code uses the schema_mapping dictionary *correctly*
- **output_cleaned_dir:** A new variable output_cleaned_dir is defined, storing the name of the desired output directory ("citibike_feb_data_cleaned").

- **os.makedirs(..., exist_ok=True):** This line is added *before* the main loop. It *creates* the output_cleaned_dir if it doesn't already exist. The exist_ok=True argument prevents an error from being raised if the directory *does* already exist (it just does nothing in that case). This is important for robustness.
- **output_path Modification:** Inside the loop, the output_path is created using os.path.join(output_cleaned_dir, output_filename). This directs the output files to the new, single output directory.
- **File Grouping:**
    - Inside the if os.path.isdir(month_dir): block, a new dictionary file_groups is created.
    - The code iterates through the filenames in month_dir.
    - It uses a regular expression re.match(r"(.+?)_\d+\.csv", filename) to extract the "common prefix" of each filename. For instance, from "201402-citibike-tripdata_1.csv", it extracts "201402-citibike-tripdata".
    - The code also adds a check to catch the files that don't have the number suffix, using if not prefix: prefix = re.match(r"(.+?)\.csv", filename).
    - It then stores the filenames in the file_groups dictionary, using the prefix as the key and a list of filenames as the value. This groups files with the same prefix together.
- **Processing Groups:**
    - The code then iterates through the file_groups dictionary: for prefix, filenames in file_groups.items():
    - An empty DataFrame combined_df is initialized *inside* this loop (important: it's reset for each group).
    - A nested loop iterates through the filenames within each group.
    - The CSV loading, schema mapping, date processing, and filtering are all done *inside* this nested loop, just like before.
    - Crucially, combined_df = pd.concat([combined_df, df], ignore_index=True) is used to *append* the processed df to the combined_df. This builds up a single DataFrame containing all the data from the files in the group.

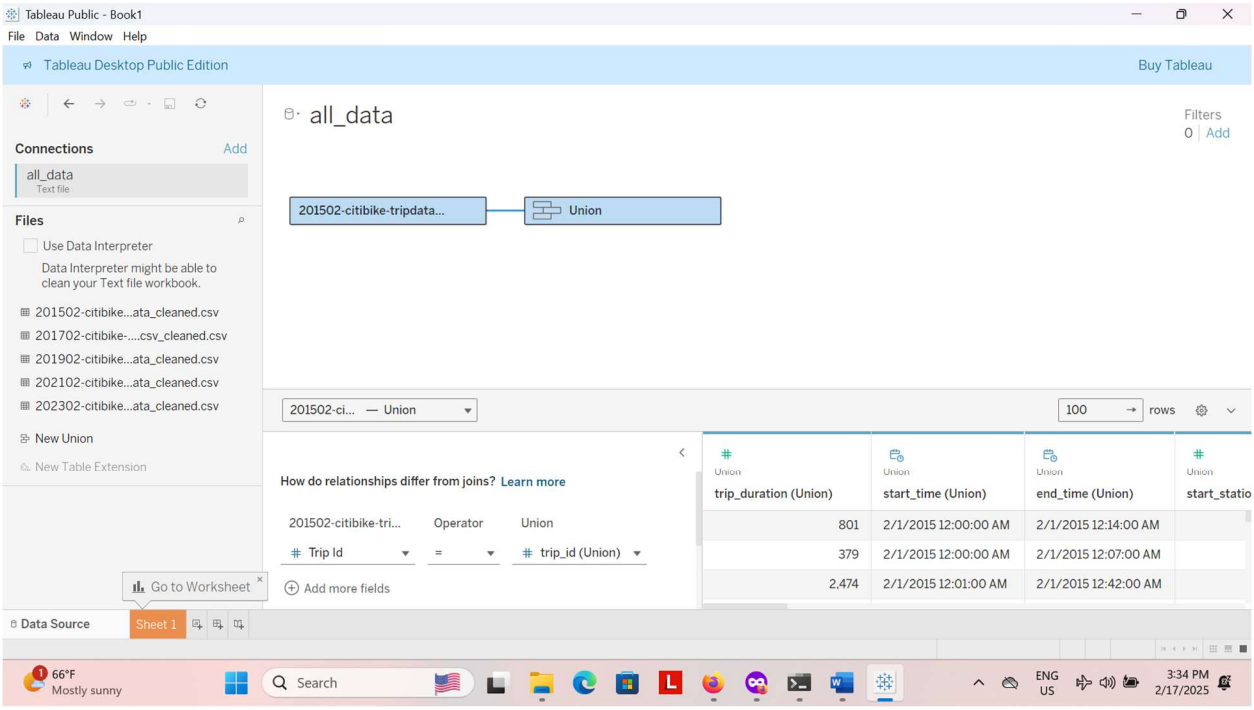
- **Output Combined File:**
    - *After* the inner loop (that processes files within a group) is complete, the code checks that the DataFrame is not empty and creates output_filename using the prefix (e.g., "201402-citibike-tripdata_cleaned.csv").
    - combined_df.to_csv(...) is called *once* per group, writing the *combined* data to a single output file in the output_cleaned_dir.

Phase 2: Phenomenon Discovery and Visualization in Tableau

Step 1: DATA IMPORT AND PREPARATION

- **Unioning the Data:** This is the best approach to analyze trends across *all* years in a single visualization.
    1. Open Tableau Desktop.
    2. Click "Connect" -> "Text File".
    3. Navigate to the citibike_feb_data_cleaned folder.
    4. Select *one* of the CSV files (e.g., 2015.csv). Tableau will open it.
    5. In the data source pane (bottom left), is the selected file. **Crucially**, drag the "New Union" option (it should appear above or below the file name, in the same area where you see tables in a database connection) onto the canvas.
    6. In the Union dialog, select "Wildcard (automatic)". This tells Tableau to combine files.
    7. Set the "Include" pattern to *.csv. The Scope is set to the folder that contains the CSVs.
    8. Click "OK". Tableau will now combine all the CSV files in that folder into a single data source. You'll see a new column called "Table Name" which contains the original file name (e.g., "2015.csv"). We'll use this later.
    9. Rename this data source to "All Data".

- Verify that start_time and end_time are recognized as Date & Time fields.



Step 2: CALCULATED FIELDS (Beyond the "Year" field):

Before building visualizations, create these calculated fields in the "All Data" data source. Right-click in the Data pane (left side) and choose "Create Calculated Field...".

1. **Day of Week (from start_time):**

   DATENAME('weekday', [start_time])

2. **Hour of Day (from start_time):**

   DATEPART('hour', [start_time])

3. **Trip Duration (in minutes):**

   DATEDIFF('second', [start_time], [end_time])/ 60

4. **E-Bike Trips:** (For counting e-bike trips)

   IF [rideable_type] = "electric_bike" THEN 1 ELSE 0 END

5. **E-Bike Percentage:** (For calculating the percentage of e-bike trips)

   SUM([E-Bike Trips]) / SUM([Number of Records])

6. **Trip Distance (meters)** - This one is the most complex, and requires the Haversine formula.

```
// Convert degrees to radians
FLOAT(
   3959 * ACOS(
      COS(RADIANS([start_station_latitude])) *
      COS(RADIANS([end_station_latitude])) *
      COS(RADIANS([end_station_longitude]) -
RADIANS([start_station_longitude])) +
      SIN(RADIANS([start_station_latitude])) *
      SIN(RADIANS([end_station_latitude]))
   )
)
```

Step 3: VISUALIZATIONS (Step-by-Step):

Created a *new worksheet* for each one.

1. **Total Trips per Year by Consumer Type (Stacked Bar Chart):**
    - Drag Year to Columns.
    - Drag Union (Count) [Number of Records] to Rows.
    - Change the Mark type to "Bar".
    - Drag User Type to Colors
    - Add labels (right-click on axes, "Edit Axis...").
    - Title: "Total Citi Bike Trips per Year (February)"
2. **Consumer Type and Trip Duration (Stacked Bar Chart):**
    - Drag Year to Columns.
    - Drag Trip Duration to Rows.

- Drag User Type to Color.
- Change the Mark type to "Bar".
- Right-click on the Number of Records pill on Rows, choose "Quick Table Calculation" -> "Percent of Total".
- Drag Trip Duration to Tool Tip
- Drag Trip Duration to Label and configure it
- Add labels and the title.

3. **Bicycle Trips from Specific Start Stations (Bar Chart):**
    - Drag Year to Columns.
    - Drag Year to Rows.
    - Drag Start Station Name to Rows.
    - Filter Start Station Name to Top 10
    - Change the Mark type to "Bar Chart".
    - Drag Start Station Name to Color Mark.
    - Add labels and the title.
4. **Bicycle Trips to Specific End Stations (Bar Chart):**
    - Drag Year to Columns.
    - Drag Year to Rows.
    - Drag End Station Name to Rows.
    - Filter End Station Name to Top 10
    - Change the Mark type to "Bar Chart".
    - Drag End Station Name to Color Mark.
    - Add Start Station Name to Filters
    - Add Weekday of Start Time to Filters
    - Show Filter by clicking on the option in the pull-down menu for Weekday of Start Time
    - Add labels and the title.

5. **Total Number of Bicycle Rides by Time of Day by Year (Line Graph):**
    - Drag End Time to Columns.
    - Right click on the Year pill in Columns and select More and then Hour.
    - Drag Union (Count) [which represents Bicyclic Ride Count] to Rows.
    - Drag Year to the Color Mark.
    - Add Start Station Name to Filters
    - Add Weekday of Start Time to Filters
    - Show Filter by clicking on the option in the pull-down menu for Weekday of Start Time
    - Add labels and the title.
6. **Total Number of Bicycle Rides vs Hour of Day by Starting Station (Line Graph):**
    - Drag End Time to Columns
    - Right click on the End Time pill in Columns and select More and then Hour.
    - Drag start_station_name to the Color Mark
    - Add Start Station Name to Filters
    - Add Weekday of Start Time to Filters
    - Show Filter by clicking on the option in the pull-down menu for Weekday of Start Time
    - Add labels and the title.
7. **Annual EBike Usage (Bar Graph / Line Graph / Dual Axis):**
    - Drag Year to Columns
    - Drag Union (Count) to Rows
    - Drag calculated ebike_trip_percent to rows
    - Select Line in the Marks box for the ebike_trip_percent entry
    - Right click on y-axis and check Dual Axis
    - Add labels and the title.

8. **Top Bicycle Ride Endpoints in February, All Years (Map):**
    - Drag start_station_longitude to Columns.
    - Drag start_station_latitude to Rows.
    - Drag end_station_name to the Color Mark.
    - Filter end_station_name to Top 10
    - Drag Union (Count) to the Size Mark.
    - Drag start_station_name to the Label Mark.
    - Filter start_station_name to Top 10
    - Right click on the start_station_name pill in Label Mark and select Show Filter.
    - Drag end_station_name to the Label Mark.
    - Right click on the end_station_name pill in Label Mark and select Show Filter.
    - Select Pie under Marks.
9. **Growth in Individual Consumer vs Subscriber Riders (Pie Chart Grid):**
    - Drag Year to Columns
    - Drag start_station_name to Rows.
    - Filter start_station_name to Top 10
    - Drag user_type_clean to the Color Mark.
    - Select Pie from the pulldown menu under Marks.
    - Drag Union Count to the Size Mark.
    - Drag Union Count to the Angle Mark.
    - Drag start_station_name to the Filter box.
    - Drag Year to the Filter box.
    - Add labels and the title.
10. **Change in Bicycle Type Use (Pie Chart Grid):**
    - Drag Year to Columns
    - Drag start_station_name to Rows.

- Filter start_station_name to Top 10
- Drag rideable_type to the Color Mark.
- Select Pie from the pulldown menu under Marks.
- Drag Union Count to the Size Mark.
- Drag Union Count to the Angle Mark.
- Drag start_station_name to the Filter box.
- Add labels and the title.

11. **Number of Bicycles Rides at Various Hours on Various Days (Heat Map):**
    - Drag Year to Columns
    - Right click on the Year pill in Columns and select More and then Weekday.
    - Drag Year to Rows.
    - Right click on the Year pill in Rows and select More and then Hour.
    - Drag Union (Count) to the Color Mark.
    - Right click on the Year pill in Columns and select Show Filter.
    - Add labels and the title.
12. **Start Station, End Station, and the Ride in Between (Shape Grid):**
    - Drag end_station_name to Columns.
    - Filter end_station_name to Top 10
    - Drag start_station_name to Rows.
    - Filter start_station_name to Top 10
    - Drag trip-distance to the Color Mark.
    - Drag trip_duration to Size Mark.
    - Drag trip_distance to Filters
    - Drag start_station_name to Filters.
    - Drag end_station_name to Filters.
    - Add labels and the title.

IV. DASHBOARDS:

Create at least three dashboards:

1. **Bicycle Ride Characteristics:**
    - Combines visualizations 8 and 12 above
    - Characterizes trips btw. the top 10 starting stations & the top 10 ending stations.
    - Shows the number of rides btw pairs of stations, the distance between where the bicycle is picked up and where it is returned, and the amount of time btw. when the bicycle is picked up and when it is returned
2. **Station Popularity and Usage Dashboard:**
    - Combines visualizations 3 and 4 above
    - Shows how the top 10 ranked starting and ending stations for bicycle rides changes year to year and the number of bicycle rides which start or end at a particular station.
    - Allows comparison of the net flux of bicycles over the course of a year from one station to another and of the change in rankings among starting and ending stations over the course of a year.
3. **Time of Day Dependency of Bicycle Rides:**
    - Combines visualizations 5 and 6 above.
    - Characterizes how ride volume varied by day of week and hour of the day.
    - Shows the consistency of the temporal dependency of ride volume across years and across starting stations.
4. **Growth in eBike Usage:**
    - Combines visualizations 7 and 10 above.
    - Characterizes how ebike usage grew during the first two years during which ebikes were deployed.
    - Shows that both the number and proportion of ebikes used increased between 2022 and 2023.

5. **Types of Consumers:**
    - Combines visualizations 1, 2, and 9.
    - Characterizes the shift from subscriber to individual use across years and across starting stations.
    - Shows a consistent increase in casual or individual customer rentals over subscriber rentals, although they remain less than 15% of rentals.

V. TABLEAU STORIES:

1. Bicycle Ride Usage

- **Top Stations:** A relatively small number of stations consistently rank among the top 10 most popular starting and ending stations across all years. These "hotspots" include locations like Pershing Square North, W 21 St & 6 Ave, 8 Ave & W 31 St, and Broadway & E 14 St. These are likely near major transportation hubs, office buildings, or popular destinations.
- **Station Rank Changes:** While some stations remain consistently popular, there are year-over-year fluctuations in the rankings. This could be due to various factors, such as changes in local businesses, construction, or the addition of new stations nearby.
- **Net Flux:** By comparing the top starting and ending stations, we can infer the net flow of bikes. For example, if a station is consistently a top starting station but not a top ending station, it suggests that bikes are being taken *from* that station to other areas.
- **Key Insight:** Identifying and monitoring the top stations is crucial for ensuring adequate bike availability and managing redistribution efforts. Understanding the net flow between stations helps optimize bike placement.

2. Time Dependency of Bicycle Usage

- **Strong Weekly Pattern:** There's a clear weekly usage pattern. Weekdays (Monday-Friday) have significantly higher ridership than weekends (Saturday-Sunday). This indicates that Citi Bike is heavily used for commuting purposes.
- **Consistent Daily Pattern:** Across all years, there are two distinct peaks in ridership each day: a morning rush hour (around 8-9 AM) and an evening rush hour (around 5-7 PM). This further reinforces the commuting use case.

- **Year-over-Year Growth:** The overall height of the lines increases over the years, showing the general growth in ridership. The shape of the curves (the daily and weekly patterns) remains very consistent.
- **Station-Specific Variations:** When looking at usage by starting station, we see variations in the prominence of the peaks. Some stations have very pronounced rush hour spikes, while others have a more spread-out usage throughout the day. This likely reflects the location of the stations (e.g., near offices vs. residential areas).
- **Key Insight:** Citi Bike usage is highly predictable based on time of day and day of week, driven primarily by commuting patterns. Understanding station-specific temporal patterns can help with bike redistribution and maintenance scheduling.

2. Station Utilization

- **Consistent Top Stations, with Dynamic Ranking:** While a core group of stations consistently rank among the top 10 for both starting and ending points (e.g., "W 21 St & 6 Ave," "8 Ave & W 33 St"), their precise order and the inclusion of other stations fluctuate year-over-year. This highlights a dynamic system responding to changing conditions.
- **Identifying Net Bike Flow (In/Out):** The key comparison is between the starting and ending station charts. Stations appearing significantly higher on one chart versus the other reveal the net directional flow:
    - **Source Stations (High Starting, Low Ending):** More riders are *leaving* these stations than arriving, indicating a net outflow of bikes.
    - **Sink Stations (Low Starting, High Ending):** More riders are *arriving* at these stations than leaving, indicating a net inflow and accumulation of bikes.
- **Overall System Growth:** The increasing height of the bars across the years (particularly post-2019) visually confirms the overall growth in Citi Bike usage during February, consistent with previous findings.
- **Redistribution Optimization Potential:** This visualization's primary value is for optimizing bike *redistribution*. Identifying source and sink stations allows Citi Bike to strategically move bikes from areas of surplus to areas of high demand.
- **Contextual Influences:** Fluctuations in station popularity can be attributed to various factors:

- New station openings or closures.
- Local developments (new buildings, attractions).
- Construction or road closures.
- Special events.
- Weather.

- **Key Insight:** Comparing starting and ending station popularity reveals the net flow of bikes across the network, enabling data-driven decisions for bike redistribution and operational efficiency. The top stations are not static; understanding the dynamics helps anticipate demand.

4. Individual vs Subscription Rides

- **Subscriber Dominance, but Growth in Casual Use:** As seen in the first story, subscribers make up the vast majority of rides. However, the pie chart grid shows a slow but steady increase in the proportion of casual ("Customer") rides over time, particularly at certain stations.
- **Station-Specific Differences:** The proportion of subscriber vs. customer rides varies significantly between stations. Some stations have a much higher percentage of casual users, possibly indicating tourist hotspots or locations near attractions.
- **Key Insight:** While subscribers are the main user base, there's a growing trend of casual use, and this trend is more pronounced at specific stations. This information can be used to target marketing efforts and pricing strategies.

5. Growth in Electric Bicycle Usage

- **Rapid Adoption:** E-bikes were introduced relatively recently (likely around 2020-2021 based on the data), and their usage has grown dramatically. Both the absolute number of e-bike trips and the *percentage* of total trips that are e-bike trips have increased significantly.
- **Significant Proportion:** By 2023, e-bikes account for a substantial portion of total rides.
- **Station-Specific Variations:** The pie chart grid shows that the proportion of e-bike usage varies between stations. Some stations have a much higher adoption rate, perhaps due to factors like terrain (e-bikes are helpful on hills) or the demographics of the area.

- **Key Insight**:
  The deployment of ebikes significantly boosted the number of bicycle rides.

Note: This work can be found on Tableau Public at:

https://public.tableau.com/app/profile/alex.gerwer/viz/citi_bike_data_Visualization_Tableau/BicycleRideUsage