City Bike-Share Usage and Weather Impact: Data Quality Assessment and Cleaning Report

Authors and Contributors: Neha Kulkarni & Rushabh Mehrotra

1. Project Context and Motivation

Urban bike-share systems are a cornerstone of sustainable transportation, but their daily usage patterns are shaped by dynamic factors, chief among them, weather conditions. This project aims to enable evidence-based decision-making for city planners and bike-share operators by assessing how daily weather impacts ridership across four major U.S. cities: **New York City, Chicago, Boston**, and **Washington, D.C.**

Our central research question:

> *How do weather variables like temperature, precipitation, and wind affect daily bike-share ridership across cities?*

To answer this, we developed a reproducible ETL pipeline that performs quality assessment and cleaning of open-access bike-share and weather data. Our goal was to integrate these datasets into a unified, analysis-ready table to support robust, trustworthy insight generation.

---

2. Data Overview and Profiling

2.1 Bike-Share Trip Data

We sourced September 2024 trip records from each city's publicly available datasets:

- **Divvy** – Chicago
- **Citi Bike** – New York City
- **Bluebikes** – Boston
- **Capital Bikeshare** – Washington, D.C.

Each record includes:

- Ride ID, start/end timestamps
- Start/end station names and coordinates
- Rideable type (e.g., classic_bike, electric_bike)
- Membership type (casual/member)

Profiling was done using `ydata_profiling` and `pandas`. Key findings:

- Minor missing values in station names, IDs, and geocoordinates
- Implausible durations (under 60 seconds or exceeding 24 hours)
- Rare but present invalid latitude/longitude values falling outside city boundaries

2.2 Weather Data

We used the **Meteostat** Python API to extract historical daily weather for each city:

- Average, min, max temperature
- Precipitation
- Wind speed

---

3. Data Quality Assessment and Cleaning Workflow

We adopted a **modular pipeline-based workflow** for transparency, repeatability, and scalability. Each stage of the workflow applied relevant data quality principles and techniques.

3.1 Ingestion and Local Storage

All datasets were downloaded to local disk, with raw snapshots stored under `/raw_data/` for reproducibility. Versioning of source files ensures future traceability. Scripts were configured through a shared `config.py`, making ingestion consistent and maintainable.

3.2 Exploratory Profiling and Assessment

We performed extensive exploratory data analysis using:

- `pandas` summaries for nulls, types, and distributions
- `ydata_profiling` to generate interactive HTML reports (`bike_profiles/`) via `scripts/data_profiling.py`

Findings:

Bike trip dataset:

- Trip datasets had **nulls** in station fields and **implausible coordinates**
- We also discovered that a **significant number of trips lacked station names or IDs**. This suggests:
    - Bikes were often picked up or dropped off in non-standard ways (e.g., at flexible docking points, or mid-transfer during rebalancing),
    - Station metadata may not be fully synchronized with trip data in all cities.

    - Despite missing station identifiers, **coordinates were present**, and the trips themselves were otherwise complete and valid.
    - We chose **not to drop these records**, acknowledging their value in usage analysis. Future work could involve inferring the nearest station from start/end coordinates to enrich these entries, though this would require spatial distance computations and assumptions about station proximity thresholds.
- Some rides had **duration < 60 seconds or > 24 hours**

The weather dataset, retrieved using the Meteostat API, was in generally good shape upon profiling. The reports showed:

- No significant missing values in critical fields like `tavg`, `tmin`, `tmax`, or `prcp`
- Realistic and consistent distributions
- Minimal duplication or schema issues

3.3 Cleaning Process and Quality Fixes

The cleaning logic, implemented in Python, addressed the following quality dimensions:

Dimension	Actions Taken
Completeness	Dropped rows missing timestamps, station IDs, or coordinates.
Validity	Removed records with rideable/membership types outside expected categories.
Accuracy	Filtered based on plausible trip duration (60s–24h), validated geolocation bounds.
Uniqueness	Removed duplicate `ride_id` entries.
Consistency	Enforced a common schema across cities, normalized column names and types.

3.3.1 Bike Trip Data Cleaning

Cleaning logic is encapsulated in [`scripts/clean_bike_data.py`], run through `process_all_cities_data()`.

Key Functions and Logic:

1. **Duplicate Removal**:
    - `filter_duplicates()`: Removes duplicate `ride_id` rows
    - Example: ~200 duplicates removed from NYC's raw file
2. **Missing Value Handling**:

    - `filter_missing_values()`: Drops rows missing any of `ride_id`, timestamps, or coordinates
    - Coordinates with `0.0` are replaced with `NaN` beforehand
3. **Geographic Filtering**:
    - `filter_trips_by_geolocation_bounds()` compares trip lat/lng values against city station boundaries
    - Ensures trips are plausible and within the service zone
4. **Duration Filtering**:
    - `filter_trips_by_duration()` filters out rides outside `[1 minute, 24 hours]`
    - Duration is calculated using `started_at` and `ended_at`
5. **Distance Validation**:
    - `filter_trips_by_distance()` uses the **Haversine formula** to compute distance
    - Removes trips with zero or negative distance
6. **Schema Enforcement**:
    - A **Pandera schema** in `process_and_clean_city_data()` ensures:
        - Type safety (`datetime`, `float`, `str`)
        - Value constraints (e.g., `member_casual` must be "member" or "casual")
7. **Output**:
    - Cleaned data is written to `cleaned_data/<city>/<city>_cleaned_trips.csv`

3.3.2 Weather Data Cleaning

Implemented in [scripts/clean_weather_data.py](), executed via `clean_weather_data()`.

Key Cleaning Actions:

- Drops rows missing `date`, `tavg`, or `prcp`
- Removes unrealistic outliers (e.g., `tmax < -100°C`)
- Deduplicates based on `city + date`
- Logs and reports **missing days** in the range for completeness
- Outputs to: `cleaned_data/weather_data.csv`

3.4 Cleaning Techniques Not Applied (Justification)

Some classroom techniques were reviewed but found unnecessary for this domain:

- **Syntactic Error Detection**: Fields were either numeric or from a constrained set. No free-text data requiring regex-based checks.

- **Constraint/Rule-Based Repair**: Domain logic was embedded directly in the script (e.g., filtering by bounding box) rather than using declarative constraints.
- **Probabilistic Repair**: No ambiguous or fuzzy identifiers (e.g., names, text) requiring inference or matching.

---

4. Data Integration and Final Dataset Construction

Final dataset construction was performed in [`scripts/create_final_dataset.py`]

Steps:

1. **Trip Aggregation** via `aggregate_bike_trip_data()`:
    - Computes per-day, per-city:
        - `ride_count`, `member_count`, `casual_count`
    - Grouped using `groupby(['city', 'date'])`
    - Written to `cleaned_data/aggregated_bike_data.csv`
2. **Weather Merge** via `merge_bike_and_weather_data()`:
    - Performs a `left join` on `city + date`
    - Output: `cleaned_data/merged_bike_weather_data.csv`

This table supports comparative modeling across cities and weather conditions.

---

5. Reproducibility and Workflow Automation

Our workflow supports complete reproducibility and modularity:

- **Scripts**: Separated by function (e.g., ingest, clean, merge)
- **Logging**: Captures errors and progress
- **Schema Enforcement**: Validated with Pandera for type and constraint compliance
- **Execution**: A master orchestration script (`workflow_orchestrator.py`) runs the pipeline end-to-end

---

6. Findings, Lessons, and Future Directions

Findings

- Bike-share datasets across cities had similar core schemas, allowing for effective standardization.
- Common data quality issues included duplicate `ride_id`s, missing coordinates or timestamps, and implausible durations.
- Zero-distance trips (same start/end points) were frequent and required filtering using Haversine calculations.
- Station metadata was critical for defining realistic geographic bounds, outperforming static map-based coordinates.
- Meteostat provided accessible, city-level weather data without quota or payment constraints, unlike many commercial APIs.
- Profiling tools (`ydata_profiling`, `pandas`) were effective for quickly identifying data gaps, schema mismatches, and edge cases.
- Modular scripts and `pandera` schema checks helped enforce consistency and catch errors early.

Challenges

Throughout the project, we encountered several realistic and impactful challenges that shaped our workflow:

- **Determining how to define valid trip boundaries**:
  Initially, we attempted to use rough bounding coordinates found via Google searches (e.g., downtown coordinates + radius) to filter out geographically implausible trips. However, we quickly realized this approach was both too generic and potentially inaccurate for each city's true bike-share coverage. For example, a station in an outer borough of NYC might fall outside a naive bounding box but still be valid. Instead, we used the **actual bounding box derived from station metadata** (`min/max lat/lng` across all stations) to define valid service areas. This approach was more grounded in real operational data and ensured we only removed truly out-of-scope records.
- **Identifying a usable weather data source**:
  Many commonly known APIs (e.g., OpenWeatherMap, AccuWeather) were either paywalled, limited by quota, or required precise latitude-longitude inputs for each record—making integration with aggregated city-level data cumbersome. After evaluating options, we chose **Meteostat**, an open-source Python package that allowed fetching city-level daily weather data based on station IDs. It was free, had consistent historical data, and integrated easily into our pipeline—helping us avoid licensing or scale-related limitations.
- **Handling the volume and variability of raw data**:
  Especially in the case of New York City, the raw trip data for even a single month could exceed hundreds of thousands of records. Schema inconsistencies (e.g., different

column names, extra metadata) across cities and files also complicated ingestion. To manage this, we limited ingestion to **only the first file for NYC and focused on September 2024** as a representative pilot month. This allowed us to prototype the full pipeline before scaling to larger timeframes.

Lessons Learned

- Early profiling saves time—issues like geographic outliers or duplicate IDs can snowball if not handled first.
- Modular cleaning enables flexibility and reuse as more cities or time periods are added.

Future Work

- **Ingest More Data**: Expand to full-year or multi-year datasets.
- **Cloud Migration**: Use AWS S3 for scalable storage and querying.
- **Airflow Integration**: Automate scheduling and monitoring.
- **Live Feeds**: Extend pipeline to work with streaming weather or trip updates.
- **Infer Missing Station Data**: Estimate missing station names using start/end coordinates and nearest-station logic.
- **Enhance Geospatial Validation**: Use refined city boundaries or shapefiles to improve filtering of invalid trips.
- **Scale Cleaning Scripts**: Generalize cleaning functions to support more cities and larger timeframes.
- **Add Cleaning Logs**: Record row-level actions (e.g., dropped, corrected) for better traceability and quality auditing.

Team Contributions

Rushabh developed scripts to acquire and profile the bike-share trip data. Neha built scripts for acquiring and profiling station and weather data. Cleaning strategies were discussed jointly based on profiling results, with Neha implementing the cleaning workflows through modular scripts. Rushabh created the aggregation script for summarizing daily ride metrics, and Neha developed the final integration script to combine trip and weather data into a single, analysis-ready dataset.

---

7. Supplementary Materials & References

- **Code & Docs**: https://github.com/neha181298/bikeTripAnalytics/tree/main
- **Raw/cleaned data samples** in `/raw_data/` and `/cleaned_data/`
- **Interactive profiling reports**: `/bike_profiles/`
- **Meteostat Documentation:** https://dev.meteostat.net/python
- **NYC Open Data Portal**: https://www.nyc.gov/html/dot/html/bicyclists/bikeshare.shtml

- **Chicago Bike Data**: https://www.divvybikes.com/system-data
- **Boston Bike Data**: https://www.bluebikes.com/system-data
- **Capital Bikeshare Data**: https://www.capitalbikeshare.com/system-data