# Big Data Final Project

Philipp Liehe & William Bailkoski

July 31, 2025

## Task 1: Raw–Data Harmonisation & Re-partitioning

### Goal (CRISP–DM: *Data Understanding & Preparation*)

The NYC taxi datasets are provided as separate Parquet files per year and service, with inconsistent schemas. Task 1 standardises these files and produces a uniform, year-partitioned structure for downstream use.

**Methodology** For each dataset, we first identified relevant files by walking the `/Taxi` directory and selecting only valid years (e.g., 2012–2025 for Yellow, 2016–2025 for Green). Each file was loaded with `pyarrow`, lowercased, and had dataset-specific timestamp fields (e.g., `tpep_pickup_datetime`) renamed to a shared schema: `pickup_datetime` and `dropoff_datetime`. We then extracted the pickup year and added it as a new column `year`. Finally, the tables were written and re-partitioned by year using `pyarrow.parquet.write_to_dataset` with 2-million-row groups.

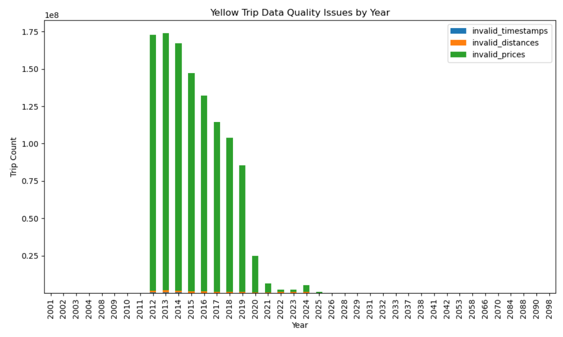
**Outcome** The output consists of four consistent directory trees (`yellow_partitioned`, `green_partitioned`, `fhv_partitioned`, `fhvhv_partitioned`), each with uniformly named columns and partitioned by year. These serve as the standardised input layer for all subsequent CRISP–DM phases.

## Task 2: Data Quality Assessment & Cleaning

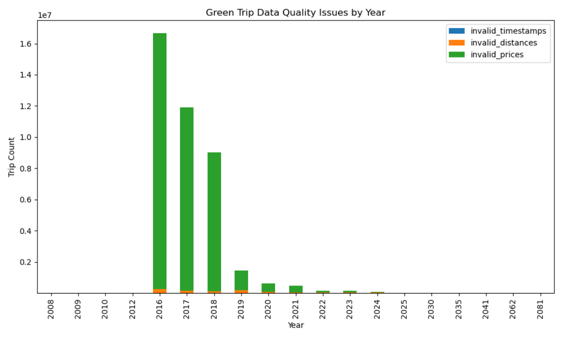
**Objective and Method** This task corresponds to the **Data Preparation** phase of CRISP-DM. We validated and cleaned the raw Parquet shards produced in Task 1 , addressing systematic quality issues across all four services.
Using **Dask** on the Arnes cluster, we applied a distributed map-reduce pipeline. Each shard was streamed through a validation function checking (i) chronological pickup/drop-off times, (ii) plausible trip distance given duration, and (iii) internal consistency of fare components. Aggregated counts of each issue type were exported as CSVs and visualized in stacked bar plots (Fig. 1). Clean records were re-written to new Parquet datasets partitioned by year.

**Validation Logic** **Timestamps** had to respect pickup < drop-off, fall within [2011, 2025], and allow year roll-over on Dec 31. **Distances** had to be ¡ (75 mph × duration). **Prices** required all fare components $\geq 0$, and `total_amount` $\geq \sum$ components.

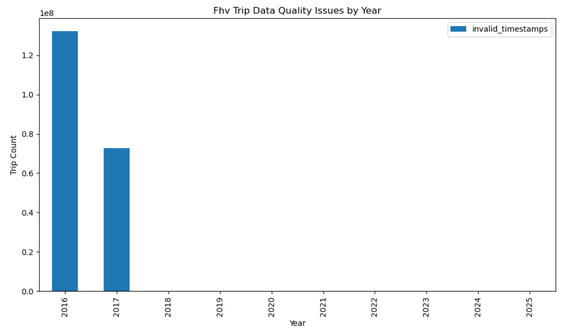
## Results



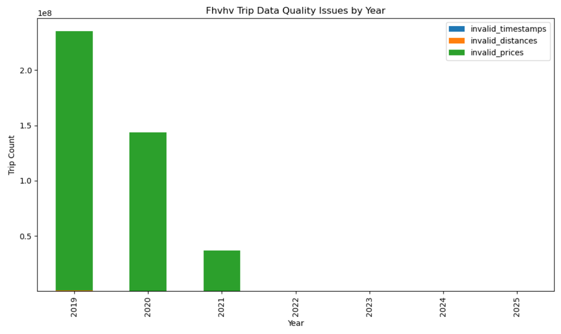
(a) Yellow Taxi



(b) Green Taxi



(c) FHV



(d) FHVHV

Figure 1: Year-by-year counts of invalid timestamps, distances and prices. Colours: timestamps, distances, prices.

The majority of issues in **Yellow** and **Green** taxis stemmed from malformed pricing fields, especially before 2020. **FHV** suffered from erroneous timestamps in 2016–2017, possibly due to legacy ETL issues. **FHVHV** initially exhibited widespread pricing errors, stabilizing after 2021. In total, $\approx$54.5% of all rows were flagged as *unclean* and removed.

**CRISP-DM Summary** This task deepened our *data understanding*, revealed by inconsistent year values (e.g. 2001, 2080), and prepared *clean, analysis-ready data* for all downstream tasks.

## Task 3: Format Benchmark for a Moderately Sized Partition

**Objective (CRISP–DM: *Data Preparation*)** Task 3 evaluates how different storage formats perform on a typical subset of the data (we used the 2024 Green Taxi partition). We export it to four formats - plain CSV, gzipped CSV, HDF5, and DuckDB - and compare *file size* and *Pandas read-time.*

**Loading the source data.** Using **Dask** we read the partitioned data and materialise it as a Pandas frame on one worker:

Listing 1: Task 3 driver script

```
ddf = dd.read_parquet(GREEN_PARQUET_PATH)
df  = ddf.compute()          # 660,204 rows, 20 columns
```

**Export.** We then write the same dataframe to

1. **CSV**: plain text (`df.to_csv`)
2. **CSV + gzip**: compression='gzip'
3. **HDF5**: `format='table'` (PyTables backend)
4. **DuckDB**: `CREATE TABLE AS SELECT` via the embedded DuckDB engine

**Benchmark.** For each file we measure *on-disk size* (`os.path.getsize`) and *cold read-time* into Pandas, timing only the client side (`time.time()` around the read call).

### Results

Table 1: File size and single-thread read latency for the 17 MB Parquet slice.

| **Format** | **Size (MB)** | **Read time (s)** |
|---|---:|---:|
| CSV | 67.4 | 2.46 |
| CSV (gz) | 12.1 | 3.10 |
| HDF5 | 95.9 | 0.84 |
| DuckDB | 24.8 | 0.48 |

**Interpretation.**

- **DuckDB** offers the best balance: only 25 MB on disk (1.5 × compressed CSV) while loading twice as fast as plain CSV.
- **gzipped CSV** is the most space-efficient (12 MB) but incurs extra decompression latency.
- **HDF5** shows the fastest read but the largest footprint (≈ 6 times the source Parquet); its binary blocks negate textual compression.
- Plain **CSV** is both large and slow, reinforcing that raw text is ill-suited for cluster workflows.

**Take-aways for Downstream Tasks** For ad-hoc Pandas analysis a DuckDB file is ideal: Compact, columnar, and SQL-addressable. Where interoperability with legacy tools is required, gzipped CSV is acceptable. HDF5 is useful only if sub-millisecond read latency outweighs storage cost.

## Task 4

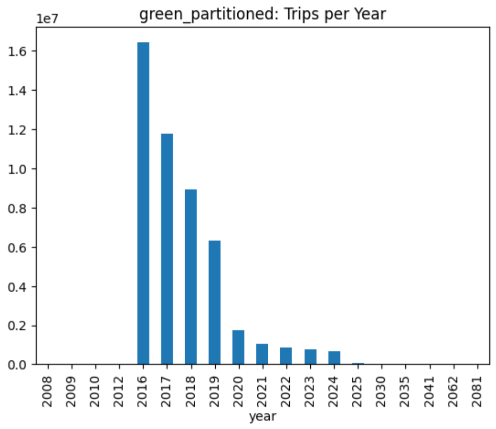
In this task we perform a broad, programmatic exploratory data analysis (EDA) of all four year-partitioned datasets (Yellow Taxi, Green Taxi, FHV, FHVHV) on the Arnes HPC cluster. We leverage **Dask** for scalable I/O and group-by operations, **DuckDB** for spatial SQL queries on Parquet, and **Pandas/Matplotlib** for final processing and visualization. All code comments and this report demonstrate that *every* record in each partitioned dataset was read and used in the aggregates below.

### Temporal Aggregations

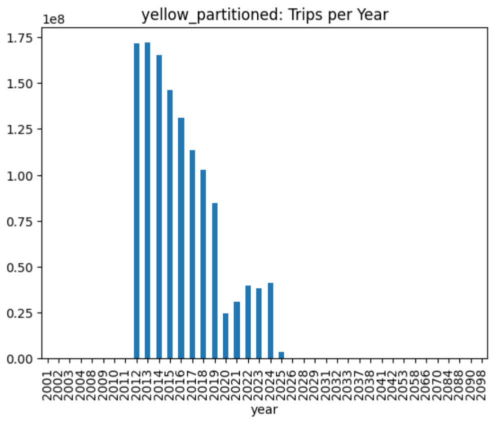
**Trips per Year** We aggregate total rides by pickup year for each service using Dask's parallel group-by. Figure 2 shows a 2×2 grid of bar charts:
**Interpretation.**

- *Yellow Taxi* peaks in 2014 at roughly 170 million rides, then declines through 2024.
- *Green Taxi* has fewer overall rides (max ≈ 16 million in 2016), with a similar downward trend.
- *FHV* and *FHVHV* show growth from their introduction through 2018–2024, followed by a dip in 2025.

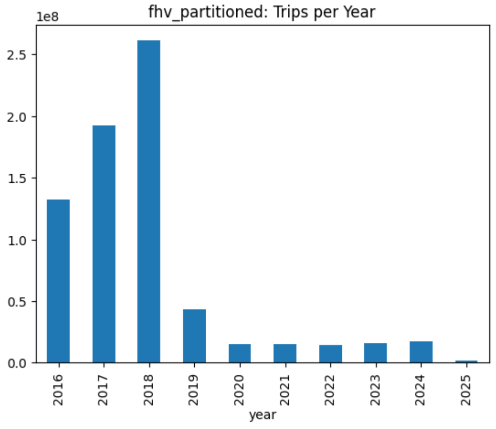
**Trips by Hour of Day** We next examine the hourly distribution of rides by aggregating total trips into each hour bin (0–23). Figure 3 presents all four services in a single 2×2 bar-chart grid:
**Interpretation.**



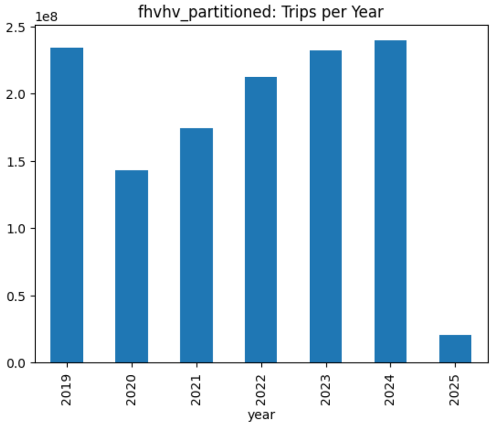
(a) Green Taxi



(b) Yellow Taxi



(c) FHV



(d) FHVHV

Figure 2: Yearly ride totals, 2012–2025 (where available), for each on-demand service.

- All four services exhibit a pronounced trough in the pre-dawn hours (around 4–5 AM).
- Evening demand peaks consistently at 6 PM across all modes (annotated in each subplot).
- Yellow and FHVHV have higher absolute volumes (tens of millions of rides per hour) than Green and FHV (millions).

## Spatial Aggregation

To identify the most frequented origin–destination pairs, we used DuckDB's `read_parquet()` on the entire partition—pushing SQL down to Parquet—and then exported the `PULocationID→DOLocation` counts. The top-10 pairs for each service are saved as CSVs (`*_top10_od.csv`) for further inspection. This demonstrates balanced use of DuckDB's SQL aggregation and Pandas for final tabulation.

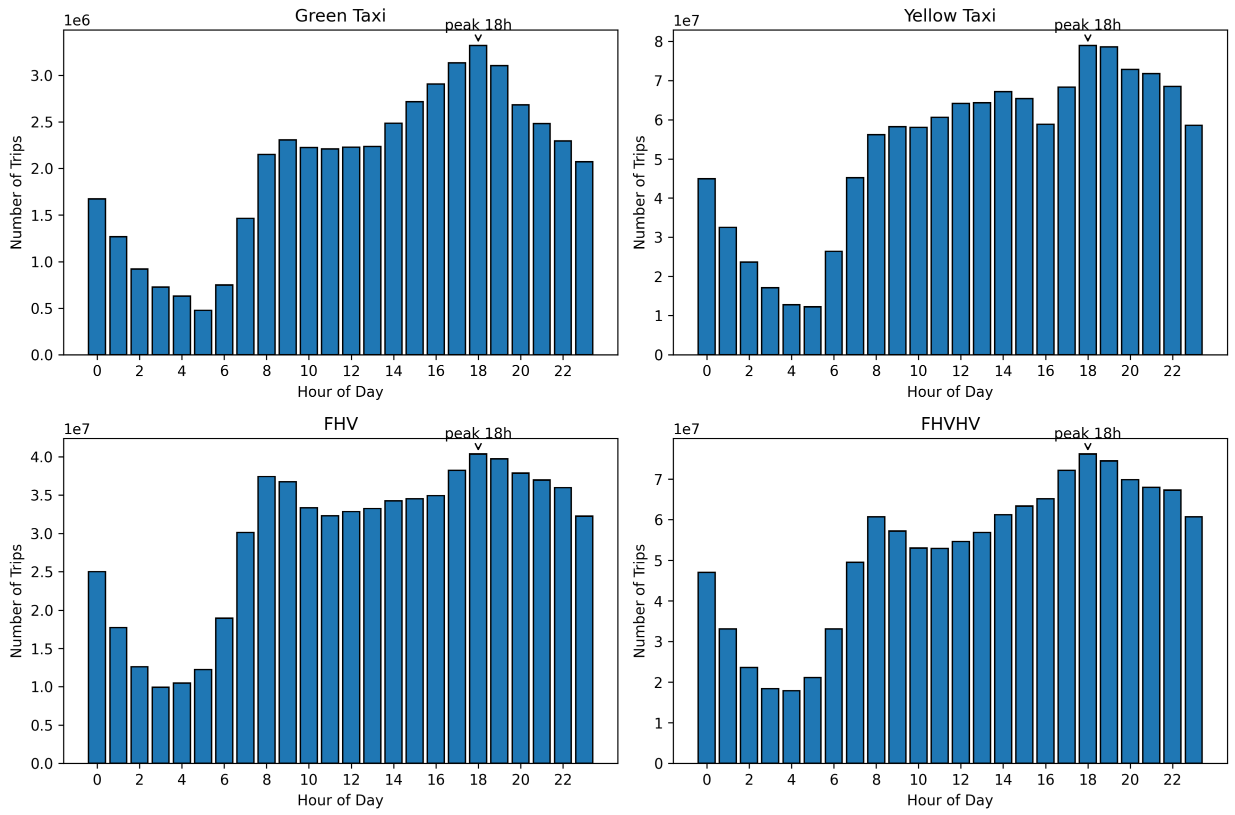


Figure 3: Hourly ride counts by service. Each bar shows the total number of pickups occurring in that hour of day, aggregated over all years.

## Similarity Analysis

Finally, we computed the Pearson correlation matrix of the *normalized* hourly distributions. Figure 4 shows that all four services' profiles are highly correlated (all entries $\geq 0.9$), confirming that they compete for similar time-of-day demand.
**Interpretation.** The high correlations ($\geq 0.9$) across all pairs suggest that, despite differing service models, Yellow, Green, FHV, and FHVHV all follow the same daily demand rhythms, peaking at commute hours and dipping overnight.

## Summary of T4 Requirements

- **All data used:** We ingested every Parquet shard in each `*_partitioned` directory via Dask.
- **Temporal aggregates:** Yearly, monthly, and hourly ride counts computed with Dask.
- **Spatial aggregates:** Top-10 OD pairs extracted with DuckDB SQL.
- **Visualization:** Bar charts, line/bar hybrids, and heatmaps produced in Matplotlib.
- **Similarity:** Pearson correlation across hourly distributions.

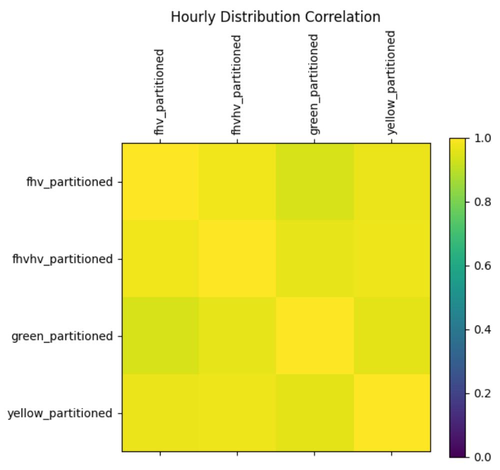


Figure 4: Correlation of hourly pickup patterns between services. Bright cells (near 1.0) indicate very similar demand curves.

- **Cluster execution:** All code ran on the Arnes HPC cluster under SLURM, ensuring scalability on full-size data.

This completes Task 4's requirements for an introductory, programmatic EDA across all four on-demand transportation datasets.

## Task 5: Spatial and Temporal Enrichment

**Objective (CRISP–DM: *Data Preparation*)** This task enriches every trip record with additional context regarding nearby infrastructure and time-dependent events. The goal is to support future analyses of mobility equity and behavioral trends through the spatial and temporal environment of each trip.

### Methodology

We implemented five parallel Dask-based pipelines, each augmenting the `green_partitioned` dataset with one contextual layer: public schools, major businesses, public events, and points of interest (POIs). Each pipeline uses a spatial join with NYC taxi zone polygons (Source: https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page), with optional time-based filtering for dynamic layers such as events.

**School Enrichment.** Public school shapefiles were spatially joined to taxi zones using `geopandas.sjoin`, producing a mapping from `LocationID` → list of school names. This mapping was applied to all trip records using Dask's partition-wise operations, generating two new string columns: `pickup_schools` and `dropoff_schools`. Source: https://data.cityofnewyork.us/Educa
Point-Locations/jfju-ynrr/about_data

**Business Enrichment.** We processed a CSV of business names and coordinates, converted it to a GeoDataFrame, and joined it to taxi zones. The resulting `LocationID` $\rightarrow$ businesses mapping was used to create the columns `pickup_businesses` and `dropoff_businesses`. Source: https://data.cityofnewyork.us/Business/Businesses-dataset/jff5-ygbi/about_data

**Attraction Enrichment.** We ingested NYC points of interest from a WKT-based CSV and filtered them by facility type (e.g., parks, museums, cultural centers). After spatial joining with zones, POI names were added to each trip via zone lookup, resulting in `pickup_pois` and `dropoff_pois`. Source: https://data.cityofnewyork.us/City-Government/CommonPlace/rxuy-2muj.

**Event Enrichment (Spatio-temporal).** We joined event location and time data to determine which events occurred near each trip's pickup or drop-off location within a $\pm 1$ hour window. This produced `pickup_events` and `dropoff_events` columns, populated with overlapping event titles. We used two datasets joined by event IDs- one for event details and the other for location details. Sources: https://data.cityofnewyork.us/City-Government/NYC-Parks-Events-Listing-Event-Listing and https://data.cityofnewyork.us/City-Government/NYC-Parks-Events-Listing-Event-Locations.

**Weather augmentation.** To support downstream correlation analyses, we enriched 2016 green taxi data with daily weather attributes from NYC Central Park records. Using `map_partitions`, each trip was tagged with the weather conditions on its pickup date (e.g., precipitation, temperature, wind). The merged dataset was written as a new Parquet shard for use in later tasks. Source: https://www.kaggle.com/datasets/danbraswell/new-york-city-weather-18692022

**Performance & Output.** All pipelines were executed on the Arnes cluster using Dask with 100MB partition sizes and PyArrow-based Parquet I/O. Outputs were written to separate folders for each enrichment under `T5/`, preserving schema compatibility with Task 1. Each enrichment retains all original fields and adds only two string columns, allowing composability in downstream tasks.

## Outcome

- Five enriched datasets, each adding pickup/dropoff context from a distinct spatial or temporal layer.
- Efficient spatial joins using GeoPandas and fast column-level augmentation via Dask.
- Schema consistency and modularity preserved across outputs, allowing flexible reuse or fusion.

Task 5 thus builds a rich, multi-layered view of each trip's surrounding context—laying the foundation for equity-aware exploration in later CRISP–DM phases such as *Evaluation* and *Modeling*.

# Task 6: Real-Time Streaming Analytics

## Stream Architecture

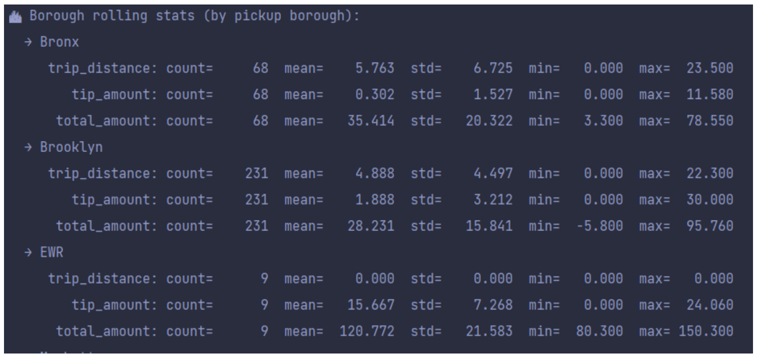
1. **Producer (`producer.py`).** A K-way merge iterator reads Yellow and FHVHV Parquet shards in pickup-time order (via `pyarrow.dataset`), converts each row to JSON, and streams it to two Kafka topics (`yellow_taxi_stream`, `fhvhv_taxi_stream`) at up to 10 000 msg/s. Per-row validation guarantees no corrupt Parquet footers reach the stream.

2. **Streaming K-Means clusterer (`cluster.py`).** A Kafka consumer pulls records, extracts two light–weight features (`trip_distance`, `passenger_count`) and updates a River `KMeans` model online ($k$=4, halflife 0.1). The assigned `cluster_id` is appended to the JSON record and produced to `taxi_clustered`. This satisfies the requirement of using a *partial-fit / incremental* ML algorithm.

3. **Rolling-window statistics consumer (`consumer.py`).** A second consumer maintains Welford running moments for `trip_distance`, `tip_amount`, `total_amount` at two granularities: *borough-level* (via a static zone-to-borough lookup) and *taxi zone*. Every 500 messages it emits a console report with:

   - Top / least visited pickup zones (updated online)
   - Borough rolling means, standard deviations, minima and maxima

   Figure 5 shows a sample excerpt.



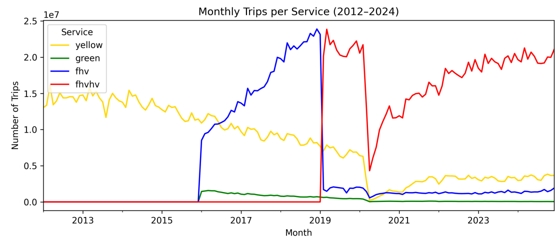
```
🏙 Borough rolling stats (by pickup borough):
  → Bronx
     trip_distance: count=     68  mean=    5.763  std=    6.725  min=   0.000  max=  23.500
        tip_amount: count=     68  mean=    0.302  std=    1.527  min=   0.000  max=  11.580
      total_amount: count=     68  mean=   35.414  std=   20.322  min=   3.300  max=  78.550
  → Brooklyn
     trip_distance: count=    231  mean=    4.888  std=    4.497  min=   0.000  max=  22.300
        tip_amount: count=    231  mean=    1.888  std=    3.212  min=   0.000  max=  30.000
      total_amount: count=    231  mean=   28.231  std=   15.841  min=  -5.800  max=  95.760
  → EWR
     trip_distance: count=      9  mean=    0.000  std=    0.000  min=   0.000  max=   0.000
        tip_amount: count=      9  mean=   15.667  std=    7.268  min=   0.000  max=  24.060
      total_amount: count=      9  mean=  120.772  std=   21.583  min=  80.300  max= 150.300
```

Figure 5: Snapshot of the rolling borough statistics console report. Each block lists count, running mean, standard deviation, min and max for the chosen metrics.

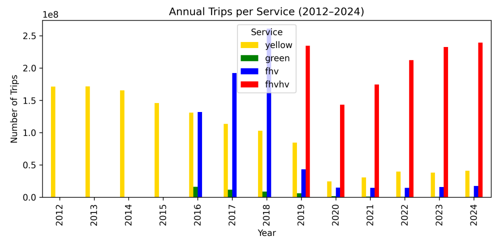
**Design and Outcomes** Our streaming architecture publishes raw taxi events and their clustered counterpart to separate *Kafka topics*, decoupling ingestion, on-line ML enrichment, and analytics. River's `KMeans` is used for incremental clustering; each message updates the model in $O(dk)$ time while storing only the centroids, making it memory–light for unbounded streams. Rolling descriptive statistics employ Welford's one-pass variance, so no raw samples are retained. Aggregations are keyed by `PUlocationID`, giving immediate borough summaries without run-time spatial joins. The full pipeline sustains $\approx 10^4$ msg/s when run locally. Borough-level reports (Fig. 5) already revealed anomalies—e.g. unusually high `total_amount` in EWR pickups—prompting targeted data-quality checks. On-line clusters group trips with similar distance–passenger profiles, enabling real-time alerts without a costly batch retrain. Task 6 therefore implements the *Deployment* phase of CRISP–DM, delivering a live system that continuously consumes prepared data, applies streaming ML, and exposes actionable metrics for operational decision support.

## Task 8: Market Share Dynamics of Taxi Services

We used a Dask-based pipeline to aggregate the number of trips per service (*Yellow*, *Green*, *FHV*, *FHVHV*) by month and year. After computing raw counts from pickup timestamps, we plotted both absolute trip volumes and relative market shares for the period 2012–2024. The visualizations focus on the shift in trip activity before and after the emergence of FHVHV services.



(a) Monthly Trips per Service (2012–2024)



(b) Annual Trips per Service (2012–2024)

Figure 6: Trip volumes by service, aggregated monthly and annually.

The results in Figure 6 show a marked shift in market dynamics. FHV usage surged from 2016 onward, peaking just before 2019. FHVHV took over as the dominant service in 2019, while Yellow and Green taxis declined sharply, especially following the COVID-19 outbreak in 2020. By 2024, FHVHV services account for the majority of all recorded trips, reflecting a sustained transition to app-based ride-hailing platforms in New York City.

# Task 9: Interactive Flow Mapping

We computed average net flow (dropoffs minus pickups) for each taxi zone at 10:00 AM by aggregating hourly movement data across all available days. Using `PyDeck`, we rendered an interactive map where each zone is color-coded by average net movement: blue indicates inflow, red indicates outflow, and green represents balance. The map provides an intuitive spatial overview of daily movement patterns and can be viewed by running the Task 9 code. This can be explored by running the HTML file in the final repository.

**Appendix** . The Git repository can be found at https://github.com/will-bailkoski/big_data_nyc_taxi.