RouteSavvy

Krapa Karthik - kk5754
Shreyansh Bhardwaj- sb10261
Sourik Dutta - sd5913
Dev Thakkar - djt8795

Problem Statement

Commuters in NYC often face delays and overcrowding on the subway, with little visibility into real-time crowd conditions. Existing apps don’t account for station-level congestion. Although the MTA publishes detailed ridership data, it isn’t used effectively for live route optimization — creating an opportunity to build smarter, data-driven transit recommendations.

Project vision and mission

RouteSavvy aims to transform the daily commute by using real-time insights from MTA ridership data. Our mission is to make urban travel smarter and less crowded by delivering route recommendations that respond to changing station traffic — helping people get where they need to go, faster and more comfortably.

01. To revolutionize urban commuting by making it more intelligent, adaptive, and comfortable through data-driven insights into transit congestion.

02. To develop a scalable big data pipeline that processes hourly MTA ridership data and delivers optimized subway routes in real time via APIs.

03. To enhance commuter satisfaction by minimizing time spent in crowded stations and improving route reliability using predictive analytics.

Dataset Overview

Source

NY State Open Data Portal – Socrata API

https://data.ny.gov/Transportation/MTA-Subway-Hourly-Ridership-2020-2024/wujg-7c2s/about_data

Timeframe

January 2020 - December 2024

Covers subway ridership trends across five years, including pre-pandemic, pandemic, and post-pandemic periods — useful for longitudinal transit analysis.

Granularity

Hourly records per subway station complex

Each data point represents one hour of ridership at a subway station complex, which can serve multiple train lines. This level of detail helps capture commuter flow patterns throughout the day.

Volume

The dataset comprises over 100 million records from January 2020 to December 2024. With approximately 60,000 entries per day, the data's scale necessitates the use of streaming and distributed processing tools like Kafka and PySpark.

Dataset Attributes

What Each Record Tells Us

transit_timestamp
The exact f-hour the ridership data was recorded – allows time-based fitering and aggregation.

station_complex
The name of the subway station complex (e.g., 14 St-Union Sq), often serving multiple train lines

borough
Indicates which borough the station is located in - Manhattan, Brooklyn, Queens, Bronx, or Staten Island

payment_method
Whether the fare was paid using MetroCard, OMNY, etc.

Indicates which borough the station is located in ·· Manhattan, Brooklyn, Queens, Bronx, or Staten Island

ridership
Total number of entries recorded at that station in that hour

latitude, longitude
Geographic coordinates- useful for mapping ridership trends spatially

fare_class_category
Type of fare used (e.g., Full Fare, Student, Senior, Fair Fare)

System Architecture

Pipeline Overview

We simulate live subway activity by publishing hourly ridership data from the MTA dataset into Kafka topics using Python. Confluent Kafka is used for managed stream orchestration.

We use **Sodapy** to connect to the **Socrata Open Data API** and fetch hourly MTA ridership data programmatically. This allows efficient and reliable data pulls directly from the New York State open data portal.

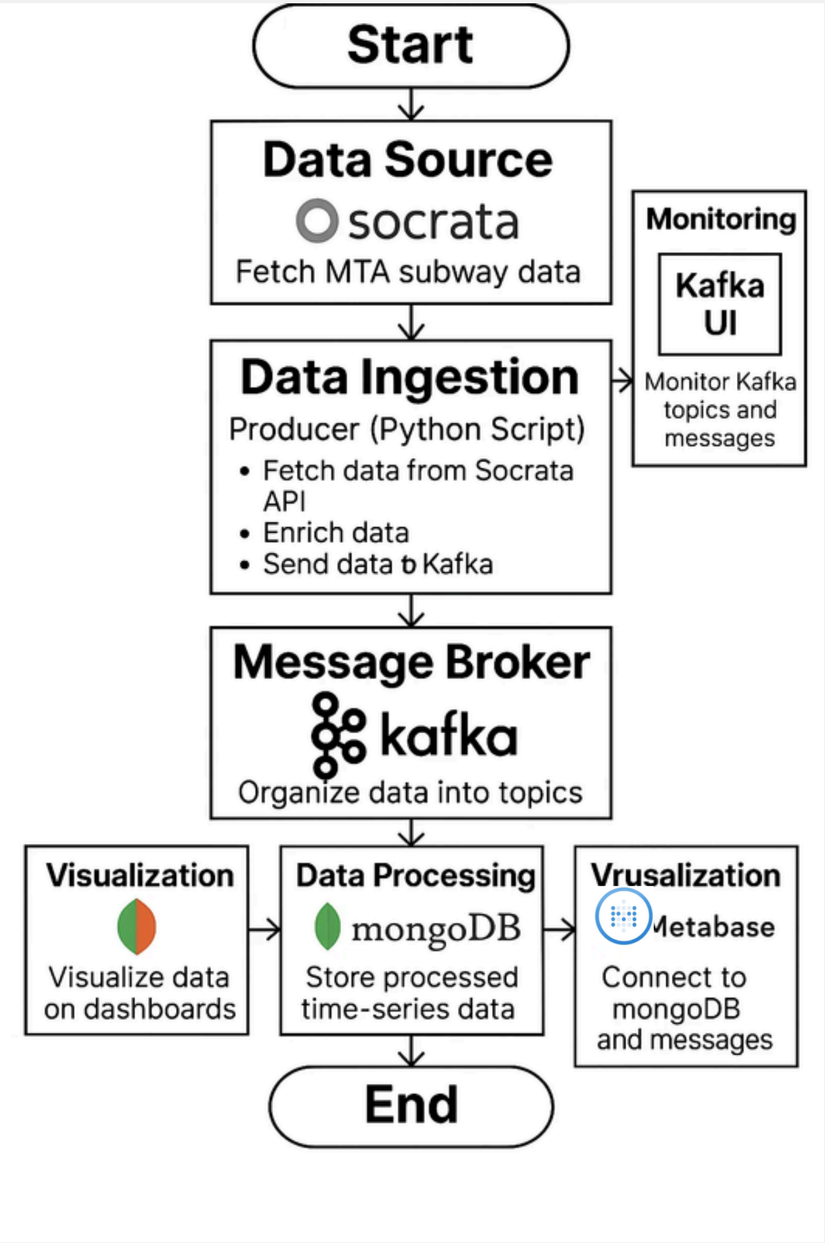
Spark Structured Streaming reads from Kafka and transforms data into crowd-level insights. Aggregations are performed over sliding windows to simulate live congestion detection.

Jupyter Notebooks served as our sandbox environment for simulating real-time data queries, visualizing subway ridership patterns using Pandas and Matplotlib, and validating API responses. It enabled rapid iteration and clear presentation of trends like peak hours, top stations, and rider flow across the day.

MetaBase was integrated to visualize metrics such as message flow rates, consumer lag, and Kafka topic health. It had free mongodb integration.

We used Docker to containerize the entire streaming stack — including Kafka, Spark, and data producers. This ensured consistency across environments, simplified deployment, and made local testing much easier.

How RouteSavvy Works



MongoDB Compass - localhost:27017/mta_data.subway_data_clean3

Compass

My Queries

CONNECTIONS (2)

Search connections

- kk5754-mac-mongo
- localhost:27017
  - admin
  - config
  - local
  - mta_data
    - subway_data_clean
    - subway_data_clean2
    - subway_data_clean3
  - urban_mobility

mta_data | subway_data_clean3

localhost:27017 > mta_data > subway_data_clean3

Open MongoDB shell

Documents 416.3K | Aggregations | Schema | Indexes 1 | Validation

Type a query: { field: 'value' } or Generate query

Explain | Reset | Find | Options

ADD DATA | EXPORT DATA | UPDATE | DELETE

25 | 1 - 25 of 497152

```
borough : "manhattan"
transit_mode : "subway"
transit_timestamp : 2024-05-01T00:00:00.000+00:00
year : 2024
month : 5
day : 1
hour : 0
ridership : 2
transfers : 0
transfer_ratio : 0
latitude : 40.79164
longitude : -73.9647
peak_hour : 0
```

```
_id: ObjectId('68227735f238156910d9f379')
station_complex_id : "240"
station_complex : "7 av (f,g)"
borough : "brooklyn"
transit_mode : "subway"
transit_timestamp : 2024-05-01T00:00:00.000+00:00
year : 2024
month : 5
day : 1
hour : 0
ridership : 22
transfers : 0
transfer_ratio : 0
latitude : 40.66627
```

```
{
  "transit_timestamp": "2024-05-01T00:00:00.000",
  "transit_mode": "subway",
  "station_complex_id": "336",
  "station_complex": "Hoyt St (2,3)",
  "borough": "Brooklyn",
  "payment_method": "metrocard",
  "fare_class_category": "Metrocard - Full Fare",
  "ridership": "3.0",
  "transfers": "0.0",
  "latitude": "40.690544",
  "longitude": "-73.98506",
  "georeference": {
    "type": "Point",
    "coordinates": [
      -73.98506,
      40.690544
    ]
  },
  "year": 2024,
  "month": 5,
  "day": 1,
  "hour": 0,
  "peak_hour": 0,
  "transfer_ratio": 0.0,
  "cumulative_ridership": 145.0
}
```

Search

New

Home

How to use Metabase

COLLECTIONS

Our analytics

Your personal collection

Examples

BROWSE

Models

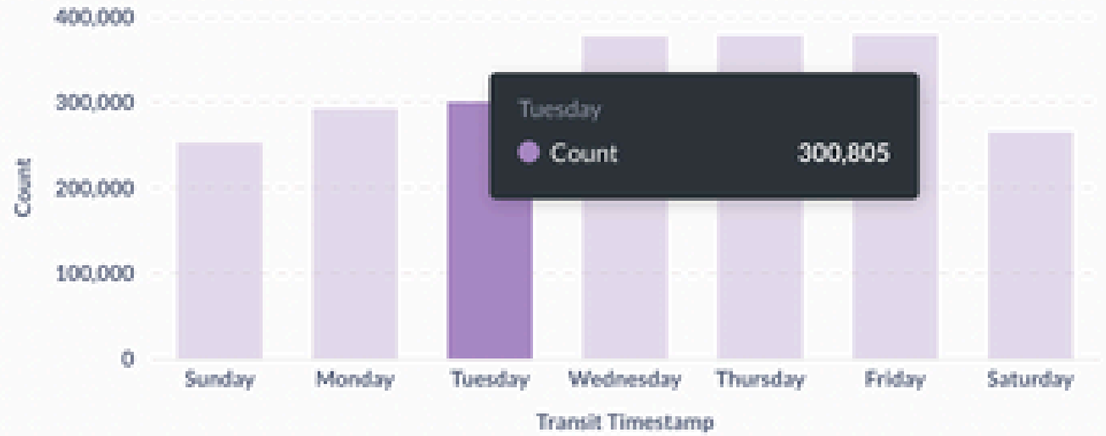
Databases

Metrics

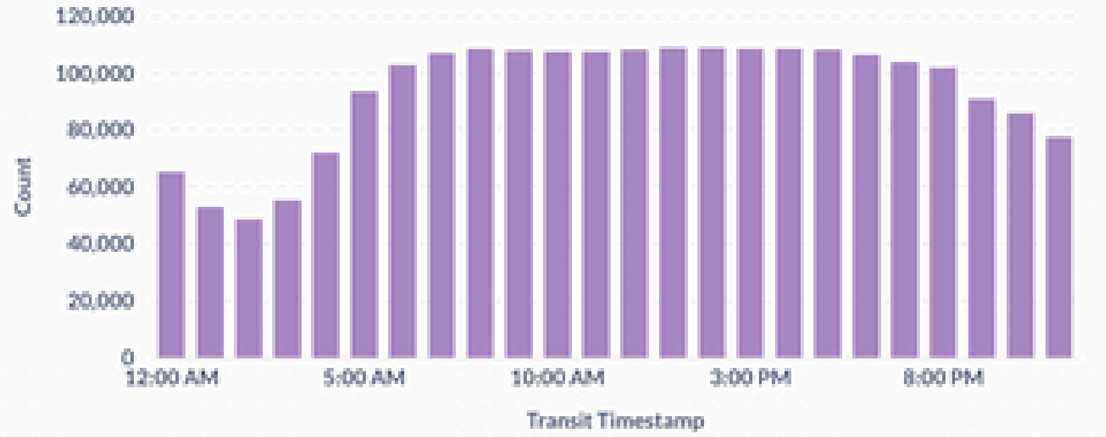
Trash

These Subway Data Clean2 across time

Transit Timestamp by day of the week



Transit Timestamp by hour of the day



Transit Timestamp by month of the year

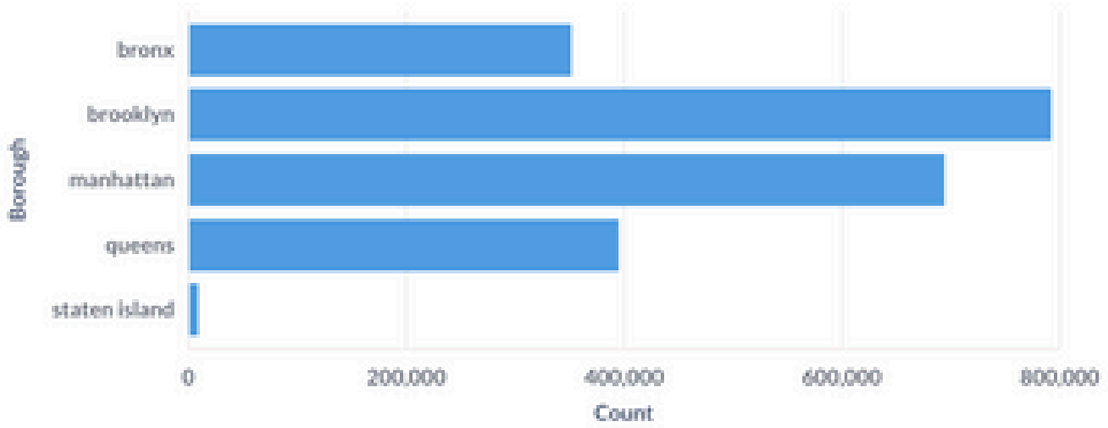
Home
How to use Metabase

COLLECTIONS
Our analytics
Your personal collection
Examples

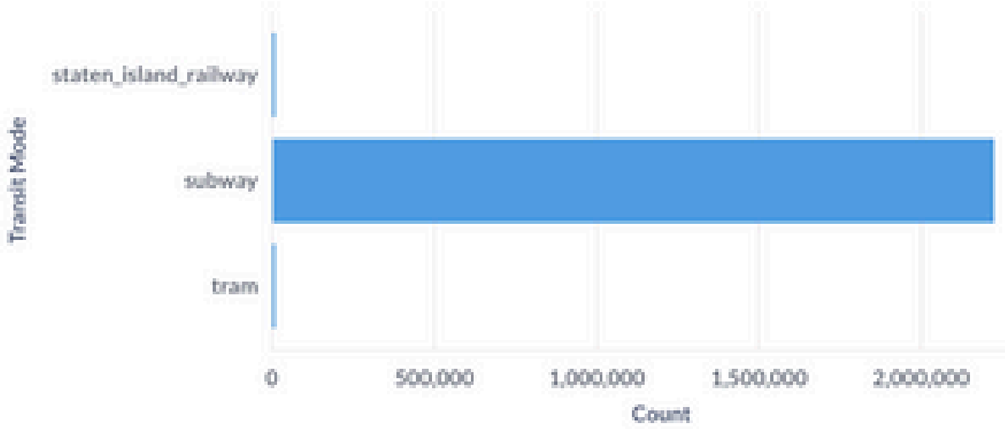
BROWSE
Models
Databases
Metrics

Trash

Subway Data Clean2 per Borough



Subway Data Clean2 per Transit Mode



These are the same for all Subway Data Clean2

Spark Master at spark://192.168.128.4:7077

URL: spark://192.168.128.4:7077
Alive Workers: 2
Cores in use: 2 Total, 2 Used
Memory in use: 6.0 GiB Total, 2.0 GiB Used
Resources in use:
Applications: 1 Running, 20 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (2)

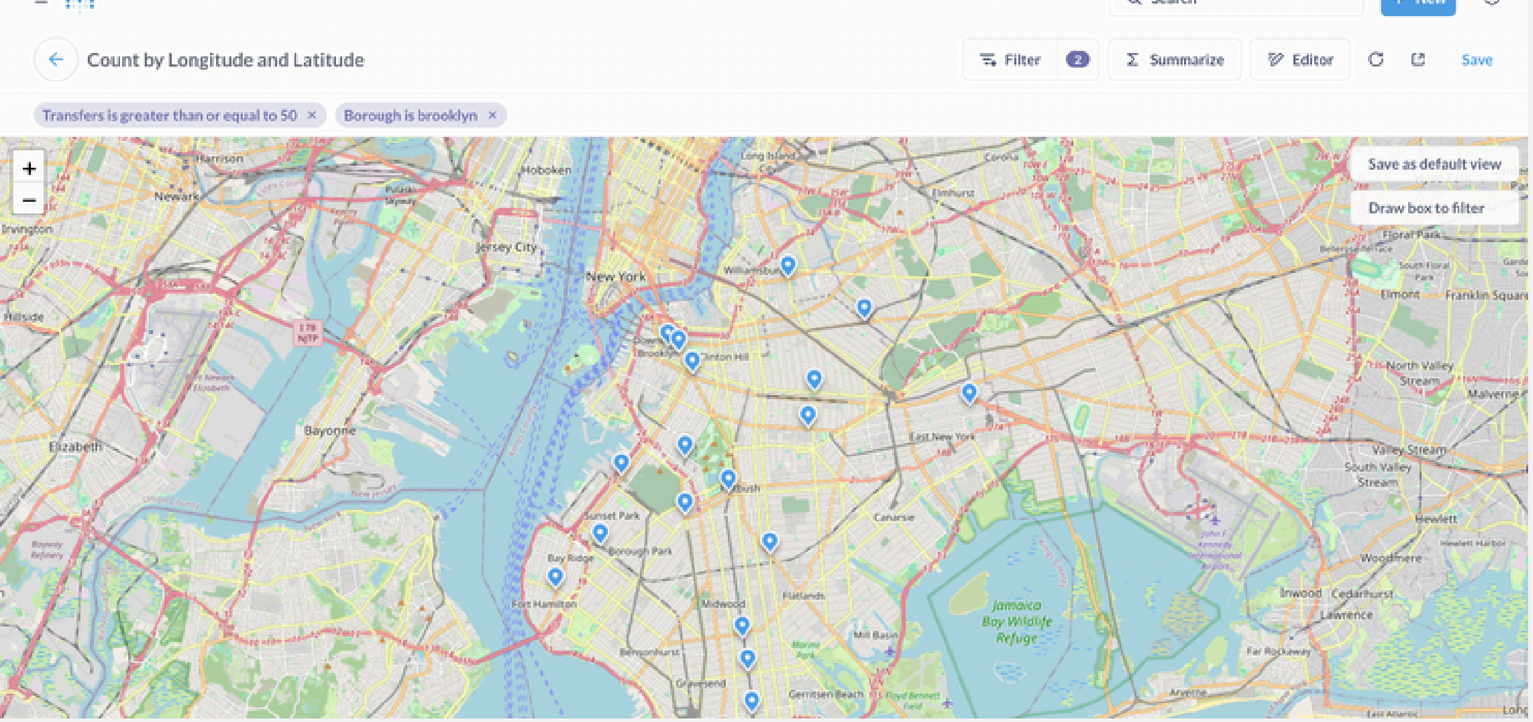
Worker Id	Address	State	Cores	Memory	Resources
worker-20250512180528-192.168.128.7-41937	192.168.128.7:41937	ALIVE	1 (1 Used)	3.0 GiB (1024.0 MiB Used)	
worker-20250512180528-192.168.128.9-43869	192.168.128.9:43869	ALIVE	1 (1 Used)	3.0 GiB (1024.0 MiB Used)	

Running Applications (1)

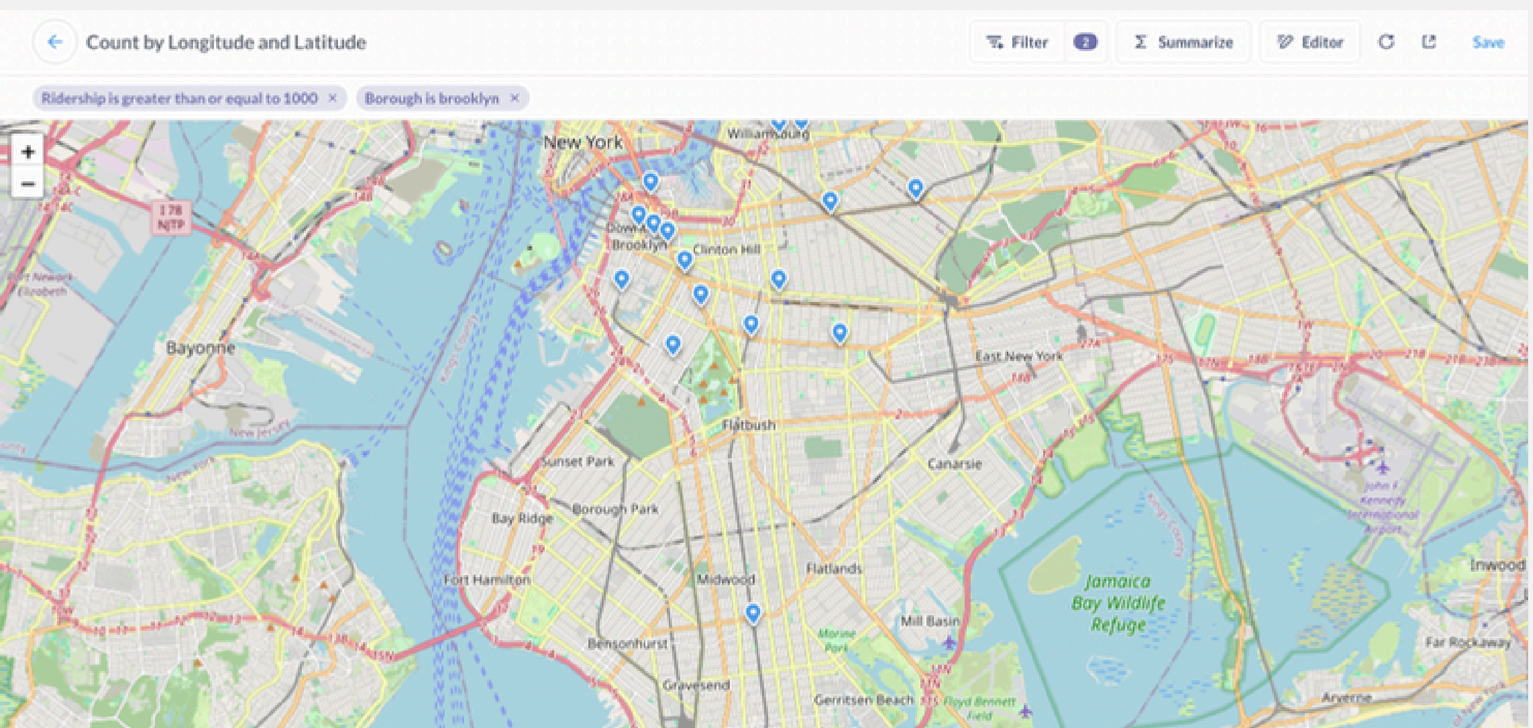
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20250512212723-0020 (kill)	mta_stream_kafka_to_mongodb	2	1024.0 MiB		2025/05/12 21:27:23	root	RUNNING	15 min

Completed Applications (20)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20250512212343-0019	mta_stream_kafka_to_mongodb	2	1024.0 MiB		2025/05/12 21:23:43	root	FINISHED	38 s
app-20250512211845-0018	mta_stream_kafka_to_mongodb	2	1024.0 MiB		2025/05/12 21:18:45	root	FINISHED	4.5 min
app-20250512211315-0017	mta_stream_kafka_to_mongodb	2	1024.0 MiB		2025/05/12 21:13:15	root	FINISHED	2.8 min
app-20250512210759-0016	mta_stream_kafka_to_mongodb	2	1024.0 MiB		2025/05/12 21:07:59	root	FINISHED	2.6 min
app-20250512210353-0015	mta_stream_kafka_to_mongodb	2	1024.0 MiB		2025/05/12 21:03:53	root	FINISHED	3.3 min
app-20250512204204-0014	mta_stream_kafka_to_mongodb	2	1024.0 MiB		2025/05/12 20:42:04	root	FINISHED	11 min

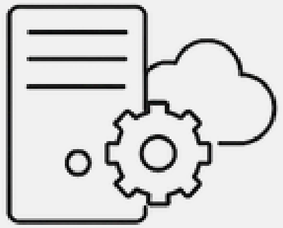


This map visualizes high-transfer subway stations in Brooklyn, filtered for transfer counts over 50 per hour. Atlantic Av–Barclays Center stands out as the busiest, serving as a major hub for multiple lines and heavy commuter flow.

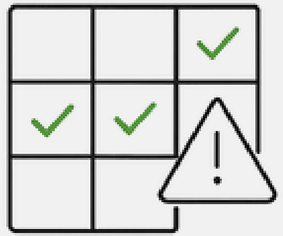


This map displays Brooklyn stations with hourly ridership over 1,000. Atlantic Av–Barclays Center again emerges as the busiest, highlighting its critical role in Brooklyn's transit network

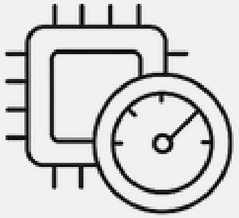
Challenges We Faced



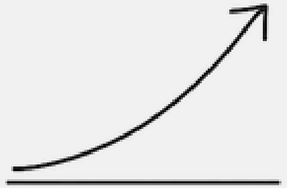
System Setup



Data Quality

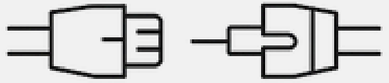


Resource Constraints



Tool Learning Curve

Access & Scalabiliity



Connectivity Issues

What we Learned

Data Integration & Streaming

Connected to MTA subway data using Sodapy + Socrata

Streamed data through Apache Kafka for real-time processing

Learned how to enrich, validate, and route data within pipelines

Stream Processing

Built scalable data workflows using PySpark Structured Streaming

Applied windowing, aggregations, and filtering on large datasets

Understood the role of Spark in handling high-volume data streams

Architecture & Deployment

Designed a robust system with Kafka, Spark.

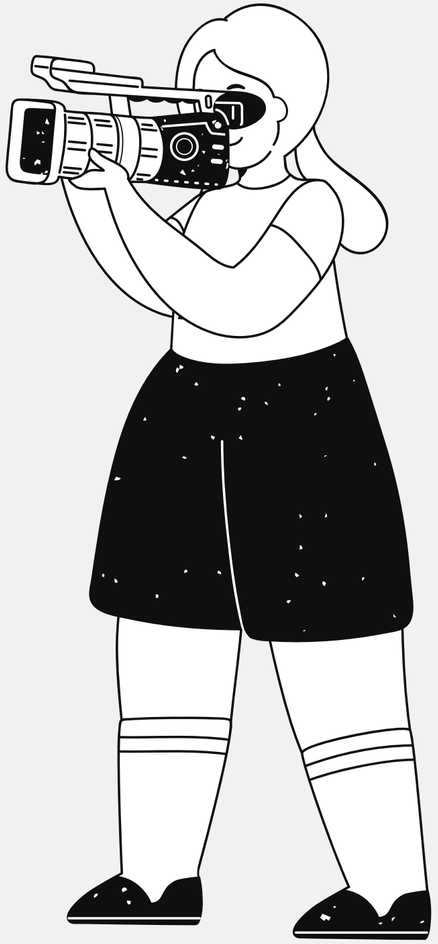
Used Docker and Docker Compose for containerized deployment

Improved understanding of service orchestration and fault tolerance

Visualization & Monitoring

Created Metabase dashboards to monitor Kafka and Spark metrics

Used MongoDB as a time-series backend for pipeline



Future Steps

Going forward, we plan to enhance the current system by integrating more contextual data such as bike-share availability and basic weather conditions. This would allow for more nuanced route suggestions without drastically increasing complexity.

We also aim to improve the modularity of our codebase, making it easier to plug in new data sources or swap out components like Spark or Kafka if needed. This flexibility will help adapt RouteSavvy to different use cases or datasets in the future.

Thank you!