Improving Urban Livability:

An Analysis of 311 Complaints and Their Relationship to Short-Term Rentals

Group - 1 Team :

Shreya Akotiya, Nikitha Seelam Balaji, Ashmitha Paruchuri Balaji, Shashira Guntuka

Abstract

- Growing popularity of Airbnb is creating concerns about how short-term rentals are affecting livability in neighborhoods.
- Utilized combined data of Airbnb listings and NYC 311 complaint information.
- Creating an analysis tool that will use 311 complaint data from New York City to assess its relationship to the presence of short-term rentals (like Airbnb) and to tourism.
- Using data to provide insight into how cities can strike the right balance between tourism and the quality of life for residents.

Motivation

- Short term rentals may be beneficial economically; however, they are detrimental to neighborhoods by creating noise, sanitation, parking issues and negatively impacting housing affordability in many of these neighborhoods.

- Airbnb has been found to have an impact on quality of life in NYC as 311 complaint rates were significantly higher in areas that had the highest levels of short-term rental activity.

- Therefore, a methodical evaluation which combines 311 complaint information and short-term rental data will assist public officials in finding the right balance for short term rental growth and neighborhood livability.

Problem Statement

“How does short-term rental activity correlate with 311 complaint patterns in NYC?”

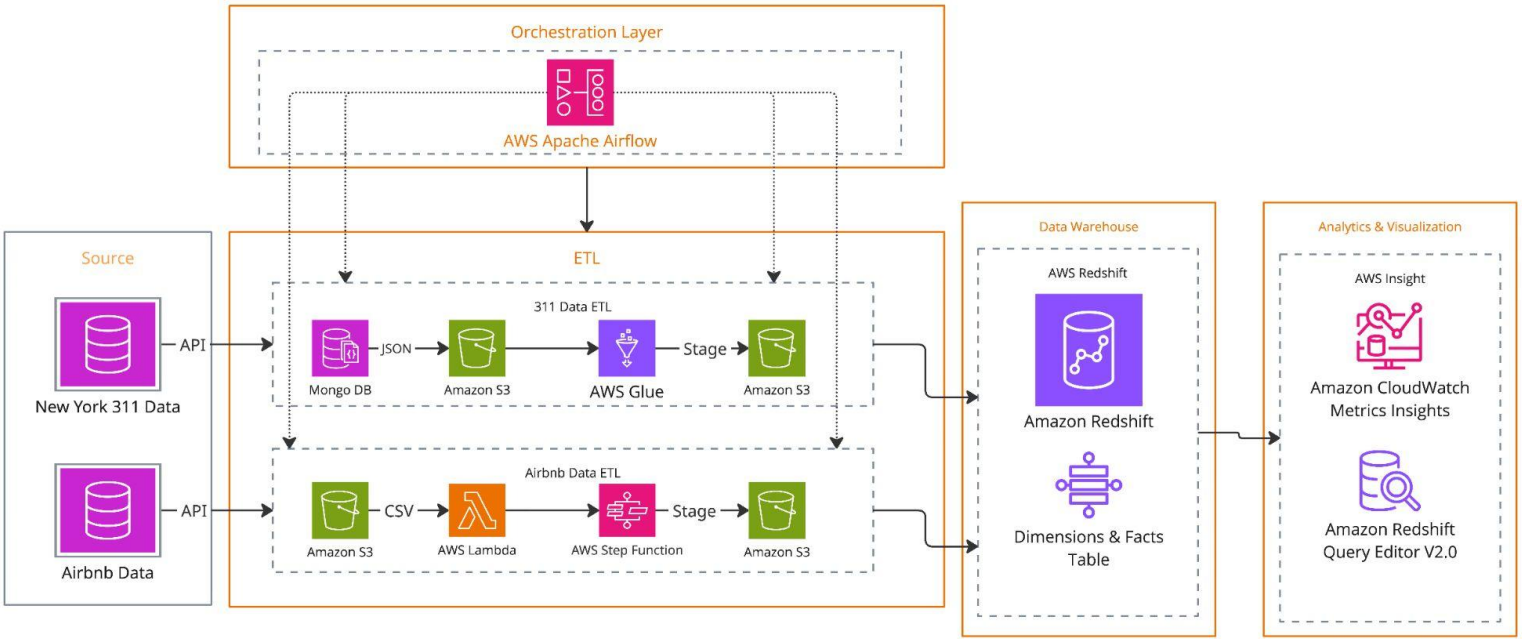
Objectives

- Identify correlation between short-term rental density and 311 complaints
- Analyze complaint patterns in New York City
- Provide actionable insights for policy and urban planning

Literature review

- Whole-home rentals and professional operators significantly increase rents and housing prices (Lee & Kim, 2023)
- Noise complaints peak in high-density and lower-income neighborhoods (NYC 311 Study)
- 311 data has reporting bias affluent areas report more than lower-income neighborhoods (Kontokosta et al., 2017)
- Resident perceptions of Airbnb vary by socioeconomic status (Perceived Impacts Study, 2024)
- Higher Airbnb listings correlate with increased disturbance complaints in multiple cities (Service Calls Study, 2021)

System Architecture



System Requirements

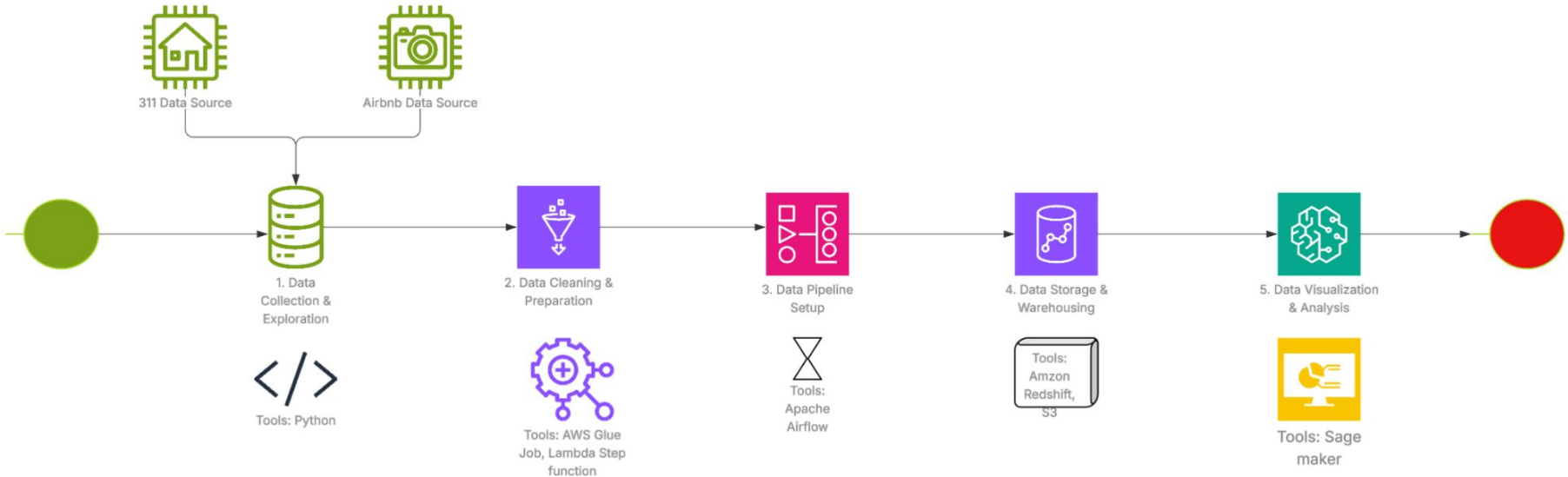
Functional Requirements

- Data Pre-Processing
- Data Modeling
- ETL Pipeline
- Data Warehousing for Analytics
- Analysis using SQL statements
- Data Visualization

Software Requirements

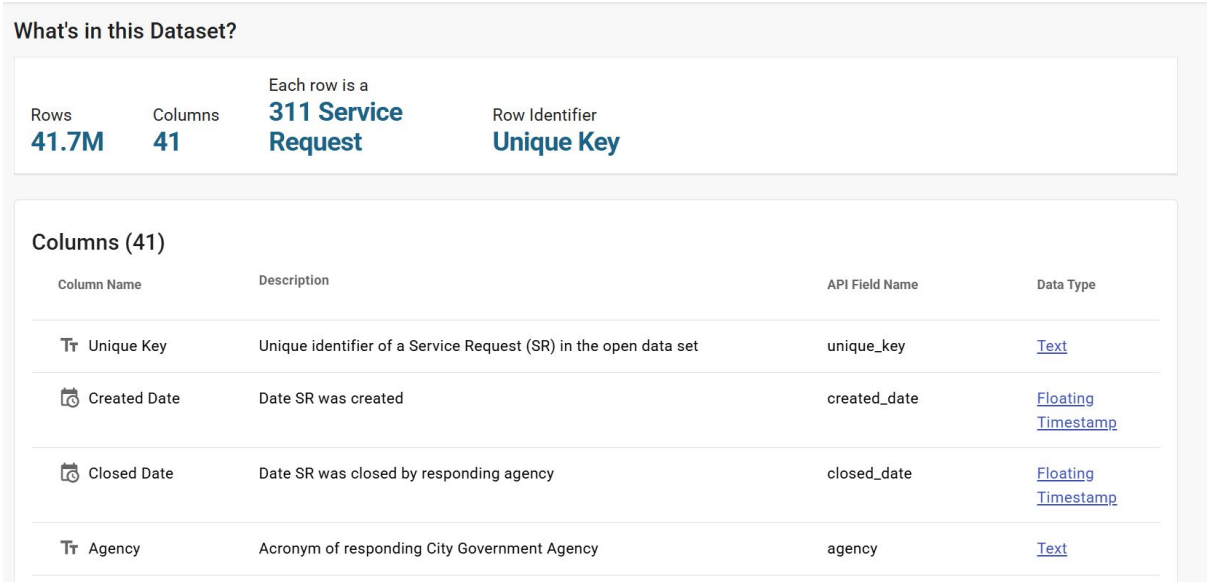
- Programming Languages:
  - Python
  - SQL
- Databases and Storage:
  - MongoDB
  - Amazon S3
  - Amazon Redshift
- Data Processing and Workflow Management:
  - AWS Glue
  - Apache Airflow
  - AWS Lambda
- Analysis and Visualization:
  - Amazon
  - SageMaker
- Development and Collaboration Tools:
  - Jupyter Notebook
  - GitHub
  - AWS

Workflow


311 Data Source
Airbnb Data Source
1. Data Collection & Exploration
Tools: Python
2. Data Cleaning & Preparation
Tools: AWS Glue Job, Lambda Step function
3. Data Pipeline Setup
Tools: Apache Airflow
4. Data Storage & Warehousing
Tools: Amzon Redshift, S3
5. Data Visualization & Analysis
Tools: Sage maker

Data Collection & Exploration

311 complaints dataset



What's in this Dataset?

Rows	Columns	Each row is a	Row Identifier
41.7M	41	311 Service Request	Unique Key

Columns (41)

Column Name	Description	API Field Name	Data Type
Unique Key	Unique identifier of a Service Request (SR) in the open data set	unique_key	Text
Created Date	Date SR was created	created_date	Floating Timestamp
Closed Date	Date SR was closed by responding agency	closed_date	Floating Timestamp
Agency	Acronym of responding City Government Agency	agency	Text

- NYC 311 Complaints Dataset: Collected from NYC Open Data Portal.
- Provides detailed service request records, covering various complaint types across NYC.

Data Collection & Exploration

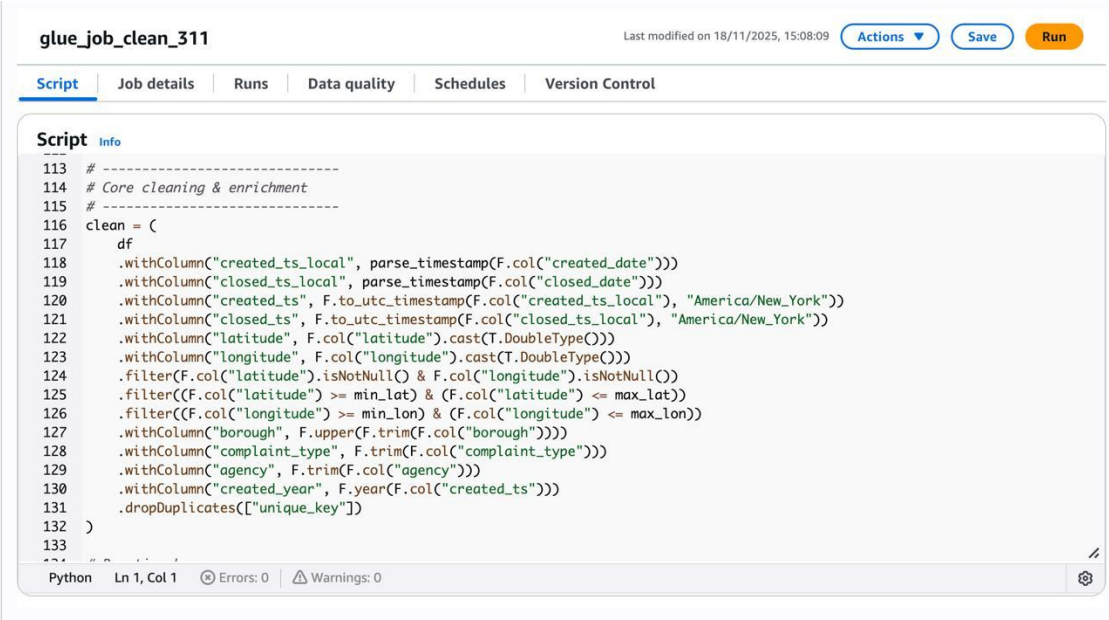
File Name	Description
listings.csv.gz	Detailed Listings data
calendar.csv.gz	Detailed Calendar Data
reviews.csv.gz	Detailed Review Data
listings.csv	Summary information and metrics for listings in Albany (good for visualisations).
reviews.csv	Summary Review data and Listing ID (to facilitate time based analytics and visualisations linked to a listing).
neighbourhoods.csv	Neighbourhood list for geo filter. Sourced from city or open source GIS files.
neighbourhoods.geojson	GeoJSON file of neighbourhoods of the city.

Airbnb Dataset

- Collected from Inside Airbnb.
- Contains five large CSV files for New York City.
- Includes details on listings, hosts, prices, availability, reviews, and locations.

311 complaints Data Cleaning

- We focused on improving data accuracy and consistency across sources.
- Removed entries missing essential location details (latitude, longitude, address).
- Filtered out duplicates based on time, location, and description.
- Ensured the dataset was reliable for spatial and neighborhood level analysis.



Airbnb listings Data Cleaning

- Removed invalid entries, listings without coordinates or host information.
- Kept only listings within NYC boundaries (lat 40.4–41.0, long −74.3– −73.7).
- Cleaned price data by removing symbols and excluding zero/negative values.
- Standardized text fields (neighborhood, host name, room type) for consistency.
- Deleted duplicate listing IDs to ensure each property was unique.

```
# ========================================

# 1. Remove nulls in critical fields
print("\n🧹 Cleaning Step 1: Removing null IDs and coordinates...")
df_clean = df.dropna(subset=['id', 'latitude', 'longitude'])
print(f"  ✓ Removed {initial_count - len(df_clean)} records with nulls")


# 3. Clean price field
print("\n💰 Cleaning Step 3: Cleaning price field...")
if 'price' in df_clean.columns:
    before = len(df_clean)
    # Remove $ and commas
    df_clean['price'] = df_clean['price'].astype(str).str.replace('$', '', regex=False)
    df_clean['price'] = df_clean['price'].str.replace(',', '', regex=False)
    df_clean['price'] = pd.to_numeric(df_clean['price'], errors='coerce')


# 4. Normalize text fields
print("\n📝 Cleaning Step 4: Normalizing text fields...")
text_fields = {
    'neighbourhood_group': 'borough',
    'neighbourhood': 'neighbourhood',
    'room_type': 'room_type'
}

for field, alias in text_fields.items():
    if field in df_clean.columns:
        df_clean[field] = df_clean[field].str.upper().str.strip()
        print(f"  ✓ Normalized {field}")

# 5. Data type conversions (FIXED - ensures proper types for Parquet)
print("\n🔢 Cleaning Step 5: Converting data types...")
df_clean['latitude'] = df_clean['latitude'].astype(float)
df_clean['longitude'] = df_clean['longitude'].astype(float)
```

Feature Engineering

```
134  # Duration hours
135  clean = clean.withColumn(
136      "duration_hours",
137      F.when(F.col("closed_ts").isNotNull(),
138             (F.col("closed_ts").cast("long") - F.col("created_ts").cast("long")) / 3600.0)
139  )
140
141  # Geohash
142  clean = clean.withColumn("geohash", encode_geohash_udf(F.col("latitude"), F.col("longitude")))
143
144  final_df = clean
145
```

- Created new features like geohash to accurately match Airbnb listings and 311 complaints by location.
- Added time based metrics complaint resolution duration, availability, and review counts to analyze trends and correlations.

Data Preparation

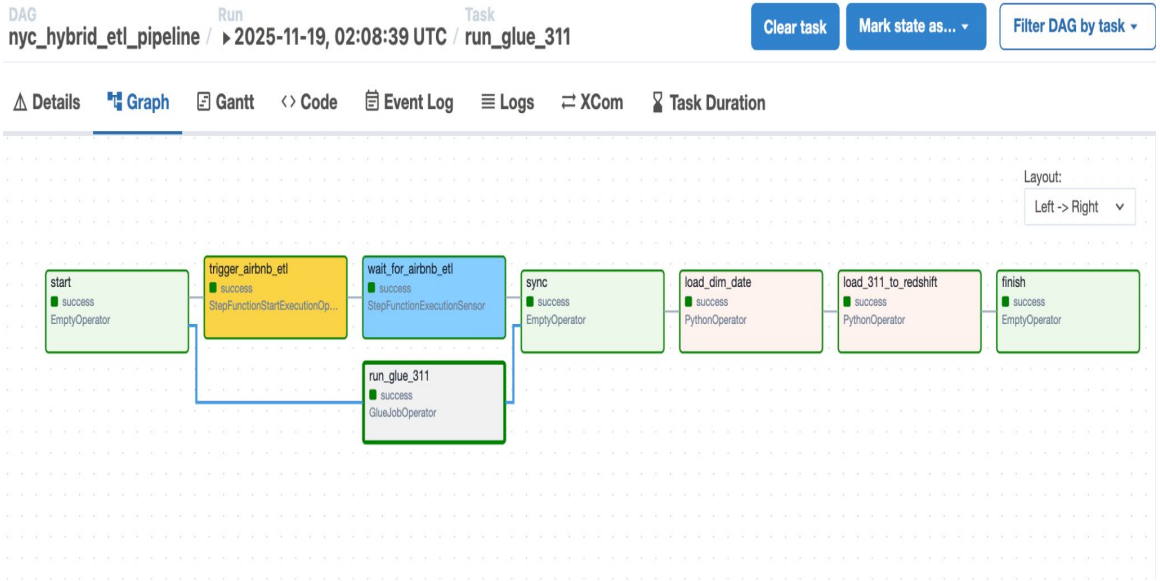
- We converted cleaned datasets to Parquet format for efficient storage and querying.
- The data is stored in Amazon S3 and loaded into Amazon Redshift for analysis.
- Enabled easy joining by common attributes such as date, borough, and geohash.

```
df = pd.read_parquet("part-00000-ce312e92-2bf3-4bcf-97ab-ef85626b1143.c000.snappy.parquet", engine="fastparquet")
print(df.head())
```

```
...                         _id address_type agency  \
0  6902ab8dc1083eb256397dae      ADDRESS    DOT
1  6902ab8dc1083eb256397ba8      ADDRESS   DSNY
2  6902ab8dc1083eb2563973d2      ADDRESS   DSNY
3  6902ab8dc1083eb2563979bd      ADDRESS   DSNY
4  6902ab8dc1083eb256397293      ADDRESS   NYPD

                       agency_name         bbl   borough  \
0     Department of Transportation  3026810033  BROOKLYN
1         Department of Sanitation  4113240009    QUEENS
2         Department of Sanitation  3070170035  BROOKLYN
3         Department of Sanitation  3003420023  BROOKLYN
4  New York City Police Department        None  BROOKLYN

  bridge_highway_direction bridge_highway_name bridge_highway_segment  \
0                     None                None                   None
1                     None                None                   None
2                     None                None                   None
3                     None                None                   None
4                     None                None                   None

             city  ... y_coordinate_state_plane    created_ts_local  \
0        BROOKLYN  ...                  202560 2024-12-02 11:40:09
1  CAMBRIA HEIGHTS  ...                  192662 2024-12-02 12:24:44
2        BROOKLYN  ...                  149559 2024-12-02 15:11:33
3        BROOKLYN  ...                  188507 2024-12-02 13:02:59
4        BROOKLYN  ...                  184663 2024-12-02 15:36:19

       closed_ts_local          created_ts           closed_ts duration_hours  \
0 2025-01-13 14:21:55 2024-12-02 16:40:09 2025-01-13 19:21:55    1010.696111
1 2024-12-04 13:38:19 2024-12-02 17:24:44 2024-12-04 18:38:19      49.226389
2 2024-12-04 14:28:25 2024-12-02 20:11:33 2024-12-04 19:28:25      47.281111
3 2024-12-02 21:27:55 2024-12-02 18:02:59 2024-12-03 02:27:55       8.415556
4 2024-12-02 17:11:42 2024-12-02 20:36:19 2024-12-02 22:11:42       1.589722
```

Data Pipeline Setup

- Developed two ETL pipelines one for NYC 311 and one for Airbnb, due to differing data formats and sizes.
- Automated and orchestrated with Apache Airflow in Docker, ensuring stable, reusable, and fully automated execution across environments.

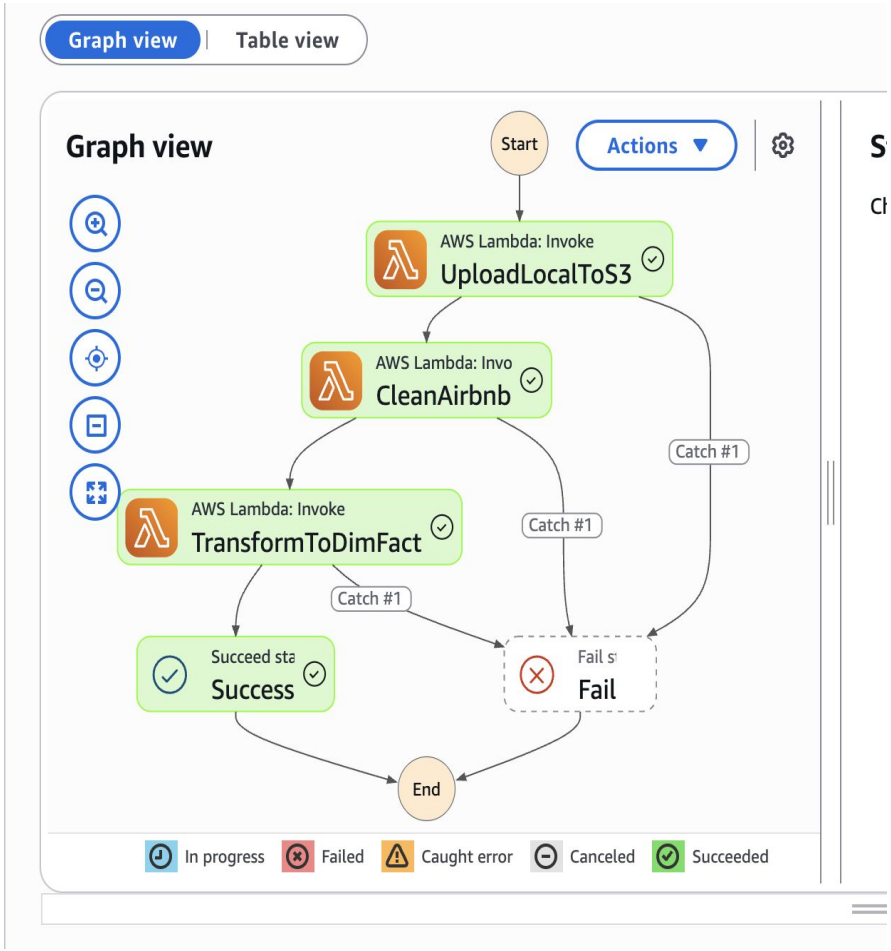


Data Ingestion

- Dynamic data ingestion via Python + Airflow: 311 data pulled from API, Airbnb data uploaded from CSV both stored in Amazon S3 as the data lake.
- Structured S3 storage (raw/311, raw/airbnb) with Airflow scheduling, retries, and logging ensuring reliability and automation.

ETL for 311 Complaints Data (AWS Glue)

- We used AWS Glue for large scale semi structured data processing.
- Automated extraction & transformation from NYC Open Data API to S3.
- Performed data cleaning, validation, and normalization within AWS Glue.
- Stored processed data in Parquet format for efficient Redshift loading.
- Orchestrated with Apache Airflow for reliable, repeatable updates.



ETL for Airbnb Data (AWS Lambda + Step Functions)

- We built a serverless ETL using AWS Lambda and Step Functions.
- We initiated automated upload, cleaning, and transformation of Airbnb data to S3 through Airflow.
- Cleaning included coordinate fixes, price normalization, and deduplication.
- Generated dimension and fact tables (host, property, location, listings).

S3 to Amazon Redshift

- Data loaded from S3 to Redshift using COPY from Parquet for efficiency.
- We built a star schema with two fact tables and multiple shared dimensions.
- Fact Tables: 311 complaints & Airbnb listings.
- Dimension Tables: date, location, host, property, agency, complaint, etc.
- We used GeoHash as common column for both the dataset.

- End-to-end automation using Apache Airflow in Docker.
- Airflow DAGs handled API ingestion, ETL jobs, Redshift loads, and dashboard updates.
- We used containerized setup to ensure stable, reproducible pipeline deployments.

Data Modeling

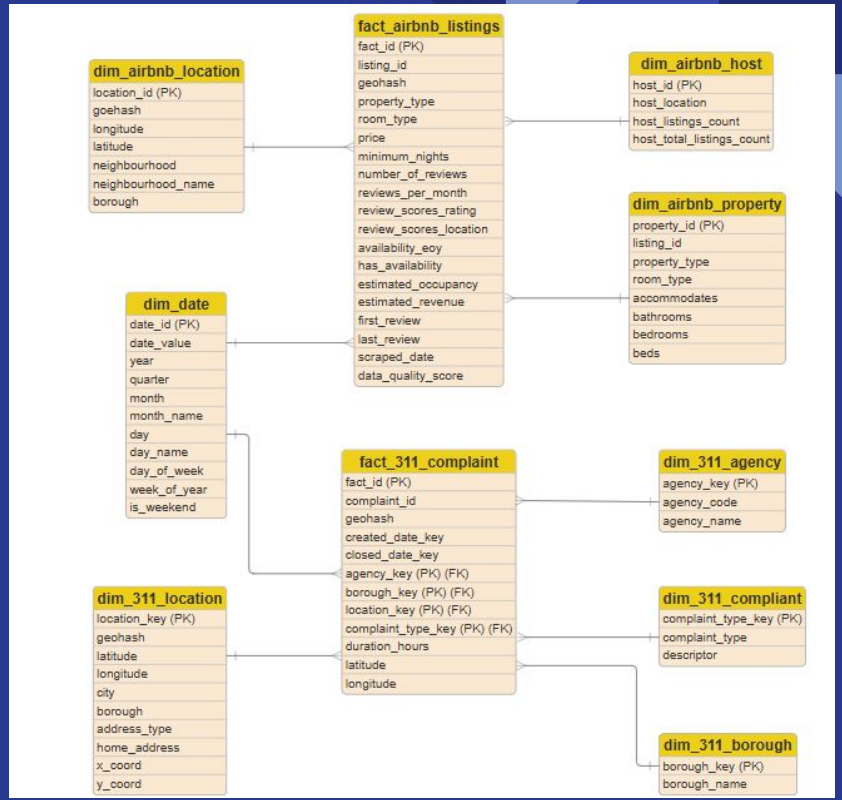
The model follows a star schema approach.

Fact Tables:

- fact_311_complaint contains detailed records of 311 complaints with references to dimensions : date, location, agency, borough, and complaint type.
- fact_airbnb_listings stores Airbnb listing details and key aggregates and facts such as price, review scores, and availability.

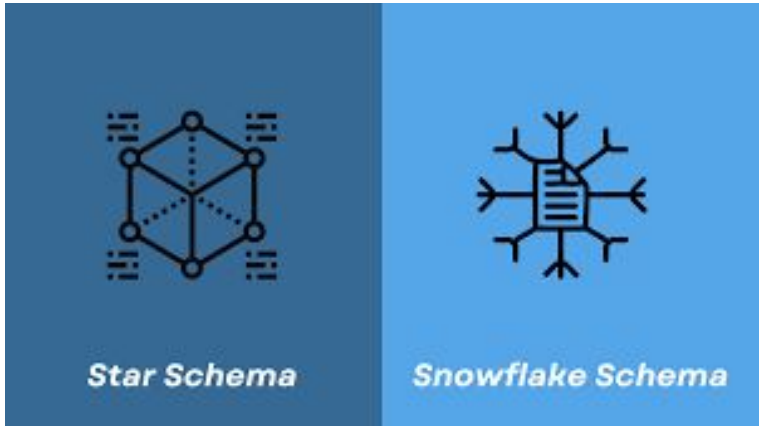
Dimension Tables:

- dim_date (common dimension)
- dim_311_borough
- dim_311_location
- dim_311_agency
- dim_311_complaint
- dim_airbnb_location
- dim_airbnb_property
- dim_airbnb_host

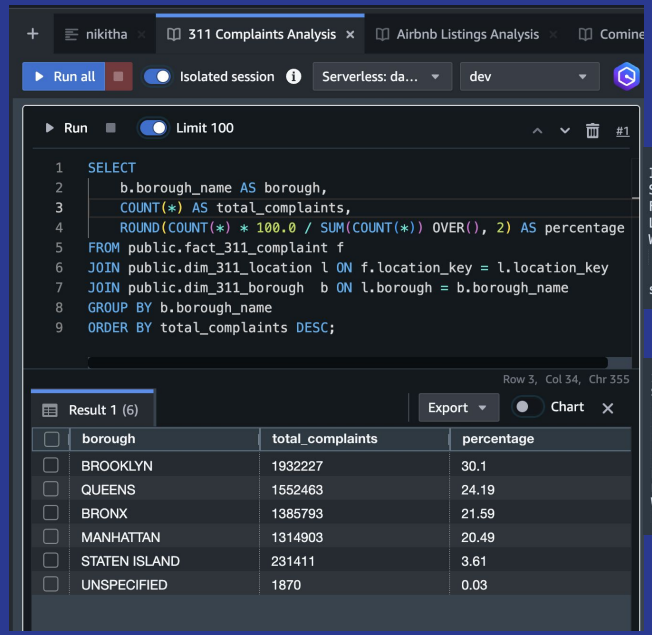


Star Schema over Snowflake

- We preferred Star Schema over snowflake, because we wanted data modeling concepts to be simple and avoid joins.
- Redshift works efficiently with denormalized data, so using a star schema helped us reduce the number of joins in queries. This makes data retrieval faster and improves performance when we were working with large datasets of 311 complaints and Airbnb listings.
- Star Schema works better tools like AWS Glue, Redshift, and QuickSight also It supported easy loading from S3 .



Amazon Redshift Queries




```
INSERT INTO public.dim_311_agency (agency_name)
SELECT DISTINCT TRIM(s.agency)
FROM staging.staging_311 s
LEFT JOIN public.dim_311_agency a ON a.agency_name = TRIM(s.agency)
WHERE s.agency IS NOT NULL
  AND a.agency_key IS NULL;

select * from public.dim_311_agency
```

```
INSERT INTO public.dim_311_location (geohash, latitude, longitude, city, borough)
SELECT DISTINCT
  s.geohash,
  s.latitude,
  s.longitude,
  NULLIF(TRIM(s.city),'')     AS city,
  NULLIF(UPPER(TRIM(s.borough)),'') AS borough
FROM staging.staging_311 s
LEFT JOIN public.dim_311_location d ON d.geohash = s.geohash
WHERE s.geohash IS NOT NULL
  AND d.location_key IS NULL;
```

Run Limit 100

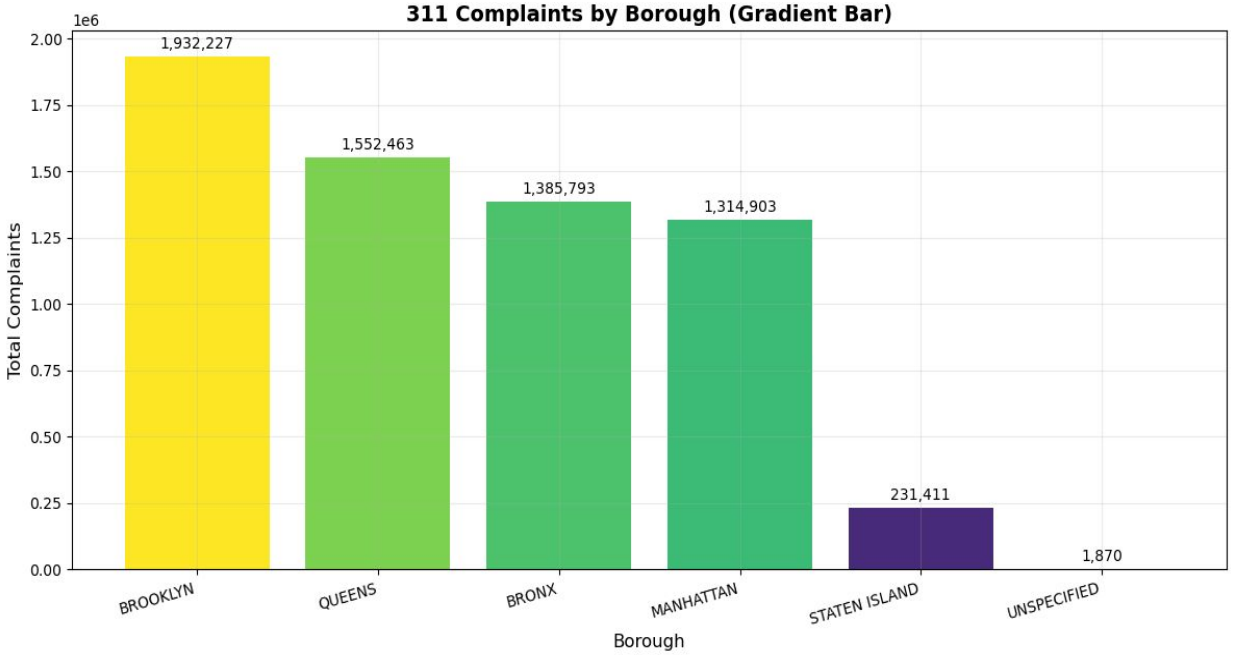
```
SELECT
    al.borough,
    COUNT(*) AS total_listings,
    ROUND(AVG(TRY_CAST(f.price AS DECIMAL(18,2))), 2) AS avg_price,
    MIN(TRY_CAST(f.price AS DECIMAL(18,2))) AS min_price,
    MAX(TRY_CAST(f.price AS DECIMAL(18,2))) AS max_price
FROM public.fact_airbnb_listings f
JOIN public.dim_airbnb_location al
  ON f.geohash = al.geohash
WHERE al.borough IS NOT NULL
  AND f.price NOT IN ('NaN', 'nan', '', 'INF', '-INF')
  AND TRY_CAST(f.price AS DECIMAL(18,2)) IS NOT NULL
GROUP BY al.borough
ORDER BY total_listings DESC;
```

Run Limit 100

```
SELECT
    COALESCE(f.room_type, dp.room_type) AS room_type,
    al.borough,
    COUNT(*) AS listing_count,
    ROUND(AVG(TRY_CAST(f.price AS DECIMAL(18,2))), 2) AS avg_price
FROM public.fact_airbnb_listings f
JOIN public.dim_airbnb_location al
```

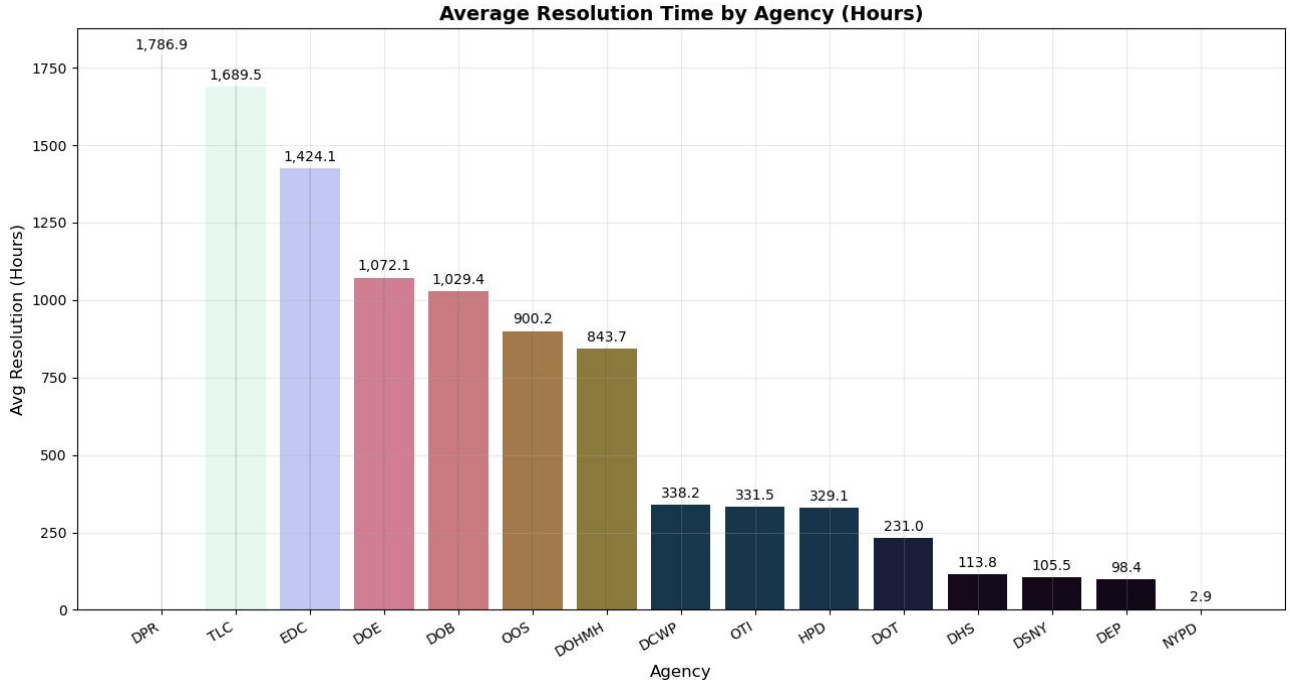
Data was loaded into Amazon Redshift using SQL queries.Redshift queries were used to aggregate data for analysis.Multiple joins between fact and dimension tables was performed to detect the patterns and relationships od short term rentals to 311 complaints .

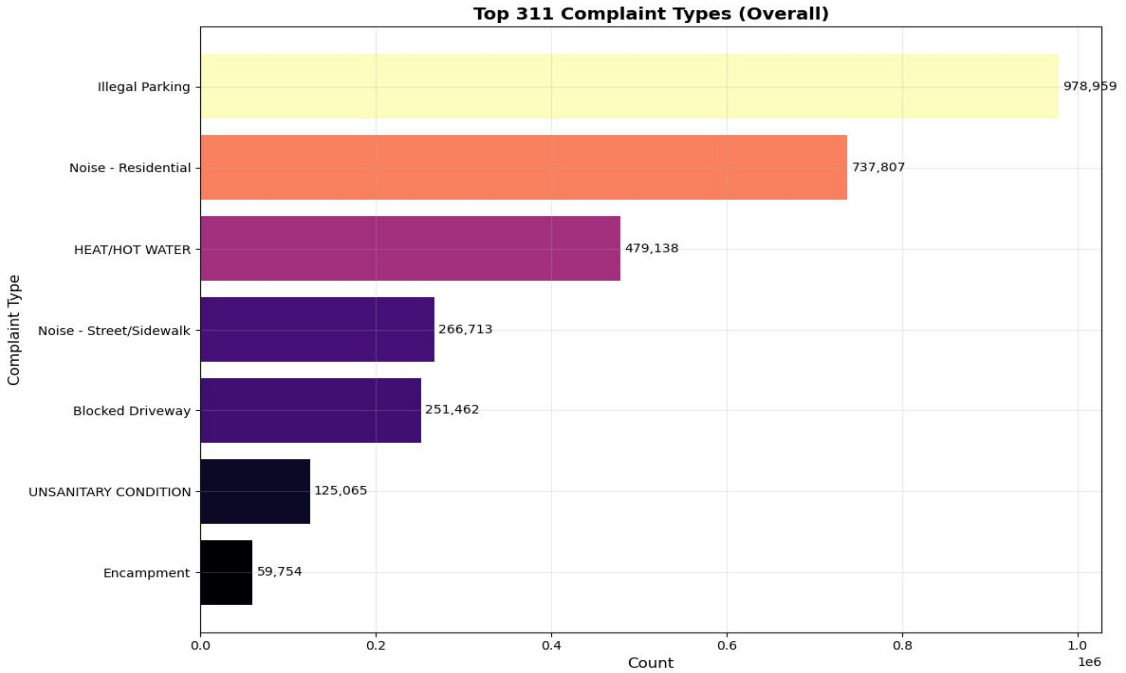
Analysis & Visualization



- Brooklyn reports the highest complaints (~2M), followed by Queens, Bronx, and Manhattan.
- Staten Island and Unspecified categories show minimal complaint volumes.
- Brooklyn's high count reflects its large population and dense short-term rental activity.

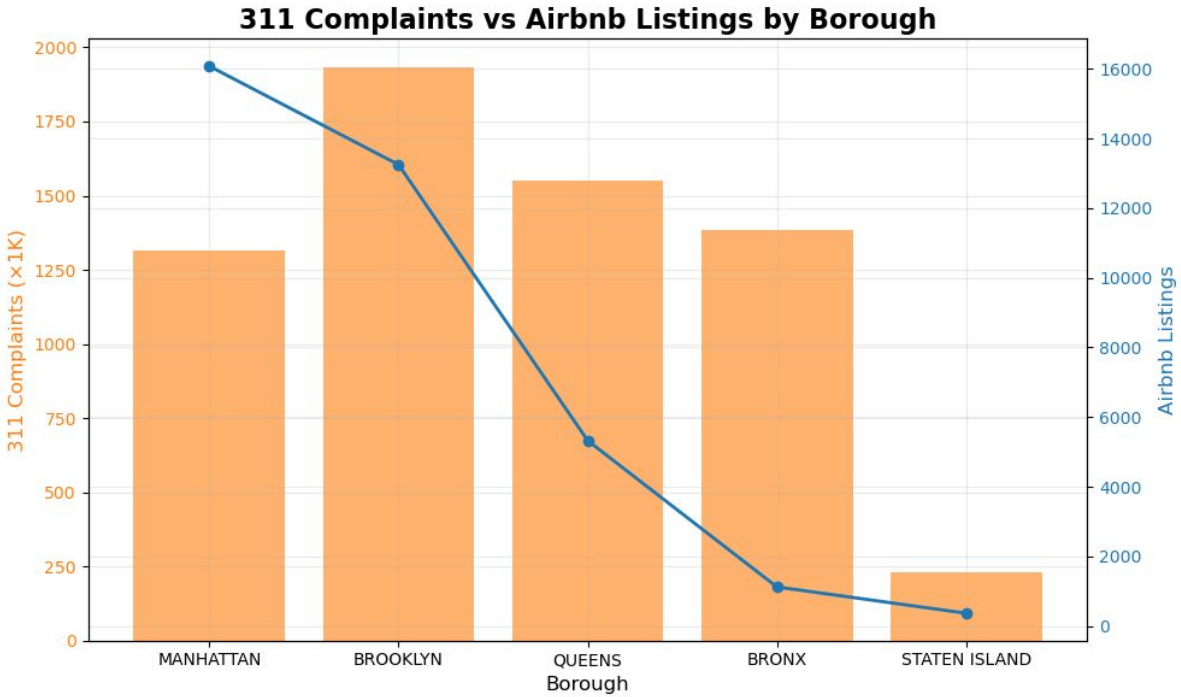
- DPR, TLC, and EDC show the longest resolution times (>1,000 hrs), reflecting complex or lower-priority cases.
- NYPD, DEP, and DSNY resolve complaints much faster, typically within a few days.



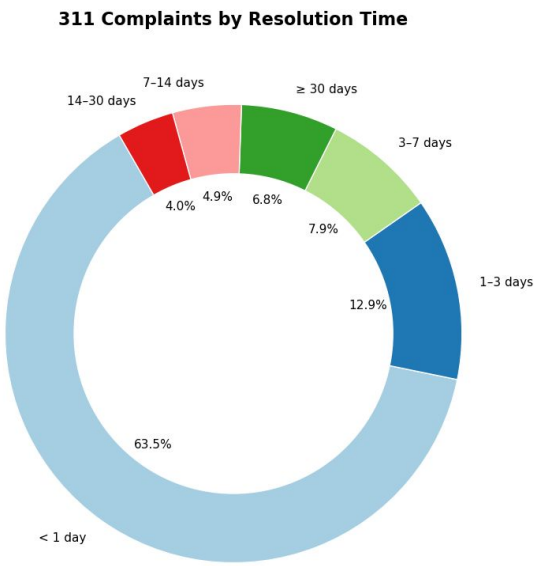


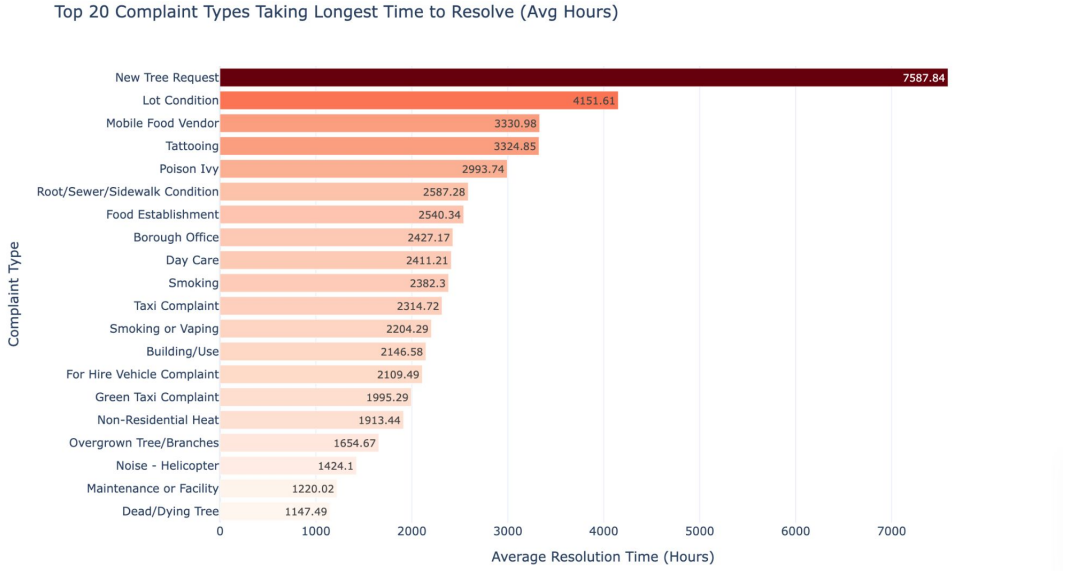
- Illegal Parking and Noise Residential are the most frequent complaints, followed by Heat / Hot Water issues.
- Noise-related complaints remain common in densely populated and tourist heavy areas.

- Manhattan leads in both listing volume and average price, with the highest revenue potential.
- Brooklyn shows moderate listings and prices, but still notable revenue clusters.



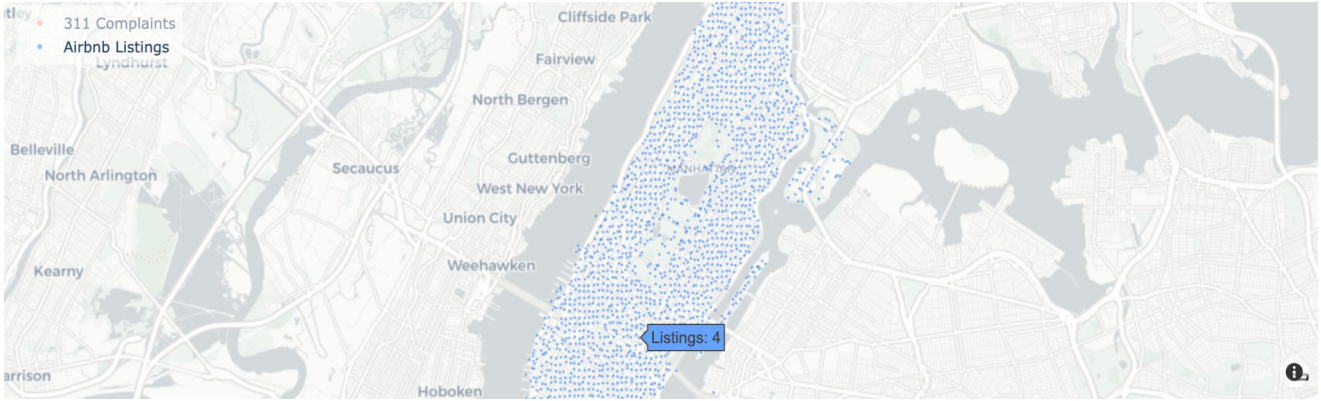
- Manhattan and Brooklyn show the highest Airbnb listings and 311 complaint volumes.
- Indicates a positive correlation between rental activity and resident complaints.





- **~75% of complaints** resolved within **3 days**, showing **efficient agency response**.
- **Longer resolutions (2 to 4 weeks)** mostly involve **fieldwork heavy cases** like *New Tree Request* and *Lot Condition*.

- Strong correlation between Airbnb listings and 311 complaints across boroughs.
- Manhattan leads in both metrics, indicating highest rental activity and complaint density.



- Scatter map shows areas with more Airbnb listings also have more 311 complaints.
- Manhattan displays the strongest spatial correlation between listings and complaints.

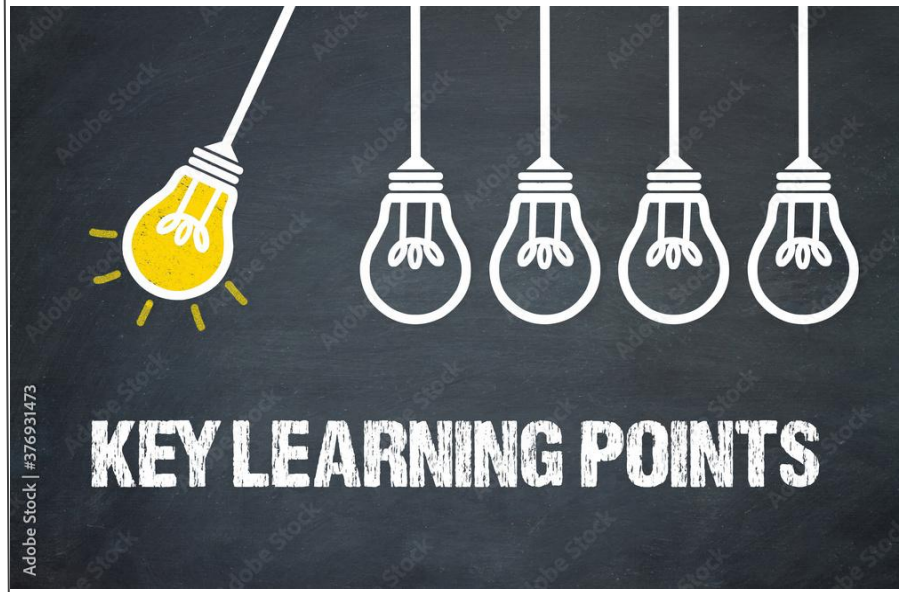
Optimization Techniques

- **Incremental Loads:** Only new/changed data ingested faster, avoids duplicates.
- **Staging & Merge Pattern:** Data validated in staging before merge; ensures lineage & rollback safety.
- **Parquet + COPY:** Compressed columnar loads from S3 for faster I/O and lower cost.
- **Set-Based SQL:** Bulk inserts/joins for parallelized, high-speed Redshift processing.
- **Airflow DAG Tuning:** Modular, idempotent tasks with retry logic for fault tolerance.
- **Shared Dimensions:** Reused dim_date across datasets for consistency.
- **Optimized Joins:** Cached dimension lookups reduce CPU overhead in Redshift.
- **Serverless Architecture:** AWS Glue, Step Functions, and Redshift Serverless for elastic scaling.
- **Data Cleaning:** Applied TRIM, UPPER, NULLIF for standardized, efficient joins.

Technical Difficulties

- Merging large public datasets such as NYC 311 complaints and InsideAirbnb required resolving schema discrepancies and structural inconsistencies.
- Building an orchestrated ETL pipeline using Apache Airflow, AWS Step Functions, and AWS Glue required careful dependency management.
- Implementing a star schema in Amazon Redshift with incremental loading was challenging.
- Spatial and Temporal Mapping.
- To build fault-tolerant ETL system capable of integrating structured and semi-structured data into a unified warehouse.
- Redshift Performance Tuning.

- Higher Airbnb listing density correlates with more 311 complaints, especially noise and quality-of-life issues.
- Manhattan shows the strongest overlap between listings and complaint volume.
- ~75% of complaints resolved within 3 days; longer timelines tied to fieldwork-heavy cases.
- Spatial visualizations confirm clear clustering of listings and complaints.
- End-to-end AWS-based data pipeline ran efficiently with accurate visual outputs.



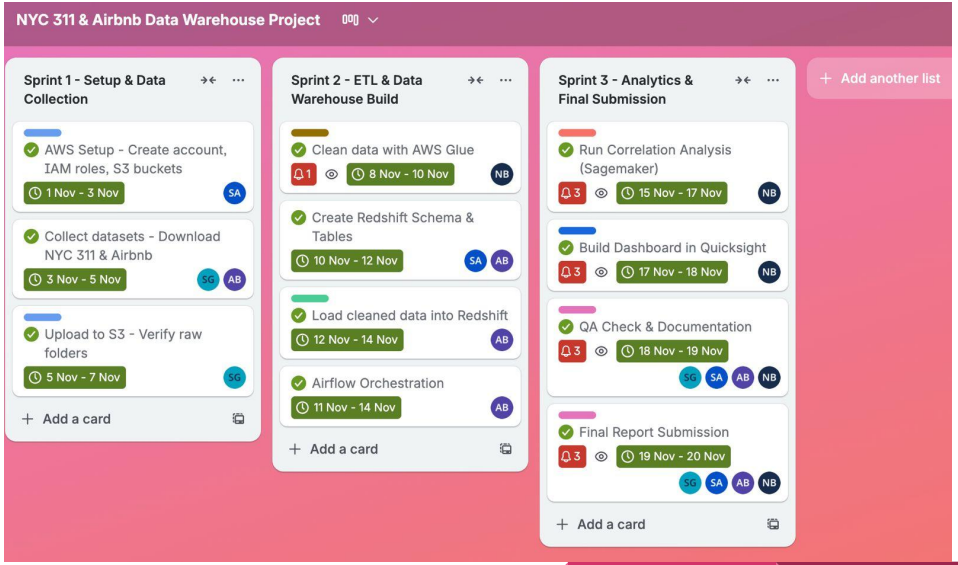
- Gained experience handling real-world, messy, and incomplete data responsibly.
- Learned API limit management, automation, and error handling for reliability.
- Built confidence in using cloud tools (MongoDB, AWS Glue, Redshift) for end-to-end pipelines.
- Developed patience, problem-solving, and professional data engineering skills.

- Built a single, automatic data pipeline that incorporates many different technological tools.
- Data Flow:

  NYC Open Data API -> MongoDB -> AWS Glue -> Redshift -> QuickSight -> SageMaker.
- Designed a Cloud-based workflow that allows for instant access to insights about urban trends and patterns in real time.
- Shows how cities can use cloud based systems to track their residents' well-being.

AGILE AND SCRUM MEETINGS

- Followed Agile methodology with a Scrum-based approach.
- Organized into three sprints: setup & data collection, ETL & warehouse build, and analytics & submission.
- Conducted regular sprint planning and reviews via Google Meet.
- Shreya served as Scrum Master; team collaborated through a Kanban board for tracking the tasks.
- Sprint retrospectives held after each phase to assess progress and improvements.




Contributor Roles
Project Proposal
Shashira Guntuka
Nikitha Seelam Balaji
Shreya Akotiya
Ashmitha Paruchuri Balaji
Data Processing and Cleaning
Shreya Akotiya
Nikitha Seelam Balaji
Dimension Modelling
Ashmitha Paruchuri Balaji
Validation
Shreya Akotiya
Data Import
Ashmitha Paruchuri Balaji
Literature Survey
Nikitha Seelam Balaji
ETL Pipeline
Ashmitha Paruchuri Balaji
Shreya Akotiya
Writing – Original Draft
Shreya Akotiya
GitHub Maintenance & Scrum Planning
Shashira Guntuka
Nikitha Seelam Balaji
Visualization
Nikitha Seelam Balaji
Logical Planning
Ashmitha Paruchuri Balaji
Shreya Akotiya
Writing – Review & Editing
Nikitha Seelam Balaji
Ashmitha Paruchuri Balaji
Shreya Akotiya
Methodology
Shreya Akotiya

New ETL Tool used

Luigi

- Built a Luigi pipeline to automate tokenization of NYC 311 complaint text.
- Used tiktoken library for ChatGPT-compatible tokenization and count generation.
- Pipeline handles batch processing, creates outputs automatically, and logs warnings for invalid data.
- Ensures scalable, reproducible preprocessing of large complaint datasets.

Lambda +Step Function as ETL tool

- Created a serverless data pipeline by utilizing AWS Lambda to perform ETL (extract, transform, load), as well as AWS Step Functions to manage the flow of my workflow.
- AWS Lambda was utilized to extract the data from the source system, clean the data and then load it into a target system, while AWS Step Functions were used to manage the entire flow of my data workflow.
- This method is cost effective and scalable since the Lambda function will run only when I need it to, and automatically scale based on the needs of the function.

Key Links

- Github Link :

  https://github.com/Sbnikitha/ADI-226-Datawarehouse-project

- Dataset Link :

  311 Service Requests from 2010 to Present | NYC Open Data

  Get the Data | Inside Airbnb

Thank You!!