Project Phase-2
Group-3

Task 1: Data Cleaning - Identify dirty data and code for cleaning.

The code for cleaning the data is in clean_data.py
The queries used to identify the dirty data/ retrieve it is as below. This was slightly modified and 'Delete From' was used to delete the dirty data.

Location Table:

Query 1:

```
SELECT *
FROM Location
WHERE Borough IS NULL OR Zone IS NULL;
```

Returns: 2 rows that should be deleted.
Reasoning:
In the Location table, both the Borough and Zone columns are crucial for accurately identifying a location. Rows where either of these fields is NULL indicate missing or incomplete data. Such incomplete records can cause problems in data analysis, like incorrect mapping of trips to locations or errors in aggregating data by borough or zone. To maintain data quality and ensure accurate analyses, these rows with missing Borough or Zone information should be deleted.

Query 2:

```
SELECT Borough, Zone, COUNT(*) AS duplicates
FROM Location
GROUP BY Borough, Zone
HAVING COUNT(*) > 1 ;
```

Returns: 5 rows
Reasoning:
Each combination of `Borough` and `Zone` in the `Location` table should be unique, representing a distinct geographic area. Cleaning these duplicates helps in improving data integrity and the reliability of any location-based analysis.

Time Table:

Query 3:

```
SELECT * FROM Time WHERE DropOffDate < PickUpDate
OR (DropOffDate = PickUpDate AND DropOffTime < PickUpTime);
```

Returns: 62 rows.
Reasoning:
In the Time table, it's essential that the drop-off date and time occur after the pick-up date and time, as a trip cannot end before it starts. Hence we deleted such data since it doesn't make sense.

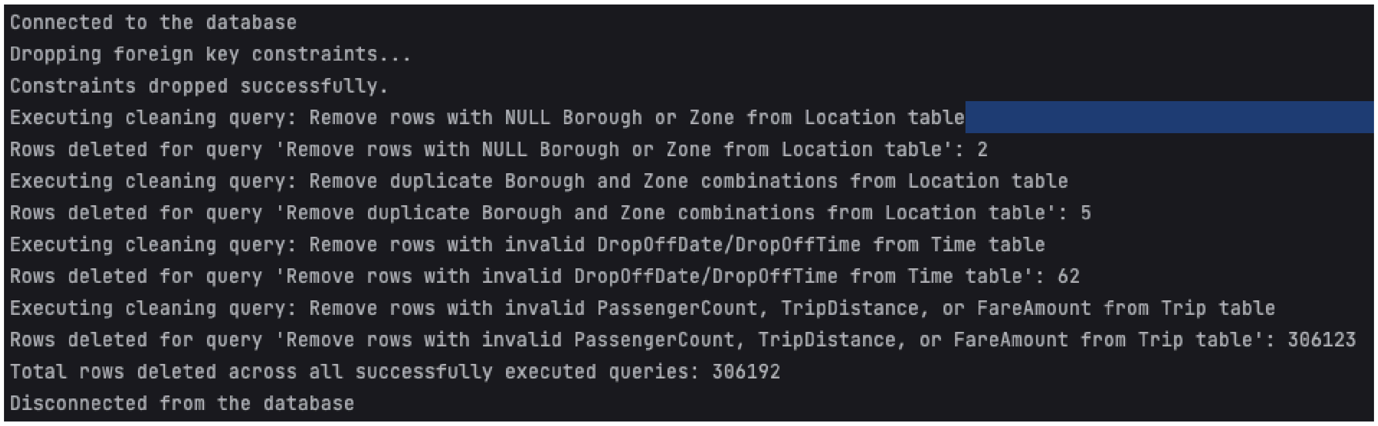
Trip Table:

Query 4:

```
SELECT * FROM Trip
WHERE PassengerCount < 1
OR TripDistance < 0 OR FareAmount < 0;
```

Returns: 306123 rows
Reasoning:
In the Trip table, fields like PassengerCount, TripDistance, and FareAmount should logically have positive values. These likely seem to be a mistake and such data was removed from the tables.

Output:



```
Connected to the database
Dropping foreign key constraints...
Constraints dropped successfully.
Executing cleaning query: Remove rows with NULL Borough or Zone from Location table
Rows deleted for query 'Remove rows with NULL Borough or Zone from Location table': 2
Executing cleaning query: Remove duplicate Borough and Zone combinations from Location table
Rows deleted for query 'Remove duplicate Borough and Zone combinations from Location table': 5
Executing cleaning query: Remove rows with invalid DropOffDate/DropOffTime from Time table
Rows deleted for query 'Remove rows with invalid DropOffDate/DropOffTime from Time table': 62
Executing cleaning query: Remove rows with invalid PassengerCount, TripDistance, or FareAmount from Trip table
Rows deleted for query 'Remove rows with invalid PassengerCount, TripDistance, or FareAmount from Trip table': 306123
Total rows deleted across all successfully executed queries: 306192
Disconnected from the database
```

Task 2: Itemset Mining - Discover frequent itemsets.

I. Data Preprocessing: [Code can be found in preprocess.py file]

- **Transaction Definition:**
  Each trip is treated as a transaction, and a subset of interesting columns is selected to represent items within the transaction. These columns capture relevant trip attributes and are transformed to create discrete item categories.
- **Selected items:**
  The following columns were chosen based on their relevance to trip characteristics and their potential to reveal meaningful patterns.
  - passenger_count
  - trip_distance
  - pickup_location
  - dropoff_location
  - rate_code
  - store_and_forward_flag
  - payment_type
  - fare_amount
  - tip_amount
  - toll_amount
  - total_amount
- **Preprocessing Steps:**
  - **Discretize Continuous Data**:
    Continuous attributes were grouped into meaningful categories to reduce granularity and make patterns more apparent.
    - passenger_count: <6, 6+
    - trip_distance: <1, 1-5, 5-10, 10+
    - fare_amount: <100, 100-500, 500-1000, 1000+
    - tip_amount: 0-50, 50-100, 100+
    - toll_amount: <50, 50-100, 100+
    - total_amount: <100, 100-500, 500-1000, 1000+
  - **Tagging and Labeling:**
    - Each column value is combined with its corresponding attribute name to create an item tag. For example, passenger_count:<6, fare_amount:<100, tip_amount:<50, etc.
  - **Item Generation:**
    - After discretization, a total of 554 unique item categories were identified across all transactions.
  - **Structuring Transactions:**
    - Each trip is assigned a unique transaction ID (tid). Items from the trip are then inserted into a new relation table in the form (tid, item).

II. Itemset Mining: [Code can be found in itemset_mining.py file]

- **Process Overview:** Using the items relation generated in the preprocessing step, frequent itemsets were mined as follows.
  1. **Generate L1:**
     a) Identify all items (1-itemsets) that satisfy the minimum support threshold.
  2. **Generate Candidates for k=2 (C2):**
     a) Combine items from L1 that belong to the same transaction to generate candidate 2-itemsets. Filter these candidates based on the minimum support threshold to form L2.
  3. **Iterative Process:**
     a) Repeat the previous step iteratively for k=3, k=4, and so on, generating and filtering candidates until no further frequent itemsets are found.
- **Results:**
  1. For the current dataset, itemsets were generated up to size 4 (k=4) with a minimum support threshold of 1000. Below are examples of mined itemsets along with their interpretations:
  2. **Mined Itemsets:**
     a) 1-Itemsets (k=1):
        (1) dolocationid:10 => Drop-off location: Queens, Baisley Park
        (2) pulocationid:41 => Pick-up location: Manhattan, Central Harlem
        (3) tipamount:<50 => Tip amount less than $50
        (4) tripdistance:10+ => Trip distance more than 10 miles

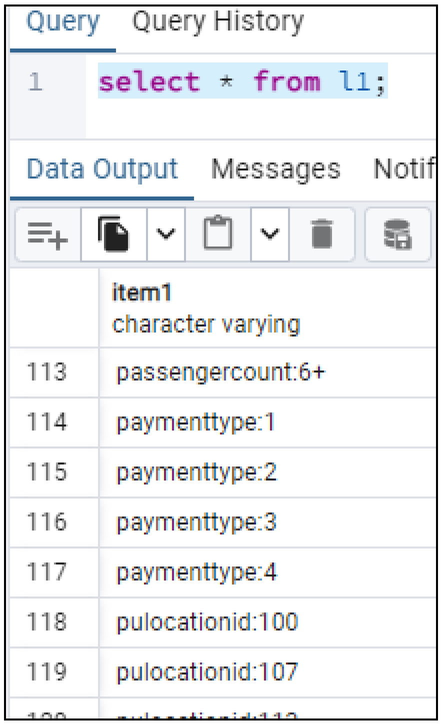
     b) 2-Itemsets (k=2):
        (1) dolocationid:10, fareamount:<100 => Drop-off location: Queens, Baisley Park | Fare less than $100
        (2) dolocationid:114, pulocationid:79 => Drop-off location: Manhattan, Greenwich Village South | Pick-up location: Manhattan, East Village
        (3) fareamount:<100, tipamount:<50 => Fare less than $100 | Tip amount less than $50
        (4) passengercount:<6, pulocationid:100 => Passenger count less than 6 | Pick-up location: Manhattan, Garment District

     c) 3-Itemsets (k=3):
        (1) dolocationid:1, passengercount:<6, tipamount:<50 => Drop-off location: Newark Airport (EWR) | Passenger count less than 6 | Tip amount less than $50

        (2) dolocationid:107, fareamount:<100, tripdistance:1-5 => Drop-off location: Manhattan, Gramercy | Fare less than $100 | Trip distance between 1-5 miles
        (3) dolocationid:113, passengercount:<6, paymenttype:1 => Drop-off location: Manhattan, Greenwich Village North | Passenger count less than 6 | Payment mode: Credit card
    d) **4-Itemsets (k=4):**
        (1) dolocationid:132, fareamount:<100, tollsamount:<50, tripdistance:10+ => Drop-off location: Manhattan, Gramercy | Fare less than $100 | Tolls less than $50 | Trip distance more than 10 miles
        (2) dolocationid:143, storeandfwdflag:N, tollsamount:<50, tripdistance:1-5 => Drop-off location: Manhattan, Lincoln Square West | No store-and-forward flag | Tolls less than $50 | Trip distance between 1-5 miles

3. **Possible Applications of the Results:**
    a) **Managing Services by Fare Patterns:**
        (1) The itemset dolocationid:10, fareamount:<100 indicates that trips ending in Queens, Baisley Park are more likely to have a fare below $100. This insight can help optimize service allocation and pricing strategies in this area to maximize profitability.
    b) **Route Optimization and Resource Allocation:**
        (1) The itemset dolocationid:114, pulocationid:79 highlights that trips between Greenwich Village South (drop-off) and East Village (pick-up) are common. Increasing the availability of taxis on this route could improve service efficiency and customer satisfaction.
    c) **Understanding Customer Behavior:**
        (1) The itemset dolocationid:113, passengercount:<6, paymenttype:1 suggests that trips to Greenwich Village North with fewer passengers often use credit card payments. This could inform marketing or promotional strategies targeted at frequent travelers in this region.

- By systematically interpreting the mined itemsets, we can derive actionable insights that enhance service planning, improve customer experience, and optimize operational efficiency.

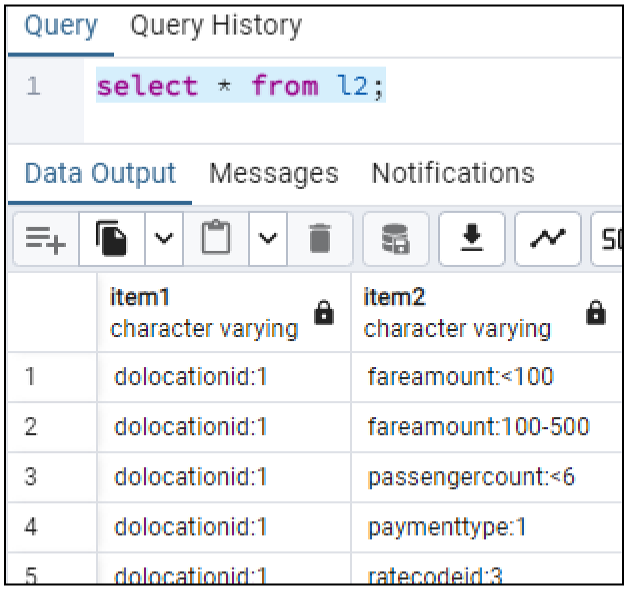

Query Query History

```sql
select * from l1;
```

Data Output Messages Notif

	item1 character varying
113	passengercount:6+
114	paymenttype:1
115	paymenttype:2
116	paymenttype:3
117	paymenttype:4
118	pulocationid:100
119	pulocationid:107

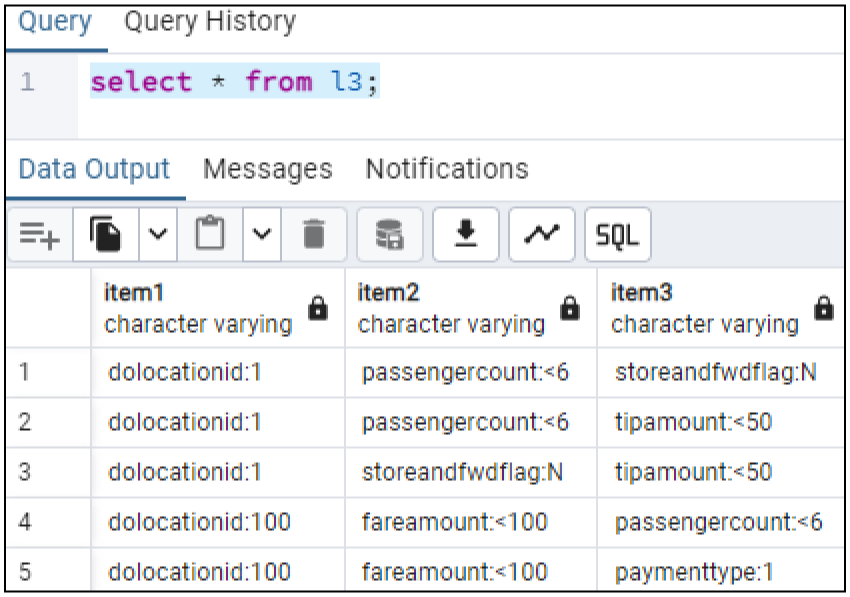
○


Query Query History

```sql
select * from l2;
```

Data Output Messages Notifications

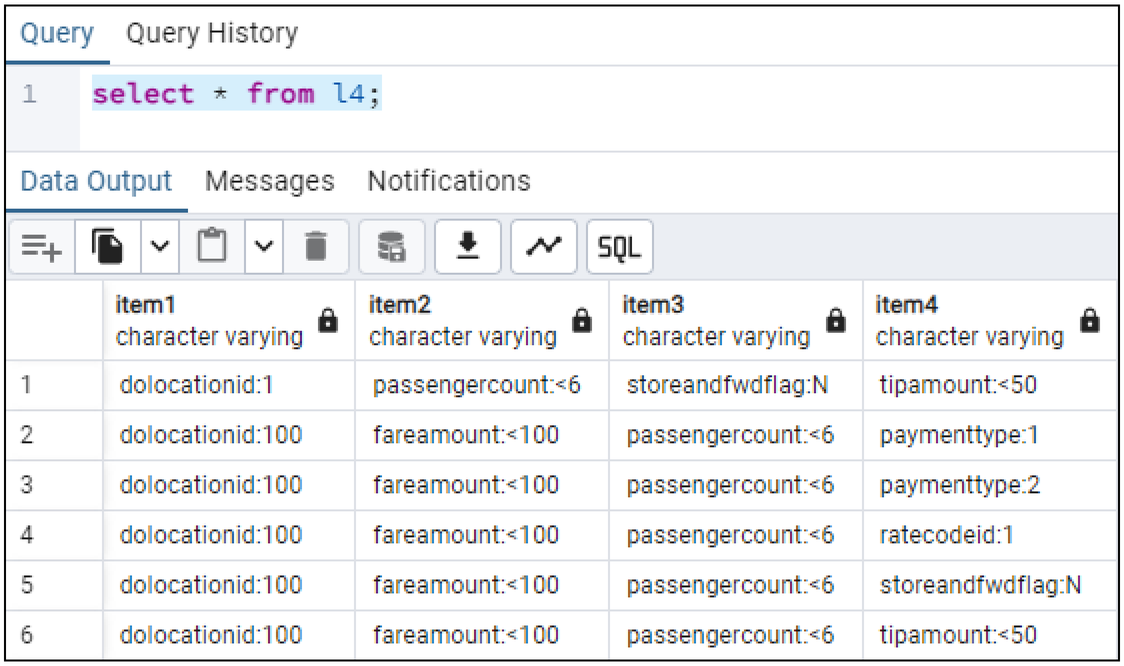
	item1 character varying	item2 character varying
1	dolocationid:1	fareamount:<100
2	dolocationid:1	fareamount:100-500
3	dolocationid:1	passengercount:<6
4	dolocationid:1	paymenttype:1
5	dolocationid:1	ratecodeid:3

- 
Query Query History

```
select * from l3;
```

Data Output Messages Notifications

	item1 character varying	item2 character varying	item3 character varying
1	dolocationid:1	passengercount:<6	storeandfwdflag:N
2	dolocationid:1	passengercount:<6	tipamount:<50
3	dolocationid:1	storeandfwdflag:N	tipamount:<50
4	dolocationid:100	fareamount:<100	passengercount:<6
5	dolocationid:100	fareamount:<100	paymenttype:1

- 
Query Query History

```
select * from l4;
```

Data Output Messages Notifications

	item1 character varying	item2 character varying	item3 character varying	item4 character varying
1	dolocationid:1	passengercount:<6	storeandfwdflag:N	tipamount:<50
2	dolocationid:100	fareamount:<100	passengercount:<6	paymenttype:1
3	dolocationid:100	fareamount:<100	passengercount:<6	paymenttype:2
4	dolocationid:100	fareamount:<100	passengercount:<6	ratecodeid:1
5	dolocationid:100	fareamount:<100	passengercount:<6	storeandfwdflag:N
6	dolocationid:100	fareamount:<100	passengercount:<6	tipamount:<50

Task 3: Association Rules **[Code can be found in association_rules.py file]**

Process Overview:

Using the frequent itemsets discovered during the itemset mining step, association rules were generated to identify relationships between items within transactions. The process involved the following steps:

a. **Support Calculation:**
    i. Support for a rule X→Y is calculated as:
        1. support(X->Y) = (No. of transactions with items X U Y) / Total number of transactions

b. **Confidence Calculation:**
    i. Confidence for a rule X→Y measures the likelihood that Y occurs given X:
        1. confidence(X->Y) = support(X U Y)/support(X)

c. **Rule Generation and Filtering:**
    i. For each k-itemset (e.g.,{A,B,C}), generate all possible rules by splitting the itemset. For example, possible rules from {A,B,C} include: A→BC, B→AC, C→AB, AB→C, AC→B, BC→A
    ii. Evaluate each rule based on the minimum support and confidence thresholds. Only rules satisfying both thresholds are retained.

d. **Thresholds Used:**
    i. Minimum Support: 0.08
    ii. Minimum Confidence: 0.8

These thresholds are configurable in the implementation.

Results and possible applications of the association rules identified:

1. [tipamount:<50] -> [passengercount:<6]

- Explanation: Trips with lower tips (under $50) tend to have fewer passengers (less than 6).

- Possible Reasons:

- Shorter trips or less crowded rides typically result in smaller tips
- Single passengers or small groups are less likely to generate large tips

- Applications:

- Taxi driver revenue prediction
- Understanding tipping behavior in different ride scenarios

2. [tollsamount:<50] -> [passengercount:<6]

- Explanation: Trips with lower toll costs are associated with fewer passengers.

- Possible Reasons:

- Shorter routes or routes avoiding toll roads
- Individual or small group travels

- Applications:

- Route optimization

  - Passenger group size prediction

3. [storeandfwdflag:N] -> [totalamount:<100]
 - Explanation: Non-store and forward trips tend to have lower total amounts.
 - Possible Reasons:
  - Store and forward flag indicates trips during network issues
  - Regular trips might be shorter or have fewer additional services
 - Applications:
  - Fare estimation
  - Understanding taxi dispatch system characteristics

4. [fareamount:<100] -> [passengercount:<6,storeandfwdflag:N]
 - Explanation: Lower fare amounts are associated with fewer passengers and non-store and forward trips.
 - Possible Reasons:
  - Shorter trips with fewer passengers
  - Regular, non-problematic taxi dispatches
 - Applications:
  - Fare structure analysis
  - Passenger demand forecasting

5. [passengercount:<6,storeandfwdflag:N] -> [fareamount:<100]
 - Explanation: Small passenger groups in non-store and forward trips tend to have lower fares.
 - Possible Reasons:
  - Shorter, more direct trips
  - Less complex travel scenarios
 - Applications:
  - Pricing strategy
  - Travel pattern understanding

6. [storeandfwdflag:N,tipamount:<50] -> [fareamount:<100,passengercount:<6]
 - Explanation: Non-store and forward trips with low tips are associated with lower fares and fewer passengers.
 - Possible Reasons:
  - Short, routine trips
  - Less complicated taxi services
 - Applications:
  - Service optimization
  - Tip and fare correlation analysis

7. [fareamount:<100,passengercount:<6,storeandfwdflag:N] -> [tipamount:<50]

- **Explanation:** Lower fare trips with few passengers and non-store and forward status tend to have smaller tips.
- **Possible Reasons:**
  - Short trips generate less gratitude or perceived service value
  - Less interaction or service complexity
- **Applications:**
  - Tipping behavior research
  - Service quality assessment

General Insights and Broader Applications:

- These rules suggest a strong correlation between trip characteristics:
  - Passenger count
  - Fare amount
  - Tip amount
  - Store and forward status
- Potential uses include:
  - Predictive modeling for taxi services
  - Pricing optimization
  - Understanding urban transportation patterns
  - Developing more efficient taxi dispatch systems

Note: Execution steps for itemset and association rule mining code.

i. python3 preprocess.py
ii. python3 itemset_mining.py
iii. python3 association_rules.py

Relational vs Document Database

We decided to go with our relational database for cleaning, itemset mining, and defining association rules. Our dataset is structured i.e. it is organized into tables with clearly defined relationships between different entities. Because of this, it naturally fits into a relational database. Relational databases are built to handle structured data and allow us to use their superior capability to perform joins and complex queries. These features are essential for our project.

Frequent itemset mining and association rule discovery involves running many queries to analyze patterns and relationships in the data. For example, frequent itemset mining looks at a large number of transactions to find items that often appear together. This process needs operations like filtering, grouping, and combining data, all of which relational databases excel at. Similarly, association rule discovery uses frequent itemsets to create rules and relationships between items. These tasks depend heavily on the ability to run efficient and complex queries, making a relational database the obvious choice.

On the other hand, while document databases like MongoDB have their strengths, such as flexibility in handling unstructured or semi-structured data, these features are not what we need for this project. Document databases are better for datasets that don't follow a fixed structure or schema, but our data is already structured. Performing joins and more complicated queries in a document database often requires extra work, such as restructuring the data, and it is much less straightforward and efficient.

Additionally, relational databases have good query optimization features that make them better suited for tasks like item set mining and rule generation, where many iterative queries are involved. Document databases, while flexible, lack this level of efficiency for complex data processing.

The choice of a relational database aligns with the structured nature of our data and the analytical needs of the project:

1. **Data Cleaning**: Relational databases simplify the identification and correction of anomalies like missing values, duplicates, or inconsistent records through SQL queries.
2. **Itemset Mining**: The ability to perform efficient joins, grouping, and aggregations is crucial for discovering frequent itemsets.
3. **Association Rules**: With the use of relational databases handling the iterative processing and evaluation of rules becomes easy.
4. **Efficiency and Accuracy**: Relational databases ensure data integrity and enforce constraints, which are critical for such dataset to produce good reliable results.