startegies_multiple_transformations

```sql
SELECT
  CASE
    WHEN zipcount < 100 THEN 'other'
    ELSE zip
  END AS zip_recoded,
  sum(zipcount) AS zipsum
FROM (
  SELECT
    zip,
    count(*) AS zipcount
  FROM evanston311
  GROUP BY zip
) AS fullcounts
GROUP BY zip_recoded
ORDER BY zipsum DESC;
```

Explanation:

- This SQL query analyzes the evanston311 table to summarize the number of occurrences of different zip codes. It groups zip codes with fewer than 100 occurrences into an "other" category and then sums the counts for each zip code (or the "other" category). Finally, it orders the results by the total count in descending order, showing which zip codes (or "other") have the highest number of occurrences.

```sql
-- Fill in the command below with the name of the temp table
DROP TABLE IF EXISTS recode;

-- Create and name the temporary table
CREATE TEMP TABLE recode AS
-- Write the select query to generate the table with distinct values of category and standardized
values
 SELECT DISTINCT category,
     RTRIM(split_part(category, '-', 1)) AS standardized
  -- What table are you selecting the above values from?
  FROM evanston311;

-- Look at a few values before the next step
SELECT DISTINCT standardized
 FROM recode
 WHERE standardized LIKE 'Trash%Cart'
  OR standardized LIKE 'Snow%Removal%';
```

Explanation:

- This SQL code first drops a temporary table named recode if it already exists. Then, it creates a temporary table recode by selecting distinct values from the evanston311 table. The standardized column is created by taking the part of the category column before the first hyphen (-), using split_part, and removing trailing whitespace using RTRIM. Finally, it selects distinct values from the standardized column of the newly created recode table, filtering for rows where standardized contains ‘Trash Cart’ or ‘Snow Removal’. The use of LIKE with % acts as a wildcard, matching any string containing those phrases.

```
-- Code from previous step
DROP TABLE IF EXISTS recode;

CREATE TEMP TABLE recode AS
 SELECT DISTINCT category,
     rtrim(split_part(category, '-', 1)) AS standardized
  FROM evanston311;

-- Update to group trash cart values
UPDATE recode
  SET standardized='Trash Cart'
 WHERE standardized LIKE 'Trash%Cart';

-- Update to group snow removal values
UPDATE recode
  SET standardized='Snow Removal'
 WHERE standardized LIKE 'Snow%Removal%';

-- Examine effect of updates
SELECT DISTINCT standardized
 FROM recode
 WHERE standardized LIKE 'Trash%Cart'
  OR standardized LIKE 'Snow%Removal%';
```

Explanation:

- This SQL code snippet cleans and standardizes categories from a table named evanston311. It first creates a temporary table recode containing unique categories and a new ‘standardized’ column. This column is created by taking the first part of the category (before the first hyphen), removing trailing spaces. Then, it updates the recode table to group similar categories like “Trash Cart” and “Snow Removal” into standardized labels, using LIKE for pattern matching. Finally, it verifies the effect of these updates.

```sql
-- Code from previous step
DROP TABLE IF EXISTS recode;

CREATE TEMP TABLE recode AS
 SELECT DISTINCT category,
       rtrim(split_part(category, '-', 1)) AS standardized
   FROM evanston311;

-- Standardize 'Trash Cart' entries
UPDATE recode SET standardized='Trash Cart'
 WHERE standardized LIKE 'Trash%Cart';

-- Standardize 'Snow Removal' entries
UPDATE recode SET standardized='Snow Removal'
 WHERE standardized LIKE 'Snow%Removal%';

-- Update to group unused/inactive values
UPDATE recode
  SET standardized='UNUSED'
 WHERE standardized IN ('THIS REQUEST IS INACTIVE...Trash Cart',
                '(DO NOT USE) Water Bill',
                'DO NOT USE Trash',
                'NO LONGER IN USE');

-- Examine effect of updates
SELECT DISTINCT standardized
 FROM recode
 ORDER BY standardized;
```

Explanation:

- This SQL code snippet cleans and standardizes categories from a table named evanston311. It creates a temporary table recode containing distinct categories and a new standardized column. The code then uses UPDATE statements to standardize variations of “Trash Cart” and “Snow Removal” and groups various inactive or unused categories under the label “UNUSED”. Finally, it selects the distinct standardized categories to show the result of the cleaning process.

```sql
-- Code from previous step
DROP TABLE IF EXISTS recode;
CREATE TEMP TABLE recode AS
 SELECT DISTINCT category,
       rtrim(split_part(category, '-', 1)) AS standardized
 FROM evanston311;
UPDATE recode SET standardized='Trash Cart'
 WHERE standardized LIKE 'Trash%Cart';
UPDATE recode SET standardized='Snow Removal'
```

```sql
 WHERE standardized LIKE 'Snow%Removal%';
UPDATE recode SET standardized='UNUSED'
 WHERE standardized IN ('THIS REQUEST IS INACTIVE...Trash Cart',
          '(DO NOT USE) Water Bill',
          'DO NOT USE Trash', 'NO LONGER IN USE');

-- Select the recoded categories and the count of each
SELECT standardized, COUNT(*)
-- From the original table and table with recoded values
  FROM evanston311
     LEFT JOIN recode
     -- What column do they have in common?
     ON evanston311.category = recode.category
 -- What do you need to group by to count?
 GROUP BY standardized
 -- Display the most common val values first
 ORDER BY COUNT DESC;
```

Explanation:

- This SQL code first creates a temporary table recode to standardize categories from the evanston311 table. It extracts the part of the category before the first hyphen, removes trailing whitespace, and then applies further updates to standardize specific similar categories (like various phrases relating to “Trash Cart”). Finally, it joins evanston311 with recode to count the occurrences of each standardized category, ordering the results by frequency.

```sql
-- To clear table if it already exists
DROP TABLE IF EXISTS indicators;

-- Create the indicators temp table
CREATE TEMP TABLE indicators AS
  -- Select id
  SELECT id,
       -- Create the email indicator (find @)
       CAST (description LIKE '%@%' AS integer) AS email,
       -- Create the phone indicator
       CAST (description LIKE '%___-___-____%' AS integer) AS phone
   -- What table contains the data?
   FROM evanston311;

-- Inspect the contents of the new temp table
SELECT *
  FROM indicators;
```

Explanation:

- This SQL code first drops a table named indicators if it already exists. Then, it creates a temporary table called indicators by selecting the id column from the evanston311 table and adding two new columns: email and phone. The email column is 1 if the description contains “@” and 0 otherwise; the phone column is 1 if the description matches a simple phone number pattern and 0 otherwise. Finally, it selects all data from the newly created indicators table to display its contents. The LIKE operator with wildcards (% and _) is used for pattern matching within the descriptions.

```sql
-- Drop the temporary table if it already exists
DROP TABLE IF EXISTS indicators;

-- Create the temporary table with indicators
CREATE TEMP TABLE indicators AS
 SELECT id,
     CAST(description LIKE '%@%' AS integer) AS email,
     CAST(description LIKE '%___-___-____%' AS integer) AS phone
  FROM evanston311;

-- Select priority and compute the proportion of rows with each indicator
SELECT priority,
    SUM(email)/COUNT(*)::numeric AS email_prop,
    SUM(phone)/COUNT(*)::numeric AS phone_prop
 FROM evanston311
    LEFT JOIN indicators
    ON evanston311.id = indicators.id
 GROUP BY priority;
```

Explanation:

- This SQL code analyzes the evanston311 table to determine the proportion of requests containing email addresses and phone numbers for each priority level. It creates a temporary table indicators to flag the presence of these indicators (email @ symbol and a phone number pattern) in the descriptions. Then, it joins this temporary table back to the original table and calculates the proportion of emails and phones for each priority using aggregate functions. The ::numeric cast ensures that the division results in a floating-point number representing the proportion.