

Dataset: Cohort_Raw

Performing the Exploratory Data Analysis

Step 1: Understand the Dataset

- View the first few rows to get an overview of the data.
- Check column names, data types, and the total number of records.

Research how to inspect a table's structure and data in PostgreSQL.

PostgreSQL Queries to Use:

1. View First Few Rows:

```
SELECT *
FROM cohort_raw
LIMIT 5;
```

🔍 This gives you a quick peek at the dataset — ideal for spotting structure or obvious issues.

	cohort_id	cohort_code	start_date	end_date	size
1	Cohort#	B456514	1.6805E+12	1.68296E+12	1500
2	Cohort#	B328821	1.67385E+12	1.67691E+12	1000
3	Cohort#	B289256	1.6684E+12	1.67108E+12	100000
4	Cohort#	B0VCB0F	1.66389E+12	1.66389E+12	40
5	Cohort#	B908347	1.67324E+12	1.67631E+12	100000

2. Check Column Names & Data Types:

```
SELECT column_name, data_type
```

```
FROM information_schema.columns  
WHERE table_name = 'cohort_raw';
```

- This tells you what each column is, and whether it's stored as text, numeric, timestamp, etc.

The screenshot shows a SQL query editor interface. At the top, there is a toolbar with tabs for "Query" and "Query History". Below the toolbar, the query text is displayed:

```
1 SELECT column_name, data_type  
2 FROM information_schema.columns  
3 WHERE table_name = 'cohort_raw';  
4
```

To the right of the query text is a button labeled "Execute script" with the F5 key icon. Below the query text, the results are shown in a table titled "Data Output". The table has two columns: "column_name" and "data_type". The data is as follows:

column_name	data_type
name	character varying
size	integer
cohort_id	text
cohort_code	text
start_date	text
end_date	text

3. Count Total Rows:

```
SELECT COUNT(*) FROM cohort_raw;
```

The screenshot shows a SQL query editor interface. At the top, there is a toolbar with tabs for "Query" and "Query History". Below the toolbar, the query text is displayed:

```
1 SELECT COUNT(*) FROM cohort_raw;
```

To the right of the query text, the results are shown in a table titled "Data Output". The table has one column: "count". The data is as follows:

count
639

Optional: Full Table Structure (DDL)

If you want to inspect the table's SQL definition:

```
SELECT * FROM pg_catalog.pg_tables WHERE tablename = 'cohort_raw';
```

	schemaname	tablename	tableowner	tablespace	hasindexes	hasrules	hastriggers	rowsecurity
1	public	cohort_raw	postgres	[null]	false	false	false	false

Step 2: Summarize the Data

- Generate basic statistics like mean, median, min, max, and standard deviation to understand the distribution.

Explore how to compute summary statistics in SQL.

Since the `CohortRaw(in).csv` dataset contains one numeric field of interest — `size` (number of learners in each cohort) — we'll compute:

- Minimum
- Maximum
- Mean (average)
- Median
- Standard deviation
- Count (just for completeness)

SQL Queries for Summary Statistics (PostgreSQL):

1. Basic Descriptive Stats:

```
SELECT
    MIN(size) AS min_size,
    MAX(size) AS max_size,
    AVG(size) AS mean_size,
    STDDEV(size) AS std_dev,
    COUNT(size) AS total_records
FROM cohort_raw;
```

The screenshot shows a PostgreSQL query editor interface. At the top, there are tabs for 'Query' and 'Query History'. Below the tabs is a code editor containing the provided SQL query. The code editor has line numbers on the left and syntax highlighting for keywords like 'SELECT', 'MIN', 'MAX', etc.

Below the code editor is a 'Data Output' tab, which is currently selected. It displays the results of the query in a table format. The table has one row and six columns:

	min_size	max_size	mean_size	std_dev	total_records
1	3	100000	5741.4241001564945227	20994.26661203	639

At the bottom right of the table area, it says 'Showing rows: 1 to 1'.

2. Median (PostgreSQL does not have a built-in MEDIAN() function, but you can get it like this):

```
SELECT PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY size) AS median_size
FROM cohort_raw;
```

The screenshot shows a PostgreSQL query editor interface. At the top, there are tabs for 'Query' and 'Query History'. Below the tabs is a code editor containing the provided SQL query. The code editor has line numbers on the left and syntax highlighting for keywords like 'SELECT', 'PERCENTILE_CONT', 'WITHIN GROUP', etc.

Below the code editor is a 'Data Output' tab, which is currently selected. It displays the results of the query in a table format. The table has one row and one column:

median_size
800

At the bottom right of the table area, it says 'Showing rows'.

What These Queries Will Show:

Statistic	Description
MIN(size)	Smallest cohort size
MAX(size)	Largest cohort size
AVG(size)	Average cohort size
STDDEV(size)	Standard deviation — shows how spread out the sizes are
COUNT(size)	Total number of cohort records
MEDIAN(size)	Middle value of all sizes (better indicator than average if there are outliers)

Step 3: Identify Missing and Duplicate Values

- Detect missing data to determine if any cleaning is needed.
- Identify duplicate records that may need removal.

Look up methods to check for missing values and duplicates in PostgreSQL.

Part 1: Detect Missing (NULL) Values

1. Missing Count for Each Column

```
SELECT
    COUNT(*) AS total_rows,
    COUNT(cohort_id) AS cohort_id_not_null,
    COUNT(cohort_code) AS cohort_code_not_null,
    COUNT(start_date) AS start_date_not_null,
    COUNT(end_date) AS end_date_not_null,
    COUNT(size) AS size_not_null
FROM cohort_raw;
```

Query Query History 

```

1 ▼ SELECT
2   COUNT(*) AS total_rows,
3   COUNT(cohort_id) AS cohort_id_not_null,
4   COUNT(cohort_code) AS cohort_code_not_null,
5   COUNT(start_date) AS start_date_not_null,
6   COUNT(end_date) AS end_date_not_null,
7   COUNT(size) AS size_not_null
8   FROM cohort_raw;
9

```

Data Output Messages Notifications

Showing rows: 1 to 1

	total_rows	cohort_id_not_null	cohort_code_not_null	start_date_not_null	end_date_not_null	size_not_null
1	639	639	639	639	639	639

2. Show Rows With Any NULL

```

SELECT *
FROM cohort_raw
WHERE cohort_id IS NULL
  OR cohort_code IS NULL
  OR start_date IS NULL
  OR end_date IS NULL
  OR size IS NULL;

```

Query Query History 

```

1 ▼ SELECT *
2   FROM cohort_raw
3   WHERE cohort_id IS NULL
4     OR cohort_code IS NULL
5     OR start_date IS NULL
6     OR end_date IS NULL
7     OR size IS NULL;
8

```

Data Output Messages Notifications

	cohort_id	cohort_code	start_date	end_date	size
	text	text	text	text	integer

Part 2: Detect Duplicate Records

3. Fully Duplicate Rows (every column repeated)

```
SELECT cohort_id, cohort_code, start_date, end_date, size, COUNT(*) AS count
FROM cohort_raw
GROUP BY cohort_id, cohort_code, start_date, end_date, size
HAVING COUNT(*) > 1;
```

- No duplications in this.

The screenshot shows a SQL query editor interface. At the top, there are tabs for 'Query' and 'Query History'. Below the tabs is a code editor containing the following SQL query:

```
1 SELECT cohort_id, cohort_code, start_date, end_date, size, COUNT(*) AS count
2 FROM cohort_raw
3 GROUP BY cohort_id, cohort_code, start_date, end_date, size
4 HAVING COUNT(*) > 1;
5
```

Below the code editor is a toolbar with icons for file operations like new, open, save, and execute. The main area is labeled 'Data Output' and shows a table structure with columns: cohort_id, cohort_code, start_date, end_date, size, and count. The 'size' and 'count' columns have a lock icon next to them, indicating they are calculated or derived fields.

4. Partially Duplicate Rows (e.g., duplicate cohort_code)

To check if a column supposed to be unique has duplicates:

```
SELECT cohort_code, COUNT(*) AS count
FROM cohort_raw
GROUP BY cohort_code
HAVING COUNT(*) > 1;
```

- No partial duplications also.

```

Query   Query History
1 ✓ SELECT cohort_code, COUNT(*) AS count
2   FROM cohort_raw
3   GROUP BY cohort_code
4   HAVING COUNT(*) > 1;
5

Data Output  Messages  Notifications
SQL
cohort_code  count
text        bigint

```

What to Look For:

Feature	Insight
NULL Values	Indicate incomplete or missing data that may need cleaning
Fully Duplicate Rows	Often accidental — likely safe to remove
Partially Duplicate Rows	May indicate inconsistent or repeated identifiers (e.g., cohort_code)

Step 4: Spot Outliers and Anomalies

- Use techniques like the Interquartile Range (IQR) to find unusual values that might distort analysis.

Research how outliers are detected using SQL and statistical methods.

Interquartile Range (IQR) Method Overview

Steps:

1. Calculate Q1 (25th percentile) and Q3 (75th percentile)
2. Compute $IQR = Q3 - Q1$
3. Define outlier bounds:
 - o Lower Bound = $Q1 - 1.5 \times IQR$
 - o Upper Bound = $Q3 + 1.5 \times IQR$

4. Flag values outside these bounds as outliers
-

PostgreSQL Queries to Detect Outliers

1. Get Q1, Q3, and IQR

```
SELECT
    PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY size) AS q1,
    PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY size) AS q3
FROM cohort_raw;
```

The screenshot shows a PostgreSQL query editor interface. At the top, there are tabs for "Query" and "Query History", with "Query" selected. Below the tabs is a code editor containing the following SQL query:

```
1 SELECT
2     PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY size) AS q1,
3     PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY size) AS q3
4 FROM cohort_raw;
5
```

Below the code editor is a "Data Output" tab, which is currently active. It displays a results table with two columns: "q1" and "q3". The "q1" column has a value of 500, and the "q3" column has a value of 1500. The table has a header row and a data row.

Once you have Q1 and Q3:

Q1= 500 , Q3= 1500

$$\begin{aligned} \text{IQR} &= Q3 - Q1 \\ \text{IQR} &= 1500 - 500 \\ \text{IQR} &= 1000 \\ \text{Lower Bound} &= Q1 - 1.5 * \text{IQR} \\ &= 500 - 1.5 * 1000 \\ &= \mathbf{498500} \\ \text{Upper Bound} &= Q3 + 1.5 * \text{IQR} \\ &= 1500 + 1.5 * 1000 \\ &= \mathbf{1501500} \end{aligned}$$

2. Find Outliers Using Calculated Bounds

Assume you manually calculate and plug in the bounds, e.g.:

```
SELECT *
FROM cohort_raw
WHERE size < 498500
    OR size > 1501500;
```

[Query](#) [Query History](#)

```
1 ▾ SELECT *
2   FROM cohort_raw
3   WHERE size < 498500
4     OR size > 1501500;
```

[Data Output](#) [Messages](#) [Notifications](#)

Showing rows: 1 to

	cohort_id	cohort_code	start_date	end_date	size
	text	text	text	text	integer
1	Cohort#	B456514	1.6805E+12	1.68296E+12	1500
2	Cohort#	B328821	1.67385E+12	1.67691E+12	1000
3	Cohort#	B289256	1.6684E+12	1.67108E+12	100000
4	Cohort#	B0VCB0F	1.66389E+12	1.66389E+12	40
5	Cohort#	B908347	1.67324E+12	1.67631E+12	100000
6	Cohort#	B306047	1.67324E+12	1.67631E+12	10000
7	Cohort#	B883644	1.65665E+12	1.65924E+12	1500
8	Cohort#	B280844	1.6684E+12	1.67108E+12	100000
9	Cohort#	B466039	1.66477E+12	1.66745E+12	100000
10	Cohort#	B606140	1.67324E+12	1.67631E+12	10000

Total rows: 639 Query complete 00:00:00.202

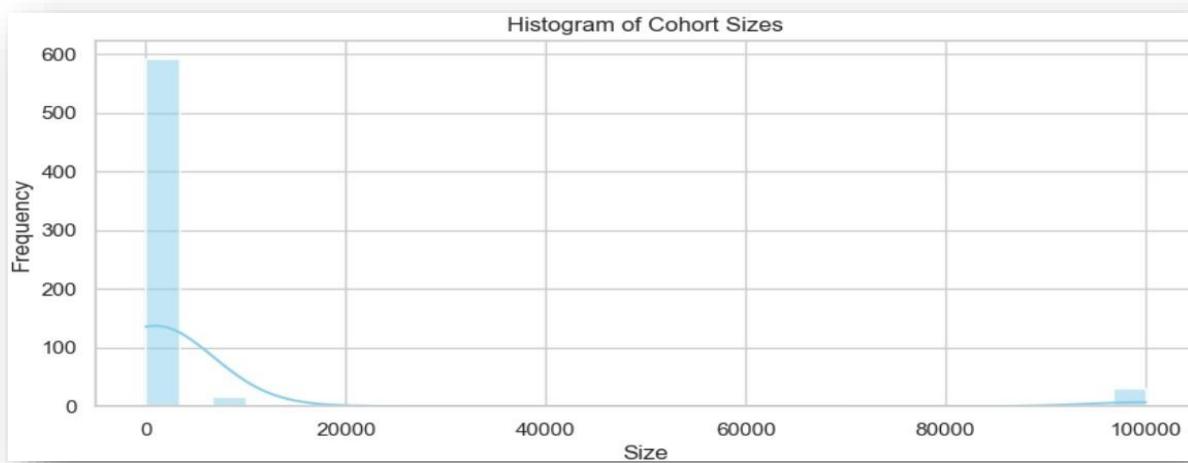
What to Look For:

Metric	Purpose
Q1 & Q3	Measure the range where most data lies
Outliers	Any value falling far outside this range
Next Step	Review outliers — correct, cap, or remove as needed

Step 5: Visualize Data Trends

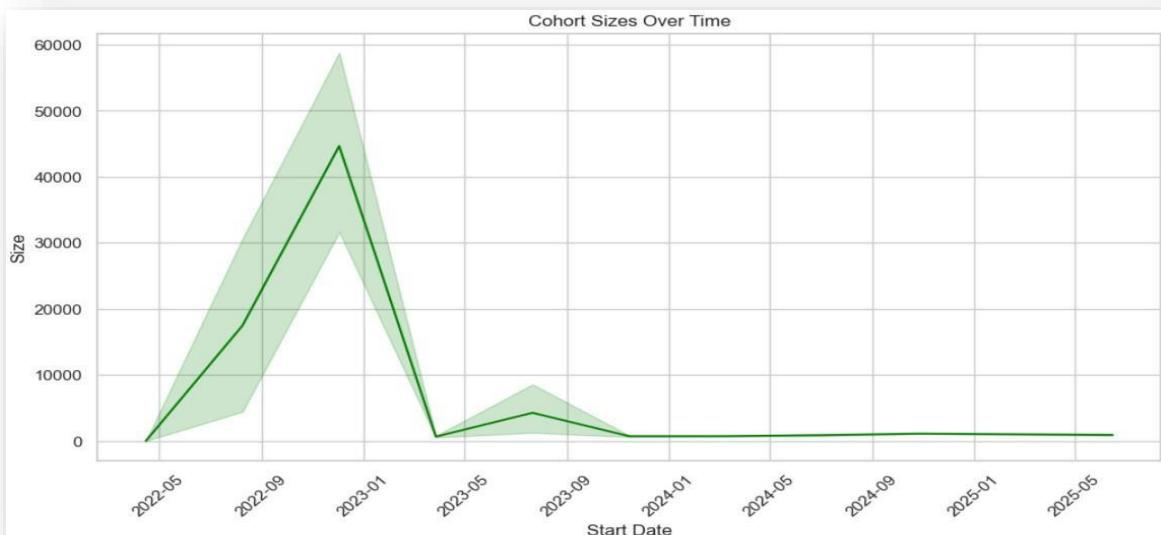
- Use graphs like histograms, line charts, and box plots to identify patterns.
- Analyze relationships between different variables.

Performed in Python

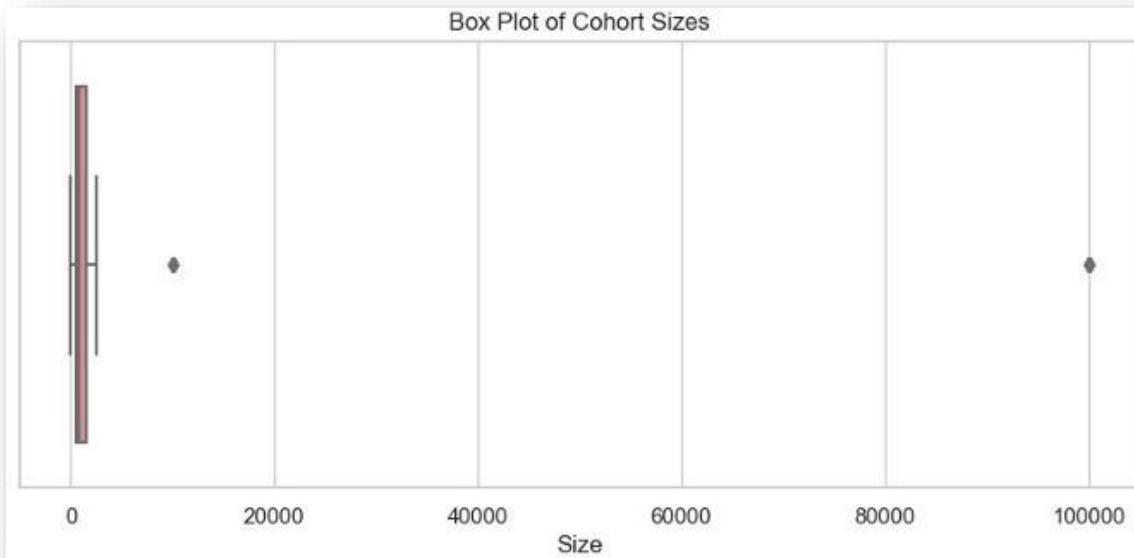


Histogram

Line Chart



Box Plot



Dataset: Learneropportunity_Raw

Performing the Exploratory Data Analysis

Step 1: Understand the Dataset

- View the first few rows to get an overview of the data.
- Check column names, data types, and the total number of records.

Research how to inspect a table's structure and data in PostgreSQL.

Objective:

Understand the structure of the dataset by inspecting:

- First few rows
 - Column names
 - Data types
 - Total number of records
-

Queries in PostgreSQL:

1. View First Few Rows:

```
SELECT *  
FROM learneropportunity_raw  
LIMIT 5;
```

This gives a quick snapshot of the data format and sample values.

The screenshot shows a database interface with a query editor and a results table.

Query:

```
1 ✓ SELECT *
2   FROM learneropportunity_raw
3   LIMIT 5;
4
```

Execute script button (F5)

Data Output:

	enrollment_id text	learner_id text	assigned_cohort text	apply_date timestamp without time zone	status integer
1	Learner#4e6f78a9-f9b2-4352-ad22-d43dc46f5ff7	Opportunity#000000010WCBS50CYGDX97ES4	BAM6HBR	2024-04-10 06:28:31.902	1070
2	Learner#4e79d245-3436-4fec-9906-901a03639ad1	Opportunity#000000010WCBS50CYGDX97ES4	BAM6HBR	2023-11-15 03:08:17.442	1120
3	Learner#4e9f5cb5-0576-4dbc-b7f5-1faef5f29b2df	Opportunity#000000010WCBS50CYGDX97ES4	BAM6HBR	2024-04-06 14:07:01.322	1070
4	Learner#4ea61aa9-17da-4b60-9872-359b8e1e16...	Opportunity#000000010WCBS50CYGDX97ES4	BT4YTCR	2024-04-11 22:01:13.548	1070
5	Learner#4eb218c7-467a-470a-9e3e-a2b7bc649e18	Opportunity#000000010WCBS50CYGDX97ES4	BT4YTCR	2024-10-22 15:44:13.402	1120

2. Check Column Names and Data Types:

```
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = 'learneropportunity_raw';
```

Lists each column with its data type (e.g., text, integer, timestamp).

The screenshot shows a database interface with a query editor and a results table.

Query:

```
1 ✓ SELECT column_name, data_type
2   FROM information_schema.columns
3   WHERE table_name = 'learneropportunity_raw';
4
```

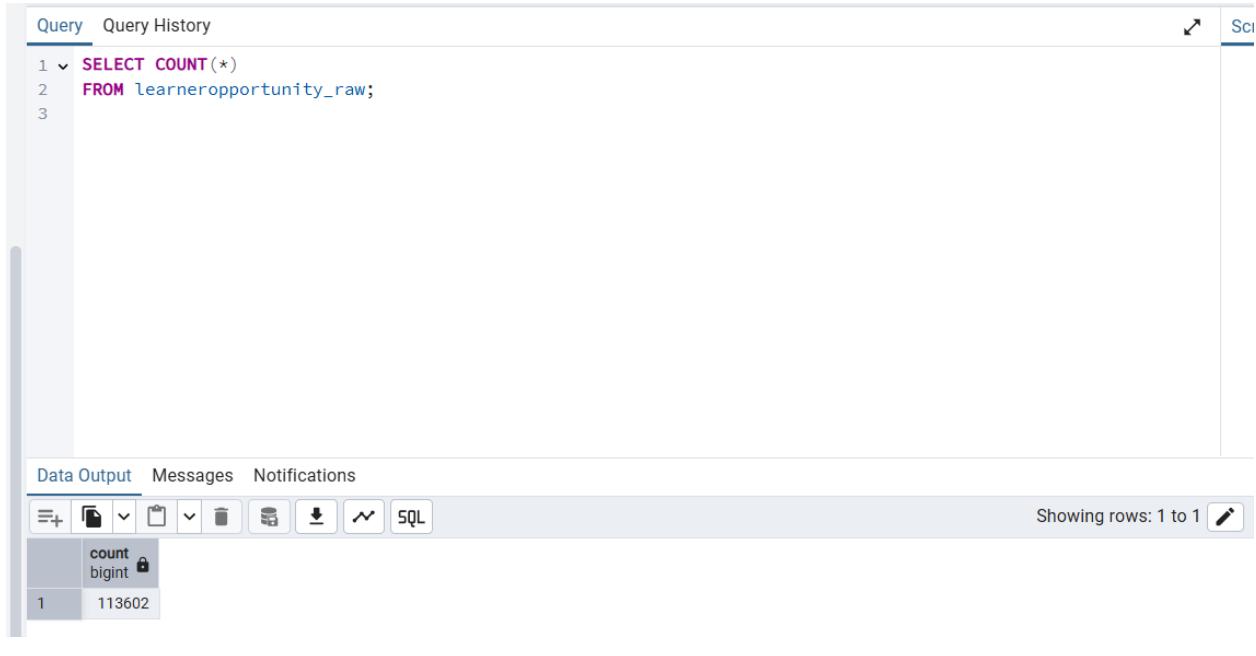
Data Output:

	column_name name	data_type character varying
1	apply_date	timestamp without time zone
2	status	integer
3	enrollment_id	text
4	learner_id	text
5	assigned_cohort	text

3. Get Total Number of Records:

```
SELECT COUNT(*)
FROM learneropportunity_raw;
```

- Confirms how many records are present in the dataset.



The screenshot shows a SQL query editor interface. At the top, there are tabs for 'Query' and 'Query History'. Below the tabs, the SQL code is displayed:

```
1 SELECT COUNT(*)
2 FROM learneropportunity_raw;
3
```

At the bottom of the editor, there is a 'Data Output' tab, which is currently selected. It displays the results of the query:

	count	bigint
1	113602	

On the right side of the interface, there are icons for saving, opening, and other database operations. A status bar at the bottom right indicates 'Showing rows: 1 to 1'.

Step 2: Summarize the Data

- Generate basic statistics like mean, median, min, max, and standard deviation to understand the distribution.

Explore how to compute summary statistics in SQL.

QL Queries for Summary Statistics (PostgreSQL):

1. Basic Descriptive Stats:

```
SELECT
    MIN(status) AS min_status,
    MAX(status) AS max_status,
```

```
AVG(status) AS mean_status,  
STDDEV(status) AS std_dev,  
COUNT(status) AS total_records  
FROM learneropportunity_raw;
```

The screenshot shows a PostgreSQL query editor interface. At the top, there are tabs for 'Query' and 'Query History'. Below the tabs is a code editor containing the following SQL script:

```
1 SELECT  
2     MIN(status) AS min_status,  
3     MAX(status) AS max_status,  
4     AVG(status) AS mean_status,  
5     STDDEV(status) AS std_dev,  
6     COUNT(status) AS total_records  
7 FROM learneropportunity_raw;  
8
```

On the right side of the code editor, there is a button labeled 'Execute script' with an 'F5' icon. Below the code editor is a 'Data Output' tab, which is currently selected. The data output section displays a table with the following columns and data:

	min_status	max_status	mean_status	std_dev	total_records
1	1010	1120	1068.1924508005925090	21.0275258469164881	113416

At the bottom right of the data output area, it says 'Showing rows: 1 to 1'.

2. Median:

(PostgreSQL does not have a built-in MEDIAN() function, but you can calculate it using PERCENTILE_CONT.)

```
SELECT  
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY status) AS median_status  
FROM learneropportunity_raw;
```

The screenshot shows a database query editor interface. At the top, there's a toolbar with tabs for 'Query' and 'Query History'. A button labeled 'Execute script' with 'F5' is visible. Below the toolbar, the SQL query is displayed:

```
1 SELECT
2     PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY status) AS median_status
3 FROM learneropportunity_raw;
4
```

Below the query, the results are shown in a table titled 'Data Output'. The table has one row with one column. The column header is 'median_status' and its value is 'double precision'. The data row shows a single value '1070'. The table includes standard data export icons (CSV, Excel, PDF, etc.) and a 'SQL' link.

At the bottom right of the results area, it says 'Showing rows: 1 to 1' with a small edit icon.

What These Queries Will Show:

Statistic	Description
MIN(status)	Smallest status value (possibly earliest stage)
MAX(status)	Highest status code (possibly latest/confirmed/complete)
AVG(status)	Mean status value — useful if numerical progression has meaning
STDDEV(status)	Standard deviation — shows variation or tightness of clustering
COUNT(status)	Total number of status entries (non-null)
MEDIAN(status)	Middle status code — useful for detecting skew from outliers

Step 3: Identify Missing and Duplicate Values

- Detect missing data to determine if any cleaning is needed.
- Identify duplicate records that may need removal.

Look up methods to check for missing values and duplicates in PostgreSQL

Objective:

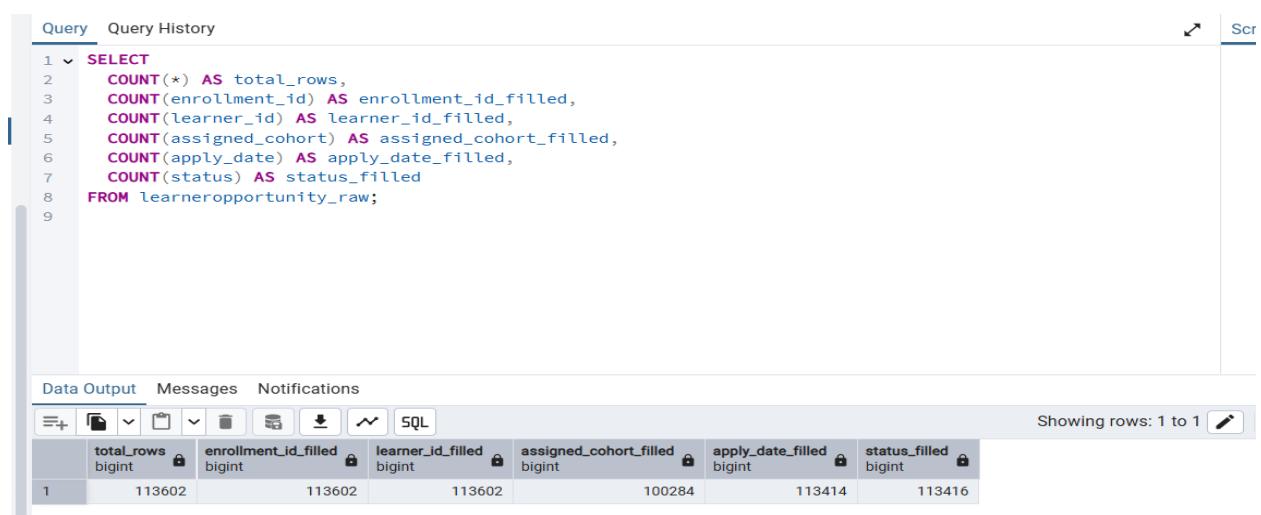
- Detect missing (NULL) values
- Identify fully or partially duplicate records

Part A: Check for Missing Values

1. Count of NULLs for All Columns

```
SELECT
    COUNT(*) AS total_rows,
    COUNT(enrollment_id) AS enrollment_id_filled,
    COUNT(learner_id) AS learner_id_filled,
    COUNT(assigned_cohort) AS assigned_cohort_filled,
    COUNT(apply_date) AS apply_date_filled,
    COUNT(status) AS status_filled
FROM learneropportunity_raw;
```

Compare filled counts with total rows to detect missing values.



	total_rows	enrollment_id_filled	learner_id_filled	assigned_cohort_filled	apply_date_filled	status_filled
1	113602	113602	113602	100284	113414	113416

2. View Records With Any NULL

```
SELECT *
FROM learneropportunity_raw
WHERE enrollment_id IS NULL
    OR learner_id IS NULL
    OR assigned_cohort IS NULL
    OR apply_date IS NULL
    OR status IS NULL;
```

Allows visual inspection of missing fields.

The screenshot shows a database interface with a query history tab and a scratch pad tab. The query history tab contains the following SQL code:

```
1 ✓ SELECT *
2   FROM learneropportunity_raw
3   WHERE enrollment_id IS NULL
4     OR learner_id IS NULL
5     OR assigned_cohort IS NULL
6     OR apply_date IS NULL
7     OR status IS NULL;
```

The results table has columns: enrollment_id, learner_id, assigned_cohort, apply_date, and status. The apply_date column is timestamp without time zone and the status column is integer. The results show 13320 rows, all with null values in at least one of the specified fields.

	enrollment_id	learner_id	assigned_cohort	apply_date	status
1	Learner#5cdba097-3fcf-47b0-b911-194a1485f8fe	Opportunity#000000010WCBS50CYGDX97...	[null]	2024-09-27 15:48:08.083	1030
2	Learner#b9bb5b13-16e5-4765-ad8a-b13868545...	Opportunity#000000010WCBS50CYGDX97...	[null]	2025-02-09 16:12:44.387	1030
3	Learner#be016e3b-25c4-4d01-8f6d-92e0a05fe1...	Opportunity#000000010WCBS50CYGDX97...	[null]	2024-10-02 22:36:50.892	1030
4	Learner#fdfe1f4c-cfb7-44e9-a56c-cde1fa4926b	Opportunity#000000010WCBS50CYGDX97...	[null]	2024-09-27 11:24:55.622	1030
5	Learner#f1b19a00-fa79-4031-8131-22e9cb8eda...	Opportunity#000000010WCBS50CYGDX97...	[null]	2024-09-26 13:50:15.117	1030
6	Learner#f22613d0-3251-4c53-be96-7757d5699...	Opportunity#000000010WCBS50CYGDX97...	[null]	2025-02-09 03:39:25.5	1030
7	Opportunity#	Opportunity#000000010X2B85MQE0B6RN...	[null]	[null]	[null]
8	Opportunity#	Opportunity#000000010X2B85MQE0B6RN...	[null]	[null]	[null]
9	Learner#2e0651aa-0601-487c-9187-b7beff7abe...	Opportunity#000000010XDF0W6XBJ0Y5V...	[null]	2024-09-25 08:07:44	1030
10	Learner#31409662-af3c-454a-9fb0-765df2fb36c5	Opportunity#000000010XDF0W6XBJ0Y5V...	[null]	2024-09-25 03:52:58.329	1030

Total rows: 13320 Query complete 00:00:01.401

Part B: Check for Duplicates

3. Fully Duplicate Records

```
SELECT enrollment_id, learner_id, assigned_cohort, apply_date, status,
COUNT(*) AS count
FROM learneropportunity_raw
GROUP BY enrollment_id, learner_id, assigned_cohort, apply_date, status
HAVING COUNT(*) > 1;
```

These rows are identical in all fields — likely safe to remove.

No full duplicate rows

The screenshot shows a SQL query editor interface. At the top, there are tabs for 'Query' and 'Query History'. Below the tabs is a code editor containing the following SQL query:

```
1 v SELECT enrollment_id, learner_id, assigned_cohort, apply_date, status, COUNT(*) AS count
2 FROM learneropportunity_raw
3 GROUP BY enrollment_id, learner_id, assigned_cohort, apply_date, status
4 HAVING COUNT(*) > 1;
5
```

Below the code editor is a 'Data Output' tab, which is currently selected. Underneath it is a toolbar with various icons for file operations like copy, paste, and save. A table structure is displayed below the toolbar, showing columns: enrollment_id (text), learner_id (text), assigned_cohort (text), apply_date (timestamp without time zone), status (integer), and count (bigint). The table is currently empty.

4. Partially Duplicate Enrollment IDs

```
SELECT enrollment_id, COUNT(*) AS count
FROM learneropportunity_raw
GROUP BY enrollment_id
HAVING COUNT(*) > 1;
```

⚠ If `enrollment_id` is expected to be unique, this flags potential issues.

Yes the `enrollment_id` is a duplicate value column that contains values 20572

Query History

```

1
2 v SELECT enrollment_id, COUNT(*) AS count
3   FROM learneropportunity_raw
4   GROUP BY enrollment_id
5   HAVING COUNT(*) > 1;
6

```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No:

	enrollment_id	count
1	Learner#2444d3b7-3204-4b66-a1e2-72172db26b...	3
2	Learner#d7b8c0bd-8fc7-4a9c-a617-369e3fc6ed7d	2
3	Learner#6b9871b2-7830-4866-81ad-8674120eb2...	2
4	Learner#725951b3-90ed-4494-9bb7-573721cfce79	2
5	Learner#0b248ca5-aa1e-49c6-b447-4b85701d448a	4
6	Learner#b1aa02ae-c3cd-475a-8f5f-a0ebef98113e	2
7	Learner#66cda57a-fac3-4b04-bda9-6ede6a2cb674	2
8	Learner#a63382a5-b12c-4bdb-9138-f68096a4f232	4
9	Learner#8ea9690a-b4e4-4a18-a09b-8efe8fd64022	2
10	Learner#eb671993-c62b-4236-a251-124e32f38ed3	4

Total rows: 20572 Query complete 00:00:00.421

What to Do Next:

Issue Type	Meaning
NULLs	May indicate missing inputs, timing issues, or data pipeline errors
Full Duplicates	Usually safe to delete unless time-sensitive duplicates are allowed
Partial Duplicates	May indicate errors in key uniqueness or merging process

Step 4: Spot Outliers and Anomalies

- Use techniques like the Interquartile Range (IQR) to find unusual values that might distort analysis.

Research how outliers are detected using SQL and statistical methods.

Objective:

Use the **Interquartile Range (IQR)** method to identify **outliers** — values that are unusually high or low and may distort analysis or reflect errors.

Step-by-Step: Outlier Detection Using IQR in SQL

1. Get Q1 (25th percentile) and Q3 (75th percentile)

```
SELECT
    PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY status) AS q1,
    PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY status) AS q3
FROM learneropportunity_raw;
```

	q1	q3
1	double precision	double precision
1	1070	1070

2. Calculate IQR and Bounds (do this manually after running above)

$$\text{IQR} = Q3 - Q1$$

```

Q1 = 1070 , Q3 = 1070
IQR = 1070 - 1070
IQR = 0
Lower Bound = Q1 - 1.5 × IQR
              = 1070 - 1.5 * 0
              = 0
Upper Bound = Q3 + 1.5 × IQR
              = 1070 + 1.5 * 0
              = 0

```

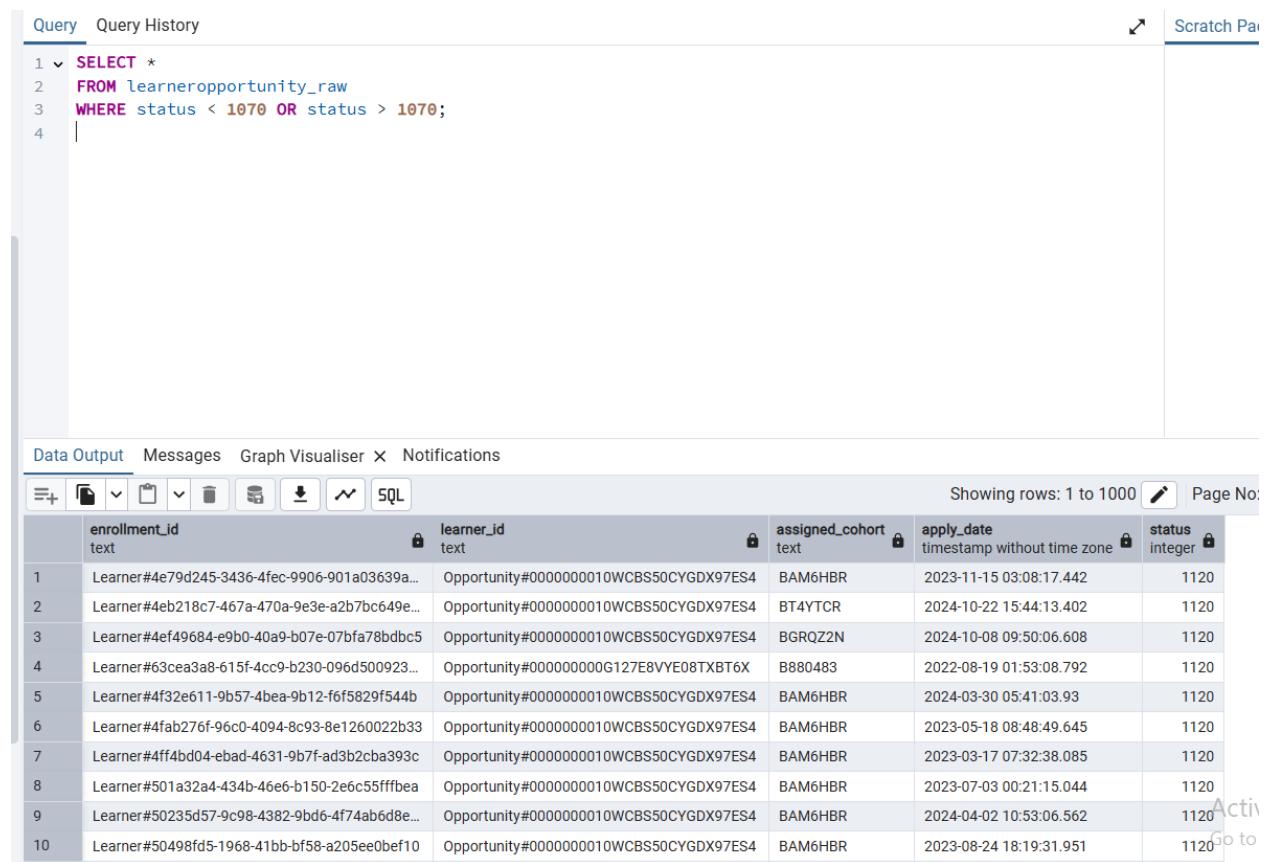
You'll substitute the calculated bounds into the next query.

Q 3. Find Outliers (values outside bounds)

```

SELECT *
FROM learneropportunity_raw
WHERE status < <lower_bound> OR status > <upper_bound>;

```



The screenshot shows a database interface with a query editor and a data output table.

Query Editor:

```

1 ✓ SELECT *
2   FROM learneropportunity_raw
3   WHERE status < 1070 OR status > 1070;
4

```

Data Output:

	enrollment_id	learner_id	assigned_cohort	apply_date	status
1	Learner#4e79d245-3436-4fec-9906-901a03639a...	Opportunity#0000000010WCBS50CYGDX97ES4	BAM6HBR	2023-11-15 03:08:17.442	1120
2	Learner#4eb218c7-467a-470a-9e3e-a2b7bc649e...	Opportunity#0000000010WCBS50CYGDX97ES4	BT4YTCR	2024-10-22 15:44:13.402	1120
3	Learner#4ef49684-e9b0-40a9-b07e-07bfa78bdbc5	Opportunity#0000000010WCBS50CYGDX97ES4	BGRQZ2N	2024-10-08 09:50:06.608	1120
4	Learner#63cea3a8-615f-4cc9-b230-096d500923...	Opportunity#000000000G127E8VYE08TXBT6X	B880483	2022-08-19 01:53:08.792	1120
5	Learner#4f32e611-9b57-4bea-9b12-f6f5829f544b	Opportunity#0000000010WCBS50CYGDX97ES4	BAM6HBR	2024-03-30 05:41:03.93	1120
6	Learner#4fab276f-96c0-4094-8c93-8e1260022b33	Opportunity#0000000010WCBS50CYGDX97ES4	BAM6HBR	2023-05-18 08:48:49.645	1120
7	Learner#4ff4bd04-ebad-4631-9b7f-ad3b2cba393c	Opportunity#0000000010WCBS50CYGDX97ES4	BAM6HBR	2023-03-17 07:32:38.085	1120
8	Learner#501a32a4-434b-46e6-b150-2e6c55ffffbea	Opportunity#0000000010WCBS50CYGDX97ES4	BAM6HBR	2023-07-03 00:21:15.044	1120
9	Learner#50235d57-9c98-4382-9bd6-4f74ab6d8e...	Opportunity#0000000010WCBS50CYGDX97ES4	BAM6HBR	2024-04-02 10:53:06.562	1120
10	Learner#50498fd5-1968-41bb-bf58-a205ee0bef10	Opportunity#0000000010WCBS50CYGDX97ES4	BAM6HBR	2023-08-24 18:19:31.951	1120

- All values are 1070, or nearly all — which is why:
 - $Q1 = Q3 = 1070$
 - $IQR = 0$

- So there are no outliers
 - This is **not an error**, just an indicator that:
 - The status field has **very low variance**
 - It's probably **not useful for outlier analysis**
-

Conclusion:

No outliers exist in the `status` column using IQR, because all values are clustered at 1070.

Dataset: Marketing Campaign Data All Accounts (2023-2024)(Detail1)

Performing the Exploratory Data Analysis

Step 1: Understand the Dataset

- View the first few rows to get an overview of the data.
- Check column names, data types, and the total number of records.

Research how to inspect a table's structure and data in PostgreSQL.

Step Objective:

Understand the structure of the dataset by inspecting:

- First few rows
 - Column names
 - Data types
 - Total number of records
-

Queries in PostgreSQL:

1. View First Few Rows:

```
SELECT *  
FROM marketing_campaign_raw  
LIMIT 5;
```

→ Gives a quick snapshot of sample records, formatting, and value types.

The screenshot shows a database interface with a query editor and a data output viewer.

Query Editor:

```

1
2 v SELECT *
3   FROM marketing_campaign_raw
4   LIMIT 5;
5

```

Data Output:

	ad_account_name	campaign_name	delivery_status	delivery_level	reach	outbound_clicks	landing_page_views	result_type
1	SLU	[null]	completed	campaign	102962	1815	1310	Website applications submitted
2	SLU	#B2: Digital Marketing Intern - May Ads: Website Leads Prospecting 18 to 35 years - Copy 3	completed	campaign	180175	3378	2152	Website applications submitted
3	SLU	#B2: Digital Marketing Intern - May Ads: Website Leads Prospecting 18 to 35 years - Copy 3	inactive	campaign	173118	3591	2767	Website applications submitted
4	SLU	#Brand Awareness: UGC Video - March - Copy	inactive	campaign	18355415	5431	1019	Reach
5	SLU	#Data Analyst Associate Internship	inactive	campaign	2448	111	62	Website leads

2. Check Column Names and Data Types:

```

SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = 'marketing_campaign_raw';

```

→ Lists each column along with its data type — such as text, integer, timestamp, etc.

The screenshot shows a database interface with a query editor and a data output viewer.

Query Editor:

```

1 v SELECT column_name, data_type
2   FROM information_schema.columns
3   WHERE table_name = 'marketing_campaign_raw';
4

```

Data Output:

	column_name	data_type
1	reporting_starts	date
2	landing_page_views	integer
3	results	integer
4	cost_per_result	numeric
5	amount_spent_aed	numeric
6	cpc_cost_per_link_click	numeric
7	reach	bigint
8	outbound_clicks	integer
9	campaign_name	text
10	delivery_status	text
11	delivery_level	text
12	ad_account_name	text
13	result_type	text

3. Get Total Number of Records:

```
SELECT COUNT(*)  
FROM marketing_campaign_raw;
```

→ Confirms how many rows (records) exist in the dataset

The screenshot shows a SQL query editor interface. At the top, there's a toolbar with tabs for 'Query' (which is selected), 'Query History', and other options. Below the toolbar, the SQL query is displayed:

```
1 ▾ SELECT COUNT(*)  
2   FROM marketing_campaign_raw  
3
```

Underneath the query, there's a 'Data Output' section which includes tabs for 'Messages', 'Graph Visualiser', and 'Notifications'. The main area shows the results of the query:

	count	bigint
1	148	

Step 2: Summarize the Data

- Generate basic statistics like mean, median, min, max, and standard deviation to understand the distribution.

Explore how to compute summary statistics in SQL.

Focused Numeric Columns from Your Sample:

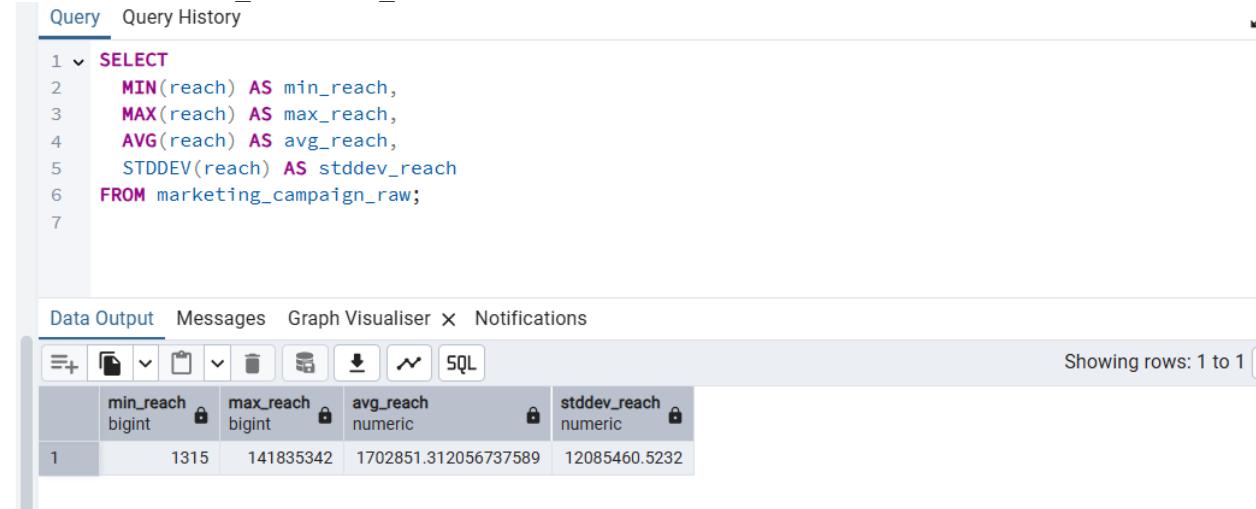
- Reach
- Outbound clicks
- Landing page views
- Results
- Cost per result
- Amount spent (AED)
- CPC (cost per link click)

PostgreSQL Queries for Descriptive Statistics

1. Basic Descriptive Statistics:

- Reach

```
SELECT
    MIN(reach) AS min_reach,
    MAX(reach) AS max_reach,
    AVG(reach) AS avg_reach,
    STDDEV(reach) AS stddev_reach
FROM marketing_campaign_raw;
```



The screenshot shows a PostgreSQL query editor interface. The query window contains the following SQL code:

```
1 ✓ SELECT
2   MIN(reach) AS min_reach,
3   MAX(reach) AS max_reach,
4   AVG(reach) AS avg_reach,
5   STDDEV(reach) AS stddev_reach
6 FROM marketing_campaign_raw;
7
```

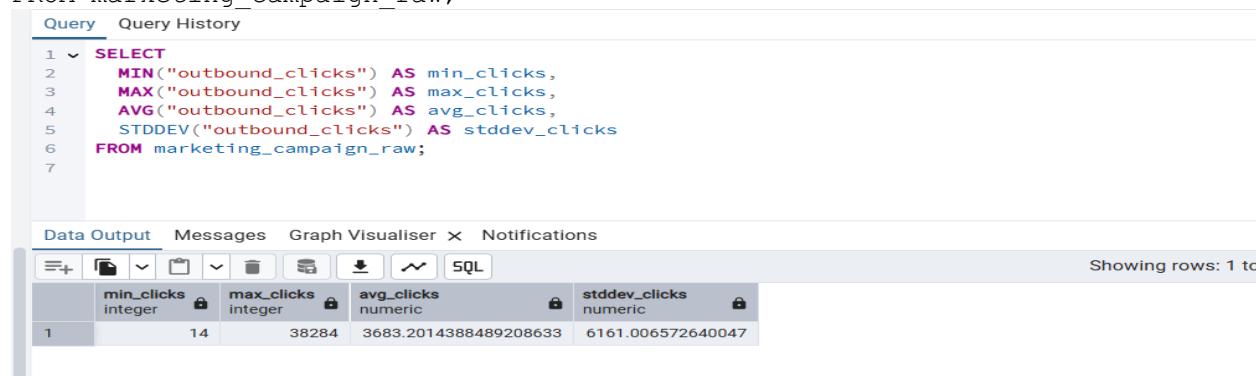
The results pane displays the output of the query:

	min_reach	max_reach	avg_reach	stddev_reach
1	1315	141835342	1702851.312056737589	12085460.5232

Below the results, the status bar indicates "Showing rows: 1 to 1".

- Outbound clicks

```
SELECT
    MIN("outbound_clicks") AS min_clicks,
    MAX("outbound_clicks") AS max_clicks,
    AVG("outbound_clicks") AS avg_clicks,
    STDDEV("outbound_clicks") AS stddev_clicks
FROM marketing_campaign_raw;
```



The screenshot shows a PostgreSQL query editor interface. The query window contains the following SQL code:

```
1 ✓ SELECT
2   MIN("outbound_clicks") AS min_clicks,
3   MAX("outbound_clicks") AS max_clicks,
4   AVG("outbound_clicks") AS avg_clicks,
5   STDDEV("outbound_clicks") AS stddev_clicks
6 FROM marketing_campaign_raw;
7
```

The results pane displays the output of the query:

	min_clicks	max_clicks	avg_clicks	stddev_clicks
1	14	38284	3683.2014388489208633	6161.006572640047

Below the results, the status bar indicates "Showing rows: 1 to 1".

- Landing page views

```
SELECT
    MIN("landing_page_views") AS min_lpv,
    MAX("landing_page_views") AS max_lpv,
    AVG("landing_page_views") AS avg_lpv,
    STDDEV("landing_page_views") AS stddev_lpv
FROM marketing_campaign_raw;
```

The screenshot shows a database query interface with the following details:

- Query History:** A list of previous queries, with the current one expanded.
- Data Output:** A table showing the results of the query.
- Toolbar:** Includes icons for file operations, copy, paste, and SQL.
- Message Bar:** Shows "Showing rows: 1 to 1" and a pencil icon.

	min_lpv integer	max_lpv integer	avg_lpv numeric	stddev_lpv numeric
1	1	21140	2113.9712230215827338	3634.395498132674

- Results

```
SELECT
    MIN(results) AS min_results,
    MAX(results) AS max_results,
    AVG(results) AS avg_results,
    STDDEV(results) AS stddev_results
FROM marketing_campaign_raw;
```

The screenshot shows a database query interface with the following details:

- Query History:** A list of previous queries, with the current one expanded.
- Data Output:** A table showing the results of the query.
- Toolbar:** Includes icons for file operations, copy, paste, and SQL.
- Message Bar:** Shows "Showing r" (likely a truncated value).

	min_results integer	max_results integer	avg_results numeric	stddev_results numeric
1	1	182515486	2570647.577464788732	19420171.5844

- Cost per result

```
SELECT
    MIN("cost_per_result") AS min_cost_result,
    MAX("cost_per_result") AS max_cost_result,
    AVG("cost_per_result") AS avg_cost_result,
    STDDEV("cost_per_result") AS stddev_cost_result
FROM marketing_campaign_raw;
```

Query Query History

```
1 ▾ SELECT
2   MIN("cost_per_result") AS min_cost_result,
3   MAX("cost_per_result") AS max_cost_result,
4   AVG("cost_per_result") AS avg_cost_result,
5   STDDEV("cost_per_result") AS stddev_cost_result
6 FROM marketing_campaign_raw;
7
```

Data Output Messages Graph Visualiser X Notifications

Showing rows: 1 to 1

	min_cost_result numeric	max_cost_result numeric	avg_cost_result numeric	stddev_cost_result numeric
1	0.003665	47.08949	4.1639695886524823	4.9823117795664137

- Amount spent (AED)

```
SELECT
    MIN("amount_spent_aed") AS min_spent,
    MAX("amount_spent_aed") AS max_spent,
    AVG("amount_spent_aed") AS avg_spent,
    STDDEV("amount_spent_aed") AS stddev_spent
FROM marketing_campaign_raw;
```

Query Query History

```
1 ▾ SELECT
2   MIN("amount_spent_aed") AS min_spent,
3   MAX("amount_spent_aed") AS max_spent,
4   AVG("amount_spent_aed") AS avg_spent,
5   STDDEV("amount_spent_aed") AS stddev_spent
6 FROM marketing_campaign_raw;
7
```

Data Output Messages Graph Visualiser X Notifications

Showing rows: 1 to 1

	min_spent numeric	max_spent numeric	avg_spent numeric	stddev_spent numeric
1	0.99	25531.47	2400.2125102605633803	3937.369148029759

- CPC (cost per link click)

```

SELECT
    MIN("cpc_cost_per_link_click") AS min_cpc,
    MAX("cpc_cost_per_link_click") AS max_cpc,
    AVG("cpc_cost_per_link_click") AS avg_cpc,
    STDDEV("cpc_cost_per_link_click") AS stddev_cpc

FROM marketing_campaign_raw;

```

The screenshot shows a SQL query editor with the following details:

- Query History:** A dropdown menu showing the executed query.
- Data Output:** The results of the query are displayed in a table.
- Table Headers:** min_cpc, max_cpc, avg_cpc, stddev_cpc.
- Table Data:**

	min_cpc numeric	max_cpc numeric	avg_cpc numeric	stddev_cpc numeric
1	0.048919	5.597563	1.0566550642857143	0.93042100449303613622
- Notifications:** None.
- Showing rows:** 1 to 1

2. □ Median for Each Numeric Column (requires PERCENTILE_CONT)

Example for one column — apply similarly for others:

1-Median of reach:

```

SELECT
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY reach) AS median_reach
FROM marketing_campaign_raw;

```

The screenshot shows a SQL query editor with the following details:

- Query History:** A dropdown menu showing the executed query.
- Data Output:** The results of the query are displayed in a table.
- Table Headers:** median_reach.
- Table Data:**

	median_reach double precision
1	148357
- Notifications:** None.
- Showing rows:** 1 to 1

2-Median of outbound_clicks:

```
SELECT
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY outbound_clicks) AS
median_outbound_clicks
FROM marketing_campaign_raw;
```

The screenshot shows a database query interface with the following details:

- Query History:** A dropdown menu showing the executed SQL query.
- Data Output:** The results of the query are displayed in a table.
- Table Headers:** The table has one header row: "median_outbound_clicks double precision".
- Data Rows:** One data row is present, showing the value 1604.
- Toolbar:** Includes icons for file operations (New, Open, Save, Print, Copy, Paste, Find, Refresh, Help), a download icon, and a SQL button.
- Filter:** Shows "Showing rows: 1 to 1" with a pencil icon.

3-Median of landing_page_views:

```
SELECT
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY landing_page_views) AS
median_landing_page_views
FROM marketing_campaign_raw;
```

The screenshot shows a database query interface with the following details:

- Query History:** A dropdown menu showing the executed SQL query.
- Data Output:** The results of the query are displayed in a table.
- Table Headers:** The table has one header row: "median_landing_page_views double precision".
- Data Rows:** One data row is present, showing the value 868.
- Toolbar:** Includes icons for file operations (New, Open, Save, Print, Copy, Paste, Find, Refresh, Help), a download icon, and a SQL button.
- Filter:** Shows "Showing rows: 1 to 1" with a pencil icon.

4-Median of results:

```
SELECT
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY results) AS median_results
```

```
FROM marketing_campaign_raw;
```

Query Query History

```
1 ▼ SELECT
2   PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY results) AS median_results
3   FROM marketing_campaign_raw;
4
```

Data Output Messages Graph Visualiser × Notifications

	median_results	double precision
1		384

Showing rows: 1 to 1

5-Median of cost_per_result:

```
SELECT
  PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY cost_per_result) AS
median_cost_per_result
FROM marketing_campaign_raw;
```

Query Query History

```
1 ▼ SELECT
2   PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY cost_per_result) AS median_cost_per_result
3   FROM marketing_campaign_raw;
4
```

Data Output Messages Graph Visualiser × Notifications

	median_cost_per_result	double precision
1		2.920683

Showing rows: 1 to 1

6-Median of amount_spent_aed:

```
SELECT
  PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY amount_spent_aed) AS
median_amount_spent_aed
FROM marketing_campaign_raw;
```

The screenshot shows a SQL query editor interface. At the top, there's a toolbar with tabs for 'Query' (which is selected) and 'Query History'. Below the toolbar is the SQL code:

```

1 v SELECT
2   PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY amount_spent_aed) AS median_amount_spent_aed
3   FROM marketing_campaign_raw;
4

```

Below the code is a 'Data Output' tab, followed by 'Messages', 'Graph Visualiser', and 'Notifications' tabs. The 'Data Output' tab is active, showing a table with one row:

	median_amount_spent_aed	double precision
1	1253.225	

At the bottom right of the data output area, it says 'Showing rows: 1 to 1'.

7-Median of cpc_cost_per_link_click:

```

SELECT
  PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY cpc_cost_per_link_click) AS
median_cpc
FROM marketing_campaign_raw;

```

The screenshot shows a SQL query editor interface. At the top, there's a toolbar with tabs for 'Query' (selected) and 'Query History'. Below the toolbar is the SQL code:

```

1 v SELECT
2   PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY cpc_cost_per_link_click) AS median_cpc
3   FROM marketing_campaign_raw;
4

```

Below the code is a 'Data Output' tab, followed by 'Messages', 'Graph Visualiser', and 'Notifications' tabs. The 'Data Output' tab is active, showing a table with one row:

	median_cpc	double precision
1	0.7637075	

At the bottom right of the data output area, it says 'Showing rows: 1 to 1'.

What These Stats Reveal:

Metric	Insight Example
MAX (Reach)	Identifies your most-viewed campaign
AVG (Outbound clicks)	Shows how effective your campaigns are at driving traffic
STDDEV (Amount Spent)	Large spread may indicate outlier spending in some campaigns

Metric	Insight Example
MEDIAN (Results)	More stable than average, especially if there are outliers
MIN (CPC)	Useful for identifying most cost-efficient ad

Step 3: Identify Missing and Duplicate Values

- Detect missing data to determine if any cleaning is needed.
- Identify duplicate records that may need removal.

Look up methods to check for missing values and duplicates in PostgreSQL.

Dataset: marketing_campaign_raw

Objective:

- Detect missing (NULL) values in each column
- Identify duplicate records that might need removal or review

1. Check for Missing (NULL) Values in Each Column

```
SELECT
    COUNT(*) AS total_rows,
    COUNT(ad_account_name) AS filled_ad_account_name,
    COUNT(campaign_name) AS filled_campaign_name,
    COUNT(delivery_status) AS filled_delivery_status,
    COUNT(delivery_level) AS filled_delivery_level,
    COUNT(reach) AS filled_reach,
    COUNT(outbound_clicks) AS filled_outbound_clicks,
    COUNT(landing_page_views) AS filled_landing_page_views,
    COUNT(result_type) AS filled_result_type,
    COUNT(results) AS filled_results,
    COUNT(cost_per_result) AS filled_cost_per_result,
    COUNT(amount_spent_aed) AS filled_amount_spent_aed,
    COUNT(cpc_cost_per_link_click) AS filled_cpc,
    COUNT(reporting_starts) AS filled_reporting_starts
FROM marketing_campaign_raw;
```

Query Query History Scratchpad

```

1 SELECT
2   COUNT(*) AS total_rows,
3   COUNT(ad_account_name) AS filled_ad_account_name,
4   COUNT(campaign_name) AS filled_campaign_name,
5   COUNT(delivery_status) AS filled_delivery_status,
6   COUNT(delivery_level) AS filled_delivery_level,
7   COUNT(reach) AS filled_reach,
8   COUNT(outbound_clicks) AS filled_outbound_clicks,
9   COUNT(landing_page_views) AS filled_landing_page_views,
10  COUNT(result_type) AS filled_result_type,
11  COUNT(results) AS filled_results,
12  COUNT(cost_per_result) AS filled_cost_per_result,
13  COUNT(amount_spent_aed) AS filled_amount_spent_aed,
14  COUNT(cpc_cost_per_link_click) AS filled_cpc,
15  COUNT(reporting_starts) AS filled_reporting_starts
16 FROM marketing_campaign_raw;

```

Data Output Messages Graph Visualiser × Notifications

	total_rows bigint	filled_ad_account_name bigint	filled_campaign_name bigint	filled_delivery_status bigint	filled_delivery_level bigint	filled_reach bigint	filled_outbound_clicks bigint
1	148	141	139	141	141	141	139

filled_results bigint	filled_cost_per_result bigint	filled_amount_spent_aed bigint	filled_cpc bigint	filled_reporting_starts bigint
142	141	142	140	141

🔍 Interpretation:

- Compare each filled count with `total_rows`
- If any column has significantly fewer values, it's sparsely populated → candidate for cleaning/imputation.

2. Check for Duplicate Entire Rows

```

SELECT
  ad_account_name, campaign_name, delivery_status, delivery_level,
  reach, outbound_clicks, landing_page_views, result_type,
  results, cost_per_result, amount_spent_aed, cpc_cost_per_link_click,
  reporting_starts,
  COUNT(*) AS duplicates
FROM marketing_campaign_raw
GROUP BY
  ad_account_name, campaign_name, delivery_status, delivery_level,
  reach, outbound_clicks, landing_page_views, result_type,
  results, cost_per_result, amount_spent_aed, cpc_cost_per_link_click,
  reporting_starts
HAVING COUNT(*) > 1;

```

Query History

```

1 ✓ SELECT
2   ad_account_name, campaign_name, delivery_status, delivery_level,
3   reach, outbound_clicks, landing_page_views, result_type,
4   results, cost_per_result, amount_spent_aed, cpc_cost_per_link_click, reporting_starts,
5   COUNT(*) AS duplicates
6   FROM marketing_campaign_raw
7   GROUP BY
8     ad_account_name, campaign_name, delivery_status, delivery_level,
9     reach, outbound_clicks, landing_page_views, result_type,
10    results, cost_per_result, amount_spent_aed, cpc_cost_per_link_click, reporting_starts
11   HAVING COUNT(*) > 1;
12

```

Data Output Messages Graph Visualiser X Notifications

Showing rows: 1 to 1

	ad_account_name	campaign_name	delivery_status	delivery_level	reach	outbound_clicks	landing_page_views	result_type
1	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]

	results	cost_per_result	amount_spent_aed	cpc_cost_per_link_click	reporting_starts	duplicates
1	[null]	[null]	[null]	[null]	[null]	6

Interpretation:

- Shows exact duplicate rows
- If these exist, confirm whether they're valid or redundant before removing.

Count of All Duplicates:

```

SELECT COUNT(*) - COUNT(DISTINCT ROW(
  ad_account_name, campaign_name, delivery_status, delivery_level,
  reach, outbound_clicks, landing_page_views, result_type,
  results, cost_per_result, amount_spent_aed, cpc_cost_per_link_click,
  reporting_starts
)) AS total_duplicate_rows
FROM marketing_campaign_raw;

```

Total Duplicates are 6.

Step 4: Spot Outliers and Anomalies

- Use techniques like the Interquartile Range (IQR) to find unusual values that might distort analysis.

Research how outliers are detected using SQL and statistical methods.

reach — Detect Outliers

```
SELECT
    PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY reach) AS q1,
    PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY reach) AS q3
FROM marketing_campaign_raw;
```

The screenshot shows a SQL query editor interface. At the top, there are tabs for 'Query' (which is selected) and 'Query History'. Below the tabs, the query code is displayed:

```
1 SELECT
2     PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY reach) AS q1,
3     PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY reach) AS q3
4 FROM marketing_campaign_raw;
5
```

Below the code, the results are shown in a table:

	q1	q3
1	double precision	double precision
	44420	422292

The 'Data Output' tab is selected at the top of the interface.

Q1 = 44420 , Q3 = 422292

```
SELECT *
FROM marketing_campaign_raw
WHERE reach < (44420 - 1.5 * (422292 - 44420))
    OR reach > (422292 + 1.5 * (422292 - 44420));
```

Query Query History Scratch Pad

```
1 SELECT *
2 FROM marketing_campaign_raw
3 WHERE reach < (44420 - 1.5 * (422292 - 44420))
4     OR reach > (422292 + 1.5 * (422292 - 44420));
5
```

Data Output Messages Graph Visualiser Notifications

Showing rows: 1 to 26 Page No: 1 of 1

	ad_account_name	campaign_name	delivery_status	delivery_level	reach	outbound_clicks	landing_page_id
10	Brand Awareness	DEC: FB only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	completed	campaign	1881735	9695	
11	Brand Awareness	DEC: IG only_AWARENESS: Excelerate_Carousel_Ad (with Exclusion) * Copy	completed	campaign	1494425	1041	
12	Brand Awareness	DEC: IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	completed	campaign	1707489	2528	
13	Brand Awareness	FB only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	completed	campaign	3157807	3228	
14	SLU	Feb INTERNSHIP Virtual Internship	active	campaign	1403023	18976	
15	Brand Awareness	Feb 2025: IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	not_delivering	campaign	1020338	619	
16	Brand Awareness	IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion) - Copy	completed	campaign	1932709	2890	
17	Brand Awareness	Jan 2025 Masterclass SBD Dust Challenge	inactive	campaign	1679722	14090	
18	Brand Awareness	JAN 2025: IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	inactive	campaign	2737467	3177	
19	SLU	JAN. INTERNSHIP: Virtual Internship	inactive	campaign	1580670	27483	
20	Brand Awareness	JAN: FB only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	completed	campaign	2694803	4301	
21	SLU	March Brand Awareness	archived	campaign	141835342	32289	
22	SLU	May Awareness (Reach) campaign	completed	campaign	17331595	13469	
23	Brand Awareness	NOV: IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion) * Copy	completed	campaign	1603743	2334	
24	Brand Awareness	NOV: IG only_AWARENESS: Shrishi_video_Ad (with Exclusion)	completed	campaign	1961895	2519	
25	Brand Awareness	OCT: FB only_AWARENESS: Excelerate_Video_Ad (with Exclusion) - Copy	completed	campaign	3130793	3708	
26	Brand Awareness	OCT: IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	completed	campaign	1637784	2386	

2. outbound_clicks — Detect Outliers

```
SELECT
    PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY outbound_clicks) AS q1,
    PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY outbound_clicks) AS q3
FROM marketing_campaign_raw;
```

Query Query History

```
1 SELECT
2     PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY outbound_clicks) AS q1,
3     PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY outbound_clicks) AS q3
4 FROM marketing_campaign_raw;
5
```

Data Output Messages Graph Visualiser Notifications

Showing rows: 1 to 1

	q1	q3
1	531	3202.5

Q1 = 531 , Q3 = 3202.5

```
SELECT *
FROM marketing_campaign_raw
WHERE outbound_clicks < (531 - 1.5 * (3202.5 - 531))
    OR outbound_clicks > (3202.5 + 1.5 * (3202.5 - 531));
```

The screenshot shows a database interface with a query history tab and a scratch pad tab. The query history contains the following SQL code:

```

1 v SELECT *
2   FROM marketing_campaign_raw
3  WHERE outbound_clicks < (531 - 1.5 * (3202.5-531))
4    OR outbound_clicks > (3202.5 + 1.5 * (3202.5-531));
5

```

The data output tab displays a table with the following columns: ad_account_name, campaign_name, delivery_status, delivery_level, reach bigint, outbound_clicks integer, and landing_page integer. The table contains 19 rows of data.

	ad_account_name	campaign_name	delivery_status	delivery_level	reach bigint	outbound_clicks	landing_page
1	SLU	A1: Outreach Consultant Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	inactive	campaign	1136229	12798	
2	SLU	A2: Digital Strategy Associate Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	inactive	campaign	682351	9510	
3	SLU	A3: Data Analyst Associate Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	inactive	campaign	1077778	21464	
4	SLU	A4: Project Management Associate Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	inactive	campaign	934611	13060	
5	SLU	A5: Business Strategy Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	inactive	campaign	999159	13337	
6	SLU	B3: Data Analyst Associate Intern - May Ads: Website Leads Prospecting 18 to 35 years - Copy 3	completed	campaign	417944	10165	
7	SLU	B4: Project Management Associate Intern - May Ads: Website Leads Prospecting 18 to 35 years - Copy 3	completed	campaign	411592	9159	
8	SLU	Career Essentials - March Ads: Website Leads Prospecting 18 to 35 years	inactive	campaign	1084993	11508	
9	SLU	CPR - March Ads: Website Leads Prospecting 18 to 35 years - Copy	inactive	campaign	4434511	38284	
10	SLU	Dec. INTERNSHIP: Virtual Internship	completed	campaign	1010394	16961	
11	Brand Awareness	DEC: FB only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	completed	campaign	1881735	9695	
12	SLU	Feb INTERNSHIP Virtual Internship	active	campaign	1403023	18976	
13	SLU	INTERNSHIP: Virtual Internship	completed	campaign	982208	18267	
14	Brand Awareness	Jan 2025 Masterclass SBD Dust Challenge	inactive	campaign	1679722	14090	
15	SLU	JAN. INTERNSHIP: Virtual Internship	inactive	campaign	1580670	27483	
16	SLU	March Brand Awareness	archived	campaign	141835342	32289	
17	SLU	May Awareness (Reach) campaign	completed	campaign	17331595	13469	
18	SLU	Oct. INTERNSHIP: Virtual Internship	completed	campaign	141835342	32289	
19	SLU	Oct. INTERNSHIP: Virtual Internship	completed	campaign	141835342	32289	

Total rows: 19 Query complete 00:00:00.148 CRLF Ln 4, Col 49

◆ 3. landing_page_views — Detect Outliers

```

SELECT
  PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY landing_page_views) AS q1,
  PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY landing_page_views) AS q3
FROM marketing_campaign_raw;

```

The screenshot shows a database interface with a query history tab and a scratch pad tab. The query history contains the following SQL code:

```

1 v SELECT
2   PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY landing_page_views) AS q1,
3   PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY landing_page_views) AS q3
4   FROM marketing_campaign_raw;
5

```

The data output tab displays a table with the following columns: q1 and q3. The table contains 1 row of data.

	q1	q3
1	260.5	1788.5

Q1 = 260.5 , Q3 = 1788.5

```
SELECT *
FROM marketing_campaign_raw
WHERE landing_page_views < (260.6 - 1.5 * (1788.5 - 260.5))
OR landing_page_views > (1788.5 + 1.5 * (1788.5 - 260.5));
```

The screenshot shows a SQL query being run in a tool with tabs for 'Query' (selected), 'Query History', 'Scratch Pad', 'Data Output', 'Messages', 'Graph Visualiser', and 'Notifications'. The 'Data Output' tab is active, displaying the results of the query.

Query:

```
1 v SELECT *
2   FROM marketing_campaign_raw
3 WHERE landing_page_views < (260.6 - 1.5 * (1788.5 - 260.5))
4   OR landing_page_views > (1788.5 + 1.5 * (1788.5 - 260.5));
5
```

Data Output:

	ad_account_name	campaign_name	delivery_status	delivery_level	reach	outbound_clicks	landing_page_views
1	SLU	A1: Outreach Consultant Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	inactive	campaign	1136229	12798	
2	SLU	A2: Digital Strategy Associate Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	inactive	campaign	682351	9510	
3	SLU	A3: Data Analyst Associate Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	inactive	campaign	1077778	21464	
4	SLU	A4: Project Management Associate Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	inactive	campaign	934611	13060	
5	SLU	A5: Business Strategy Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	inactive	campaign	999159	13337	
6	SLU	B3: Data Analyst Associate Intern - May Ads: Website Leads Prospecting 18 to 35 years - Copy 3	completed	campaign	417944	10165	
7	SLU	B4: Project Management Associate Intern - May Ads: Website Leads Prospecting 18 to 35 years - Copy 3	completed	campaign	411592	9159	
8	SLU	Career Essentials - March Ads: Website Leads Prospecting 18 to 35 years	inactive	campaign	1084993	11508	
9	SLU	CPR - March Ads: Website Leads Prospecting 18 to 35 years - Copy	inactive	campaign	4434511	38284	
10	SLU	Dec. INTERNSHIP: Virtual Internship	completed	campaign	1010394	16961	
11	SLU	Feb INTERNSHIP Virtual Internship	active	campaign	1403023	18976	
12	SLU	INTERNSHIP: Virtual Internship	completed	campaign	982208	18267	
13	Brand Awareness	Jan 2025 Masterclass SBD Dust Challenge	inactive	campaign	1679722	14090	
14	SLU	JAN. INTERNSHIP: Virtual Internship	inactive	campaign	1580670	27483	
15	SLU	March Brand Awareness	archived	campaign	141835342	32289	
16	SLU	Oct. INTERNSHIP: Virtual Internship	completed	campaign	910690	16262	
17	SLU	[null]	completed	campaign	911773	15679	

Total rows: 17 Query complete 00:00:00.134 CRLF Ln 5, Col 1

◆ 4. results — Detect Outliers

```
SELECT
    PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY results) AS q1,
    PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY results) AS q3
FROM marketing_campaign_raw;
```

The screenshot shows a SQL query being run in a tool with tabs for 'Query' (selected), 'Query History', 'Scratch Pad', 'Data Output', 'Messages', 'Graph Visualiser', and 'Notifications'. The 'Data Output' tab is active, displaying the results of the query.

Query:

```
1 v SELECT
2     PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY results) AS q1,
3     PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY results) AS q3
4   FROM marketing_campaign_raw;
5
```

Data Output:

	q1	q3
1	161.25	2022.75

Showing rows: 1 to 1

Q1 = 161.25 , Q2 = 2022.75

```
SELECT *
FROM marketing_campaign_raw
WHERE results < (161.25 - 1.5 * (2022.75 - 161.25))
OR results > (2022.75 + 1.5 * (2022.75 - 161.25));
```

Query History

```
1 SELECT *
2 FROM marketing_campaign_raw
3 WHERE results < (161.25 - 1.5 * (2022.75 - 161.25))
4 OR results > (2022.75 + 1.5 * (2022.75 - 161.25));
5
```

Data Output

	ad_account_name	campaign_name	delivery_status	delivery_level	reach	outbound_clicks	landing_page_views
1	SLU	#Brand Awareness: UGC Video - March - Copy	inactive	campaign	18355415	5431	1019
2	SLU	A3: Data Analyst Associate Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	inactive	campaign	1077778	21464	16914
3	SLU	Awareness: Do you want to stand out? Reel and story Video Views	completed	campaign	1261904	3644	808
4	Brand Awareness	AWARENESS: Excelerate_Video_Ad (with Exclusion)	completed	campaign	1560112	1850	1850
5	Brand Awareness	DEC: FB only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	completed	campaign	1881735	9695	3274
6	Brand Awareness	DEC: IG only_AWARENESS: Excelerate_Carousel_Ad (with Exclusion) â€“ Copy	completed	campaign	1494425	1041	200
7	Brand Awareness	DEC: IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	completed	campaign	1707489	2528	1083
8	SLU	Excelerate Brand Awareness (March Campaign) - CBO Activated - Copy	inactive	campaign	774703	203	30
9	Brand Awareness	FB only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	completed	campaign	3157807	3228	438
10	SLU	Feb INTERNSHIP Virtual Internship	active	campaign	1403023	18976	13835
11	Brand Awareness	Feb 2025: FB only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	not_delivering	campaign	719805	2242	2242
12	Brand Awareness	Feb 2025: IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	not_delivering	campaign	1020338	619	619
13	Brand Awareness	IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion) - Copy	completed	campaign	1932709	2890	804
14	Brand Awareness	JAN 2025: IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	inactive	campaign	2737467	3177	1281
15	SLU	JAN: INTERNISHIP: Virtual Internship	inactive	campaign	1580670	27483	2114
16	Brand Awareness	JAN: FB only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	completed	campaign	2694803	4301	1249
17	SLU	March Brand Awareness	archived	campaign	141835342	32289	5584

Total rows: 25 Query complete 00:00:00.153

◆ 5. cost_per_result — Detect Outliers

```
SELECT
PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY cost_per_result) AS q1,
PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY cost_per_result) AS q3
FROM marketing_campaign_raw;
```

Query History

```
1 SELECT
2 PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY cost_per_result) AS q1,
3 PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY cost_per_result) AS q3
4 FROM marketing_campaign_raw;
5
```

Data Output

	q1	q3
1	1.2831	5.5095

Q1 = 1.2831 , Q3 = 5.5095

```
SELECT *
FROM marketing_campaign_raw
WHERE cost_per_result < (1.2831 - 1.5 * (5.5095 - 1.2831))
    OR cost_per_result > (5.5095 + 1.5 * (5.5095 - 1.2831));
```

The screenshot shows a database interface with a query editor and a data output viewer.

Query History:

```
1 v SELECT *
2   FROM marketing_campaign_raw
3 WHERE cost_per_result < (1.2831 - 1.5 * (5.5095 - 1.2831))
4   OR cost_per_result > (5.5095 + 1.5 * (5.5095 - 1.2831));
5
```

Data Output:

Showing rows: 1 to 5 / Page No: 1 of 1

	ad_account_name	campaign_name	delivery_status	delivery_level	reach	outbound_clicks	landing_page_views	result_text
1	Brand Awareness	Jan 2025 Masterclass SBD Dust Challenge	inactive	campaign	1679722	14090	6700	Websi
2	SLU	Nov Experience Gps Beyond Boundaries	completed	campaign	29914	1088	887	Websi
3	SLU	Oct COMPETITION: Youth Culture Insight challenge	completed	campaign	45204	522	328	Websi
4	SLU	PROSPECTING: Xperience Design Hackathon - Website LEADS/SIGN UPS	inactive	campaign	36143	181	60	Websi
5	SLU	Storysprint Empower Impactful Narratives 1	inactive	campaign	72339	520	309	Websi

◆ 6. amount_spent_aed — Detect Outliers

```
SELECT
    PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY amount_spent_aed) AS q1,
    PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY amount_spent_aed) AS q3
FROM marketing_campaign_raw;
```

The screenshot shows a database interface with a query editor and a data output viewer.

Query History:

```
1 v SELECT
2   PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY amount_spent_aed) AS q1,
3   PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY amount_spent_aed) AS q3
4   FROM marketing_campaign_raw;
5
```

Data Output:

Showing rows: 1 to 1 /

	q1	q3
1	480.71500000000003	2206.3424999999997

Q1 = 480.71500000000003, Q3 = 2206.3424999999997

```
SELECT *
FROM marketing_campaign_raw
WHERE amount_spent_aed < (480.71500000000003 - 1.5 * (2206.3424999999997 -
480.71500000000003))
    OR amount_spent_aed > (2206.3424999999997 + 1.5 * (2206.3424999999997 -
480.71500000000003));
```

Query Query History Scratch Pad

```

1 SELECT *
2 FROM marketing_campaign_raw
3 WHERE amount_spent_aed < (480.7150000000003 - 1.5 * (2206.342499999997 - 480.7150000000003))
4 OR amount_spent_aed > (2206.342499999997 + 1.5 * (2206.342499999997 - 480.7150000000003));
5
6

```

Data Output Messages Graph Visualiser Notifications

Showing rows: 1 to 10 Page No: 1 of 1

	ad_account_name	campaign_name	delivery_status	delivery_level	reach	outbound_clicks	landing_page_views	result_type
1	SLU	Career Essentials - March Ads: Website Leads Prospecting 18 to 35 years	inactive	campaign	1084993	11508	4826	Website le
2	SLU	CPR - March Ads: Website Leads Prospecting 18 to 35 years - Copy	inactive	campaign	4434511	38284	16060	Website le
3	SLU	Dec. INTERNSHIP: Virtual Internship	completed	campaign	1010394	16961	12387	Website ap
4	SLU	Feb INTERNSHIP Virtual Internship	active	campaign	1403023	18976	13835	Website ap
5	SLU	INTERNSHIP: Virtual Internship	completed	campaign	982208	18267	13468	Website ap
6	Brand Awareness	Jan 2025 Masterclass SBD Dust Challenge	inactive	campaign	1679722	14090	6700	Website ap
7	SLU	JAN. INTERNSHIP: Virtual Internship	inactive	campaign	1580670	27483	21140	Website ap
8	SLU	March Brand Awareness	archived	campaign	141835342	32289	5584	Reach
9	SLU	Oct. INTERNSHIP: Virtual Internship	completed	campaign	910690	16262	11451	Website ap
10	SLU	[null]	completed	campaign	911773	15679	10904	Website ap

◆ 7. cpc_cost_per_link_click — Detect Outliers

```

SELECT
    PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY cpc_cost_per_link_click) AS q1,
    PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY cpc_cost_per_link_click) AS q3
FROM marketing_campaign_raw;

```

Query Query History

```

1 SELECT
2     PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY cpc_cost_per_link_click) AS q1,
3     PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY cpc_cost_per_link_click) AS q3
4 FROM marketing_campaign_raw;
5

```

Data Output Messages Graph Visualiser Notifications

Showing rows: 1 to 1

	q1	q3
1	0.4287435000000006	1.3596395000000001

Q1 = 0.4287435000000006 , Q3 = 1.3596395000000001

```

SELECT *
FROM marketing_campaign_raw
WHERE cpc_cost_per_link_click < (0.4287435000000006 - 1.5 *
(1.3596395000000001 - 0.4287435000000006))

```

```
OR cpc_cost_per_link_click > (1.3596395000000001 + 1.5 *
(1.3596395000000001 - 0.4287435000000006));
```

The screenshot shows a database interface with a toolbar at the top containing various icons for file operations, search, and navigation. Below the toolbar, a tab bar includes 'Query History' (selected), 'Scratches', 'Data Output', 'Messages', 'Graph Visualiser', and 'Notifications'. The main area displays a SQL query and its execution results.

SQL Query:

```

1  SELECT *
2  FROM marketing_campaign_raw
3  WHERE cpc_cost_per_link_click < (0.4287435000000006 - 1.5 * (1.3596395000000001 - 0.4287435000000006))
4      OR cpc_cost_per_link_click > (1.3596395000000001 + 1.5 * (1.3596395000000001 - 0.4287435000000006));
5

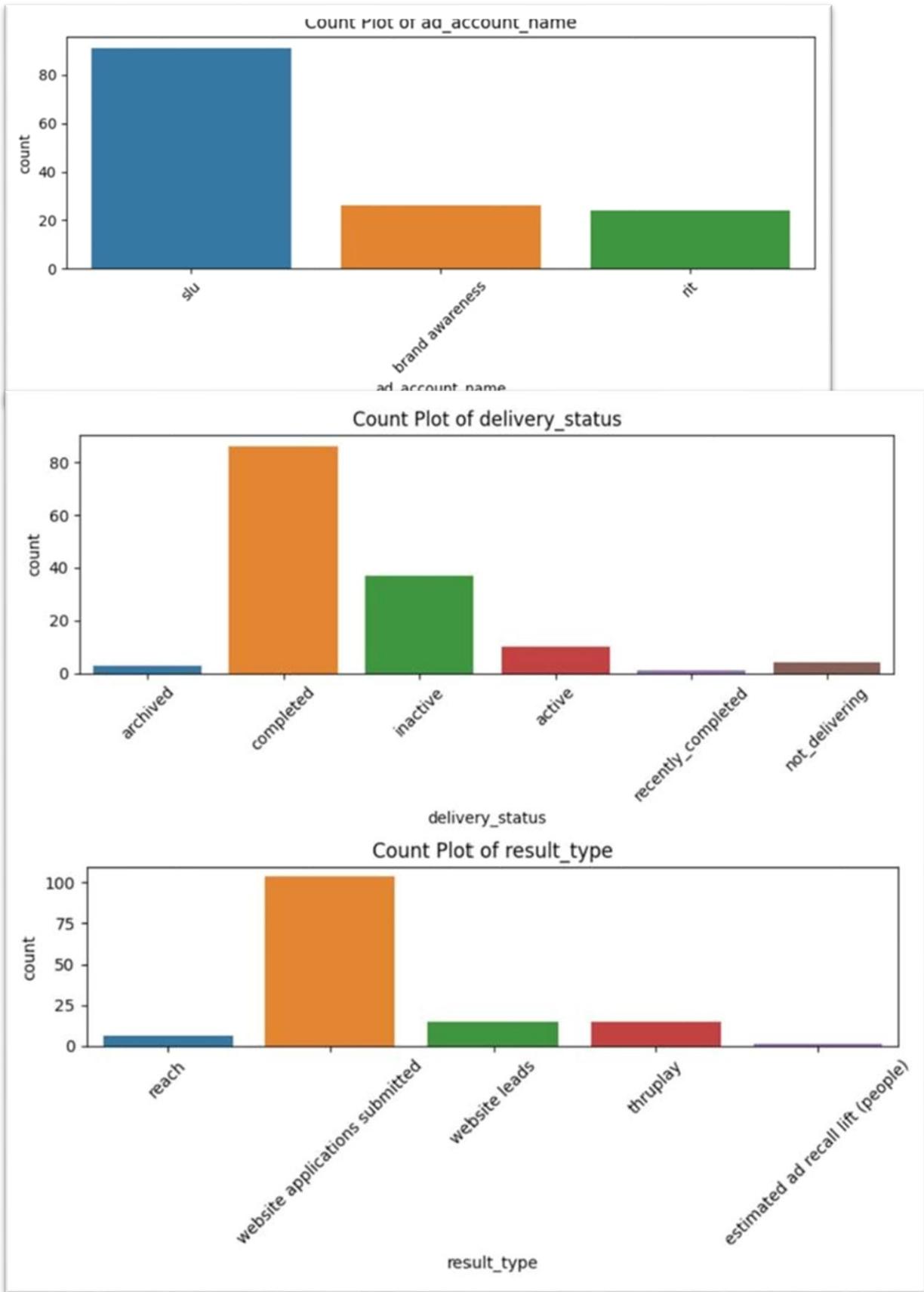
```

Data Output:

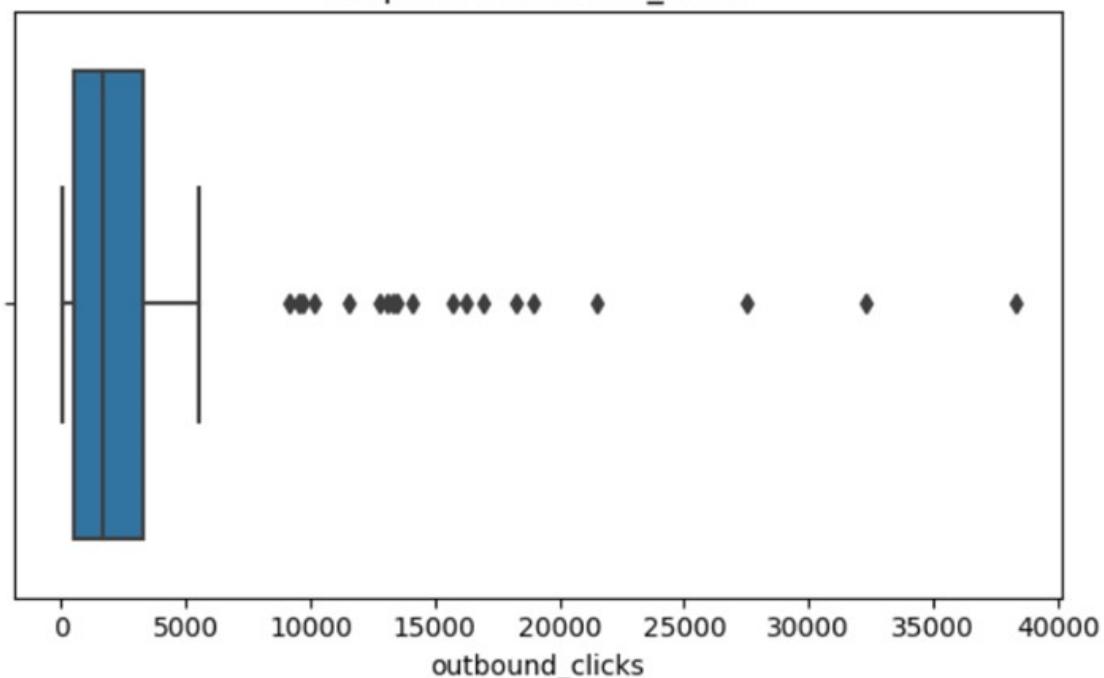
	ad_account_name	campaign_name	delivery_status	delivery_level	reach	outbound_clicks
1	SLU	COMPETITION: Youth Culture Insight challenge " Copy	completed	campaign	76502	
2	Brand Awareness	Dec Power Skill Course Secrets to Operational Excellence	completed	campaign	32403	
3	SLU	Feb Power Course Digital Wellness & Mindful Tech,	active	campaign	59934	
4	SLU	Jan Power Course Digital Wellness & Mindful Tech - Copy	completed	campaign	96595	
5	Brand Awareness	Jan Power Skill Course Pepagora Secrets to Operational Excellence - Copy	completed	campaign	37197	
6	SLU	Nov Experience Gps Beyond Boundaries	completed	campaign	29914	
7	SLU	Oct COMPETITION: Youth Culture Insight challenge	completed	campaign	45204	
8	SLU	Oct_Power Course: Digital Wellness	inactive	campaign	78954	
9	SLU	Storysprint Empower Impactful Narratives 1	inactive	campaign	72339	

Step 5: Visualize Data Trends

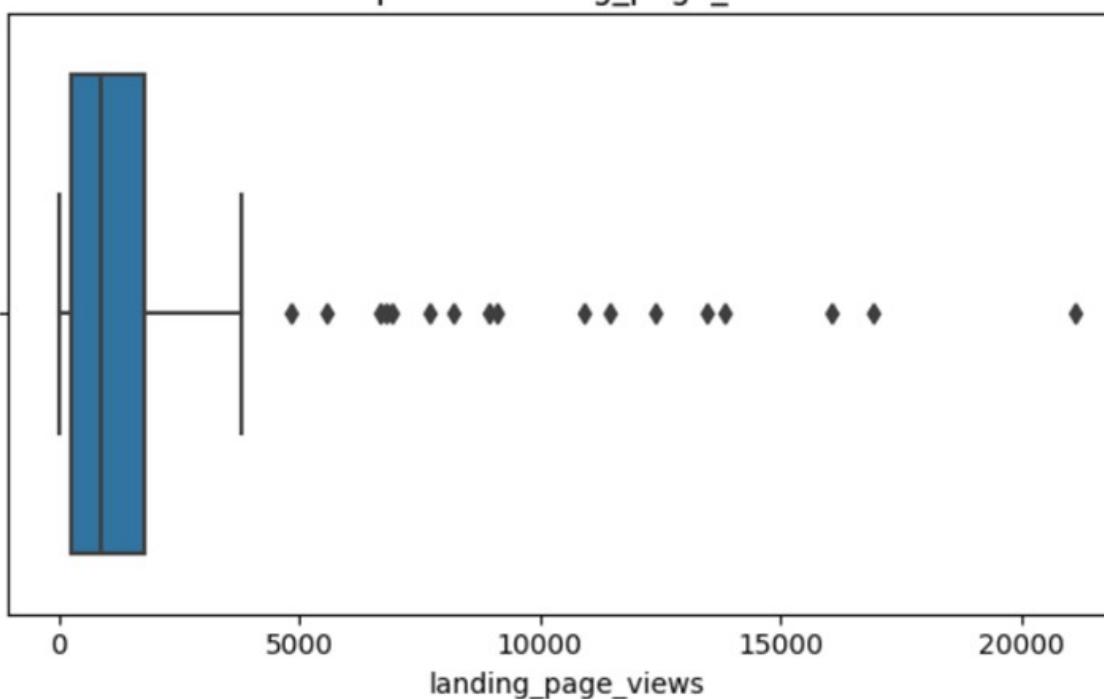
- Use graphs like histograms, line charts, and box plots to identify patterns.
- Analyze relationships between different variables.



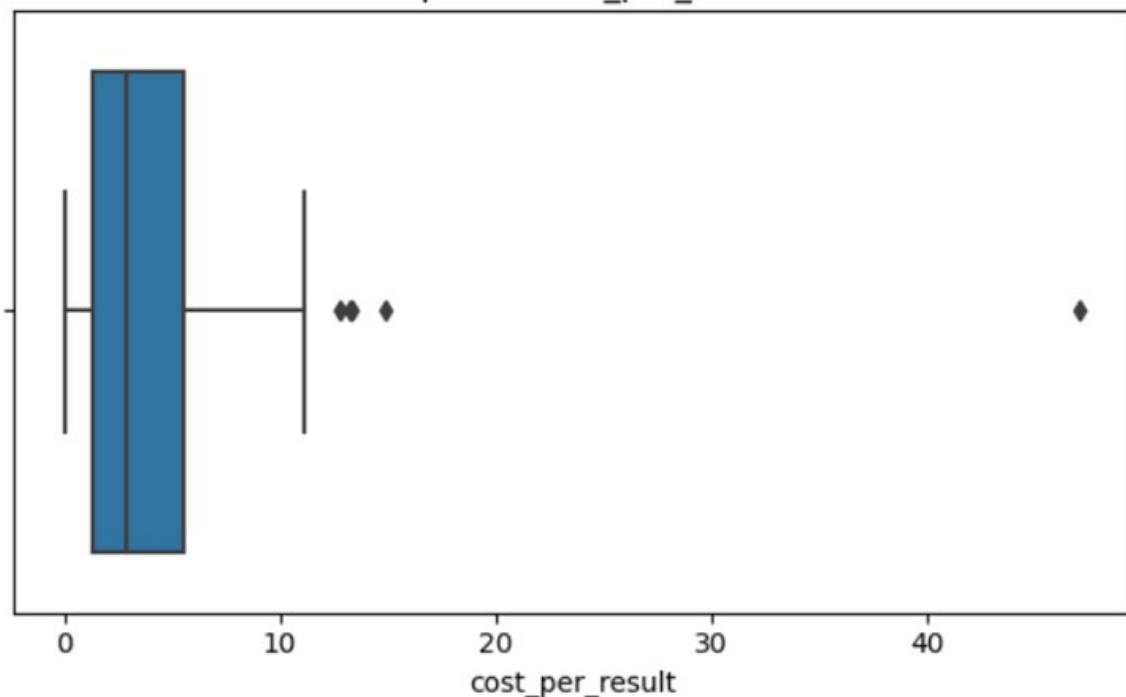
Boxplot of outbound_clicks



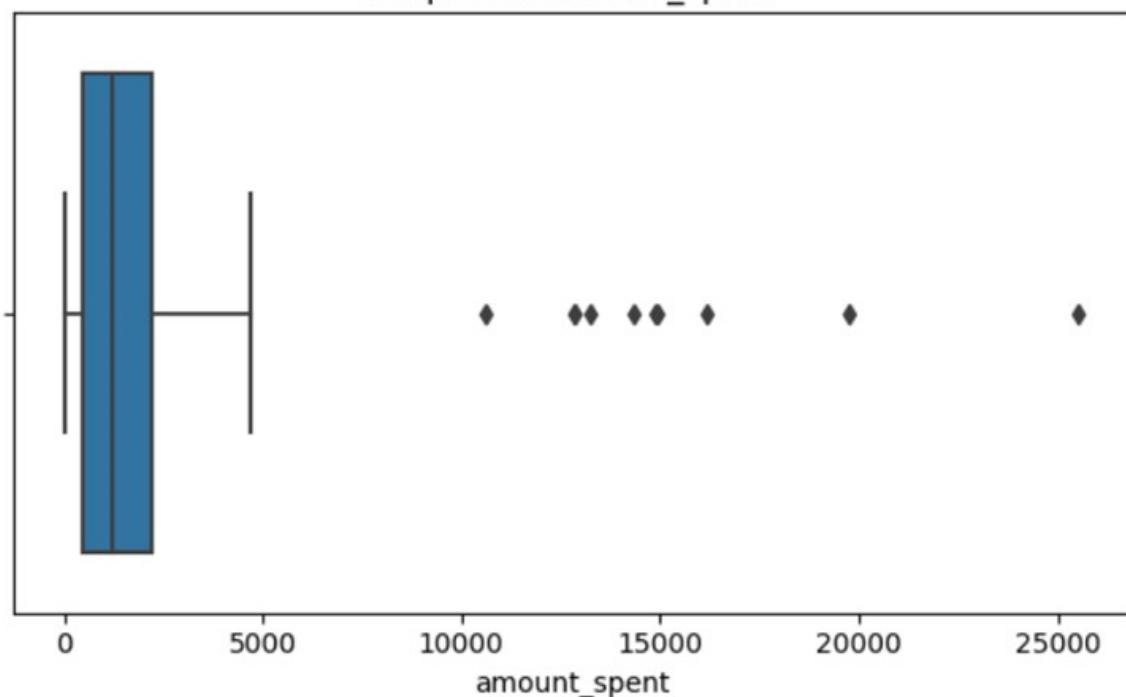
Boxplot of landing_page_views



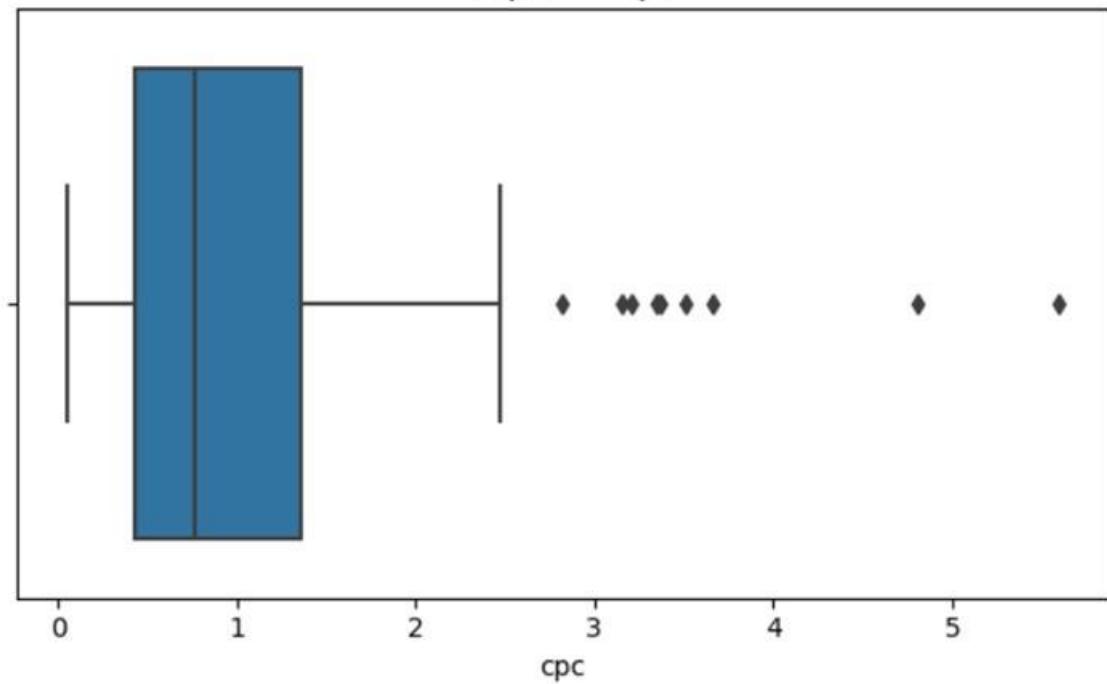
Boxplot of cost_per_result



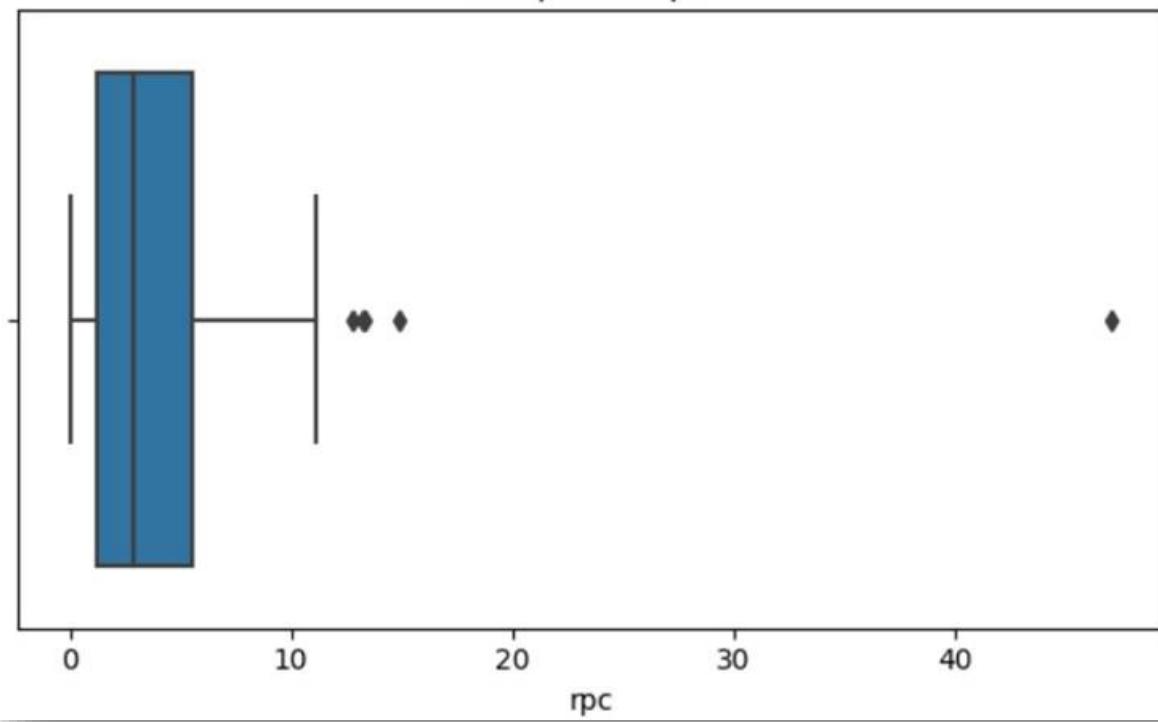
Boxplot of amount_spent



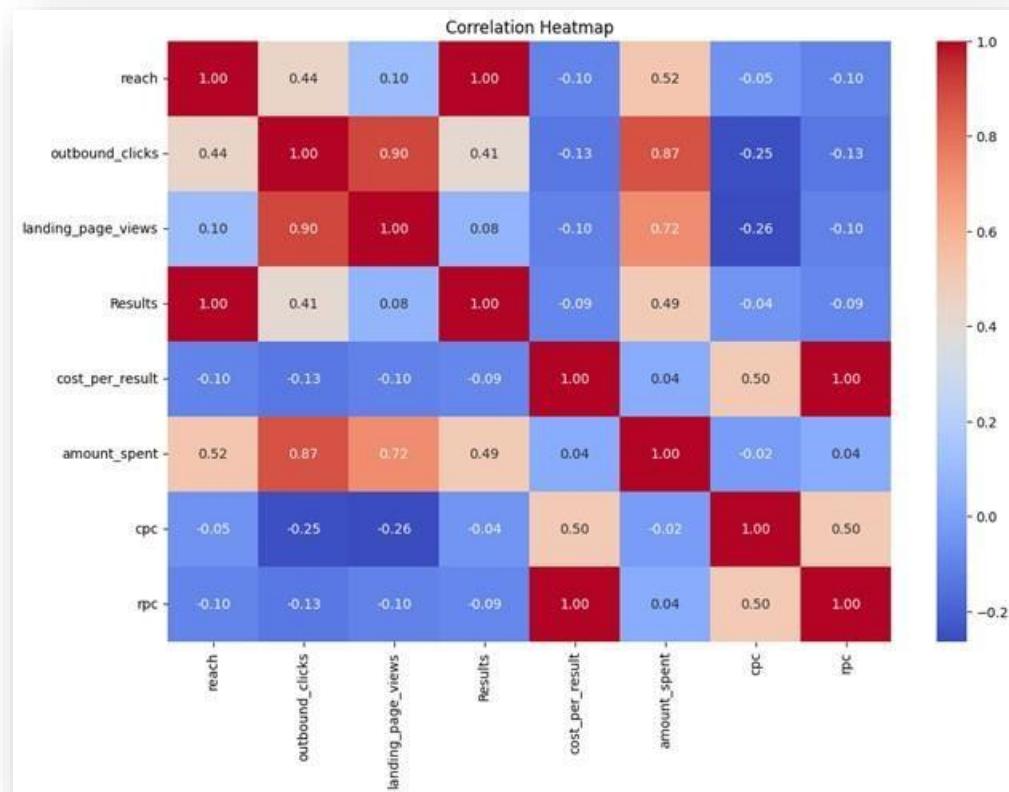
Boxplot of cpc



Boxplot of rpc



❖ Heat Map



Dataset: Cognito_Raw

Performing the Exploratory Data Analysis using Python

Step 1: Understand the Dataset

- View the first few rows to get an overview of the data.
- Check column names, data types, and the total number of records.

Research how to inspect a table's structure and data in Visual Studio Code using Python.

Pandas is a package in Python used to read data, manipulation and analysis.

Python Codes Used:

1. Reading the CSV File:

```
cognito_df = pd.read_csv(r"E:\ECCELERATE\Data\Cognito_Raw2(in).csv")
```

2. View First Few Rows:

```
cognito_df.head(5)
```

🔍 This gives you a quick peek at the dataset — ideal for spotting structure or obvious issues.

cognito_df.head(5)									Python	
	user_id	email	gender	UserCreateDate	UserLastModifiedDate	birthday	city	zip	state	...
0	00010567-1336-433c-a941-a612b3d	gikonyosalome19@gmail.com	Female	2024-11-17T21:25:56.381Z	2024-11-17T21:32:50.783Z	5/4/1996	NAIVASHA	20117	NAKURU	
1	aab8bd87-af83-4e21-816b-101cf051	evelyn.natasha.guo@gmail.com	Missing value	2025-01-19T02:07:06.113Z	2025-01-19T02:07:20.726Z	Missing value	Missing value	Missing value	Missing value	
2	46560951-a932-4889-a9e6-3b77f160	lauren.singh@rocketmail.com	Female	2024-03-26T23:35:43.292Z	2024-09-27T13:47:51.806Z	4/5/1990	Queens Village	11428	NY	
3	76b5629f-a024-4de8-9f10-59ebf8fd	anihmerry2019@gmail.com	Female	2024-03-31T19:04:21.735Z	2024-09-27T16:12:28.564Z	12/28/1998	Ibadan	200221	Oyo	
4	db17206b-2017-4b6a-9462-fc2bc7f7	lagrimasamie@gmail.com	Female	2024-03-25T20:36:26.352Z	2024-04-08T16:10:24.503Z	5/5/1999	Malolos City	3000	Bulacan	

5 rows x 9 cols 10 per page ⏪ Page 1 of 1 ⏩ 🔍 ⏹ ⏷ ⏸

3. Check Column Names & Data Types:

```
cognito_df.info()
```

⌚ This tells you what each column is, and whether it's stored as text, numeric, timestamp, etc.

```
cognito_df.info()
✓ 0.0s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 129178 entries, 0 to 129177
Data columns (total 9 columns):
 #   Column           Non-Null Count   Dtype  
--- 
 0   user_id          129178 non-null    object  
 1   email            129178 non-null    object  
 2   gender           86316 non-null    object  
 3   UserCreateDate   129178 non-null    object  
 4   UserLastModifiedDate 129178 non-null    object  
 5   birthdate        86316 non-null    object  
 6   city             86305 non-null    object  
 7   zip              86251 non-null    object  
 8   state            86154 non-null    object  
dtypes: object(9)
memory usage: 8.9+ MB
```

Here in the image we can see some of the date columns are in object datatype, so we have to change its datatype into DATETIME

Datatype is changed and the dataset is stored in the new variable final_cognito_df

```
final_cognito_df["UserCreateDate"] = pd.to_datetime(final_cognito_df["UserCreateDate"])
final_cognito_df["UserLastModifiedDate"] = pd.to_datetime(final_cognito_df["UserLastModifiedDate"])
final_cognito_df["birthdate"] = pd.to_datetime(final_cognito_df["birthdate"])
✓ 0.5s
```

After Changing the Datatype:

```
final_cognito_df.info()
✓ 0.0s

<class 'pandas.core.frame.DataFrame'>
Index: 86096 entries, 0 to 129174
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   user_id          86096 non-null   object  
 1   email             86096 non-null   object  
 2   gender            86096 non-null   object  
 3   UserCreateDate    86096 non-null   datetime64[ns, UTC]
 4   UserLastModifiedDate 86096 non-null   datetime64[ns, UTC]
 5   birthdate         86096 non-null   datetime64[ns] 
 6   city              86096 non-null   object  
 7   zip               86096 non-null   object  
 8   state              86096 non-null   object  
dtypes: datetime64[ns, UTC](2), datetime64[ns](1), object(6)
memory usage: 6.6+ MB
```

4. Count Total Rows & Columns:

```
cognito_df.shape
```

```
cognito_df.shape
✓ 0.0s
(129178, 9)
```

Step 2: Summarize the Data

- Generate basic statistics like mean, median, min, max, and standard deviation to understand the distribution.

Explore how to compute summary statistics in Python.

Since the `CognitoRaw(in).csv` dataset contains no numeric field column to which we can do basic statistics ,we are converting the birthdate column to age and we'll compute :

- Minimum
 - Maximum
 - Mean (average)
 - Median
 - Standard deviation
 - Count (just for completeness)
-

Python Code for Summary Statistics (VSCode):

Converting Birthdate into Age

```
#Converting 'birthdate' column to age
cognito_df['birthdate'] = pd.to_datetime(cognito_df['birthdate'], errors='coerce')
today = pd.to_datetime("today")
cognito_df['age'] = (today - cognito_df['birthdate']).dt.days // 365
```

1. Basic Descriptive Stats:

```
#Drop rows with missing age
df_age = cognito_df.dropna(subset=['age'])

#Describe the 'age' column
print(df_age['age'].describe())
```

```
#Describe the 'age' column
df_age['age'].describe()
✓ 0.3s
```

# age	
count	86316.0
mean	25.649833171138607
std	5.280843731899279
min	3.0
25%	22.0
50%	25.0
75%	28.0
max	101.0

8 rows x 1 cols 10 per page

What These Codes Will Show:

Statistic	Description
MIN (Age)	Smallest cohort size
MAX(Age)	Largest cohort size
AVG(Age)	Average cohort size
STDDEV (Age)	Standard deviation — shows how spread out the sizes are
COUNT (Age)	Total number of cohort records
MEDIAN (Age)	Middle value of all sizes (better indicator than average if there are outliers)

Step 3: Identify Missing and Duplicate Values

- Detect missing data to determine if any cleaning is needed.
- Identify duplicate records that may need removal.

Look up methods to check for missing values and duplicates in VS-Code.

Part 1: Detect Missing (NULL) Values

1. Missing Count for Each Column

```
cognito_df.isnull().sum()
```

The screenshot shows a Jupyter Notebook cell with the following content:

```
cognito_df.isnull().sum()
```

Execution results:

✓ 0.0s

0

user_id	0
email	0
gender	42862
UserCreateDa	0
UserLastModi	0
birthdate	42862
city	42873
zip	42927
state	43024

9 rows x 1 cols 10 ▾ per page

Part 2: Detect Duplicate Records

2. Fully Duplicate Rows (every column repeated)

```
cognito_df.duplicated().sum()  
cognito_df[cognito_df.duplicated()]
```

- No duplications in this.

```
cognito_df.duplicated().sum()
✓ 0.5s
np.int64(0)

cognito_df[cognito_df.duplicated()]
✓ 0.4s
✉ user_id ✉ email ✉ gender ✉ UserCreateDate ✉ UserLastModifiedDate ✉ birthdate ✉ city ✉ zip ✉ state ...
```

What to Look For:

Feature	Insight
NULL Values	Indicate incomplete or missing data that may need cleaning
Fully Duplicate Rows	Often accidental — likely safe to remove

Step 4: Spot Outliers and Anomalies

- Use techniques like the Interquartile Range (IQR) to find unusual values that might distort analysis.

Research how outliers are detected using SQL and statistical methods.

Interquartile Range (IQR) Method Overview

Steps:

5. Calculate Q1 (25th percentile) and Q3 (75th percentile)
6. Compute $IQR = Q3 - Q1$
7. Define outlier bounds:
 - o Lower Bound = $Q1 - 1.5 \times IQR$
 - o Upper Bound = $Q3 + 1.5 \times IQR$
8. Flag values outside these bounds as outliers

PostgreSQL Queries to Detect Outliers

1. Get Q1, Q3, and IQR

```

Q1 = cognito_df['age'].quantile(0.25)

Q3 = cognito_df['age'].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR

iqr_outliers = cognito_df[(cognito_df['age'] < lower_bound) |
(cognito_df['age'] > upper_bound)]
print(f"IQR Outliers Count: {len(iqr_outliers)}")

```

```

Q1 = cognito_df['age'].quantile(0.25)
Q3 = cognito_df['age'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

iqr_outliers = cognito_df[(cognito_df['age'] < lower_bound) | (cognito_df['age'] > upper_bound)]

print(f"IQR Outliers Count: {len(iqr_outliers)}")
✓ 0.0s
IQR Outliers Count: 2492

```

What to Look For:

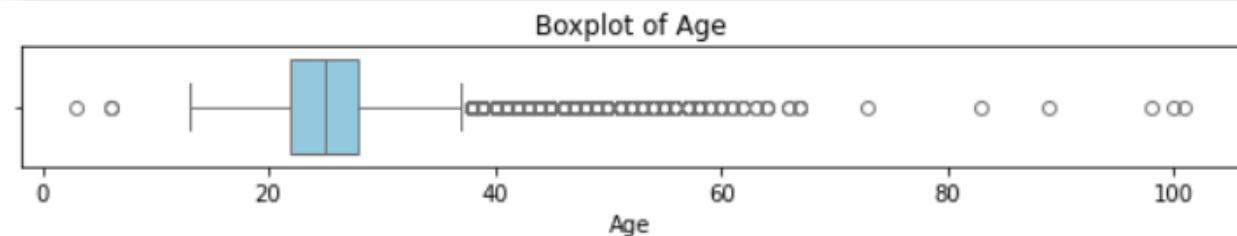
Metric	Purpose
Q1 & Q3	Measure the range where most data lies
Outliers	Any value falling far outside this range
Next Step	Review outliers — correct, cap, or remove as needed

Step 5: Visualize Data Trends

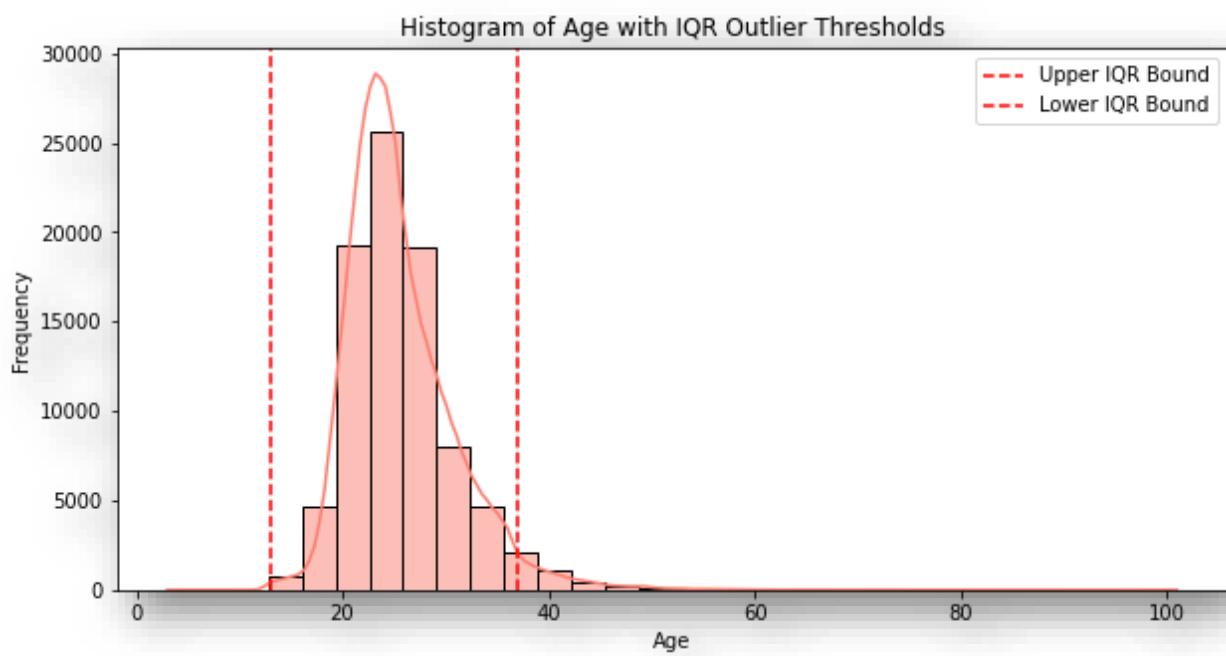
- Use graphs like histograms, line charts, and box plots to identify patterns.
- Analyze relationships between different variables.

Find out how to create basic visualizations for data analysis.

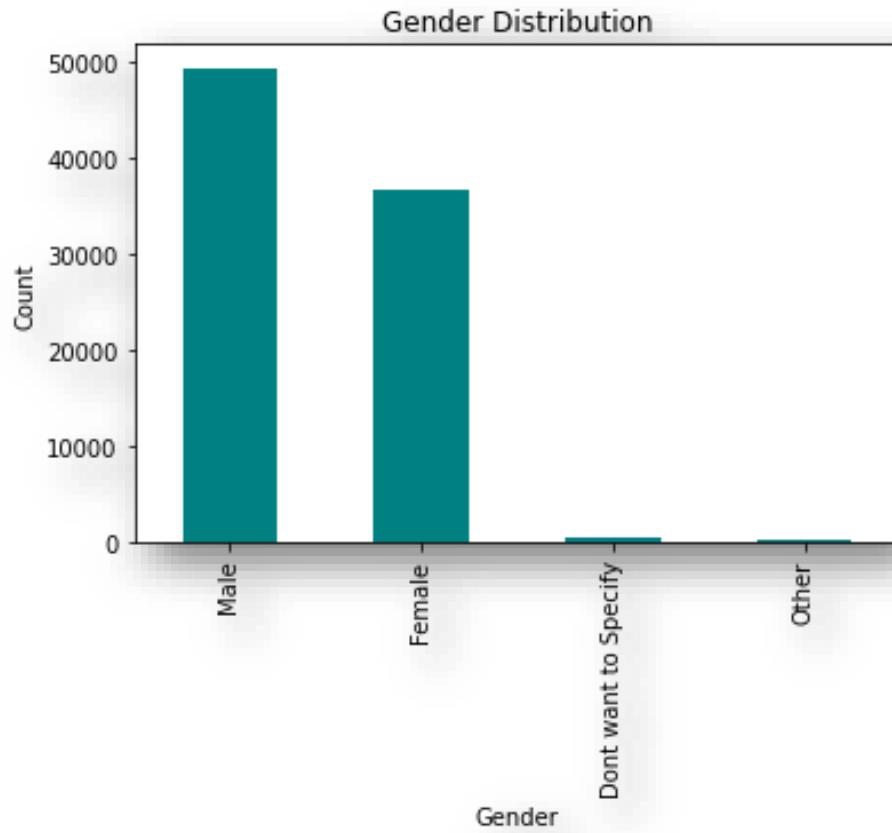
1. Visualization of Outliers:



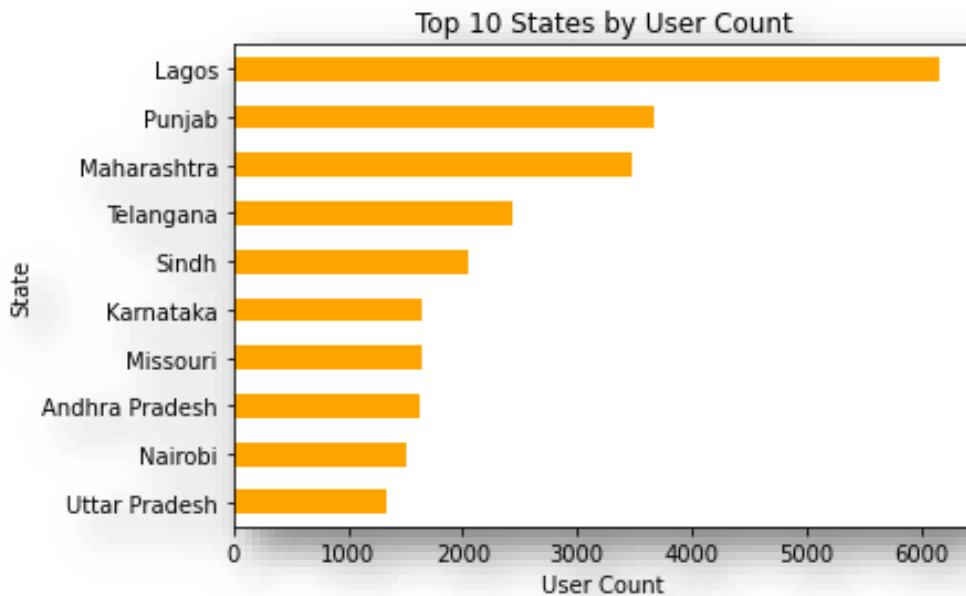
2. Distribution of User based on Age:



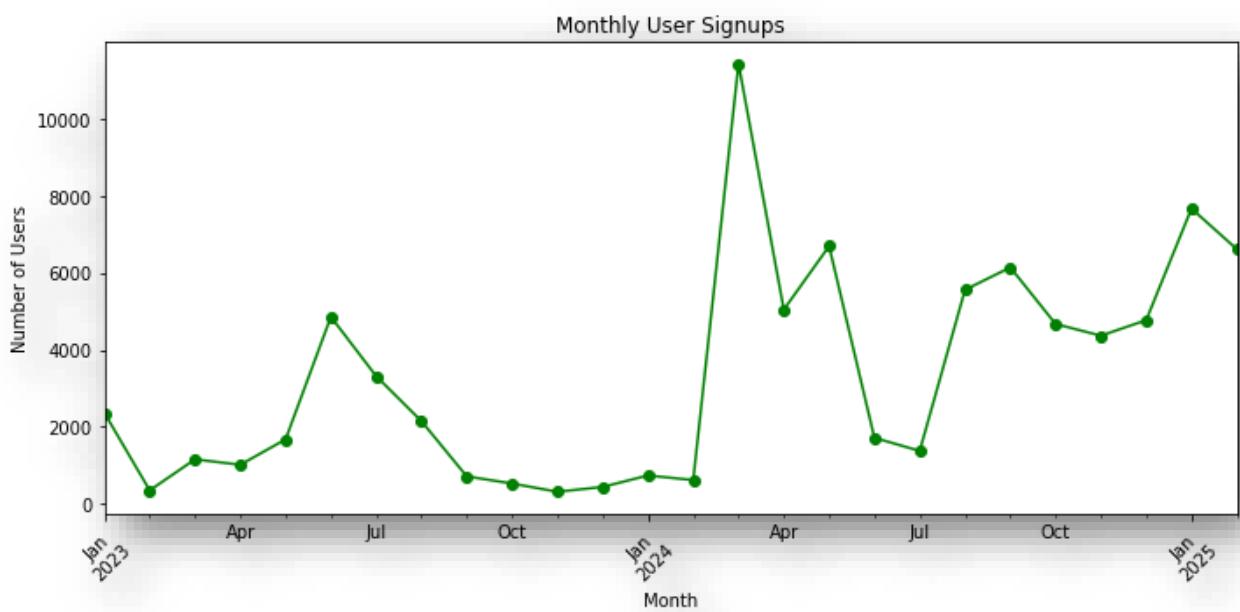
3.Gender Distribution



4.Users by State (Top 10 States)



5. Account Creation Trend Over the Specified Period



Dataset: Learner_Raw

Performing the Exploratory Data Analysis

Step 1: Understand the Dataset

- View the first few rows to get an overview of the data.
- Check column names, data types, and the total number of records.

Research how to inspect a table's structure and data in Visual Studio Code.

Pandas is a package in Python used to read data, manipulation and analysis.

Objective:

Understand the structure of the dataset by inspecting:

- First few rows
 - Column names
 - Data types
 - Total number of records
-

Codes in VS-Code:

1. Reading the CSV File:

```
learner_df = pd.read_csv(r"E:\ECCELERATE\Data\Learner_Raw(in).csv")
```

1. View First Few Rows:

```
learner_df.head(5)
```

This gives a quick snapshot of the data format and sample values.

learner_df.head(5)

✓ 0.0s Open 'learner_df' in Data Wrangler

	learner_id	country	degree	institution	major
0	Learner#00004f18-8b86-4fe4-ad7e-1	Nigeria	Undergraduate Student	Federal University of Technology Ow	Civil Engineering
1	Learner#00006478-745f-49bf-b126-1	Nigeria	Missing value	Missing value	Missing value
2	Learner#00010567-1336-433c-a941-1	Kenya	Graduate Student	UNICAF UNIVERSITY	Environmental Sustainability
3	Learner#00011c80-0c5c-4601-9696-1	Bangladesh	Missing value	Missing value	Missing value
4	Learner#000141a7-4c82-401f-a2e6-1	Nigeria	Missing value	Missing value	Missing value

5 rows x 5 cols 10 per page

« < Page 1 of 1 > »

2. Check Column Names and Data Types:

```
learner_df.info()
```

Lists each column with its data type (e.g., text, integer, timestamp).

learner_df.info()

✓ 0.1s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 129259 entries, 0 to 129258
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   learner_id  129259 non-null   object 
 1   country     126984 non-null   object 
 2   degree      76566 non-null   object 
 3   institution 76186 non-null   object 
 4   major       76388 non-null   object 
dtypes: object(5)
memory usage: 4.9+ MB
```

3. Get Total Number of Rows & Columns:

```
Learner_df.shape
```

□ Confirms how many records are present in the dataset.

```
learner_df.shape
✓ 0.0s
(129259, 5)
```

Step 2: Summarize the Data

- Generate basic statistics like mean, median, min, max, and standard deviation to understand the distribution.

Since the Learner_Raw.csv dataset contains no numeric fields its not possible to do basic statistics

Step 1: Identify Missing and Duplicate Values

- Detect missing data to determine if any cleaning is needed.
- Identify duplicate records that may need removal.

Look up methods to check for missing values and duplicates in Visual Studio Code

Objective:

- Detect missing (NULL) values
- Identify fully or partially duplicate records

Part A: Check for Missing Values

1. Count of NULLs for All Columns

```
learner_df.isnull().sum()
```

⌚ Compare filled counts with total rows to detect missing values.

```
learner_df.isnull().sum()
✓ 0.0s
# 0
```

learner_id	0
country	2275
degree	52693
institution	53073
major	52871

Part B: Check for Duplicates

3. Fully Duplicate Records

```
learner_df.duplicated().sum()
learner_df[learner_df.duplicated()]
```

❖ These rows are identical in all fields — likely safe to remove.

```
learner_df.duplicated().sum()
✓ 0.0s
np.int64(0)

learner_df[learner_df.duplicated()]
✓ 0.1s
```

learner_id	country	degree	institution	major
1	1	1	1	1

- No full duplicate rows

What to Do Next:

Issue Type	Meaning
NULLs	May indicate missing inputs, timing issues, or data pipeline errors
Full Duplicates	Usually safe to delete unless time-sensitive duplicates are allowed

Step 4: Spot Outliers and Anomalies

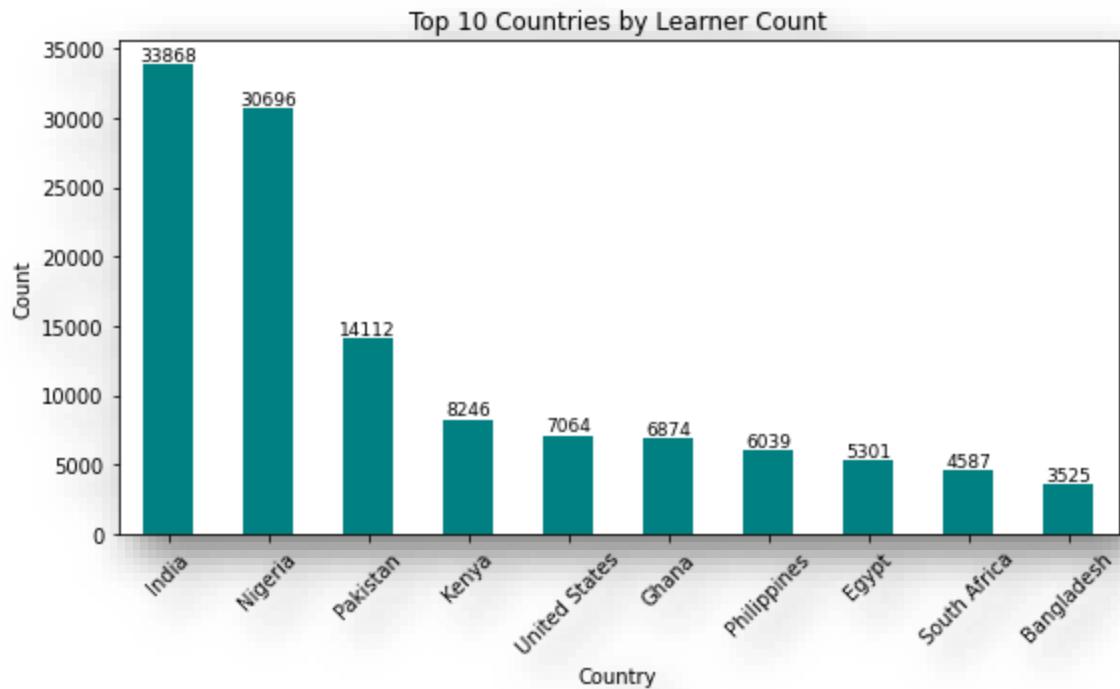
- Use techniques like the Interquartile Range (IQR) to find unusual values that might distort analysis.
 - Objective of finding the outliers is to find which record is being odd from the other records
 - But here all the columns are categorial so we can't do usual IQR method to find the outliers.
 - If we go for rare, uncommon, low frequency occurrence data as outliers we may lose data of users.
 - So we can't do outlier analysis for this dataset
-

Step 5: Visualize Data Trends

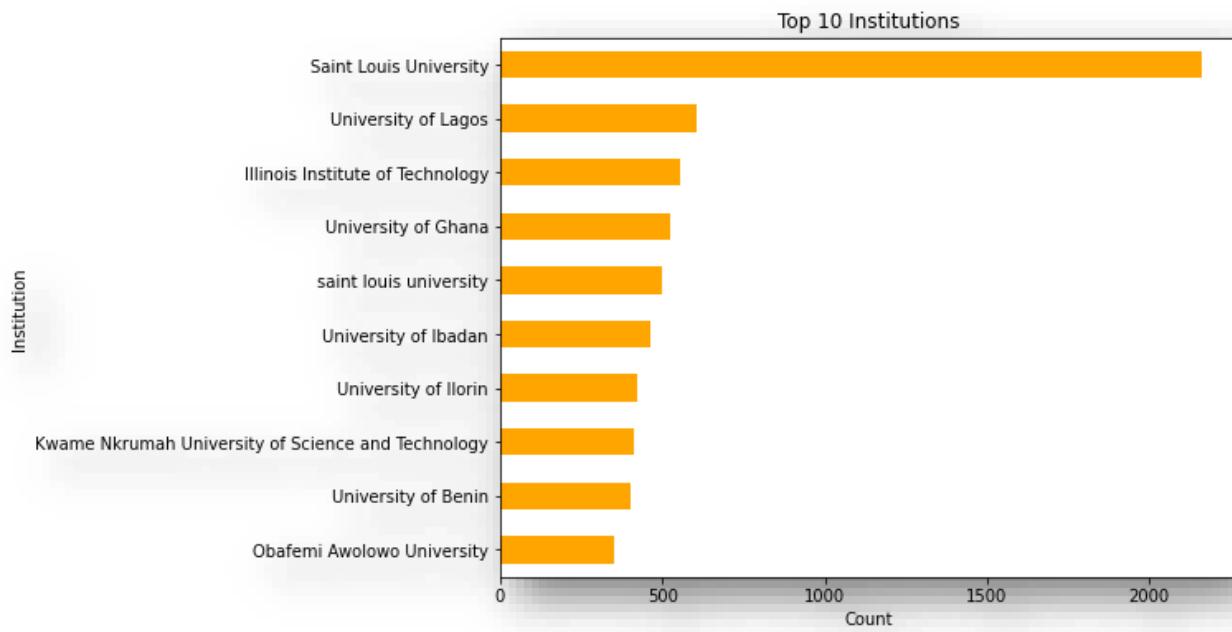
- Use graphs like histograms, line charts, and box plots to identify patterns.
- Analyze relationships between different variables.

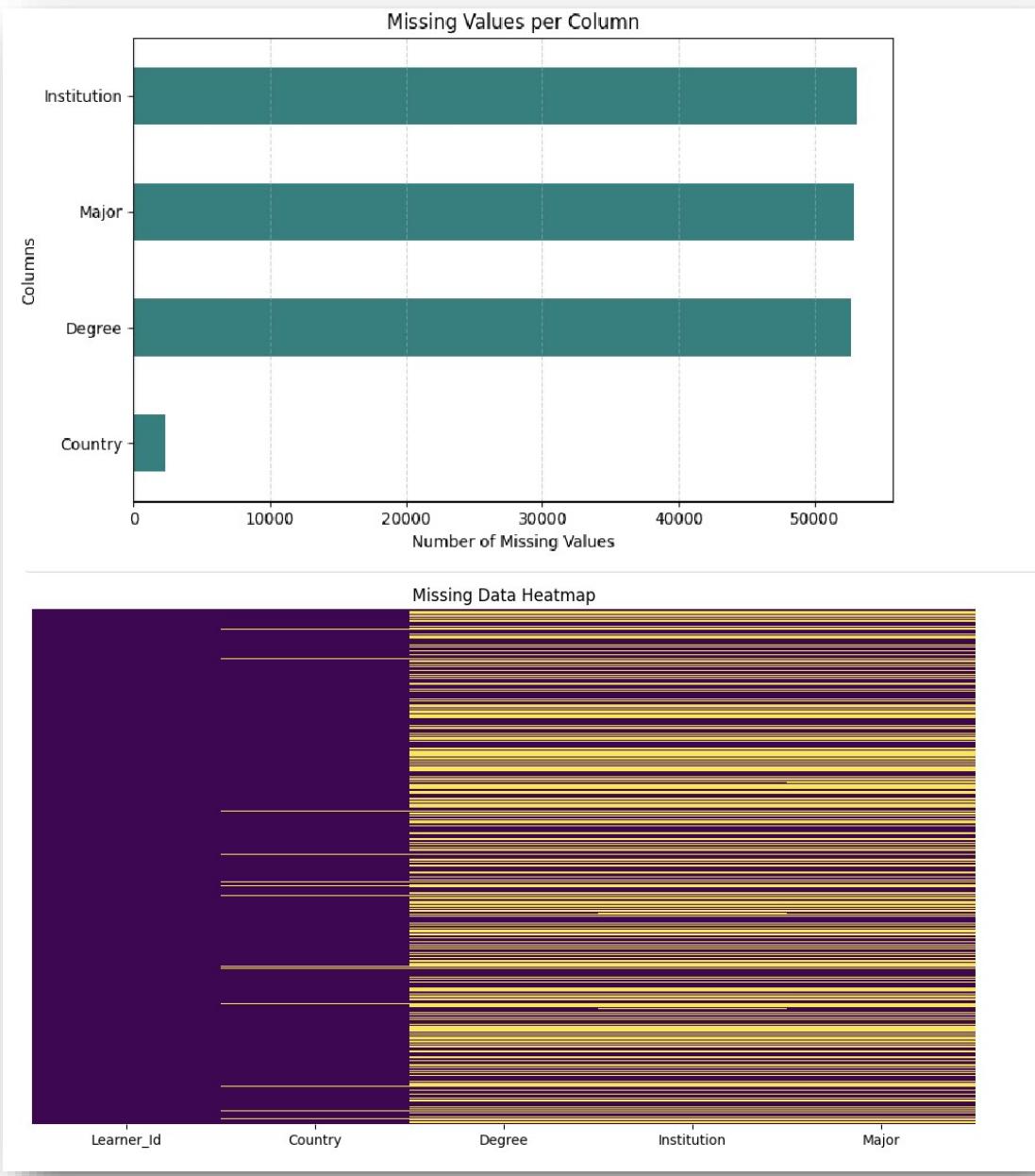
Find out how to create basic visualizations for data analysis.

1. Top Countries by Learner Count (Top 10 Countries)



2. Top 10 Institutions





Dataset: Opportunity_Raw

Performing the Exploratory Data Analysis

Step 1: Understand the Dataset

- View the first few rows to get an overview of the data.
- Check column names, data types, and the total number of records.

Research how to inspect a table's structure and data in Visual Studio Code.

View First Few Rows (Top 5 Records)

```
SELECT *
FROM opportunity_raw
LIMIT 5;
```

The screenshot shows a SQL editor interface. At the top, there is a toolbar with a 'Query' tab, a 'Query History' tab, and a 'Execute script' button (labeled F5). Below the toolbar, the query text is displayed:

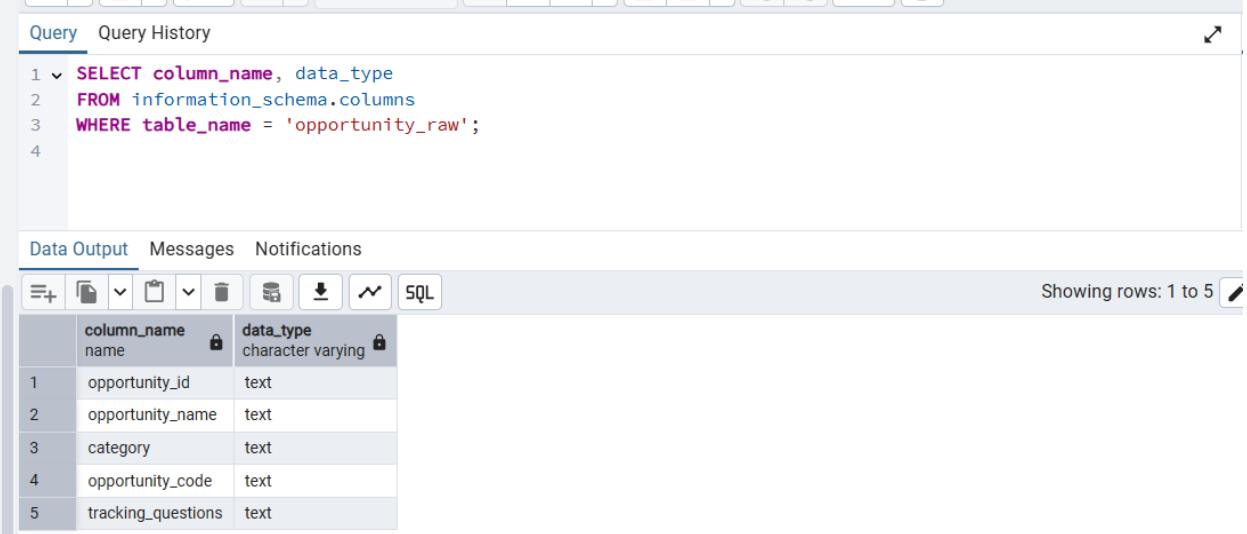
```
1 v SELECT *
2   FROM opportunity_raw
3   LIMIT 5;
4
```

Below the query text, there is a 'Data Output' tab selected. The results section shows the following data:

	opportunity_id	opportunity_name	category	opportunity_code	tracking_questions
1	Opportunity#00000000G127E8VYE08TXBT6X	Choosing and Planning for Your Major	Event	E501873	[null]
2	Opportunity#00000000G2PB6VB4ANR28CV2P	The Financial: Article Writing Competition Test	Competition	M523594	[null]
3	Opportunity#00000000G4AM4J9NBMPK3TJ...	Entrepreneurship and Innovation	Internship	I289641	[null]
4	Opportunity#00000000G4F19XBEXPWKS8F3N	Statement of Purpose (SOP) Writing Worksh...	Event	E258709	[null]
5	Opportunity#00000000G4KVEP36NNJR5YWTJ	Project Management	Internship	I584159	[null]

Check Column Names & Data Types

```
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = 'opportunity_raw';
```



Query History

```
1 SELECT column_name, data_type
2 FROM information_schema.columns
3 WHERE table_name = 'opportunity_raw';
4
```

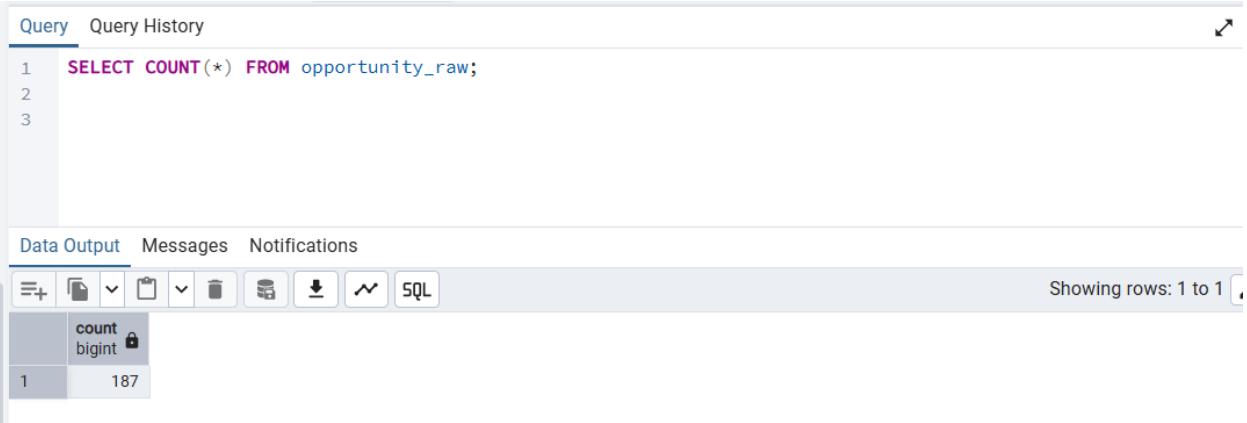
Data Output Messages Notifications

Showing rows: 1 to 5

	column_name	data_type
1	opportunity_id	text
2	opportunity_name	text
3	category	text
4	opportunity_code	text
5	tracking_questions	text

Count Total Rows

```
SELECT COUNT(*) FROM opportunity_raw;
```



Query History

```
1 SELECT COUNT(*) FROM opportunity_raw;
2
3
```

Data Output Messages Notifications

Showing rows: 1 to 1

	count
1	187

Step 2: Summarize the Data

- Generate basic statistics like mean, median, min, max, and standard deviation to understand the distribution.

Explore how to compute summary statistics in SQL.

Objective:

- Generate basic statistics like mean, median, min, max, and standard deviation to understand the distribution.
 - Explore how to compute summary statistics in SQL.
-

Not Applicable to This Dataset

The `opportunity_raw` dataset **does not contain any numeric columns**, which means **standard statistical functions such as**:

- `MIN(size)`
- `MAX(size)`
- `AVG(size)`
- `STDDEV(size)`
- `MEDIAN(size)`

are not applicable to this table.

All columns — such as `opportunity_id`, `opportunity_name`, `category`, `opportunity_code`, and `tracking_questions` — are of **textual (string) data type**, not numeric.

Step 3: Identify Missing and Duplicate Values

- Detect missing data to determine if any cleaning is needed.
- Identify duplicate records that may need removal.

Look up methods to check for missing values and duplicates in PostgreSQL.

Objective:

- Detect missing data to determine if any cleaning is needed
 - Identify duplicate records that may need removal
 - Use PostgreSQL queries only — no Python
-

◆ Part A: Check for Missing (NULL) Values**Query:**

```

SELECT
  COUNT(*) FILTER (WHERE opportunity_id IS NULL) AS null_opportunity_id,
  COUNT(*) FILTER (WHERE opportunity_name IS NULL) AS null_opportunity_name,
  COUNT(*) FILTER (WHERE category IS NULL) AS null_category,
  COUNT(*) FILTER (WHERE opportunity_code IS NULL) AS null_opportunity_code,
  COUNT(*) FILTER (WHERE tracking_questions IS NULL) AS
null_tracking_questions
FROM opportunity_raw;

```

Query History

```

2 COUNT(*) FILTER (WHERE opportunity_id IS NULL) AS null_opportunity_id,
3 COUNT(*) FILTER (WHERE opportunity_name IS NULL) AS null_opportunity_name,
4 COUNT(*) FILTER (WHERE category IS NULL) AS null_category,
5 COUNT(*) FILTER (WHERE opportunity_code IS NULL) AS null_opportunity_code,
6 COUNT(*) FILTER (WHERE tracking_questions IS NULL) AS null_tracking_questions
7 FROM opportunity_raw;
8

```

Data Output

	null_opportunity_id	null_opportunity_name	null_category	null_opportunity_code	null_tracking_questions
1	0	0	0	0	69

Showing rows: 1 to 1

- **tracking_questions** is completely empty (NULL in all records), likely optional or inactive.

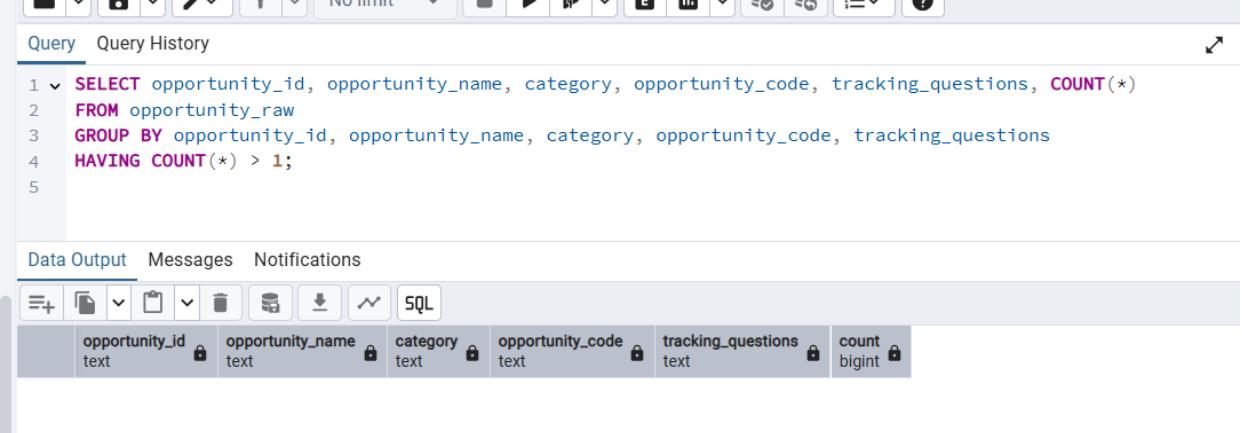
◆ Part B: Check for Fully Duplicate Records

Query:

```

SELECT opportunity_id, opportunity_name, category, opportunity_code,
tracking_questions, COUNT(*)
FROM opportunity_raw
GROUP BY opportunity_id, opportunity_name, category, opportunity_code,
tracking_questions
HAVING COUNT(*) > 1;

```



```

1 v SELECT opportunity_id, opportunity_name, category, opportunity_code, tracking_questions, COUNT(*)
2 FROM opportunity_raw
3 GROUP BY opportunity_id, opportunity_name, category, opportunity_code, tracking_questions
4 HAVING COUNT(*) > 1;
5

```

The screenshot shows a SQL query editor interface. The top bar has various icons for file operations like Open, Save, Print, and Help. Below the bar, there's a tab labeled "Query History". The main area contains a SQL query. At the bottom, there are tabs for "Data Output", "Messages", and "Notifications". Below these tabs is a toolbar with icons for creating new queries, opening files, saving, printing, and running SQL. A table definition is shown below the toolbar:

	opportunity_id	opportunity_name	category	opportunity_code	tracking_questions	count
	text	text	text	text	text	bigint

Result Interpretation:

- **0 fully duplicate records** were found.
 - No rows are exact matches across all columns.
-

What to Do Next:

Issue Type	Recommended Action
tracking_questions (all NULLs)	Investigate if this column is still needed or was never populated
Full Duplicates	✓ None found — no cleanup needed

Summary:

Step 3 confirms that the dataset has **no duplicate records** and only **one column (tracking_questions) with 100% NULL values**.

No urgent cleaning is needed — but tracking data can be archived or flagged for review.

Step 4: Spot Outliers and Anomalies

- Use techniques like the Interquartile Range (IQR) to find unusual values that might distort analysis.

Research how outliers are detected using SQL and statistical methods.

Objective:

- Use techniques like the Interquartile Range (IQR) to detect unusual values
 - First: **Determine if outlier detection is relevant for this dataset**
-

Is Outlier Detection Applicable?

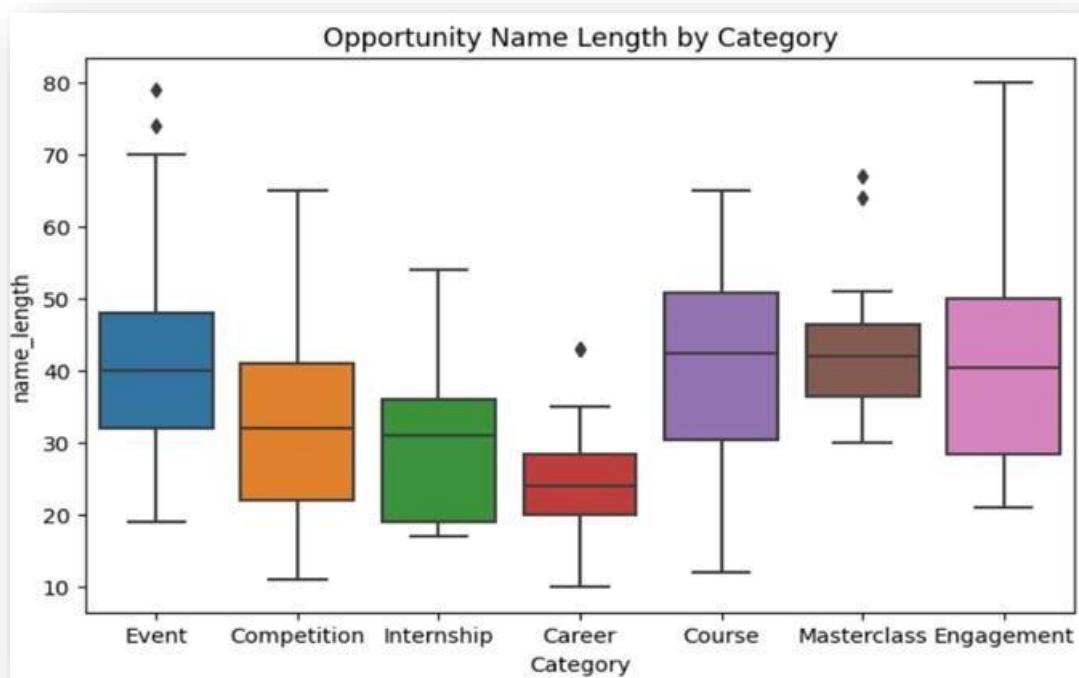
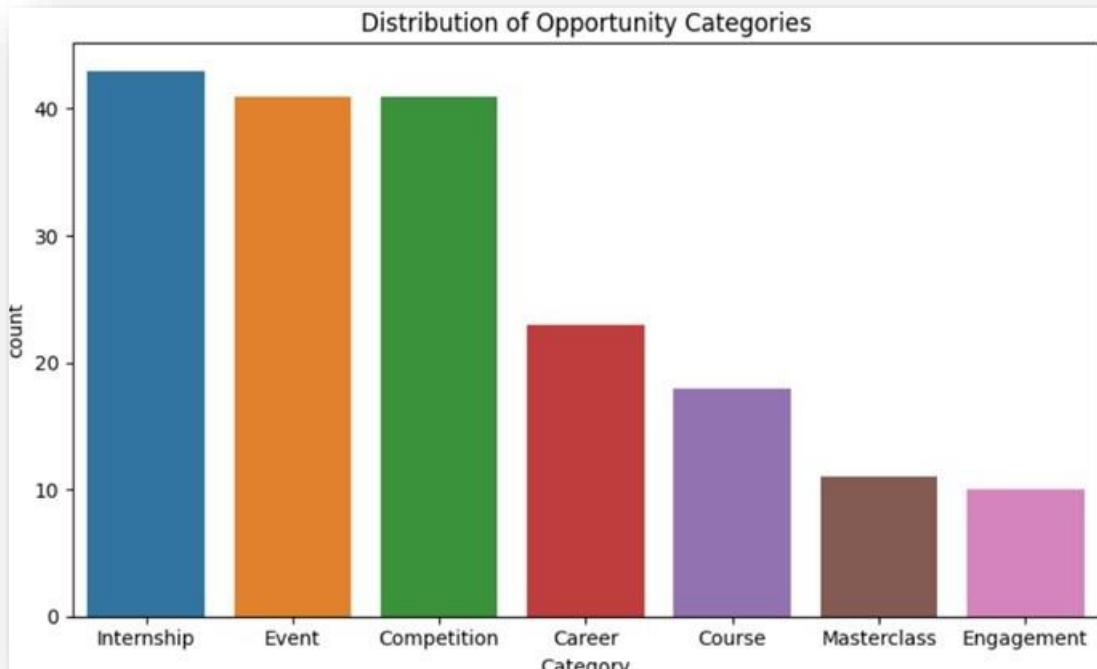
Requirement	opportunity_raw Status
Contains numeric data	✗ No — all columns are of type TEXT
Has measurable values (e.g. size, price, score)	✗ No — only identifiers, names, and categories
Columns suitable for IQR/mean/std-based checks	✗ None present

Conclusion:

✗ **Outlier detection is not applicable** to the opportunity_raw dataset.
 The dataset contains only **categorical and identifier fields**, with no numeric data to evaluate using statistical methods like **IQR, z-scores, or standard deviation**.

Step 5: Visualize Data Trends

- Use graphs like histograms, line charts, and box plots to identify patterns.
- Analyze relationships between different variables.



Key Findings & Next Steps for Data Cleaning and Transformation

Contain all Datasets

1. Learner_Raw Dataset

Key Findings:

- Primary Key `learner_id` is valid — unique and clean.
- High missing values in:
 - `degree`: ~41%
 - `institution`: ~41%
 - `major`: ~41%
 - `country`: 2,275 missing entries.
- Inconsistent text formatting:
 - Degree types: "Bachelor", "Bachelors", "BS", etc.
 - Major: Case inconsistencies (e.g., "computer science" vs "Computer Science").
- No valid Foreign Key (FK) relationships with `Cognito_Raw` or `LearnerOpportunity_Raw`.

Next Steps for Cleaning:

- Normalize text (standardize casing for degree and major).
 - Deduplicate vague entries (e.g., "null", "Other").
 - Consider imputation:
 - Categorical mode for `degree`, `institution`, `major` if used in modeling.
 - Retain as independent dimension table until foreign key alignment is confirmed.
-

2. Opportunity_Raw Dataset

Key Findings:

- `opportunity_id`: Valid Primary Key.
- `opportunity_code`: Appears unique; possible alternate key.
- `tracking_questions`: ~90% NULL → likely deprecated.
- Duplicate `opportunity_name` values exist.

Next Steps for Cleaning:

- Drop `tracking_questions` unless confirmed relevant.

- Deduplicate or classify `opportunity_name` if required for reporting.
 - Standardize `category` (Event, Internship, Competition) for consistency.
 - Prepare for join with `LearnerOpportunity_Raw` via `opportunity_id`.
-

3. Cognito_Raw2 Dataset

Key Findings:

- `user_id`: Clean, unique — **strong PK**.
- `email`: Mostly complete, but **duplicate risk**.
- Missing/inconsistent in:
 - `gender` (~22% missing)
 - `birthdate` (~10%)
 - `city, zip, state` (~15%)
- Date fields (`UserCreateDate`, `UserLastModifiedDate`) stored as **strings**.
- `custom:learner_id` does **not match `learner_id`** in `Learner_Raw`.

Next Steps for Cleaning:

- Convert all date fields to proper `datetime` format.
 - Standardize:
 - `gender, city, state` values
 - Validate uniqueness of `email` if used as a secondary ID.
 - Assess if `user_id` or `email` can act as bridge to learner-level datasets.
-

4. Cohort_Raw Dataset

Key Findings:

- `cohort_code`: Clean, appears **unique** — candidate PK.
- `cohort_id`: **Redundant**, same placeholder in all rows.
- `start_date, end_date`: Stored in **epoch timestamp (ms)** → unreadable as-is.
- `size`: Numerical, clean — but **outliers suspected** (e.g., 100,000).

Next Steps for Cleaning:

- Drop `cohort_id` — not useful.
- Convert `start_date` and `end_date` to human-readable datetime.
- Analyze `size` outliers; define thresholds for valid cohort sizes.
- Confirm whether size reflects **actual enrollment** or **capacity**.

5. LearnerOpportunity_Raw Dataset

Key Findings:

- **186 exact duplicates** found.
- `assigned_cohort`: Missing in ~11.7% of records.
- `apply_date`: Valid ISO format but stored as **text** — 188 missing.
- `status`: Numeric codes (e.g., 1070, 1010) — **undocumented** meanings.
- `enrollment_id` is **not unique** — consider composite key with `learner_id`.

Next Steps for Cleaning:

- **Remove exact duplicate rows.**
 - **Decode status values** — obtain mapping documentation.
 - **Convert apply_date** to datetime.
 - **Impute assigned_cohort** if used in cohort-level analysis.
 - **Treat enrollment_id + learner_id** as composite primary key.
-

6. Marketing_Campaign_Raw Dataset

Key Findings:

- No single primary key — **possible composite key**: Campaign name + Result type + Reporting starts.
- **5 duplicate rows**.
- Missing data in critical fields (4–6%):
 - `reach`, `results`, `clicks`, `cost per result`, etc.
- **Outliers**:
 - Reach: Max = 141M+, Mean = 1.7M
 - Results: Max = 182M
- Reporting starts: Only **1 unique value** — not useful for time-series.

Next Steps for Cleaning:

- **Remove duplicate rows.**
- **Impute or drop missing** numeric fields (consider imputing with 0 or median, depending on business context).
- **Investigate and handle outliers** (cap or segment).
- **Ignore or enrich Reporting starts** — may require full timeline data.
- Standardize categorical fields (e.g., Delivery status, Result type).

Executive Summary: Cross-Dataset Next Steps

Task	Applies To	Priority
Normalize and standardize text	Learner_Raw, Cognito_Raw2	High
Convert text to datetime	Cohort_Raw, Cognito_Raw2, LearnerOpp	High
Remove duplicates	LearnerOpp_Raw, Marketing_Campaign_Raw	High
Handle outliers	Cohort_Raw, Marketing_Campaign_Raw	Medium
Decode numeric codes	LearnerOpp_Raw (status)	High
Drop deprecated columns	Opportunity_Raw (tracking_questions)	High
Impute missing data	Across all datasets	High
Validate key relationships	All datasets (joins, PK/FK alignment)	High