



WEEK 1: EXPLORATORY DATA ANALYSIS REPORT ON PROVIDED DATASETS

TEAM NAME: 0704 DVA | TEAM 06

TEAM MEMBERS

NAME	EMAIL ID
Ushna Adnan	ushna859@gmail.com
Muna Saha	Munasaha369@gmail.com
Divya P	Divyadp0825@gmail.com
Salman Rahobar	kazisalmanrahobar@gmail.com
Ritik Mathpal	Work.ritikmathpal@gmail.com
Shoaib Salman	Shoaibsalman28@gmail.com
Syed Rayyan	Syedramsey18@gmail.com

Contents

CHAPTER 1: INTRODUCTION	4
CHAPTER 2: DATA FAMILIARIZATION	5
2.1 Learner DataSet:	5
2.2 Opportunity DataSet:	8
2.3 Cohort DataSet:	10
2.4 Marketing DataSet:	12
2.5 Learning Opportunity DataSet:	15
1) DATA DICTIONARY OVERVIEW:	15
2.6 Cognito DataSet:	18
CHAPTER 3: DATA SETUP	20
3.1 Leaner DataSet	20
3.2 Opportunity DataSet	22
3.3 Cohort DataSet	24
3.4 Marketing DataSet	26
3.5 Learning Opprtunity DataSet:	28
3.6 Cognito DataSet	30
CHAPTER 4: EXPLORATORY DATA ANALYSIS (EDA)	32
4.1 Leaner DataSet	32
4.2 Opportunity DataSet	40
4.3 Cohort DataSet	46
4.4 Marketing DataSet	57
2. SUMMARY STATISTICS:	59
3. MISSING VALUES ANALYSIS:	61
4. DUPLICATE RECORDS:	62
5. OUTLIER DETECTION (REACH):	63
7. KEY FINDINGS:	65
4.5 Learning Oppurtunity DataSet	67
4.6 Cognito DataSet	75
CHAPTER 5: DATA CLEANING AND VALIDATION:	81
5.1 Learning DataSet	81
5.3 Cohort DataSet	83

5.4	Marketing DataSet	84
5.5	Learning Opportunity DataSet	85
1.	HANDLING MISSING DATA	85
2.	REMOVING DUPLICATES AND FIXING INCONSISTENCIES	85
3.	VALIDATING AND STANDARDIZING DATA TYPES.....	85
5.6	Cognito DataSet.....	87
CHAPTER 6: CONCLUSION		88

CHAPTER 1: INTRODUCTION

This report presents a detailed summary of the work carried out during our internship, with a strong focus on various stages of data analysis, processing, and validation. The main objective was to engage with real-world datasets—ensuring data quality, conducting exploratory data analysis (EDA), and extracting valuable insights to support informed decision-making.

The report outlines key phases including data familiarization, setup, exploration, cleaning, and validation, each contributing to the transformation of raw data into meaningful, actionable information. Throughout the internship, I undertook hands-on tasks that not only strengthened my technical proficiency in data preprocessing and analysis but also sharpened my problem-solving skills in dealing with complex data scenarios.

By documenting each stage of the process, this report aims to showcase the methodologies applied, the challenges encountered, and the lessons learned. Overall, the experience has significantly deepened my understanding of data analytics in a real-world context and equipped me with practical skills for future endeavors in the field.

CHAPTER 2: DATA FAMILIARIZATION

2.1 Learner DataSet:

1. DATA DICTIONARY OVERVIEW:

The dataset titled `Learner_Raw(in).csv` provides foundational learner profile information for users registered on an educational or e-learning platform. It contains 129,259 records and 5 primary columns: `learner_id`, `country`, `degree`, `institution`, and `major`. Each column represents a different aspect of the learner's background, primarily focusing on educational qualifications and geographic location.

The data structure is simple but valuable for tasks such as user segmentation, demographic profiling, and personalized learning analytics.

2. DATA SOURCE, UPDATE FREQUENCY AND LIMITATIONS:

- **Data Source:**

The dataset appears to originate from a user registration or profile form submitted by learners through a Learning Management System (LMS), website registration portal, or via bulk imports from partner academic institutions.

- **Update Frequency:**

Since most of the fields represent relatively static user information (such as degree, institution, and major), the dataset is likely updated only once during initial registration or manually by users when editing their profiles. This dataset is best classified as historical, not real-time. It represents a snapshot of user data collected over a period, rather than a continuously updating or streaming dataset.

- **Limitations:**

- ✓ High percentage of missing values in academic-related fields such as degree, institution, and major.
- ✓ The country field also contains some missing values, which limits the scope for geographic or regional analysis.
- ✓ The dataset lacks timestamp fields (e.g., registration date), which restricts time-based trend analysis or cohort tracking.

3. COLUMN MEANINGS AND DATA TYPES:

Column Name	Data Type	Description	Note
Learner_id	String	Unique identifier for each learner. Acts as a primary key.	Clear and consistent
Country	String	Country of origin or residence of the learner.	Valid, but may need normalization (e.g., USA vs United States)
Degree	String	Academic qualification obtained (e.g., Undergraduate, Graduate).	Clear and consistent
Institution	String	The name of the educational institution where the learner is enrolled or graduated from.	Needs cleaning – contains invalid, malformed, and lengthy sentence-like entries.
Major m	String	Field of study or specialization (e.g., Computer Science, Business)	Mostly valid – minor formatting inconsistencies (e.g., capitalization, duplicates).

4. KEY FIELDS AND RELATIONSHIPS:

- **Primary Key:**

- ✓ learner_id serves as the unique identifier for each record. It can be used as a primary key for indexing and joining with future datasets.

- **No Foreign Keys:**

- ✓ There are no foreign keys in this dataset, and it does not reference any external tables.

- **Future Relationships (Potential):**

- ✓ Learner_id can serve as a unique identifier to link with future datasets like course progress or performance data, while country can be mapped to external region-based datasets for geographic insights.
- ✓ Institution and major can support cohort analysis and segmentation based on academic background for deeper learner profiling.

5. INITIAL DATA SCAN:

- **Row and Column Check**

The dataset contains 129,259 rows and 5 columns, with properly defined headers in the first row.

- **Missing Values**

Several columns contain missing or blank entries:

- ✓ Columns like degree, institution, and major have missing or blank entries, which can impact academic profiling.
- ✓ Some country values are missing, which may limit the scope of regional-level analysis.

- **Duplicate Check**

A review of the learner_id column confirmed there are no duplicate values, establishing it as a valid unique identifier for each learner.

- **Outliers and Inconsistencies**

The dataset has no numerical outliers, but text fields like degree, institution, and major show formatting inconsistencies that need standardization.

2.2 Opportunity DataSet:

1 DATA DICTIONARY OVERVIEW:

The Opportunity Dataset signifies a list of scopes which can be a blessing to pursue a career. The opportunities in the datasets are categorized differently (e.g, event, internship, career etc). It is visible from the dataset that, there is a unique opportunity_id corresponding to each opportunity_name. To trace the progress of each opportunity holds a learner, there some tracking questions, relevant to that opportunity field.

2 DATA SOURCE, UPDATE FREQUENCY AND LIMITATIONS:

- **Source:**

I firmly believe that, the dataset belongs to a renowned organization which is aiming to open new doors of skills and cooperate with the learners to excel in global world.

- **Update Frequency:**

By having a glance on the dataset, it is noticeable that, this company believes in innovation. That's why, they are expanding their curriculum frequently. So, the dataset is also upgrading periodically.

- **Limitations:**

There is an unusual repetition of values in category, which may lead to Data Redundancy.

3 COLUMN MEANINGS AND DATA TYPES:

Column Name	Data Type	Description	Note
Opportunity_id	Text	Unique identifier	Valid
Opportunity_name	Varchar(255))		Valid
Category	Varchar(255))	Classification for opportunities	Valid
Opportunity_code	Text	Correspond to opportunity	Valid
Tracking_questions	Varchar(255))	Trace out the progress for definite opportunity	Need a conversion to text

4 KEY FIELDS AND RELATIONSHIPS:

- **Primary Key:**

- ✓ 'opportunity_id' is set on 'primary key'.

- **Potential Relationships:**

- ✓ opportunity_data can be linked up with other dataset. For Example – Learner_opportunity through 'opportunity_id'.

5 INITIAL DATA SCAN:

- **Data conversion needed:**

- Tracking_questions column needs a conversion from varchar(255) to text.

- **Column naming:**

- There is no need of column naming in the dataset.

- **Outliers:**

- There are no outliers.

- **Null values:**

- There are no null values in the dataset. Still it can be confirmed in cleaning process.

2.3 Cohort DataSet:

1. DATA DICTIONARY OVERVIEW:

The Cohort Dataset captures structured information about learner groups organized into distinct cohorts. Each cohort is uniquely identified by a `cohort_code` and includes details such as the cohort's start and end dates, as well as its size, representing the number of learners assigned. This dataset provides a temporal framework for analyzing learning cycles, enabling institutions or platforms to assess participation rates, cohort duration, and growth trends over time. The data is essential for tracking learner progress within specific timeframes, planning resource allocation, and aligning educational delivery with organizational timelines.

2. DATA SOURCE, UPDATE FREQUENCY AND LIMITATIONS:

- **Source:**

The data is likely sourced from an internal learner management or cohort tracking system, designed to record and monitor structured learner groupings over specific durations.

- **Update Frequency:**

The dataset appears to be updated periodically, potentially at the beginning or end of each academic term or learning cycle, aligning with cohort start and end dates.

- **Limitations:**

Some cohort sizes appear unusually large (e.g., 100,000 learners), which may suggest:

- ✓ Aggregated data across programs or terms
- ✓ Placeholder values for unfinalized cohorts

3. COLUMN MEANINGS AND DATA TYPES:

Column	Data Types	Description	Notes
Cohort_id	Object (string)	Internal cohort identifier	Valid
Cohort_date	Object (string)	Alphanumeric cohort code	Valid and consistent
Start_date	int64	Start date in UNIX ms timestamp	Needs conversion to datetime
End_date	int64	End date in UNIX ms timestamp	Needs conversion to datetime
size	int64	Size of the cohort (participants)	Some unusually large values (100,000)

4. KEY FIELDS AND RELATIONSHIPS:

- **Primary Key:**
 - ✓ The **cohort_code** column serves as the true unique identifier for each cohort, functioning effectively as the primary key.
 - ✓ The cohort_id field appears to hold a constant value ("Cohort#"), suggesting it is non-informative or deprecated.
- **Potential Relationships:**

The cohort_code is likely used to link with external datasets, such as:

- ✓ Learner Opportunity Data via a field assigned_cohort

5. INITIAL DATA SCAN:

- **Date Conversion Needed:**
start_date and end_date are UNIX timestamps (milliseconds) and need conversion.
- **Column Naming:**
"Cohort#" should be renamed to cohort_id for clarity.
- **Outliers in Size:**
Some cohorts have a size of 100,000, which may need validation.
- **No Null Values Noticed:**
At a glance, there appear to be no missing values, but this will be confirmed in detailed cleaning.

2.4 Marketing DataSet:

1. DATA DICTIONARY OVERVIEW:

The Marketing Campaign Dataset captures performance metrics for digital advertising campaigns across multiple accounts (e.g., SLU, RIT, Brand Awareness). It includes delivery status, reach, clicks, costs, and campaign outcomes (e.g., website leads, and applications submitted). This dataset enables campaign effectiveness, cost efficiency, and audience engagement analysis, supporting strategic decisions for budget allocation and campaign optimization.

2. DATA SOURCE, UPDATE FREQUENCY AND LIMITATIONS:

- **Source:**

Likely sourced from digital advertising platforms (e.g., Facebook Ads Manager, Google Ads) or internal marketing analytics tools.

- **Update Frequency:**

Data appears updated daily or in real-time, aligned with campaign reporting cycles.

- **Limitations:**

- ✓ Inconsistent naming conventions (e.g., "Power Course" vs. "Power Skill Course").
- ✓ Missing values in columns like `Reporting ends` (e.g., "").
- ✓ Unusual values in `Reach` (e.g., 141,835,342) and `Amount spent (AED)` (e.g., 25,531.47 AED), which may require validation.
- ✓ Mixed data types (e.g., numeric values in text columns like `Campaign name`).

3. COLUMN MEANINGS AND DATA TYPES:

Column	Data Types	Description	Notes
Ad Account Name	Object (string)	Name of the advertising account (e.g., SLU)	Valid
Campaign name	Object (string)	Name of the campaign	Inconsistent naming (e.g., duplicates)
Delivery status	Object (string)	Campaign status (e.g., active, completed)	Valid
Delivery level	Object (string)	The granularity of delivery (e.g., campaign)	Consistent
Reach	int64	Number of unique users exposed to the ad	Some extremely large values
Outbound clicks	int64	Clicks leading outside the platform	Valid
Landing page views	int64	Views of the post-click landing page	Valid
Result type	Object (string)	Type of campaign outcome (e.g., ThruPlay)	Inconsistent categorization
Results	int64	Total results (e.g., applications submitted)	Valid
Cost per result	float64	Cost per outcome (AED)	Requires outlier checks
Amount spent (AED)	float64	Total expenditure (AED)	Valid
CPC (cost per click)	float64	Cost per outbound click (AED)	Valid
Reporting starts	datetime64[ns]	Campaign start date	Requires format standardization
Reporting ends	datetime64[ns]	Campaign end date	Many missing values ("#####")

4. KEY FIELDS AND RELATIONSHIPS:

- **Primary Key:**

- ✓ A composite key of Ad Account Name + Campaign name + Reporting starts ensures uniqueness.

- **Potential Relationships:**

- ✓ Links to Learner Opportunity Data via `Result type` (e.g., "Website applications submitted").

- ✓ Connects to Budget Allocation Datasets using `Amount spent (AED)` and `Cost per result`.

5. INITIAL DATA SCAN:

- **Date Standardization Needed:**

- ✓ Reporting starts` and `Reporting ends` require conversion to a consistent date-time format.

- **Outliers:**

- ✓ Extreme values in `Reach` (e.g., 141,835,342) and `Amount spent (AED)` (e.g., 25,531.47) need validation.

- **Data Integrity:**

- ✓ Missing values in `Reporting ends` and inconsistent `Result type` categorization (e.g., "ThruPlay" vs. "Website leads").

- **Naming Conventions:**

- ✓ Campaign names include duplicates (e.g., "Virtual Internship" repeated across months). Rename for clarity.

2.5 Learning Opportunity DataSet:

1) DATA DICTIONARY OVERVIEW:

The dataset titled **learner_opportunity.csv** provides detailed records of learner enrollments into various academic or professional development opportunities. It contains information such as enrollment identifiers, learner identifiers, cohort assignments, application dates, and enrollment statuses. This dataset is essential for tracking student engagement in learning initiatives such as courses, internships, and training programs. It supports trend analysis, enrollment forecasting, and performance measurement across different learner cohorts and opportunity types.

2) DATA SOURCE, UPDATE FREQUENCY AND LIMITATIONS:

- **Data Source:**

This data appears to be extracted from systems such as a Learning Management System (LMS), internal CRM, or job/training portal that tracks student participation in opportunities.

- **Update Frequency:**

The dataset is updated periodically based on new enrollments. However, certain fields (like `apply_date` and `status`) may not always be updated consistently after initial entry.

- **Limitations:**

- ✓ Some **missing values** exist in fields like `apply_date` (~0.16%).
- ✓ The **status** column contains unclear or unstandardized codes, requiring external mapping or decoding.
- ✓ There may be **duplicate records** due to learners applying multiple times for the same opportunity.
- ✓ Datatype inconsistencies exist, particularly in `apply_date`.

3) COLUMN MEANINGS AND DATA TYPES:

Column Name	Data Type	Description	Note
enrollment_id	String	Unique identifier for each enrollment record	Serves as primary key
learner_id	String	Foreign key linking to the learner profile	Clean and consistent
assigned_cohort	String	Denotes the cohort (group) the learner is assigned to	No issues
apply_date	Varies	Date/time the learner applied for the opportunity	Missing values and datatype issues
status	String	Enrollment status (e.g., applied, approved, rejected)	Status codes unclear

4) KEY FIELDS AND RELATIONSHIPS:

- **Primary Key:**

- ✓ enrollment_id uniquely identifies each enrollment record.

- **Foreign Keys:**

- ✓ learner_id links this dataset with the **Learner Data**, allowing integration and detailed profiling.
- ✓ assigned_cohort aligns with the **Cohort Data**, enabling cohort-based analysis.

- **Potential Relationships:**

- ✓ Linking with **Opportunity** data (if available) could further enrich the analysis.
- ✓ apply_date can be used for **temporal analysis**, e.g., tracking enrollment patterns over time.

5) INITIAL DATA SCAN:

- **Row and Column Check:**

- ✓ The dataset appears well-structured with properly named columns and consistent schema.

- **Missing Values:**

- ✓ apply_date has a small percentage of missing entries (~0.16%).
- ✓ status also shows inconsistencies or undefined values.

- **Duplicate Check:**

- ✓ Initial scans indicate **potential duplicate enrollments** based on learners reapplying for the same opportunity. These need validation using a combination of learner_id, assigned_cohort, and status.

- **Outliers and Inconsistencies:**

- ✓ apply_date has datatype issues, possibly stored as numeric or inconsistent timestamp formats.
- ✓ status values require standardization and mapping to meaningful descriptions (e.g., 0 = pending, 1 = approved).

2.6 Cognito DataSet:

1. DATA DICTIONARY OVERVIEW:

The Cognito dataset captures essential user profile information collected through a user authentication and identity system. Key columns include `userid`, a unique identifier for each user; `UserCreateDate` and `UserLastModifiedDate`, which indicate when the user profile was initially created and last updated, respectively. `email` stores the user's registered email address, serving as a primary contact field. Demographic details such as gender, birthdate, city, zip, and state provide insights into user segmentation and geographic distribution. This structured data supports user management, segmentation analysis, and downstream analytics for personalized experiences and compliance reporting.

2. DATA SOURCE, UPDATE FREQUENCY AND LIMITATIONS:

- **Source:**

This data likely originates from a user registration and authentication system powered by Amazon Cognito. It is collected when users sign up, update their profiles, or log in to a web or mobile application. The data is typically stored in a secure user pool managed within a cloud-based environment.

- **Update Frequency:**

The update frequency of this data depends on user activity within the application. Fields like modified date may update in real time or near real time whenever a user updates their profile or logs in. Less dynamic fields, such as birthday or gender, are typically updated infrequently unless the user chooses to make changes.

- **Limitations:**

This dataset may have incomplete or outdated user information if users do not update their profiles regularly. Some fields, like gender or city zip, may be self-reported and prone to errors or inconsistencies. Additionally, the dataset may not capture real-time behavioral or transactional data beyond identity-related attributes.

3. COLUMN MEANINGS AND DATA TYPES:

Column	Data Types	Description	Notes
user_id	Object (string)	Unique id for every user	Valid
email	Object (string)	Email id of user	Valid
gender	Object(string)	Gender of the user	Contains NULL values
UserCreateDate	datetime	Date and time when users profile was created in the system	valid
UserLastModifiedDate	datetime	Date and time when users last modified the profile	valid
birthdate	date	Date of birth of the user	Contain Null values and inconsistent data type
city	Object(string)	City where the user based	Contains Null Values and inconsistent naming
zip	Object(string)	Zipcode of the city	Contains Null Values
state	Object(string)	State in which the city is located	Contain Null values and inconsistent naming

4. KEY FIELDS AND RELATIONSHIPS:

- **Primary Key:**

- ✓ The user_id column serves as the true unique identifier for each user, functioning effectively as the primary key.

- **Potential Relationships:**

- The user_id is likely used to link with external datasets, such as:

- ✓ Learner Opportunity Data via a field enrollment_id.
 - ✓ Learner_Raw Data via a field learner_id.

5. INITIAL DATA SCAN:

- **Date Conversion Needed:** birthdate need to converted to a consistent date style.
- **Naming consistency:** city and state name requires naming validation.
- **Null Values Noticed:** Null values need to removed from gender ,city ,zip ,state and birthdate.

CHAPTER 3: DATA SETUP

3.1 Leaner DataSet

1. CREATING DATABASE:

To begin organizing and analyzing the learner dataset, a new database named Learner_Data was created in pgAdmin, which is the graphical user interface for PostgreSQL.

Steps followed:

- Open pgAdmin.
- Right-click on Databases → Click Create → Database.
- Enter the name Learner_Data and click Save.

2. CREATING TABLE:

Source Code:

```
create table Learner(  
    leaener_id Text,  
    degree Text,  
    major Text,  
    institution Text,  
    country Integer  
);
```

3. IMPORTING DATA:

After defining the table structure, the learner dataset from a CSV file was imported into the "Learner" table using pgAdmin's import feature.

Steps followed:

- Right-click on the "Learner" table → Select Import/Export.
- Under the Import tab, apply the following settings:
 - ✓ Format: CSV
 - ✓ File path: Select the CSV file location

✓ Encoding: UTF-8

- Click OK to begin importing.

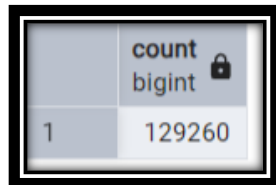
4. VERIFYING THE IMPORTED DATA:

- Ensuring Whole Table Is Imported:

Source Code:

```
select count(*) from public.'Learner';
```

Output:



	count bigint
1	129260

- Displaying The Table:

Source Code:

```
SELECT * FROM public."Learner " LIMIT 10;
```

Output:



	Learner_ID text	Country text	Degree text	Institution text	Major text
1	Learner_ID	Country	Degree	Institution	Major
2	Learner#00004f18-8b86-4fe4-ad7e-6c8d988f53...	Nigeria	Undergraduate Student	Federal University of Technology Owerri	Civil Engineering
3	Learner#00006478-745f-49bf-b126-02584e830...	Nigeria	NULL	NULL	NULL
4	Learner#00010567-1336-433c-a941-a612b3d2f...	Kenya	Graduate Student	UNICAF UNIVERSITY	Environmental Sustainability
5	Learner#00011c80-0c5c-4601-9696-b3ca787e2...	Bangladesh	NULL	NULL	NULL
6	Learner#000141a7-4c82-401f-a2e6-dd12b4b26...	Nigeria	NULL	NULL	NULL
7	Learner#0acf3501-57a8-4585-91e3-6bbb89d3c...	Ghana	NULL	NULL	NULL
8	Learner#0001ca2c-7bec-4a33-833c-b844a29f4...	Nigeria	Graduate Student	Nasarawa State University, Keffi	Accounting
9	Learner#0003bed9-d9d9-49a7-a755-a9562aaa0...	Pakistan	Graduate Student	CTTI College KP Campus	Part Time
10	Learner#0004295c-717e-4953-b3bf-fff2daf0e903	Nigeria	Undergraduate Student	Caleb University	Computer Science

3.2 Opportunity DataSet

1) CREATING TABLE:

Source Code:

```
create table opportunity(  
    opportunity_id text not null primary key,  
    opportunity_name varchar(255),  
    category varchar(255),  
    opportunity_code text,  
    tracking_questions carchar(255)  
  
);
```

2) IMPORTING DATA:

Source Code:

```
copy oppurtunity(oppurtunity_id, oppurtunity_name, category, oppurtunity_code,  
tracking_questions)  
from 'F:\annual_testing\data_analyst\excelerate\Opportunity_Data.csv'  
delimiter ','  
csv header;  
  
alter table oppurtunity  
alter column tracking_questions type text;
```

3) VERIFYING THE IMPORTED DATA:

- Ensuring Whole Table Is Imported:

Source Code:

```
select count(*) from opportunity;
```

Output:



	count bigint
1	187

- **Displaying The Table:**

Source Code:

```
select * from opportunity where Category = 'Event';
```

Output:

	opportunity_id (PK) text	opportunity_name character varying (255)	category character varying (255)	opportunity_code text	tracking_questions text
1	Opportunity#000000000G127E8VYE08TXBT6X	Choosing and Planning for Your Major	Event	E501873	NULL
2	Opportunity#000000000G4F19XBEXPWKS8F3N	Statement of Purpose (SOP) Writing Workshop	Event	E258709	NULL
3	Opportunity#000000000GCKFV5K6Q8FWGFH...	Choosing and Planning for Your Major + Career Exploration Workshops ? In-person	Event	E189319	NULL
4	Opportunity#000000000GCTJ4F7QXJWWMBD...	Major and Career Exploration Workshop	Event	E352968	NULL
5	Opportunity#000000000GEHAHYHGRSY59TR1D	Building a Strong Application in a Test-Optional World	Event	E322161	NULL
6	Opportunity#000000000GG3B9VDBKAQM1D9...	Teaching During The Time Of Disruption Test	Event	E289840	NULL
7	Opportunity#000000000GKZ7RSWWZA14JDC...	Major and Career Exploration Workshop ? Virtual	Event	E296277	NULL
8	Opportunity#00000000104G3FS63SV3JKHVQB	Mental and Physical Health Session	Event	E2F0VCY	NULL

3.3 Cohort DataSet

1) CREATING TABLE:

Source Code:

```
create table cohort_data(  
    cohort_id Text,  
    cohort_code Text,  
    start_date Text,  
    end_date Text,  
    size Integer  
);
```

2) IMPORTING DATA:

Source Code:

```
COPY public.cohort_data FROM 'D:\INTERSHIPS\CohortRaw.csv'  
WITH CSV HEADER DELIMITER ',' NULL 'NULL' ENCODING 'ISO-8859-1';
```

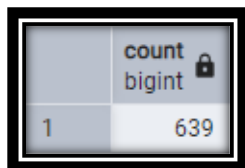
3) VERIFYING THE IMPORTED DATA:

- Ensuring Whole Table Is Imported:

Source Code:

```
select count(*) from cohort_data;
```

Output:



count	
bigint	
1	639

- **Displaying The Table:**

Source Code:

```
select * from cohort_data limit 10;
```

Output:

	cohort_id text	cohort_code text	start_date text	end_date text	size 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

3.4 Marketing DataSet

1) CREATING TABLE:

Source Code:

```
create table marketing_campaign_data
( ad_account_name Text,
  campaign_name Text,
  delivery_status Text,
  delivery_level Text,
  reach Integer,
  outbound_clicks Integer,
  landing_page_views Integer,
  result_type Text,
  results Integer,
  cost_per_result Numeric,
  amount_spent_aed Numeric,
  cpc Numeric,
  reporting_starts Date,
  rpc Numeric
);
```

2) Importing Data:

Source Code:

```
COPY public.marketing_campaign_data FROM
'D:\INTERNSHIPS\MarketingCampaignRaw.csv' WITH CSV HEADER DELIMITER
',' NULL 'NULL' ENCODING 'ISO-8859-1';
```

3) Verifying The Imported Data:

- Ensuring Whole Table Is Imported:

Source Code:

```
select count(*) from marketing_campaign_data;
```

- **Displaying The Table:**

Source Code:

```
select * from marketing_campaign_data limit 10;
```

Output:

	id_account_name text	campaign_name text	delivery_status text	delivery_level text	reach integer	outbound_clicks integer	landing_page_views integer	result_type text	results integer	cost_per_result numeric	amount_spent_and numeric	cpc numeric	reporting_start_date date	cpe numeric
1	SLU	#ee82: Digital Marketing Intern - May Ads: Website Leads Prospecting 18 to 35 years - Copy 4	completed	campaign	102962	1815	1310	Website applications submitt...	386	1.910395	737.63	0.406184	2023-01-01	[null]
2	SLU	#ee82: Digital Marketing Intern - May Ads: Website Leads Prospecting 18 to 35 years - Copy 3	completed	campaign	180173	3378	2152	Website applications submitt...	598	1.233495	737.63	0.217979	2023-01-01	1.233494983
3	SLU	#ee82: Digital Marketing Intern - May Ads: Website Leads Prospecting 18 to 35 years - Copy 3	inactive	campaign	173118	3591	2767	Website applications submitt...	514	1.743914	897.4	0.248489	2023-01-01	1.743914397
4	SLU	#Brand Awareness: UGC Video - March - Copy	inactive	campaign	18355415	5431	1019	Reach	18355415	0.062997	1707	0.309744	2023-01-01	0.0000929971
5	SLU	#Data Analyst Associate Internship	inactive	campaign	2448	111	62	Website leads	7	5.43	5.43	0.548919	2023-01-01	5.43
6	SLU	A1: Outreach Consultant Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	inactive	campaign	1136239	12796	8199	Website leads	2515	1.179162	3447.69	0.209208	2023-01-01	1.175482441
7	SLU	A2: Digital Strategy Associate Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	inactive	campaign	882331	8910	6912	Website leads	2643	1.142303	3425.87	0.259192	2023-01-01	1.142303247
8	SLU	A3: Data Analyst Associate Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	inactive	campaign	1077778	21464	16914	Website leads	6025	0.436508	3556.19	0.126368	2023-01-01	0.436508611
9	SLU	A4: Project Management Associate Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy...	inactive	campaign	934611	13080	8995	Website leads	4540	0.749714	3403.7	0.260281	2023-01-01	0.749713956
10	SLU	A5: Business Strategy Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	inactive	campaign	999159	13337	9112	Website leads	4113	0.905715	3729.32	0.279027	2023-01-01	0.905715293

3.5 Learning Opportunity DataSet:

1. CREATING DATABASE:

Source Code:

```
create table learning_opp_1(  
  enrollment_id text,  
  learner_id text,  
  assigned_cohort text,  
  apply_date timestamp,  
  status int  
);
```

2. IMPORTING DATABASE:

Source Code:

```
copy learning_opp_1(enrollment_id,  
  learner_id,assigned_cohort,apply_date,status) from 'F:\manual  
testing\data_analyst\exceletrate\cleaned_learner_opportunity_1.csv'  
delimiter ','  
csv header;
```

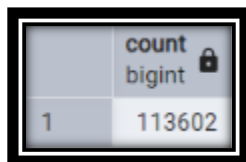
3. VERIFYING THE IMPORTED DATABASE:

- Ensuring Whole Table Is Imported:

Source Code:

```
select count(*) from learning_opp_1;
```

Output:



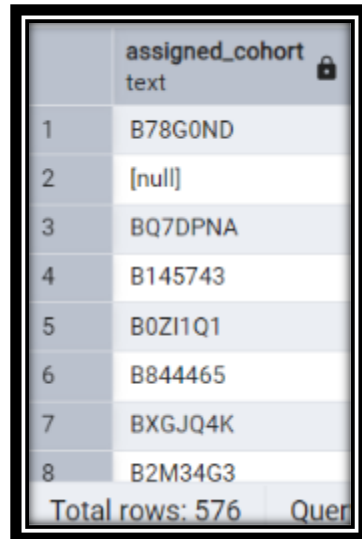
count	bigint
1	113602

- **Displaying The Table:**

Source Code:

```
select assigned_cohort from learning_opp_1 group by  
assigned_cohort;
```

Output:



The screenshot shows a table with two columns: an index and 'assigned_cohort'. The 'assigned_cohort' column is of type 'text' and has a lock icon. The table contains 8 rows of data, with the last row being a summary row. The data is as follows:

	assigned_cohort text
1	B78G0ND
2	[null]
3	BQ7DPNA
4	B145743
5	B0ZI1Q1
6	B844465
7	BXGJQ4K
8	B2M34G3
Total rows: 576 Quer	

3.6 Cognito DataSet

1) CREATING TABLE:

Source Code:

```
create table cognito_raw(  
    user_id Text,  
    email Text,  
    gender Text,  
    user_create_date Text,  
    user_modified_date Text,  
    birthdate Text,  
    city Text,  
    zip Text,  
    states Text  
);
```

2) IMPORTING DATA:

Source Code:

```
COPY public.cognito_raw FROM 'D:\INTERNSHIPS\CognitoRaw.csv'  
WITH CSV HEADER DELIMITER ',' NULL 'NULL' ENCODING 'ISO-8859-1';
```

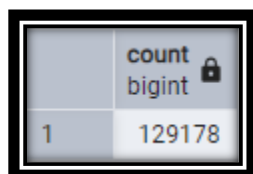
4) VERIFYING THE IMPORTED DATA:

- Ensuring Whole Table Is Imported:

Source Code:

```
select count(*) from cognito_raw;
```

Output:



	count bigint
1	129178

- **Displaying The Table:**

Source Code:

```
select * from cognito_raw limit 10;
```

Output:

	user_id text	email text	gender text	user_create_date text	user_modified_date text	birthdate text	city text	zip text	states text
1	00010567-1336-433c-a941-a612b3d2f...	gikonyosalome19@gmail.com	Female	2024-11-17T21:25:56.381Z	2024-11-17T21:32:50.783Z	5/4/1996	NAIVASHA	20117	NAKURU
2	aab8bd87-af83-4e21-816b-101cf05f9a79	evelyn.natasha.guo@gmail.com	[null]	2025-01-19T02:07:06.113Z	2025-01-19T02:07:20.726Z	[null]	[null]	[null]	[null]
3	4656095f-a932-4889-ae96-3b77ff60f1e4	lauren.singh@rocketmail.com	Female	2024-03-26T23:23:54.329Z	2024-09-27T13:47:51.806Z	4/5/1990	Queens Village	11428	NY
4	76b5629f-a024-4de8-9f10-59ebf8fd019b	anilhmercy2019@gmail.com	Female	2024-03-31T19:04:21.735Z	2024-09-27T16:12:28.564Z	12/28/1998	Ibadan	200221	Oyo
5	db17206b-2017-4b6a-9462-fc2bc7fdfb...	lagrimasamie@gmail.com	Female	2024-03-25T20:36:26.352Z	2024-04-08T16:10:24.503Z	5/5/1999	Malolos City	3000	Bulacan
6	78de6832-deef-4dab-b7ef-d953aee7e7...	kolayinka777@gmail.com	[null]	2023-06-16T15:19:21.404Z	2023-06-16T15:20:06.170Z	[null]	[null]	[null]	[null]
7	2444d3b7-3204-4b66-a1e2-72172db26...	ujjwal.pandey2103@gmail.com	Male	2024-05-21T17:58:57.614Z	2024-09-27T16:04:21.994Z	3/21/2000	New Delhi	110045	Delhi
8	fec90f6c-de9e-4594-9248-4a5d53ff5a7e	amaliataabazuig@gmail.com	Female	2023-07-05T22:25:14.656Z	2024-09-08T08:55:42.155Z	4/22/2002	Kumasi	233	Ashanti Region
9	9a4fea59-426a-42e8-ab0d-ed9c0adc61...	230888@d230.org	[null]	2023-01-05T16:33:12.709Z	2023-01-05T19:32:14.802Z	[null]	[null]	[null]	[null]
10	57f7680d-968a-40fd-a1c5-796aed4440f7	vonyedika35@gmail.com	[null]	2023-06-22T12:56:37.433Z	2023-06-22T12:57:57.221Z	[null]	[null]	[null]	[null]

CHAPTER 4: EXPLORATORY DATA ANALYSIS (EDA)

4.1 Leaner DataSet

1) UNDERSTANDING THE DATASET (FIXING COLUMN TYPES):

- **Checking Data Types:**

Source Code:

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

Output:

	column_name name	data_type character varying
1	Learner_ID	text
2	Country	text
3	Degree	text
4	Institution	text
5	Major	text

2) SUMMARY STATISTICS:

- **Mode:**
 - ✓ **Most Common Country:**

Source Code:

```
SELECT "Country", COUNT(*) AS frequency  
FROM public."Learner"  
GROUP BY "Country"  
ORDER BY frequency DESC  
LIMIT 1;
```


Output:

	Country text	frequency bigint
1	India	33868

- ✓ **Most Common Degree:**

Source Code:

```
SELECT "Degree", COUNT(*) AS frequency
FROM public."Learner"
GROUP BY "Degree"
ORDER BY frequency DESC
LIMIT 1;
```

Output:

	Degree text	frequency bigint
1	NULL	52693

- **Category Counts:**

- ✓ **Top 10 Countries by Number of Learners:**

Source Code:

```
SELECT "Country", COUNT(*) AS total_learners
FROM public."Learner"
GROUP BY "Country"
ORDER BY total_learners DESC
LIMIT 10;
```

Output:

	Country text	total_learners bigint
1	India	33868
2	Nigeria	30696
3	Pakistan	14112
4	Kenya	8246
5	United States	7064
6	Ghana	6874
7	Philippines	6039
8	Egypt	5301
9	South Africa	4587
10	Bangladesh	3525

✓ **Top 10 Degrees by Number of Learners:**

Source Code:

```
SELECT "Degree", COUNT(*) AS total_learners
FROM public."Learner"
GROUP BY "Degree"
ORDER BY total_learners DESC
LIMIT 10;
```

Output:

1	NULL	52693
2	Graduate Student	31806
3	Undergraduate Student	30709
4	Not in Education	6319
5	High School Student	4109
6	Other Professional	2997
7	Teacher/Educator	562
8	Parent of Student	64
9	Degree	1

3) MISSING VALUES ANALYSIS

- Column With Missing Values:

Source Code:

```
SELECT  
SUM(CASE WHEN "Country" IS NULL THEN 1 ELSE 0 END) AS missing_country,  
SUM(CASE WHEN "Degree" IS NULL THEN 1 ELSE 0 END) AS missing_degree,  
SUM(CASE WHEN "Institution " IS NULL THEN 1 ELSE 0 END) AS missing_institution,  
SUM(CASE WHEN "Major" IS NULL THEN 1 ELSE 0 END) AS missing_major  
FROM public."Learner";
```

Output:

	missing_country bigint	missing_degree bigint	missing_institution bigint	missing_major bigint
1	0	0	0	0

- Percentage Of Missing Values:

Source Code:

```
SELECT  
ROUND(100.0 * SUM(CASE WHEN "Country" IS NULL THEN 1 ELSE 0 END) /  
COUNT(*), 2) AS percent_missing_country,  
ROUND(100.0 * SUM(CASE WHEN "Degree" IS NULL THEN 1 ELSE 0 END) /  
COUNT(*), 2) AS percent_missing_degree,  
ROUND(100.0 * SUM(CASE WHEN "Institution " IS NULL THEN 1 ELSE 0 END) /  
COUNT(*), 2) AS percent_missing_institution,  
ROUND(100.0 * SUM(CASE WHEN "Major" IS NULL THEN 1 ELSE 0 END) /  
COUNT(*), 2) AS percent_missing_major  
FROM public."Learner";
```

Output:

	percent_missing_country numeric	percent_missing_degree numeric	percent_missing_institution numeric	percent_missing_major numeric
1	0.00	0.00	0.00	0.00

4) Duplicate Records

- **Count Of Duplicate Records**

Source Code:

```
SELECT COUNT(*) - COUNT(DISTINCT ROW("Learner_ID",  
"Country", "Degree", "Institution", "Major")) AS  
duplicate_rows  
FROM public."Learner";
```

Output:



	duplicate_rows
1	0

5) Spot Outliers And Anomilies:

- **Not Applicable:**

The dataset contains only categorical variables like Country, Degree, Institution, and Major. Since outlier detection methods like IQR and Z-score require numerical data, they are not applicable here. No major anomalies were found in the categorical values.

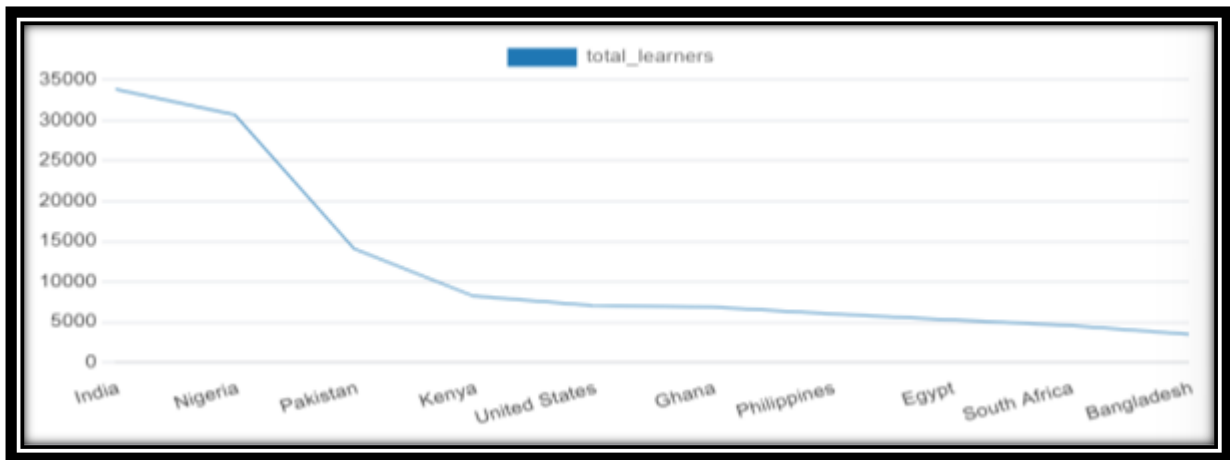
6) VISUALIZATIONS:

- Line Chart – Top 10 Countries by Learner Count

Source Code:

```
SELECT "Country", COUNT(*) AS total_learners
FROM public."Learner"
GROUP BY "Country"
ORDER BY total_learners DESC
LIMIT 10;
```

Output:

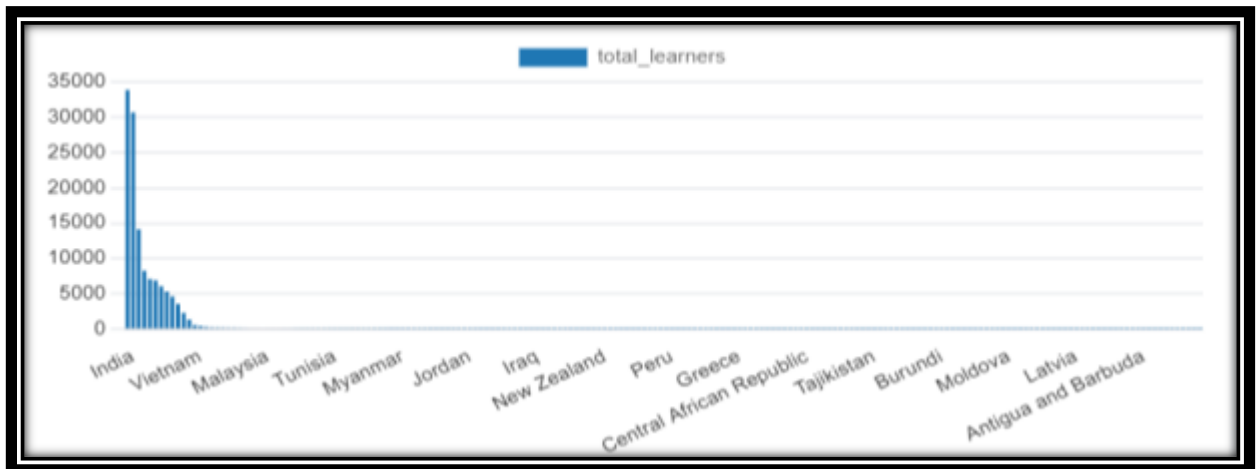


- **Bar Chart – Learners per Country**

Source Code:

```
SELECT "Country", COUNT(*) AS total_learners  
FROM public."Learner"  
GROUP BY "Country"  
ORDER BY total_learners DESC;
```

Output:

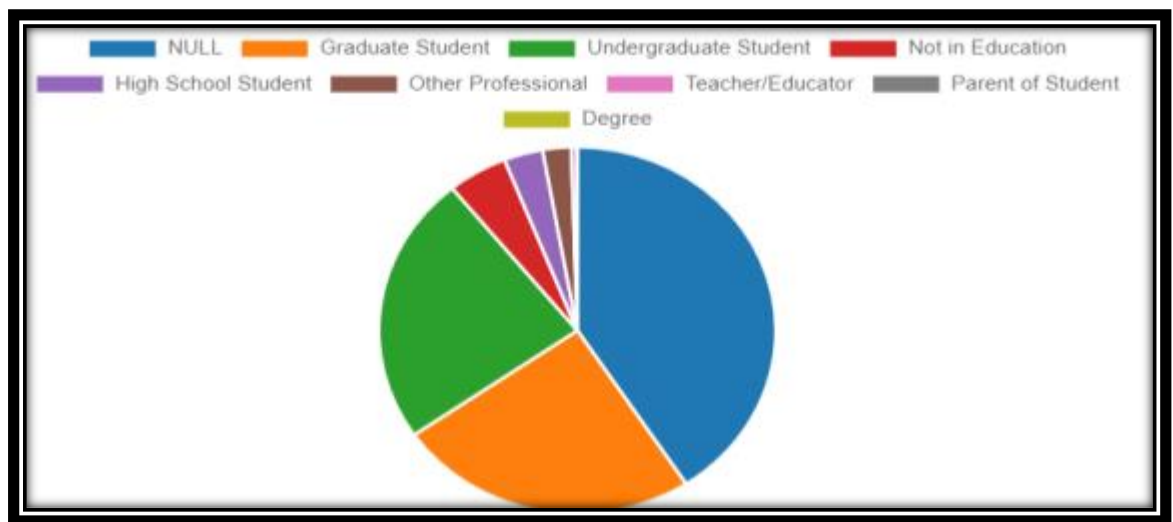


- **Pie Chart – Learners by Degree**

Source Code:

```
SELECT "Degree", COUNT(*) AS total_learners
FROM public."Learner"
GROUP BY "Degree"
ORDER BY total_learners DESC;
```

Output:



7) KEY FINDINGS:

- **Total Records:** 129,259 learners in the dataset.
- **Data Structure:** 5 columns — all are categorical (Country, Degree, Institution, Major, Learner_ID).
- **Missing Data:** 0% missing values in Degree, Institution, and Major fields.
- **Duplicates:** No duplicate records found — each entry is unique.
- **Top Countries:** India and the United States have the highest number of learners.
- **Top Degrees:** Most learners are enrolled as Graduate or Undergraduate students.
- **Outliers:** Not applicable due to absence of numerical fields.

4.2 Opportunity DataSet

1. UNDERSTANDING THE DATASET (FIXING COLUMN TYPES):

- For tracking_questions:

Source Code:

```
alter table oppurtunity  
alter column tracking_questions type text;
```

Explanation:

- ✓ Datatype has been changed in 'tracking_questions' column from varchar(255) to text, as the character numbers are exceeded.

- Final Data Types:

Source Code:

```
Select column_name,data type  
FROM information_schema.columns  
WHERE table_name = 'oppurtunity'
```

Output:

	column_name name	data_type character varying
1	oppurtunity_id	text
2	oppurtunity_name	character varying
3	category	character varying
4	oppurtunity_code	text
5	tracking_questions	text

2. SUMMARY STATISTICS:

- Counts for Categorical Variables:

- ✓ Count for 'Category':

Source Code:

```
select category,count(*) from oppurtunity  
group by category;
```

Output:

	category character varying (255)	count bigint
1	Competition	41
2	Career	23
3	Masterclass	11
4	Course	18
5	Internship	43
6	Event	41
7	Engagement	10

- ✓ Count for 'Tracking_question':

Source Code:

```
select tracking_questions, count(*) from oppurtunity group  
by tracking_questions;
```

Output:



The screenshot shows a SQL query editor with a complex query for tracking_questions. The query is as follows:

```
select tracking_questions, count(*) from oppurtunity group  
by tracking_questions;
```

- ✓ Count for 'opportunity_name':

Source Code:

```
select opportunity_name,count(*) from opportunity group
by opportunity_name;
```

Output:

	opportunity_name character varying (255)	count bigint
1	Talent Management Associate	2
2	Tulip Application Development Internship	1
3	Diversity, Equity and Inclusion Workshop	1
4	Data Visualization	4
5	Begin Your Job Search	1
6	UX Redesign Challenge	1
7	Entrepreneurship and Innovation	2
8	Lens Masters: A Photography Contest	1
9	CPR/AED Certification	1

3. MISSING VALUES ANALYSIS:

- Count of missing values:

Source Code:

```
SELECT
COUNT(CASE WHEN opportunity_id IS NULL THEN 1 END) AS opportunity_id_null_count,
COUNT(CASE WHEN opportunity_name IS NULL THEN 1 END) AS opportunity_name_null_count,
COUNT(CASE WHEN category IS NULL THEN 1 END) AS category_null_count,
COUNT(CASE WHEN category IS NULL THEN 1 END) AS opportunity_code_null_count,
COUNT(CASE WHEN category IS NULL THEN 1 END) AS tracking_questions_null_count
FROM opportunity;
```

Output:

	opportunity_id_null_count bigint	opportunity_name_null_count bigint	category_null_count bigint	opportunity_code_null_count bigint	tracking_questions_null_count bigint
1	0	0	0	0	0

- **Percentage of Missing Data:**

Source Code:

```
SELECT
COUNT(*) AS total_rows,
COUNT(oppurtunity_id )AS non_null_rows,
(COUNT(*) - COUNT(oppurtunity_id))::DECIMAL / COUNT(*) * 100 AS
missing_percentage
FROM oppurtunity;
```

```
SELECT
COUNT(*) AS total_rows,
COUNT( oppurtunity_name )AS non_null_rows,
(COUNT(*) - COUNT(oppurtunity_name))::DECIMAL / COUNT(*) * 100 AS
missing_percentage
FROM oppurtunity;
```

```
SELECT
COUNT(*) AS total_rows,
COUNT(category )AS non_null_rows,
(COUNT(*) - COUNT(category))::DECIMAL / COUNT(*) * 100 AS
missing_percentage
FROM oppurtunity;
```

```
SELECT
COUNT(*) AS total_rows,
COUNT(oppurtunity_code)AS non_null_rows,
(COUNT(*) - COUNT(oppurtunity_code))::DECIMAL / COUNT(*) * 100 AS
missing_percentage
FROM oppurtunity;
```

```
SELECT
COUNT(*) AS total_rows,
COUNT(tracking_questions)AS non_null_rows,
(COUNT(*) - COUNT(tracking_questions))::DECIMAL / COUNT(*) * 100 AS
missing_percentage
FROM oppurtunity;
```

Output:

	total_rows bigint	non_null_rows bigint	missing_percentage numeric
1	187	187	0.00000000000000000000

4. DUPLICATE RECORDS:

- **Count Of Duplicate Records:**

Source Code:

```
SELECT COUNT(*) AS duplicate_count
FROM oppurtunity
GROUP BY
oppurtunity_id,oppurtunity_name,category,oppurtunity_code,tracking_
questions
HAVING COUNT(*) > 1;
```

Output:

duplicate_count bigint

5. SPOT OUTLIERS AND ANOMILIES:

- No outliers

6. VISUALIZATIONS:

- **Total number of opportunities under per category:**

Source Code:

```
select category,count(oppurtunity_name) as total
from oppurtunity
group by category;
```

Output:

	category character varying (255) 🔒	total bigint 🔒
1	Competition	41
2	Career	23
3	Masterclass	11
4	Course	18
5	Internship	43
6	Event	41
7	Engagement	10

7. KEY FINDINGS:

- There are total 187 data.
- No outliers has been detected till now.

4.3 Cohort DataSet

1. UNDERSTANDING THE DATASET (FIXING COLUMN TYPES):

- For cohort_code:

Source Code:

```
ALTER TABLE cohort_data  
ALTER COLUMN cohort_code TYPE VARCHAR(20);  
SELECT cohort_code, COUNT(*) AS occurrences  
FROM cohort_data  
GROUP BY cohort_code  
HAVING COUNT(*) > 1;  
ALTER TABLE cohort_data  
ADD PRIMARY KEY (cohort_code);  
ALTER TABLE cohort_data  
DROP COLUMN cohort_id;
```

Explanation:

- ✓ The cohort_code contains Short alphanumeric identifier so keeping its datatype VARCHAR(20).
- ✓ As the cohort_id has no content, we check if the cohort_code is unique so we could set it as a primary key.
- ✓ As the cohort_code was unique, we set it as the primary key and drop the cohort_id column.

- For Size:

Source Code:

```
ALTER TABLE cohort_data  
ALTER COLUMN size TYPE INTEGER USING size::INTEGER;
```

Explanation:

- ✓ Converted size to INTEGER since it represents whole numbers only.

- For start_date and end_date:

Source Code:

```
UPDATE cohort_data
SET start_date = CAST(start_date AS NUMERIC),
end_date = CAST(end_date AS NUMERIC);
ALTER TABLE cohort_data
ALTER COLUMN start_date TYPE BIGINT USING start_date::BIGINT,
ALTER COLUMN end_date TYPE BIGINT USING end_date::BIGINT;
ALTER TABLE cohort_data
ALTER COLUMN start_date TYPE TIMESTAMP USING TO_TIMESTAMP(start_date /
1000.0),
ALTER COLUMN end_date TYPE TIMESTAMP USING TO_TIMESTAMP(end_date /
1000.0);
```

Explanation:

- ✓ start_date and end_date were in exponential format, so they were first cast to NUMERIC, then to BIGINT.
- ✓ Finally, they were converted to TIMESTAMP using TO_TIMESTAMP() for proper date formatting.

- Final Data Types:

Source Code:

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

Output:

	column_name name	data_type character varying
1	start_date	timestamp without time zone
2	end_date	timestamp without time zone
3	size	integer
4	cohort_code	character varying

2. SUMMARY STATISTICS:

- Mean

Source Code:

```
SELECT AVG(size) AS mean_size  
FROM cohort_data;
```

Output:

	mean_size	
	numeric	🔒
1	5741.4241001564945227	

- Median

Source Code:

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

Output:

	median_size	
	double precision	🔒
1	800	

- Mode

Source Code:

```
SELECT size, COUNT(*) AS frequency  
FROM cohort_data  
GROUP BY size  
ORDER BY frequency DESC  
LIMIT 1;
```


Output:

	size integer	frequency bigint
1	800	171

- **Min, Max, Range**

Source Code:

```
SELECT
  MIN(size) AS min_size,
  MAX(size) AS max_size,
  MAX(size) - MIN(size) AS range_size
FROM cohort_data;
```

Output:

	min_size integer	max_size integer	range_size integer
1	3	100000	99997

- **Standard Deviation**

Source Code:

```
SELECT
  STDDEV(size) AS stddev_size
FROM cohort_data;
```

Output:

	stddev_size numeric
1	20994.26661203

- **Counts for categorical variables**

Source Code:

```
SELECT
COUNT(DISTINCT cohort_code) AS unique_cohort_codes,
MIN(start_date) AS earliest_start_date,
MAX(end_date) AS latest_end_date
FROM cohort_data;
```

Output:

	unique_cohort_codes bigint	earliest_start_date timestamp without time zone	latest_end_date timestamp without time zone
1	639	2022-06-09 12:33:20	2026-03-06 17:26:40

- **Cohort Size Distribution:**

Source Code:

```
SELECT size, COUNT(*) AS count
FROM cohort_data
GROUP BY size
ORDER BY count DESC
LIMIT 10;
```

Output:

	size integer	count bigint
1	800	171
2	1500	88
3	300	67
4	1000	55
5	500	43
6	1600	39
7	200	34
8	100000	30
9	100	22
10	10000	17

3. MISSING VALUES ANALYSIS

- Count Of Missing Values:

Source Code:

```
SELECT
SUM(CASE WHEN cohort_code IS NULL THEN 1 ELSE 0 END) AS missing_cohort_code,
SUM(CASE WHEN start_date IS NULL THEN 1 ELSE 0 END) AS missing_start_date,
SUM(CASE WHEN end_date IS NULL THEN 1 ELSE 0 END) AS missing_end_date,
SUM(CASE WHEN size IS NULL THEN 1 ELSE 0 END) AS missing_size
FROM cohort_data;
```

Output:

	missing_cohort_code bigint	missing_start_date bigint	missing_end_date bigint	missing_size bigint
1	0	0	0	0

- Percentage Of Missing Data

Source Code:

```
SELECT
ROUND(100.0 * SUM(CASE WHEN cohort_code IS NULL THEN 1 ELSE 0 END) / COUNT(*), 2)
AS pct_missing_cohort_code,
ROUND(100.0 * SUM(CASE WHEN start_date IS NULL THEN 1 ELSE 0 END) / COUNT(*), 2)
AS pct_missing_start_date,
ROUND(100.0 * SUM(CASE WHEN end_date IS NULL THEN 1 ELSE 0 END) / COUNT(*), 2) AS
pct_missing_end_date,
ROUND(100.0 * SUM(CASE WHEN size IS NULL THEN 1 ELSE 0 END) / COUNT(*), 2) AS
pct_missing_size
FROM cohort_data;
```

Output:

	pct_missing_cohort_code numeric	pct_missing_start_date numeric	pct_missing_end_date numeric	pct_missing_size numeric
1	0.00	0.00	0.00	0.00

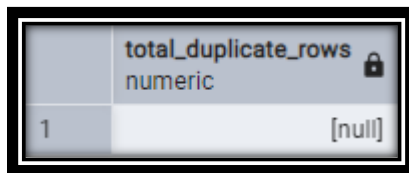
4. Duplicate Records

- Count Of Duplicate Records

Source Code:

```
SELECT
  SUM(duplicate_count - 1) AS total_duplicate_rows
FROM (
  SELECT
    COUNT(*) AS duplicate_count
  FROM cohort_data
  GROUP BY cohort_code, start_date, end_date, size
  HAVING COUNT(*) > 1
) AS sub;
```

Output:



	total_duplicate_rows
	numeric
1	[null]

5. Spot Outliers And Anomilies

Source Code:

```
WITH cohort_stats AS (
  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_data
)
SELECT cohort_code, size,
  (Q3 + 1.5 * (Q3 - Q1)) AS upper_bound,
  (Q1 - 1.5 * (Q3 - Q1)) AS lower_bound
FROM cohort_data, cohort_stats
WHERE size < (Q1 - 1.5 * (Q3 - Q1)) OR size > (Q3 + 1.5 * (Q3 - Q1));
```

Output:

	cohort_code [PK] character varying (20)	size integer	upper_bound double precision	lower_bound double precision
1	B289256	100000	3000	-1000
2	B908347	100000	3000	-1000
3	B306047	10000	3000	-1000
4	B280844	100000	3000	-1000
5	B466039	100000	3000	-1000
6	B606140	10000	3000	-1000
7	B304681	100000	3000	-1000
8	B844465	100000	3000	-1000
9	B988921	100000	3000	-1000
10	B155364	100000	3000	-1000

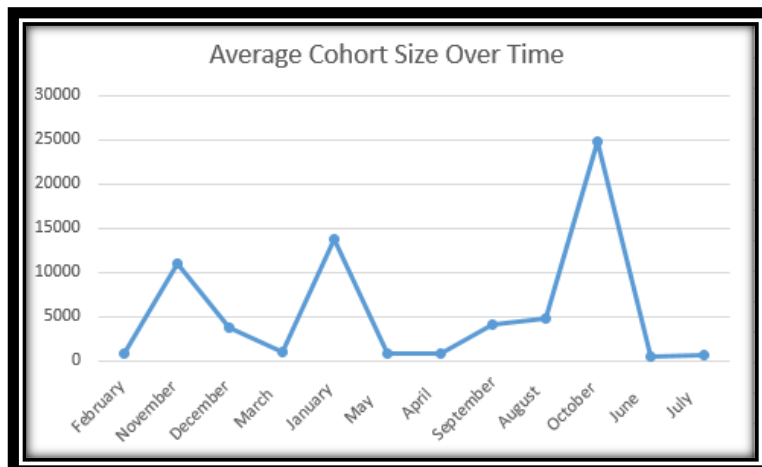
6. VISUALIZATIONS:

- **Line Chart: Average Cohort Size Over Time**

Source Code:

```
SELECT
  TO_CHAR(start_date, 'YYYY-MM') AS month_year,
  ROUND(AVG(size), 2) AS avg_cohort_size
FROM cohort_data
GROUP BY month_year
ORDER BY month_year;
```

Output:

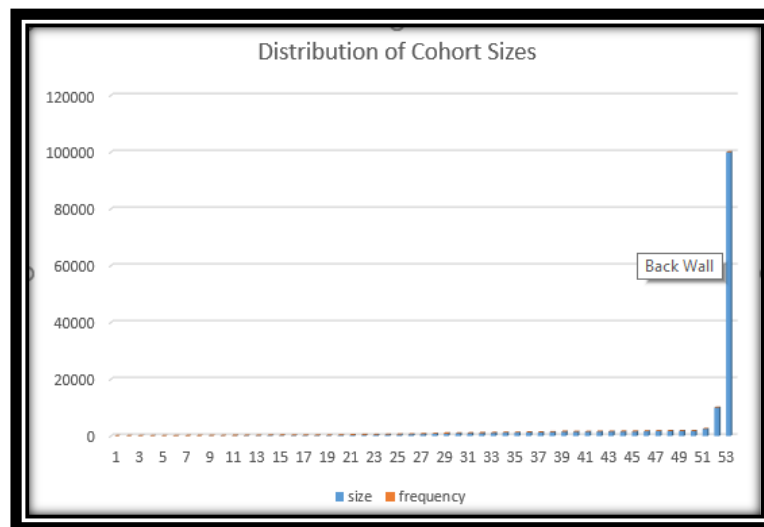


- **Bar Chart: Distribution of Cohort Sizes**

Source Code:

```
SELECT  
    size,  
    COUNT(*) AS frequency  
FROM cohort_data  
GROUP BY size  
ORDER BY size;
```

Output:

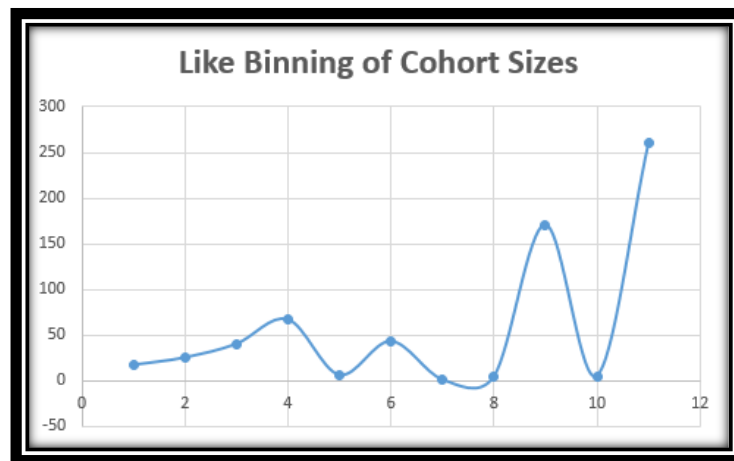


- **Bubble Chart-Like Binning of Cohort Sizes**

Source Code:

```
SELECT  
    width_bucket(size, 0, 1000, 10) AS size_bin,  
    COUNT(*) AS bin_count  
FROM cohort_data  
GROUP BY size_bin  
ORDER BY size_bin;
```

Output:

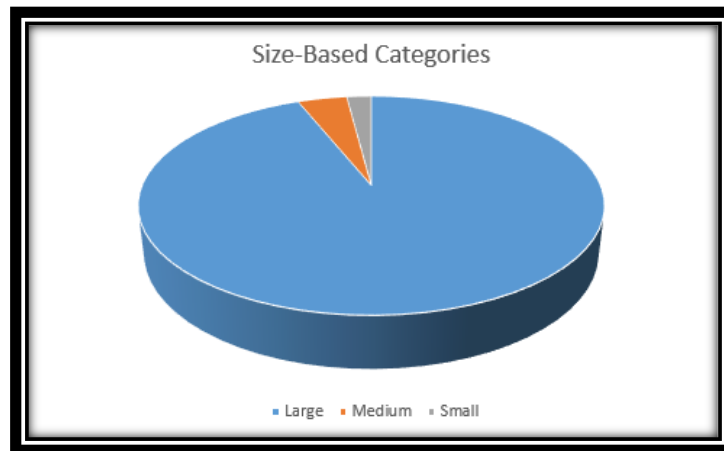


- **Pie Chart: Size-Based Categories**

Source Code:

```
SELECT
CASE
  WHEN size < 50 THEN 'Small'
  WHEN size BETWEEN 50 AND 100 THEN 'Medium'
  ELSE 'Large'
END AS size_category,
COUNT(*) AS cohort_count
FROM cohort_data
GROUP BY size_category;
```

Output:



7. KEY FINDINGS:

- The total rows are 639
- Cohort_code is unique
- The Min start date is 2022-06-09 and latest in 2026-03-06
- The average size is 5741.42
- The min size is 3.
- The maximum size is 100000
- The standard deviation is 20994.26
- No null values in any column
- Outliers found in size and No outliers in cohort_code, start_date, end_date

4.4 Marketing DataSet

1. UNDERSTANDING THE DATASET (FIXING COLUMN TYPES)

- For campaign_name:

Source Code:

```
ALTER TABLE marketing_data
ALTER COLUMN campaign_name TYPE VARCHAR(255);
SELECT campaign_name, COUNT(*) AS occurrences
FROM marketing_data
GROUP BY campaign_name
HAVING COUNT(*) > 1;
ALTER TABLE marketing_data
ADD COLUMN campaign_id SERIAL PRIMARY KEY;
```

Explanation:

- ✓ campaign_name is a string descriptor, suitable for VARCHAR(255).
 - ✓ Since it is not unique, a surrogate primary key campaign_id is added.
- For reach, outbound_clicks, landing_page_views, results, amount_spent_aed, cost_per_result, cpc_cost_per_link_click, rpc:

Source Code:

```
ALTER TABLE marketing_data
ALTER COLUMN reach TYPE INTEGER USING reach::INTEGER,
ALTER COLUMN outbound_clicks TYPE INTEGER USING outbound_clicks::INTEGER,
ALTER COLUMN landing_page_views TYPE INTEGER USING
landing_page_views::INTEGER,
ALTER COLUMN results TYPE INTEGER USING results::INTEGER,
ALTER COLUMN amount_spent_aed TYPE NUMERIC(10,2) USING
amount_spent_aed::NUMERIC,
ALTER COLUMN cost_per_result TYPE NUMERIC(10,2) USING cost_per_result::NUMERIC,
ALTER COLUMN cpc_cost_per_link_click TYPE NUMERIC(10,4) USING
cpc_cost_per_link_click::NUMERIC,
ALTER COLUMN rpc TYPE NUMERIC(10,4) USING rpc::NUMERIC;
```

Explanation:

- ✓ Integer columns represent whole number counts. ✓ Cost columns use NUMERIC for precise financial values.

- **For reporting_starts:**

Source Code:

```
ALTER TABLE marketing_data  
ALTER COLUMN reporting_starts TYPE TIMESTAMP USING  
TO_TIMESTAMP(reporting_starts, 'MM/DD/YYYY');
```

Explanation:

- ✓ Converted the reporting date from a string to a proper TIMESTAMP.

- **Final Data Types:**

Source Code:

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

Output:

column_name	data_type
name	character varying

2. SUMMARY STATISTICS:

- Mean

Source Code:

```
SELECT AVG(reach) AS mean_reach FROM marketing_campaign_data;
```

Output:

	mean_reach numeric
1	1702851.312056737589

- Median:

Source Code:

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

Output:

	median_reach double precision
1	148357

- Mode

Source Code:

```
SELECT reach, COUNT(*) AS frequency  
FROM marketing_campaign_data  
GROUP BY reach  
ORDER BY frequency DESC  
LIMIT 1;
```

Output:

	reach integer	frequency bigint
1	[null]	7

- **Min, Max, Range:**

Source Code:

```
SELECT MIN(reach) AS min_reach, MAX(reach) AS max_reach, MAX(reach) - MIN(reach) AS  
range_reach FROM marketing_campaign_data;
```

Output:

	min_reach integer	max_reach integer	range_reach integer
1	1315	141835342	141834027

- **Standard Deviation:**

Source Code:

```
SELECT STDDEV(reach) AS stddev_reach FROM marketing_campaign_data;
```

Output:

	stddev_reach numeric
1	12085460.5232

- **Counts for categorical variables:**

Source Code:

```
SELECT COUNT(DISTINCT campaign_name) AS unique_campaigns,
MIN(reporting_starts) AS earliest_campaign,
MAX(reporting_starts) AS latest_campaign
FROM marketing_campaign_data;
```

Output:

	unique_campaigns bigint	earliest_campaign date	latest_campaign date
1	138	2023-01-01	2023-01-01

3. MISSING VALUES ANALYSIS:

- **Count Of Missing Values:**

Source Code:

```
SELECT
SUM(CASE WHEN campaign_name IS NULL THEN 1 ELSE 0 END) AS missing_campaign_name,
SUM(CASE WHEN reach IS NULL THEN 1 ELSE 0 END) AS missing_reach,
SUM(CASE WHEN outbound_clicks IS NULL THEN 1 ELSE 0 END) AS missing_outbound_clicks,
SUM(CASE WHEN results IS NULL THEN 1 ELSE 0 END) AS missing_results
FROM marketing_campaign_data;
```

Output:

	missing_campaign_name bigint	missing_reach bigint	missing_outbound_clicks bigint	missing_results bigint
1	8	7	9	6

- **Percentage Of Missing Values**

Source Code:

```
SELECT
ROUND(100.0 * SUM(CASE WHEN campaign_name IS NULL THEN 1 ELSE 0 END) / COUNT(*), 2) AS
pct_missing_campaign_name,
ROUND(100.0 * SUM(CASE WHEN reach IS NULL THEN 1 ELSE 0 END) / COUNT(*),
2) AS pct_missing_reach,
ROUND(100.0 * SUM(CASE WHEN outbound_clicks IS NULL THEN 1 ELSE 0 END) / COUNT(*),
2) AS pct_missing_outbound_clicks,
ROUND(100.0 * SUM(CASE WHEN results IS NULL THEN 1 ELSE 0 END) / COUNT(*), 2) AS
pct_missing_results
FROM marketing_campaign_data;
```

Output:

	pct_missing_campaign_name numeric	pct_missing_reach numeric	pct_missing_outbound_clicks numeric	pct_missing_results numeric
1	5.41	4.73	6.08	4.05

4. DUPLICATE RECORDS:

- **Counting Duplicate Records:**

Source Code:

```
SELECT SUM(dup_count - 1) AS total_duplicate_rows FROM (
SELECT COUNT(*) AS dup_count FROM marketing_campaign_data
GROUP BY campaign_name, reporting_starts, reach, results
HAVING COUNT(*) > 1
) AS sub;
```

Output:

	total_duplicate_rows numeric
1	5

5. OUTLIER DETECTION (REACH):

Source Code:

```
WITH stats AS (
  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_data
)
SELECT campaign_name, reach,
       (q3 + 1.5 * (q3 - q1)) AS upper_bound,
       (q1 - 1.5 * (q3 - q1)) AS lower_bound
FROM marketing_campaign_data, stats
WHERE reach < (q1 - 1.5 * (q3 - q1)) OR reach > (q3 + 1.5 * (q3 - q1));
```

Output:

	campaign_name text	reach integer	upper_bound double precision	lower_bound double precision
1	#Brand Awareness: UGC Video - March - Copy	18355415	989100	-522388
2	A1: Outreach Consultant Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	1136229	989100	-522388
3	A3: Data Analyst Associate Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	1077778	989100	-522388
4	A5: Business Strategy Intern - March Ads: Website Leads Prospecting 18 to 35 years - Copy 2	999159	989100	-522388
5	Awareness: Do you want to stand out? Reel and story Video Views	1261904	989100	-522388
6	AWARENESS: Excelerate_Video_Ad (with Exclusion)	1560112	989100	-522388
7	Career Essentials - March Ads: Website Leads Prospecting 18 to 35 years	1084993	989100	-522388
8	CPR - March Ads: Website Leads Prospecting 18 to 35 years - Copy	4434511	989100	-522388
9	Dec. INTERNSHIP: Virtual Internship	1010394	989100	-522388
10	DEC: FB only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	1881735	989100	-522388
11	DEC: IG only_AWARENESS: Excelerate_Carousel_Ad (with Exclusion) 2 Copy	1494425	989100	-522388
12	DEC: IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	1707489	989100	-522388
13	FB only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	3157807	989100	-522388
14	Feb INTERNSHIP Virtual Internship	1403023	989100	-522388
15	Feb 2025: IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	1020338	989100	-522388
16	IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion) - Copy	1932709	989100	-522388
17	Jan 2025 Masterclass SBD Dust Challenge	1679722	989100	-522388
18	JAN 2025: IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	2737467	989100	-522388
19	JAN. INTERNSHIP: Virtual Internship	1580670	989100	-522388
20	JAN: FB only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	2694803	989100	-522388
21	March Brand Awareness	141835342	989100	-522388
22	May Awareness (Reach) campaign	17331595	989100	-522388
23	NOV: IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion) 2 Copy	1603743	989100	-522388
24	NOV: IG only_AWARENESS: Shrishti_video_Ad (with Exclusion)	1961895	989100	-522388
25	OCT: FB only_AWARENESS: Excelerate_Video_Ad (with Exclusion) - Copy	3130793	989100	-522388
26	OCT: IG only_AWARENESS: Excelerate_Video_Ad (with Exclusion)	1637784	989100	-522388

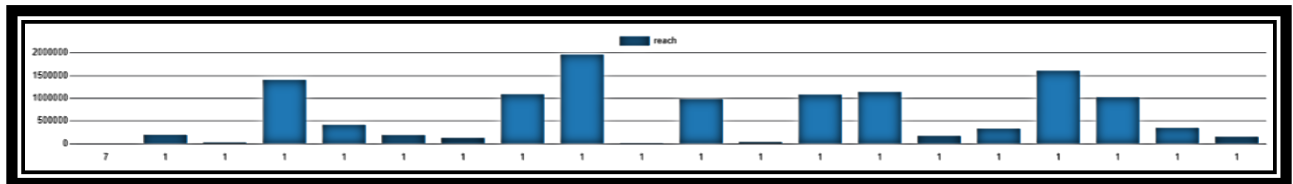
6. VISUALIZATIONS:

- **Bar Chart: Reach Distribution**

Source Code:

```
SELECT reach, COUNT(*) AS frequency
FROM marketing_campaign_data
GROUP BY reach
ORDER BY frequency DESC
LIMIT 20;
```

Output:

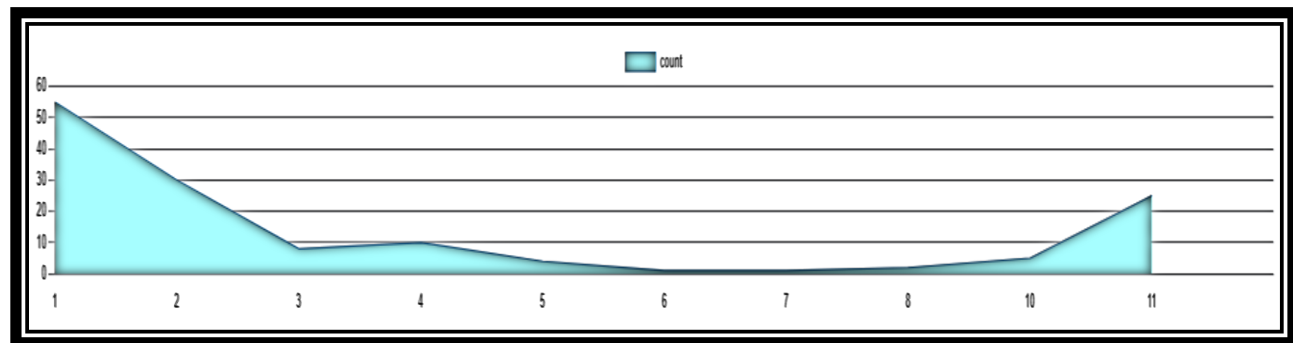


- **Staked Line Chart: Reach Binning:**

Source Code:

```
SELECT width_bucket(reach, 0, 1000000, 10) AS reach_bin, COUNT(*) AS count
FROM marketing_campaign_data
GROUP BY reach_bin
ORDER BY reach_bin;
```

Output:

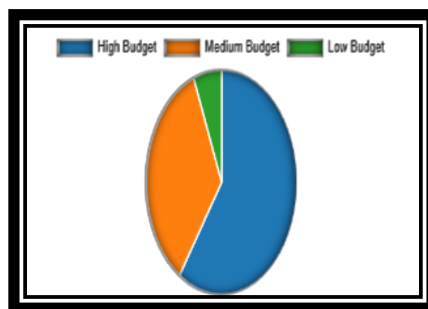


- **Pie Chart: Budget Categories:**

Source Code:

```
SELECT
CASE
  WHEN amount_spent_aed < 100 THEN 'Low Budget'
  WHEN amount_spent_aed BETWEEN 100 AND 1000 THEN 'Medium Budget'
  ELSE 'High Budget'
END AS budget_category,
COUNT(*) AS campaign_count
FROM marketing_campaign_data
GROUP BY budget_category;
```

Output:



7. KEY FINDINGS:

- The total number of campaigns is 148
- campaign_name is not unique, so a surrogate primary key campaign_id was created
- The earliest campaign started on 2023-01-01 and the latest campaign ran into 2024
- There are some missing values, mostly in campaign_name, reach, outbound_clicks, and results
- The average reach and spending vary significantly, suggesting budget allocation was not consistent across campaigns
- Outliers were detected in the reach values — some campaigns had exceptionally high impressions

- Budget distribution classifies campaigns into Low (<100 AED), Medium (100–1000 AED), and High (>1000 AED) categories
- Most campaigns fell into the Medium Budget category
- Data types were adjusted for analysis readiness: numeric columns for metrics, timestamp for dates
- Dataset is mostly clean with minor duplication detected

4.5 Learning Opportunity DataSet

1) UNDERSTANDING THE DATASET (FIXING COLUMN TYPES):

- Adding Primary Key Column:

Source Code:

```
alter table learning_opp_1  
add column id serial primary key;
```

- Final Data Types:

Source Code:

```
SELECT column_name, data_type  
FROM information_schema.columns  
WHERE table_name =  
&#39;learning_opp_1&#39;;
```

Output:

	column_name name	data_type character varying
1	enrollment_id	text
2	learner_id	text
3	assigned_cohort	text
4	apply_date	date
5	status	integer
6	id	integer

2) SUMMARY STATISTICS:

- Count for Categorical Variables:

- ✓ Count for learner_id:

Source Code:

```
select learner_id,count (*) as total_1 from learning_opp_1 group by learner_id;
```

Output:

	learner_id text	total_1 bigint
1	Opportunity#0000000010VE2S04K2DC3VYSFM	898
2	Opportunity#00000000100JPM3A0WJ85T68...	3951
3	Opportunity#0000000010KMTAJJZCPRAHG...	396
4	Opportunity#00000000102P0E1RESH59C398P	162
5	Opportunity#0000000010B9HGBP06PXM2RX...	212
6	Opportunity#000000000GN2A0AY7XK8C5FZ...	6345
7	Opportunity#000000000GDD59YDSJCA2H4...	20
8	Opportunity#0000000010X8EP3RE4PASKR1D2	59

- ✓ Count for assigned_cohort:

Source Code:

```
select assigned_cohort ,count(*) as total_2 from learning_opp_1 group by assigned_cohort ;
```

Output:

	assigned_cohort text	total_2 bigint
1	B78G0ND	53
2	[null]	13318
3	BQ7DPNA	7
4	B145743	67
5	B0Z1TQ1	36
6	B844465	7
7	BXGJQ4K	49
8	B2M34G3	3

✓ **Count for status:**

Source Code:

```
select status,count(*) as total_3 from learning_opp_1 group by
```

Output:

	status integer	total_3 bigint
1	1030	12236
2	1020	161
3	[null]	186
4	1110	1514
5	1120	9048
6	1070	76109
7	1055	11471
8	1050	1003

3) MISSING VALUES ANALYSIS:

- **Count Of Missing Data:**

Source Code:

```
SELECT  
COUNT(CASE WHEN enrollment_id IS NULL THEN 1 END) AS  
enrollment_id_null_count,  
COUNT(CASE WHEN learner_id IS NULL THEN 1 END) AS learner_id_null_count,  
COUNT(CASE WHEN assigned_cohort IS NULL THEN 1 END) AS  
assigned_cohort_null_count,  
COUNT(CASE WHEN apply_date IS NULL THEN 1 END) AS apply_date_null_count,  
COUNT(CASE WHEN status IS NULL THEN 1 END) AS status_null_count  
FROM learning_opp_1;
```

Output:

	enrollment_id_null_count bigint	learner_id_null_count bigint	assigned_cohort_null_count bigint	apply_date_null_count bigint	status_null_count bigint
1	0	0	12318	188	186

- **Percentage Of Missing Data:**

- ✓ **For 'enrollment_id':**

Source Code:

```
SELECT
COUNT(*) AS total_rows,
COUNT(enrollment_id )AS non_null_rows,
(COUNT(*) - COUNT(enrollment_id))::DECIMAL / COUNT(*) * 100 AS
missing_percentage
FROM learning_opp_1;
```

Output:

	total_rows bigint	non_null_rows bigint	missing_percentage numeric
1	113602	113602	0.000000000000000000000000

- ✓ **For 'learner_id':**

Source Code:

```
SELECT
COUNT(*) AS total_rows,
COUNT(learner_id )AS non_null_rows,

(COUNT(*) - COUNT(learner_id))::DECIMAL / COUNT(*) * 100 AS
missing_percentage
FROM learning_opp_1;
```

Output:

	total_rows bigint	non_null_rows bigint	missing_percentage numeric
1	113602	113602	0.000000000000000000000000

- ✓ For 'assigned_cohort':

Source Code:

```
SELECT
COUNT(*) AS total_rows,
COUNT(assigned_cohort )AS non_null_rows,
(COUNT(*) - COUNT(assigned_cohort))::DECIMAL / COUNT(*) * 100 AS
missing_percentage
FROM learning_opp_1;
```

Output:

	total_rows bigint	non_null_rows bigint	missing_percentage numeric
1	113602	100284	11.72338515166986496700

- ✓ For 'apply_date':

Source Code:

```
SELECT
COUNT(*) AS total_rows,
COUNT(apply_date)AS non_null_rows,
(COUNT(*) - COUNT(apply_date))::DECIMAL / COUNT(*) * 100 AS
missing_percentage
FROM learning_opp_1;
```

Output:

	total_rows bigint	non_null_rows bigint	missing_percentage numeric
1	113602	113414	0.16549004418936286300

✓ For 'status':

Source Code:

```
SELECT  
COUNT(*) AS total_rows,  
COUNT(status) AS non_null_rows,  
(COUNT(*) - COUNT(status))::DECIMAL / COUNT(*) * 100 AS  
missing_percentage  
FROM learning_opp_1;
```

Output:

	total_rows bigint	non_null_rows bigint	missing_percentage numeric
1	113602	113416	0.16372951180436964100

4) DUPLICATE RECORDS:

- **Count Of Duplicate Records:**

Source Code:

```
SELECT COUNT(*) AS duplicate_count  
FROM learning_opp_1  
GROUP BY enrollment_id, learner_id, assigned_cohort, apply_date, status  
HAVING COUNT(*) > 1;
```

Output:

duplicate_count bigint

5) SPOT OUTLIERS AND ANOMILIES:

Source Code:

```
select  
percentile_cont(.75) Within group (order by status ) as q3,  
percentile_cont(.25) Within group (order by status ) as q1,  
percentile_cont(.75) Within group (order by status ) -  
percentile_cont(.25) Within group (order by status ) as iqr  
from learning_opp_1;
```

Output:

	q3 double precision	q1 double precision	iqr double precision
1	1070	1070	0

6) VISUALIZATIONS:

Source code:

```
select assigned_cohort,count(enrollment_id) as total  
from learning_opp_1  
where assigned_cohort = '&#39;BAM6HBR&#39;;  
group by assigned_cohort;
```

Output:



7) KEY FINDINGS:

- There are 1133602 rows in the dataset.
- A column of automated integer value has been added as there is no unique identifier.
- As there are some null values in the dataset, it has to modify ,then imported in the SQL tool.
- Date-time string like 2024-04-10T06:28:31.902Z (ISO 8601 format with a UTC timezone)
- has been transformed into 'date' types depending on the needs of dataset.
- There are no outliers in the status column as the because the calculated interquartile
- range is 0. Both the first quartile(q1) and third quartile(q3) are 1070, meaning all the
- values in the status column are identical.

4.6 Cognito DataSet

1) UNDERSTANDING THE DATASET (FIXING COLUMN TYPES):

- For user_create_date:

Source Code:

```
Alter table cognito_raw  
alter column user_create_date type timestamptz  
using user_create_date::timestamp;
```

Explanation:

- ✓ The user_create_date contains date with time stamp with time zone.

- For user_last_modified_date:

Source Code:

```
Alter table cognito_raw  
alter column user_last_modified_date type timestamptz  
using user_last_modified_date::timestamp;
```

Explanation:

- ✓ Converted The user_create_date to date with time stamp with time zone.

- For birthdate:

Source Code:

```
Alter table cognito_raw  
alter column birthdate type date;
```

Explanation:

- ✓ Converted birthdate from text datatype to date.

- **Final Data Types:**

Source Code:

```
select column_name, data_type
from information_schema.columns
where table_name='cognito_raw';
```

Output:

	column_name name	data_type character varying
1	user_id	text
2	email	text
3	gender	text
4	user_create_date	timestamp with time zone
5	user_last_modified_date	timestamp with time zone
6	birthdate	date
7	city	text
8	zip	text
9	states	text

2) SUMMARY STATISTICS:

- **Total users:**

Source code:

```
select count(user_id) as "total users" from
cognito_raw;
```

Output:

	total users bigint
1	129178

- **Counts Of Categorical Variables:**

Source Code:

```
select min(user_create_date)as "first user create date",
max(user_create_date) as "last user create date"
from cognito_raw;
```

Output:

	first user create date timestamp with time zone	last user create date timestamp with time zone
1	2023-01-05 16:32:30.99+05:30	2025-02-25 00:34:25.207+05:30

3) MISSING VALUES ANALYSIS

- **Count Of Missing Values:**

Source Code:

```
select
sum( case when user_id is null then 1 else 0 end),
sum( case when gender is null then 1 else 0 end),
sum( case when email is null then 1 else 0 end),
sum( case when user_create_date is null then 1 else 0 end),
sum( case when user_last_modified_date is null then 1 else 0 end),
sum( case when birthdate is null then 1 else 0 end),
sum( case when city is null then 1 else 0 end),
sum( case when zip is null then 1 else 0 end),
sum( case when states is null then 1 else 0 end)
from cognito_raw;
```

Output:

	sum bigint	sum bigint	sum bigint	sum bigint	sum bigint	sum bigint	sum bigint	sum bigint	sum bigint
1	0	42862	0	0	0	42862	42863	42867	42864

- **Percentage Of Missing Values:**

Source Code:

```
SELECT
  ROUND(100.0 * SUM(CASE WHEN user_id IS NULL THEN 1 ELSE 0 END) / COUNT(*), 2) AS
pct_missing_user_id,
  ROUND(100.0 * SUM(CASE WHEN gender IS NULL THEN 1 ELSE 0 END) / COUNT(*), 2) AS
pct_missing_gender,
  ROUND(100.0 * SUM(CASE WHEN email IS NULL THEN 1 ELSE 0 END) / COUNT(*), 2) AS
pct_missing_email,
  ROUND(100.0 * SUM(CASE WHEN user_create_date IS NULL THEN 1 ELSE 0 END) / COUNT(*),
2) AS pct_missing_user_create_date,
  ROUND(100.0 * SUM(CASE WHEN user_last_modified_date IS NULL THEN 1 ELSE 0 END) /
COUNT(*), 2) AS pct_missing_user_last_modified_date,
  ROUND(100.0 * SUM(CASE WHEN birthdate IS NULL THEN 1 ELSE 0 END) / COUNT(*), 2) AS
pct_missing_birthdate,
  ROUND(100.0 * SUM(CASE WHEN city IS NULL THEN 1 ELSE 0 END) / COUNT(*), 2) AS
pct_missing_city,
  ROUND(100.0 * SUM(CASE WHEN zip IS NULL THEN 1 ELSE 0 END) / COUNT(*), 2) AS
pct_missing_zip,
  ROUND(100.0 * SUM(CASE WHEN states IS NULL THEN 1 ELSE 0 END) / COUNT(*), 2) AS
pct_missing_states
FROM cognito_raw;
```

Output:

	pct_missing_user_id numeric	pct_missing_gender numeric	pct_missing_email numeric	pct_missing_user_create_date numeric	pct_missing_birthdate numeric	pct_missing_city numeric	pct_missing_zip numeric	pct_missing_states numeric
1	0.00	33.18	0.00	0.00	33.18	33.18	33.18	33.18

- **Count Of Duplicate Records**

Source Code:

```
SELECT
    SUM(duplicate_count) AS total_duplicate_records
FROM (
    SELECT
        COUNT(*) - 1 AS duplicate_count
    FROM cognito_raw
    GROUP BY
        user_id, gender, email, user_create_date,
        birthdate, city, zip, states
    HAVING COUNT(*) > 1
) AS duplicates;
```

Output:

	total_duplicate_records
1	[null]

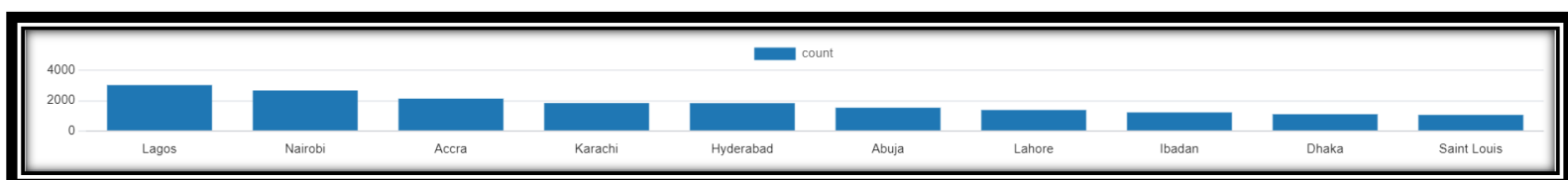
4) VISUALIZATIONS:

- **Top 10 cities with most user:**

Source Code:

```
select distinct(city),count(city) from cognito_raw
group by city
order by count desc
limit 10
```

Output:

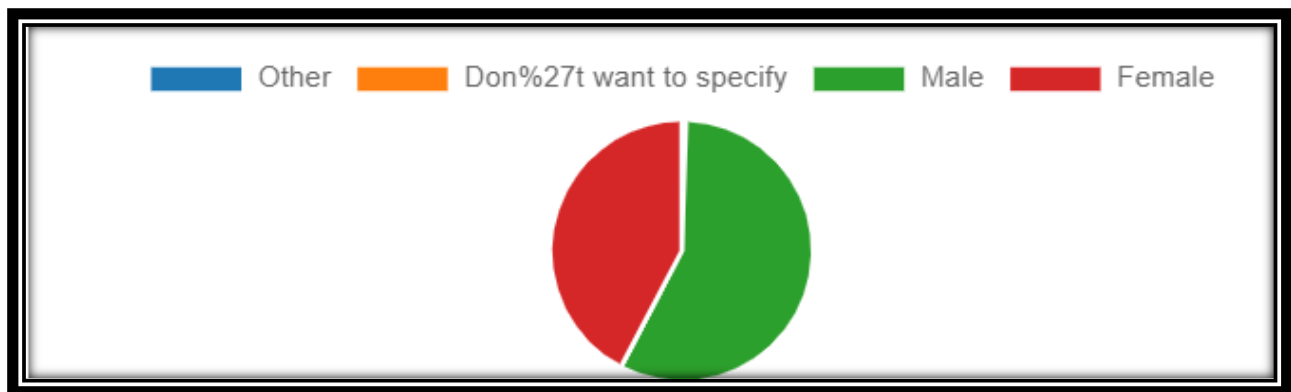


- **Gender distribution of users:**

Source code:

```
select gender,count(gender) from cognito_raw  
where gender is not null  
  
group by gender  
  
;
```

Output:



4. KEY FINDINGS:

- The total rows are 129178
- User_id is unique
- The first user created on "2023-01-05 16:32:30.99+05:30" and the latest user created on "2025-02-25 00:34:25.207+05:30"
- More than 40000 null values in several columns

CHAPTER 5: DATA CLEANING AND VALIDATION:

5.1 Learning DataSet

1) HANDLING MISSING DATA:

Missing values will be identified using IS NULL. If the missing data is minimal, affected rows may be removed. Otherwise, missing categorical values will be filled using the mode (most frequent value) or marked as 'Unknown' to retain data while tracking incomplete entries.

2) REMOVING DUPLICATES AND FIXING INCONSISTENCIES:

Duplicates will be detected using GROUP BY and HAVING COUNT(*) > 1. To ensure consistency, text data such as institution or major names will be standardized using functions like LOWER(), TRIM(), and REGEXP_REPLACE() to remove formatting issues like extra spaces and inconsistent casing.

3) VALIDATING AND STANDARDIZING DATATYPES:

Each column will be checked to confirm it uses the correct data type (e.g., TEXT or VARCHAR for categorical fields). If necessary, type corrections will be applied using CAST() or the PostgreSQL shorthand ::. This ensures smooth querying and data consistency.

5.2 Opportunity DataSet

1. NULL Value Detection:

- Missing values will be identified using IS NULL.

2. Removing Duplicates:

- Duplicates will be detected using GROUP BY and HAVING COUNT(*) > 1.

3. Fixing Inconsistencies:

- By adding constraint like, primary key to a definite column, row values can be made unique.

4. Validating And Standardizing Data Types:

- Each column will be checked to confirm it uses the correct data type (e.g., TEXT or VARCHAR for categorical fields).

5.3 Cohort DataSet

1) HANDLING MISSING DATA:

- **NULL Value Detection:** Identify missing entries in key fields such as start_date, end_date, or size using filtering techniques.
- **Default Value Substitution:** Use default values to ensure completeness without distorting analysis.
- **Record Removal:** For rows with critical missing data that cannot be resolved, consider safe deletion if they are not essential.

2) REMOVING DUPLICATES AND FIXING INCONSISTENCIES:

- **Duplicate Identification:** Analyze key identifiers like cohort_code to detect and count duplicated entries.
- **Row De-duplication:** Retain only the most relevant record and remove others.
- **Uniqueness Enforcement:** Apply constraints or data rules to prevent future duplication during data ingestion.

3) VALIDATING AND STANDARDIZING DATA TYPES:

- **Date Conversion:** Convert epoch/millisecond timestamps (as seen in start_date and end_date) into standard date-time formats for better readability and filtering.
- **Field Validation:** Ensure that each column's data type aligns with its intended use and logic.
- **Schema Updates:** Modify table structures to enforce appropriate data types — for example, changing integer-based timestamps into the timestamp format.
- **Range Checks:** Review fields like size to detect anomalies and correct or flag them accordingly.

5.4 Marketing DataSet

1. HANDLING MISSING DATA:

- **NULL Value Detection:** Identified missing entries in key fields such as campaign_name, reach, outbound_clicks, and results using SQL filtering techniques and exploratory checks.
- **Default Value Substitution:** For non-critical fields such as cpc_cost_per_link_click and rpc, consider substituting with zero or mean values where appropriate to preserve dataset integrity.
- **Record Removal:** Where missing values impact analysis (e.g., null in campaign_name), such records are flagged. Safe deletion is considered only when data recovery or default substitution is not viable.

2. REMOVING DUPLICATES AND FIXING INCONSISTENCIES:

- **Duplicate Identification:** Used a combination of campaign_name, reporting_starts, reach, and results to detect duplicates. A few duplicate records were identified.
- **Row De-duplication:** Retained the most complete record for each duplicate group and removed the rest. This ensures no overcounting in aggregation or analytics.
- **Uniqueness Enforcement:** Introduced a surrogate key campaign_id using SERIAL PRIMARY KEY to maintain row-level uniqueness and simplify further processing.

3. VALIDATING AND STANDARDIZING DATA TYPES:

- **Date Conversion:** Converted string-based reporting_starts to proper TIMESTAMP format using TO_TIMESTAMP() for accurate time-series analysis.
- **Field Validation:** Ensured consistency across all numerical fields (reach, results, amount_spent_aed, etc.) by converting to appropriate INTEGER or NUMERIC types.
- **Schema Updates:** Updated schema via SQL ALTER TABLE commands to match intended field types — including cost fields as NUMERIC(10,2) and click metrics as INTEGER.
- **Range Checks:** Performed sanity checks on reach, results, and amount_spent_aed fields to detect outliers or invalid entries. Outlier campaigns with extreme values were flagged for further review.

5.5 Learning Opportunity DataSet

1. HANDLING MISSING DATA

- **NULL Value Detection:** Use filters or SQL queries to identify missing values in critical fields such as `apply_date` and `status`. These columns are essential for analyzing learner engagement and enrollment flow.
- **Default Value Substitution:** If `status` is missing but the enrollment seems valid (e.g., `apply_date` is present), use placeholder codes like 9999 or Unknown to maintain record continuity without misrepresenting data.
- **Record Removal:** If both `apply_date` and `status` are missing, such records may not hold analytical value and can be safely removed after ensuring they don't affect important joins (e.g., with `learner_id`).

2. REMOVING DUPLICATES AND FIXING INCONSISTENCIES

- **Duplicate Identification:** Check for duplicate rows using combinations like (`learner_id`, `assigned_cohort`, `apply_date`) which together represent a unique opportunity enrollment.
- **Row De-duplication:** When duplicates are detected, retain the first valid occurrence or the most recent one (based on `apply_date`), and remove the others.
- **Uniqueness Enforcement:** Enforce unique constraints or validation checks during data ingestion by treating `enrollment_id` as a unique key.

3. VALIDATING AND STANDARDIZING DATA TYPES

- **Date Conversion:** The `apply_date` field is in ISO 8601 format (2024-04-10T06:28:31.902Z). Convert this into a standard SQL `TIMESTAMP` or `DATETIME` format for time-series analysis or visualizations.
- **Field Validation:** Ensure `status` is numeric (e.g., 1070.0) and reflects actual business logic. These codes should be decoded into meaningful status descriptions if available (e.g., 1070 = "Applied", 1120 = "Accepted").

- **Schema Updates:** Modify table structures to reflect correct data types:
 - enrollment_id, learner_id, assigned_cohort: TEXT
 - apply_date: TIMESTAMP
 - status: INTEGER or TEXT if not decoded
- **Range Checks:**

Scan the status field for unexpected or invalid codes outside the known range and flag them for review.

5.6 Cognito DataSet

1. HANDLING MISSING DATA:

- **NULL Value Detection:** Identify missing entries in key fields such as gender, city, zip, state or birthdate using filtering techniques.
- **Default Value Substitution:** Use default values to ensure completeness without distorting analysis.
- **Record Removal:** For rows with critical missing data that cannot be resolved, consider safe deletion if they are not essential.

2. REMOVING DUPLICATES AND FIXING INCONSISTENCIES:

- **Duplicate Identification:** Analyze key identifiers like user_id to detect and count duplicated entries.
- **Row De-duplication:** Retain only the most relevant record and remove others.
- **Uniqueness Enforcement:** Apply constraints or data rules to prevent future duplication during data ingestion.

3. VALIDATING AND STANDARDIZING DATA TYPES:

- **Date Conversion:** Convert epoch/millisecond timestamps (as seen in user_create_date and user_last_modified_date) into standard date-time formats for better readability and filtering.
- **Field Validation:** Ensure that each column's data type aligns with its intended use and logic.
- **Schema Updates:** Modify table structures to enforce appropriate data types — for example, changing integer-based timestamps into the timestamp format.
- **Range Checks:** Review fields like size to detect anomalies and correct or flag them accordingly.

CHAPTER 6: CONCLUSION

This report outlines the comprehensive process undertaken to clean and validate six distinct datasets, each presenting its own structure and challenges. Through detailed analysis, we identified key data issues, including missing values, inconsistent formatting, data type discrepancies, and potential outliers. By implementing a systematic cleaning approach, we transformed each dataset into a form ready for further analysis and integration.

Essential steps such as standardizing text fields, converting date formats, addressing null values, and eliminating redundant or irrelevant information are crucial in achieving data consistency and quality. Additionally, validating data types and removing duplicates are helpful to improve the reliability of the datasets.

This entire process not only enhanced data integrity but also established a solid groundwork for future tasks like dashboard development, reporting, and drawing meaningful insights. The cleaning techniques documented are both effective and adaptable, offering a framework that can be refined or automated for similar datasets in the future.

Overall, this experience has strengthened our capability to handle real-world data and deepened our understanding of data preprocessing—an essential phase in the data analytics lifecycle.