## **Project Description**

This project is all about using data to improve how a company runs and spotting sudden changes in key numbers. As a Lead Data Analyst, my job is to work with different teams—like operations, support, and marketing—to find useful insights in the data they collect.

The project is divided into two main parts:

- **1.Job Data Analysis** Looking at how jobs are reviewed, how often they happen, which languages are used, and whether there are any duplicate records.
- **1.Investigating Metric Spikes** Studying user behavior, engagement, and email activity to find patterns, growth trends, and sudden drops or spikes in activity.

## How will I approach the analysis

Writing SQL Queries: I'll use SQL to analyze data, track daily job reviews, check language trends, and find duplicate records.

**Finding Trends Over Time**: Instead of just looking at daily numbers, I'll calculate 7-day averages to get a clearer picture of how things change over time.

**Understanding User Activity**: I'll track user growth, weekly engagement, and retention to see how well a product is performing.

**Detecting Unusual Spikes**: If there's a sudden jump or drop in important metrics, I'll investigate what caused it and suggest possible solutions.

## Tech-Stack Used

For this project, I used the following tools and technologies to analyze data, write queries, and derive meaningful insights:

- **1.MySQL (Ver 8.0.41)** The DBMS for storing and querying structured data. It helped in writing SQL queries to analyze job reviews, user engagement, and metric trends.
- **1.MySQL Workbench** Used for writing and testing SQL queries, visualizing database schemas, and managing data efficiently.

## Steps for creating the database and the table

```
use jobDB;
```

Creating the table as per the Data available

```
create table job_data (
   ds date,
   job_id int not null,
   actor_id int not null,
   event varchar(20),
   language varchar(20),
   time_spent int,
   org varchar(2)
);
```

## Inserted data through query and the output

```
INSERT INTO job_data (job_id, actor_id, event, language, time_spent, org, ds)
VALUES

(21, 1001, 'skip', 'English', 15, 'A', '2020-11-30'),
(22, 1006, 'transfer', 'Arabic', 25, 'B', '2020-11-30'),
(23, 1003, 'decision', 'Persian', 20, 'C', '2020-11-29'),
(23, 1005, 'transfer', 'Persian', 22, 'D', '2020-11-28'),
(25, 1002, 'decision', 'Hindi', 11, 'B', '2020-11-28'),
(11, 1007, 'decision', 'French', 104, 'D', '2020-11-27'),
(23, 1004, 'skip', 'Persian', 56, 'A', '2020-11-26'),
(20, 1003, 'transfer', 'Italian', 45, 'C', '2020-11-25');
```

	ds	job_id	actor_id	event	language	time_spent	org
•	2020-11-30	21	1001	skip	English	15	Α
	2020-11-30	22	1006	transfer	Arabic	25	В
	2020-11-29	23	1003	decision	Persian	20	C
	2020-11-28	23	1005	transfer	Persian	22	D
	2020-11-28	25	1002	decision	Hindi	11	В
	2020-11-27	11	1007	decision	French	104	D
	2020-11-26	23	1004	skip	Persian	56	Α
	2020-11-25	20	1003	transfer	Italian	45	С

## Task 1

#### **Jobs Reviewed Over Time:**

- A. Objective: Calculate the number of jobs reviewed per hour for each day in November 2020.
- B. Your Task: Write an SQL query to calculate the number of jobs reviewed per hour for each day in November 2020.

## Query shot for the task & the outcome

```
SELECT
    ds AS DtReviewed,
    count(*) as JobIDs,
    round ((86400 / SUM(time_spent)), 0) AS JobsReviewedPerDay,
    round (((86400 / SUM(time spent)) / 24), 0) AS JobsReviewedPerHour
FROM job data
GROUP BY ds
having ds between '2020-11-01' and '2020-11-30'
ORDER BY ds;
```

	DtReviewed	JobIDs	JobsReviewedPerDay	JobsReviewedPerHour
•	2020-11-25	1	1920	80
	2020-11-26	1	1543	64
	2020-11-27	1	831	35
	2020-11-28	2	2618	109
	2020-11-29	1	4320	180
	2020-11-30	2	2160	90

#### Consideration

**Every day is of 24 hours Every day = 86400 seconds** 

## Task 2

#### **Throughput Analysis:**

- A. Objective: Calculate the 7-day rolling average of throughput (number of events per second).
- A. Your Task: Write an SQL query to calculate the 7-day rolling average of throughput. Additionally, explain whether you prefer using the daily metric or the 7-day rolling average for throughput, and why.

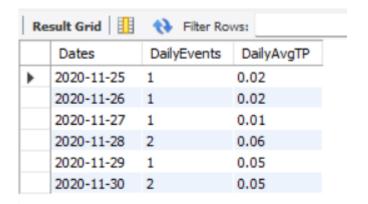
Project by – Alokk Joshi

To get the throughput data
We need to divide the number of events by total time spent
Below are the queries for daily and weekly throughput

Query and outcome for

Daily average throughput

```
select
ds as Dates,
count(event) as DailyEvents,
round(count(event) / sum(time_spent), 2) as DailyAvgTP
from job_data
group by Dates
order by Dates;
```



#### Project by – Alokk Joshi

To get the throughput data
We need to divide the number of events by total time spent
Below are the queries for daily and weekly throughput

Query and outcome for

Weekly average throughput

Week = no of distinct dates provided

```
select
count(distinct ds) as DateCount,
round(count(event) / sum(time_spent), 2) as WeeklyAvgTP
from job_data;
```



Project by – Alokk Joshi

7-Day Rolling Average is more preferred and reliable than a daily metric

#### because:

- 1. Smooths out fluctuations Daily metrics can be highly volatile due to outliers (like a sudden spike or drop in workload).
- 2. Provides a better trend analysis It helps spot gradual increases or decreases over time.
- 3. Reduces confusion Eliminates misleading insights caused by a single day's unusual activity.

However, daily metrics are useful for immediate monitoring and real-time decision-making, especially in dynamic and demanding environments.

#### Finally ---->

For long-term analysis, the 7-day rolling average is preferred. But for daily performance tracking, the daily metric is valuable.

## Task 3

#### **Language Share Analysis:**

- A. Objective: Calculate the percentage share of each language in the last 30 days.
- A. Your Task: Write an SQL query to calculate the percentage share of each language over the last 30 days.

Project by – Alokk Joshi

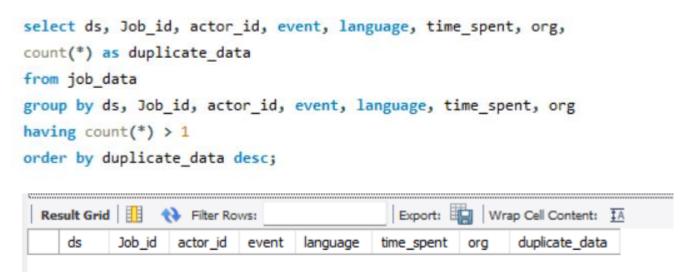
```
select language,
count(job_id),
round(count(job_id) * 100 / (select count(*) from job_data), 2) as Percentage
from job_data
group by language;
```

	language	count(job_id)	Percentage
•	English	1	12.50
	Arabic	1	12.50
	Persian	3	37.50
	Hindi	1	12.50
	French	1	12.50
	Italian	1	12.50

## Task 4

#### **Duplicate Rows Detection:**

- A. Objective: Identify duplicate rows in the data.
- B. Your Task: Write an SQL query to display duplicate rows from the job\_data table.



This query does not return any value because there are no duplicate rows in the table, although there are individual duplicate items like job id and actor id but they have different set of row values

Project by – Alokk Joshi

#### **Case Study 2: Investigating Metric Spike**

#### You will be working with three tables:

•users: Contains one row per user, with descriptive information about that user's account.

•events: Contains one row per event, where an event is an action that a user has taken (e.g., login, messaging, search).

•email\_events: Contains events specific to the sending of emails.

#### Project by – Alokk Joshi

#### First Create all the tables

```
use jobDB;

create table users

(
user_id int,
created_at datetime,
company_id int,
language varchar(20),
activated_at datetime,
state varchar(15)
);
```

```
CREATE TABLE events (
    user_id INT,
    occurred_at datetime,
    event_type VARCHAR(30),
    event_name VARCHAR(30),
    location VARCHAR(30),
    device VARCHAR(60),
    user_type VARCHAR(4)
);
```

```
CREATE TABLE email_events (
    user_id INT,
    occurred_at datetime,
    action VARCHAR(50),
    user_type VARCHAR(2)
);
```

Project by – Alokk Joshi

Next bigger task is to clean the data for uploading

In the csv files all the other data were clean except date format

First, we have to convert all the dates into same data format

For that, I chose Excel power query And within power query

I converted the dates as per English(India) locale first and then customized the dates into below format

yyyy-mm-dd hh:mm:ss

Saved the csv files

#### Then loaded all the files through below queries

```
LOAD DATA INFILE 'D:/Trainty/MySql/Project 3 files/users.csv'
          INTO TABLE users
          FIELDS TERMINATED BY ','
                                                                            Users data
          ENCLOSED BY '"'
          LINES TERMINATED BY '\n'
          IGNORE 1 ROWS
          (user_id, created_at, company_id, language, activated_at, device, state);
LOAD DATA INFILE 'D:/Trainty/MySql/Project 3 files/events.csv'
INTO TABLE events
                                                                     Events data
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
                                                                                                                  Email events data
IGNORE 1 ROWS
(user_id, occurred_at, event_type, event_name, location, device, user_type);
                                                   LOAD DATA INFILE 'D:/Trainty/MySql/Project 3 files/email events.csv'
                                                   INTO TABLE email events
                                                   FIELDS TERMINATED BY ','
                                                   ENCLOSED BY '"'
                                                   LINES TERMINATED BY '\n'
                                                   IGNORE 1 ROWS
                                                   (user id, occurred at, action, user type);
```

## Task 1

#### **Weekly User Engagement:**

- A. Objective: Measure the activeness of users on a weekly basis.
- B. Your Task: Write an SQL query to calculate the weekly user engagement.

For this, we need to get the weeks first and then collate the data group by weeks We also need to consolidate all the events from event and email\_events tables

Project by – Alokk Joshi

I used UNION ALL for consolidating all the events from events and email\_events tables

```
(select user_id, occurred_at from events
union all
select user_id, occurred_at from email_events
) as tota_events
```

Project by – Alokk Joshi

Used the YEAR and WEEK function to extract years and weeks from occurred\_at column Count(distinct user\_id) will return the count of active users for respective weeks And then ran the below query

```
select
year(occurred_at) as Years,
week(occurred_at, 1) as WeekNumber,
count(distinct user_id) as active_users
from
     (select user_id, occurred_at from events
     union all
     select user_id, occurred_at from email_events
    ) as tota_events
group by Years, WeekNumber
order by Years, WeekNumber;
```

### Project by – Alokk Joshi

### Below is the outcome

	Years	WeekNumber	active_users
١	2014	1	684
	2014	2	1835
	2014	6	2416
	2014	10	1777
	2014	14	828
	2014	15	2043
	2014	19	2539
	2014	20	2386
	2014	21	3082
	2014	22	3140
	2014	23	2324
	2014	24	1011
	2014	25	3468
	2014	26	3556
	2014	27	1848

Result Grid	Filte	r Rows:
Years	WeekNumber	active_users
2014	28	2129
2014	29	3880
2014	30	3987
2014	31	3494
2014	32	2402
2014	33	2708
2014	34	4458
2014	35	4577
2014	36	2383
2014	37	182
2014	40	142
2014	41	1669
2014	45	2755
2014	49	1781
2014	50	975

Project by – Alokk Joshi

MinUserWeek

# With subqueries and CTE we can also find in which weeks we had max and min active users respectively

```
select
  year(occurred at) as Years,
  week(occurred_at, 1) as WeekNumber,
  count(distinct user id) as active users
                                                                           Result Grid
                                                                                              Filter Rows:
  from
                                                                               MaxActiveUsers
                                                                                                                    MinActiveUsers
                                                                                                  MaxUserWeek
      (select user id, occurred at from events
      union all
                                                                              4577
                                                                                                  35
                                                                                                                   142
      select user_id, occurred_at from email_events
      ) as tota events
  group by Years, WeekNumber
  order by Years, WeekNumber
  select
  Max(active users) as MaxActiveUsers,
  (select WeekNumber from maxmin where active users = (select max(active users) from maxmin)) as MaxUserWeek,
  Min(active_users) as MinActiveUsers,
  (select WeekNumber from maxmin where active users = (select min(active users) from maxmin)) as MinUserWeek
  from maxmin;
```

## Task 2

#### **User Growth Analysis:**

- A. Objective: Analyze the growth of users over time for a product.
- B. Your Task: Write an SQL query to calculate the user growth for the product.

For this we will need to check how many users were activated over a period of time

We will check it month wise as well as year wise

Project by – Alokk Joshi

#### Month wise user growth

```
select
year(activated_at) as Years,
month(activated_at) as Months,
count(user_id) as Users,
lag(count(user_id)) over (order by year(activated_at), month(activated_at)) as Prev_Month_Users,
count(user_id) - lag(count(user_id)) over(order by year(activated_at), month(activated_at)) as Growth
from users
group by Years, Months
order by Years, Months;
```

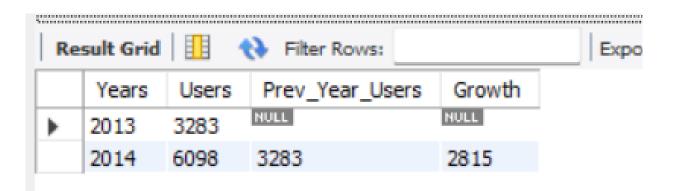
Re	esult Grid		Filter R	OWS:	Exp
	Years	Months	Users	Prev_Month_Users	Growth
•	2013	1	160	NULL	HULL
	2013	2	160	160	0
	2013	3	150	160	-10
	2013	4	181	150	31
	2013	5	214	181	33
	2013	6	213	214	-1
	2013	7	284	213	71
	2013	8	316	284	32
	2013	9	330	316	14
	2013	10	390	330	60
	2013	11	399	390	9
	2013	12	486	399	87

2014	1	552	486	66
2014	2	525	552	-27
2014	3	615	525	90
2014	4	726	615	111
2014	5	779	726	53
2014	6	873	779	94
2014	7	997	873	124
2014	8	1031	997	34

Project by – Alokk Joshi

#### year wise user growth

```
select
year(activated_at) as Years,
count(user_id) as Users,
lag(count(user_id)) over (order by year(activated_at)) as Prev_Year_Users,
count(user_id) - lag(count(user_id)) over(order by year(activated_at)) as Growth
from users
group by Years
order by Years;
```



-27

```
with Gro AS
                                                                                                            roject by – Alokk Joshi
⊖ (select
  year(activated at) as Years,
  month(activated at) as Months,
  count(user id) as Users,
  lag(count(user id)) over (order by year(activated at), month(activated at)) as Prev Month Users,
  count(user id) - lag(count(user id)) over(order by year(activated at), month(activated at)) as Growth
  from users
  group by Years, Months
  order by Years, Months
                                                                                                         Max & Min growth of
                                                                                                         users in respective years
                                                                                                         and month
  select
  (select Years from Gro where Growth = (select max(Growth) from Gro)) as MaxGrowthYear,
  (Select Months from Gro where Growth = (select max(Growth) from Gro)) as MaxGrowMonth,
  Max(Growth) as MaxGrowthUsers,
  (select Years from Gro where Growth = (select min(Growth) from Gro)) as MinGrowthYear,
  (select Months from Gro where Growth = (select min(Growth) from Gro)) as MinGrowMonth,
  Min(Growth) as MinGrowthUsers
  from Gro;
                                                 Result Grid Filter Rows:
                                                                                      Export: Wrap Cell Content: IA
                                                    MaxGrowthYear
                                                                 MaxGrowMonth
                                                                               MaxGrowthUsers
                                                                                             MinGrowthYear
                                                                                                          MinGrowMonth
                                                                                                                       MinGrowthUsers
```

124

2014

2014

## Task 3

#### **Weekly Retention Analysis:**

- A. Objective: Analyze the retention of users on a weekly basis after signing up for a product.
- B. Your Task: Write an SQL query to calculate the weekly retention of users based on their sign-up cohort.

Project by – Alokk Joshi

Since, there are id creations from years 2013 and 2014
We will check the yearly and monthly user creation and retention as well and after that week too

#### Yearly and monthly retention

```
SELECT
    month(created_at) AS signup_month,
    year(created_at) as signup_year,
    COUNT(DISTINCT u.user_id) AS total_signed_up,
    COUNT(DISTINCT e.user_id) AS active_users,
    round ((count(distinct e.user_id) * 100 / count(distinct u.user_id)), 2) as Ret_Percentage
FROM users u
left JOIN events e ON u.user_id = e.user_id
GROUP BY signup_year, signup_month;
```

## Project by – Alokk Joshi

	signup_month	signup_year	total_signed_up	active_users	Ret_Percentage
•	1	2013	160	57	35.63
	2	2013	160	70	43.75
	3	2013	150	67	44.67
	4	2013	181	83	45.86
	5	2013	214	84	39.25
	6	2013	213	84	39.44
	7	2013	284	108	38.03
	8	2013	316	130	41.14
	9	2013	330	132	40.00
	10	2013	390	146	37.44
	11	2013	399	159	39.85
	12	2013	486	178	36.63

1	2014	552	213	38.59
2	2014	525	218	41.52
3	2014	615	271	44.07
4	2014	726	462	63.64
5	2014	779	779	100.00
6	2014	873	873	100.00
7	2014	997	997	100.00
8	2014	1031	1031	100.00



Project by – Alokk Joshi

```
week(created_at, 1) AS signup_week,
    COUNT(DISTINCT u.user_id) AS total_signed_up,
    COUNT(DISTINCT e.user_id) AS active_users,
    round ((count(distinct e.user_id) * 100 / count(distinct u.user_id)), 2) as Ret_Percentage
FROM users u
left JOIN events e ON u.user_id = e.user_id
GROUP BY signup_week;
```

Res	sult Grid	Name of the Filter Rows:		Export: V	 	<b>(*</b>			Ne	suit aria   🔡	Name of the Filter Rows:		Export:	Wrap
	signup_week	total_signed_up	active_users	Ret_Percentage	signup_week	total_signed_up	active_users	Ret_Percentage		signup_week	total_signed_up	active_users	Ret_Percentage	
•	1	117	39	33.33	19	205	181	88.29		35	345	300	86.96	_
	2	151	53	35.10	20	241	210	87.14		36	65	24	36.92	
	3	159	64	40.25	21	218	190	87.16		37	71	31	43.66	
	4	149	55	36.91	22	235	198	84.26		38	84	31	36.90	
	5	160	73	45.63	23	248	217	87.50		39	92	36	39.13	
	6	180	83	46.11	24	249	220	88.35		40	81	34	41.98	
	7	176	69	39.20	25	268	241	89.93		41	88	33	37.50	
	8	166	71	42.77	26	267	231	86.52		42	74	24	32.43	
	9	160	60	37.50	27	256	217	84.77		43	97	35	36.08	
	10	178	72	40.45	28	275	241	87.64		44	92	36	39.13	
	11	185	83	44.86	29	286	243	84.97		45	97	39	40.21	
	12	164	73	44.51	30	294	255	86.73		46	94	40	42.55	
	13	184	84	45.65	31	303	261	86.14		47	82	31	37.80	
	14	201	98	48.76	32	255	216	84.71		48	103	44	42.72	
	15	201	110	54.73	33	323	279	86.38		49	96	42	43.75	
	16	207	118	57.00	34	330	291	88.18		50	117	41	35.04	
	17	224	140	62.50	35	345	300	86.96		51	123	42	34.15	

## Task 4

#### **Weekly Engagement Per Device:**

- A. Objective: Measure the activeness of users on a weekly basis per device.
- B. Your Task: Write an SQL query to calculate the weekly engagement per device.

Before going ahead let check how users are using devices to engage.

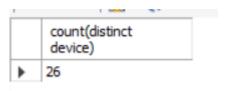
It shows that maximum activity is happening on macbook pro

device,
COUNT(\*) AS TotalActivity,
count(distinct user\_id) as Users
FROM events
GROUP BY device
ORDER BY TotalActivity desc limit 10;

And there are total 26 devices on which users do activity

select count(distinct device) from events;

Re	sult Grid 🔢 🙌 Filb	er Rows:	
	device	TotalActivity	Users
•	macbook pro	57295	1952
	lenovo thinkpad	36978	1309
	macbook air	26786	950
	iphone 5	25883	1025
	dell inspiron notebook	19669	677
	samsung galaxy s4	18653	803
	nexus 5	16502	621
	iphone 5s	15929	626
	dell inspiron desktop	10141	360
	iphone 4s	9615	409



### Project by – Alokk Joshi

#### Query for weekly device wise engagement is as below

SELECT
<pre>week(occurred_at, 1) As WeekActivity,</pre>
device,
COUNT(*) AS TotalActivity,
<pre>count(distinct user_id) as Users</pre>
FROM events
GROUP BY device, WeekActivity
ORDER BY WeekActivity;

This is Week 1 data
And likewise it goes for all weeks
For all devices

Re	esult Grid	Name of the Property of the Pr	E	export:
	WeekActivity	device	TotalActivity	Users
١	1	acer aspire desktop	38	5
	1	acer aspire notebook	67	10
	1	amazon fire phone	29	2
	1	asus chromebook	95	9
	1	dell inspiron desktop	51	8
	1	dell inspiron notebook	221	22
	1	hp pavilion desktop	45	4
	1	htc one	37	5
	1	ipad air	93	8
	1	ipad mini	47	6
	1	iphone 4s	116	10
	1	iphone 5	240	22
	1	iphone 5s	157	18
	1	kindle fire	12	2
	1	lenovo thinkpad	306	40
	1	mac mini	40	4
	1	macbook air	194	22
	1	macbook pro	518	51
	1	nexus 10	31	4
	1	nexus 5	101	13

1	Result Grid	N Filter Rows:	Export:		
	WeekActivity	device	TotalActivity	Users	
	1	nexus 10	31	4	
	1	nexus 5	101	13	
	1	nexus 7	66	6	
	1	nokia lumia 635	27	3	
	1	samsumg galaxy tablet	39	4	
	1	samsung galaxy note	48	5	
	1	samsung galaxy s4	245	31	
	1	windows surface	45	4	

## Task 5

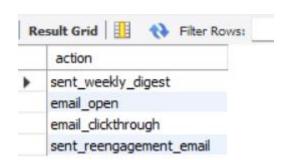
#### **Email Engagement Analysis:**

- A. Objective: Analyze how users are engaging with the email service.
- B. Your Task: Write an SQL query to calculate the email engagement metrics.

Lets first check how many actions users are taking on email service

select distinct action from email\_events;

There are 4 actions that users are taking on email



Project by – Alokk Joshi

#### Which activities are done the most

```
select
action as Actions,
Count(*) as ActionFrequency,
round(count(*) * 100 / (select count(*) from email_events), 2) as ActPercentage
from email_events
group by Actions
order by ActionFrequency desc;
```

Users do 'Sent\_weekly\_digest' activity the most As high as 63%

And the lest usage is of 'Sent\_reengagemet\_email' upto just 4.04%

	Actions	ActionFrequency	ActPercentage
•	sent_weekly_digest	57267	63.36
	email_open	20459	22.63
	email_clickthrough	9010	9.97
	sent_reengagement_email	3653	4.04

Project by – Alokk Joshi

#### Mail opening to clickthrough ratio

It shows how users interact with emails 44% activity happens for clickthrough out of 100% email events

Project by – Alokk Joshi

#### **Insights - What I Learned from the Project**

- •Understanding Data with SQL I learned how to write SQL queries to analyze different types of data, such as job reviews, user activity, and email engagement.
- •Tracking Trends Over Time Instead of just looking at daily numbers, I used techniques like the 7-day rolling average to get a better understanding of trends. This helped smooth out sudden spikes and dips.
- •Investigating Unusual Activity
  I found that daily data can be unpredictable, so analyzing weekly engagement gave a clearer picture of user behavior.
- •User Growth and Retention
  I saw how users sign up over time and how many of them stay active. Retention analysis helped me understand when users stop using a service.

Project by – Alokk Joshi

#### Results - What I Achieved and How It Helped

•Improved SQL Skills

I learned how to clean data, join multiple tables, and calculate insights efficiently using SQL.

Better Understanding of User Behavior

By analyzing weekly engagement and retention, I now understand how to measure user activity over time and what factors influence their continued usage.

- Practical Business Insights
  - The 7-day rolling average is more reliable for analyzing job reviews than daily metrics.
  - Users are most active on MacBook Pro among all devices.
  - The most common email activity is reading weekly digest emails.
  - User growth varies monthly and yearly, and retention drops over time.

#### Decision-Making Power

This analysis can help businesses improve email marketing strategies, optimize device experiences, and focus on retaining users.

Project by – Alokk Joshi

#### Device-Based Engagement

I found that users are active on multiple devices, with **MacBook Pro** being the most used device. This kind of insight can help in making decisions about improving user experience.

#### Email Engagement Behavior

The most common email activity was "Sent\_Weekly\_Digest", which made up 63% of all email interactions. The least common was "Sent\_Reengagement\_Email" at only 4.04%. This shows that most users engage with regular updates but are less responsive to reengagement emails.

#### Duplicate Data Issues

There were no completely duplicate rows in the job data, but some values like job\_id and actor\_id appeared multiple times with different details.