Business problem and details of dataset

The task is to analyze Twitter usage patterns among different age groups to understand how younger and older generations engage with the platform. The goal is to uncover insights that can inform marketing strategies, user engagement plans, and product development.

The dataset includes:

1. Tweets: This table comprises user_id , tweet

2. Users: This table includes user_id , username , and user

3. Followers: This table contains user_id and Requirements: Link. timestamp , and tweet text . age . follower_id pairs

Platform used

Deepnote

Upload datasets

The three datasets were dragged and dropped into the file section of Deep note. Since the SQL can not directly interact with CSV files, therefore, tables were created and CSV files were fed into them.

Creating tables

3 tables - users, tweets and followers were created.

The tables were created using the below format:

```
CREATE TABLE table_name (
   column1 datatype constraint,
   column2 datatype constraint,
   column3 datatype constraint,
   ....
);
```

Appropriate constraints were added while creating the tables to ensure the accuracy and reliability of data.

```
CREATE TABLE users (
    id int NOT NULL, /*adding constraints. It ensures that column can never be null*/
    user varchar(255),
    age varchar(255),
    PRIMARY KEY (id) /*to uniquely identify each record in a table. It can not be null and must be unique*/
)
```

	Count int64

0 rows, 1 col, showing 10 rows/page Page 0 of 0

```
CREATE TABLE tweets (
    id int NOT NULL , /*adding constraints*/ /*It was not made a primary key because one id can
                        do multiple tweets*/
    timestamp timestamp, /*data type is timestamps as it is in the format of YYYY-MM-DD HH:MI:SS*/
    text TEXT, /* holds long texts*/
)
```

	Count int64

0 rows, 1 col, showing 10 rows/page Page 0 of 0

```
CREATE TABLE followers(
    id int NOT NULL,
    following int,
)
```

	Count int64

```
0 rows, 1 col, showing 10 rows/page    Page 0 of 0
```

Inserting data to the tables

The format used on this platform of DeepNote is as follows:

COPY table_name(column1, column2......)
FROM '<path to csv file'
DELIMITER ','
CSV HEADER;

```
COPY users(id, user, age)
FROM '/work/users.csv'
DELIMITER ','
CSV HEADER;
```

	Count int64
0	31185

```
1 row, 1 col, showing 10 rows/page    Page 1 of 1
```

```
COPY tweets(id, timestamp, text)
FROM '/work/tweets.csv'
DELIMITER ','
CSV HEADER;
```

	Count int64
0	31217

```
1 row, 1 col, showing 10 rows/page    Page 1 of 1
```

```
COPY followers(id, following)
FROM '/work/followers.csv'
DELIMITER ','
CSV HEADER;
```

	Count int64
0	104668

```
1 row, 1 col, showing 10 rows/page    Page 1 of 1
```

Describing tables and displaying first 5 rows

Tables were described using the keyword 'describe'. The output tells us about the column name, column datatype, if it allows null values or not, if it is primary key or foreign or none, any default value, and any extra value.

The first 5 rows were displayed using the the keyword 'limit' which only displays 5 rows.

```
describe users
```

	column_name object	column_type object	null object	key object	default object	extra float64
0	id	INTEGER	NO	PRI	None	nan
1	user	VARCHAR	YES	None	None	nan
2	age	VARCHAR	YES	None	None	nan

```
3 rows, 6 cols, showing 10 rows/page    Page 1 of 1
```

```
select * from users limit 5
```

	id int32	user object	age object
0	8653	hpfangirl94	old
1	12020	HybridMink	young
2	23858	driveaway2008	old
3	51844	lennytoups	old
4	52341	Jemimus	young

5 rows, 3 cols, showing 10 rows/page Page 1 of 1

```
describe tweets
```

	column_name object	column_type object	null object	key object	default object	extra float64
0	id	INTEGER	NO	None	None	nan
1	timestamp	TIMESTAMP	YES	None	None	nan
2	text	VARCHAR	YES	None	None	nan

3 rows, 6 cols, showing 10 rows/page Page 1 of 1

```
select * from tweets limit 5
```

	id int32	timestamp datetim...	text object
0	8653	2009-04-06 21:2...	falling asleep. jus...
1	12020	2009-04-06 21:2...	i have a sad feeli...
2	23858	2009-04-06 21:2...	@statravelau just...
3	51844	2009-04-06 21:3...	@djalizay i really ...
4	52341	2009-04-06 21:3...	my mind and bod...

5 rows, 3 cols, showing 10 rows/page Page 1 of 1

```
describe followers
```

	column_name object	column_type object	null object	key object	default object	extra float64
0	id	INTEGER	NO	None	None	nan
1	following	INTEGER	YES	None	None	nan

2 rows, 6 cols, showing 10 rows/page Page 1 of 1

```
select * from followers limit 5
```

	id int32	following int32
0	210569926	8653
1	709656306	8653
2	496010887	12020
3	544857586	12020
4	576431861	12020

5 rows, 2 cols, showing 10 rows/page Page 1 of 1

Checking for null values

We aim to count the number of rows from tables where there are any null values. This can be done at the level of entire table or column wise. Both the ways are discussed.

```sql
Select count(*) from users
where users is null
```

	count_star() int64
0	0

1 row, 1 col, showing 10 rows/page Page 1 of 1

```sql
select count(*) from users
where id is null OR user is null or age is null
```

	count_star() int64
0	0

1 row, 1 col, showing 10 rows/page Page 1 of 1

```sql
Select count(*) from tweets
where tweets is null
```

	count_star() int64
0	0

1 row, 1 col, showing 10 rows/page Page 1 of 1

```sql
select count(*) from tweets
where id is null OR timestamp is null or text is null
```

	count_star() int64
0	0

1 row, 1 col, showing 10 rows/page Page 1 of 1

```sql
Select count(*) from followers
where followers is null
```

	count_star() int64
0	0

1 row, 1 col, showing 10 rows/page Page 1 of 1

```sql
select count(*) from followers
where id is null OR following is null
```

	count_star() int64
0	0

1 row, 1 col, showing 10 rows/page Page 1 of 1

We conclude that there are no null values in any of the tables.

Data analysis

Tweet count by age group

For the above query we consider the tables user and tweets. The users table has id,user and age. The tweets table has id, timestamp and text. The id column is the common between these two tables.

Therefore, to get the number of tweets age wise, we need to inner join (because only the common ids would be considered for counting) two mentioned tables on id column, count the text column and group by age. We further order the output in descending order using the 'order by' and 'desc'.

The format for inner join is:

SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;

```sql
select age, count(text) as tweet_counts /*alias is used*/
from users
inner join tweets
on users.id=tweets.id
GROUP BY age
ORDER BY tweet_counts desc
```

	age object	tweet_counts int64
0	young	18003
1	old	13214

2 rows, 2 cols, showing 10 rows/page Page 1 of 1

Young people tweet more as compared to old people

Number of followers by age group

We consider the tables followers and users.

Followers has id and following and users has id, user, age as columns. The two tables can be inner joined on id column, grouped by age, and order by average of number of followers.

To see the number of followers, we counted the 'following' column from followers table as it would give the person who has been followed and by how many.

```sql
SELECT age, count(followers.following) AS avg_followers
FROM users
inner join followers
on users.id=followers.id
GROUP BY age
ORDER BY avg_followers DESC;
```

	age object	avg_followers int64
0	young	68342
1	old	36326

2 rows, 2 cols, showing 10 rows/page Page 1 of 1

Top 10 hashtags

To achieve this, the text column in tweets data set was divided into words and words starting with # were counted and sorted in decending order.

The subquery SELECT unnest(string_to_array(t.text, ' ')) AS parts extracts individual words from the text field in tweets by splitting the text on spaces and unnesting the resulting array into separate rows.

```
SELECT
  parts,
  COUNT(*) AS count
FROM (
  SELECT
    id,
    unnest(string_to_array(text, ' ')) AS parts
  FROM tweets
)
WHERE parts LIKE '#%'
GROUP BY parts
ORDER BY count DESC
LIMIT 10;
```

	parts object	count int64
	#musicmond... 10% #music 10% 8 others 80%	11 - 397
0	#musicmonday	397
1	#music	68
2	#followfriday	45
3	#fb	43
4	#familyforce5	32
5	#iranelection	27
6	#travel	15
7	#ff	14
8	#myweakness	13
9	#	11

10 rows, 2 cols, showing 10 rows/page Page 1 of 1

The top 10 hashtags used by younger users.

To count the hashtags, we used tweets table.

we need to extract hashtags from tweets.text. For this we had to read the text column from tweets table, search for words starting with '#' through regular expression matching.

To group it by age group, we inner joined with users table on id.

Make a temporary table which will bifurcate the words in the tweets.text column and count the words starting with #. This temporary table was queried for the count of top 10 hashtag used by young people

```sql
-- Step 1: Create a temporary table for the parts
CREATE TEMP TABLE temp_parts AS
SELECT
  unnest(string_to_array(t.text, ' ')) AS parts
FROM tweets t
INNER JOIN users u ON t.id = u.id
WHERE u.age = 'young';

-- Step 2: Query the temporary table
SELECT
  parts,
  COUNT(*) AS count
FROM temp_parts
WHERE parts LIKE '#%'
GROUP BY parts
ORDER BY count DESC
LIMIT 10;
```

	parts object	count int64
	#musicmond... 10% #music 10% 8 others 80%	8 - 397
0	#musicmonday	397
1	#music	68
2	#iranelection	26
3	#followfriday	26
4	#myweakness	12
5	#fb	11
6	#music4good	11
7	#	9
8	#inappropriatemo...	8
9	#1	8

10 rows, 2 cols, showing 10 rows/page Page 1 of 1

Whether young or older users generally follow users in their same age group.

We used followers table to find of the id and following belong to same age group. If yes then it was considered as 1 and number of instances of 1s were averaged.

```sql
SELECT
  CASE
    WHEN u1.age = 'young' THEN 'Younger'
    ELSE 'Older'
  END AS age_group,
  AVG(CASE WHEN u1.age = u2.age THEN 1 ELSE 0 END) AS same_age_group_follow_rate
FROM users u1
JOIN followers f ON u1.id = f.id
JOIN users u2 ON f.following = u2.id
GROUP BY age_group;
```

	age_group object	same_age_group_...
0	Older	0.929664703
1	Younger	0.9103772205

2 rows, 2 cols, showing 10 rows/page Page 1 of 1

We observe that more older people follow users in their own age group.

Most Active Users by Age Group

The aim is to find the user_id with maximum tweets. We used tweets and user tables.

```sql
SELECT
  u.id,
  age,
  COUNT(t.id) AS tweet_count
FROM users u
LEFT JOIN tweets t ON u.id = t.id
GROUP BY u.id, u.age
ORDER BY tweet_count DESC
LIMIT 10;
```

	id int32 90896557 - 49860...	age object young 60% old 40%	tweet_count int64 2 - 2
0	90896557	young	2
1	208383833	young	2
2	208500675	young	2
3	217899625	old	2
4	366219601	old	2
5	366936762	young	2
6	210945402	young	2
7	413288278	young	2
8	497791396	old	2
9	498601874	old	2

10 rows, 3 cols, showing 10 rows/page Page 1 of 1

Most Common Words in Tweets by Older Users

From the tweets table, the text column was divided into words and words were counted and sorted in descending order.

```sql
-- Step 1: Create a temporary table for the parts
CREATE TEMP TABLE temp_parts_old AS
SELECT
  unnest(string_to_array(t.text, ' ')) AS parts
FROM tweets t
INNER JOIN users u ON t.id = u.id
WHERE u.age = 'old';

-- Step 2: Query the temporary table
SELECT
  parts,
  COUNT(*) AS count
FROM temp_parts
GROUP BY parts
ORDER BY count DESC
LIMIT 10;
```

	parts object 10% i 10% 8 others 80%	count int64 2912 - 22208
0		22208
1	i	9065
2	to	8209
3	the	8105
4	music	6403
5	and	4702
6	a	4159
7	my	3943
8	of	2920
9	is	2912

10 rows, 2 cols, showing 10 rows/page Page 1 of 1

Average Number of Tweets Per User by Age Group

```
SELECT
  u.age,
  AVG(user_tweet_count.tweet_count) AS average_tweets_per_user
FROM users u
INNER JOIN (
  SELECT
    t.id,
    COUNT(*) AS tweet_count
  FROM tweets t
  GROUP BY t.id
) AS user_tweet_count ON u.id = user_tweet_count.id
GROUP BY u.age;
```

	age object	average_tweets_p...
0	old	1.000833144
1	young	1.001167835

2 rows, 2 cols, showing 10 rows/page Page 1 of 1

We observe that there is not much difference in the number of tweets done by both old and young people.