SECOND EXERCISE – SQL Queries

How do the final datasets look like?

Users and Transactions

```
[394] users_transact_inner = pd.merge(transact_final, users_final, left_on='USER_ID', right_on='ID', how='inner')
      users_transact_inner.shape
```

```
(130, 14)
```

Number of common records between Users and Transactions

```
users_transact_left = pd.merge(transact_final, users_final, left_on='USER_ID', right_on='ID', how='left')
users_transact_left.shape
```

```
(24852, 14)
```

Left Join of Users and Transactions

It was found that there are only 130 common records (might contain recurring user ids) when Users and Transactions were joined using an inner join. It was observed that seen that there were plenty of users who made purchases, but their records were missing from the users table. That would adversely affect the business analyses which could be drawn from the users data based on the transactions made because, a lot of valid transactions would go unaccounted for because of the missing users data!

Products and Transactions

After performing an inner join it was found that more than half of the transactions have been missed after the join. This means that there were a lot of products that were bought that didn't have records in the products table.

```
product_transact_inner = pd.merge(transact_final[transact_final['BARCODE'].notna()],
                                  product_final2[product_final2['BARCODE'].notna()], left_on='BARCODE', right_on='BARCODE', how='inner')
product_transact_inner.shape
```

```
(12363, 14)
```

Inner Join of Products and Transactions

SQL Answers

Question 1:

What are the top 5 brands by receipts scanned among users 21 and over?

```
SELECT p.brand, COUNT(t.receipt_id) AS receipt_count

FROM users u

LEFT JOIN transactions t ON u.u_id = t.user_id

LEFT JOIN products p ON t.barcode = p.barcode

WHERE DATE_PART ('year', AGE(t.scan_date, u.birth_date)) >= 21  AND p.brand IS NOT NULL

GROUP BY p.brand

ORDER BY receipt_count DESC

LIMIT 5;
```

Explanation:

Firstly, the Transactions table and Users table were joined to obtain the users who made transactions then the resultant data was joined with the Products data to get the brands that were a part of the transactions. The data was filtered by using the AGE query on the scan_date and the birth_date to consider only the users who are above 21 years of age. The top brands were obtained by calculating the number of receipt_ids as receipt_count and arranging them in a descending order according to each brand. The top 5 brands according to the number of receipts are:

	brand character varying	receipt_count bigint
1	DOVE	3
2	NERDS CANDY	3
3	HERSHEY'S	2
4	COCA-COLA	2
5	GREAT VALUE	2

The number is quite low because the analysis is carried out according to the users' data and like discussed earlier, a lot of user_ids present in the transactions are missing in the Users table. So, there are a lot of users who made legitimate purchases, but their personal data is not available for this analysis which requires the users' age.

Question 2:

What are the top 5 brands by sales among users that have had their account for at least six months?

```
SELECT p.brand, SUM(t.final_sale * final_quantity) AS total_sales

FROM users u

RIGHT JOIN transactions t ON u.u_id = t.user_id
```

```
INNER JOIN products p ON t.barcode = p.barcode

WHERE t.purchase_date >= u.created_date + INTERVAL '6 months'

GROUP BY p.brand

ORDER BY total_sales DESC

LIMIT 5;
```

Explanation:

Like in the previous problem, Transactions table and Users table were joined to obtain the users who made transactions then the resultant data was joined with the Products data to get the brands that were a part of the transactions. The data was filtered according to the purchase_date and created date to only consider the users who have had their account for at least 6 months. The top brands were obtained by multiplying final_sale and final_quantity from the Transactions table and storing it as Total_sales and arranging them in a descending order according to each brand. The top 5 brands according to the Total_sales are:

	brand character varying	total_sales numeric
1	CVS	72.00
2	DOVE	30.910
3	TRESEMMÉ	29.160
4	TRIDENT	23.360
5	COORS LIGHT	17.480

Just like the scenario before, there were many users whose transactions could not be included in the above analysis because their personal data was absent in the Users table.

Question 3:

What is the percentage of sales in the Health & Wellness category by generation?

Assumption – The users were categorized into to 5 generations according to the birthdate – Silent Generation, Baby Boomer, Generation X, Millennial, Generation Z and Others.

```
WITH user_generation AS (

SELECT u_id,

    CASE

        WHEN birth_date::DATE BETWEEN DATE '1900-01-01' AND DATE '1945-12-31' THEN 'Silent Generation'

        WHEN birth_date::DATE BETWEEN DATE '1946-01-01' AND DATE '1964-12-31' THEN 'Baby Boomer'

        WHEN birth_date::DATE BETWEEN DATE '1965-01-01' AND DATE '1980-12-31' THEN 'Generation X'

        WHEN birth_date::DATE BETWEEN DATE '1981-01-01' AND DATE '1996-12-31' THEN 'Millennial'

        WHEN birth_date::DATE >= DATE '1997-01-01' THEN 'Generation Z'
```

```
        ELSE 'Other'
END AS generation
FROM users
),
category_sales AS (
SELECT u.generation, SUM(CAST(t.final_sale AS DECIMAL)) AS total_sales,
            SUM(CASE WHEN p.category_1 = 'Health & Wellness'
           THEN CAST(t.final_sale AS DECIMAL) ELSE 0 END) AS health_wellness_sales
FROM transactions t
LEFT JOIN products p ON t.barcode = p.barcode
JOIN user_generation u ON t.user_id = u.u_id
WHERE t.barcode IS NOT NULL OR p.brand IS NOT NULL
GROUP BY u.generation
)
SELECT generation,
              COALESCE((health_wellness_sales / NULLIF(total_sales, 0)) * 100, 0)
              AS health_wellness_percentage
FROM category_sales
ORDER BY health_wellness_percentage DESC;
```

Explanation:

The code consists of 2 CTEs. The first CTE categorizes the users into generations depending on the birthdates. In the second CTE, Transactions table and Users table were joined to obtain the users who made transactions then the resultant data was joined with the Products data to get the brands that were a part of the transactions and the total number of sales along with the total number of sales in the 'Health and Wellness' category are calculated and grouped by generations. Finally, the percentage of health and wellness sales are calculated and grouped by generations. The output is given below:

	generation text	health_wellness_percentage numeric
1	Baby Boomer	40.24174802417480241700
2	Generation X	24.05981546293350302300
3	Millennial	22.02668002755683597400
4	Generation Z	0.00000000000000000000
5	Other	0.0000000000000000
6	Silent Generation	0.00000000000000000000

Question 4:

Who are Fetch's power users?

Assumption - Fetch's power users were defined according to three assumed criteria while keeping in mind the nature of the data. They are:

1) Transaction Count > = 5 – Transaction count is defined as the total number of transactions per user from the time of account creation till the purchase date of the transaction according to the number of receipt_ids. The limit is chosen as 5
2) Total amount Spent > 30 – Total amount spent is the amount of money spent on purchases per user. The lower limit is set at $30.
3) The user must have made at least 1 purchase in the last 6 months.

```
WITH user_activity AS (
SELECT t.user_id, COUNT(t.receipt_id) AS transaction_count,
    SUM(t.final_sale * t.final_quantity) AS total_spent,
    MAX(t.purchase_date) AS last_purchase_date
FROM transactions t
LEFT JOIN users u ON u.u_id = t.user_id
WHERE (t.purchase_date >= u.created_date OR u.created_date IS NULL)
   AND t.barcode IS NOT NULL            -- Exclude transactions with missing barcode
GROUP BY t.user_id
)
SELECT ua.user_id, ua.transaction_count, ua.total_spent
FROM user_activity ua
WHERE ua.transaction_count >= 5                                    -- Setting the criterion for the number of transactions
   AND ua.total_spent > 30                                         -- Setting the criterion for the total amount spent
```

```
AND ua.last_purchase_date >= NOW() - INTERVAL '6 months'-- Setting the criterion for activity
ORDER BY ua.transaction_count DESC, ua.total_spent DESC;
```

Explanation:

The code consists of a CTE where the receipt_ids are counted, total amount spent is calculated and the last purchase date is extracted and grouped according to each user while ignoring the missing barcode values in the transactions data. The second part of the code consists of the above-mentioned conditions to filter out the users and arrange them according to the transaction count and then by the total amount spent. Here are the power users of Fetch according to the criteria mentioned above:

	user_id character varying	transaction_count bigint	total_spent numeric
1	64063c8880552327897186a5	9	30.610
2	60a5363facc00d347abadc8e	6	1180.060
3	5d77d06b0d1bff4316a3ef47	6	67.770
4	653a0f40909604bae9071473	6	50.740
5	605a982894a5c74ba439e5ab	6	41.890
6	5eb59d6be7012d13941af5e2	6	34.990
7	63f1904938f010745b9a2b60	6	33.020
8	62d861e81d76344f1a35fa02	5	115.880
9	619c6018f998e47aad318354	5	38.430
10	653d691d909604bae9076d1a	5	35.220
11	6240f64ee073a81bcca57670	5	34.650
12	62099fee0ede521c349b0cb3	5	34.400
13	65d4915916cc391732127174	5	34.270
14	61545c3b504f3536dc3423b6	5	32.510
15	649d0d38127ddb5d7f0182b8	5	31.690

Question 5:

Which is the leading brand in the Dips & Salsa category?

Before writing the SQL code, the data was investigated and was cleaned properly to get the best results possible. Firstly, it was found that there were 6329 products in the ‘Dips & Salsa’ category that had missing brand information.

```
#Finding the break down of the missing values of brands in the dips and salsa products
product_dands['BRAND'].isna().value_counts()
```

BRAND	count
False	9939
True	6329

Number of records where the brand is missing

Then, to impute the missing brand information using a credible logic, here is the method that was followed:

1) Firstly, the brand breakdown was checked for the products with the known brands in the Dips and Salsa category.

```
# Checking the list of brands and their frequency
dands_brand_nona['BRAND'].value_counts().head(30)
```

BRAND	
PRIVATE LABEL	1166
SABRA	268
WHOLLY	217
TOSTITOS	149
PACE	119
GOOD FOODS	115
MEIJER	112
MARZETTI	109
BRAND NOT KNOWN	108
ROBERT ROTHSCHILD	107

Brand breakdown of the products in the Dips and Salsa category

2) Then one brand is chosen and the details of all the products in that brand are explored.
Aim: The aim is to notice some pattern or sequence in the product details of the same brand. For example, all the barcodes of a particular brand in the dips and salsa category might have the same last 4 digits.

```
#Let us check the details of the products belonging to the "WHOLLY" brand
n = dands_brand_nona[dands_brand_nona['BRAND']=='WHOLLY']
n.head(10)
```

	CATEGORY_1	CATEGORY_2	CATEGORY_3	CATEGORY_4	MANUFACTURER	BRAND	BARCODE	barcode_is_12_digit
1374	Snacks	Dips & Salsa	Guacamole	NaN	HORMEL FOODS	WHOLLY	616112279526	True
4641	Snacks	Dips & Salsa	Guacamole	NaN	HORMEL FOODS	WHOLLY	616112681510	True
5781	Snacks	Dips & Salsa	Guacamole	NaN	HORMEL FOODS	WHOLLY	016112039210	True
10458	Snacks	Dips & Salsa	Guacamole	NaN	HORMEL FOODS	WHOLLY	616112027868	True
11391	Snacks	Dips & Salsa	Guacamole	NaN	HORMEL FOODS	WHOLLY	616112329535	True
13738	Snacks	Dips & Salsa	Guacamole	NaN	HORMEL FOODS	WHOLLY	616112028551	True
18343	Snacks	Dips & Salsa	Guacamole	NaN	HORMEL FOODS	WHOLLY	616112031544	True
22085	Snacks	Dips & Salsa	Guacamole	NaN	HORMEL FOODS	WHOLLY	700718414648	True
28261	Snacks	Dips & Salsa	Guacamole	NaN	HORMEL FOODS	WHOLLY	616112029381	True

	CATEGORY_1	CATEGORY_2	CATEGORY_3	CATEGORY_4	MANUFACTURER	BRAND	BARCODE	barcode_is_12_digit
1337	Snacks	Dips & Salsa	Hummus	NaN	PEPSICO	SABRA	040822344708	True
1573	Snacks	Dips & Salsa	Hummus	NaN	PEPSICO	SABRA	700720008569	True
4961	Snacks	Dips & Salsa	Hummus	NaN	PEPSICO	SABRA	040822433655	True
7093	Snacks	Dips & Salsa	Hummus	NaN	PEPSICO	SABRA	040822344043	True
16050	Snacks	Dips & Salsa	Hummus	NaN	PEPSICO	SABRA	040822012157	True

Exploration of the products in a single brand

Most of the barcodes' first 6 digits were mostly identical!

3) Once a pattern was recognized, matches in the pattern are searched in the other dataset with the missing brand values keeping in mind the following assumption.

Assumption: It is assumed that all the products with an identical barcode pattern belong to the same brand for a given category.

```
# Checking for the pattern match in the dataset with missing brand names
result_m2 = dands_brand_isna[(dands_brand_isna['BARCODE'].str.contains('040822', case=False, na=False))]
result_m2.shape
```

```
(1, 8)
```

Finding a pattern match in the other dataset with the missing brands

Only one product's barcode pattern matched the common pattern. Nonetheless it was good news! Because that match was just for one brand with one barcode pattern. It was logically assumed that there were many such groups present following different fixed patterns. Matching all the products in the dips and salsa category according to the barcode patterns and filling the missing brand names with the corresponding brand seemed like the best idea because through that process many missing values could be imputed.

4) Keeping the above assumption in mind, the missing brands were imputed in the original dataset with the brands names of those products whose barcode patterns match with those of the products with missing brand names belonging to the 'Dips and Salsa' category.

```python
# Function to fill missing brands based on barcode prefixes for the rows with 'CATEGORY_2' = 'Dips & Salsa'
def fill_missing_brands(product_final):
    # Create a dictionary to store the first 6 characters of barcode and their corresponding brand (only for 'Dips & Salsa' rows)
    barcode_to_brand = {}

    # Iterate over the DataFrame and populate the barcode_to_brand dictionary with non-missing brands for 'Dips & Salsa' rows
    for index, row in product_final.iterrows():
        if pd.notnull(row['BARCODE']) and pd.notnull(row['BRAND']) and row['CATEGORY_2'] == 'Dips & Salsa':
            barcode_prefix = row['BARCODE'][:6]  # Get first 6 characters of the barcode
            barcode_to_brand[barcode_prefix] = row['BRAND']

    # Now, fill missing Brand values based on barcode_prefix matching (consider only non-null barcodes and 'Dips & Salsa' rows)
    for index, row in product_final.iterrows():
        if pd.isnull(row['BRAND']) and pd.notnull(row['BARCODE']) and row['CATEGORY_2'] == 'Dips & Salsa':
            barcode_prefix = row['BARCODE'][:6]
            if barcode_prefix in barcode_to_brand:
                product_final.at[index, 'BRAND'] = barcode_to_brand[barcode_prefix]  # Fill missing brand with the known brand

    return product_final
```

Imputation process

```python
product_dands2['BRAND'].isna().value_counts()
```

BRAND	count
False	11801
True	4467

dtype: int64

Number of missing brands after imputation

Through the imputation process, 1862 brand names were filled in the ‘Dips & Salsa’ category.

SQL Code:

```
WITH dipandsal_sales AS (
SELECT p.brand,
    COUNT(t.receipt_id) AS num_sales,                -- Number of sales based on receipts
    SUM(t.final_sale * t.final_quantity) AS total_sales_value  -- Total sales value
FROM transactions t
LEFT JOIN products p ON t.barcode = p.barcode -- Joining to get product information based on barcode
WHERE t.barcode IS NOT NULL          -- Excluding transactions with missing barcodes
   AND t.final_sale IS NOT NULL      -- Ensure we have a valid price for sales calculation
   AND p.category_2 = 'Dips & Salsa'  -- Filtering for "Dips & Salsa" in category_2
GROUP BY p.brand                    --Grouping the calculations based on brand
)
```

```
SELECT brand, num_sales, total_sales_value

FROM dipandsal_sales

ORDER BY total_sales_value DESC,    -- Order by total sales value first

         num_sales DESC          -- Order by the number of sales as a secondary criterion

LIMIT 1;                -- Get the top brand
```

Explanation:

The code consists of a CTE called 'dipandsal_sales' where the number of sales and the total sales are calculated using the number of receipt_ids and the product of final_quantity and final_sale respectively. These are obtained after joining the Transactions table with the Products table before grouping them according to the brands and filtering the results according to dips and salsa category. The second part of the code is where the results are ordered according to the total sales value and the number of sales. Here are the results:

	brand character varying	num_sales bigint	total_sales_value numeric
1	TOSTITOS	36	197.240

Question 6:

At what percent has Fetch grown year over year?

To estimate the growth of Fetch year over year, the chosen metrics are listed below:

1) New Users Count – These are the number of new users who create a fetch account counted yearly.
2) Total Transactions – These are the total transactions in which the new users participated counted yearly
3) Cumulative New User Growth Percentage – This is the year-on-year cumulative growth percentage of new users.
4) Cumulative Total Transactions Growth Percentage – This is the year-on-year cumulative growth percentage of the total number of transactions of new users.

SQL Code:

```
WITH yearly_new_users AS (

SELECT EXTRACT(YEAR FROM u.created_date) AS yr,

    COUNT(DISTINCT u.u_id) AS new_users_count

FROM users u
```

```
GROUP BY EXTRACT(YEAR FROM u.created_date)
),
total_transactions AS (
SELECT EXTRACT(YEAR FROM u.created_date) AS yr,
    COUNT(t.receipt_id) AS total_transactions
FROM users u
LEFT JOIN transactions t ON u.u_id = t.user_id
GROUP BY EXTRACT(YEAR FROM u.created_date)
),
yearly_data AS (
SELECT y.yr, y.new_users_count,
    COALESCE(tt.total_transactions, 0) AS total_transactions
FROM yearly_new_users y
LEFT JOIN total_transactions tt ON y.yr = tt.yr
),
cumulative_data AS (
SELECT yr, new_users_count, total_transactions,
     -- Calculating the cumulative sums for new users and transactions
     SUM(new_users_count) OVER (ORDER BY yr) AS cumulative_new_users,
     SUM(total_transactions) OVER (ORDER BY yr) AS cumulative_total_transactions
FROM yearly_data
)
SELECT yr, new_users_count, total_transactions,
  -- Calculating YoY cumulative growth for new users, handling zero in previous cumulative values
CASE WHEN LAG(cumulative_new_users) OVER (ORDER BY yr) > 0 THEN
   ROUND((cumulative_new_users - LAG(cumulative_new_users)
          OVER (ORDER BY yr)) / LAG(cumulative_new_users) OVER (ORDER BY yr) * 100, 2)
   ELSE NULL
END AS cumulative_new_user_growth_percentage,
```

```
  -- Calculating YoY cumulative growth for total transactions, handling zero in previous cumulative values
CASE WHEN LAG(cumulative_total_transactions) OVER (ORDER BY yr) > 0 THEN
  ROUND((cumulative_total_transactions - LAG(cumulative_total_transactions)
      OVER (ORDER BY yr)) / LAG(cumulative_total_transactions) OVER (ORDER BY yr) * 100, 2)
  ELSE NULL
END AS cumulative_total_transactions_growth_percentage
FROM cumulative_data
ORDER BY yr;
```

Explanation:

The code consists of 4 CTEs:

1st CTE is called yearly_new_users and it is used to calculate the number of distinct user_ids and group them according to the year in the created date. 2nd CTE calculates the total number of transactions by counting the total number of receipts in the transactions, grouped by year. 3rd CTE joins the tables formed by the previous 2 CTEs over the common years. The 4th CTE is used to calculate the cumulative values of both the values calculated in the first 2 CTEs. The final portion of the code is used to calculate the year-on-year growth percentages using the LAG query and ordering the results by the year. The yearly growth of Fetch is as follows:

	yr numeric	new_users_count bigint	total_transactions bigint	cumulative_new_user_growth_percentage numeric	cumulative_total_transactions_growth_percentage numeric
1	2014	30	0	[null]	[null]
2	2015	51	0	170.00	[null]
3	2016	70	0	86.42	[null]
4	2017	645	2	427.15	[null]
5	2018	2171	3	272.74	150.00
6	2019	7096	11	239.16	220.00
7	2020	16885	22	167.79	137.50
8	2021	19169	15	71.13	39.47
9	2022	26811	32	58.14	60.38
10	2023	15451	29	21.19	34.12
11	2024	11620	16	13.15	14.04