Ajay Venkatesh Pandravada

Fetch Rewards Coding Exercise - Data Analyst

Answering Questions from a business stakeholder

NOTE: MySQL has been used to write the queries and for ease of access sqlite3 has been used in google collab to run the queries. Required Tables have been created from the Data frames.

```
import sqlite3

# Create an in-memory SQLite database
conn = sqlite3.connect(':memory:')

# Load the DataFrame into the database as a table
items_table.to_sql('items_table', conn, index=False)
user_table.to_sql('user_table', conn, index=False)
rewards_table.to_sql('rewards_table', conn, index=False)
category_table.to_sql('category_table', conn, index=False)
brand_table.to_sql('brand_table', conn, index=False)
receipts_clean.to_sql('rewards_test', conn, index=False)
```

```
6941
```

- **What are the top 5 brands by receipts scanned for most recent month?**

Query:

```
[34] query = """
     SELECT b.name, COUNT(r.receipt_id) AS receipts_scanned
     FROM brand_table b
     JOIN items_table i ON b.barcode = i.barcode
     JOIN rewards_table r ON i.receipt_id = r.receipt_id
     WHERE r.dateScanned >= '2021-01-01 00:00:00'
     GROUP BY b.name
     ORDER BY receipts_scanned DESC
     LIMIT 5;
     """
     result = pd.read_sql_query(query, conn)
```

Output:

```
[35] result
```

	name	receipts_scanned
0	Tostitos	23
1	Swanson	11
2	Cracker Barrel Cheese	10
3	Prego	7
4	Diet Chris Cola	7

The query groups the data by brand name (GROUP BY b.name). This groups the retrieved data based on individual brands, allowing aggregating and counting the number of receipts associated with each brand**.**
The results are sorted in descending order based on the count of receipts scanned (ORDER BY receipts_scanned DESC). By doing so, we can identify the top brands with the highest number of receipts scanned. WHERE clause helps us filter out receipts produced in the most recent month.
This analysis helps identify the brands that have been most frequently scanned by customers, providing insights into brand popularity and customer engagement.

- **When considering *average spend* from receipts with 'rewardsReceiptStatus' of 'Accepted' or 'Rejected', which is greater?**

Query:

```
[36] query3_1 = """Select AVG(totalSpent)
     From rewards_table
     Where rewardsReceiptStatus = "FINISHED";
     """
     result3_1 = pd.read_sql_query(query3_1, conn)
```

Output:

```
[38] result3_1
```

	AVG(totalSpent)
0	80.854305

```
[37] query3_2 = """Select AVG(totalSpent)
     From rewards_table
     Where rewardsReceiptStatus = "REJECTED";
     """
     result3_2 = pd.read_sql_query(query3_2, conn)
```

```
[39] result3_2
```

	AVG(totalSpent)
0	23.326056

NOTE: In the data **'rewardsReceiptStatus'** doesn't have "accepted", so used "finished" instead.

The Query selects the data where rewardsReceipStatus is accepted(3_1) and rejected(3_2). Then aggregates the average of total spent amount from rewards_table. The analysis reveals that the average spending for the "Accepted" status is **3.5 times** higher than that of the "Rejected" status, highlighting a significant difference in spending patterns between the two groups.

- **When considering *total number of items purchased* from receipts with 'rewardsReceiptStatus' of 'Accepted' or 'Rejected', which is greater?**

Query:

```
[40] query4_1 = """SELECT SUM(purchasedItemCount)
     From rewards_table
     Where rewardsReceiptStatus = "FINISHED";
     """
     result4_1 = pd.read_sql_query(query4_1, conn)
```

Output:

```
[42] result4_1
```

	SUM(purchasedItemCount)
0	8184.0

```
[41] query4_2 = """SELECT SUM(purchasedItemCount)
     From rewards_table
     Where rewardsReceiptStatus = "REJECTED";
     """
     result4_2 = pd.read_sql_query(query4_2, conn)
```

```
[43] result4_2
```

	SUM(purchasedItemCount)
0	173.0

NOTE: In the data **'rewardsReceiptStatus'** doesn't have "accepted", so used "finished" instead.
The Query selects the data where rewardsReceipStatus is accepted(4_1) and rejected(4_2). Then aggregates the sum of purchasedItemCount from rewards_table. The analysis reveals that the number of items purchased for the "Accepted" status is **47 times** higher than that of the "Rejected" status, highlighting a significant difference in spending patterns between the two groups.

- **Which brand has the most *spend* among users who were created within the past 6 months?**

Query:

```
[44] query5 = """SELECT b.name, SUM(r.totalSpent) AS total_spend, u.createdDate
     FROM brand_table b
     JOIN items_table i ON b.barcode = i.barcode
     JOIN rewards_table r ON i.receipt_id = r.receipt_id
     JOIN user_table u ON r.userId = u.user_id
     WHERE u.createdDate >= '2020-08-01 00:00:00'
     GROUP BY b.name
     ORDER BY total_spend DESC
     LIMIT 1; """

     result5 = pd.read_sql_query(query5, conn)
```

Output:

```
[45] result5
```

	name	total_spend	createdDate
0	Tostitos	23812.97	2021-01-16 23:13:45

NOTE: In the query above, date (2020-08-01) was hardcoded as sqlite3 was giving issues in accessing date-time.

The provided SQL query retrieves the brand with the highest total spend among users created within the past 6 months. It achieves this by joining the brand_table, items_table, rewards_table, and user_table based on matching barcode, receipt ID, and user ID values. The query calculates the sum of totalSpent for each brand, groups the results by brand name, sorts them in descending order by total spend, and returns the brand with the highest spend.
This analysis helped identify that **"Tostitos"** was brand that generated the most spending of **$23812.97** among users created within the past 6 months, providing insights into brand preference and customer engagement.

- **Which brand has the most *transactions* among users who were created within the past 6 months?**

Query:

```
[46] query6 = """SELECT b.name, COUNT(*) AS transaction_count, u.createdDate
     FROM brand_table b
     JOIN items_table i ON b.barcode = i.barcode
     JOIN rewards_table r ON i.receipt_id = r.receipt_id
     JOIN user_table u ON r.userId = u.user_id
     WHERE u.createdDate >= '2020-08-01 00:00:00'
     GROUP BY b.name
     ORDER BY transaction_count DESC
     LIMIT 1;"""

     result6 = pd.read_sql_query(query6, conn)
```

Result:

```
[47] result6
```

	name	transaction_count	createdDate
0	Tostitos	43	2021-01-16 23:13:45

NOTE: In the query above, date (2021-01-01) was hardcoded as sqlite3 was giving issues in accessing date-time.

The provided SQL query retrieves the brand with the most transactions among users created within the past 6 months. It achieves this by joining the brand_table, items_table, rewards_table, and user_table based on matching barcode, receipt ID, and user ID values. The query then groups the results by brand name, sorts them in descending order by the transaction count, and selects the brand with the highest count. We can see that again **"Tostitos"** is the brand which has Highest transactions which is **43**.