Exploration of Transactions Table

Displaying Missing Values from the Transactions table:

Query:

```
print("\nMissing Values before dropping duplicate records:")
print("Transactions:", transactions_df.isnull().sum())
```

Output:

```
Missing Values before dropping duplicate records:
>>> print(transactions_df.isnull().sum())
RECEIPT_ID            0
PURCHASE_DATE         0
SCAN_DATE             0
STORE_NAME            0
USER_ID               0
BARCODE            5762
FINAL_QUANTITY        0
FINAL_SALE            0
```

Displaying count of Duplicate records:

Query:

```
print("\nDuplicate Rows Count:")
print("Transactions:", transactions_df.duplicated().sum())
```

Output:

```
Number of Duplicate records:
>>> print(transactions_df.duplicated().sum())
171
```

Dropping duplicate rows from each table:

```
# Dropping duplicate rows from each table
users_df = users_df.drop_duplicates()
transactions_df = transactions_df.drop_duplicates()
products_df = products_df.drop_duplicates()
```

Converting Date based columns to datetime format:

```
transactions_df['PURCHASE_DATE'] = pd.to_datetime(transactions_df['PURCHASE_DATE'],
errors='coerce')
transactions_df['SCAN_DATE'] = pd.to_datetime(transactions_df['SCAN_DATE'],
errors='coerce')
```

Converting 'FINAL_QUANTITY' & 'FINAL_SALE' columns to numeric value:

```
transactions_df['FINAL_QUANTITY'] =
pd.to_numeric(transactions_df['FINAL_QUANTITY'], errors='coerce')
transactions_df['FINAL_SALE'] = pd.to_numeric(transactions_df['FINAL_SALE'],
errors='coerce')
```

After rechecking the missing values in transactions table the count of FINAL_QUANTITY and FINAL_SALE changed from 0 -> 12491 and 0->12486 respectively

Checking for count of null values in all the columns:

Query:

```
print("Transactions:", transactions_df.isnull().sum())
```

Output:

```
Transactions: RECEIPT_ID
PURCHASE_DATE          0
SCAN_DATE              0
STORE_NAME             0
USER_ID                0
BARCODE             5735
FINAL_QUANTITY     12491
FINAL_SALE         12486
dtype: int64
```

Creating a new column to store the converted value - '`FINAL_QUANTITY2`' & '`FINAL_SALE2`':

```
transactions_df['FINAL_QUANTITY2'] =
pd.to_numeric(transactions_df['FINAL_QUANTITY'], errors='coerce')
transactions_df['FINAL_SALE2'] = pd.to_numeric(transactions_df['FINAL_SALE'],
errors='coerce')
```

While converting 'FINAL_QUANTITY' to numeric field and creating a new field 'FINAL_QUANTITY2' some of the rows had errors and got NaN as a result to check what is the actual value for FINAL_QUANTITY where the error is occuring I am taking the data where FINAL_QUANTITY2 = 'NaN':

```
filtered_df = transactions_df[transactions_df['FINAL_QUANTITY2'].isna()]
```

```
# Getting unique values from the column which are not getting converted properly
print(filtered_df['FINAL_QUANTITY'].unique())
```

The unique value not converted properly in FINAL_QUANTITY field:

Replace the 'zero' with integer value '0' and convert the column to numeric:

```
# Replace the 'zero' with integer value '0' and convert the column to numeric
transactions_df['FINAL_QUANTITY'] =
transactions_df['FINAL_QUANTITY'].replace('zero', '0')
transactions_df['FINAL_QUANTITY'] =
pd.to_numeric(transactions_df['FINAL_QUANTITY'], errors='coerce')
#Remove the column 'FINAL_QUANTITY2'
transactions_df.drop(columns=['FINAL_QUANTITY2'], inplace=True)
```

Repeating same steps for converting FINAL_SALE to numeric data type:

```
# Repeating same steps for FINAL_SALE column
filtered_df = transactions_df[transactions_df['FINAL_SALE2'].isna()]

# Getting unique values from the column which are not getting converted properly
print(filtered_df['FINAL_SALE'].unique())

# # Replace the ' ' with integer value '0' and convert the column to numeric
transactions_df['FINAL_SALE'] = transactions_df['FINAL_SALE'].replace(' ', '0')
transactions_df['FINAL_SALE'] = pd.to_numeric(transactions_df['FINAL_SALE'],
errors='coerce')

# #Remove the column 'FINAL_QUANTITY2'
transactions_df.drop(columns=['FINAL_SALE2'], inplace=True)
```

For the BARCODE column, checking if there is any barcode with ‘0’ as value so that N/A can be replaced with ‘0’:

Query:

```
# Checked to see if there is any barcode with '0' as value so that N/A can be
replaced with 0
print((transactions_df['BARCODE'] == 0).unique())
print((products_df['BARCODE'] == 0).unique())
```

Output:

```
RECEIPT_ID            0
PURCHASE_DATE         0
SCAN_DATE             0
STORE_NAME            0
USER_ID               0
BARCODE            5735
FINAL_QUANTITY        0
FINAL_SALE            0
```

Foundings:

- As shown above there are **5735** rows with missing barcode values.
- Before recoding the **missing values to 0**, I considered the following factors:
    - If each product had unique values, I could have imputed the barcode based on FINAL_QUANTITY and FINAL_SALES.
    - Another approach was to impute the barcode based on the most frequently purchased product by a user, though this might not yield accurate results.
    - Ultimately, I decided not to remove the rows entirely since, even without barcode values, other columns still provide valuable insights into the transaction dataset.

Calculating and finding the total duration of transaction data:

Query:

```
# Find start and end dates
start_date = transactions_df['SCAN_DATE'].min()
end_date = transactions_df['SCAN_DATE'].max()

# Calculate the total duration
duration = end_date - start_date

# Print duration of transaction
print(f"Total duration of transaction data: {duration.days} days")
```

Output:

```
>>> print(f"Total duration of transaction data: {duration.days} days")
Total duration of transaction data: 88 days
```

Counting rows where 'FINAL_SALE' has value but 'FINAL_QUANTITY' is zero:

Query:

```
# Count rows where FINAL_SALE has values but FINAL_QUANTITY is zero
final_sale_nonzero_quantity_zero = transactions_df[
   (transactions_df['FINAL_SALE'] > 0) & (transactions_df['FINAL_QUANTITY'] == 0)
].shape[0]

# Count rows where FINAL_QUANTITY has values but FINAL_SALE is zero
final_quantity_nonzero_sale_zero = transactions_df[
   (transactions_df['FINAL_QUANTITY'] > 0) & (transactions_df['FINAL_SALE'] == 0)
].shape[0]

# Display results
print(f"Rows where FINAL_SALE > 0 but FINAL_QUANTITY == 0:
{final_sale_nonzero_quantity_zero}")
print(f"Rows where FINAL_QUANTITY > 0 but FINAL_SALE == 0:
{final_quantity_nonzero_sale_zero}")
```

Output:

```
>>> print(f"Rows where FINAL_SALE > 0 but FINAL_QUANTITY == 0: {final_sale_nonzero_quantity_zero}")
Rows where FINAL_SALE > 0 but FINAL_QUANTITY == 0: 12341
>>> print(f"Rows where FINAL_QUANTITY > 0 but FINAL_SALE == 0: {final_quantity_nonzero_sale_zero}")
Rows where FINAL_QUANTITY > 0 but FINAL_SALE == 0: 12821
```

Number of rows for 'FINAL_SALE' > 0 but 'FINAL_QUANTITY' == 0 is: 12,332 rows

Number of rows for 'FINAL_QUANTITY' > 0 but 'FINAL_SALE' == 0 is: 12,800 rows

Finding 'SCAN_DATE' & 'PURCHASE_DATE' inconsistencies:

Query:

```
# Create new columns without the timestamp
transactions_df['SCAN_DATE_ONLY'] = transactions_df['SCAN_DATE'].dt.date
transactions_df['PURCHASE_DATE_ONLY'] = transactions_df['PURCHASE_DATE'].dt.date

# Now check where SCAN_DATE_ONLY is before PURCHASE_DATE_ONLY
scan_before_purchase = transactions_df[transactions_df['SCAN_DATE_ONLY'] <
transactions_df['PURCHASE_DATE_ONLY']]

# Count occurrences
```

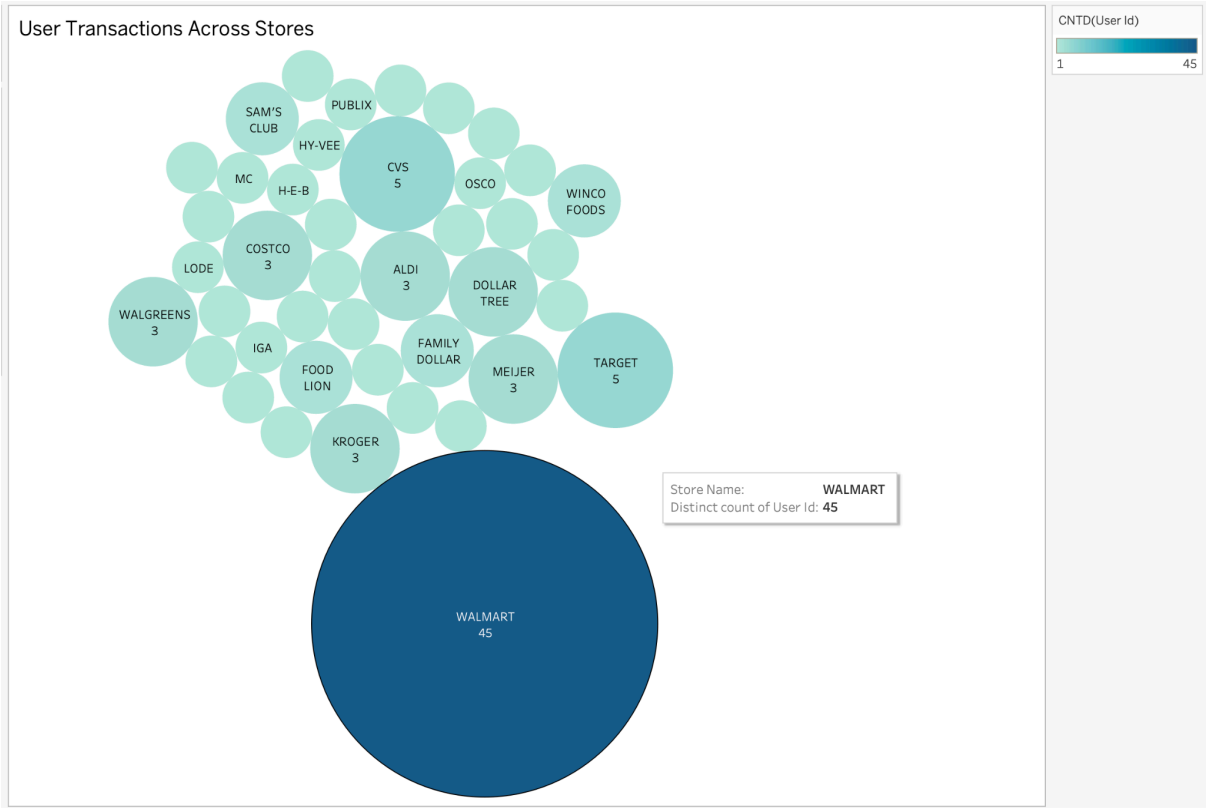
```
count_scan_before_purchase = scan_before_purchase.shape[0]

# Display results
print(f"Number of rows where SCAN_DATE_ONLY is before PURCHASE_DATE_ONLY:
{count_scan_before_purchase}")

# Display some of these rows
print(scan_before_purchase[['PURCHASE_DATE_ONLY', 'SCAN_DATE_ONLY']].head(10))
```

Output:

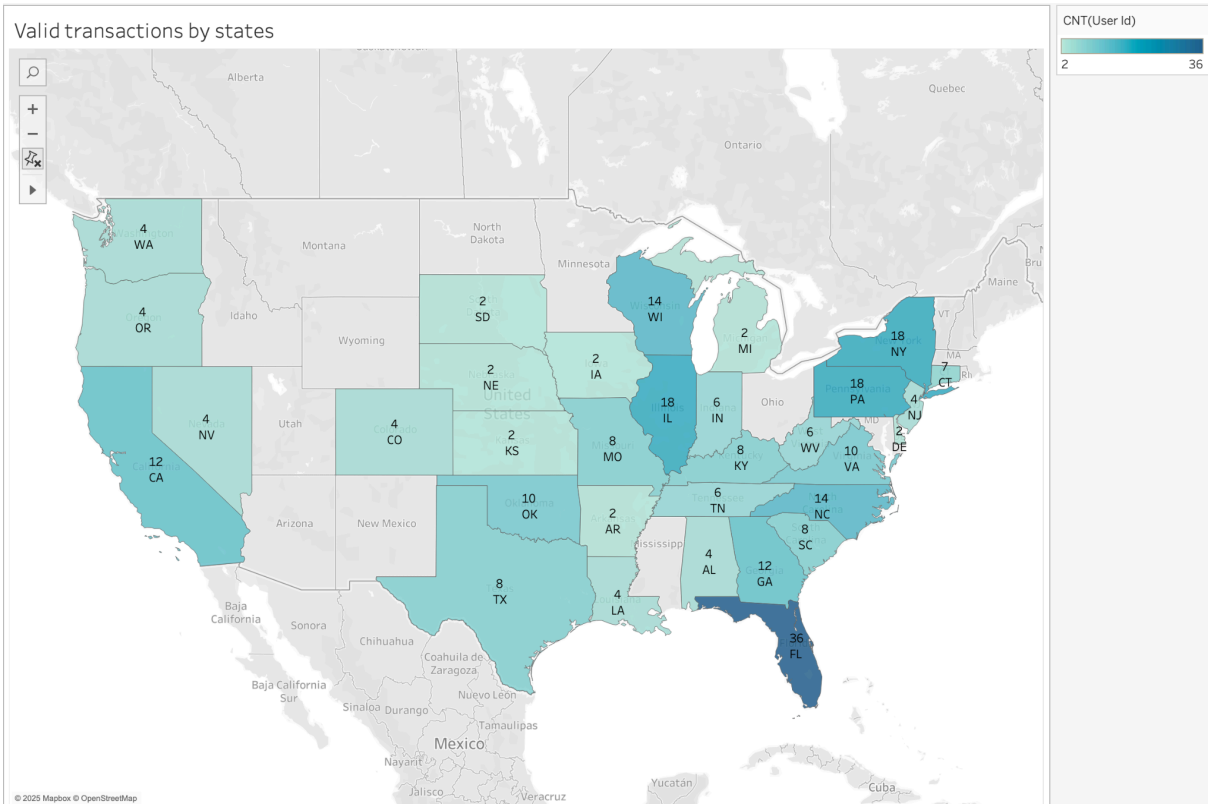
```
Number of rows where SCAN_DATE_ONLY is before PURCHASE_DATE_ONLY: 94
     PURCHASE_DATE_ONLY SCAN_DATE_ONLY
51           2024-07-21     2024-07-20
455          2024-06-29     2024-06-28
494          2024-09-08     2024-09-07
675          2024-06-22     2024-06-21
870          2024-06-22     2024-06-21
1126         2024-08-10     2024-08-09
2342         2024-07-09     2024-07-08
3648         2024-06-29     2024-06-28
4159         2024-06-18     2024-06-17
4532         2024-09-05     2024-09-04
```

Some interesting trends found in transaction table:

Graphical view of user transactions across stores:



Foundings:
Majority of the transactions with valid ‘user_id’ in users table made their purchase at WALMART.



Majority of valid transactions happened in the state of **Florida**, which is around **36**

Assessment Answers: Data Quality & Challenges in the Transactions Table

Q. Are there any data quality issues present?

Yes, several data quality issues were found in the transactions table:

1. **Missing or Invalid Values**

    a. There are 5735 rows with missing **BARCODE** values.

    b. Before recoding the missing values to 0, I considered the following factors:
        i. If each product had unique values, I could have imputed the barcode based on FINAL_QUANTITY and FINAL_SALES.
        ii. Another approach was to impute the barcode based on the most frequently purchased product by a user, though this might not yield accurate results.

iii. Ultimately, I decided not to remove the rows entirely since, even without barcode values, other columns still provide valuable insights into the transaction dataset.

c. **FINAL_QUANTITY** had non-numeric values like 'zero'.
   i. Replaced 'zero' with integer 0 and converted the column to numeric.

d. **FINAL_SALE** had empty string values (' '), making conversion to numeric difficult.
   i. Replaced empty strings with 0 before conversion.

2. **Potential Data Inconsistencies**
   a. **FINAL_SALE > 0 but FINAL_QUANTITY == 0 (12,332 rows). The receipt shows a total price but does not specify a quantity. Reasons:**

      i. Weight-based items (e.g., produce, meat, bulk goods) might be recorded without a unit quantity.
      ii. Non-physical products like gift cards, fees, or services may not have a quantity field.
      iii. Data extraction errors where the receipt processing system failed to capture the quantity.
      iv. Discounted or adjusted items where the quantity might have been removed but the final sale price remained.

   b. **FINAL_QUANTITY > 0 but FINAL_SALE == 0 (12,800 rows). The receipt lists a quantity of an item but no total sale amount. Possible reasons:**

      i. Promotional or free items (e.g., BOGO deals, store giveaways).
      ii. Coupons covering full cost, making the final sale amount zero.
      iii. Refunded items where quantity remains in records but the final sale was adjusted to zero.
      iv. Receipt scanning or OCR errors causing the price to be missed.

   c. **SCAN_DATE occurs before PURCHASE_DATE in 94 cases:**

      i. This would indicate an error since scanning should happen after or on the purchase date.

Q. Are there any fields that are challenging to understand?

There are a few fields in the **Transactions Table** that may be challenging to understand due to inconsistencies and missing values:

1. **FINAL_SALE vs FINAL_QUANTITY**

    a. There are cases where **FINAL_SALE has a value but FINAL_QUANTITY is 0** (12,332 rows).
    b. Conversely, there are cases where **FINAL_QUANTITY has a value but FINAL_SALE is 0** (12,800 rows).
    c. Possible explanations include weight-based products, discounts, promotions, or data extraction errors.
2. **BARCODE Missing for Many Transactions**
    a. There are **5,735 rows with missing BARCODE values**.
    b. The missing barcodes could indicate incomplete receipts, scanned items without product-level details, or errors in data capture.
3. **SCAN_DATE vs PURCHASE_DATE**
    a. In **94 cases, SCAN_DATE occurs before PURCHASE_DATE**, which is logically incorrect.
    b. This could be due to data entry errors or system processing issues.

While these fields are generally understandable, their inconsistencies require further investigation to ensure data accuracy.