Consumer Retail Rewards - Data Quality Assessment

First: Explore the data

1. Are there any data quality issues present?

- The data exploration phase focuses on understanding the dataset's structure, quality, and patterns.
- **Data quality** is assessed by identifying **missing or incomplete data, checking field validity, and detecting duplicates and inconsistencies.**
- Additionally, **data types are validated, and formatting** is standardized across the `Users`, `Products`, and `Transactions` datasets to ensure reliability for analysis.

1. Missing & Incomplete data

a. **User:** Key fields have low missing rates, making them reliable for analysis. While language has higher missing rate it may not have significant impact on demographic analysis

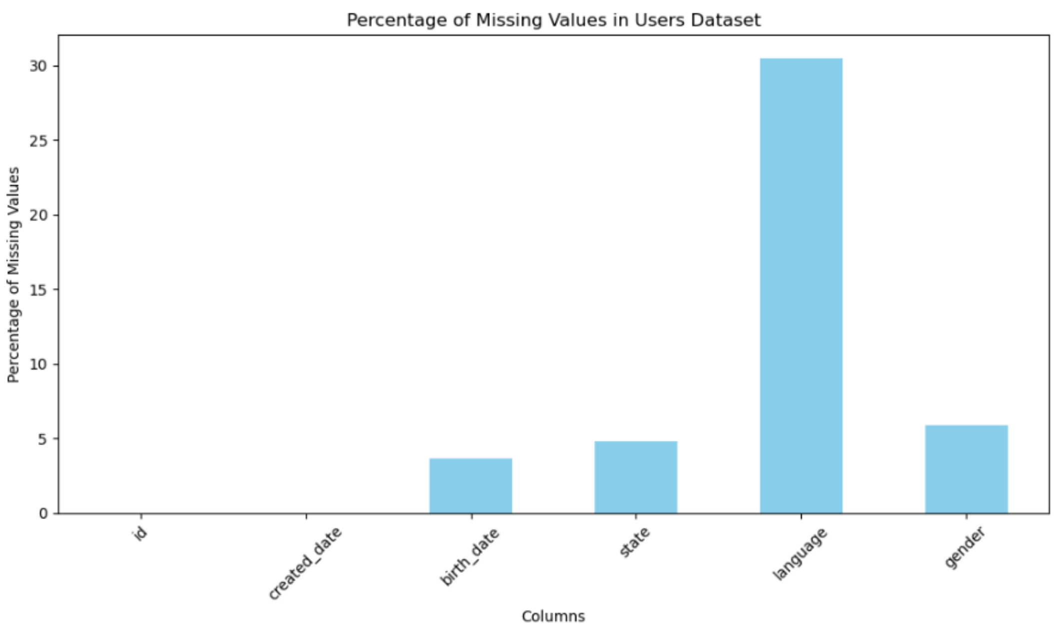
```
import matplotlib.pyplot as plt

# Total number of records in each dataset
users_row_count = users_df.shape[0]
products_row_count = products_df.shape[0]
transactions_row_count = transactions_df.shape[0]

# Checking for missing values in each column in the dataset
users_missing_values = users_df.isna().sum()
products_missing_values = products_df.isna().sum()
transactions_missing_values = transactions_df.isna().sum()

# Calculate the percentage of missing values in each column in the dataset
users_missing_percentage = (users_missing_values / users_row_count) * 100
products_missing_percentage = (products_missing_values / products_row_count) * 100
transactions_missing_percentage = (transactions_missing_values / transactions_row_count) * 100
```

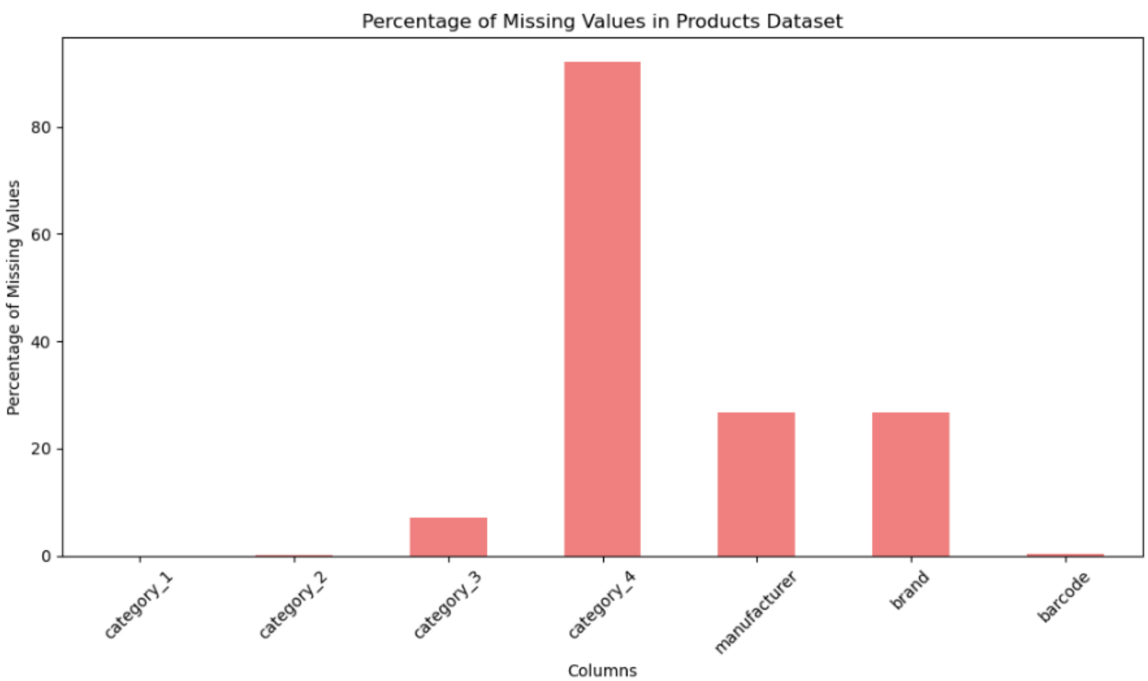
```
Total Rows in Users DataFrame: 100000
Missing % in Users DataFrame:
id                   0.000
created_date         0.000
birth_date           3.675
state                4.812
language            30.508
gender               5.892
dtype: float64
```



b. **Products:** The key fields **Brand** and **Manufacturer** have high missing values (~26%), which can impact product-level analysis.

The **Category_4** field has very high missing values, which may affect the hierarchical product classification. Given the hierarchical structure of the four category fields, it's essential to determine if there are specific rules governing when **Category_4** should contain a value

```
Total Rows in Products DataFrame: 845552
Missing % in Products DataFrame:
category_1       0.013128
category_2       0.168411
category_3       7.162895
category_4      92.021898
manufacturer    26.784160
brand           26.783923
barcode          0.476020
dtype: float64
```

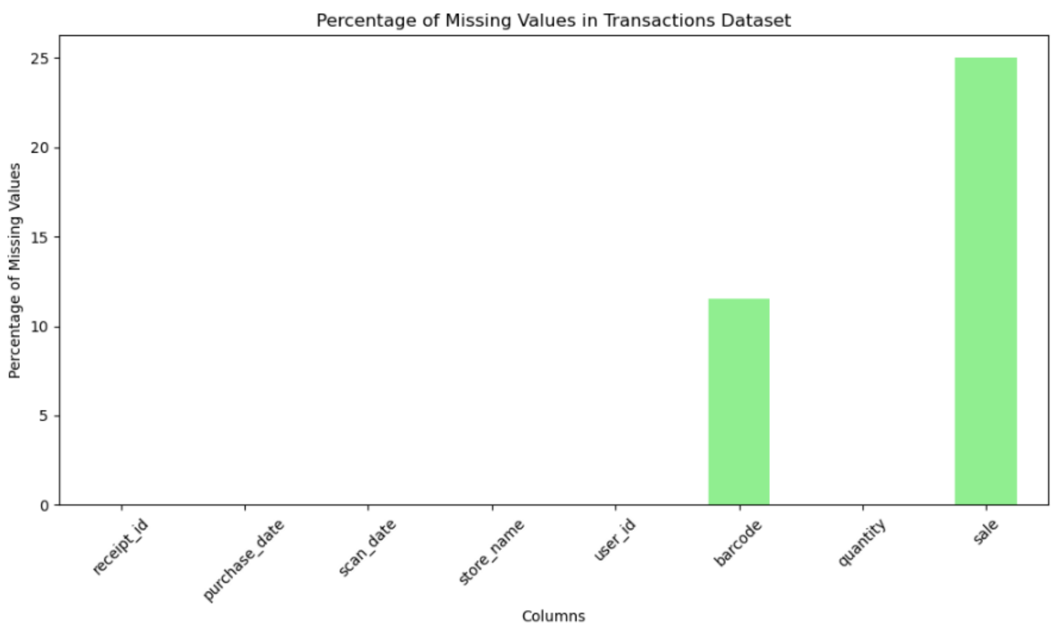


c. **Transactions:** In the `Transactions` table, `barcode`, `quantity`, and `sale` fields have notable missing percentages.

High missing value in `barcode` which is a critical field for linking products can lead to to inaccuracies in understanding popular brands, categories, and total revenue distribution.

Similarly high missing rate in sale can lead to missed **Revenue Analysis, Customer Spending Insights etc.**

```
Total Rows in Transactions DataFrame: 50000
Missing % in Transactions DataFrame:
receipt_id         0.000
purchase_date      0.000
scan_date          0.000
store_name         0.000
user_id            0.000
barcode           11.524
quantity          0.000
sale              25.000
dtype: float64
```



2. **Validity of linked fields - id & barcode**

   There are ~18% of user ids that are present in transactions dataset but not in users dataset & ~ 4% of barcode that is present in transactions but not in products

   Impact on overall Analysis:

- **User Demographics:** Missing 18% of `user_id`s may distort insights into demographics, affecting metrics like spending and transaction frequency.
- **Product Popularity and Sales:** Missing 4% of `barcodes` leads to incomplete product-level analysis, impacting revenue and brand insights.

```python
# Count the total users
total_users = len(users_df)

# Find distinct user IDs in transactions_df that are not in users_df and exclude nulls
distinct_user_ids_not_in_users_count = transactions_df[
    ~transactions_df['user_id'].isin(users_df['id']) &
    transactions_df['user_id'].notna()
]['user_id'].nunique()

# Display the result for user IDs
print(f"Count of distinct user IDs in transactions dataset not present in users dataset: {distinct_user_ids_not_in_users_count}, "
      f"percentage: {((distinct_user_ids_not_in_users_count) / total_users) * 100:.2f}%")

# Filter distinct barcodes in transactions that are not in products and exclude nulls
distinct_barcodes_not_in_products_count = transactions_df[
    ~transactions_df['barcode'].isin(products_df['barcode']) &
    transactions_df['barcode'].notna()
]['barcode'].nunique()

# Display the result for barcodes
print(f"Count of distinct barcodes in transactions dataset not present in products dataset: {distinct_barcodes_not_in_products_count}, "
      f"percentage: {round((distinct_barcodes_not_in_products_count / total_users) * 100, 2)}%")
```

```
Count of distinct user IDs in transactions dataset not present in users dataset: 17603, percentage: 17.60%
Count of distinct barcodes in transactions dataset not present in products dataset: 4465, percentage: 4.46%
```

3. **Duplicates**

User ID	Product Barcodes	Transaction Receipt ID
No duplicates	0.5%	51.2%

```python
# Check for duplicates in primary keys and calculate percentage

# Users DataFrame - duplicate 'id'
duplicate_user_count = users_df['id'].duplicated().sum()
duplicate_user_percentage = (duplicate_user_count / len(users_df)) * 100
print(f"Duplicate User IDs: {duplicate_user_count} ({duplicate_user_percentage:}%)")

# Products DataFrame - duplicate 'barcode'
duplicate_product_count = products_df['barcode'].duplicated().sum()
duplicate_product_percentage = round(((duplicate_product_count / len(products_df)) * 100),2)
print(f"Duplicate Product Barcodes: {duplicate_product_count} ({duplicate_product_percentage:}%)")

# Transactions DataFrame - duplicate 'receipt_id'
duplicate_receipt_count = transactions_df['receipt_id'].duplicated().sum()
duplicate_receipt_percentage = round(((duplicate_receipt_count / len(transactions_df)) * 100),2)
print(f"Duplicate Receipt IDs: {duplicate_receipt_count} ({duplicate_receipt_percentage:}%)")
```

```
Duplicate User IDs: 0 (0.0%)
Duplicate Product Barcodes: 4209 (0.5%)
Duplicate Receipt IDs: 25560 (51.12%)
```

3. **Data Inconsistency**

a. **Users:**

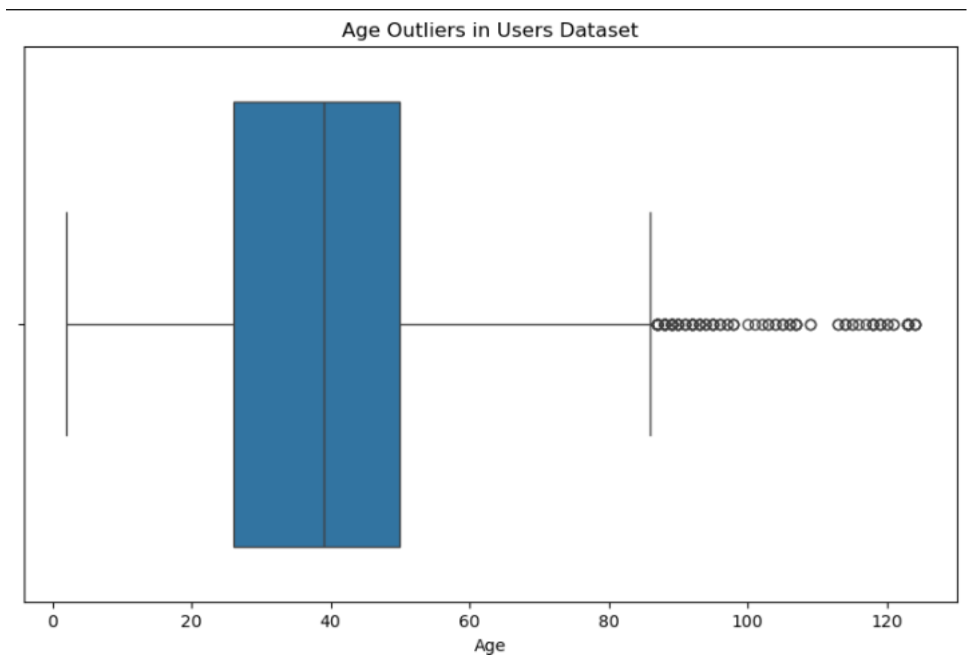
1. 61 invalid birth dates, outliers in age calculation

```
# Define minimum and maximum birth year
min_birth_year = today.year - 100  # Maximum age is 100 years
max_birth_year = today.year        # Minimum age is 0 years

# Filter for outliers in 'birth_date'
birth_date_outliers = users_df[
    (users_df['birth_date'].dt.year < min_birth_year) |
    (users_df['birth_date'].dt.year > max_birth_year)
]

invalid_birth_date_count = len(birth_date_outliers)
print("Number of records with invalid 'birth_date':", invalid_birth_date_count)
```

2. Outlier in age as calculated from birthdate:



3. Birth dates that are frequently repeated - Could be an error or default value used

`1970-01-01 00:00:00    1272`

```python
# Calculate the frequency of each birth date
birth_date_counts = users_df['birth_date'].value_counts()

# Find the 90th percentile of these frequencies
threshold = birth_date_counts.quantile(0.90)

# Filter for birth dates that are repeated above the 90th percentile
frequent_birth_dates = birth_date_counts[birth_date_counts > threshold].head(10)

# Display the result
print("Birth dates that are frequently repeated (above 90th percentile):")
print(frequent_birth_dates)
```

b. Products

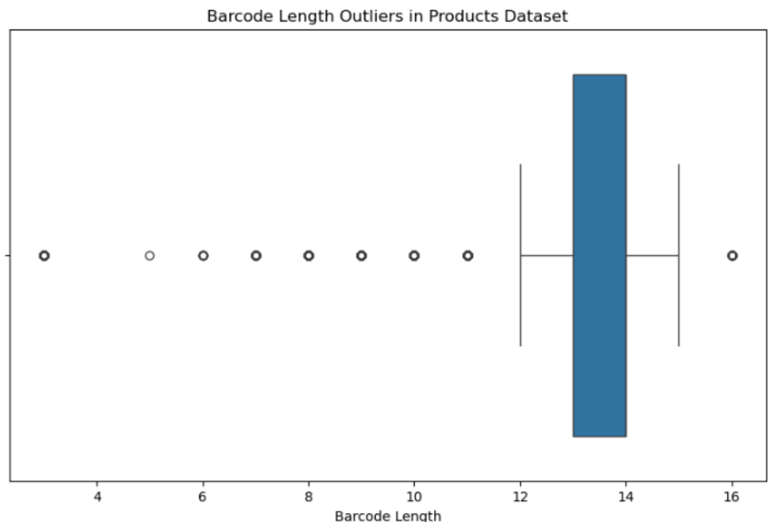
1. Bar code length is not consistent

```python
# Ensure 'barcode' is treated as a string to calculate length, then count occurrences by length
barcode_lengths = products_df['barcode'].dropna().astype(str).apply(len)
barcode_length_counts = barcode_lengths.value_counts()

# Display the counts for each barcode length
print("Counts of each barcode length:\n", barcode_length_counts)
```

```
Counts of each barcode length:
 barcode
14    502091
13    296376
15     30144
12      7591
10      1907
11      1762
9        950
8        570
7         83
16        44
6          8
5          1
Name: count, dtype: int64
```

2. A box plot of barcode lengths to identify any unusual lengths:



Standard Barcode Lengths:

- **13 & 12 digits**: Match **EAN-13** and **UPC-A** standards.
- **8 digits**: Likely **EAN-8**.
- **Other lengths (e.g., 5-7, 9-16 digits)**: Non-standard; may indicate errors or placeholders.

c. Transactions

1. 94 records with scan date before purchase date, 12,500 records with string value 'zero' for quantity

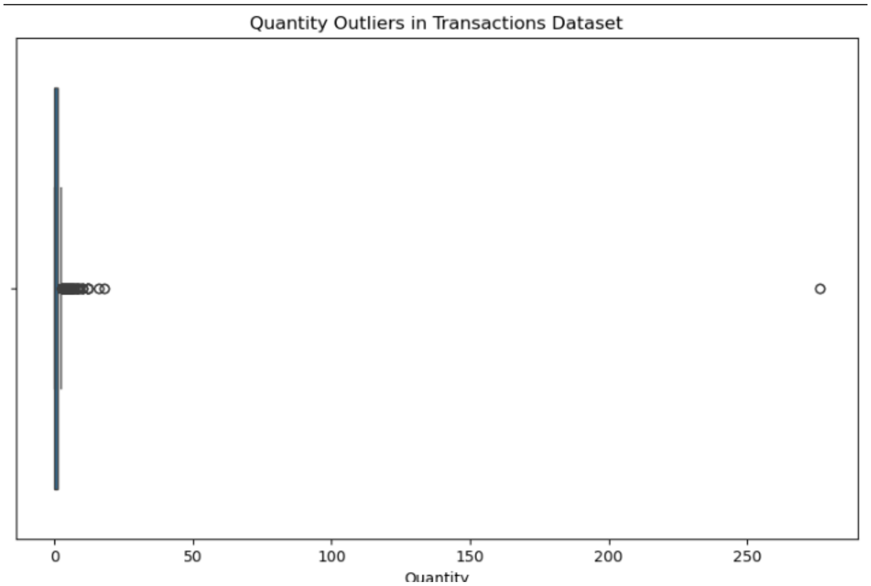
```
# Convert scan_date and receipt_date columns to same format of date time
transactions_df['scan_date'] = pd.to_datetime(transactions_df['scan_date'], errors='coerce').dt.tz_localize(None)
transactions_df['purchase_date'] = pd.to_datetime(transactions_df['purchase_date'], errors='coerce').dt.tz_localize(None)

# Check if scan_date is after receipt_date
invalid_dates = transactions_df[transactions_df['scan_date'] < transactions_df['purchase_date']]

# Display count of rows with invalid dates
invalid_date_count = invalid_dates.shape[0]
print("Count of rows where scan_date is before purchase_date:", invalid_date_count)
```

```
Count of rows where scan_date is before purchase_date: 94
```

2. A box plot for quantity in transactions, revealing high or low quantities.



b. There are 321 records with value in quantity > 0 but corresponding value in sales as 0

```
# Ensure quantity and sale are numeric
transactions_df['quantity'] = pd.to_numeric(transactions_df['quantity'], errors='coerce')
transactions_df['sale'] = pd.to_numeric(transactions_df['sale'], errors='coerce')

# Count records with numeric quantity, non-missing sales, and sales value is 0
zero_sales_count = transactions_df[
    (transactions_df['quantity'].notna()) & (transactions_df['quantity'] > 0) &
    (transactions_df['sale'].notna()) & (transactions_df['sale'] == 0)
].shape[0]

# Display the count
print("Count of records with numeric quantity but sales value is 0:", zero_sales_count)
```

```
Count of records with numeric quantity but sales value is 0: 321
```

c. There are 110 records in quantity that are not whole number. may be produce but that is not a significant number to consider.

```
# Ensure quantity is numeric
transactions_df['quantity'] = pd.to_numeric(transactions_df['quantity'], errors='coerce')

# Find records where quantity is negative or not a whole number
invalid_quantity_records = transactions_df[
    (transactions_df['quantity'] < 0) |
    (transactions_df['quantity'] % 1 != 0)]

# Display count of records with invalid quantity
invalid_quantity_count = invalid_quantity_records.shape[0]
print("Count of records with invalid quantity (negative or non-whole number):", invalid_quantity_count)
```

```
Count of records with invalid quantity (negative or non-whole number): 110
```

d. Sale: A box plot for sale amounts in transactions, which helps in identifying extreme sale values.

3. **Datatype validation**

Below code checks for data type present in the actual dataset.

```
# 5.Data Type Validation
# Check data types for each DataFrame
print("Data types in Users DataFrame:")
print(users_df.dtypes)

print("\nData types in Products DataFrame:")
print(products_df.dtypes)

print("\nData types in Transactions DataFrame:")
print(transactions_df.dtypes)
```

Below fields are converted to align with the datatype provided ER model provided.

Users: created_date & birth_date

Transactions: purchase_date, scan_date, quantity, sale

4. **Data Formatting**

Users

1. Gender field contains non-standard values like "transgender," "prefer_not_to_say," and "non_binary," which don't align with expected options.

```
# Expected gender values in the correct format
valid_genders = ['male', 'female', 'non-binary', "my gender isn't listed", 'prefer not to say']

# Check for values that don't match the exact expected format
incorrect_category_genders = users_df[~users_df['gender'].isin(valid_genders)]

# Display results
if not incorrect_category_genders.empty:
    print("Records with other categories in the gender field:\n", incorrect_category_genders['gender'].value_counts())
```

```
Records with other categories in the gender field:
 gender
transgender          1772
prefer_not_to_say    1350
non_binary            473
unknown               196
not_listed            180
not_specified          28
Name: count, dtype: int64
```

Product

a. **Inconsistent barcode lengths (5-16 digits)**

b. **Minor formatting issues in brand and manufacturer fields**

c. **Formatting issues in Category 3 and Category 4**

```
# Find rows where the store_name field is non-null and does not match the title-cased version
unformatted_store_name = transactions_df[
    transactions_df['store_name'].notna() &
    (transactions_df['store_name'] != transactions_df['store_name'].str.strip().str.upper())
]

# Display the count and sample of unformatted category_1 values
print("Number of unformatted store_name values (excluding nulls):", unformatted_store_name.shape[0])
print("Sample of unformatted store_name values:")
print(unformatted_store_name['store_name'].head())
```

```
Number of unformatted store_name values (excluding nulls): 2
Sample of unformatted store_name values:
13649    TINKER COMMISsARY
45075    TINKER COMMISsARY
Name: store_name, dtype: object
```

```
# Find rows where the Brand field is non-null and does not match the title-cased version
unformatted_brands = products_df[
    products_df['brand'].notna() &
    (products_df['brand'] != products_df['brand'].str.strip().str.upper())
]

# Display the count and sample of unformatted Brand values
print("Number of unformatted Brand values (excluding nulls):", unformatted_brands.shape[0])
print("Sample of unformatted Brand values:")
print(unformatted_brands['brand'].head())
```

```
Number of unformatted Brand values (excluding nulls): 3
Sample of unformatted Brand values:
298995    Listerine
315241    Listerine
553728    Kellogg's
Name: brand, dtype: object
```

```
# Find rows where the category_3 field is non-null and does not match the title-cased version
unformatted_category_3 = products_df[
    products_df['category_3'].notna() &
    (products_df['category_3'] != products_df['category_3'].str.strip().str.title())
]

# Display the count and sample of unformatted category_1 values
print("Number of unformatted category_3 values (excluding nulls):", unformatted_category_3.shape[0])
print("Sample of unformatted category_3 values:")
print(unformatted_category_3['category_3'].head())
```

```
Number of unformatted category_3 values (excluding nulls): 12896
Sample of unformatted category_3 values:
7          Men's Deodorant & Antiperspirant
18         Men's Deodorant & Antiperspirant
99      Foot Care Devices and Grooming Aids
117     Foot Care Devices and Grooming Aids
166        Men's Deodorant & Antiperspirant
Name: category_3, dtype: object
```

```python
# Find rows where the category_4 field is non-null and does not match the title-cased version
unformatted_category_4 = products_df[
    products_df['category_4'].notna() &
    (products_df['category_4'] != products_df['category_4'].str.strip().str.title())
]

# Display the count and sample of unformatted category_1 values
print("Number of unformatted category_4 values (excluding nulls):", unformatted_category_4.shape[0])
print("Sample of unformatted category_4 values:")
print(unformatted_category_4['category_4'].head())
```

```
Number of unformatted category_4 values (excluding nulls): 9708
Sample of unformatted category_4 values:
25     Women's Shaving Gel & Cream
44     Women's Shaving Gel & Cream
82     Women's Shaving Gel & Cream
109                   Men's Razors
162    Women's Shaving Gel & Cream
Name: category_4, dtype: object
```

Transaction

Some store names have case differences and extra whitespace

```python
# Find rows where the store_name field is non-null and does not match the title-cased version
unformatted_store_name = transactions_df[
    transactions_df['store_name'].notna() &
    (transactions_df['store_name'] != transactions_df['store_name'].str.strip().str.upper())
]

# Display the count and sample of unformatted category_1 values
print("Number of unformatted store_name values (excluding nulls):", unformatted_store_name.shape[0])
print("Sample of unformatted store_name values:")
print(unformatted_store_name['store_name'].head())
```

```
Number of unformatted store_name values (excluding nulls): 2
Sample of unformatted store_name values:
13649    TINKER COMMISsARY
45075    TINKER COMMISsARY
Name: store_name, dtype: object
```

2. **Are there any fields that are challenging to understand?**

   User:

   1. **Approximately 18% of `user_id`s in the transactions dataset are missing in the users dataset**, raising questions about the `id` field's definition in the users table. Clarify if `id` represents only active or registered users, or if it excludes certain user types.

   Product:

1. **Are all barcodes unique to each product**, and is there a standard length or format?
2. What is the structure of product categories? **Understanding the hierarchy of categories** can help with detailed product segmentation.
3. Manufacturer field has a **~10% (86902)** records with value "**Placeholder Manufacturer**" What does this mean? Is "Placeholder Manufacture" a temporary value, or should it be treated as unknown or missing data? It can introduce inaccuracies in manufacturer-based analysis.

Transactions:

1. Approximately 4% of `barcode`s in the transactions dataset are missing in the products dataset, limiting product-specific analysis and potentially skewing category and brand insights. Clarifying whether these missing barcodes are expected (e.g., discontinued items) or indicate incomplete data will help improve accuracy.
2. **Approximately 50% (24,440) of `receipt_id`s in the dataset are unique**, Could you clarify whether unique `receipt_id`s are expected in cases of single-item transactions, or should we anticipate multiple items per receipt? Understanding this will help determine if the high number of unique `receipt_id`s aligns with expected transaction behavior.
3. **In `final_quantity` there were records with value 'zero', records with value greater than 0 but had Final_sale amount as 0.** Are there any significance to these type of values? Can it be returned transactions or free item or any business or system specific logic
4. **There are 94 records that have purchase date after scanned date.** Could this be a result of time zone conversion, or processing delays?