**Third: Evaluate Data Quality Issues in the Data Provided**

In Part 1, I discussed the process of flattening the JSON files and applying compliance rules, resulting in the data being split into four tables:

1. Users
2. Brands
3. Receipts
4. Rewards\_Receipts.

This document outlines the **Data Quality Checks** applied to these tables and identifies any potential issues that may impact the integrity of the data.

**Data Quality Assessment – Users Table:**

Overview of data quality checks for the Users table:

1. **Schema verification:** Ensuring the Schema of the table (possibly new data) be consistent with the already established schema for the Users table.
2. **Checking for Duplicates:** Ensure there are no duplicate records in the **Users** table, particularly for columns like *user\_id*, which should uniquely identify each user.
3. **Verifying Uniqueness of Primary Key:** Confirm that the assigned primary key (*user\_id*) is unique across all records, ensuring each user is represented only once in the table.
4. **Missing Values:** Identify any columns with missing values and calculate the percentage of missing data. This helps assess the completeness of the dataset and ensures that key fields like *user\_id* and *created\_date\_time* are fully populated.
5. **Timestamp Validation:** Verify that the *lastLogin\_date\_time* field occurs after the *created\_date\_time*, ensuring chronological consistency in the data.
6. **Data Type Validation:** Check that all columns have the appropriate data types. For example, the *active* field should be a Boolean, and all date-related fields (e.g., *created\_date\_time*, *lastLogin\_date\_time*) should be properly formatted as date/time. Any missing dates should be filled with NaT (Not a Time) instead of NaN to avoid inconsistencies in date handling.
7. **Acceptable Range of values:** Ensuring the values that are to be entered would follow in the range of the values. This ensures flagging unwanted data.

**1) Schema verification**

It is expected that the table would have the following columns: *active, signUpSource, created\_date\_time, user\_id, lastLogin\_date\_time, state*. From the jupyter notebook, this is verified. The test also ensures if there are any spelling mistakes in the column names to maintain consistency.

**2) Checking for Duplicates**

It can be found that **57.17 %** of the data entries are duplicate. This is a **Data Quality Issue** that needs to be addressed. It can be resolved by deleting the duplicate records.

A screen shot of a computer code

Description automatically generated

**3) Verifying Uniqueness of Primary Key (*user\_id*)**

The primary key of the table – user\_id needs to be unique and not missing in the table.

A screen shot of a computer

Description automatically generated

This shows that the primary key *user\_id* indeed is unique and not missing in the table.

**4) Missing Values:**

Here is the information about the missing values in Users Table:

Some notable issues are with the following columns:

1. signUpSource
2. state
3. lastLogin\_date\_time

A screenshot of a computer

Description automatically generated

Analyzing the missing values in *SignupSource* and *state*:

A screenshot of a computer

Description automatically generated

3 out of 6 *signUpSource* missing values occur for the user who is a fetch-staff (Mild Data Quality Issue). The unique values for signup source are Email and Google. It is very much possible that fetch-staff can sign up using an **internal source** which is not email or google. The resources list could be expanded including possible valid sources such as **App Store**, **Referral**, and other relevant platforms, to guarantee that this field is complete for all users. If the field is still missing, it could be imputed as 'others'.

State missing values could be imputed with 'DNF' (Did not fill) or 'N/A' or 'None' appropriately (Mild Data Quality Issue).

Analyzing the missing values in *lastLogin\_date\_time*:

The missing last login data indicates that users are not immediately logging in after signing up. This suggests a potential bottleneck in the sign-up process, where users may be unable to log in right after signing up. One possible cause could be permission issues or other technical restrictions that prevent the app from opening or proceeding to the login page after email registration. Further investigation is needed to identify and address this issue to streamline the sign-up and login experience, especially for email users.

I’m curious to understand if the users who never logged in, made a purchase. One would expect that it would not.

A screenshot of a computer code

Description automatically generated

**Data Quality Issue spotted**

From the above figure,

It is interesting to see that the data suggests that **57** users with no recorded last login date are still able to scan receipts and earn bonus points. This is unexpected behavior, as we would typically expect a last login date to be present once a user interacts with the app to scan receipts. This anomaly needs to be investigated further, as it could indicate a logging issue or a flaw in the system where login events are not properly captured before allowing user actions such as receipt scanning.

**5) Timestamp validation**

It is expected that the last login date and time would always be after the creation of the account/user.

A computer code with text

Description automatically generated with medium confidence

The above figure validates that the last login datetime is always after creation of the account.

**6) Appropriate Data formatting**

Columns are expected to have the correct data types, such as *datetime* for date fields instead of *object* types. It is also important to ensure consistency in object-type columns to avoid discrepancies like ‘Google’ vs. ‘GOOGLE’.A screenshot of a computer code

Description automatically generated

* No object discrepancies or errors were found.
* Datetime columns were successfully converted from object to datetime types.
* Boolean columns were verified to ensure they contain valid boolean values.

**7) Range type checks**

The *state* column was checked to ensure all entries are valid U.S. state abbreviations, adhering to the official two-letter codes for U.S. states. The test results from the Jupyter notebook confirm that this check passed successfully.

For the date columns, validations were performed to ensure that the values fall within a developer-defined period (Example: Year 2000 to 2100). This step is essential to prevent unreasonable or erroneous dates. The notebook results show that the date range checks also passed without issues.

A number of numbers and numbers

Description automatically generated with medium confidence

Fig: 3.

However, while reviewing the distribution of the *created\_date\_tim*e variable, a concerning pattern was observed. There are significant gaps in the data, with entire years such as 2016, 2018, and 2019 missing. This sporadic distribution of user creation dates seems unusual and could indicate a **Data Quality Issue**. Further investigation is recommended to understand the cause of these gaps.

**Data Quality Assessment – Brands Table:**

This section outlines the data quality checks conducted on the **Brands** table, following a structure like that used for the **Users** table. A new additional check has been added (which discusses relationships between the columns). These checks are designed to ensure data consistency, integrity, and overall quality.

**1) Schema Verification:** The schema of the **Brands** table was validated to ensure consistency with the established schema, ensuring that any new data conforms to the expected structure. This ensures that the **Brands** table is aligned with other related tables, such as the **Users** table, and prevents schema drift.  
**2) Duplicate Records Check:** The table was examined for any duplicate records, as duplicate entries can distort analysis and lead to inaccurate insights. The data passed this check, ensuring that all records in the **Brands** table are unique.  
**3) Primary Key Uniqueness:** To maintain data integrity, it is critical that the primary key (*brand\_id*) remains unique across all records. The **Brands** was validated for primary key uniqueness, confirming that no duplicate *brand\_id* exist.  
**4) One-to-One Relationship checks:** To maintain data consistency, it is important to take an educated guess if the columns are maintaining one-to-one relationship and check if there is possible explanation of inconsistency if exists. This check explores possible human errors or data entry errors to some level that are discussed below.  
**5) Missing Values Analysis:** A detailed analysis was conducted to identify any missing values across columns. While missing values are expected in some cases, understanding which fields are incomplete can help guide future imputation strategies or data collection improvements.  
**6) Data Type Validation:** Each column in the **Brands** table was assessed to ensure it is stored in the correct data type. For instance, categorical fields like *name*and *brandCode* should be stored as strings, while fields like *topBrand* should be Boolean. Ensuring appropriate data types prevents downstream errors in analysis and reporting.

**Schema Validation, Duplicates Records Check, Primary Key Uniqueness Check:**

**A screenshot of a computer code

Description automatically generated**

A screenshot of a computer code

Description automatically generated

From the above figures, it can be shown that Data Quality in terms of Schema Validation, Primary Key Uniqueness, Duplicate records are verified.

**One-to-One Relationship Checks:**

I’ll start with an example. It’s reasonable to make an educated guess that brand name (*name*) and brand code (*brandCode*) should maintain a one-to-one relationship. If they don’t, and the variance is significant, the initial assumption may be incorrect. However, if the variance is minimal, it might indicate small errors in data entry. The notebook code below explores this further:

A screenshot of a computer

Description automatically generated

The table above shows the variance I mentioned, and it appears to be quite small. For instance, the brand name ‘Baken-Ets’ has two unique brand codes. Let’s dive into possible reasons for this occurrence by analyzing some of the brand names highlighted in the table:A screenshot of a computer code

Description automatically generated

This is a **Data Quality Issue.** The spelling and punctuation mistakes in the brandCode highlight the inconsistency in the relationship between brandCode and brand name. This data quality issue needs to be addressed, as brandCode is a crucial parameter for identifying a brand. It is a serious problem when trying to understand and analyze brand related KPIs. Additionally, brandCode serves as a foreign key in the **Rewards\_Receipts** table, as discussed in the first part of this project. To ensure data integrity, it is essential to have a consistent, error-free one-to-one mapping between brandCode and brand name.

The same approach can be applied to other column pairs where one might reasonably assume a one-to-one relationship. This has been checked for the following columns: ['name', 'brandCode', 'topBrand', 'brand\_id']A screenshot of a computer code

Description automatically generated

It could be unnecessary to go into full detail here, but a couple of inconsistencies are highlighted. for example:

A screenshot of a computer code

Description automatically generated

This shows that some of the brands like ‘Huggies’ are flagged as both ‘top brand’ and ‘not a top brand’ which is a **Data Quality Issue**. This raises the need to reconsider how *topBrand* is being assigned to each brand.

**Missing Value Checks:**

**A screenshot of a computer code

Description automatically generated**

13% of the category data is missing, which is a **Data Quality Issue**. Can the category field be imputed based on the brand name? Maybe, but there are cases like the brand ‘Nike’, which sells both shoes and apparel, making the category ambiguous. So, imputing the category just by brand name isn’t straightforward.

Exploring methods like automating data entry with reference tables that map barcode or brandCode to specific *category* and *categoryCode* could be a viable solution.

We could verify if the missing *categoryCode* is consistent across the *category* column which are present.

A screenshot of a computer program

Description automatically generated

The table above shows that for a given category (e.g., ‘Grocery’), some *categoryCode* values are missing, while others are correctly filled with ‘GROCERY’.

**Data Quality Issue** has been spotted. The table above reveals that for certain categories, the *categoryCode* is missing in some instances, while it is properly filled for the same category in other records. This inconsistency highlights a data quality issue where *categoryCode* has not been filled in appropriately, despite having clear information on how it should be completed.

A similar logic could be applied for brandCode information given the brand name. Firstly, 23% of the brandCodes are missing. This is a huge **Data Quality issue** as *brandCode* is going to play a key role in connecting tables (connecting **Rewards\_Receipts** table in this project). The missing values need to be imputed appropriately. Coming back to the logic, here is the table that shows if the missing brandCodes is consistent everywhere given a brand name.

A screenshot of a phone

Description automatically generated

the brandCode is missing in some instances, while it is properly filled for the same brand in other records. This inconsistency highlights a data quality issue where brandCode has not been filled appropriately, despite having clear information on how it should be completed.

Missing top\_brand information can be flagged as a mild Data Quality Issue. A third category, such as ‘undecided,’ could be introduced to handle these cases.

**Consistent Data Format Checks:**

1. TopBrand Column needs to be Boolean. In the notebook, *topBrand* column is converted to Boolean type.
2. Barcodes need to follow a consistent format (For example, GS1 barcodes should be 12 digits)

A screen shot of a computer code

Description automatically generated

**Data Quality Assessment – Receipts table:**

This section outlines the data quality checks conducted on the **Receipts** table, following a structure like that used for the **Users** and **Brands** table. These checks are designed to ensure data consistency, integrity, and overall quality.

**Standard Data Quality Checks (Similar to Users and Brands tables):**

The **Receipts** table underwent key data quality checks, following a similar structure used for the **Users** and **Brands** tables. This included schema verification, duplicate detection, primary key uniqueness validation, assessment of missing values, and verifying if the dates variables would make sense. Data types were reviewed to confirm proper formatting for key fields, such as *receipt\_id* and date columns.

**Additional Checks:**

1. **Chronology of Date variables:** Given date variables like modified, purchased, and scanned, it’s generally expected that events occur in a chronological order.
2. **Foreign Key *userId* Integrity check:** It is important to ensure that the *user\_id* column, which links the **Receipts** and **Users** tables, contains only *user\_id* values present in the **Users** table. This check guarantees proper data linkage between the two tables.
3. **Range of the Numerical Data checks:** The data can be analyzed to assess the distribution, and a deeper investigation can be carried out if anything unusual is flagged.

Standard Data Quality Checks:

From the code given in the Jupyter notebook, it can be verified that the check is passed.

A screenshot of a computer code

Description automatically generated

**Missing Value Checks:**

Here is the Missing Values percentage for the **Receipts** table:

A screenshot of a computer screen

Description automatically generated

**Understanding the reason for missing points related variables:**

In the table below, all missing points statuses are either submitted, pending, flagged, or rejected (essentially not finished), which makes sense since the receipt might be awaiting points or a decision not to reward, and hence the missing values.

A screenshot of a computer

Description automatically generated

Given the above information, I would not flag the missing points related values, or the values could be imputed to 0.

**Understanding the reason for missing items list:**

The output snippet of missing values shows that apart from Receipt ID, Rewards Status, User ID, and the created-scanned-modified date and time, most other fields are significantly missing. This suggests that while a registered user was able to create and scan a receipt, no contents were captured. It could be that the receipt scanning algorithm is not capturing the contents of some receipts. This could be due to illegible receipt content or technical issues with the computer vision scanning algorithm. While I wouldn’t flag this as a major data quality issue, I would store this information and use it to notify users that their receipts couldn’t be scanned, prompting them to try again later.

A screenshot of a computer

Description automatically generated

**Chronology of Date variables check:**

Here is the rough chronological order the process of scanning a receipt would follow: first, the items are purchased (1) -> then the instance is created (2) -> followed by the contents being scanned (3)-> then finished (4)-> and finally modified (5). Here is the code that checks the order:

A screen shot of a computer code

Description automatically generated

**1.6%** of the data violates the discussed chronology. Since the proportion is less, I would flag this as Mild Data Quality Issue. But I would deep dive into understanding if there is an appropriate reason for the anomaly.

**Foreign Key Integrity Check (userId):**

Since userId is the foreign key, it is expected that the userId is a subset of the **Users** table. However, from the code snippet below, it shows that **45.3%** of the User ids are not present in the **Users** table. This is a serious **Data Quality Issue**. This suggests that the **Users** table is not being refreshed in a timely manner, and there is a need to ensure consistency.

A screen shot of a computer code

Description automatically generated

**Range of Numerical Data Check:**

A preliminary check could be applied to the numerical values to understand the basic distribution of the variables. Here is the snippet of the output.

A screenshot of a computer screen

Description automatically generated

From the table above, we can see that the median is much smaller than the average, indicating a right-skewed distribution. This suggests the presence of outliers in the data. While outliers don't necessarily signal a major data quality issue, they could be flagged as a Mild Data Quality Concern. These outliers should be monitored closely for any signs of fraudulent scans or unusual patterns, but they don’t indicate a serious data quality problem on their own.

**Range of Date variables check:**

A date range from **2000-01-01** to **2100-12-31** was selected, and a check was implemented to ensure all date variables fall within this range. The output from the notebook confirms the success of this check.

A close up of text

Description automatically generated

**Data Quality Assessment - Rewards\_Receipts table:**

This section outlines the data quality checks conducted on the **Rewards\_Receipts** table, following a structure like that used for the previous tables. These checks are designed to ensure data consistency, integrity, and overall quality.

**Standard Data Quality Checks (Similar to Users and Brands tables):**

The **Rewards**\_**Receipts** table underwent key data quality checks, following a similar structure used for the **Users** and **Brands** tables. This included schema verification, duplicate detection, primary key uniqueness validation, assessment of missing values, and verifying if the dates variables would make sense. Data types were reviewed to confirm proper formatting for key fields.

**Additional Checks:**

1. **Missing Foreign Key values and a possible solution:** The variables related to the brand are **brandCode** and **barcode**, but there are many missing values. A potential method for imputing missing brandCode is discussed.
2. **Foreign Key *userId* Integrity check:** It is important to ensure that the *brandCode* column, which links the **Rewards\_Receipts** and **Brands** tables, contains only *brandCode* values present in the **Brands** table. This check guarantees proper data linkage between the two tables.
3. **Data Format for Barcodes check:** The data can be analyzed to assess the distribution, and a deeper investigation can be carried out if anything unusual is flagged.

**Standard Data Quality Checks:**

Here is the code and output snippet that conducted the standard data quality checks. The checks are a clean pass.

A screenshot of a computer code

Description automatically generated

**Missing Value Checks:**

A significant portion of the data is missing, with some occurring in groups, such as the *'userflagged\_'* variables, which have many missing values. A reasonable assumption is that these missing values correspond to instances where the user did not flag anything. Similarly, missing values in *pointsNotAwardedReason* might indicate that points were awarded. Another similar group is *OriginalMetaBrite*. It should be reviewed whether it's valuable to retain such largely missing data, depending on the business use case. Further investigation is needed to determine the reason for these missing values. Since the foreign key used is *brandCode*, we can dig deeper to explore potential improvements.

**Missing brandCode and Imputation techniques:**

Out of many variables present in the table, only two of the columns seemed to connect the **Brands** table: *brandCode* and *barcode.* I have reservations about barcode being the connector since barcode is specific to a product not a brand. We also observed that there are many missing values in the 2 variables. We can do a small comparison to check which column has the most number of intersections between **Rewards\_Receipts** table and **Brand** table. From the figure below (FILL FIG NUMBER), even though the intersection number is less, brandCode has most intersections. It reinforces my idea of choosing brandCode to be the foreign key. Can this number be improved? I propose a union of two imputation techniques:

1) The description field often contains brand information, particularly in the first word. Extracting that could serve as a potential imputation method for missing *brandCode* in the rewards\_receipts table.

2) Impute missing *brandCode* in the brands table using the brand's name.

A screenshot of a computer code

Description automatically generated

Low intersection numbers suggests that there is **Data Quality Issue.**

My observations pertaining to the first word of the description for missing brandCode values:A screenshot of a computer

Description automatically generatedA screenshot of a table

Description automatically generated

Because of the above imputation method, brands like KLARBRUNN (128 entries), HUGGIES (93 entries), MUELLER (44 entries) etc. are coming into picture. Of course, the imputation is not perfect (for example: ITEM NOT FOUND would become ITEM). Therefore, when performing brand related queries, one needs to keep in mind about this imputation method.

Similarly, we can see in brands table, how Brand name be used to impute brandCode value:

A screenshot of a computer

Description automatically generated

Comparing if there are any improvements in terms of intersections:

A screenshot of a computer code

Description automatically generated

There has been an increase of over 200% due to the new imputation techniques. But is this the best approach? Not necessarily, though it is better than leaving the brandCodes missing. Other potential methods to explore include extracting brand information directly from the barcodes, especially if there is a pattern, such as the first four digits indicating something relevant. An efficient way of pairing barcodes with brand information is quite necessary. The proper approach would be to map the barcodes to their respective brands, store that information, and then use it to accurately extract brand details.

**Foreign Key Integrity Check:**

Let's understand about the brandCodes which are present in rewards\_receipts table but not present in the Brands table, verifying foreign key integrity. Note that the follow test is done before the proposed imputation.

Here is a sample list of Brand Codes that are present in **Rewards\_Receipts** table but not in **Brands** table.

A screen shot of a computer code

Description automatically generated

Are the Brand Codes actually missing?  
Answer: No.

After some initial eyeball checks, a few observations were made:

1) **'7UP'** in the rewards\_receipts table and **'7 up'** in brands\_df refer to the same brand but lack consistency in structure.  
2) **"BEN AND JERRYS"** in **Rewards\_Receipts** table and **"BEN & JERRY'S"** in **Brands** table are also inconsistent in spelling and punctuation.  
3) These are just a few examples of errors, whether in spelling or punctuation, which impact the consistency of brandCodes.

Given that brandCode is used as a foreign key, this poses a significant **Data Quality Issue.**Creating a data pipeline to directly replace eyeball checks for grammatical errors would be challenging. The core issue, as mentioned earlier, lies in the lack of a proper mapping between barcode and brand information. Establishing this mapping would greatly improve data quality.

**Barcode format Consistency check:**

Given the presence of barcodes, it is essential that they follow a consistent format:   
1) numeric   
2) 12 digits long.

From the code from Jupyter notebook, it can be shown that the datatype is an object which is a **Data Quality Issue.**

Also the 12 digit consistency has failed as well.

A screen shot of a computer code

Description automatically generated

Some of the barcodes contain only 4 digits or 11 digits. Some of the barcodes contain letters which violates the check and shows **Data Quality Issue.**

**Automating the Data Quality Checks:**

What we discussed were some important data quality checks. It’s useful to write Python pandas code in a Jupyter notebook and carefully perform these checks one by one. However, in practice, especially in high-volume data environments, manually writing checks like this might not be scalable or efficient. To address this, building automated data quality pipelines becomes essential. These pipelines ensure that incoming data is automatically validated, and any errors or anomalies are flagged for further investigation. Tools like **Great Expectations** provide robust frameworks for defining, validating, and documenting data quality rules, while **Airflow DAGs** can orchestrate these pipelines, ensuring checks run on schedule or in response to new data.

Here is a sample Great Expectation code to check the uniqueness of the Primary key *user\_id* in **Users** table. The expectation that is used is **“ge.expectations.ExpectColumnValuesToBeUnique”**

A screenshot of a computer program

Description automatically generated

Here is a list of other useful ge.expectations

**“ExpectColumnDistinctValuesToBeInSet”; “ExpectColumnValuesToNotBeNull”; “ExpectColumnDistinctValuesToBeInSet”; “ExpectColumnMaxToBeBetween”** etc.

Write about WAP: