This Notebook

In this notebook, we'll explore relationships between and integrity in data stored in json from the files `receipts.json` , `users.json` , and `brands.json` . Our stakeholder, Fetch Rewards, has asked us to:

1. Develop a Structured Relational Data Model
2. Query the Database using SQL to provide business intelligence
3. Evaluate the data for issues in quality

At the end of this notebook, there will be a summary answering the main data evaluation questions, for the purpose of meeting requirement 3.

Exploring Provided Data Files

To be able to provide answers to the above, we must first have a strong understanding of the information available. Exploring the data will allow us to determine how we can structure our data warehouse by finding relationships between the tables, reveal what we may need to ask for from our stakeholders in providing insight, and expose problems with how the data is currently stored.

In [1]:
```
import pandas as pd
import json
```

In [2]:
```
receipts_df = pd.read_json('./data/receipts.json.gz', lines = 'true')
users_df = pd.read_json('./data/users.json.gz', lines = 'true')
brands_df = pd.read_json('./data/brands.json.gz', lines = 'true')
```

Pandas is a good tool for looking at and working with tabular data, storing data in SQL table-like structures called dataframes. We've read our files into these dataframes and can explore the columns. For receipts, we are given information about the columns:

```
    _id: uuid for this receipt
    bonusPointsEarned: Number of bonus points that were awarded upon rec
eipt completion
    bonusPointsEarnedReason: event that triggered bonus points
    createDate: The date that the event was created
    dateScanned: Date that the user scanned their receipt
    finishedDate: Date that the receipt finished processing
    modifyDate: The date the event was modified
    pointsAwardedDate: The date we awarded points for the transaction
    pointsEarned: The number of points earned for the receipt
    purchaseDate: the date of the purchase
    purchasedItemCount: Count of number of items on the receipt
    rewardsReceiptItemList: The items that were purchased on the receipt
    rewardsReceiptStatus: status of the receipt through receipt validati
on and processing
    totalSpent: The total amount on the receipt
    userId: string id back to the User collection for the user who scann
ed the receipt
```

In [3]:
```
receipts_df.head()
```

Out[3]:

	_id	bonusPointsEarned	bonusPointsEarnedReason	createDate	d
0	{'$oid': '5ff1e1eb0a720f0523000575'}	500.0	Receipt number 2 completed, bonus point schedu...	{'$date': 1609687531000}	160!
1	{'$oid': '5ff1e1bb0a720f052300056b'}	150.0	Receipt number 5 completed, bonus point schedu...	{'$date': 1609687483000}	160!
2	{'$oid': '5ff1e1f10a720f052300057a'}	5.0	All-receipts receipt bonus	{'$date': 1609687537000}	160!
3	{'$oid': '5ff1e1ee0a7214ada100056f'}	5.0	All-receipts receipt bonus	{'$date': 1609687534000}	160!
4	{'$oid': '5ff1e1d20a7214ada1000561'}	5.0	All-receipts receipt bonus	{'$date': 1609687506000}	160!

I'm noticing that a lot of the columns, such as the date columns and the id column, have nested structures. We will have to ask whether it is necessary to store information in this way, or if it can be cleaned in retrieval or prior to database storage, as it will add extra steps in processing. I will also need information on how to convert these data codes to translate these date codes into an interpretable format. First questions are regarding the full content of `rewardsReceiptItemList` and `bonusPointsEarnedReason` , so lets check the first entry of each of those.

In [4]:
```
receipts_df['rewardsReceiptItemList'][0]
```

Out[4]:
```
[{'barcode': '4011',
  'description': 'ITEM NOT FOUND',
  'finalPrice': '26.00',
  'itemPrice': '26.00',
  'needsFetchReview': False,
  'partnerItemId': '1',
  'preventTargetGapPoints': True,
  'quantityPurchased': 5,
  'userFlaggedBarcode': '4011',
  'userFlaggedNewItem': True,
  'userFlaggedPrice': '26.00',
  'userFlaggedQuantity': 5}]
```

Hmm.. 'ITEM NOT FOUND'? Also, this one column has a lot of information in it. It would probably be best to make this its own table in the database, and add a column for the `receipt_id` as a foreign key so that it could be linked back to the receipts table.

In [5]:
```
receipts_df['bonusPointsEarnedReason'][0]
```

Out[5]:
```
'Receipt number 2 completed, bonus point schedule DEFAULT (5cefdcacf3693e0b50e8
3a36)'
```

From this entry, it seems as if there is a increasing return for customers based on the number of receipts submitted, as well as different progression schedules. We will have to ask our stakeholder about this. I'm wondering if `bonusPointsEarned` and `pointsEarned` are the same, as they look equivalent in the first few rows.

In [6]:
```
(receipts_df['bonusPointsEarned'] == receipts_df['pointsEarned']).value_counts()
```

Out[6]:
```
False    768
True     351
dtype: int64
```

Not entirely, but more information is required. There are a few other columns that look like they might always be the same, such as `createDate` and `dateScanned` .

In [7]:
```
(receipts_df['createDate'] == receipts_df['dateScanned']).value_counts()
```

Out[7]:
```
True     1118
False       1
dtype: int64
```

Hmm.. almost always. They may not necessarily be, I suppose, if someone started working on an event but got distracted and for some reason scanned the receipt in later days.

In [8]:
```
receipts_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1119 entries, 0 to 1118
Data columns (total 15 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   _id                      1119 non-null   object 
 1   bonusPointsEarned        544 non-null    float64
 2   bonusPointsEarnedReason  544 non-null    object 
 3   createDate               1119 non-null   object 
 4   dateScanned              1119 non-null   object 
 5   finishedDate             568 non-null    object 
 6   modifyDate               1119 non-null   object 
 7   pointsAwardedDate        537 non-null    object 
 8   pointsEarned             609 non-null    float64
 9   purchaseDate             671 non-null    object 
 10  purchasedItemCount       635 non-null    float64
 11  rewardsReceiptItemList   679 non-null    object 
 12  rewardsReceiptStatus     1119 non-null   object 
 13  totalSpent               684 non-null    float64
 14  userId                   1119 non-null   object 
dtypes: float64(4), object(11)
memory usage: 131.3+ KB
```

In [9]:
```
receipts_df.rewardsReceiptStatus.value_counts()
```

Out[9]:
```
FINISHED     518
SUBMITTED    434
REJECTED      71
PENDING       50
FLAGGED       46
Name: rewardsReceiptStatus, dtype: int64
```

So we are seeing that while some columns have values for every entry, there are a lot of `null` values in our dataframe. It is beneficial that every entry has a receipt status and can be attached to a user by an id, and there is a status for every event. A lot of the `null` values seem like they will come from entries that haven't been filled yet because their status is still 'SUBMITTED'. However, the missing values certainly beg some questions, such as, why are their different numbers of values in `pointsEarned` and `bonusPointsEarned` and why do both columns exist as they appear to be equivalent, and why is the number of available entries not equal in either case to the number of entries in the `pointsAwardedDate` column, or for that matter the number of real entries in `finishedDate` or the amount of receipts categorized as `'FINISHED'` in `rewardsReceiptsStatus` ? Broadly, if the solution is working as intended but there are frequently missing values and inconsistencies, what is really necessary to store going forward? With regards to data types, many are object (strings), which doesn't need to be true for things that could be converted to dates, if extracted from a nested structure.

I will also make a note here that one of the business questions asks for aggregations related to `rewardsReceiptStatus` of either 'Rejected' or 'Accepted' but 'Accepted' does not appear in this column as a value. Based on 'Finished' being present and similar in number to the number of real values for columns related to points being awarded, I will guess that 'Accepted' and 'Finished' are synonymous. Further investigation could help verify this guess, such as comparing rows where the value is 'Finished' to see if they coincide with rows where points are awarded.

For users, we are given column information:

```
    _id: user Id
    state: state abbreviation
    createdDate: when the user created their account
    lastLogin: last time the user was recorded logging in to the app
    role: constant value set to 'CONSUMER'
    active: indicates if the user is active; only Fetch will de-activate
an account with this flag
```

In [10]:
```
users_df.head()
```

Out[10]:

	_id	active	createdDate	lastLogin	role	signUpSource	st
0	{'$oid': '5ff1e194b6a9d73a3a9f1052'}	True	{'$date': 1609687444800}	{'$date': 1609687537858}	consumer	Email	
1	{'$oid': '5ff1e194b6a9d73a3a9f1052'}	True	{'$date': 1609687444800}	{'$date': 1609687537858}	consumer	Email	
2	{'$oid': '5ff1e194b6a9d73a3a9f1052'}	True	{'$date': 1609687444800}	{'$date': 1609687537858}	consumer	Email	
3	{'$oid': '5ff1e1eacfcf6c399c274ae6'}	True	{'$date': 1609687530554}	{'$date': 1609687530597}	consumer	Email	
4	{'$oid': '5ff1e194b6a9d73a3a9f1052'}	True	{'$date': 1609687444800}	{'$date': 1609687537858}	consumer	Email	

Similarly nested structures as to receipts table. One issue present here, due to the nested structures, is that the users table `_id` column is nested in a dictionary, while its existence as a foreign key in the receipts table is not nested, so trying to look them up by looking for where they are equal doesn't work. You can workaround it by accessing the value in the dictionary, but that requires extra steps and will be confusing to people who are new to the database and how the values are stored. Also, do we need to have the `role` column if it is always consumer? Maybe `role` is referenced elsewhere.

In [11]:
```
users_df['signUpSource'].value_counts()
```

Out[11]:
```
Email     443
Google      4
Name: signUpSource, dtype: int64
```

Like SSO 'click to sign in with Google?' vs setting up a sign on with your own email?

In [12]:
```
users_df['active'].value_counts()
```

Out[12]:
```
True     494
False      1
Name: active, dtype: int64
```

What happens when active is false?

In [13]:
```
users_df['state'].unique()
```

Out[13]:
```
array(['WI', 'KY', 'AL', 'CO', 'IL', nan, 'OH', 'SC', 'NH'], dtype=object)
```

In [14]:
```
users_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 495 entries, 0 to 494
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   _id           495 non-null    object
 1   active        495 non-null    bool  
 2   createdDate   495 non-null    object
 3   lastLogin     433 non-null    object
 4   role          495 non-null    object
 5   signUpSource  447 non-null    object
 6   state         439 non-null    object
dtypes: bool(1), object(6)
memory usage: 23.8+ KB
```

In [15]:
```
users_df.lastLogin.isnull().value_counts()
```

Out[15]:
```
False    433
True      62
Name: lastLogin, dtype: int64
```

In this table we seem to have most of our information. Perhaps `lastLogin` is sometimes `null` since they have only ever used the service the first time they created it. State may not be mandatory, or people signed up from outside the US? Anyway, while a lot of this information is nice to have, probably okay we are missing some values, though we can look into the reason why they may not have been recorded.

For brands, we were given the information:

```
    _id: brand uuid
    barcode: the barcode on the item
    brandCode: String that corresponds with the brand column in a partne
r product file
    category: The category name for which the brand sells products in
    categoryCode: The category code that references a BrandCategory
    cpg: reference to CPG collection
    topBrand: Boolean indicator for whether the brand should be featured
as a 'top brand'
    name: Brand name
```

In [16]:
```
brands_df.head()
```

Out[16]:

	_id	barcode	category	categoryCode	
0	{'$oid': '601ac115be37ce2ead437551'}	511111019862	Baking	BAKING	'601ac114be37ce2ea
1	{'$oid': '601c5460be37ce2ead43755f'}	511111519928	Beverages	BEVERAGES	'5332f5fbe4b03c9a
2	{'$oid': '601ac142be37ce2ead43755d'}	511111819905	Baking	BAKING	'601ac142be37ce2ea
3	{'$oid': '601ac142be37ce2ead43755a'}	511111519874	Baking	BAKING	'601ac142be37ce2ea
4	{'$oid': '601ac142be37ce2ead43755e'}	511111319917	Candy & Sweets	CANDY_AND_SWEETS	'5332fa12e4b03c9a

Nested structures again. Also, it is not immediately apparent the differences between `category` and `categoryCode` , and `name` and `brandCode` ; perhaps we can condense them or retain the more useful of the two. Maybe the barcode here can be linked to the barcodes in `rewardsReceiptItemList` , otherwise I'm not really sure how to connect this table back to the others. Let's also investigate `cpg` .

In [17]:
```
brands_df['cpg'][0]
```

Out[17]: `{'$id': {'$oid': '601ac114be37ce2ead437550'}, '$ref': 'Cogs'}`

Without further info, I can't be sure what this means. It says reference to CPG collection in our column information, but we are not currently privy to that data as of now.

In [18]:
```
brands_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1167 entries, 0 to 1166
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   _id           1167 non-null   object 
 1   barcode       1167 non-null   int64  
 2   category      1012 non-null   object 
 3   categoryCode  517 non-null    object 
 4   cpg           1167 non-null   object 
 5   name          1167 non-null   object 
 6   topBrand      555 non-null    float64
 7   brandCode     933 non-null    object 
dtypes: float64(1), int64(1), object(6)
memory usage: 73.1+ KB
```

So there are more `category` and `name` entries than their respective Code columns in both cases. Maybe this is a result of codes not being developed internally for all categories? It also appears that the `topBrand` bool is often missing. Between these notes, I would guess that there either are some items bought that have not been found in an internal reference and should be added and categorized, if having this information is useful. They may also have been bought from e-commerce sites that do not provide this information.

In [19]:
```
brands_df.loc[brands_df.name.duplicated(keep = False)].sort_values('name')
```

Out[19]:

	_id	barcode	category	categoryCode	
848	{'$oid': '585a961fe4b03e62d1ce0e76'}	511111701781	Snacks	NaN	{'ref' :' '53:
574	{'$oid': '5d9d08d1a60b87376833e348'}	511111605546	Snacks	NaN	{'ref' :' '53:
140	{'$oid': '5a4d23dae4b0bcb2c74ea77e'}	511111000518	Beverages	NaN	{'ref' :' '53:
740	{'$oid': '5d601d74a3a018514994f422'}	511111004912	Snacks	NaN	{'ref' :' '53e
1007	{'$oid': '5d658ffa6d5f3b23d1bc7914'}	511111205227	NaN	NaN	{'ref' :' '53e
1006	{'$oid': '5d66d597a3a018093ab34726'}	511111805298	Magazines	NaN	{'ref' :' '5d5
1163	{'$oid': '5dc1fca91dda2c0ad7da64ae'}	511111706328	Breakfast & Cereal	NaN	{'ref' :' '53e
1081	{'$oid': '5dc2d9d4a60b873d6b0666d2'}	511111206330	Breakfast & Cereal	NaN	{'ref' :' '53:
194	{'$oid': '5d6415d5a3a018514994f429'}	511111605058	Magazines	NaN	{'ref' :' '5d5
596	{'$oid': '5f298852be37ce7958c5952d'}	511111915287	Magazines	MAGAZINES	{'ref' :' '5d6
1074	{'$oid': '5c7d9cb395144c337a3cbfbb'}	511111707202	Baby	BABY	{'ref' :' '545
628	{'$oid': '5bd2011f90fa074576779a17'}	511111704652	Baby	NaN	{'ref' :' '550
846	{'$oid': '5e710dd5ee7f2d0b35b2a193'}	511111314097	Dairy & Refrigerated	DAIRY_AND_REFRIGERATED	{'ref' :' '5e5
176	{'$oid': '592486bee410d61fcea3d12d'}	511111700814	Dairy	NaN	{'ref' :' '53e
339	{'$oid': '5e5ff265ee7f2d0b35b2a18f'}	511111914051	Health & Wellness	NaN	{'ref' :' '53e

	_id	barcode	category	categoryCode	
64	{'$oid': '5da609991dda2c3e1416ae90'}	511111805854	Health & Wellness	NaN	{'ref' :' '53e
978	{'$oid': '5db3288aee7f2d6de4248977'}	511111312949	Baby	NaN	{'ref' :' '550t
126	{'$oid': '5bd201a990fa074576779a19'}	511111104698	Baby	NaN	{'ref' :' '550t
282	{'$oid': '5332f608e4b03c9a25efd0c1'}	511111903901	NaN	NaN	{'ref' :' '53e
1116	{'$oid': '5d66e07da3a018093ab3472d'}	511111205500	Beverages	NaN	{'ref' :' '53
477	{'$oid': '5bcdfc5a965c7d66d92731e9'}	511111304616	Beverages	NaN	{'ref' :' '53e
1025	{'$oid': '5bcdfc5990fa074576779a15'}	511111804604	Beverages	NaN	{'ref' :' '5a7

I was looking to verify there was only one of each brand in the brands table to determine the table relationships, and it appears there are some dulicate brands. Some of them have conflicting information, such as in `topBrand` and `brandCode` . Perhaps due to a relationship change with partners? These internal definitions should be updated. Additionally, while I suspected before that `barcode` may be the item barcode and we could use it to link the brand table with the receipts table, seeing it used as a brand code makes this feel less likely, but I'll still check. Let's do that now.

Investigating Brands table Relationship with Receipts Table

From the provided columns, it isn't immediately obvious how brands can be joined with the other two tables. So, we're going to search some of the nested structures to see if we can find a way.

In [20]:
```
item_lists = receipts_df.rewardsReceiptItemList.copy()
```

In [21]:
```
item_lists.dropna(inplace = True)
item_lists
```

Out[21]:
```
0       [{'barcode': '4011', 'description': 'ITEM NOT ...
1       [{'barcode': '4011', 'description': 'ITEM NOT ...
2       [{'needsFetchReview': False, 'partnerItemId': ...
3       [{'barcode': '4011', 'description': 'ITEM NOT ...
4       [{'barcode': '4011', 'description': 'ITEM NOT ...
                              ...
1106    [{'barcode': 'B076FJ92M4', 'description': 'mue...
1112    [{'barcode': 'B076FJ92M4', 'description': 'mue...
1113    [{'barcode': 'B076FJ92M4', 'description': 'mue...
1114    [{'barcode': 'B076FJ92M4', 'description': 'mue...
1117    [{'barcode': 'B076FJ92M4', 'description': 'mue...
Name: rewardsReceiptItemList, Length: 679, dtype: object
```

In [22]:
```
item_lists.apply(lambda x: len(x)).value_counts()
```

Out[22]:
```
1      377
2      118
5       77
4       30
11      13
10      11
9       10
8        3
6        2
7        2
125      2
99       2
91       1
108      1
101      1
21       1
83       1
47       1
39       1
25       1
116      1
114      1
459      1
450      1
154      1
381      1
217      1
203      1
194      1
185      1
183      1
176      1
155      1
148      1
124      1
147      1
146      1
141      1
137      1
131      1
130      1
127      1
126      1
123      1
Name: rewardsReceiptItemList, dtype: int64
```

Huh.. many have more than one item, and some have a lot of items.

In [23]:
```
item_lists[0][0]
```

Out[23]:
```
{'barcode': '4011',
 'description': 'ITEM NOT FOUND',
 'finalPrice': '26.00',
 'itemPrice': '26.00',
 'needsFetchReview': False,
 'partnerItemId': '1',
 'preventTargetGapPoints': True,
 'quantityPurchased': 5,
 'userFlaggedBarcode': '4011',
 'userFlaggedNewItem': True,
 'userFlaggedPrice': '26.00',
 'userFlaggedQuantity': 5}
```

In [24]:
```
barcodes = []
for lst in item_lists:
    for item in lst:
        try:
            item['barcode']
            barcodes.append(item['barcode'])
        except:
            continue
```

In [25]:
```
print(len(barcodes))
print(barcodes[:10])
```

```
3090
['4011', '4011', '028400642255', '4011', '4011', '1234', '4011', '04600083251
7', '4011', '4011']
```

Okay, now time to see if these bar codes pop up in any of the brand barcodes.

In [26]:
```
brands_df['inBarcodes'] = brands_df.barcode.apply(lambda x: 1 if x in barcodes el
```

In [27]:
```
brands_df.inBarcodes.value_counts()
```

Out[27]:
```
0    1167
Name: inBarcodes, dtype: int64
```

Okay, so it doesn't seem like you can join the tables in this manner. I do have one more quick thing to try, which is to see if there are any keys in any of the item lists that are anything like brand.

In [28]:
```
unique_keys = []
for lst in item_lists:
    for item in lst:
        for key in item.keys():
            if key not in unique_keys:
                unique_keys.append(key)
```

In [29]:
```
unique_keys
```

Out[29]:
```
['barcode',
 'description',
 'finalPrice',
 'itemPrice',
 'needsFetchReview',
 'partnerItemId',
 'preventTargetGapPoints',
 'quantityPurchased',
 'userFlaggedBarcode',
 'userFlaggedNewItem',
 'userFlaggedPrice',
 'userFlaggedQuantity',
 'needsFetchReviewReason',
 'pointsNotAwardedReason',
 'pointsPayerId',
 'rewardsGroup',
 'rewardsProductPartnerId',
 'userFlaggedDescription',
 'originalMetaBriteBarcode',
 'originalMetaBriteDescription',
 'brandCode',
 'competitorRewardsGroup',
 'discountedItemPrice',
 'originalReceiptItemText',
 'itemNumber',
 'originalMetaBriteQuantityPurchased',
 'pointsEarned',
 'targetPrice',
 'competitiveProduct',
 'originalFinalPrice',
 'originalMetaBriteItemPrice',
 'deleted',
 'priceAfterCoupon',
 'metabriteCampaignId']
```

Huh okay, so some do contain `brandCode` . We can check that.

In [30]:
```python
brandCodes = []
for lst in item_lists:
    for item in lst:
        try:
            item['brandCode']
            brandCodes.append(item['brandCode'])
        except:
            continue
```

In [31]:
```
print(len(brandCodes))
print(brandCodes[:10])
print(pd.Series(brandCodes).nunique())
```

```
2600
['MISSION', 'BRAND', 'KRAFT EASY CHEESE', 'PEPSI', 'DORITOS', 'KLEENEX', 'WINGS
TOP', 'WINGSTOP', 'BRAND', 'BRAND']
227
```

In [32]:
```
brands_df['inBrandCodes'] = brands_df.brandCode.apply(lambda x: 1 if x in brandCo
brands_df['inBrandName'] = brands_df.name.apply(lambda x: 1 if x in brandCodes el
print(brands_df.inBrandCodes.value_counts())
print(brands_df.inBrandName.value_counts())
```

```
0    1125
1      42
Name: inBrandCodes, dtype: int64
0    1164
1       3
Name: inBrandName, dtype: int64
```

Wow! So we found a few. I'm guessing the few in `name` are some of the ones where the brand code and name are equivalent. It seems that these files are maybe subsections of a full database, so maybe this would be a decent way to join the brands and receipts tables.. however, we have also seen that some item lists do not contain `brandCode` , so it wouldn't always work. Improving the consistency of this connection would benefit our database structure. One way this could be done is by adding brands on sold items that arent already in our brand table to the table, being careful not to duplicate brands. Knowing that the brands table also connects to a products table and CPG table, maybe there are ways to join to the receipts table that are easier and more obvious, such as through the prdoucts/items table. Also, having a list of all the possible keys in `rewardsReceiptItemList` again begs the question of what is really important and what should we keep, and for the ones that are important, why don't they appear in all of the entries?

Summary of Data Evaluation

We were provided with some questions to lead our communication with our stakeholders. We will use them here to summarize our thoughts regarding the explored data.

1. What questions do you have about the data?

   There is a long list, but my main questions revolve around the way the data is collected, what the company uses the data for now and what they imagine it might be useful for in the future. The first, because it will help with issues in how the data is stored, such as data existing in nested structures which makes accessing complicated. Can we extract the useful information, such as the id string, from these structures? We can improve the step where we extract the data, or the step where we process it for loading into the database, to make sure it is easily useable when retrieved. The last two questions, because they will inform what we need to continue retrieving, and how to best structure it in storage, again so that retrieval is easy and comprehensible when analysis needs to be done. For example, in the rewardsReceiptsItemsList column, which contains all the bought items from that receipt, there are many unique bits of information, depending on the item. What is useful to keep, and how can we make it more obvious that that information exists? Finally, for some specific questions, the questions 1, 2, 5, and 6 asked by the stakeholder can technically be answered, but as I have discovered, brand information is linked to so few items purchased, so the answer wouldn't necessarily be representative of the entire data collection. How can we make sure that this information is collected for more items going forward?

2. How did you discover the data quality issues?

   I discovered the data quality issues by loading the data from the given files into pandas dataframes in python, where I could easily view and work with them. I then was able to see the values in the different columns, as well as explore missing and duplicate values, as well as manipulate entries to check for relationships between columns within the same table and across tables.

3. What do you need to know to resolve the data quality issues?

   I need to know what the data is being used for and what information is important to retain. With those answers, I can reduce redundancy and remove irrelevant information. Besides that, working with the step of the process where the data is retrieved I can improve data quality before it is entered into tables, eliminating issues with respect to how information is stored and making sure desirable features are present.

4. What other information would you need to help you optimize the data assets you're trying to create?

   Information from the referenced Product and CPG tables would be useful, as well as what the date codes in the various date columns translate to.

5. What performance and scaling concerns do you anticipate in production and how do you plan to address them?

   By resolving issues with data integrity, we future proof data storage. If we confirm what information we currently need and what we may need and what will be actionable on in the future, we can make sure we aren't taking up space storing unnecessary data. Further, if the data is stored in an easily comprehensible way, it will eliminate problems in the future related to troubleshooting due to poor quality or confusing structure. As the number of entries grows, it becomes harder to nail down incorrect entries and understand why data isn't being accessed when expected, so getting it right at the small scale helps manage the rise of issues in the future. I plan to address current problems by narrowing down the collected information and documenting and structuring it clearly. For example, I suggest creating a new table for the database with all individual items sold, with receipt_id as a key that you can join to the receipts table to make that information more visible, along with nailing down the columns that should be present in that table.

Getting dataframe for answering business questions 3 and 4

While the SQL query would get this answer should all the information be stored in a database, I can quickly use the dataframes to answer the question as well. I'll do this to be able to provide an answer in the email to the stakeholder.

In [33]:
```
receipts_acc_rej = receipts_df[['rewardsReceiptStatus','purchasedItemCount','tota
                                                     | (receipts_df['rewardsReceiptStatus'] == 'REJ
```

In [34]:
```
receipts_acc_rej.head()
```

Out[34]:

	rewardsReceiptStatus	purchasedItemCount	totalSpent
0	FINISHED	5.0	26.0
1	FINISHED	2.0	11.0
2	REJECTED	1.0	10.0
3	FINISHED	4.0	28.0
4	FINISHED	2.0	1.0

In [35]:
```
receipts_acc_rej.rewardsReceiptStatus.value_counts()
```

Out[35]:
```
FINISHED    518
REJECTED     71
Name: rewardsReceiptStatus, dtype: int64
```

In [36]:
```
acc_rej_summary = receipts_acc_rej.groupby('rewardsReceiptStatus').agg(
    average_spend = ('totalSpent', 'mean'), total_items_purchased = ('purchasedIt
```

In [37]:
```
acc_rej_summary
```

Out[37]:

	average_spend	total_items_purchased
rewardsReceiptStatus		
FINISHED	80.854305	8184.0
REJECTED	23.326056	173.0

In [38]:
```
acc_rej_summary.index = ['ACCEPTED', 'REJECTED']
```

In [39]:
```
acc_rej_summary
```

Out[39]:

	average_spend	total_items_purchased
ACCEPTED	80.854305	8184.0
REJECTED	23.326056	173.0

Thus, we can confirm that receipts accepted by the system tend to have greater spending, and the total amount of items purchased on accepted receits is also much greater.