Data Quality Findings and Checks

In [1]:
```
import pandas as pd
import json
pd.set_option('display.max_columns', None)
```

Receipt Data Set

In [2]:
```
receipt_data = pd.read_json('receipts.json', lines = True)
receipt_data.head()
```

Out[2]:

	_id	bonusPointsEarned	bonusPointsEarnedReason	createD
0	{'$oid': '5ff1e1eb0a720f0523000575'}	500.0	Receipt number 2 completed, bonus point schedu...	{'$da 16096875310
1	{'$oid': '5ff1e1bb0a720f052300056b'}	150.0	Receipt number 5 completed, bonus point schedu...	{'$da 16096874830
2	{'$oid': '5ff1e1f10a720f052300057a'}	5.0	All-receipts receipt bonus	{'$da 16096875370
3	{'$oid': '5ff1e1ee0a7214ada100056f'}	5.0	All-receipts receipt bonus	{'$da 16096875340
4	{'$oid': '5ff1e1d20a7214ada1000561'}	5.0	All-receipts receipt bonus	{'$da 16096875060

From the above, we can see we need to flatten the data further and pull out rewardsReceiptItemList as a separate dataframe. Once we get the data in readable dataframe format, we can perform quality checks further

In [3]:
```
#Remove and flatten oid variables.
def extract_oid(oid_dict):
    try:
        if isinstance(oid_dict, dict) and '$oid' in oid_dict:  # Check if i
            return oid_dict['$oid']
        elif isinstance(oid_dict, str):  # If it's already a string, return
            return oid_dict
        else:
            return None
    except (TypeError, KeyError):  # Handles other unexpected types or missi
        return None



object_id_fields = ['_id', 'userId']

for field in object_id_fields:
    receipt_data[field] = receipt_data[field].apply(extract_oid)

# Flatten and convert date columns to dateTime.
```

```
date_cols = ['createDate', 'dateScanned', 'finishedDate', 'modifyDate', 'poi
for col in date_cols:
    receipt_data[col] = pd.to_datetime(receipt_data[col].apply(lambda x: x[
receipt_data.head()
```

Out[3]:

	_id	bonusPointsEarned	bonusPointsEarnedReason	createDate	da
0	5ff1e1eb0a720f0523000575	500.0	Receipt number 2 completed, bonus point schedu...	2021-01-03 15:25:31	
1	5ff1e1bb0a720f052300056b	150.0	Receipt number 5 completed, bonus point schedu...	2021-01-03 15:24:43	
2	5ff1e1f10a720f052300057a	5.0	All-receipts receipt bonus	2021-01-03 15:25:37	
3	5ff1e1ee0a7214ada100056f	5.0	All-receipts receipt bonus	2021-01-03 15:25:34	
4	5ff1e1d20a7214ada1000561	5.0	All-receipts receipt bonus	2021-01-03 15:25:06	

We can now flatten the rewardsReceiptItemList as a separate date frame, since as per our model we would want it to be a separate table that holds the key for brand and receipt table.

Receipt Item Data Set

In [5]:
```
df_exploded = receipt_data.explode("rewardsReceiptItemList")
receipt_item_data = pd.json_normalize(df_exploded["rewardsReceiptItemList"]
receipt_item_data.head()
```

Out[5]:

	barcode	description	finalPrice	itemPrice	needsFetchReview	partnerItemId	prev
0	4011	ITEM NOT FOUND	26.00	26.00	False	1	
1	4011	ITEM NOT FOUND	1	1	NaN	1	
2	028400642255	DORITOS TORTILLA CHIP SPICY SWEET CHILI REDUCE...	10.00	10.00	True	2	
3	NaN	NaN	NaN	NaN	False	1	
4	4011	ITEM NOT FOUND	28.00	28.00	False	1	

Receipt Joined With Receipt_Item Dataset

In [6]:
```
receipt_joined_with_receipt_item = pd.concat([df_exploded.drop(columns=["rew
receipt_joined_with_receipt_item.head()
```

Out[6]:

	_id	bonusPointsEarned	bonusPointsEarnedReason	createDate	da
0	5ff1e1eb0a720f0523000575	500.0	Receipt number 2 completed, bonus point schedu...	2021-01-03 15:25:31	
1	5ff1e1bb0a720f052300056b	150.0	Receipt number 5 completed, bonus point schedu...	2021-01-03 15:24:43	
2	5ff1e1bb0a720f052300056b	150.0	Receipt number 5 completed, bonus point schedu...	2021-01-03 15:24:43	
3	5ff1e1f10a720f052300057a	5.0	All-receipts receipt bonus	2021-01-03 15:25:37	
4	5ff1e1ee0a7214ada100056f	5.0	All-receipts receipt bonus	2021-01-03 15:25:34	

In [7]:

```
receipt_joined_with_receipt_item.shape
```

Out[7]:

```
(7381, 48)
```

Performing Data Quality Checks for Receipt Data Set

In [8]:

```
receipt_data.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   datetime64[ns]
 4   dateScanned              1119 non-null   datetime64[ns]
 5   finishedDate             568 non-null    datetime64[ns]
 6   modifyDate               1119 non-null   datetime64[ns]
 7   pointsAwardedDate        537 non-null    datetime64[ns]
 8   pointsEarned             609 non-null    float64
 9   purchaseDate             671 non-null    datetime64[ns]
 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: datetime64[ns](6), float64(4), object(5)
memory usage: 131.3+ KB
```

From the above information about the receipt_data, we can infer the following:

1. Receipt Item List is missing for 440 reciepts. (Based on the assumption that every receipt, should have an item present in it)

2. Likewise, there are missing entries for other critical attributes like: total_spent, purchase_date. This will help stakeholders to understand what brands sell the most, avg spend value and other metrics that may frame future strategies.
3. While it's okay to have missing or null values for bonus points, points earned, points awarded date (as not every item maybe eligigble for rewards or bonus rewards). In addition to that, finished_date so also be not null a every created_date for the receipt needs to have finished_date (irrespective of the processing result i.e Accepted or Rejected).
4. We see all dates are in ms, however for readability to our analytics team, we can convert them into data_time format as done above. ids (receipt_id and user_id) can be string characters to keep the data consistent across all our datasets.

Summary Statistics for Receipt Data

In [9]:
```
receipt_data.describe()
```

Out[9]:

	bonusPointsEarned	pointsEarned	purchasedItemCount	totalSpent
count	544.000000	609.000000	635.00000	684.000000
mean	238.893382	585.962890	14.75748	77.796857
std	299.091731	1357.166947	61.13424	347.110349
min	5.000000	0.000000	0.00000	0.000000
25%	5.000000	5.000000	1.00000	1.000000
50%	45.000000	150.000000	2.00000	18.200000
75%	500.000000	750.000000	5.00000	34.960000
max	750.000000	10199.800000	689.00000	4721.950000

Let's ensure receipt_id column does not have duplicates, as it's necessary for us to keep unique value

In [10]:
```
duplicate_receipt_count = receipt_data['_id'].duplicated().value_counts()
print(duplicate_receipt_count)
```

```
False    1119
Name: _id, dtype: int64
```

Performing Data Quality Checks for Receipt Item

In [11]:
```
receipt_item_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7381 entries, 0 to 7380
Data columns (total 34 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   barcode                            3090 non-null   object 
 1   description                        6560 non-null   object 
 2   finalPrice                         6767 non-null   object 
 3   itemPrice                          6767 non-null   object 
 4   needsFetchReview                   813 non-null    object 
 5   partnerItemId                      6941 non-null   object 
 6   preventTargetGapPoints             358 non-null    object 
 7   quantityPurchased                  6767 non-null   float64
 8   userFlaggedBarcode                 337 non-null    object 
 9   userFlaggedNewItem                 323 non-null    object 
 10  userFlaggedPrice                   299 non-null    object 
 11  userFlaggedQuantity                299 non-null    float64
 12  needsFetchReviewReason             219 non-null    object 
 13  pointsNotAwardedReason             340 non-null    object 
 14  pointsPayerId                      1267 non-null   object 
 15  rewardsGroup                       1731 non-null   object 
 16  rewardsProductPartnerId            2269 non-null   object 
 17  userFlaggedDescription             205 non-null    object 
 18  originalMetaBriteBarcode           71 non-null     object 
 19  originalMetaBriteDescription       10 non-null     object 
 20  brandCode                          2600 non-null   object 
 21  competitorRewardsGroup             275 non-null    object 
 22  discountedItemPrice                5769 non-null   object 
 23  originalReceiptItemText            5760 non-null   object 
 24  itemNumber                         153 non-null    object 
 25  originalMetaBriteQuantityPurchased 15 non-null     float64
 26  pointsEarned                       927 non-null    object 
 27  targetPrice                        378 non-null    object 
 28  competitiveProduct                 645 non-null    object 
 29  originalFinalPrice                 9 non-null      object 
 30  originalMetaBriteItemPrice         9 non-null      object 
 31  deleted                            9 non-null      object 
 32  priceAfterCoupon                   956 non-null    object 
 33  metabriteCampaignId                863 non-null    object 
dtypes: float64(3), object(31)
memory usage: 1.9+ MB
```

From the above we can see there are a lot of data discrepency.

1. Every item that is being sold via a receipt, should have a bar_code and also brand_code. This will inturn tie up our data cleanly with the brands table.
2. Every item list should have it's own item_line_id for uniqueness. This will prevent duplication.
3. Consistent data formats for needsFetchReview -- boolean

In [12]:
```
receipt_item_data.describe()
```

Out[12]:

	quantityPurchased	userFlaggedQuantity	originalMetaBriteQuantityPurchased
count	6767.000000	299.000000	15.000000
mean	1.386139	1.872910	1.200000
std	1.204363	1.314823	0.414039
min	1.000000	1.000000	1.000000
25%	1.000000	1.000000	1.000000
50%	1.000000	1.000000	1.000000
75%	1.000000	3.000000	1.000000
max	17.000000	5.000000	2.000000

Users Data Set

In [13]:
```
user_data = pd.read_json('users.json', lines = True)
user_data.head()
```

Out[13]:

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

In [14]:
```
object_id_fields = ['_id']

for field in object_id_fields:
    user_data[field] = user_data[field].apply(extract_oid)

# Flatten and convert date columns to dateTime.
date_cols = ['createdDate', 'lastLogin']
for col in date_cols:
    user_data[col] = pd.to_datetime(user_data[col].apply(lambda x: x['$date
```

In [15]:
```
user_data.head()
```

Out[15]:

	_id	active	createdDate	lastLogin	role	signUpSou
0	5ff1e194b6a9d73a3a9f1052	True	2021-01-03 15:24:04.800	2021-01-03 15:25:37.857999872	consumer	E
1	5ff1e194b6a9d73a3a9f1052	True	2021-01-03 15:24:04.800	2021-01-03 15:25:37.857999872	consumer	E
2	5ff1e194b6a9d73a3a9f1052	True	2021-01-03 15:24:04.800	2021-01-03 15:25:37.857999872	consumer	E
3	5ff1e1eacfcf6c399c274ae6	True	2021-01-03 15:25:30.554	2021-01-03 15:25:30.596999936	consumer	E
4	5ff1e194b6a9d73a3a9f1052	True	2021-01-03 15:24:04.800	2021-01-03 15:25:37.857999872	consumer	E

Performing Data Quality Checks for Users

In [16]:

```
user_data.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    datetime64[ns]
 3   lastLogin     433 non-null    datetime64[ns]
 4   role          495 non-null    object
 5   signUpSource  447 non-null    object
 6   state         439 non-null    object
dtypes: bool(1), datetime64[ns](2), object(4)
memory usage: 23.8+ KB
```

Let's ensure _id or user_id is not duplicate as we would want to maintain a unique list of user_ids that have signed up in our database.

In [17]:

```
duplicate_user_count = user_data['_id'].duplicated().value_counts()
print(duplicate_user_count)
```

```
True     283
False    212
Name: _id, dtype: int64
```

From the above we can infer and learn that

1. There are 283 duplicate user_ids created in our system. Meaning we have people with same information or some missing information being signed up multiple times in our system. We can prevent this from having right authentication and verification. To avoid inconsistencies and biased decision making.
2. As far data formats go, we can make role, state, and signUpSource as strings.

Brand Data Set

In [18]:
```
brand_data = pd.read_json('brands.json', lines = True)
brand_data.head()
```

Out[18]:

	_id	barcode	category	categoryCode	
0	{'$oid': '601ac115be37ce2ead437551'}	511111019862	Baking	BAKING	'601ac114be3
1	{'$oid': '601c5460be37ce2ead43755f'}	511111519928	Beverages	BEVERAGES	'5332f5fbe4l
2	{'$oid': '601ac142be37ce2ead43755d'}	511111819905	Baking	BAKING	'601ac142be3
3	{'$oid': '601ac142be37ce2ead43755a'}	511111519874	Baking	BAKING	'601ac142be3
4	{'$oid': '601ac142be37ce2ead43755e'}	511111319917	Candy & Sweets	CANDY_AND_SWEETS	'5332fa12e4

In [19]:
```
### Extracting oid
for field in object_id_fields:
    brand_data[field] = brand_data[field].apply(extract_oid)
brand_data.head()
```

Out[19]:

	_id	barcode	category	categoryCode	
0	601ac115be37ce2ead437551	511111019862	Baking	BAKING	'601ac114be37
1	601c5460be37ce2ead43755f	511111519928	Beverages	BEVERAGES	'5332f5fbe4b0
2	601ac142be37ce2ead43755d	511111819905	Baking	BAKING	'601ac142be37
3	601ac142be37ce2ead43755a	511111519874	Baking	BAKING	'601ac142be37
4	601ac142be37ce2ead43755e	511111319917	Candy & Sweets	CANDY_AND_SWEETS	'5332fa12e4b0

Performing Data Quality Checks for Brands

In [20]:
```
brand_data.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
```

Let's ensure brand_id or id is unqiue to the brand data set

In [21]:
```
duplicate_brand_count = brand_data['_id'].duplicated().value_counts()
print(duplicate_brand_count)
```

```
False    1167
Name: _id, dtype: int64
```

Checking Distinct Categorical Variables

In [22]:
```
brand_data['category'].unique()
```

Out[22]:
```
array(['Baking', 'Beverages', 'Candy & Sweets', 'Condiments & Sauces',
       'Canned Goods & Soups', nan, 'Magazines', 'Breakfast & Cereal',
       'Beer Wine Spirits', 'Health & Wellness', 'Beauty', 'Baby',
       'Frozen', 'Grocery', 'Snacks', 'Household', 'Personal Care',
       'Dairy', 'Cleaning & Home Improvement', 'Deli',
       'Beauty & Personal Care', 'Bread & Bakery', 'Outdoor',
       'Dairy & Refrigerated'], dtype=object)
```

In [23]:
```
brand_data['categoryCode'].unique()
```

Out[23]:
```
array(['BAKING', 'BEVERAGES', 'CANDY_AND_SWEETS', nan,
       'HEALTHY_AND_WELLNESS', 'GROCERY', 'PERSONAL_CARE',
       'CLEANING_AND_HOME_IMPROVEMENT', 'BEER_WINE_SPIRITS', 'BABY',
       'BREAD_AND_BAKERY', 'OUTDOOR', 'DAIRY_AND_REFRIGERATED',
       'MAGAZINES', 'FROZEN'], dtype=object)
```

Learnings from the above:

1. topBrand can be a boolean instead of float
2. brandCode is null for some brands, however there are names present for all. This should be in sync and there can be data cleaning that can take place here.
3. name, category, categoryCode, brandCode -- can be string.
4. cpg is another data set that can be extracted separately. (Refer to the ER & Data Model). However to keep it consice, I do not intend to expand and find quality issues in that.

Conclusion:

There are multiple data cleaning steps required to be performed and executed after we have explored the data and understood the business goal for it.

1. If there are few rows with null values, we can enitrely drop them -- However this might not be useful for a criticial data set like ours which might lead to biased decision making.
2. We can perform data imputation (replacing null or empty values with mean, median (incase of numerical data), mode (can be applied for numerical as well as categorical data). They can allow us to maintain the structure of our data model
3. We can also drop columns that are not very useful or may not answer the business questions directly.
4. Remove outliers that might skew the data, this can be done via box-plot
5. We can also keep data types consistent, as there are inconsistencies detected.
6. We can also ensure input sanitization is carried out, to keep our user_input data clean and consistent
7. Last but not the least as a part of our data exploratory analysis, we can also plot histograms to bucket our categorical variables and see various trends for total spent for brands, items and receipt status to name a few