First: Review Existing Unstructured Data and Diagram a New Structured Relational Data Model

Below are the libraries/packages that are going to be used throughout the assignment

```python
import json
import gzip
import shutil
import os
import pandas as pd
from tempfile import mkstemp
from shutil import move
from os import fdopen, remove
from ast import literal_eval
from datetime import datetime
from sqlalchemy import create_engine
pd.set_option("display.max_columns", None)
```

Since the json files were provided as zipped files or with .gz extensions that cannot be directly read by regular read_json method of pandas. To solve this issue, we first make use of gzip library to open the zipped files and check for these in our working directory using 'os' package. Once we detect them, we replace the extension of the unzipped file by reading and writing it back as .json extension which can be further read using regular read_json method.

```python
for files in os.listdir():
    if 'json' in files:
        with gzip.open(files, 'rb') as file_input:
            with open(files.replace('.gz', ''), 'wb') as file_output:
                shutil.copyfileobj(file_input, file_output)
```

Below all the json files are converted into a dataframe by using read_json method of pandas

```python
receipts_data = pd.read_json('receipts.json', lines=True)
brands_data = pd.read_json('brands.json',lines=True)
users_data = pd.read_json('users.json',lines=True)
```

```python
#receipts data exploration
receipts_data.head()
```

	_id	bonusPointsEarned	bonusPointsEarnedReason	createDate	dateScanned	finishedDate	modifyDate
0	{'$oid': '5ff1e1eb0a720f0523000575'}	500.0	Receipt number 2 completed, bonus point schedu...	{'$date': 1609687531000}	{'$date': 1609687531000}	{'$date': 1609687531000}	{'$date': 1609687536000}
1	{'$oid': '5ff1e1bb0a720f052300056b'}	150.0	Receipt number 5 completed, bonus point schedu...	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687488000}
2	{'$oid': '5ff1e1f10a720f052300057a'}	5.0	All-receipts receipt bonus	{'$date': 1609687537000}	{'$date': 1609687537000}	NaN	{'$date': 1609687542000}
3	{'$oid': '5ff1e1ee0a7214ada100056f'}	5.0	All-receipts receipt bonus	{'$date': 1609687534000}	{'$date': 1609687534000}	{'$date': 1609687534000}	{'$date': 1609687539000}
4	{'$oid': '5ff1e1d20a7214ada1000561'}	5.0	All-receipts receipt bonus	{'$date': 1609687506000}	{'$date': 1609687506000}	{'$date': 1609687511000}	{'$date': 1609687511000}

Below is the sample record of receipt item list before exploding it where we can notice how nested json is filled up in that column.

```python
receipts_data['rewardsReceiptItemList'][1]
```

```
[{'barcode': '4011',
  'description': 'ITEM NOT FOUND',
  'finalPrice': '1',
  'itemPrice': '1',
  'partnerItemId': '1',
  'quantityPurchased': 1},
 {'barcode': '028400642255',
  'description': 'DORITOS TORTILLA CHIP SPICY SWEET CHILI REDUCED FAT BAG 1 OZ',
  'finalPrice': '10.00',
  'itemPrice': '10.00',
  'needsFetchReview': True,
  'needsFetchReviewReason': 'USER_FLAGGED',
  'partnerItemId': '2',
  'pointsNotAwardedReason': 'Action not allowed for user and CPG',
  'pointsPayerId': '5332f5fbe4b03c9a25efd0ba',
  'preventTargetGapPoints': True,
  'quantityPurchased': 1,
  'rewardsGroup': 'DORITOS SPICY SWEET CHILI SINGLE SERVE',
  'rewardsProductPartnerId': '5332f5fbe4b03c9a25efd0ba',
  'userFlaggedBarcode': '028400642255',
  'userFlaggedDescription': 'DORITOS TORTILLA CHIP SPICY SWEET CHILI REDUCED FAT BAG 1 OZ',
  'userFlaggedNewItem': True,
  'userFlaggedPrice': '10.00',
  'userFlaggedQuantity': 1}]
```

After analysing the data, we can see that there are columns that have multiple keys and values which needs to be separated into different identifiable columns to make them as a structured data for further analysis. Pandas DataFrame has an inbuilt method as dataframe.explode('column_name') which explodes list-likes

```python
receipts_data = receipts_data.explode('rewardsReceiptItemList')
receipts_data = receipts_data.reset_index(drop=True)
receipts_data.head()
```

	_id	bonusPointsEarned	bonusPointsEarnedReason	createDate	dateScanned	finishedDate	modifyDate
0	{'$oid': '5ff1e1eb0a720f0523000575'}	500.0	Receipt number 2 completed, bonus point schedu...	{'$date': 1609687531000}	{'$date': 1609687531000}	{'$date': 1609687531000}	{'$date': 1609687536000}
1	{'$oid': '5ff1e1bb0a720f052300056b'}	150.0	Receipt number 5 completed, bonus point schedu...	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687488000}
2	{'$oid': '5ff1e1bb0a720f052300056b'}	150.0	Receipt number 5 completed, bonus point schedu...	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687488000}
3	{'$oid': '5ff1e1f10a720f052300057a'}	5.0	All-receipts receipt bonus	{'$date': 1609687537000}	{'$date': 1609687537000}	NaN	{'$date': 1609687542000}
4	{'$oid': '5ff1e1ee0a7214ada100056f'}	5.0	All-receipts receipt bonus	{'$date': 1609687534000}	{'$date': 1609687534000}	{'$date': 1609687534000}	{'$date': 1609687539000}

Below is the overview of the receipts_data which indicates the number of total enteries as 64241 for 15 columns - also describes the data type, non-null value count

```python
receipts_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7381 entries, 0 to 7380
Data columns (total 15 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   _id                      7381 non-null   object
 1   bonusPointsEarned        5980 non-null   float64
 2   bonusPointsEarnedReason  5980 non-null   object
 3   createDate               7381 non-null   object
 4   dateScanned              7381 non-null   object
 5   finishedDate             5970 non-null   object
 6   modifyDate               7381 non-null   object
 7   pointsAwardedDate        6080 non-null   object
 8   pointsEarned             6253 non-null   float64
 9   purchaseDate             6923 non-null   object
 10  purchasedItemCount       6897 non-null   float64
 11  rewardsReceiptItemList   6941 non-null   object
 12  rewardsReceiptStatus     7381 non-null   object
 13  totalSpent               6946 non-null   float64
 14  userId                   7381 non-null   object
dtypes: float64(4), object(11)
memory usage: 865.1+ KB
```

Below is the sample data for the same index after exploding that column which basically separates the data into multiple column and rows.

```python
sample_data_item_list = receipts_data.get("rewardsReceiptItemList")
sample_data_item_list[1]
```

```
{'barcode': '4011',
 'description': 'ITEM NOT FOUND',
 'finalPrice': '1',
 'itemPrice': '1',
 'partnerItemId': '1',
 'quantityPurchased': 1}
```

Since we know the importance of analysing the number of missing values, Nan values, NA values and their datatypes, I wrote a generic function to build a table to calulate the information for any dataframe passed.

In the first 2 lines we get the shape of the dataframe which is 49 columns and 13,643 rows and apparently, every column has missing values. Either way, we can handle them using different methods or drop them. Technically they cannot be counted as outliers just because the percentage is beyond the normal threshold and investigating them on what could have caused it could be determined by the way json_normalise and explode() method works. Having more number of missing values doesnt necessarily means data is bad.

```python
def missing_zero_values_table(df):
        zero_val = (df == 0.00).astype(int).sum(axis=0)
        mis_val = df.isnull().sum()
        mis_val_percent = 100 * df.isnull().sum() / len(df)
        mz_table = pd.concat([zero_val, mis_val, mis_val_percent], axis=1)
        mz_table = mz_table.rename(
        columns = {0 : 'Zero Values', 1 : 'Missing Values', 2 : '% of Total Values'})
        mz_table['Total Zero Missing Values'] = mz_table['Zero Values'] + mz_table['Missing Values']
        mz_table['% Total Zero Missing Values'] = 100 * mz_table['Total Zero Missing Values'] / len(df)
        mz_table['Data Type'] = df.dtypes
        mz_table = mz_table[
            mz_table.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns and " + str(df.shape[0]) + " Rows.
            "There are " + str(mz_table.shape[0]) +
              " columns that have missing values.")
        return mz_table
```

```python
missing_zero_values_table(receipts_final)
```

```
Your selected dataframe has 48 columns and 227 Rows.
There are 31 columns that have missing values.
```

	Zero Values	Missing Values	% of Total Values	Total Zero Missing Values	% Total Zero Missing Values	Data Type
rewardsReceiptItemList_userFlaggedDescription	0	227	100.0	227	100.0	object
rewardsReceiptItemList_originalMetaBriteBarcode	0	227	100.0	227	100.0	object
rewardsReceiptItemList_targetPrice	0	227	100.0	227	100.0	object
rewardsReceiptItemList_deleted	0	227	100.0	227	100.0	object
rewardsReceiptItemList_needsFetchReview	0	227	100.0	227	100.0	object
rewardsReceiptItemList_preventTargetGapPoints	0	227	100.0	227	100.0	object
rewardsReceiptItemList_userFlaggedBarcode	0	227	100.0	227	100.0	object
rewardsReceiptItemList_userFlaggedNewItem	0	227	100.0	227	100.0	object
rewardsReceiptItemList_userFlaggedPrice	0	227	100.0	227	100.0	object
rewardsReceiptItemList_userFlaggedQuantity	0	227	100.0	227	100.0	float64
rewardsReceiptItemList_needsFetchReviewReason	0	227	100.0	227	100.0	object
rewardsReceiptItemList_originalMetaBriteQuantityPurchased	0	227	100.0	227	100.0	float64
rewardsReceiptItemList_originalMetaBriteDescription	0	227	100.0	227	100.0	object
rewardsReceiptItemList_originalMetaBriteItemPrice	0	227	100.0	227	100.0	object
rewardsReceiptItemList_originalFinalPrice	0	227	100.0	227	100.0	object
rewardsReceiptItemList_itemNumber	0	225	99.1	225	99.1	object
rewardsReceiptItemList_priceAfterCoupon	0	223	98.2	223	98.2	object
rewardsReceiptItemList_pointsNotAwardedReason	0	209	92.1	209	92.1	object
rewardsReceiptItemList_pointsEarned	0	203	89.4	203	89.4	object
rewardsReceiptItemList_competitorRewardsGroup	0	192	84.6	192	84.6	object
rewardsReceiptItemList_competitiveProduct	12	187	82.4	199	87.7	object
rewardsReceiptItemList_pointsPayerId	0	185	81.5	185	81.5	object
rewardsReceiptItemList_metabriteCampaignId	0	181	79.7	181	79.7	object
rewardsReceiptItemList_rewardsGroup	0	163	71.8	163	71.8	object
rewardsReceiptItemList_rewardsProductPartnerId	0	117	51.5	117	51.5	object
rewardsReceiptItemList_barcode	0	89	39.2	89	39.2	object
bonusPointsEarnedReason	0	29	12.8	29	12.8	object
bonusPointsEarned	0	29	12.8	29	12.8	float64
pointsEarned	1	26	11.5	27	11.9	float64
rewardsReceiptItemList_originalReceiptItemText	0	3	1.3	3	1.3	object
rewardsReceiptItemList_discountedItemPrice	0	1	0.4	1	0.4	object

As we can see that there are NA values in a lot of columns but currently we are looking into rewardsReceiptItemList and all the dates column. We can notice that the percentage of NA values in these columns are way high(around 50%) so we cannot drop them directly. There are different ways that NA/missingvalues can be handled. Below is one of the ways we are going to handle the date columns

```python
#all the date columns that have missing records
req_list = ['createDate', 'dateScanned','finishedDate','modifyDate','pointsAwardedDate','purchaseDate']

#for loop over the above list
for i in req_list:
    #calculating the counts of different values in a column and storing it in a variable
    res = receipts_data[i].value_counts()
    #the result is a dictionary where we need to grab the keys and feed the first one- the default is in ascenc
    #we have to handle cases where NaN values can overpower the regular non-null values
    result = res.keys()[0] if res.keys()[0] != 'NaN' else res.keys()[1]
    #filling NA values with the top results which is the MAX count(top value based on counts)
    receipts_data = receipts_data.fillna({i:f'{result}'})
    receipts_data[i] = receipts_data[i].apply(lambda x:str(x))
    receipts_data[i] = receipts_data[i].apply(literal_eval)
receipts_data = receipts_data.reset_index(drop=True)
receipts_data.head()
```

	_id	bonusPointsEarned	bonusPointsEarnedReason	createDate	dateScanned	finishedDate	modifyDate
0	{'$oid': '5ff1e1eb0a720f0523000575'}	500.0	Receipt number 2 completed, bonus point schedu...	{'$date': 1609687531000}	{'$date': 1609687531000}	{'$date': 1609687531000}	{'$date': 1609687536000}
1	{'$oid': '5ff1e1bb0a720f052300056b'}	150.0	Receipt number 5 completed, bonus point schedu...	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687488000}
2	{'$oid': '5ff1e1bb0a720f052300056b'}	150.0	Receipt number 5 completed, bonus point schedu...	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687488000}
3	{'$oid': '5ff1e1f10a720f052300057a'}	5.0	All-receipts receipt bonus	{'$date': 1609687537000}	{'$date': 1609687537000}	{'$date': 1611612263000}	{'$date': 1609687542000}
4	{'$oid': '5ff1e1ee0a7214ada100056f'}	5.0	All-receipts receipt bonus	{'$date': 1609687534000}	{'$date': 1609687534000}	{'$date': 1609687534000}	{'$date': 1609687539000}

Here for rewardsReceiptItemList column, we cannot use the max count value strategy because we wont be able to actually decide the items in that list and substituing them with the top value isnt the best way to handle it as it can mess up the rewards which can in turn cause a loss to the company when we are not sure whether they actually bought anything or not.

```python
receipts_data = receipts_data.fillna({'rewardsReceiptItemList':'{}'})
receipts_data['rewardsReceiptItemList'] = receipts_data['rewardsReceiptItemList'].apply(lambda x:str(x))
receipts_data['rewardsReceiptItemList'] = receipts_data['rewardsReceiptItemList'].apply(literal_eval)
receipts_data = receipts_data.reset_index(drop=True)
receipts_data.head()
```

	_id	bonusPointsEarned	bonusPointsEarnedReason	createDate	dateScanned	finishedDate	modifyDate
0	{'$oid': '5ff1e1eb0a720f0523000575'}	500.0	Receipt number 2 completed, bonus point schedu...	{'$date': 1609687531000}	{'$date': 1609687531000}	{'$date': 1609687531000}	{'$date': 1609687536000}
1	{'$oid': '5ff1e1bb0a720f052300056b'}	150.0	Receipt number 5 completed, bonus point schedu...	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687488000}
2	{'$oid': '5ff1e1bb0a720f052300056b'}	150.0	Receipt number 5 completed, bonus point schedu...	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687488000}
3	{'$oid': '5ff1e1f10a720f052300057a'}	5.0	All-receipts receipt bonus	{'$date': 1609687537000}	{'$date': 1609687537000}	{'$date': 1611612263000}	{'$date': 1609687542000}
4	{'$oid': '5ff1e1ee0a7214ada100056f'}	5.0	All-receipts receipt bonus	{'$date': 1609687534000}	{'$date': 1609687534000}	{'$date': 1609687534000}	{'$date': 1609687539000}

The data is in an unstructured format which cannot be parsed directly nor we can perform any operations so to flatten the nested json structure, we make use of json_normalize method which separates all the data in that one column,row into different columns with the prefix of its parent column.

```python
receipts_norm = pd.json_normalize(receipts_data['rewardsReceiptItemList'],errors='ignore',record_prefix='rewarc
.add_prefix('rewardsReceiptItemList.')
```

```python
receipts_final = pd.merge(receipts_data, receipts_norm, left_index=True, right_index=True, how='outer')
receipts_final.head()
```

	_id	bonusPointsEarned	bonusPointsEarnedReason	createDate	dateScanned	finishedDate	modifyDate
0	{'$oid': '5ff1e1eb0a720f0523000575'}	500.0	Receipt number 2 completed, bonus point schedu...	{'$date': 1609687531000}	{'$date': 1609687531000}	{'$date': 1609687531000}	{'$date': 1609687536000}
1	{'$oid': '5ff1e1bb0a720f052300056b'}	150.0	Receipt number 5 completed, bonus point schedu...	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687488000}
2	{'$oid': '5ff1e1bb0a720f052300056b'}	150.0	Receipt number 5 completed, bonus point schedu...	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687483000}	{'$date': 1609687488000}
3	{'$oid': '5ff1e1f10a720f052300057a'}	5.0	All-receipts receipt bonus	{'$date': 1609687537000}	{'$date': 1609687537000}	{'$date': 1611612263000}	{'$date': 1609687542000}
4	{'$oid': '5ff1e1ee0a7214ada100056f'}	5.0	All-receipts receipt bonus	{'$date': 1609687534000}	{'$date': 1609687534000}	{'$date': 1609687534000}	{'$date': 1609687539000}

```python
receipts_final['rewardsReceiptStatus'].value_counts()
```

```
FINISHED     5920
FLAGGED       810
SUBMITTED     434
REJECTED      167
PENDING        50
Name: rewardsReceiptStatus, dtype: int64
```

```python
receipts_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7381 entries, 0 to 7380
Data columns (total 49 columns):
 #   Column                                                Non-Null Count  Dtype
---  ------                                                --------------  -----
 0   _id                                                   7381 non-null   object
 1   bonusPointsEarned                                     5980 non-null   float64
 2   bonusPointsEarnedReason                               5980 non-null   object
 3   createDate                                            7381 non-null   object
 4   dateScanned                                           7381 non-null   object
 5   finishedDate                                          7381 non-null   object
 6   modifyDate                                            7381 non-null   object
 7   pointsAwardedDate                                     7381 non-null   object
 8   pointsEarned                                          6253 non-null   float64
 9   purchaseDate                                          7381 non-null   object
 10  purchasedItemCount                                    6897 non-null   float64
 11  rewardsReceiptItemList                                7381 non-null   object
 12  rewardsReceiptStatus                                  7381 non-null   object
 13  totalSpent                                            6946 non-null   float64
 14  userId                                                7381 non-null   object
 15  rewardsReceiptItemList.barcode                        3090 non-null   object
 16  rewardsReceiptItemList.description                    6560 non-null   object
 17  rewardsReceiptItemList.finalPrice                     6767 non-null   object
 18  rewardsReceiptItemList.itemPrice                      6767 non-null   object
 19  rewardsReceiptItemList.needsFetchReview               813 non-null    object
 20  rewardsReceiptItemList.partnerItemId                  6941 non-null   object
 21  rewardsReceiptItemList.preventTargetGapPoints         358 non-null    object
 22  rewardsReceiptItemList.quantityPurchased              6767 non-null   float64
 23  rewardsReceiptItemList.userFlaggedBarcode             337 non-null    object
 24  rewardsReceiptItemList.userFlaggedNewItem             323 non-null    object
 25  rewardsReceiptItemList.userFlaggedPrice               299 non-null    object
 26  rewardsReceiptItemList.userFlaggedQuantity            299 non-null    float64
 27  rewardsReceiptItemList.needsFetchReviewReason         219 non-null    object
 28  rewardsReceiptItemList.pointsNotAwardedReason         340 non-null    object
 29  rewardsReceiptItemList.pointsPayerId                  1267 non-null   object
 30  rewardsReceiptItemList.rewardsGroup                   1731 non-null   object
 31  rewardsReceiptItemList.rewardsProductPartnerId        2269 non-null   object
 32  rewardsReceiptItemList.userFlaggedDescription         205 non-null    object
 33  rewardsReceiptItemList.originalMetaBriteBarcode       71 non-null     object
 34  rewardsReceiptItemList.originalMetaBriteDescription   10 non-null     object
 35  rewardsReceiptItemList.brandCode                      2600 non-null   object
 36  rewardsReceiptItemList.competitorRewardsGroup         275 non-null    object
 37  rewardsReceiptItemList.discountedItemPrice            5769 non-null   object
 38  rewardsReceiptItemList.originalReceiptItemText        5760 non-null   object
 39  rewardsReceiptItemList.itemNumber                     153 non-null    object
 40  rewardsReceiptItemList.originalMetaBriteQuantityPurchased  15 non-null  float64
 41  rewardsReceiptItemList.pointsEarned                   927 non-null    object
 42  rewardsReceiptItemList.targetPrice                    378 non-null    object
 43  rewardsReceiptItemList.competitiveProduct             645 non-null    object
 44  rewardsReceiptItemList.originalFinalPrice             9 non-null      object
 45  rewardsReceiptItemList.originalMetaBriteItemPrice      9 non-null      object
 46  rewardsReceiptItemList.deleted                        9 non-null      object
 47  rewardsReceiptItemList.priceAfterCoupon               956 non-null    object
 48  rewardsReceiptItemList.metabriteCampaignId            863 non-null    object
dtypes: float64(7), object(42)
memory usage: 2.8+ MB
```

Now all the columns have 1 dictionary key:value pair, we will extract them using set funtion.

```python
required_list = ['_id','createDate','dateScanned','finishedDate','modifyDate','pointsAwardedDate','purchaseDate

for req_items in required_list:
    set().union(*(k.keys() for k in receipts_final[req_items]))
```

```
{'$oid'}
{'$date'}
{'$date'}
{'$date'}
{'$date'}
{'$date'}
{'$date'}
```

The dates provided to us in the data are in UTC format which is hard to follow so we will convert it into datetime using datetime library with utcfromtimestamp() method

```python
def date_converter(x):
  try:
    return(datetime.utcfromtimestamp(int(x['$date'])/1000).strftime('%Y-%m-%d %H:%M:%S'))
  except TypeError:
    return(None)
```

```python
receipts_final['_id'] = receipts_final['_id'].apply(lambda x: x['$oid'])
receipts_final['createDate'] = receipts_final['createDate'].apply(lambda x: date_converter(x))
receipts_final['dateScanned'] = receipts_final['dateScanned'].apply(lambda x: date_converter(x))
receipts_final['finishedDate'] = receipts_final['finishedDate'].apply(lambda x: date_converter(x))
receipts_final['modifyDate'] = receipts_final['modifyDate'].apply(lambda x: date_converter(x))
receipts_final['pointsAwardedDate'] = receipts_final['pointsAwardedDate'].apply(lambda x: date_converter(x))
receipts_final['purchaseDate'] = receipts_final['purchaseDate'].apply(lambda x: date_converter(x))
```

```python
receipts_final = receipts_final.reset_index(drop=True)
receipts_final.head()
```

	_id	bonusPointsEarned	bonusPointsEarnedReason	createDate	dateScanned	finishedDate	modifyDate	pointsAwarded
0	5ff1e1eb0a720f0523000575	500.0	Receipt number 2 completed, bonus point schedu...	2021-01-03 15:25:31	2021-01-03 15:25:31	2021-01-03 15:25:31	2021-01-03 15:25:36	2021-01-03 15:
1	5ff1e1bb0a720f052300056b	150.0	Receipt number 5 completed, bonus point schedu...	2021-01-03 15:24:43	2021-01-03 15:24:43	2021-01-03 15:24:43	2021-01-03 15:24:48	2021-01-03 15:
2	5ff1e1bb0a720f052300056b	150.0	Receipt number 5 completed, bonus point schedu...	2021-01-03 15:24:43	2021-01-03 15:24:43	2021-01-03 15:24:43	2021-01-03 15:24:48	2021-01-03 15:
3	5ff1e1f10a720f052300057a	5.0	All-receipts receipt bonus	2021-01-03 15:25:37	2021-01-03 15:25:37	2021-01-25 22:04:23	2021-01-03 15:25:42	2021-01-25 22:
4	5ff1e1ee0a7214ada100056f	5.0	All-receipts receipt bonus	2021-01-03 15:25:34	2021-01-03 15:25:34	2021-01-03 15:25:34	2021-01-03 15:25:39	2021-01-03 15:

To check which columns are unique and which are not, this small for loop conveys us the way these tables would be connected to each other.

```python
receipts_final = receipts_final.drop(columns=['rewardsReceiptItemList'], axis=1)
for i in receipts_final:
    print(f"{i} - {receipts_final[i].is_unique}")
```

```
_id - False
bonusPointsEarned - False
bonusPointsEarnedReason - False
createDate - False
dateScanned - False
finishedDate - False
modifyDate - False
pointsAwardedDate - False
pointsEarned - False
purchaseDate - False
purchasedItemCount - False
rewardsReceiptStatus - False
totalSpent - False
userId - False
rewardsReceiptItemList.barcode - False
rewardsReceiptItemList.description - False
rewardsReceiptItemList.finalPrice - False
rewardsReceiptItemList.itemPrice - False
rewardsReceiptItemList.needsFetchReview - False
rewardsReceiptItemList.partnerItemId - False
rewardsReceiptItemList.preventTargetGapPoints - False
rewardsReceiptItemList.quantityPurchased - False
rewardsReceiptItemList.userFlaggedBarcode - False
rewardsReceiptItemList.userFlaggedNewItem - False
rewardsReceiptItemList.userFlaggedPrice - False
rewardsReceiptItemList.userFlaggedQuantity - False
rewardsReceiptItemList.needsFetchReviewReason - False
rewardsReceiptItemList.pointsNotAwardedReason - False
rewardsReceiptItemList.pointsPayerId - False
rewardsReceiptItemList.rewardsGroup - False
rewardsReceiptItemList.rewardsProductPartnerId - False
rewardsReceiptItemList.userFlaggedDescription - False
rewardsReceiptItemList.originalMetaBriteBarcode - False
rewardsReceiptItemList.originalMetaBriteDescription - False
rewardsReceiptItemList.brandCode - False
rewardsReceiptItemList.competitorRewardsGroup - False
rewardsReceiptItemList.discountedItemPrice - False
rewardsReceiptItemList.originalReceiptItemText - False
rewardsReceiptItemList.itemNumber - False
rewardsReceiptItemList.originalMetaBriteQuantityPurchased - False
rewardsReceiptItemList.pointsEarned - False
rewardsReceiptItemList.targetPrice - False
rewardsReceiptItemList.competitiveProduct - False
rewardsReceiptItemList.originalFinalPrice - False
rewardsReceiptItemList.originalMetaBriteItemPrice - False
rewardsReceiptItemList.deleted - False
rewardsReceiptItemList.priceAfterCoupon - False
rewardsReceiptItemList.metabriteCampaignId - False
```

Below we will explore users dataset

```python
users_data.head()
```

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

The same function can be used to explore any dataset as we are here doing for users data. It has 7 columns in total with only 3 columns that need to be handled.

```python
missing_zero_values_table(users_data)
```

```
Your selected dataframe has 7 columns and 495 Rows.
There are 3 columns that have missing values.
```

	Zero Values	Missing Values	% of Total Values	Total Zero Missing Values	% Total Zero Missing Values	Data Type
lastLogin	0	62	12.5	62	12.5	object
state	0	56	11.3	56	11.3	object
signUpSource	0	48	9.7	48	9.7	object

Here we are dropping the NA values to clean the dataset and keep it consistent throughout. Once we drop the NA values, we will be left with one entry of key:value pair which can be extracted using set, union and keys(). The total NA values in every column consist of only 12% on average which we can ideally drop and wont affect the data overall.

```python
req_users_list = ['_id','createdDate','lastLogin']
users_data = users_data.dropna(subset=req_users_list)
for req_items in req_users_list:
    set().union(*(k.keys() for k in users_data[req_items]))
```

```
{'$oid'}
{'$date'}
{'$date'}
```

applying the date convertor function to every date column

```python
users_data['_id'] = users_data['_id'].apply(lambda x: x['$oid'])
users_data['createdDate'] = users_data['createdDate'].apply(lambda x: date_converter(x))
users_data['lastLogin'] = users_data['lastLogin'].apply(lambda x: date_converter(x))
users_final = users_data.drop_duplicates()
users_final = users_final.reset_index(drop=True)
users_final.head()
```

	_id	active	createdDate	lastLogin	role	signUpSource	state
0	5ff1e194b6a9d73a3a9f1052	True	2021-01-03 15:24:04	2021-01-03 15:25:37	consumer	Email	WI
1	5ff1e1eacfcf6c399c274ae6	True	2021-01-03 15:25:30	2021-01-03 15:25:30	consumer	Email	WI
2	5ff1e1e8cfcf6c399c274ad9	True	2021-01-03 15:25:28	2021-01-03 15:25:28	consumer	Email	WI
3	5ff1e1b7cfcf6c399c274a5a	True	2021-01-03 15:24:39	2021-01-03 15:24:39	consumer	Email	WI
4	5ff1e1f1cfcf6c399c274b0b	True	2021-01-03 15:25:37	2021-01-03 15:25:37	consumer	Email	WI

The below code is used to find out which column has unique values and which doesnt

```python
for i in users_final:
    print(f"{i} - {users_final[i].is_unique}")
```

```
_id - True
active - False
createdDate - True
lastLogin - False
role - False
signUpSource - False
state - False
```

Exploring brands dataset. We can see that this also has unstructured data which will need to be flattened out using json_normalize

```python
brands_data.head()
```

	_id	barcode	category	categoryCode	cpg	name	topBrand
0	{'$oid': '601ac115be37ce2ead437551'}	511111019862	Baking	BAKING	{'$id': {'$oid': '601ac114be37ce2ead437550'}, ...	test brand @1612366101024	0.0
1	{'$oid': '601c5460be37ce2ead43755f'}	511111519928	Beverages	BEVERAGES	{'$id': {'$oid': '5332f5fbe4b03c9a25efd0ba'}, ...	Starbucks	0.0
2	{'$oid': '601ac142be37ce2ead43755d'}	511111819905	Baking	BAKING	{'$id': {'$oid': '601ac142be37ce2ead437559'}, ...	test brand @1612366146176	0.0
3	{'$oid': '601ac142be37ce2ead43755a'}	511111519874	Baking	BAKING	{'$id': {'$oid': '601ac142be37ce2ead437559'}, ...	test brand @1612366146051	0.0
4	{'$oid': '601ac142be37ce2ead43755e'}	511111319917	Candy & Sweets	CANDY_AND_SWEETS	{'$id': {'$oid': '5332fa12e4b03c9a25efd1e7'}, ...	test brand @1612366146827	0.0

Exploring the brandCode column which is what we are interested in. We can notice that the value of null/empty has the most counts. We can also notice a trend of brandnames which starts as "TEST BRANDCODE" with some random number(maybe system generated codes for test products).

```python
brands_data['brandCode'].value_counts()[0:10]
```

```
                                  35
GOODNITES                          2
HUGGIES                            2
ROYAL DANSK                        1
SOL                                1
TEST BRANDCODE @1599159969725      1
WYLER'S                            1
TEST BRANDCODE @1601939762943      1
FAT RABBIT                         1
TOTAL                              1
Name: brandCode, dtype: int64
```

To handle the test brandcodes, I have replaced them into one common as test brandcode and grouped to differentiate among other valid brandcodes.

```python
brands_data.brandCode = brands_data.brandCode.apply(lambda x: 'TEST BRANDCODE' if 'TEST BRANDCODE' in str(x) el
```

```python
brands_data['brandCode'].value_counts()[0:10]
```

```
TEST BRANDCODE         357
                        35
GOODNITES                2
HUGGIES                  2
TOTAL                    1
MALIBU© STRAWBERRY       1
REDBREAST 12YO           1
SOL                      1
WYLER'S                  1
FAT RABBIT               1
Name: brandCode, dtype: int64
```

```python
missing_zero_values_table(brands_data)
```

```
Your selected dataframe has 8 columns and 1167 Rows.
There are 4 columns that have missing values.
```

	Zero Values	Missing Values	% of Total Values	Total Zero Missing Values	% Total Zero Missing Values	Data Type
categoryCode	0	650	55.7	650	55.7	object
topBrand	524	612	52.4	1136	97.3	float64
brandCode	0	234	20.1	234	20.1	object
category	0	155	13.3	155	13.3	object

Extracting the keys from the cpg (consumer packaged goods) rows

```python
set().union(*(k.keys() for k in brands_data['cpg']))
```

```
{'$id', '$ref'}
```

Normalising the nested json data in cpg

```python
brands_norm = pd.json_normalize(brands_data['cpg'])
brands_norm = brands_norm.add_prefix('cpg.')
```

```python
brands_final = pd.merge(brands_data, brands_norm, left_index=True, right_index=True, how='outer')
```

NOTE- empty values cannot be dropped directly using dropna method because of the way it treats NA and empty values separately. So we will convert the empty data to NaN and then dropna.

```python
nan_value = float("NaN")
#Convert NaN values to empty string
brands_final = brands_final.replace("", nan_value)
brands_final = brands_final.dropna(subset=['brandCode'])
```

```python
brands_final['brandCode'].value_counts()[0:10]
```

```
TEST BRANDCODE        357
GOODNITES               2
HUGGIES                 2
MALIBU© PINEAPPLE       1
REDBREAST 12YO          1
SOL                     1
WYLER'S                 1
FAT RABBIT              1
TOTAL                   1
PURE BLISS              1
Name: brandCode, dtype: int64
```

```python
brands_final['_id'] = brands_final['_id'].apply(lambda x: x['$oid'])
```

Exploding "cpg" column to extract rest of the data

```python
brands_final = brands_final.explode('cpg')
brands_final = brands_final.reset_index(drop=True)
brands_final.head()
```

	_id	barcode	category	categoryCode	cpg	name	topBrand	brandCode	cpg.$ref	
0	601c5460be37ce2ead43755f	511111519928	Beverages	BEVERAGES	$id	Starbucks	0.0	STARBUCKS	Cogs	5332f5fbe4b03c
1	601c5460be37ce2ead43755f	511111519928	Beverages	BEVERAGES	$ref	Starbucks	0.0	STARBUCKS	Cogs	5332f5fbe4b03c
2	601ac142be37ce2ead43755d	511111819905	Baking	BAKING	$id	test brand @1612366146176	0.0	TEST BRANDCODE	Cogs	601ac142be37ce
3	601ac142be37ce2ead43755d	511111819905	Baking	BAKING	$ref	test brand @1612366146176	0.0	TEST BRANDCODE	Cogs	601ac142be37ce
4	601ac142be37ce2ead43755a	511111519874	Baking	BAKING	$id	test brand @1612366146051	0.0	TEST BRANDCODE	Cogs	601ac142be37ce

after exploding and normalising that column, its important to drop it so the whole data stays sturctured

```python
brands_final = brands_final.drop(columns=['cpg'],axis=1)
brands_final = brands_final.reset_index(drop=True)
```

```python
for i in brands_final:
    print(f"{i} - {brands_final[i].is_unique}")
```

```
_id - False
barcode - False
category - False
categoryCode - False
name - False
topBrand - False
brandCode - False
cpg.$ref - False
cpg.$id.$oid - False
```

```python
brands_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1796 entries, 0 to 1795
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   _id           1796 non-null   object
 1   barcode       1796 non-null   int64
 2   category      1788 non-null   object
 3   categoryCode  888 non-null    object
 4   name          1796 non-null   object
 5   topBrand      970 non-null    float64
 6   brandCode     1796 non-null   object
 7   cpg.$ref      1796 non-null   object
 8   cpg.$id.$oid  1796 non-null   object
dtypes: float64(1), int64(1), object(7)
memory usage: 126.4+ KB
```

```python
brands_final.head()
```

	_id	barcode	category	categoryCode	name	topBrand	brandCode	cpg.$ref	cpg.
0	601c5460be37ce2ead43755f	511111519928	Beverages	BEVERAGES	Starbucks	0.0	STARBUCKS	Cogs	5332f5fbe4b03c9a25e
1	601c5460be37ce2ead43755f	511111519928	Beverages	BEVERAGES	Starbucks	0.0	STARBUCKS	Cogs	5332f5fbe4b03c9a25e
2	601ac142be37ce2ead43755d	511111819905	Baking	BAKING	test brand @1612366146176	0.0	TEST BRANDCODE	Cogs	601ac142be37ce2ead4
3	601ac142be37ce2ead43755d	511111819905	Baking	BAKING	test brand @1612366146176	0.0	TEST BRANDCODE	Cogs	601ac142be37ce2ead4
4	601ac142be37ce2ead43755a	511111519874	Baking	BAKING	test brand @1612366146051	0.0	TEST BRANDCODE	Cogs	601ac142be37ce2ead4

ER Diagram

Below is the ER diagram of the structured data demonstrating the relationship and the cardinalities of each table.

After exploding the nested data in columns of receipts_data and brands_data, we get multiple sub columns which narrows down all the necessary values needed to create the relationship. In receipts_data, we exploded and normalized "rewardsReceiptItemList" column and in brands_data, it was "cpg". As we can see that receipts_data and users_data both had a connection through primary key as '_id' in receipts_data, "_id" in users_data, "_id" in brands_data with "userId" being the foreign key for the respective "_id" primary for users_data. However, we notice that theres no specific key that can help relate brands_data and receipts_data since barcodes are unique per brand and NOT per item, same for brandcode and categoryCode. When cpg(consumer packaged good) was exploded, we got cpg.id.oid but it did not meet the requirements - wasnt unique. So connect them, we introduced a new column named 'brandId' as a foreign key in receipts_data to match it with the primary key '_id' in brands_data.

Now, relationship between receipts_data to users_data is "many with optional to mandatory one" because one user can have multiple receipts or no receipt but a receipt can only have one user. The relationship between receipts_data and brands_data is "many optional to many" because a receipt can have items of multiple brands or no items at all so no brand however there would be many receipts with one or many brands.

Note1 - there are many columns that are associated with 'rewardsReceiptItemList' column which are not included in this. Note2 - There are multiple issues with '_id' in all three tables as when you explode and handle the "unstructured" data to transform it into a structured format, the '_id' no longer stays unique and cant be termed as primary keys. We can either introduce new 'ids' for every table and connect them or we as mentioned in the data dictionary that they are all UUIDs are a universally unique identifier (UUID) - a 128-bit label used for information in computer systems, so we can replace all the repeated ones with new UUIDs to make all of them unique however, we need a lot more indepth information on how these uuids are connected to other tables along with whether they are unique to items or not.