In [1]:
```python
import pandas as pd
from itertools import combinations
```

In [2]:
```python
# Creating a function that comes handy for identifying the granularity of the provided datasets
def find_unique_columns(df, columns = None):

    if not columns:
        columns = df.columns

    for i in range(1, len(columns) + 1):
        for combo in combinations(columns, i):
            if not df.duplicated(subset=list(combo)).any():
                return combo  # Return the first unique combination (shortest)

    return unique_combinations
```

In [3]:
```python
# Creating a function identifying some of the common data quality issues, returns a collection of the analyzed aspects
def find_data_quality_issues(df, unique_key = None, check_big_categories = None, null_tolerance = 50):

    issues = {}

    # Check for duplicated rows
    duplicated_rows = df.duplicated().any()
    issues['duplicated_rows'] = duplicated_rows

    # Find columns with lots of missing values
    null_pct = df.isnull().mean() * 100
    over_pct_nulls = null_pct[null_pct > null_tolerance].index.tolist()
    issues[f'over_{null_tolerance}pct_nulls'] = over_pct_nulls

    # Percentage of these highly unpopulated columns
    percentage_nulls = (len(over_pct_nulls) / len(df.columns)) * 100
    issues[f'percentage_columns_with_{null_tolerance}pct_nulls'] = percentage_nulls

    # Check for columns with constant values
    constant_columns = [col for col in df.columns if df[col].nunique() == 1]
    issues['constant_columns'] = constant_columns

    # Check for columns with big categories (threshold: > 50 unique values, could potentially make this configurable)
    if check_big_categories:
        big_categories = [col for col in check_big_categories if df[col].nunique() > 50]
        issues['big_categories'] = big_categories

    # Check for numeric columns with negative values (if expected to be non-negative)
    numeric_columns = df.select_dtypes(include=['number']).columns
```

```python
    negative_values_columns = [col for col in numeric_columns if (df[col] < 0).any()]
    issues['negative_values_columns'] = negative_values_columns

    # Check if the unique key is unique
    if unique_key:
        if isinstance(unique_key, str):  # If a single column is provided
            unique_key = [unique_key]

        primary_key_unique = not df.duplicated(subset=unique_key).any()
        issues['primary_key_unique'] = primary_key_unique
    else:
        issues['primary_key_unique'] = None  # No unique key provided to check

    # Return the collection of issues
    return issues
```

In [4]:
```python
# Identifies what makes the assumed primary key not unique in the dataset and potentially what other fields we should include as a
def find_differences_within_key(df, a_key):

    # Initialize a dictionary to store results
    differences = {}

    # Group the DataFrame by the key column
    grouped = df.groupby(a_key)

    # Iterate through each group
    for key, group in grouped:
        # Find columns with differing values in the group
        differing_columns = []
        for col in df.columns:
            if col != a_key and group[col].nunique() > 1:  # Check for differing values
                differing_columns.append(col)
        if differing_columns:
            differences[key] = differing_columns

    return differences
```

In [5]:
```python
def process_sample_json(name):
    data_raw = pd.read_json(f'{name}s.json', lines=True)
    data  = pd.json_normalize(data_raw.to_dict(orient='records')).dropna(axis=1, how='all')
    data.rename(columns={
        '_id.$oid': f'{name}_uuid' if name != "user" else 'userID',
        'cpg.$id.$oid': 'cpg_id',
        'cpg.$ref': 'cpg_ref'
        }, inplace=True)
    for col in data.columns:
        if "date" in col.lower():  # Check for date-related columns
```

```python
            data[col] = pd.to_datetime(data[col], unit='ms')  # Convert milliseconds to datetime
    data.columns = [col.replace(".$date", "") for col in data.columns]
    return data
```

In [6]:
```python
# Create users table/df
users = process_sample_json('user')
users
```

Out[6]:

	active	role	signUpSource	state	userID	createdDate	lastLogin
0	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	2021-01-03 15:24:04.800	2021-01-03 15:25:37.857999872
1	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	2021-01-03 15:24:04.800	2021-01-03 15:25:37.857999872
2	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	2021-01-03 15:24:04.800	2021-01-03 15:25:37.857999872
3	True	consumer	Email	WI	5ff1e1eacfcf6c399c274ae6	2021-01-03 15:25:30.554	2021-01-03 15:25:30.596999936
4	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	2021-01-03 15:24:04.800	2021-01-03 15:25:37.857999872
...	...	...	...	...	...	...	...
490	True	fetch-staff	NaN	NaN	54943462e4b07e684157a532	2014-12-19 14:21:22.381	2021-03-05 16:52:23.204000000
491	True	fetch-staff	NaN	NaN	54943462e4b07e684157a532	2014-12-19 14:21:22.381	2021-03-05 16:52:23.204000000
492	True	fetch-staff	NaN	NaN	54943462e4b07e684157a532	2014-12-19 14:21:22.381	2021-03-05 16:52:23.204000000
493	True	fetch-staff	NaN	NaN	54943462e4b07e684157a532	2014-12-19 14:21:22.381	2021-03-05 16:52:23.204000000
494	True	fetch-staff	NaN	NaN	54943462e4b07e684157a532	2014-12-19 14:21:22.381	2021-03-05 16:52:23.204000000

495 rows × 7 columns

In [7]:
```python
users.describe()
```

Out[7]:

	createdDate	lastLogin
count	495	433
mean	2020-08-06 01:34:47.878830592	2021-01-23 07:48:00.578216960
min	2014-12-19 14:21:22.381000	2018-05-07 17:23:40.003000064
25%	2021-01-04 19:30:17.483500032	2021-01-08 18:14:53.928000
50%	2021-01-13 20:19:38.720999936	2021-01-21 13:57:48.697999872
75%	2021-01-25 17:31:59.408999936	2021-02-03 15:34:11.043000064
max	2021-02-12 14:11:06.240000	2021-03-05 16:52:23.204000

In [8]:

```
# Finding: primary key not unique and exact duplicated rows
find_data_quality_issues(users, unique_key = 'userID', check_big_categories = ['active', 'signUpSource', 'state'])
```

Out[8]:

```
{'duplicated_rows': True,
 'over_50pct_nulls': [],
 'percentage_columns_with_50pct_nulls': 0.0,
 'constant_columns': [],
 'big_categories': [],
 'negative_values_columns': [],
 'primary_key_unique': False}
```

In [9]:

```
active_users = users[users['active'] == True]
```

In [10]:

```
# Finding: the issue persists even if we filter the dataset to "active" users only
find_data_quality_issues(active_users, unique_key = 'userID', check_big_categories = ['signUpSource', 'state'])
```

Out[10]:

```
{'duplicated_rows': True,
 'over_50pct_nulls': [],
 'percentage_columns_with_50pct_nulls': 0.0,
 'constant_columns': ['active'],
 'big_categories': [],
 'negative_values_columns': [],
 'primary_key_unique': False}
```

In [11]:

```
# This is telling us all of the not unique primary keys are caused by exactly duplicated rows
find_differences_within_key(users, 'userID')
```

Out[11]:

```
{}
```

In [12]:

```
# Checking duplicated rows
users[users.duplicated()]
```

Out[12]:

	active	role	signUpSource	state	userID	createdDate	lastLogin
1	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	2021-01-03 15:24:04.800	2021-01-03 15:25:37.857999872
2	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	2021-01-03 15:24:04.800	2021-01-03 15:25:37.857999872
4	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	2021-01-03 15:24:04.800	2021-01-03 15:25:37.857999872
5	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	2021-01-03 15:24:04.800	2021-01-03 15:25:37.857999872
8	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	2021-01-03 15:24:04.800	2021-01-03 15:25:37.857999872
...	...	...	...	...	...	...	...
490	True	fetch-staff	NaN	NaN	54943462e4b07e684157a532	2014-12-19 14:21:22.381	2021-03-05 16:52:23.204000000
491	True	fetch-staff	NaN	NaN	54943462e4b07e684157a532	2014-12-19 14:21:22.381	2021-03-05 16:52:23.204000000
492	True	fetch-staff	NaN	NaN	54943462e4b07e684157a532	2014-12-19 14:21:22.381	2021-03-05 16:52:23.204000000
493	True	fetch-staff	NaN	NaN	54943462e4b07e684157a532	2014-12-19 14:21:22.381	2021-03-05 16:52:23.204000000
494	True	fetch-staff	NaN	NaN	54943462e4b07e684157a532	2014-12-19 14:21:22.381	2021-03-05 16:52:23.204000000

283 rows × 7 columns

In [13]:

```
#Checking those duplicated userIDs

# Group by the unique key column and count occurrences
duplicate_counts = users.groupby('userID').size().reset_index(name='count')

# Filter only those keys that are duplicated (i.e., count > 1)
duplicated_userIDs = duplicate_counts[duplicate_counts['count'] > 1]

# Display the duplicated keys and their counts
duplicated_userIDs
```

Out[13]:

	userID	count
0	54943462e4b07e684157a532	20
3	59c124bae4b0299e55b0f330	18
4	5a43c08fe4b014fd6b6a0612	8
8	5fa41775898c7a11a6bcef3e	18
9	5fb0a078be5fc9775c1f3945	2
...	...	...
187	60186237c8b50e11d8454d5f	5
189	60189c74c8b50e11d8454eff	7
192	60189c94c8b50e11d8454f6b	4
195	601c2c05969c0b11f7d0b097	2
203	60229990b57b8a12187fe9e0	2

70 rows × 2 columns

In [14]:

```
# Percentage of duplicate rows
100 * users.duplicated().sum()/len(users)
```

Out[14]: 57.17171717171717

In [15]:

```
# Percentage of duplicate rows looking at active records only is even slightly higher
100 * active_users.duplicated().sum()/len(active_users)
```

Out[15]: 57.28744939271255

In [16]:

```
# Create brands table/df
brands = process_sample_json('brand')
brands.rename(columns={'name': 'brandName'}, inplace=True)
brands
```

Out[16]:

	barcode	category	categoryCode	brandName	topBrand	brandCode	brand_uuid	
0	511111019862	Baking	BAKING	test brand @1612366101024	0.0	NaN	601ac115be37ce2ead437551	601ac114be37c
1	511111519928	Beverages	BEVERAGES	Starbucks	0.0	STARBUCKS	601c5460be37ce2ead43755f	5332f5fbe4b03
2	511111819905	Baking	BAKING	test brand @1612366146176	0.0	TEST BRANDCODE @1612366146176	601ac142be37ce2ead43755d	601ac142be37c
3	511111519874	Baking	BAKING	test brand @1612366146051	0.0	TEST BRANDCODE @1612366146051	601ac142be37ce2ead43755a	601ac142be37c
4	511111319917	Candy & Sweets	CANDY_AND_SWEETS	test brand @1612366146827	0.0	TEST BRANDCODE @1612366146827	601ac142be37ce2ead43755e	5332fa12e4b03
...	...	...	...	...	...	...	...	
1162	511111116752	Baking	BAKING	test brand @1601644365844	NaN	NaN	5f77274dbe37ce6b592e90c0	5f77274dbe37c
1163	511111706328	Breakfast & Cereal	NaN	Dippin Dots® Cereal	NaN	DIPPIN DOTS CEREAL	5dc1fca91dda2c0ad7da64ae	53e10d6368abd
1164	511111416173	Candy & Sweets	CANDY_AND_SWEETS	test brand @1598639215217	NaN	TEST BRANDCODE @1598639215217	5f494c6e04db711dd8fe87e7	5332fa12e4b03
1165	511111400608	Grocery	NaN	LIPTON TEA Leaves	0.0	LIPTON TEA Leaves	5a021611e4b00efe02b02a57	5332f5f6e4b03
1166	511111019930	Baking	BAKING	test brand @1613158231643	0.0	TEST BRANDCODE @1613158231644	6026d757be37ce6369301468	6026d757be37c

1167 rows × 9 columns

In [17]:

```
# Finding: The granularity of this table is also 'barcode' and 'brandCode' (composite primary key)
find_unique_columns(brands, ['barcode', 'brandCode', 'cpg_id'])
```

Out[17]: ('barcode', 'brandCode')

In [18]:

```
# Finding: some brand names are associated to more than 1 brand ID
len(find_differences_within_key(brands, 'brandName'))
```

Out[18]: 11

In [19]:
```
find_differences_within_key(brands, 'brandName')
```

Out[19]:
```
{'Baken-Ets': ['barcode', 'brandCode', 'brand_uuid'],
 "Caleb's Kola": ['barcode', 'category', 'brandCode', 'brand_uuid', 'cpg_id'],
 'Diabetic Living Magazine': ['barcode', 'brand_uuid', 'cpg_id'],
 'Dippin Dots® Cereal': ['barcode', 'brandCode', 'brand_uuid', 'cpg_id'],
 'Health Magazine': ['barcode', 'brandCode', 'brand_uuid', 'cpg_id'],
 'Huggies': ['barcode', 'topBrand', 'brand_uuid', 'cpg_id'],
 "I CAN'T BELIEVE IT'S NOT BUTTER!": ['barcode',
  'category',
  'brandCode',
  'brand_uuid',
  'cpg_id'],
 'ONE A DAY® WOMENS': ['barcode', 'brandCode', 'brand_uuid'],
 'Pull-Ups': ['barcode', 'topBrand', 'brandCode', 'brand_uuid'],
 'Sierra Mist': ['barcode', 'brand_uuid', 'cpg_id', 'cpg_ref'],
 'V8 Hydrate': ['barcode', 'brand_uuid', 'cpg_id']}
```

In [20]:
```
#Finding: test data still in the dataset
brands[brands['brandName'].str.contains('test', case=False, na=False)]
```

Out[20]:

	barcode	category	categoryCode	brandName	topBrand	brandCode	brand_uuid	
0	511111019862	Baking	BAKING	test brand @1612366101024	0.0	NaN	601ac115be37ce2ead437551	601ac114be37ce
2	511111819905	Baking	BAKING	test brand @1612366146176	0.0	TEST BRANDCODE @1612366146176	601ac142be37ce2ead43755d	601ac142be37ce
3	511111519874	Baking	BAKING	test brand @1612366146051	0.0	TEST BRANDCODE @1612366146051	601ac142be37ce2ead43755a	601ac142be37ce
4	511111319917	Candy & Sweets	CANDY_AND_SWEETS	test brand @1612366146827	0.0	TEST BRANDCODE @1612366146827	601ac142be37ce2ead43755e	5332fa12e4b03
5	511111719885	Baking	BAKING	test brand @1612366146091	0.0	TEST BRANDCODE @1612366146091	601ac142be37ce2ead43755b	601ac142be37ce
...	...	...	...	...	...	...	...	
1152	511111715559	Baking	BAKING	test brand @1597935986434	NaN	TEST BRANDCODE @1597935986434	5f3e9172be37ce5a0e01b955	5f3e9172be37ce
1158	511111716648	Baking	BAKING	test brand @1600291349042	NaN	TEST BRANDCODE @1600291349043	5f628215be37ce78e6e2efab	5f628214be37c
1162	511111116752	Baking	BAKING	test brand @1601644365844	NaN	NaN	5f77274dbe37ce6b592e90c0	5f77274dbe37ce
1164	511111416173	Candy & Sweets	CANDY_AND_SWEETS	test brand @1598639215217	NaN	TEST BRANDCODE @1598639215217	5f494c6e04db711dd8fe87e7	5332fa12e4b03
1166	511111019930	Baking	BAKING	test brand @1613158231643	0.0	TEST BRANDCODE @1613158231644	6026d757be37ce6369301468	6026d757be37ce

432 rows × 9 columns

In [21]:

```
# Finding: 37% of the brands data appears to be for testing
test_records = brands['brandName'].str.contains('test', case=False, na=False)
(test_records.sum() / len(brands)) * 100
```

Out[21]: 37.01799485861182

In [22]:
```
# Finding: brandCode not always filled
brands[brands['brandCode'].isnull()]
```

Out[22]:

	barcode	category	categoryCode	brandName	topBrand	brandCode	brand_uuid	cpg_id
0	511111019862	Baking	BAKING	test brand @1612366101024	0.0	NaN	601ac115be37ce2ead437551	601ac114be37ce2ead437550
11	511111102540	NaN	NaN	MorningStar	NaN	NaN	57c08106e4b0718ff5fcb02c	5332f5f2e4b03c9a25efd0aa
18	511111317364	Baking	BAKING	test brand @1605535049181	0.0	NaN	5fb28549be37ce522e165cb5	5fb28549be37ce522e165cb4
23	511111303947	NaN	NaN	Bottled Starbucks	NaN	NaN	5332f5fee4b03c9a25efd0bd	53e10d6368abd3c7065097cc
24	511111802914	NaN	NaN	Full Throttle	NaN	NaN	5332fa7ce4b03c9a25efd22e	5332f5ebe4b03c9a25efd0a8
...	...	...	...	...	...	...	...	...
1135	511111405184	NaN	NaN	Do It Yourself	NaN	NaN	5d658fca6d5f3b23d1bc7912	53e10d6368abd3c7065097cc
1144	511111202516	NaN	NaN	Corona	NaN	NaN	57c08242e4b0718ff5fcb032	5332f7a7e4b03c9a25efd134
1146	511111703105	NaN	NaN	Bellatoria	NaN	NaN	5332fa12e4b03c9a25efd1e6	5332fa12e4b03c9a25efd1e7
1157	511111303015	NaN	NaN	DASANI	NaN	NaN	5332fa75e4b03c9a25efd221	5332f5ebe4b03c9a25efd0a8
1162	511111116752	Baking	BAKING	test brand @1601644365844	NaN	NaN	5f77274dbe37ce6b592e90c0	5f77274dbe37ce6b592e90bf

234 rows × 9 columns

In [23]:
```
# Finding: "categoryCode" not aligned with "category", contains a lot more nulls
find_data_quality_issues(brands, unique_key = 'brand_uuid', check_big_categories = ['category', 'categoryCode', 'cpg_ref'])
```

Out[23]:
```
{'duplicated_rows': False,
 'over_50pct_nulls': ['categoryCode', 'topBrand'],
 'percentage_columns_with_50pct_nulls': 22.22222222222222,
 'constant_columns': [],
 'big_categories': [],
 'negative_values_columns': [],
 'primary_key_unique': True}
```

In [24]:
```
# Dive into the exact gap, "category" only has 13% missing values while "categoryCode" has over 55%
brands.isnull().mean().sort_values(ascending=False) * 100
```

Out[24]:
```
categoryCode    55.698372
topBrand        52.442159
brandCode       20.051414
category        13.281919
barcode          0.000000
brandName        0.000000
brand_uuid       0.000000
cpg_id           0.000000
cpg_ref          0.000000
dtype: float64
```

In [25]:
```
# Create receipts table/df
receipts_full = process_sample_json('receipt')
receipts = receipts_full.drop(columns=['rewardsReceiptItemList']) #decoupling the receiptItem info which will be a separate table
receipts
```

Out[25]:

	bonusPointsEarned	bonusPointsEarnedReason	pointsEarned	purchasedItemCount	rewardsReceiptStatus	totalSpent	
0	500.0	Receipt number 2 completed, bonus point schedu...	500.0	5.0	FINISHED	26.00	5ff1e1eacfcf6
1	150.0	Receipt number 5 completed, bonus point schedu...	150.0	2.0	FINISHED	11.00	5ff1e194b6a9c
2	5.0	All-receipts receipt bonus	5.0	1.0	REJECTED	10.00	5ff1e1f1cfcf6
3	5.0	All-receipts receipt bonus	5.0	4.0	FINISHED	28.00	5ff1e1eacfcf6
4	5.0	All-receipts receipt bonus	5.0	2.0	FINISHED	1.00	5ff1e194b6a9c
...	...	...	...	...	...	...	
1114	25.0	COMPLETE_NONPARTNER_RECEIPT	25.0	2.0	REJECTED	34.96	5fc961c3b8cfc
1115	NaN	NaN	NaN	NaN	SUBMITTED	NaN	5fc961c3b8cfc
1116	NaN	NaN	NaN	NaN	SUBMITTED	NaN	5fc961c3b8cfc
1117	25.0	COMPLETE_NONPARTNER_RECEIPT	25.0	2.0	REJECTED	34.96	5fc961c3b8cfc
1118	NaN	NaN	NaN	NaN	SUBMITTED	NaN	5fc961c3b8cfc

1119 rows × 14 columns

In [26]:

```
find_data_quality_issues(receipts, unique_key = 'receipt_uuid')
```

Out[26]:

```
{'duplicated_rows': False,
 'over_50pct_nulls': ['bonusPointsEarned',
  'bonusPointsEarnedReason',
  'pointsAwardedDate'],
 'percentage_columns_with_50pct_nulls': 21.428571428571427,
 'constant_columns': [],
 'negative_values_columns': [],
 'primary_key_unique': True}
```

In [27]:
```python
receipts.describe()
```

Out[27]:

	bonusPointsEarned	pointsEarned	purchasedItemCount	totalSpent	createDate	dateScanned	finishedDate	m
count	544.000000	609.000000	635.00000	684.000000	1119	1119	568	
mean	238.893382	585.962890	14.75748	77.796857	2021-01-28 02:09:41.600271616	2021-01-28 02:09:41.600272384	2021-01-19 12:10:05.020589568	2 15:14:28.
min	5.000000	0.000000	0.00000	0.000000	2020-10-30 20:17:59	2020-10-30 20:17:59	2021-01-03 15:24:10	2
25%	5.000000	5.000000	1.00000	1.000000	2021-01-14 19:13:03.690499840	2021-01-14 19:13:03.690499840	2021-01-08 21:22:42.500000	2 21:32
50%	45.000000	150.000000	2.00000	18.200000	2021-01-29 17:18:22	2021-01-29 17:18:22	2021-01-19 21:13:57.500000	2
75%	500.000000	750.000000	5.00000	34.960000	2021-02-07 13:20:13.736999936	2021-02-07 13:20:13.736999936	2021-01-27 17:42:13.500000	2 13:20:13.
max	750.000000	10199.800000	689.00000	4721.950000	2021-03-01 23:17:34.772000	2021-03-01 23:17:34.772000	2021-02-26 22:36:25	2 23:17
std	299.091731	1357.166947	61.13424	347.110349	NaN	NaN	NaN	

In [28]:
```python
receipts.isnull().mean().sort_values(ascending=False) * 100
```

Out[28]:
```
pointsAwardedDate          52.010724
bonusPointsEarned          51.385165
bonusPointsEarnedReason    51.385165
finishedDate               49.240393
pointsEarned               45.576408
purchasedItemCount         43.252904
purchaseDate               40.035746
totalSpent                 38.873995
rewardsReceiptStatus        0.000000
userId                      0.000000
dateScanned                 0.000000
createDate                  0.000000
receipt_uuid                0.000000
modifyDate                  0.000000
dtype: float64
```

In [29]:
```python
# Create receiptItems table/df
receiptItems_raw = receipts_full[['receipt_uuid', 'rewardsReceiptItemList']]
receiptItems_exploded = receiptItems_raw.explode('rewardsReceiptItemList').reset_index(drop=True)
```

```python
receiptItems_flattened = pd.json_normalize(receiptItems_exploded['rewardsReceiptItemList'])
receiptItems = pd.concat([receiptItems_exploded["receipt_uuid"], receiptItems_flattened], axis=1)
receiptItems['receiptItemId'] = pd.util.hash_pandas_object(receiptItems[['receipt_uuid', 'partnerItemId']], index=False)
receiptItems
```

Out[29]:

	receipt_uuid	barcode	description	finalPrice	itemPrice	needsFetchReview	partnerItemId	preventTargetGapPoints
0	5ff1e1eb0a720f0523000575	4011	ITEM NOT FOUND	26.00	26.00	False	1	True
1	5ff1e1bb0a720f052300056b	4011	ITEM NOT FOUND	1	1	NaN	1	NaN
2	5ff1e1bb0a720f052300056b	028400642255	DORITOS TORTILLA CHIP SPICY SWEET CHILI REDUCE...	10.00	10.00	True	2	True
3	5ff1e1f10a720f052300057a	NaN	NaN	NaN	NaN	False	1	True
4	5ff1e1ee0a7214ada100056f	4011	ITEM NOT FOUND	28.00	28.00	False	1	True
...	...	...	...	...	...	...	...	...
7376	603d0b710a720fde1000042a	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7377	603cf5290a720fde10000413	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7378	603ce7100a7217c72c000405	B076FJ92M4	mueller austria hypergrind precision electric ...	22.97	22.97	NaN	0	NaN
7379	603ce7100a7217c72c000405	B07BRRLSVC	thindust summer face mask - sun protection nec...	11.99	11.99	NaN	1	NaN
7380	603c4fea0a7217c72c000389	NaN	NaN	NaN	NaN	NaN	NaN	NaN

7381 rows × 36 columns

In [30]:
```python
# Findings:
    #1. significant amount of columns have less than 50% population rate, "rewardsProductPartnerId", 'brandCode' and "barcode" are
        #, which makes the join to the "brands" table less ideal
    #2. 'preventTargetGapPoints, 'userFlaggedNewItem', 'deleted' either not available (key doesn't exist in json) or True,
        # 'pointsNotAwardedReason' either not available or "Action not allowed for user and CPG"
find_data_quality_issues(receiptItems, unique_key = 'receiptItemId')
```

Out[30]:
```
{'duplicated_rows': False,
 'over_50pct_nulls': ['barcode',
  'needsFetchReview',
  'preventTargetGapPoints',
  'userFlaggedBarcode',
  'userFlaggedNewItem',
  'userFlaggedPrice',
  'userFlaggedQuantity',
  'needsFetchReviewReason',
  'pointsNotAwardedReason',
  'pointsPayerId',
  'rewardsGroup',
  'rewardsProductPartnerId',
  'userFlaggedDescription',
  'originalMetaBriteBarcode',
  'originalMetaBriteDescription',
  'brandCode',
  'competitorRewardsGroup',
  'itemNumber',
  'originalMetaBriteQuantityPurchased',
  'pointsEarned',
  'targetPrice',
  'competitiveProduct',
  'originalFinalPrice',
  'originalMetaBriteItemPrice',
  'deleted',
  'priceAfterCoupon',
  'metabriteCampaignId'],
 'percentage_columns_with_50pct_nulls': 75.0,
 'constant_columns': ['preventTargetGapPoints',
  'userFlaggedNewItem',
  'pointsNotAwardedReason',
  'deleted'],
 'negative_values_columns': [],
 'primary_key_unique': True}
```

In [31]:
```python
receiptItems[['userFlaggedNewItem',  'pointsNotAwardedReason',  'deleted']].isnull().mean().sort_values(ascending=False) * 100
```

```
Out[31]: deleted                   99.878065
         userFlaggedNewItem        95.623899
         pointsNotAwardedReason    95.393578
         dtype: float64
```

```
In [32]: # Significant amount of missing values for brand identifiers
         receiptItems[['rewardsProductPartnerId', 'barcode', 'brandCode']].isnull().mean().sort_values(ascending=False) * 100
```

```
Out[32]: rewardsProductPartnerId    69.258908
         brandCode                  64.774421
         barcode                    58.135754
         dtype: float64
```

```
In [33]: receiptItems.isnull().mean().sort_values(ascending=False) * 100
```

```
Out[33]: originalMetaBriteItemPrice          99.878065
         originalFinalPrice                  99.878065
         deleted                             99.878065
         originalMetaBriteDescription        99.864517
         originalMetaBriteQuantityPurchased  99.796776
         originalMetaBriteBarcode            99.038071
         itemNumber                          97.927110
         userFlaggedDescription              97.222599
         needsFetchReviewReason              97.032922
         competitorRewardsGroup              96.274218
         userFlaggedQuantity                 95.949058
         userFlaggedPrice                    95.949058
         userFlaggedNewItem                  95.623899
         userFlaggedBarcode                  95.434223
         pointsNotAwardedReason              95.393578
         preventTargetGapPoints              95.149709
         targetPrice                         94.878743
         competitiveProduct                  91.261347
         needsFetchReview                    88.985232
         metabriteCampaignId                 88.307817
         pointsEarned                        87.440726
         priceAfterCoupon                    87.047825
         pointsPayerId                       82.834304
         rewardsGroup                        76.547893
         rewardsProductPartnerId             69.258908
         brandCode                           64.774421
         barcode                             58.135754
         originalReceiptItemText             21.961794
         discountedItemPrice                 21.839859
         description                         11.123154
         itemPrice                            8.318656
         finalPrice                           8.318656
         quantityPurchased                    8.318656
         partnerItemId                        5.961252
         receipt_uuid                         0.000000
         receiptItemId                        0.000000
         dtype: float64
```

```
In [34]: find_data_quality_issues(receiptItems, unique_key = 'receiptItemId', null_tolerance = 90)
```

Out[34]:
```
{'duplicated_rows': False,
 'over_90pct_nulls': ['preventTargetGapPoints',
  'userFlaggedBarcode',
  'userFlaggedNewItem',
  'userFlaggedPrice',
  'userFlaggedQuantity',
  'needsFetchReviewReason',
  'pointsNotAwardedReason',
  'userFlaggedDescription',
  'originalMetaBriteBarcode',
  'originalMetaBriteDescription',
  'competitorRewardsGroup',
  'itemNumber',
  'originalMetaBriteQuantityPurchased',
  'targetPrice',
  'competitiveProduct',
  'originalFinalPrice',
  'originalMetaBriteItemPrice',
  'deleted'],
 'percentage_columns_with_90pct_nulls': 50.0,
 'constant_columns': ['preventTargetGapPoints',
  'userFlaggedNewItem',
  'pointsNotAwardedReason',
  'deleted'],
 'negative_values_columns': [],
 'primary_key_unique': True}
```