Part I: Data exploration

Import Libraries

```python
import pandas as pd
import numpy as np
import missingno as msno
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

1. User data

Get dimensions, data types, and # non-nulls.

```python
# read user data
user_dat = pd.read_csv('USER_.csv')

# 100k rows and 6 columns. All 6 columns have a generic object type.
All but 2 are missing columns.
print(user_dat.info())
user_dat.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 6 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   ID            100000 non-null  object
 1   CREATED_DATE  100000 non-null  object
 2   BIRTH_DATE    96325 non-null   object
 3   STATE         95188 non-null   object
 4   LANGUAGE      69492 non-null   object
 5   GENDER        94108 non-null   object
dtypes: object(6)
memory usage: 4.6+ MB
None

                         ID               CREATED_DATE  \
0  5ef3b4f17053ab141787697d  2020-06-24 20:17:54.000 Z
1  5ff220d383fcfc12622b96bc  2021-01-03 19:53:55.000 Z
2  6477950aa55bb77a0e27ee10  2023-05-31 18:42:18.000 Z
3  658a306e99b40f103b63ccf8  2023-12-26 01:46:22.000 Z
4  653cf5d6a225ea102b7ecdc2  2023-10-28 11:51:50.000 Z

                  BIRTH_DATE STATE LANGUAGE  GENDER
```

```
0  2000-08-11 00:00:00.000 Z    CA   es-419  female
1  2001-09-24 04:00:00.000 Z    PA       en  female
2  1994-10-28 00:00:00.000 Z    FL   es-419  female
3                        NaN    NC       en     NaN
4  1972-03-19 00:00:00.000 Z    PA       en  female
```

Let's change the data types of the date columns. The rest are appropriate.

```
# change created_date, birth_date to datetime type
user_dat['CREATED_DATE'] = pd.to_datetime(user_dat['CREATED_DATE'])
user_dat['BIRTH_DATE'] = pd.to_datetime(user_dat['BIRTH_DATE'])

# the others are fine as they are
user_dat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 6 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   ID            100000 non-null  object
 1   CREATED_DATE  100000 non-null  datetime64[ns, UTC]
 2   BIRTH_DATE    96325 non-null   datetime64[ns, UTC]
 3   STATE         95188 non-null   object
 4   LANGUAGE      69492 non-null   object
 5   GENDER        94108 non-null   object
dtypes: datetime64[ns, UTC](2), object(4)
memory usage: 4.6+ MB
```

Now that we know we have 100k rows of 6 columns and they have intuitive types, let's dig into their contents. Above we can already see that all but ID and CREATED_DATE have missing values. The language column is the least complete with ~30% missing. Let's check how many unique values are in each column.

```
# unique values by column
print(user_dat.nunique())
```

```
ID              100000
CREATED_DATE     99942
BIRTH_DATE       54721
STATE               52
LANGUAGE             2
GENDER              11
dtype: int64
```

Only ID is completely unique. Now looking at the most frequent distinct values in all columns.

```
# get 5 most frequent distinct values from each column
for column in user_dat.columns:
```

```
    print(f"Column: {column}")
    print(user_dat[column].value_counts().nlargest(5))
    print("-" * 30)
```

```
Column: ID
ID
5ef3b4f17053ab141787697d    1
5f889d85746cfc1620c10130    1
5dc6eb9192ad0e12e283bcb2    1
5f1de98e57441c14b826b270    1
5efe2d8d6e0151146c9a31bc    1
Name: count, dtype: int64
------------------------------
Column: CREATED_DATE
CREATED_DATE
2023-01-12 18:30:15+00:00    2
2019-08-28 02:21:44+00:00    2
2024-04-11 02:56:41+00:00    2
2024-03-11 17:03:02+00:00    2
2024-02-25 20:43:59+00:00    2
Name: count, dtype: int64
------------------------------
Column: BIRTH_DATE
BIRTH_DATE
1970-01-01 00:00:00+00:00    1272
1979-12-11 08:00:00+00:00      63
2000-12-12 00:00:00+00:00      28
2000-12-31 00:00:00+00:00      23
2001-01-01 00:00:00+00:00      16
Name: count, dtype: int64
------------------------------
Column: STATE
STATE
TX    9028
FL    8921
CA    8589
NY    5703
IL    3794
Name: count, dtype: int64
------------------------------
Column: LANGUAGE
LANGUAGE
en        63403
es-419     6089
Name: count, dtype: int64
------------------------------
Column: GENDER
GENDER
female                  64240
male                    25829
```

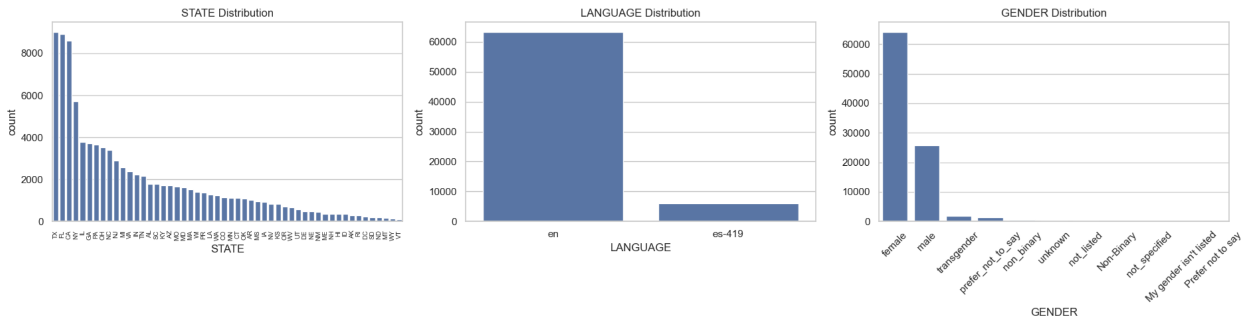
```
transgender              1772
prefer_not_to_say        1350
non_binary                473
Name: count, dtype: int64
------------------------------
```

Check for duplicates:

```
user_dat[user_dat.duplicated(keep=False)]

Empty DataFrame
Columns: [ID, CREATED_DATE, BIRTH_DATE, STATE, LANGUAGE, GENDER]
Index: []
```

There are no duplicate rows. Let's visualize this information with plots. First looking at the string columns.

```
# plot the distinct values in the three string columns
fig, axes = plt.subplots(1, 3, figsize=(19, 5))

# plot distinct values for state
order = user_dat['STATE'].value_counts().index
sns.countplot(x = 'STATE', data = user_dat, order = order, ax=axes[0])
axes[0].set_title('STATE Distribution')
axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=90)
axes[0].tick_params(axis='x', labelsize=7)

# same for language
order = user_dat['LANGUAGE'].value_counts().index
sns.countplot(x = 'LANGUAGE', data = user_dat, order = order,
ax=axes[1])
axes[1].set_title('LANGUAGE Distribution')

# same for gender
order = user_dat['GENDER'].value_counts().index
sns.countplot(x = 'GENDER', data = user_dat, order = order,
ax=axes[2])
axes[2].set_title('GENDER Distribution')
axes[2].set_xticklabels(axes[2].get_xticklabels(), rotation=45)

plt.tight_layout()
plt.show()
```

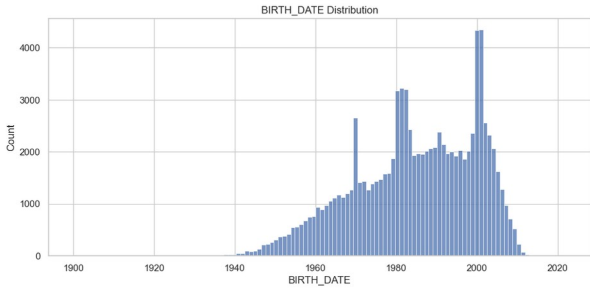


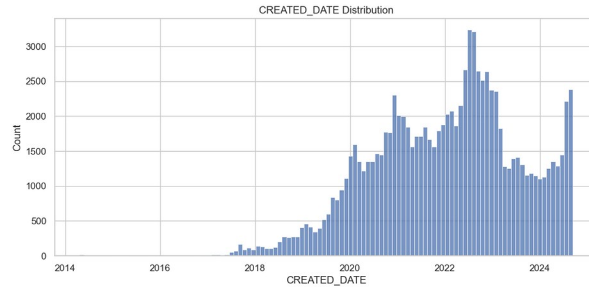
From these plots we can summarize these columns.

- State has missing rows and 52 distinct values (50 states + Puerto Rico and blanks).
- Language has missing rows and 2 distinct values for English and Spanish.
- The gender column has missing rows and 11 distinct values - it is messy. Some values code for the same thing like "not_listed" and "My gender isn't listed". This will need to be fixed.
- The meaning behind these columns as well as ID are self explanatory (they just describe the characteristics of the Fetch Awards users).

Now let's do the same plots for the 2 timestamp columns.

```
# plot the distributions of the time columns
fig, axes = plt.subplots(1, 2, figsize=(25, 5))

# plot distinct values for BIRTH_DATE
sns.histplot(data=user_dat, x='BIRTH_DATE', ax=axes[0])
axes[0].set_title('BIRTH_DATE Distribution')

# same for CREATED_DATE
sns.histplot(data=user_dat, x='CREATED_DATE', ax=axes[1])
axes[1].set_title('CREATED_DATE Distribution')

plt.show()

# print min and max of CREATED_DATE and BIRTH_DATE
print(f"CREATED_DATE min: {user_dat['CREATED_DATE'].min()}")
print(f"CREATED_DATE max: {user_dat['CREATED_DATE'].max()}")
print(f"BIRTH_DATE min: {user_dat['BIRTH_DATE'].min()}")
print(f"BIRTH_DATE max: {user_dat['BIRTH_DATE'].max()}")

# finally, check if all created dates are before the birth dates
print(user_dat[user_dat['CREATED_DATE'] < user_dat['BIRTH_DATE']])
```



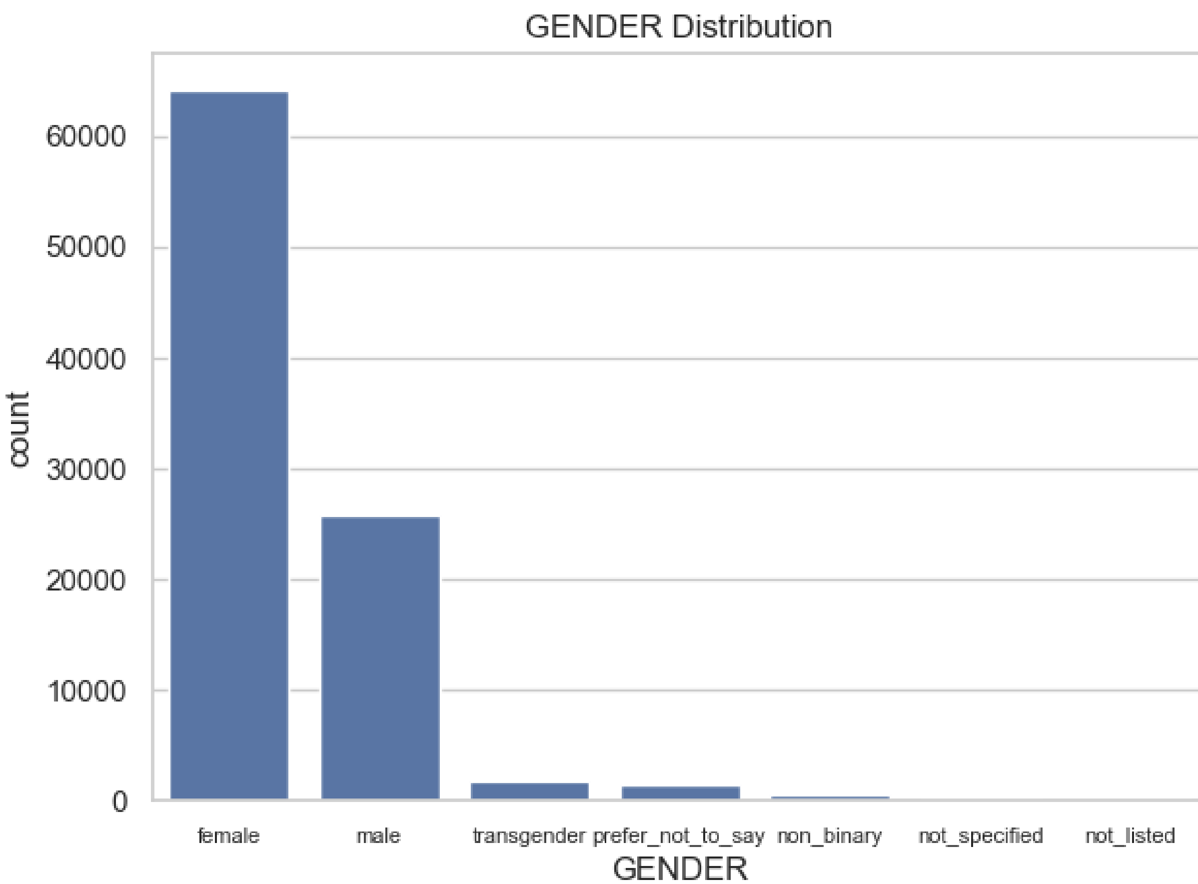



```
CREATED_DATE min: 2014-04-18 23:14:55+00:00
CREATED_DATE max: 2024-09-11 17:59:15+00:00
BIRTH_DATE min: 1900-01-01 00:00:00+00:00
BIRTH_DATE max: 2022-04-03 07:00:00+00:00
                             ID              CREATED_DATE  \
41974  5f31fc048fa1e914d38d6952 2020-08-11 02:01:41+00:00

                     BIRTH_DATE STATE LANGUAGE GENDER
41974 2020-10-02 15:27:28+00:00    CA      NaN    NaN
```

- BIRTH_DATE's meaning is self explanatory. BIRTH_DATE has a multimodal distribution, starting in 1900 and ending in 2022. I'm assuming this is user-inputed data since users with ages at both ends of the range are dubious (125 year old user vs. 2 year old user). It has peaks at 1970, 1980-1982, and in 2000.
- CREATED_DATE is presumably when the user account was made. CREATED_DATE is also multimodal and has peaks in late 2020, mid-2022 (absolute max), and at the end of the distribution around 7-2024. The range contains the last ~10 years.

Now let's clean up the gender column by consolidation. My assumptions are that the following possible values really code for the same thing.

```
# consolidate instances in the gender column that are the same
user_dat['GENDER'] = user_dat['GENDER'].replace('Non-Binary',
'non_binary')
user_dat['GENDER'] = user_dat['GENDER'].replace('Prefer not to say',
'prefer_not_to_say')
user_dat['GENDER'] = user_dat['GENDER'].replace('unknown',
'not_specified')
user_dat['GENDER'] = user_dat['GENDER'].replace("My gender isn't
listed", 'not_listed')
user_dat['GENDER'] = user_dat['GENDER'].replace('', 'not_specified')

# now check
plt.figure(figsize = (7, 5))
order = user_dat['GENDER'].value_counts().index
sns.countplot(x = 'GENDER', data = user_dat, order = order)
plt.title('GENDER Distribution')
ax = plt.gca()
ax.tick_params(axis = 'x', labelsize = 8)
plt.show()
```

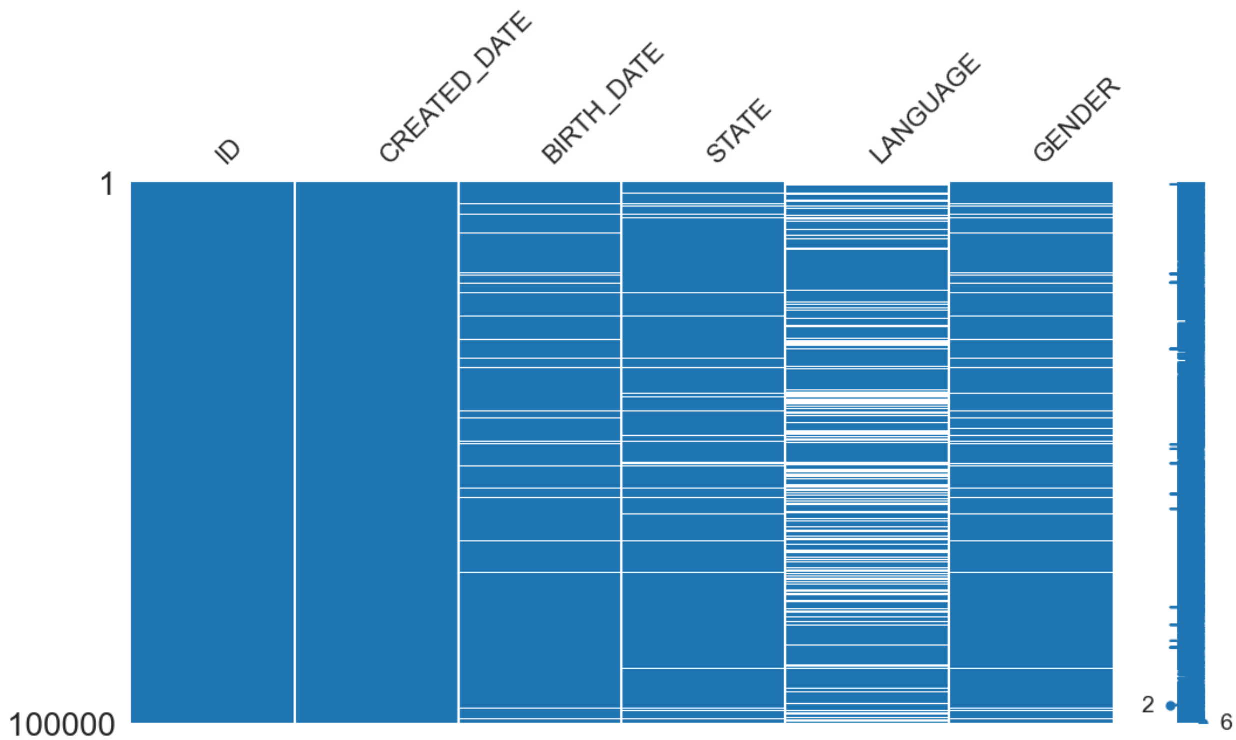


Now we only have 7 distinct gender values, instead of 11. Let's look closer at the missing data.

```
# rows of missing data / total rows
print(user_dat.isna().sum(axis=0) / user_dat.shape[0])

# visualize how much data is missing
msno.matrix(user_dat, color=(0.121, 0.46, 0.70), figsize = (12, 6))
```

```
ID              0.00000
CREATED_DATE    0.00000
BIRTH_DATE      0.03675
STATE           0.04812
LANGUAGE        0.30508
GENDER          0.05892
dtype: float64
```

```
<Axes: >
```



We have at least an ID for every observation and essentially every CREATION_DATE, while the others have substantial missing data. In summary, for the user data:

1. Are there data quality issues present?
- There is an instance of CREATED_DATE occurring before the BIRTH_DATE field.
- BIRTHDATE, STATE, LANGUAGE, and GENDER have missing rows.
- BIRTHDATE has unrealistic values, like those in the year 1900.
- GENDER has 11 distinct values, some of which code for the same thing like "not_listed" and "My gender isn't listed". This field is messy. I made the assumption that not_specified is distinct from not_listed and combined various other possible values.

1. Are there any fields that are challenging to understand?
- All fields are straightforward. ID appears to uniquely identify a Fetch user, and the other fields are attributes of the user. I'm making the assumption that this data is user provided so that is why gender has many posible values and the birthdate may not be accurate. For example some show an age of 125 and others show the birthdate occuring after the created date.

2. Transaction data

Get dimensions, data types, and # non-nulls.

```
# read transaction data and display
trans_dat = pd.read_csv('TRANSACTION_.csv')

# 50k rows and 8 columns. 7 columns are generic object type, 1 is a
float. 1 is missing rows.
```

```python
print(trans_dat.info())

# view data
trans_dat.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   RECEIPT_ID      50000 non-null  object 
 1   PURCHASE_DATE   50000 non-null  object 
 2   SCAN_DATE       50000 non-null  object 
 3   STORE_NAME      50000 non-null  object 
 4   USER_ID         50000 non-null  object 
 5   BARCODE         44238 non-null  float64
 6   FINAL_QUANTITY  50000 non-null  object 
 7   FINAL_SALE      50000 non-null  object 
dtypes: float64(1), object(7)
memory usage: 3.1+ MB
None

                             RECEIPT_ID PURCHASE_DATE  \
0  0000d256-4041-4a3e-adc4-5623fb6e0c99    2024-08-21   
1  0001455d-7a92-4a7b-a1d2-c747af1c8fd3    2024-07-20   
2  00017e0a-7851-42fb-bfab-0baa96e23586    2024-08-18   
3  000239aa-3478-453d-801e-66a82e39c8af    2024-06-18   
4  00026b4c-dfe8-49dd-b026-4c2f0fd5c6a1    2024-07-04   

                  SCAN_DATE STORE_NAME                   USER_ID  \
0  2024-08-21 14:19:06.539 Z    WALMART  63b73a7f3d310dceeabd4758   
1  2024-07-20 09:50:24.206 Z       ALDI  62c08877baa38d1a1f6c211a   
2  2024-08-19 15:38:56.813 Z    WALMART  60842f207ac8b7729e472020   
3  2024-06-19 11:03:37.468 Z  FOOD LION  63fcd7cea4f8442c3386b589   
4  2024-07-05 15:56:43.549 Z   RANDALLS  6193231ae9b3d75037b0f928   

        BARCODE FINAL_QUANTITY FINAL_SALE  
0  1.530001e+10           1.00             
1           NaN           zero       1.49  
2  7.874223e+10           1.00             
3  7.833997e+11           zero       3.49  
4  4.790050e+10           1.00             
```

After seeing the data, it looks like there are blanks that are not counted as nulls in FINAL_SALE. Let's code them as NaN so they're considered nulls.

```python
# search for rows with 1 or more spaces in FINAL_SALE. there are 12.5k of them.
trans_dat[trans_dat["FINAL_SALE"].str.contains(r'^\s*$', na = False)]
```

```python
# replace the spaces with NaN
trans_dat["FINAL_SALE"] = trans_dat["FINAL_SALE"].replace(r'^\s*$', 
np.nan, regex = True)

print(trans_dat.info())

# this is now fixed
trans_dat.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   RECEIPT_ID      50000 non-null  object 
 1   PURCHASE_DATE   50000 non-null  object 
 2   SCAN_DATE       50000 non-null  object 
 3   STORE_NAME      50000 non-null  object 
 4   USER_ID         50000 non-null  object 
 5   BARCODE         44238 non-null  float64
 6   FINAL_QUANTITY  50000 non-null  object 
 7   FINAL_SALE      37500 non-null  object 
dtypes: float64(1), object(7)
memory usage: 3.1+ MB
None

                             RECEIPT_ID PURCHASE_DATE  \
0  0000d256-4041-4a3e-adc4-5623fb6e0c99    2024-08-21   
1  0001455d-7a92-4a7b-a1d2-c747af1c8fd3    2024-07-20   
2  00017e0a-7851-42fb-bfab-0baa96e23586    2024-08-18   
3  000239aa-3478-453d-801e-66a82e39c8af    2024-06-18   
4  00026b4c-dfe8-49dd-b026-4c2f0fd5c6a1    2024-07-04   

                   SCAN_DATE STORE_NAME                   USER_ID  \
0  2024-08-21 14:19:06.539 Z    WALMART  63b73a7f3d310dceeabd4758   
1  2024-07-20 09:50:24.206 Z       ALDI  62c08877baa38d1a1f6c211a   
2  2024-08-19 15:38:56.813 Z    WALMART  60842f207ac8b7729e472020   
3  2024-06-19 11:03:37.468 Z  FOOD LION  63fcd7cea4f8442c3386b589   
4  2024-07-05 15:56:43.549 Z   RANDALLS  6193231ae9b3d75037b0f928   

        BARCODE FINAL_QUANTITY FINAL_SALE  
0  1.530001e+10           1.00        NaN  
1           NaN           zero       1.49  
2  7.874223e+10           1.00        NaN  
3  7.833997e+11           zero       3.49  
4  4.790050e+10           1.00        NaN  
```

Now let's change column types for the dates.

```
# change purchase_date, scan_date to datetime type
trans_dat['PURCHASE_DATE'] =
pd.to_datetime(trans_dat['PURCHASE_DATE'])
trans_dat['SCAN_DATE'] = pd.to_datetime(trans_dat['SCAN_DATE'])
trans_dat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   RECEIPT_ID      50000 non-null  object
 1   PURCHASE_DATE   50000 non-null  datetime64[ns]
 2   SCAN_DATE       50000 non-null  datetime64[ns, UTC]
 3   STORE_NAME      50000 non-null  object
 4   USER_ID         50000 non-null  object
 5   BARCODE         44238 non-null  float64
 6   FINAL_QUANTITY  50000 non-null  object
 7   FINAL_SALE      37500 non-null  object
dtypes: datetime64[ns, UTC](1), datetime64[ns](1), float64(1),
object(5)
memory usage: 3.1+ MB
```

Now that we know we have 50k rows of 8 columns, let's dig into their contents. Above we can already see that only BARCODE and FINAL_SALE have missing values. The FINAL_SALE column is ~25% missing. Looking at unique values:

```
# check how many unique values in each column
print(trans_dat.nunique())

# print the unique values for quantity
print(trans_dat['FINAL_QUANTITY'].unique())
```

```
RECEIPT_ID        24440
PURCHASE_DATE        89
SCAN_DATE         24440
STORE_NAME          954
USER_ID           17694
BARCODE           11027
FINAL_QUANTITY       87
FINAL_SALE         1434
dtype: int64
['1.00' 'zero' '2.00' '3.00' '4.00' '4.55' '2.83' '2.34' '0.46' '7.00'
 '18.00' '12.00' '5.00' '2.17' '0.23' '8.00' '1.35' '0.09' '2.58'
'1.47'
 '16.00' '0.62' '1.24' '1.40' '0.51' '0.53' '1.69' '6.00' '2.39'
'2.60'
 '10.00' '0.86' '1.54' '1.88' '2.93' '1.28' '0.65' '2.89' '1.44'
'2.75'
```

```
 '1.81' '276.00' '0.87' '2.10' '3.33' '2.54' '2.20' '1.93' '1.34'
'1.13'
 '2.19' '0.83' '2.61' '0.28' '1.50' '0.97' '0.24' '1.18' '6.22' '1.22'
 '1.23' '2.57' '1.07' '2.11' '0.48' '9.00' '3.11' '1.08' '5.53' '1.89'
 '0.01' '2.18' '1.99' '0.04' '2.25' '1.37' '3.02' '0.35' '0.99' '1.80'
 '3.24' '0.94' '2.04' '3.69' '0.70' '2.52' '2.27']
```

Looks like quantity can be a decimal value, this could make sense as some items can be sold by weight or other non-discrete units. No columns are completely unique for each row, meaning we'll have multiple rows with the same RECEIPT_ID and SCAN_DATES. Look at the most frequent distinct values in all columns.

```python
# get 5 most frequent distinct values from each column
for column in trans_dat.columns:
    print(f"Column: {column}")
    print(trans_dat[column].value_counts().nlargest(5))
    print("-" * 20)
```

```
Column: RECEIPT_ID
RECEIPT_ID
bedac253-2256-461b-96af-267748e6cecf    12
bc304cd7-8353-4142-ac7f-f3ccec720cb3     8
4ec870d2-c39f-4a40-bf8a-26a079409b20     8
2acd7e8d-37df-4e51-8ee5-9a9c8c1d9711     8
760c98da-5174-401f-a203-b839c4d406be     8
Name: count, dtype: int64
--------------------
Column: PURCHASE_DATE
PURCHASE_DATE
2024-06-15    774
2024-07-03    772
2024-07-01    752
2024-08-03    720
2024-07-13    712
Name: count, dtype: int64
--------------------
Column: SCAN_DATE
SCAN_DATE
2024-09-08 20:00:42.348000+00:00    12
2024-09-07 17:30:53.326000+00:00     8
2024-09-08 19:39:01.589000+00:00     8
2024-09-08 11:13:01.935000+00:00     8
2024-09-07 14:52:46.822000+00:00     8
Name: count, dtype: int64
--------------------
Column: STORE_NAME
STORE_NAME
WALMART                  21326
DOLLAR GENERAL STORE      2748
```

```
ALDI                        2640
KROGER                      1494
TARGET                      1484
Name: count, dtype: int64
--------------------
Column: USER_ID
USER_ID
64e62de5ca929250373e6cf5    22
604278958fe03212b47e657b    20
62925c1be942f00613f7365e    20
64063c8880552327897186a5    18
61d5f5d2c4525a3a478b386b    14
Name: count, dtype: int64
--------------------
Column: BARCODE
BARCODE
7.874222e+10    182
5.111115e+11    168
5.111110e+11    164
7.874229e+10    158
3.111112e+11    150
Name: count, dtype: int64
--------------------
Column: FINAL_QUANTITY
FINAL_QUANTITY
1.00    35698
zero    12500
2.00     1285
3.00      184
4.00      139
Name: count, dtype: int64
--------------------
Column: FINAL_SALE
FINAL_SALE
1.25    1323
1.00     744
2.99     588
1.99     586
3.99     567
Name: count, dtype: int64
--------------------
```

Quickly glancing at the fields, this is what I think they represent:

- RECEIPT_ID - appears to ID a receipt.
- PURCHASE_DATE - date of the purchase
- SCAN_DATE - when the Fetch membership was scanned. Not necessarily the same as the purchase date.
- STORE_NAME - location of purchase.

- USER_ID - user identifier, same as ID from user table.
- BARCODE - item barcode from the purchase.
- FINAL_QUANTITY - quantity of the product purchased. Can be a decimal.
- FINAL_SALE - dollar amount of sale.

Check for duplicates:

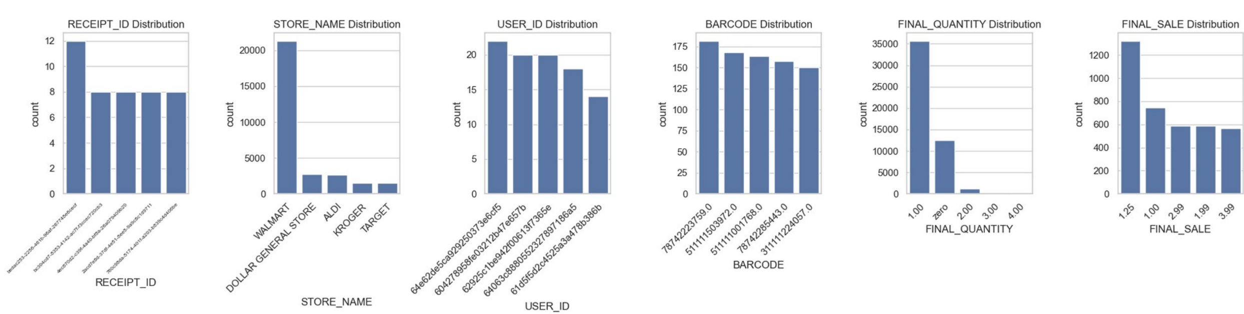
```
trans_dat[trans_dat.duplicated(keep=False)].sort_values(by=['RECEIPT_I
D']) #- yes

                                  RECEIPT_ID PURCHASE_DATE  \
40498  007d3232-3990-497f-a081-549e9e7a478b    2024-06-25
45553  007d3232-3990-497f-a081-549e9e7a478b    2024-06-25
49759  01a70fe0-026f-4bea-9da4-7d13bbf21e9a    2024-09-02
49758  01a70fe0-026f-4bea-9da4-7d13bbf21e9a    2024-09-02
32463  0273cbd8-1620-46c9-8e99-6971e850a2fc    2024-09-08
...                                      ...           ...
48463  f871a430-7fcb-4d95-989e-aa0b57497eca    2024-09-01
41604  fa8ab2d7-b051-47d7-bd56-d0d88997d367    2024-07-22
41593  fa8ab2d7-b051-47d7-bd56-d0d88997d367    2024-07-22
46640  fb825ba4-fe3b-45b4-a547-5a33d23e5e33    2024-08-24
28833  fb825ba4-fe3b-45b4-a547-5a33d23e5e33    2024-08-24

                              SCAN_DATE              STORE_NAME  \
40498 2024-06-27 21:21:53.442000+00:00  DOLLAR TREE STORES INC
45553 2024-06-27 21:21:53.442000+00:00  DOLLAR TREE STORES INC
49759 2024-09-07 16:02:39.835000+00:00                 WALMART
49758 2024-09-07 16:02:39.835000+00:00                 WALMART
32463 2024-09-08 22:17:11.989000+00:00                 WALMART
...                                ...                     ...
48463 2024-09-07 17:45:11.519000+00:00                  KROGER
41604 2024-07-31 21:26:56.929000+00:00                    ALDI
41593 2024-07-31 21:26:56.929000+00:00                    ALDI
46640 2024-08-25 13:58:08.848000+00:00                 WALMART
28833 2024-08-25 13:58:08.848000+00:00                 WALMART

                        USER_ID       BARCODE FINAL_QUANTITY
FINAL_SALE
40498  63a8dbf101cb7c888c6ad87d  7.920006e+10           1.00
1.25
45553  63a8dbf101cb7c888c6ad87d  7.920006e+10           1.00
1.25
49759  614e733372ba844aa8dc345e  4.178900e+10           1.00
0.52
49758  614e733372ba844aa8dc345e  4.178900e+10           1.00
0.52
32463  60e4f3ac34f82e1344669ee2  6.811311e+11           1.00
3.48
...                         ...           ...            ...         ..
.
```

```
48463  615c505eb220b85b9615f063           NaN           1.00
1.00
41604  653c241b909604bae9074b22           NaN           1.00
0.47
41593  653c241b909604bae9074b22           NaN           1.00
0.47
46640  61ed4fda0605d0323d86dced  7.874223e+10           1.00
0.78
28833  61ed4fda0605d0323d86dced  7.874223e+10           1.00
0.78

[320 rows x 8 columns]
```

There are 320 rows that are duplicates. There seems to be a pattern in the data for the FINAL_QUANTITY and FINAL_SALE columns. The first 25k rows of the dataset have alternating values of a blank FINAL_SALE and a "zero" FINAL_QUANTITY. Additionally there is the duplicates issue, which could be related. However this could make sense if 2 items on a receipt were identical but not included on the same line item. It depends on how this data is collected, ie if the OCR text extraction from the images is accurate or not. This point would need to be clarified with the team. Now let's visualize the data with plots.

```
# plot the distinct values in the 6 non-date columns. only look at the
top 5 values per column

fig, axes = plt.subplots(1, 6, figsize = (20, 5))

# plot distinct values for RECEIPT_ID
order = trans_dat['RECEIPT_ID'].value_counts().nlargest(5).index
sns.countplot(x = 'RECEIPT_ID', data =
trans_dat[trans_dat['RECEIPT_ID'].isin(order)], order = order,
ax=axes[0])
axes[0].set_title('RECEIPT_ID Distribution')
axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation = 45,
ha='right', rotation_mode = 'anchor')
axes[0].tick_params(axis='x', labelsize = 6)

# same for STORE_NAME
order = trans_dat['STORE_NAME'].value_counts().nlargest(5).index
sns.countplot(x = 'STORE_NAME', data =
trans_dat[trans_dat['STORE_NAME'].isin(order)], order = order,
ax=axes[1])
axes[1].set_title('STORE_NAME Distribution')
axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation = 45,
ha='right', rotation_mode = 'anchor')

# same for USER_ID
order = trans_dat['USER_ID'].value_counts().nlargest(5).index
sns.countplot(x = 'USER_ID', data =
trans_dat[trans_dat['USER_ID'].isin(order)], order = order,
```

```
ax=axes[2])
axes[2].set_title('USER_ID Distribution')
axes[2].set_xticklabels(axes[2].get_xticklabels(), rotation = 45,
ha='right', rotation_mode = 'anchor')

# same for BARCODE
order = trans_dat['BARCODE'].value_counts().nlargest(5).index
sns.countplot(x = 'BARCODE', data =
trans_dat[trans_dat['BARCODE'].isin(order)], order = order,
ax=axes[3])
axes[3].set_title('BARCODE Distribution')
axes[3].set_xticklabels(axes[3].get_xticklabels(), rotation = 45,
ha='right', rotation_mode = 'anchor')

# same for FINAL_QUANTITY
order = trans_dat['FINAL_QUANTITY'].value_counts().nlargest(5).index
sns.countplot(x = 'FINAL_QUANTITY', data =
trans_dat[trans_dat['FINAL_QUANTITY'].isin(order)], order = order,
ax=axes[4])
axes[4].set_title('FINAL_QUANTITY Distribution')
axes[4].set_xticklabels(axes[4].get_xticklabels(), rotation = 45,
ha='right', rotation_mode = 'anchor')

# same for FINAL_SALE
order = trans_dat['FINAL_SALE'].value_counts().nlargest(5).index
sns.countplot(x = 'FINAL_SALE', data =
trans_dat[trans_dat['FINAL_SALE'].isin(order)], order = order,
ax=axes[5])
axes[5].set_title('FINAL_SALE Distribution')
axes[5].set_xticklabels(axes[5].get_xticklabels(), rotation = 45,
ha='right', rotation_mode = 'anchor')

plt.tight_layout()
plt.show()
```



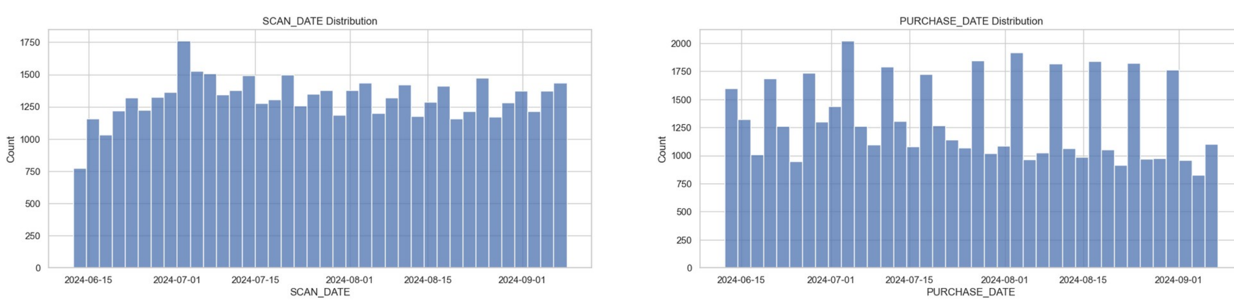
From these plots we can summarize these columns.

- RECEIPT_ID - appears clean. each value is a 36 character long alphanumeric.
- STORE_NAME - appears clean.
- USER_ID - appears clean and 24 digits long.

- BARCODE - appears clean. also is fully numeric. (had null values)
- FINAL_QUANTITY - appears clean except the weird value of "zero". I will replace this with a number and then change the type of the column.
- FINAL_SALE - appears clean. (had null values)

Now look at the date columns.

```
# date columns

# plot the distributions of the time columns
fig, axes = plt.subplots(1, 2, figsize=(25, 5))

# plot SCAN_DATE
sns.histplot(data=trans_dat, x='SCAN_DATE', ax=axes[0])
axes[0].set_title('SCAN_DATE Distribution')

# plot PURCHASE_DATE
sns.histplot(data=trans_dat, x='PURCHASE_DATE', ax=axes[1])
axes[1].set_title('PURCHASE_DATE Distribution')

plt.show()

# print min and max of PURCHASE_DATE and SCAN_DATE
print(f"PURCHASE_DATE min: {trans_dat['PURCHASE_DATE'].min()}")
print(f"PURCHASE_DATE max: {trans_dat['PURCHASE_DATE'].max()}")
print(f"SCAN_DATE min: {trans_dat['SCAN_DATE'].min()}")
print(f"SCAN_DATE max: {trans_dat['SCAN_DATE'].max()}")
```




```
PURCHASE_DATE min: 2024-06-12 00:00:00
PURCHASE_DATE max: 2024-09-08 00:00:00
SCAN_DATE min: 2024-06-12 06:36:34.910000+00:00
SCAN_DATE max: 2024-09-08 23:07:19.836000+00:00

# finally, check if all SCAN_DATEs are on the same day as the
PURCHASE_DATE
print(f"Rows with scan date on same day as purchase date:
{len(trans_dat[trans_dat['PURCHASE_DATE'].dt.date ==
trans_dat['SCAN_DATE'].dt.date])}")

# check if all SCAN_DATEs are after the PURCHASE_DATE
```

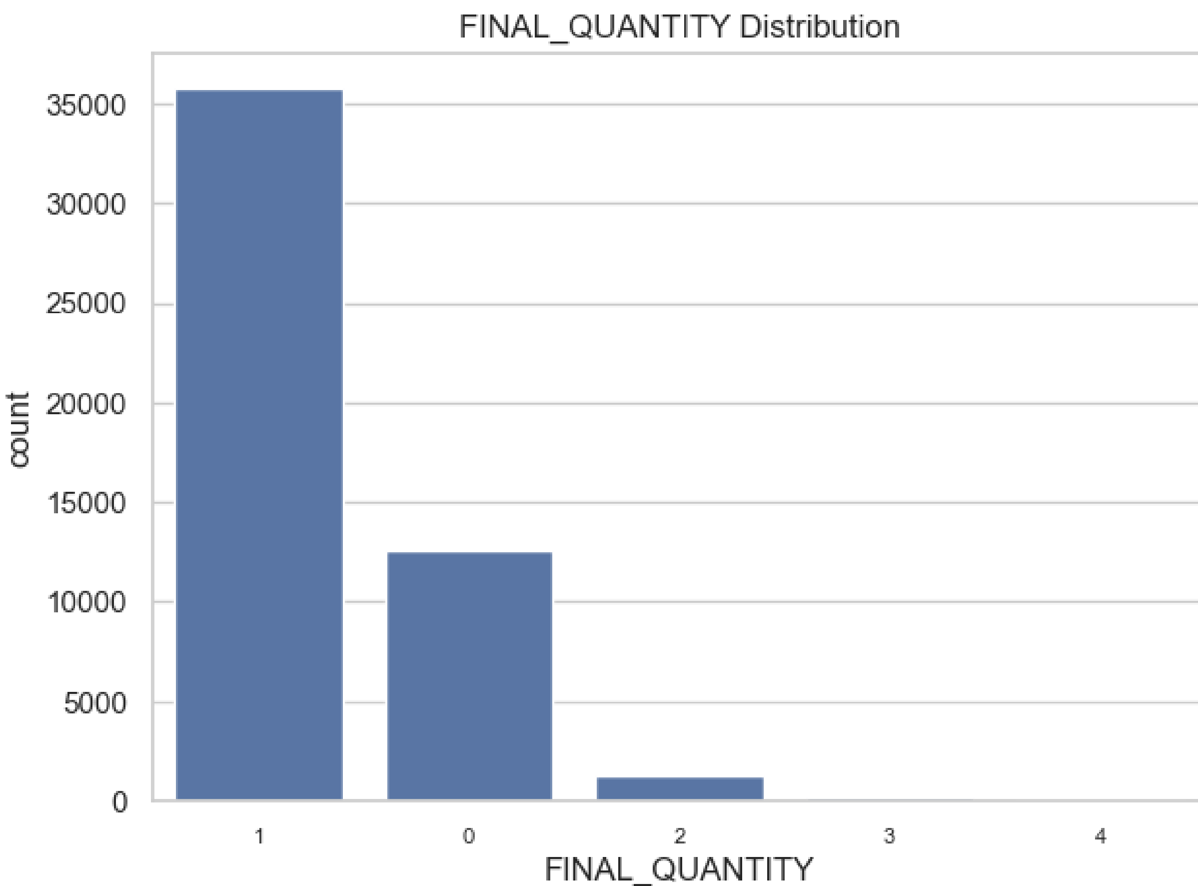
```
print(f"Rows with scan date after purchase date: 
{len(trans_dat[trans_dat['SCAN_DATE'].dt.date > 
trans_dat['PURCHASE_DATE'].dt.date])}")

# check if all SCAN_DATEs are within a week of the PURCHASE_DATE
trans_dat['SCAN_DATE_no_tz'] = 
trans_dat['SCAN_DATE'].dt.tz_localize(None).dt.date
trans_dat['check_dates'] = 
(pd.to_datetime(trans_dat['SCAN_DATE_no_tz']) - 
pd.to_datetime(trans_dat['PURCHASE_DATE'] )).dt.days <= 7
print(f"Rows with scan date within a week of purchase date: 
{len(trans_dat[trans_dat['check_dates'] == True])}")

trans_dat.drop('check_dates', axis=1, inplace=True)
```

```
Rows with scan date on same day as purchase date: 23822
Rows with scan date after purchase date: 26084
Rows with scan date within a week of purchase date: 46206
```

About half of the transactions have a SCAN_DATE and PURCHASE_DATE occuring on the same day. 92% are within the same week. The order of scanning and purchase dates can change.

Now let's do a final clean up the columns. Fix the 'zero' in FINAL_QUANTITY. Upon closer inspection of the original data, for the first 25k rows, FINAL_SALE and FINAL_QUANTITY had alternating values of "zero" and blank. These look like the only 2 columns impacted by this data corruption.

```
# replace "zero" with the number 0 in FINAL_QUANTITY
trans_dat['FINAL_QUANTITY'] = 
trans_dat['FINAL_QUANTITY'].replace('zero', 0)

# cast FINAL_QUANTITY as int
trans_dat['FINAL_QUANTITY'] = 
trans_dat['FINAL_QUANTITY'].astype(float).astype(int)

# now check
plt.figure(figsize = (7, 5))
order = trans_dat['FINAL_QUANTITY'].value_counts().nlargest(5).index
sns.countplot(x = 'FINAL_QUANTITY', data = 
trans_dat[trans_dat['FINAL_QUANTITY'].isin(order)], order = order)
plt.title('FINAL_QUANTITY Distribution')
ax = plt.gca()
ax.tick_params(axis='x', labelsize=8)
plt.show()
```

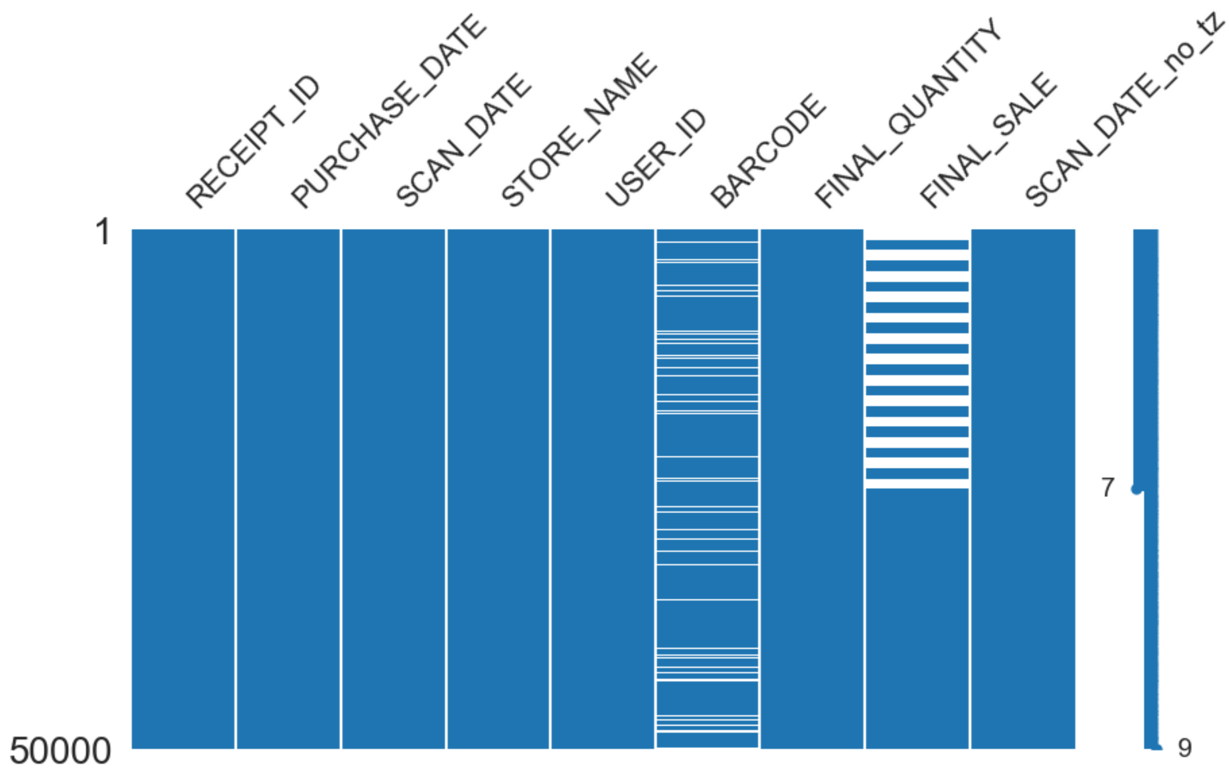


Let's look closer at the missing data.

```
# percent of missing data
print(trans_dat.isna().sum(axis=0) / rows)

# visualize how much data is missing
msno.matrix(trans_dat, color=(0.1215, 0.46, 0.70),figsize=(10, 5))
```

```
RECEIPT_ID          0.00000
PURCHASE_DATE       0.00000
SCAN_DATE           0.00000
STORE_NAME          0.00000
USER_ID             0.00000
BARCODE             0.05762
FINAL_QUANTITY      0.00000
FINAL_SALE          0.12500
SCAN_DATE_no_tz     0.00000
dtype: float64
```

```
<Axes: >
```



Every row has all columns present except BARCODE and FINAL_SALE have missing data. In summary for the transaction data:

1. Are there data quality issues present?

- STORE_NAME can be messy text. For example '\Mart' may not be an actual store name.
- BARCODE is not standardized. Some are -1, some are different lengths than others. There are also lots of missing rows.
- FINAL_QUANTITY has a value of "zero" which is a weird value. This column was fixed and the "zeros" imputed to be 0. They can also be decimals which may make sense for some items.
- FINAL_SALE has many blank rows.
- Finally there seems to be a pattern in the data for the FINAL_QUANTITY and FINAL_SALE columns. The first 25k rows of the dataset have alternating values of a blank FINAL_SALE and a "zero" FINAL_QUANTITY. Additionally there are 320 duplicate rows issue, which could be related. However this could make sense if 2 items on a receipt were identical but not included on the same line item. It depends on how this data is collected, ie if the OCR text extraction from the images is accurate or not. This point would need to be clarified with the team. Depending on the feedback, we would know if the "zero" value in FINAL_QUANTITY is a legitimate value or corrupted data. If that is the case then the imputation to "0" could be reversed.

1. Are there any fields that are challenging to understand?

- It is unclear what a BARCODE of -1 means. Assumption - could be a return or administrative scan.

- It is unclear what a FINAL_QUANTITY of "zero" means. This could be a special case like a return or a purchase with a coupon/discount, or a some type of failed transaction. Or it could be another corruption in the source of the data.
- SCAN_DATE and PURCHASE_DATE are not necessarily on the same date. This could be due to the time of day the transaction was made or the time it took to process the transaction. However, 92% of the scan dates are within a week of the purchase_date.

3. Product data

Get dimensions, data types, and # non-nulls.

```
# Get dimensions
prod_dat = pd.read_csv('PRODUCTS_.csv')

# 845k rows and 7 columns. All but 1 column has a generic object type.
All have some missing data.
print(prod_dat.info())

# view data
prod_dat.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 845552 entries, 0 to 845551
Data columns (total 7 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   CATEGORY_1    845441 non-null  object
 1   CATEGORY_2    844128 non-null  object
 2   CATEGORY_3    784986 non-null  object
 3   CATEGORY_4    67459 non-null   object
 4   MANUFACTURER  619078 non-null  object
 5   BRAND         619080 non-null  object
 6   BARCODE       841527 non-null  float64
dtypes: float64(1), object(6)
memory usage: 45.2+ MB
None

          CATEGORY_1              CATEGORY_2
CATEGORY_3  \
0  Health & Wellness           Sexual Health  Conductivity Gels &
Lotions
1             Snacks           Puffed Snacks         Cheese Curls &
Puffs
2  Health & Wellness               Hair Care        Hair Care
Accessories
3  Health & Wellness               Oral Care
Toothpaste
4  Health & Wellness  Medicines & Treatments               Essential
Oils
```

```
  CATEGORY_4                                          MANUFACTURER  \
0        NaN                                                   NaN
1        NaN                                                   NaN
2        NaN                              PLACEHOLDER MANUFACTURER
3        NaN                                       COLGATE-PALMOLIVE
4        NaN  MAPLE HOLISTICS AND HONEYDEW PRODUCTS INTERCHA...

              BRAND       BARCODE
0               NaN  7.964944e+11
1               NaN  2.327801e+10
2           ELECSOP  4.618178e+11
3           COLGATE  3.500047e+10
4   MAPLE HOLISTICS  8.068109e+11
```

Now that we know we have ~845k rows of 7 columns and they have intuitive types, let's dig into their contents. Above we can already see that all have at least some missing values. The category fields get progressively more empty. Check unique values.

```
# check how many unique values in each column
print(prod_dat.nunique())
```

```
CATEGORY_1          27
CATEGORY_2         121
CATEGORY_3         344
CATEGORY_4         127
MANUFACTURER      4354
BRAND             8122
BARCODE         841342
dtype: int64
```

No columns are completely unique. This means there are repeat BARCODEs, which should be our primay key in this table. List the most frequent distinct values for each column.

```
# get 5 most frequent distinct values from each column
for column in prod_dat.columns:
    print(f"Column: {column}")
    print(prod_dat[column].value_counts().nlargest(5))
    print("-" * 20)
```

```
Column: CATEGORY_1
CATEGORY_1
Health & Wellness          512695
Snacks                     324817
Beverages                    3990
Pantry                        871
Apparel & Accessories         846
Name: count, dtype: int64
--------------------
Column: CATEGORY_2
```

```
CATEGORY_2
Candy                          121036
Hair Care                      111482
Medicines & Treatments          99118
Bath & Body                     81469
Skin Care                       62587
Name: count, dtype: int64
--------------------
Column: CATEGORY_3
CATEGORY_3
Confection Candy                     56965
Vitamins & Herbal Supplements        55700
Chocolate Candy                      47710
Hair Styling Products                20450
Reading Glasses                      20394
Name: count, dtype: int64
--------------------
Column: CATEGORY_4
CATEGORY_4
Lip Balms                          9737
Already Popped Popcorn             6974
Sleep Aids                         4978
Hair Brushes & Combs               4724
Women's Shaving Gel & Cream        3874
Name: count, dtype: int64
--------------------
Column: MANUFACTURER
MANUFACTURER
PLACEHOLDER MANUFACTURER    86902
PROCTER & GAMBLE            21065
REM MANUFACTURER            20813
UNILEVER                    16864
L'OREAL                     16699
Name: count, dtype: int64
--------------------
Column: BRAND
BRAND
REM BRAND          20813
BRAND NOT KNOWN    17025
PRIVATE LABEL      13467
CVS                 6400
SEGO                4831
Name: count, dtype: int64
--------------------
Column: BARCODE
BARCODE
3423905.0     2
3416105.0     2
20146900.0    2
```

```
3454206.0     2
3462003.0     2
Name: count, dtype: int64
--------------------
```

Check for duplicates.

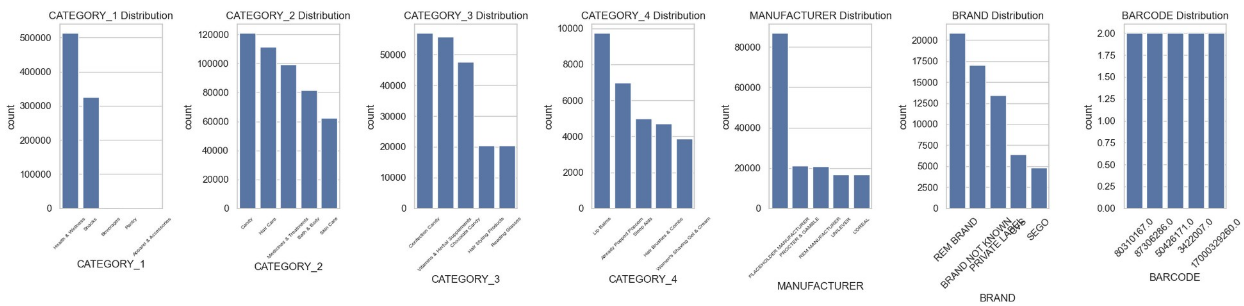
```
prod_dat_dups = 
prod_dat[prod_dat.duplicated(keep=False)].sort_values(by=['CATEGORY_1'
, 'CATEGORY_2', 'CATEGORY_3', 'CATEGORY_4', 'MANUFACTURER', 
'BRAND','BARCODE'])

# remove duplicates
prod_dat = prod_dat.drop_duplicates()

prod_dat_dups.head()

               CATEGORY_1              CATEGORY_2  \
183865            Alcohol                    Beer   
359328            Alcohol                    Beer   
695827          Beverages  Carbonated Soft Drinks   
817261          Beverages  Carbonated Soft Drinks   
82900   Health & Wellness  Medicines & Treatments   

                                    CATEGORY_3      CATEGORY_4  \
183865                                   Lager  American Lager   
359328                                   Lager  American Lager   
695827                                    Cola    Regular Cola   
817261                                    Cola    Regular Cola   
82900   Allergy & Sinus Medicines & Treatments             NaN   

                    MANUFACTURER        BRAND    BARCODE  
183865              MOLSONCOORS  COORS LIGHT        NaN  
359328              MOLSONCOORS  COORS LIGHT        NaN  
695827   THE COCA-COLA COMPANY    COCA-COLA  4904403.0  
817261   THE COCA-COLA COMPANY    COCA-COLA  4904403.0  
82900                    HALEON      FLONASE        NaN  
```

There are 400 duplicate rows. These can be removed since they're simply repeated lines and don't add to the dataset. Let's visualize this information with plots.

```
# plot the distinct values in the three string columns
fig, axes = plt.subplots(1, 7, figsize=(20, 5))

# plot distinct values for CATEGORY_1
order = prod_dat['CATEGORY_1'].value_counts().nlargest(5).index
sns.countplot(x = 'CATEGORY_1', data = 
prod_dat[prod_dat['CATEGORY_1'].isin(order)], order = order, 
ax=axes[0])
```

```
axes[0].set_title('CATEGORY_1 Distribution')
axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=45)
axes[0].tick_params(axis='x', labelsize=6)

# same for CATEGORY_2
order = prod_dat['CATEGORY_2'].value_counts().nlargest(5).index
sns.countplot(x = 'CATEGORY_2', data =
prod_dat[prod_dat['CATEGORY_2'].isin(order)], order = order,
ax=axes[1])
axes[1].set_title('CATEGORY_2 Distribution')
axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation=45)
axes[1].tick_params(axis='x', labelsize=6)

# same for CATEGORY_3
order = prod_dat['CATEGORY_3'].value_counts().nlargest(5).index
sns.countplot(x = 'CATEGORY_3', data =
prod_dat[prod_dat['CATEGORY_3'].isin(order)], order = order,
ax=axes[2])
axes[2].set_title('CATEGORY_3 Distribution')
axes[2].set_xticklabels(axes[2].get_xticklabels(), rotation=45)
axes[2].tick_params(axis='x', labelsize=6)

# same for CATEGORY_4
order = prod_dat['CATEGORY_4'].value_counts().nlargest(5).index
sns.countplot(x = 'CATEGORY_4', data =
prod_dat[prod_dat['CATEGORY_4'].isin(order)], order = order,
ax=axes[3])
axes[3].set_title('CATEGORY_4 Distribution')
axes[3].set_xticklabels(axes[3].get_xticklabels(), rotation=45)
axes[3].tick_params(axis='x', labelsize=6)

# same for MANUFACTURER
order = prod_dat['MANUFACTURER'].value_counts().nlargest(5).index
sns.countplot(x = 'MANUFACTURER', data =
prod_dat[prod_dat['MANUFACTURER'].isin(order)], order = order,
ax=axes[4])
axes[4].set_title('MANUFACTURER Distribution')
axes[4].set_xticklabels(axes[4].get_xticklabels(), rotation=45)
axes[4].tick_params(axis='x', labelsize=6)

# same for BRAND
order = prod_dat['BRAND'].value_counts().nlargest(5).index
sns.countplot(x = 'BRAND', data =
prod_dat[prod_dat['BRAND'].isin(order)], order = order, ax=axes[5])
axes[5].set_title('BRAND Distribution')
axes[5].set_xticklabels(axes[5].get_xticklabels(), rotation=45)

# same for BARCODE
order = prod_dat['BARCODE'].value_counts().nlargest(5).index
sns.countplot(x = 'BARCODE', data =
```

```
prod_dat[prod_dat['BARCODE'].isin(order)], order = order, ax=axes[6])
axes[6].set_title('BARCODE Distribution')
axes[6].set_xticklabels(axes[6].get_xticklabels(), rotation=45)

plt.tight_layout()
plt.show()
```



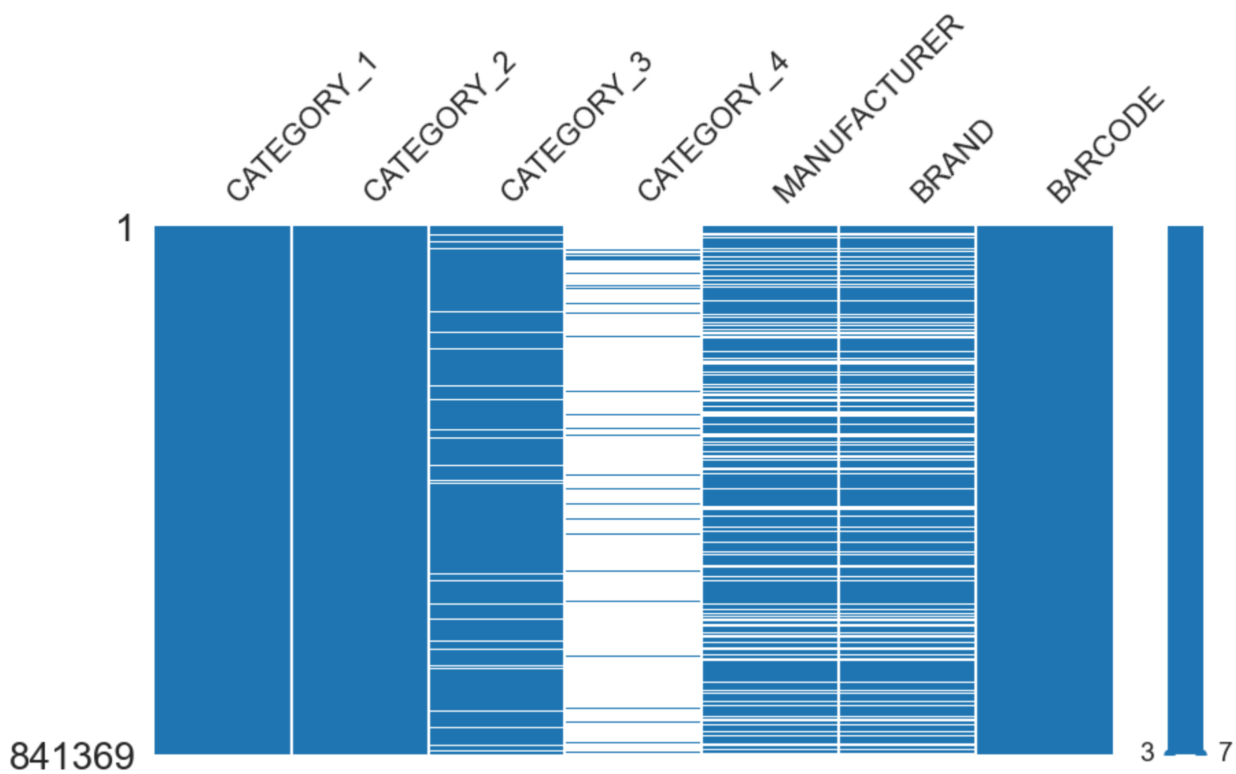
From these plots we can summarize these columns. Category 1 is the most general category and dominated by Health & Wellness and Snacks. The columns have inconsistent capitalization. Barcodes can have 2 rows in the data. Now remove rows with missing barcodes as they will not be useful for the analysis (they cannot be joined into transactions).

```
# remove rows with NA barcode
prod_dat.dropna(subset=['BARCODE'], inplace=True)

prod_dat.info()

<class 'pandas.core.frame.DataFrame'>
Index: 841369 entries, 0 to 845551
Data columns (total 7 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   CATEGORY_1    841258 non-null  object
 1   CATEGORY_2    840708 non-null  object
 2   CATEGORY_3    782655 non-null  object
 3   CATEGORY_4    67234 non-null   object
 4   MANUFACTURER  615152 non-null  object
 5   BRAND         615154 non-null  object
 6   BARCODE       841369 non-null  float64
dtypes: float64(1), object(6)
memory usage: 51.4+ MB
```

Now let's look closer at the missing data.

```
# percent of missing data
print(prod_dat.isna().sum(axis=0) / rows)

# visualize how much data is missing
msno.matrix(prod_dat, color=(0.1215, 0.46, 0.70),figsize=(10, 5))
```

```
CATEGORY_1      0.00111
CATEGORY_2      0.00661
CATEGORY_3      0.58714
CATEGORY_4      7.74135
MANUFACTURER    2.26217
BRAND           2.26215
BARCODE         0.00000
dtype: float64

<Axes: >
```



Now we have a barcode for each row. Categories have sparse data the more specific they become. MANUFACTURER and BRAND are also not at full coverage. Lastly, check on rows that have the same BARCODE but differ in other ways.

```
# barcodes can appear at most twice
print(prod_dat['BARCODE'].value_counts().nlargest(5))

# filter prod_dat for barcodes that appear more than once
dup_barcodes =
prod_dat[prod_dat['BARCODE'].duplicated(keep=False)].sort_values(by='B
ARCODE')

# 27 distinct barcodes are duplicated
print(f"Number of duplicated barcodes:
```

```
{len(dup_barcodes['BARCODE'].unique())}")

# now only look at the ones that are also in the trans_dat
prod_dat[prod_dat['BARCODE'].duplicated(keep=False) &
prod_dat['BARCODE'].isin(trans_dat['BARCODE'])]

# # filter dup_barcodes for barcodes in prod_dat
dup_barcodes2 =
dup_barcodes[dup_barcodes['BARCODE'].isin(trans_dat['BARCODE'])]

print(dup_barcodes2.head())

# drop row for the duplicate we need by index
prod_dat.drop(index=137250, inplace=True)
```

```
BARCODE
8.031017e+07    2
8.730629e+07    2
5.042617e+07    2
3.422007e+06    2
1.700033e+10    2
Name: count, dtype: int64
Number of duplicated barcodes: 27
       CATEGORY_1    CATEGORY_2       CATEGORY_3 CATEGORY_4
MANUFACTURER  \
193347     Snacks         Candy  Chocolate Candy        NaN  MARS
WRIGLEY
137250     Snacks  Nuts & Seeds          Peanuts        NaN  MARS
WRIGLEY

        BRAND    BARCODE
193347  M&M'S  4003207.0
137250  M&M'S  4003207.0
```

From above, I see that there are 27 duplicated BARCODES (there could have been more originally, but this is already a filtered dataset). The barcodes tend to be mostly the same in terms of the categories, but with minor changes. Others barcodes have a duplicate that is uncomplete, or appears to belong to an older brand. The only one we care about is the one that will merge into the transactions data, which codes for M&M's (4003207). For this analysis, I'm going to assume that the first row, classifying the product into Candy-Chocolate Candy is more appropriate than Nuts & Seeds-Peanuts. In summary:

1. Are there data quality issues present?

- BARCODE has NA values (Every other column also had blank values present). Rows with NA BARCODE were removed since the rows cannot be used in joining to transactions.
- MANUFACTURER is in all caps. Other than that all the columns including the category columns are quite clean, and even have correct capitalization.
- There were duplicate rows. These were removed. Also - there are duplicates where one row uses the placeholder manufacturer and the other row has the actual manufacturer.

1. Are there any fields that are challenging to understand?

- All fields are straightforward. A barcode mapping to 2 products with different categories is sometimes unintuitive but for our purposes will not be a problem.