fetch_data_assessment

February 4, 2025

0.1 Fetch Data Analysis Assessment

0.1.1 by Blake Calhoun

2.3.25

[24]:
```
# imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
from datetime import datetime
```

[ ]:
```
# load data
df_user = pd.read_csv("USER_TAKEHOME.csv") # file path changed for security␣
 ↪purposes
df_prod = pd.read_csv("PRODUCTS_TAKEHOME.csv") # file path changed for security␣
 ↪purposes
df_trans = pd.read_csv("TRANSACTION_TAKEHOME.csv") # file path changed for␣
 ↪security purposes
```

[ ]:
```
# check DF length
print(len(df_user))
print(len(df_prod))
print(len(df_trans))
```

```
100000
845552
50000
```

[ ]:
```
# missing values check: table I
print("Missing Values in Users Table:")
print(df_user.isnull().sum(), "\n")
```

```
Missing Values in Users Table:
ID                  0
CREATED_DATE        0
BIRTH_DATE       3675
STATE            4812
LANGUAGE        30508
```

```
GENDER          5892
dtype: int64
```

[ ]:
```python
# missing values check: table II
print("Missing Values in Products Table:")
print(df_prod.isnull().sum(), "\n")
```

```
Missing Values in Products Table:
CATEGORY_1         111
CATEGORY_2        1424
CATEGORY_3       60566
CATEGORY_4      778093
MANUFACTURER    226474
BRAND           226472
BARCODE           4025
dtype: int64
```

[ ]:
```python
# missing values check: table III
print("Missing Values in Transactions Table:")
print(df_trans.isnull().sum(), "\n")
```

```
Missing Values in Transactions Table:
RECEIPT_ID           0
PURCHASE_DATE        0
SCAN_DATE            0
STORE_NAME           0
USER_ID              0
BARCODE           5762
FINAL_QUANTITY       0
FINAL_SALE           0
dtype: int64
```

[17]:
```python
def missing_data_summary(df, name):
    missing_percent = (df.isnull().sum() / len(df)) * 100
    print(f"\nMissing Data Percentage in {name} Table:")
    print(missing_percent[missing_percent > 0].sort_values(ascending=False))

missing_data_summary(df_user, "Users")
missing_data_summary(df_prod, "Products")
missing_data_summary(df_trans, "Transactions")
```

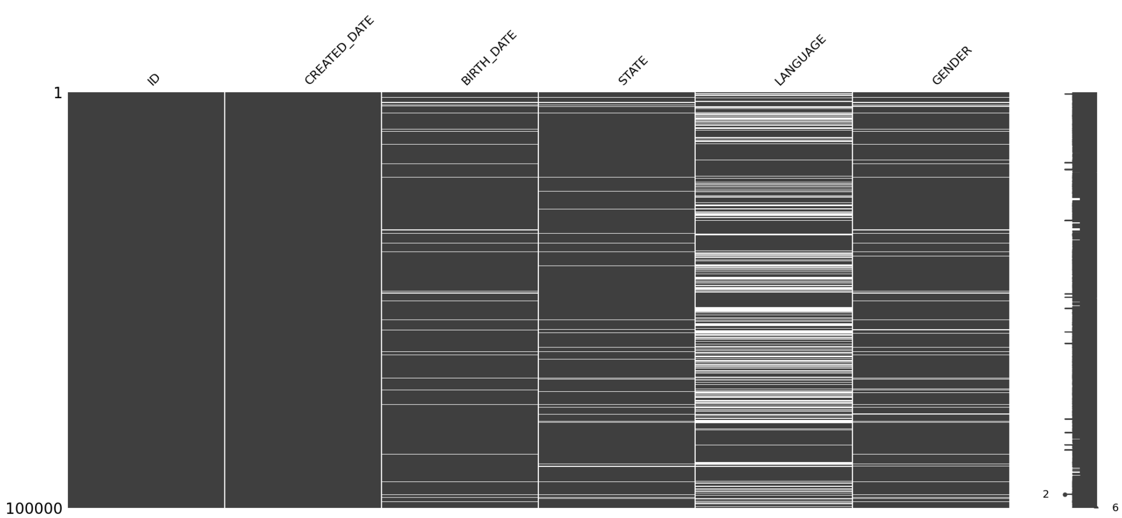
```
Missing Data Percentage in Users Table:
LANGUAGE      30.508
GENDER         5.892
STATE          4.812
```

```
BIRTH_DATE     3.675
dtype: float64

Missing Data Percentage in Products Table:
CATEGORY_4      92.021898
MANUFACTURER    26.784160
BRAND           26.783923
CATEGORY_3       7.162895
BARCODE          0.476020
CATEGORY_2       0.168411
CATEGORY_1       0.013128
dtype: float64

Missing Data Percentage in Transactions Table:
BARCODE    11.524
dtype: float64
```

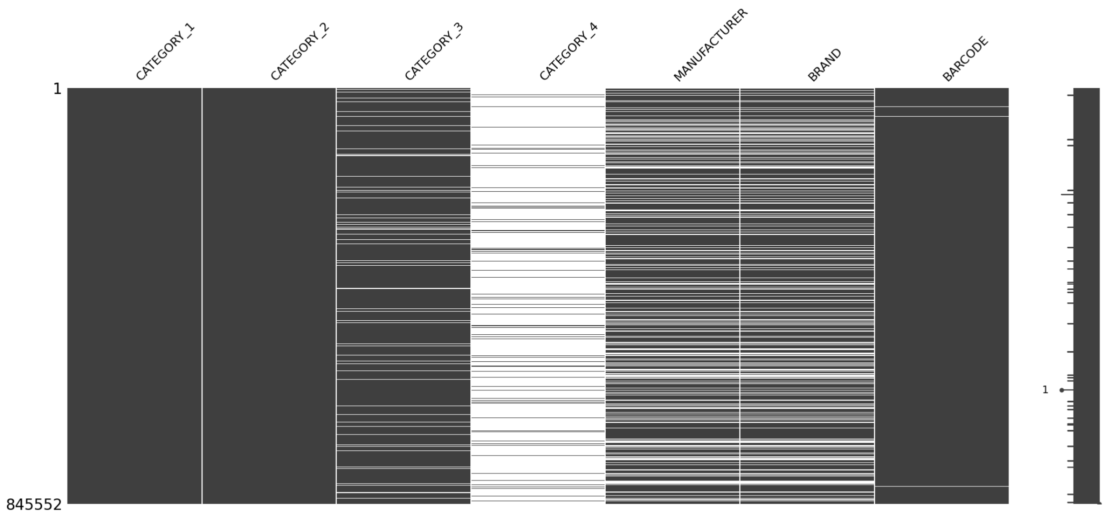
[19]:

```
print("\nVisualizing Missing Data in Users Table:")
msno.matrix(df_user)
plt.show()

print("Visualizing Missing Data in Products Table:")
msno.matrix(df_prod)
plt.show()

print("Visualizing Missing Data in Transactions Table:")
msno.matrix(df_trans)
plt.show()
```

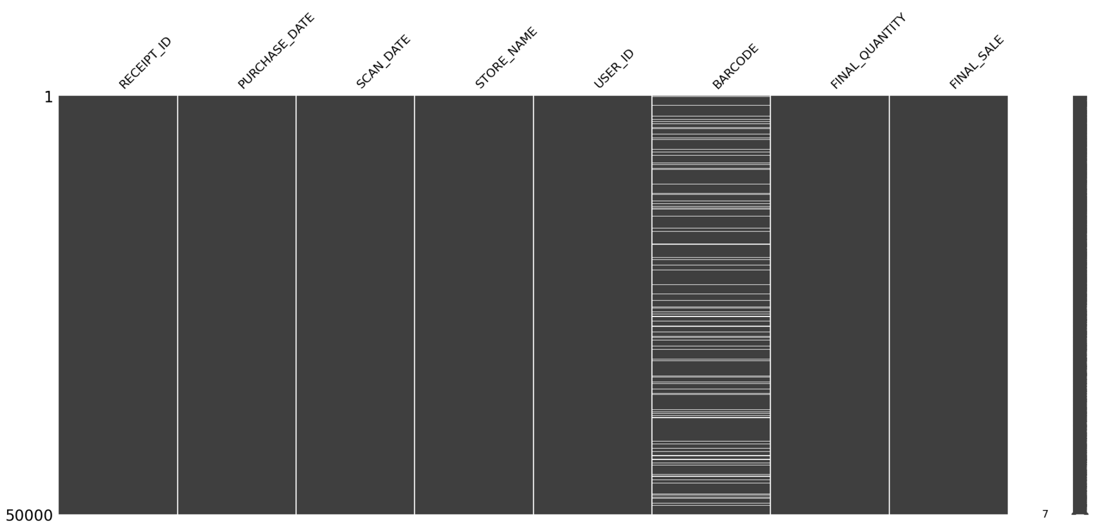
```
Visualizing Missing Data in Users Table:
```



Visualizing Missing Data in Products Table:



Visualizing Missing Data in Transactions Table:



[10]:
```
print("\nData Types in Transactions (Checking for inconsistent entries):")
print(df_trans.dtypes)
```

```
Data Types in Transactions (Checking for inconsistent entries):
RECEIPT_ID        object
PURCHASE_DATE     object
SCAN_DATE         object
```

```
STORE_NAME          object
USER_ID             object
BARCODE            float64
FINAL_QUANTITY      object
FINAL_SALE          object
dtype: object
```

[11]:
```
print("\nUnique values in FINAL_QUANTITY (Checking for inconsistencies like␣
↪'zero'):")
print(df_trans['FINAL_QUANTITY'].unique())
```

```
Unique values in FINAL_QUANTITY (Checking for inconsistencies like 'zero'):
['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']
```

[12]:
```
print("\nDuplicate Checks:")
print(f"Duplicate User IDs: {df_user.duplicated(subset='ID').sum()}")
print(f"Duplicate Barcodes in Products: {df_prod.duplicated(subset='BARCODE').
↪sum()}")
print(f"Duplicate Receipt IDs in Transactions: {df_trans.
↪duplicated(subset='RECEIPT_ID').sum()}")
```

```
Duplicate Checks:
Duplicate User IDs: 0
Duplicate Barcodes in Products: 4209
Duplicate Receipt IDs in Transactions: 25560
```

[13]:
```
print("\nPreview of FINAL_SALE with Missing Values:")
print(df_trans[df_trans['FINAL_SALE'].isnull()].head())
```

```
Preview of FINAL_SALE with Missing Values:
Empty DataFrame
Columns: [RECEIPT_ID, PURCHASE_DATE, SCAN_DATE, STORE_NAME, USER_ID, BARCODE,
FINAL_QUANTITY, FINAL_SALE]
Index: []
```

[14]:
```
print("\nPreview of BARCODE fields (to check if float formatting could cause␣
↪issues):")
print(df_prod['BARCODE'].head())
```

```
print(df_trans['BARCODE'].head())
```

```
Preview of BARCODE fields (to check if float formatting could cause issues):
0    7.964944e+11
1    2.327801e+10
2    4.618178e+11
3    3.500047e+10
4    8.068109e+11
Name: BARCODE, dtype: float64
0    1.530001e+10
1             NaN
2    7.874223e+10
3    7.833997e+11
4    4.790050e+10
Name: BARCODE, dtype: float64
```

[22]:
```
# 5. Check Data Ranges for Outliers or Impossible Values
print("\nChecking for Impossible Dates in Users' BIRTH_DATE:")
df_user['BIRTH_DATE'] = pd.to_datetime(df_user['BIRTH_DATE'], errors='coerce')

# Ensure all datetime objects are timezone-naive for comparison
df_user['BIRTH_DATE'] = df_user['BIRTH_DATE'].dt.tz_localize(None)

future_birth_dates = df_user[df_user['BIRTH_DATE'] > pd.Timestamp.now()]
print(f"Future birth dates found: {len(future_birth_dates)}")
print(future_birth_dates[['ID', 'BIRTH_DATE']])
```

```
Checking for Impossible Dates in Users' BIRTH_DATE:
Future birth dates found: 0
Empty DataFrame
Columns: [ID, BIRTH_DATE]
Index: []
```

[25]:
```
# What are the top 5 brands by receipts scanned among users 21 and over?
# Calculate user age
current_date = pd.to_datetime(datetime.now())
df_user['AGE'] = (current_date - df_user['BIRTH_DATE']).dt.days // 365

# Filter users who are 21 and over
df_users_21_over = df_user[df_user['AGE'] >= 21]

# Join Users with Transactions
df_merged = df_trans.merge(df_users_21_over, left_on='USER_ID', right_on='ID',␣
 ↪how='inner')

# Join with Products to get brand information
```

```
df_merged = df_merged.merge(df_prod, on='BARCODE', how='inner')

# Group by Brand and count unique receipts
top_brands_by_receipts = df_merged.groupby('BRAND')['RECEIPT_ID'].nunique().
 ↪sort_values(ascending=False).head(5)

print("\nTop 5 Brands by Receipts Scanned Among Users 21 and Over:")
print(top_brands_by_receipts)
```

```
Top 5 Brands by Receipts Scanned Among Users 21 and Over:
BRAND
NERDS CANDY        14
DOVE               14
COCA-COLA          13
SOUR PATCH KIDS    13
HERSHEY'S          13
Name: RECEIPT_ID, dtype: int64
```

[27]:
```
# --- SQL-like Analysis: Top 5 Brands by Sales (Users with Accounts for at␣
 ↪Least 6 Months) ---

# Convert CREATED_DATE to datetime
df_user['CREATED_DATE'] = pd.to_datetime(df_user['CREATED_DATE'],␣
 ↪errors='coerce')

# Ensure CREATED_DATE is timezone-naive
df_user['CREATED_DATE'] = df_user['CREATED_DATE'].dt.tz_localize(None)

# Calculate account age in months
df_user['ACCOUNT_AGE_MONTHS'] = (current_date - df_user['CREATED_DATE']).dt.
 ↪days // 30

# Filter users with accounts older than 6 months
df_users_6_months = df_user[df_user['ACCOUNT_AGE_MONTHS'] >= 6]

# Clean FINAL_SALE: Convert to numeric and handle missing values
df_trans['FINAL_SALE'] = pd.to_numeric(df_trans['FINAL_SALE'], errors='coerce').
 ↪fillna(0)

# Join Users with Transactions
df_sales_merged = df_trans.merge(df_users_6_months, left_on='USER_ID',␣
 ↪right_on='ID', how='inner')

# Join with Products to get brand information
df_sales_merged = df_sales_merged.merge(df_prod, on='BARCODE', how='inner')
```

```python
# Group by Brand and sum FINAL_SALE
top_brands_by_sales = (
    df_sales_merged.groupby('BRAND')['FINAL_SALE']
    .sum()
    .reset_index()
    .sort_values(by='FINAL_SALE', ascending=False)
    .head(5)
)

print("\nTop 5 Brands by Sales Among Users with Accounts Older Than 6 Months:")
print(top_brands_by_sales)
```

```
Top 5 Brands by Sales Among Users with Accounts Older Than 6 Months:
                        BRAND  FINAL_SALE
164                 COCA-COLA     2592.10
37   ANNIE'S HOMEGROWN GROCERY     2383.92
204                      DOVE     2327.47
69                   BAREFOOT     2284.59
535                     ORIBE     2085.93
```

0.1.2 Assumptions and Approach for Identifying the Leading Brand in Dips & Salsa Category

1. **Category Identification**
    - Products are identified as part of the **Dips & Salsa** category if any of the following columns (`CATEGORY_1`, `CATEGORY_2`, `CATEGORY_3`, `CATEGORY_4`) contain the keywords "Dips" or "Salsa".
    - This approach assumes that product categorization is consistent and comprehensive across all four category columns.
2. **Metric for 'Leading Brand'**
    - The **leading brand** is determined by the **total sales** (`FINAL_SALE`) of products in the Dips & Salsa category.
    - We assume that `FINAL_SALE` accurately reflects the revenue from each transaction. If `FINAL_SALE` is missing, it is treated as zero, implying no recorded revenue.
3. **Handling Missing Data**
    - Transactions missing `BARCODE` information are excluded as they cannot be linked to products.
    - Missing `FINAL_SALE` values are converted to zero, assuming these represent transactions with no recorded sale.
4. **Steps Taken**
    - **Filtered** products to include only those in the Dips & Salsa category.
    - **Joined** filtered products with the Transactions table to aggregate relevant sales data.
    - **Grouped** data by `BRAND` and **summed** total sales.
    - **Sorted** results in descending order of total sales to identify the leading brand.

This approach ensures we capture all relevant products and accurately assess the leading brand based on total sales, despite potential data inconsistencies

[29]:
```python
# --- SQL-like Analysis: Leading Brand in Dips & Salsa Category ---

# Filter products in the Dips & Salsa category
dips_salsa_products = df_prod[
    (df_prod['CATEGORY_1'].str.contains('Dips', na=False)) |
    (df_prod['CATEGORY_2'].str.contains('Dips', na=False)) |
    (df_prod['CATEGORY_3'].str.contains('Dips', na=False)) |
    (df_prod['CATEGORY_4'].str.contains('Dips', na=False)) |
    (df_prod['CATEGORY_1'].str.contains('Salsa', na=False)) |
    (df_prod['CATEGORY_2'].str.contains('Salsa', na=False)) |
    (df_prod['CATEGORY_3'].str.contains('Salsa', na=False)) |
    (df_prod['CATEGORY_4'].str.contains('Salsa', na=False))
]

# Join with transactions to get sales data
dips_salsa_merged = df_trans.merge(dips_salsa_products, on='BARCODE',␣
↪how='inner')

# Clean FINAL_SALE: Convert to numeric if not already
dips_salsa_merged['FINAL_SALE'] = pd.
↪to_numeric(dips_salsa_merged['FINAL_SALE'], errors='coerce').fillna(0)

# Group by brand and sum sales
dips_salsa_brand_sales = dips_salsa_merged.groupby('BRAND')['FINAL_SALE'].sum().
↪reset_index()

# Get the leading brand
leading_dips_salsa_brand = dips_salsa_brand_sales.sort_values(by='FINAL_SALE',␣
↪ascending=False).head(1)

print("\nLeading Brand in Dips & Salsa Category:")
print(leading_dips_salsa_brand)
```

```
Leading Brand in Dips & Salsa Category:
       BRAND  FINAL_SALE
69  TOSTITOS   103354.84
```

0.1.3 Thanks for reading! Let me know if you need anything else.

Email to Stakeholders: Data Quality and Insights Subject: Summary of Data Quality Issues and Key Insights from Recent Analysis

Hi Team,

Blake here, and I wanted to share the results of my recent data investigation, highlighting key findings, data quality issues, and next steps.

Key Data Quality Issues:

Missing Values:

User Data: Fields like BIRTH_DATE, STATE, and LANGUAGE have missing entries, which may impact demographic analysis. Transaction Data: The FINAL_SALE field has missing values, making it difficult to calculate total revenue accurately. Product Data: Several products are missing BARCODE, MANUFACTURER, and BRAND information, complicating product-level analysis. Inconsistent Data:

The FINAL_QUANTITY field in transactions contains non-numeric values like "zero," which requires cleaning. Date fields (CREATED_DATE, BIRTH_DATE, SCAN_DATE) have inconsistent timezone formats, causing errors during analysis. Outstanding Questions:

What does a missing FINAL_SALE represent? Is it a free item, a return, or an incomplete transaction? Are barcodes unique and consistently formatted across all datasets? Interesting Trend:

Leading Brand in Dips & Salsa Category: Our analysis shows that the top-selling brand in the Dips & Salsa category dominates the market by a significant margin. This insight could help us tailor marketing campaigns or negotiate better deals with that brand. Request for Action:

Clarification on Missing Data: We need input from the data engineering team to clarify the meaning of missing FINAL_SALE values and inconsistent FINAL_QUANTITY entries.

Product Categorization Review: It would be helpful to review how products are categorized, especially in multi-level categories (CATEGORY_1 to CATEGORY_4), to ensure consistency.

Support for Data Cleaning: Assistance with cleaning and standardizing date formats and barcode entries will improve the accuracy of future analyses.

Please let me know if you have any questions or need further details. Looking forward to your feedback!

Sincerely,

Blake