# Fetch - Data Analyst Take Home

**Section I: Explore the Data**  
After exploring the data these are the Initial Data Quality & Exploration Findings that I came across with each Dataset.

1. **Products Data (PRODUCTS\_TAKEHOME.csv)**
   * + **Data Quality Issues:**

* **Missing Data:**
  + - * + `CATEGORY\_1`, `CATEGORY\_2`, `CATEGORY\_3`, and especially `CATEGORY\_4` have significant missing values.
        + `MANUFACTURER` and `BRAND` are missing in nearly 27% of rows.
        + `BARCODE` is missing for ~4,000 products (out of ~845,000 products).
      * **Potential Data Consistency Issues:**
        + `MANUFACTURER` includes a large proportion labeled as "PLACEHOLDER MANUFACTURER", which is either a dummy placeholder or indicates poor data capture.
        + The `CATEGORY` columns suggest some hierarchical structure (1 → 2 → 3 → 4), but the large number of missing `CATEGORY\_3` and `CATEGORY\_4` values breaks that hierarchy.
    - **Fields Challenging to Understand:**
      * `CATEGORY\_3` and `CATEGORY\_4` are difficult to interpret without clear documentation on how these categories roll up into each other.
      * `BARCODE` has outliers, including very large and small barcodes. It’s unclear if these are valid.
* **Assumptions:**
* Missing categories might indicate incomplete product catalog integration.
* Missing `BARCODE` may indicate non-scanned products.
* **Code:**

# Import libraries

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Load Products Data

products = pd.read\_csv('PRODUCTS\_TAKEHOME.csv')

# Function to check missing data

def missing\_summary(df, name):

print(f"Missing data for {name}:\n{df.isnull().sum()}\n")

# Products Data Exploration

missing\_summary(products, "Products")

# Check top manufacturers

print("Top 10 Manufacturers:\n", products['MANUFACTURER'].value\_counts().head(10))

# Check top categories

print("Top Categories:\n", products['CATEGORY\_1'].value\_counts())

# Barcode Distribution - Outlier Check

plt.figure(figsize=(10, 5))

sns.boxplot(x=products['BARCODE'])

plt.title("Product Barcode Distribution (Outlier Check)")

plt.show()

A white background with black border

Description automatically generated

1. **Transactions Data (TRANSACTION\_TAKEHOME.csv)**

* **Data Quality Issues:**
* **Missing Data:**
  + - * + `BARCODE` is missing for about 11.5% of transactions (5,762 out of 50,000).
* **Field Format Issues:**
  + - * + `FINAL\_QUANTITY` includes "zero", which is likely meant to be `0`, indicating data type coercion (mix of numeric and text data).
        + `FINAL\_SALE` includes missing values (empty strings) — this may indicate either refunds, zero-cost items, or parsing issues.
* **Fields Challenging to Understand:**
* `FINAL\_QUANTITY`: Mixing numeric values with the string `"zero"` could cause analysis errors.
* `FINAL\_SALE`: Empty entries are ambiguous — is it a zero-value sale, or is data missing?
* **Assumptions:**
* `SCAN\_DATE` includes a timestamp (millisecond precision), which could be useful for tracking time between purchase and scan events.
* Missing `BARCODE` transactions could involve store-brand or non-scannable items (like services or produce).
* **Code:**

# Import libraries

import pandas as pd

# Load Transactions Data

transactions = pd.read\_csv('TRANSACTION\_TAKEHOME.csv')

# Function to check missing data

def missing\_summary(df, name):

print(f"Missing data for {name}:\n{df.isnull().sum()}\n")

# Transactions Data Exploration

missing\_summary(transactions, "Transactions")

# Check final quantity counts (including data cleaning need)

print("Final Quantity Counts:\n", transactions['FINAL\_QUANTITY'].value\_counts())

# Review top Final Sale values

print("Top Final Sale Values:\n", transactions['FINAL\_SALE'].value\_counts().head(10))

1. **Users Data (USER\_TAKEHOME.csv)**

* **Data Quality Issues:**
* **Missing Data:**
  + - * + `BIRTH\_DATE`, `STATE`, `GENDER`, and especially `LANGUAGE` have significant missing values.
        + `LANGUAGE` is missing in over 30% of rows, which is particularly high.
        + `BIRTH\_DATE`, `STATE`, and `GENDER` are missing in approximately 3.7%, 4.8%, and 5.9% of rows, respectively.
* **Potential Data Consistency Issues:**
  + - * + `BIRTH\_DATE` includes 1,272 entries for "1970-01-01" — this is likely a placeholder for unknown birthdates.
        + `LANGUAGE` includes only two values: `en` and `es-419` (Latin American Spanish). Is this expected for all users?
* **Fields Challenging to Understand:**
* `LANGUAGE`: If the user base is entirely US-based, this field could be more expansive (more language options).
* `GENDER`: There are 11 unique values, which could indicate non-standard or free-text gender input (beyond male/female).
* **Assumptions:**
* `1970-01-01` is treated as a placeholder date (common in legacy systems).
* Users could have signed up without specifying language or gender, hence the gaps.
* **Code:**

# Import libraries

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Load Users Data

users = pd.read\_csv('USER\_TAKEHOME.csv')

# Function to check missing data

def missing\_summary(df, name):

print(f"Missing data for {name}:\n{df.isnull().sum()}\n")

# Users Data Exploration

missing\_summary(users, "Users")

# Check most common birth dates (check for placeholder values like 1970-01-01)

print("Most Common Birth Dates:\n", users['BIRTH\_DATE'].value\_counts().head(10))

# Check unusual gender values

print("Gender Distribution:\n", users['GENDER'].value\_counts())

# Check language distribution

print("Language Distribution:\n", users['LANGUAGE'].value\_counts())

# Analyze birth year distribution for potential age segmentation

users['BIRTH\_YEAR'] = pd.to\_datetime(users['BIRTH\_DATE'], errors='coerce').dt.year

plt.figure(figsize=(12, 6))

sns.histplot(users['BIRTH\_YEAR'].dropna(), bins=30, kde=True)

plt.title("User Birth Year Distribution")

plt.show()

A graph of a graph

Description automatically generated with medium confidence

**Findings Summary:**

| **Category** | **Data Issue** | **Severity** |
| --- | --- | --- |
| **Products** | Missing categories & manufacturer data | ⚠️ High |
| **Products** | Placeholder manufacturers | ⚠️ High |
| **Transactions** | Mixed type in FINAL\_QUANTITY | ⚠️ High |
| **Transactions** | Missing barcodes | ⚠️ Medium |
| **Users** | Missing language & gender | ⚠️ High |
| **Users** | Placeholder birth dates (1970-01-01) | ⚠️ Medium |
| **Users** | Unclear gender values | ⚠️ Medium |

**Data Cleaning:**

Data cleaning is an essential first step when working with raw datasets, especially when they originate from multiple sources, such as product inventories, transactional records, and user profiles. The primary objectives of data cleaning are to Ensure Data Consistency, Handle Missing Data, Standardize Formats, Improve Data Quality for Business Insights and Support Accurate Data Joining.

1. **Products Data:**

* **Objective:**
* Ensuring all essential fields (like barcodes) are present.
* Standardizing inconsistent fields (like manufacturers and brands).
* Flagging outliers that could indicate data entry errors.
* Filling missing data where appropriate.
* **Assumptions:**
* Products missing barcodes are unusable since they cannot be linked to transactions.
* Placeholder and missing manufacturers are untrustworthy and will be normalized to "Unknown Manufacturer."
* Long barcodes (more than 15 digits) are potential outliers based on standard EAN/UPC conventions.
* **Code:**

import pandas as pd

# Load data

products = pd.read\_csv('PRODUCTS\_TAKEHOME.csv')

# Make a working copy to preserve the original

products\_clean = products.copy()

# Fill missing categories with 'Unknown' - these are optional but help with categorization completeness

products\_clean['CATEGORY\_3'] = products\_clean['CATEGORY\_3'].fillna('Unknown')

products\_clean['CATEGORY\_4'] = products\_clean['CATEGORY\_4'].fillna('Unknown')

# Standardize missing manufacturer and brand - for consistency in grouping and filtering

products\_clean['MANUFACTURER'] = products\_clean['MANUFACTURER'].fillna('Unknown Manufacturer')

products\_clean['BRAND'] = products\_clean['BRAND'].fillna('Unknown Brand')

# Replace 'PLACEHOLDER MANUFACTURER' with 'Unknown Manufacturer' to remove placeholder noise

products\_clean['MANUFACTURER'] = products\_clean['MANUFACTURER'].replace('PLACEHOLDER MANUFACTURER', 'Unknown Manufacturer')

# Drop products with missing barcodes - these cannot be linked to transactions

products\_clean = products\_clean.dropna(subset=['BARCODE'])

# Identify barcode length to check for unrealistic (too long) barcodes

products\_clean['BARCODE\_LENGTH'] = products\_clean['BARCODE'].astype(str).apply(len)

# Flag products with barcodes longer than 15 digits as potential outliers

products\_clean['BARCODE\_FLAG'] = products\_clean['BARCODE\_LENGTH'].apply(lambda x: 'OUTLIER' if x > 15 else 'OK')

# Drop the temporary BARCODE\_LENGTH column - no longer needed

products\_clean = products\_clean.drop(columns=['BARCODE\_LENGTH'])

# Summary statistics

original\_count = len(products)

cleaned\_count = len(products\_clean)

placeholder\_count = (products['MANUFACTURER'] == 'PLACEHOLDER MANUFACTURER').sum()

unknown\_manufacturer\_count = (products\_clean['MANUFACTURER'] == 'Unknown Manufacturer').sum()

barcode\_outlier\_count = (products\_clean['BARCODE\_FLAG'] == 'OUTLIER').sum()

# Create a cleanup summary dictionary

cleanup\_summary = {

'Original Product Count': original\_count,

'After Cleanup Product Count': cleaned\_count,

'Dropped Products (Missing Barcode)': original\_count - cleaned\_count,

'Replaced PLACEHOLDER MANUFACTURER': placeholder\_count,

'Total Unknown Manufacturers': unknown\_manufacturer\_count,

'Flagged Barcode Outliers': barcode\_outlier\_count

}

# Display the summary

print("Products Cleanup Summary:")

for k, v in cleanup\_summary.items():

print(f"{k}: {v}")

* **Conclusions:**
* 4,025 products (0.48%) were removed due to missing barcodes, which is acceptable given they are unusable in linking with transactions.
* 86,902 products originally had 'PLACEHOLDER MANUFACTURER', all of which were converted to 'Unknown Manufacturer'.
* 313,129 products now have 'Unknown Manufacturer', either from placeholders or missing data.
* 44 barcodes (0.005%) were flagged as outliers for being longer than 15 digits. These could be data entry errors.
* There’s a data quality concern around the high percentage of unknown manufacturers, which may limit manufacturer-level insights.
* Outlier barcodes should be further reviewed with the business team to determine if they are genuine product codes.

1. **Transactions Data:**

* **Objective:**
* Cleaning up quantity and sale fields (which have mixed types and potential errors).
* Addressing missing barcodes, which could impact joinability with the products table.
* Ensuring all key fields are in the correct format.
* Flagging any rows with unusual or missing information.
* **Assumptions:**
* Transactions with missing barcodes cannot be reliably linked to products and may need to be either excluded or categorized as "unscanned items."
* `FINAL\_QUANTITY` mixing "zero" (string) with numeric values is assumed to be a data quality issue, will be standardized to numeric `0`.
* Empty `FINAL\_SALE` values could indicate refunds, missing prices, or parsing errors, these will be reviewed and potentially flagged.
* Dates in `PURCHASE\_DATE` and `SCAN\_DATE` are expected to be ISO format (YYYY-MM-DD).
* **Code:**

import pandas as pd

# Load transactions dataset

transactions = pd.read\_csv('TRANSACTION\_TAKEHOME.csv')

# Make a working copy to preserve the original

transactions\_clean = transactions.copy()

# Convert FINAL\_QUANTITY to numeric, replacing 'zero' with 0

transactions\_clean['FINAL\_QUANTITY'] = transactions\_clean['FINAL\_QUANTITY'].replace('zero', 0).astype(float)

# Identify transactions with missing barcodes - these may be non-scanned items (like produce or services)

transactions\_clean['MISSING\_BARCODE'] = transactions\_clean['BARCODE'].isnull()

# Create a flag for transactions with missing or blank FINAL\_SALE

transactions\_clean['FINAL\_SALE\_FLAG'] = transactions\_clean['FINAL\_SALE'].apply(lambda x: 'MISSING' if pd.isnull(x) or x.strip() == '' else 'OK')

# Convert dates to datetime format to enable proper analysis

transactions\_clean['PURCHASE\_DATE'] = pd.to\_datetime(transactions\_clean['PURCHASE\_DATE'], errors='coerce')

transactions\_clean['SCAN\_DATE'] = pd.to\_datetime(transactions\_clean['SCAN\_DATE'], errors='coerce')

# Summary Statistics

original\_count = len(transactions)

missing\_barcode\_count = transactions\_clean['MISSING\_BARCODE'].sum()

missing\_sale\_count = (transactions\_clean['FINAL\_SALE\_FLAG'] == 'MISSING').sum()

# Create a cleanup summary dictionary

cleanup\_summary = {

'Original Transaction Count': original\_count,

'Transactions with Missing Barcodes': missing\_barcode\_count,

'Transactions with Missing or Blank Final Sale': missing\_sale\_count,

}

# Display the summary

print("Transactions Cleanup Summary:")

for k, v in cleanup\_summary.items():

print(f"{k}: {v}")

* **Conclusions:**
* 5,762 transactions were identified with missing barcodes, rendering them unusable for linking with product data.
* A significant portion of transactions had missing or blank `Final Sale` values, this will be summarized after further analysis.
* A total of 50,000 transactions were originally recorded in the dataset.
* Transactions data contains a notable proportion of unlinked (missing barcode) rows, which will complicate product-level sales analysis.
* Mixing numeric and string values in `FINAL\_QUANTITY` was successfully resolved.
* Missing FINAL\_SALE entries need further business interpretation, they could be refunds, loyalty items, or data entry errors.
* Dates are now in proper datetime format, enabling time-based analysis.

1. **Users Data:**

* **Objective:**
* Standardizing inconsistent gender values.
* Filling in missing demographic fields (birth date, state, language, gender).
* Identifying and handling placeholder birth dates.
* Extracting birth year to support age-based analysis in future steps.
* **Assumptions:**
* 1970-01-01 birth dates are placeholders, not actual dates of birth. These need to be identified.
* Missing LANGUAGE and GENDER data can be filled with a generic 'unknown’ category for analysis purposes.
* GENDER values were provided in multiple inconsistent formats (uppercase, lowercase, free-text). These should be standardized to:
  + - * + female
        + male
        + other (for non-binary, transgender, etc.)
        + prefer\_not\_to\_say
        + unknown
* **Code:**

import pandas as pd

# Load users dataset

users = pd.read\_csv('USER\_TAKEHOME.csv')

# Create a working copy to preserve the original data

users\_clean = users.copy()

# Convert BIRTH\_DATE to datetime; handle errors and leave invalid dates as NaT (missing)

users\_clean['BIRTH\_DATE'] = pd.to\_datetime(users\_clean['BIRTH\_DATE'], errors='coerce')

# Fill missing STATE, LANGUAGE, and GENDER with placeholders

users\_clean['STATE'] = users\_clean['STATE'].fillna('Unknown State')

users\_clean['LANGUAGE'] = users\_clean['LANGUAGE'].fillna('unknown')

users\_clean['GENDER'] = users\_clean['GENDER'].fillna('unknown')

# Standardize GENDER field to a controlled list

gender\_map = {

'female': 'female',

'male': 'male',

'transgender': 'other',

'non\_binary': 'other',

'Non-Binary': 'other',

'prefer\_not\_to\_say': 'prefer\_not\_to\_say',

'Prefer not to say': 'prefer\_not\_to\_say',

'not\_listed': 'other',

'not\_specified': 'unknown',

'My gender isn\'t listed': 'other',

'unknown': 'unknown'

}

users\_clean['GENDER'] = users\_clean['GENDER'].map(gender\_map)

# Flag placeholder birth dates (assumed placeholder date is 1970-01-01)

users\_clean['PLACEHOLDER\_BIRTHDATE'] = users\_clean['BIRTH\_DATE'].apply(

lambda x: 'PLACEHOLDER' if x == pd.Timestamp('1970-01-01') else 'OK'

)

# Extract birth year for future age-based segmentation

users\_clean['BIRTH\_YEAR'] = users\_clean['BIRTH\_DATE'].dt.year

# Summary Statistics and Findings

original\_count = len(users)

missing\_birth\_date\_count = users['BIRTH\_DATE'].isnull().sum()

missing\_state\_count = users['STATE'].isnull().sum()

missing\_language\_count = users['LANGUAGE'].isnull().sum()

missing\_gender\_count = users['GENDER'].isnull().sum()

placeholder\_birthdate\_count = (users\_clean['PLACEHOLDER\_BIRTHDATE'] == 'PLACEHOLDER').sum()

# Show Final Normalized Gender Distribution

normalized\_gender\_counts = users\_clean['GENDER'].value\_counts()

# Print Final Cleanup Summary

print("Users Cleanup Summary:")

print(f"Original User Count: {original\_count}")

print(f"Missing Birth Dates (original): {missing\_birth\_date\_count}")

print(f"Missing States (original): {missing\_state\_count}")

print(f"Missing Languages (original): {missing\_language\_count}")

print(f"Missing Genders (original): {missing\_gender\_count}")

print(f"Placeholder Birth Dates (1970-01-01): {placeholder\_birthdate\_count}")

print("\nNormalized Gender Distribution (After Cleanup):")

print(normalized\_gender\_counts)

* **Conclusions:**
* The gender field was highly inconsistent and required significant normalization.
* Missing birth dates, states, and languages were filled with appropriate placeholder values, making the dataset suitable for further analysis.
* Placeholder birth dates (1970-01-01) were successfully flagged, none remain after parsing.
* Language data is particularly sparse (30.5% missing), indicating potential issues with capturing user preferences at registration.
* Age segmentation is now possible using the extracted birth year.

**Final Gender Distribution (After Cleanup)**

| **Gender** | **Count** |
| --- | --- |
| **Female** | 64,240 |
| **Male** | 25,829 |
| **Unknown** | 6,116 |
| **Other** | 2,464 |
| **Prefer not to say** | 1,351 |

**Section II: Provide SQL Queries**

To efficiently query and analyze the cleaned datasets, I created an in-memory SQLite database using Python's built-in `sqlite3` library. This approach allowed me to:

* Store the cleaned Products, Transactions, and Users datasets as SQL tables directly from their CSV files.
* Use SQL queries to join the tables and extract insights across all three datasets.
* Take advantage of SQL’s powerful filtering, grouping, and aggregation functions to answer business questions efficiently.
* Combine SQL results with Python data visualization libraries such as `matplotlib` and `seaborn` to present findings graphically.

**SQLite Setup:**

Before running any queries, I used the following code to set up an in-memory SQLite database and load the cleaned datasets (`PRODUCTS\_TAKEHOME\_CLEANED.csv`, `TRANSACTION\_TAKEHOME\_CLEANED.csv`, `USER\_TAKEHOME\_CLEANED.csv`). This setup step ensures that all queries operate on the same unified database, allowing seamless joins between products, transactions, and users. Since this setup is common for all queries, it is placed at the top of the script and does not need to be repeated for each individual query.

**Code:**

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load Data into SQLite In-Memory Database

conn = sqlite3.connect(":memory:")

# Load cleaned data into SQLite tables

products\_cleaned = pd.read\_csv('PRODUCTS\_TAKEHOME\_CLEANED.csv')

transactions\_cleaned = pd.read\_csv('TRANSACTION\_TAKEHOME\_CLEANED.csv')

users\_cleaned = pd.read\_csv('USER\_TAKEHOME\_CLEANED.csv')

products\_cleaned.to\_sql('products', conn, index=False, if\_exists='replace')

transactions\_cleaned.to\_sql('transactions', conn, index=False, if\_exists='replace')

users\_cleaned.to\_sql('users', conn, index=False, if\_exists='replace')

print("Tables loaded into SQLite:")

print(pd.read\_sql("SELECT name FROM sqlite\_master WHERE type='table';", conn))

**Closed-Ended Questions:**

1. **What are the top 5 brands by receipts scanned among users 21 and over?**

* **Assumptions:**
* Age Calculation: User age is calculated using the formula:  
  (CurrentYear−BirthYear)(CurrentYear−BirthYear)
* Users with missing or invalid birth years are excluded from this analysis.
* A "receipt scanned" is defined as a unique receipt ID — not individual items on the receipt.
* Users must be at least 21 years old at the time of analysis to be included.
* **Code:**

# Top 5 Brands by Receipts Scanned for Users 21+

query\_1 = '''

SELECT p.BRAND, COUNT(DISTINCT t.RECEIPT\_ID) AS receipt\_count

FROM transactions t

JOIN products p ON t.BARCODE = p.BARCODE

JOIN users u ON t.USER\_ID = u.ID

WHERE (strftime('%Y', 'now') - u.BIRTH\_YEAR) >= 21

GROUP BY p.BRAND

ORDER BY receipt\_count DESC

LIMIT 5;

'''

top\_brands = pd.read\_sql(query\_1, conn)

print("\nTop 5 Brands by Receipts Scanned (21+):")

print(top\_brands)

# Horizontal Bar Chart

plt.figure(figsize=(10, 6))

sns.barplot(y='BRAND', x='receipt\_count', data=top\_brands)

plt.title('Top 5 Brands by Receipts Scanned (21+)')

plt.show()

* **Output:**

**A graph with colorful bars

Description automatically generated with medium confidence**

1. **What are the top 5 brands by sales among users that have had their account for at least six months?**

* **Assumptions:**
* Account Age: Users are considered to have an account for at least 6 months if their CREATED\_DATE is at least 6 months prior to today's date (current system date).
* Sales Definition: Total sales is the sum of the FINAL\_SALE column, after converting it to a numeric type.
* Transactions with missing or blank FINAL\_SALE values are excluded from the sales total.
* **Code:**

# Top 5 Brands by Sales for Users with 6+ Months Account Age

query\_2 = '''

SELECT p.BRAND, SUM(CAST(t.FINAL\_SALE AS FLOAT)) AS total\_sales

FROM transactions t

JOIN products p ON t.BARCODE = p.BARCODE

JOIN users u ON t.USER\_ID = u.ID

WHERE date('now') >= date(u.CREATED\_DATE, '+6 months')

AND t.FINAL\_SALE\_FLAG = 'OK'

GROUP BY p.BRAND

ORDER BY total\_sales DESC

LIMIT 5;

'''

top\_sales\_brands = pd.read\_sql(query\_2, conn)

print("\nTop 5 Brands by Sales (6+ Months Users):")

print(top\_sales\_brands)

# Pie Chart

plt.figure(figsize=(8, 8))

plt.pie(top\_sales\_brands['total\_sales'], labels=top\_sales\_brands['BRAND'], autopct='%1.1f%%', startangle=140)

plt.title('Top 5 Brands by Sales (6+ Months Users)')

plt.show()

* **Output:**

A pie chart with numbers and a diagram

Description automatically generated

1. **What is the percentage of sales in the Health & Wellness category by generation?**

* **Assumptions:**
* Generation Definitions: Users are grouped into generations based on:
* Gen Z: Born 1997 or later.
* Millennials: Born 1981 to 1996.
* Gen X: Born 1965 to 1980.
* Baby Boomers: Born before 1965.
* Missing or Invalid Birth Dates: Users without valid birth dates are excluded from generation classification.
* Category Identification: Transactions are assigned to the Health & Wellness category based on CATEGORY\_1.
* Sales Definition: Total sales is the sum of the FINAL\_SALE column, after converting it to numeric.
* Overall Sales Comparison: The percentage of Health & Wellness sales is calculated as a percentage of all valid sales across all categories.
* **Code:**

# Percentage of Health & Wellness Sales by Generation

query\_3 = '''

WITH user\_gen AS (

SELECT

ID,

CASE

WHEN BIRTH\_YEAR >= 1997 THEN 'Gen Z'

WHEN BIRTH\_YEAR BETWEEN 1981 AND 1996 THEN 'Millennials'

WHEN BIRTH\_YEAR BETWEEN 1965 AND 1980 THEN 'Gen X'

ELSE 'Baby Boomers'

END AS generation

FROM users

)

, category\_sales AS (

SELECT

g.generation,

SUM(CAST(t.FINAL\_SALE AS FLOAT)) AS health\_wellness\_sales

FROM transactions t

JOIN products p ON t.BARCODE = p.BARCODE

JOIN user\_gen g ON t.USER\_ID = g.ID

WHERE p.CATEGORY\_1 = 'Health & Wellness'

AND t.FINAL\_SALE\_FLAG = 'OK'

GROUP BY g.generation

)

, total\_sales AS (

SELECT SUM(CAST(t.FINAL\_SALE AS FLOAT)) AS total\_sales

FROM transactions t

WHERE t.FINAL\_SALE\_FLAG = 'OK'

)

SELECT

cs.generation,

(cs.health\_wellness\_sales \* 100.0 / ts.total\_sales) AS health\_wellness\_percentage

FROM category\_sales cs, total\_sales ts

ORDER BY health\_wellness\_percentage DESC;

'''

gen\_health\_wellness = pd.read\_sql(query\_3, conn)

print("\nHealth & Wellness Sales by Generation:")

print(gen\_health\_wellness)

# Vertical Bar Chart

plt.figure(figsize=(8, 5))

sns.barplot(x='generation', y='health\_wellness\_percentage', data=gen\_health\_wellness)

plt.title('Health & Wellness Sales by Generation')

plt.show()

**Output:**

**A graph with numbers and a bar

Description automatically generated with medium confidence**

**Open-Ended Questions:**

1. **Who are Fetch’s power users?**

* **Assumptions:**
* Definition of Power User: Power users are those who:
* Scanned the highest number of distinct receipts.
* Purchased the highest total quantity of items.
* Spent the most money overall.
* Missing Sales or Quantities: Transactions with missing or non-numeric FINAL\_SALE or FINAL\_QUANTITY values are excluded.
* Users without valid transactions are **excluded** from the power user list.
* **Code:**

# Fetch Power Users

query\_4 = '''

SELECT

u.ID AS user\_id,

COUNT(DISTINCT t.RECEIPT\_ID) AS total\_receipts,

SUM(t.FINAL\_QUANTITY) AS total\_items\_purchased,

SUM(CAST(t.FINAL\_SALE AS FLOAT)) AS total\_spent

FROM transactions t

JOIN users u ON t.USER\_ID = u.ID

WHERE t.FINAL\_SALE\_FLAG = 'OK'

GROUP BY u.ID

ORDER BY total\_receipts DESC, total\_spent DESC, total\_items\_purchased DESC

LIMIT 10;

'''

power\_users = pd.read\_sql(query\_4, conn)

print("\nFetch Power Users:")

print(power\_users)

# Scatter Plot - Total Receipts vs Total Spent

plt.figure(figsize=(8, 5))

plt.scatter(power\_users['total\_receipts'], power\_users['total\_spent'], color='green')

plt.title('Top 10 Power Users - Receipts vs Spend')

plt.xlabel('Total Receipts')

plt.ylabel('Total Spent')

plt.grid(True)

plt.show()

* **Output:**

**A screenshot of a graph

Description automatically generated**

1. **Which is the leading brand in the Dips & Salsa category?**

* **Assumptions:**
* Category Identification: Products are classified into the Dips & Salsa category using CATEGORY\_3.
* Sales Definition: Total sales is the sum of FINAL\_SALE, after conversion to numeric.
* Transactions with **missing or blank FINAL\_SALE values** are excluded.
* Only products that have been properly termed as Dip or Salsa are considered.
* **Code:**

# Leading Brand in Dips & Salsa

query\_5 = '''

SELECT p.BRAND, SUM(CAST(t.FINAL\_SALE AS FLOAT)) AS total\_sales

FROM transactions t

JOIN products p ON t.BARCODE = p.BARCODE

WHERE LOWER(p.CATEGORY\_3) LIKE '%dip%' OR LOWER(p.CATEGORY\_3) LIKE '%salsa%'

AND t.FINAL\_SALE\_FLAG = 'OK'

GROUP BY p.BRAND

ORDER BY total\_sales DESC

LIMIT 1;

'''

top\_dips\_brand = pd.read\_sql(query\_5, conn)

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

print(top\_dips\_brand)

# Simple Text Output

print(f"The leading brand in Dips & Salsa is: {top\_dips\_brand['BRAND'][0]} with sales of ${top\_dips\_brand['total\_sales'][0]:,.2f}")

* **Output:**

A close up of a sign

Description automatically generated

1. **At what percent has Fetch grown year over year?**

* **Assumptions:**
* Growth Metric that I am using to check Fetch's growth is defined by the number of new user accounts created each year.
* Users with missing creation dates (CREATED\_DATE) are excluded from the analysis.
* Users are grouped by year using the CREATED\_DATE field, where each user is assigned to the year in which their account was created.
* Year-over-Year growth is calculated as:

(UsersThisYear−UsersLastYear)UsersLastYear×100%UsersLastYear(UsersThisYear−UsersLastYear)​×100%

* This formula compares the count of new users in each year to the previous year to compute the percentage growth rate.
* The first year with data does not have a prior year for comparison, so it will have a NULL growth rate.
* User counts are based solely on account creation date. No adjustments are made for user churn, inactivity, or multiple accounts per user.
* **Code:**

# Year-over-Year Growth (Based on User Account Creation Date)

# Ensure dates in users table are correctly formatted in SQLite (done during initial load)

query\_6 = '''

WITH yearly\_users AS (

SELECT

strftime('%Y', CREATED\_DATE) AS year,

COUNT(\*) AS user\_count

FROM users

WHERE CREATED\_DATE IS NOT NULL

GROUP BY year

)

, yoy\_growth AS (

SELECT

year,

user\_count,

LAG(user\_count) OVER (ORDER BY year) AS previous\_year\_user\_count,

CASE

WHEN LAG(user\_count) OVER (ORDER BY year) IS NOT NULL

THEN (user\_count - LAG(user\_count) OVER (ORDER BY year)) \* 100.0 / LAG(user\_count) OVER (ORDER BY year)

ELSE NULL

END AS yoy\_growth\_percent

FROM yearly\_users

)

SELECT \* FROM yoy\_growth;

'''

yoy\_growth = pd.read\_sql(query\_6, conn)

print("\nYear-over-Year User Growth (Based on Account Creation Date):")

print(yoy\_growth)

# Line Chart for YoY Growth

plt.figure(figsize=(10, 6))

plt.plot(yoy\_growth['year'], yoy\_growth['yoy\_growth\_percent'], marker='o', linestyle='-', color='purple')

plt.title('Year-over-Year User Growth (Based on Account Creation Date)')

plt.xlabel('Year')

plt.ylabel('YoY Growth (%)')

plt.grid(True)

plt.show()

* **Output:**

A graph with a line and numbers

Description automatically generated

### Section III: Communicate with Stakeholders

Here’s a professional yet accessible email to communicate my findings and next steps to stakeholders, business or product leadership.

**Subject: Initial Findings & Data Quality Review – Fetch Data Analysis**

Hi [Stakeholders/Product/Business Leader's Name],

As part of the data quality and exploratory analysis for the Fetch datasets, I wanted to share initial findings, highlight data quality concerns, and propose next steps to enhance the accuracy and value of my analysis.

**Key Data Quality Issues Identified:**

1. **Product Data Gaps:**

* ~27% of products are missing key fields like manufacturer and brand, limiting product-level insights.
* Over 4,000 products lack barcodes, making them unmatchable with transactions.
* Placeholder values like “PLACEHOLDER MANUFACTURER” were found, raising data reliability concerns.

1. **Transactions Data Inconsistencies:**

* Approximately 11.5% of transactions lack valid barcodes, making them difficult to link to products.
* The FINAL\_QUANTITY field mixes numeric values with text (e.g., “zero”), requiring cleanup.
* FINAL\_SALE contains blanks, which could indicate refunds, loyalty transactions, or errors - clarification needed.

1. **User Data Completeness Issues:**

* Over 30% of users are missing language information, which could impact segmentation.
* There are placeholder birthdates (e.g., 1970-01-01) that need to be excluded or handled appropriately.
* The gender field contains inconsistent values (e.g., free-text and mixed formats), requiring standardization.

**One Interesting Trend Observed:**

**Generational Sales Bias in Health & Wellness**

* **Key Findings:**
* Millennials are the dominant generation driving Health & Wellness sales on Fetch.
* This generation alone accounts for the highest total sales volume in this category, outpacing Gen Z, Gen X, and Baby Boomers.
* This aligns with broader industry trends, where Millennials are known to prioritize health, wellness, and self-care purchases — from supplements to organic foods and personal care products.
* **Additional Insights:**
* Gen Z, while younger and still developing purchasing power, already represents a non-trivial share of Health & Wellness sales, indicating potential for future growth as this generation ages.
* Baby Boomers have the lowest total spend in this category, likely reflecting differing shopping habits and product preferences.
* **Interpretation for Business Leaders:**

This insight can help Fetch’s product and marketing teams in the following ways:

* Targeted Campaigns: Develop messaging and promotions that resonate with Millennials' interest in health and sustainability.
* Product Curation: Prioritize onboarding new brands that appeal to Millennials and younger Gen Z consumers.
* Generational Personalization: Use this insight to personalize offers within the app, Baby Boomers may prefer functional health products, while Millennials may lean toward organic or premium health goods.

**A graph with numbers and a bar

Description automatically generated with medium confidence**

**Outstanding Questions for Business Clarification:**

1. FINAL\_SALE Interpretation

* How should transactions with missing or zero `FINAL\_SALE` values be treated?
* Are these refunds, loyalty-based redemptions, or incomplete data?

1. Placeholder Data Handling

* Are placeholder birthdates (e.g., 1970-01-01) standard practice for unknown values, or do they represent something else?
* Should placeholder manufacturers (e.g., "PLACEHOLDER MANUFACTURER") be excluded or rolled into an "Unknown" bucket?

1. Barcode-Free Transactions

* Should transactions missing product barcodes be excluded from product-level analysis, or do they represent valuable non-scanned items (e.g., fresh produce or services)?

I would appreciate input from the product and data teams on these questions so we can finalize data cleaning rules and ensure our analysis accurately reflects user behavior and product performance. Please let me know if you’d like me to schedule a working session to review these findings in more detail.

Thank you,

Sai Raj Peddi