# Excersise 1 - Data Exploration and Vizualization

February 17, 2025

## 1 Data Quality Check

Fetch - Data Analyst Take Home Created by: Sakshi Rajendra Khadayate

[1]:
```
# Import necessary libraries
import pandas as pd
import unicodedata
```

[2]:
```
# Import CSV files using pandas
# Encoding is set to 'utf-8' to handle any special characters that may be␣
  ↪present in the files
df_users = pd.read_csv('USER_TAKEHOME.csv', encoding='utf-8')
df_products = pd.read_csv('PRODUCTS_TAKEHOME.csv', encoding = 'utf-8')
df_transactions = pd.read_csv('TRANSACTION_TAKEHOME (1).csv', encoding='utf-8')
```

[3]:
```
df_users.head()
```

[3]:
```
                         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
```

[4]:
```
df_products.head()
```

[4]:
```
          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  7.964940e+11
1              NaN  2.327801e+10
2          ELECSOP  4.618180e+11
3          COLGATE  3.500047e+10
4  MAPLE HOLISTICS  8.068110e+11
```

[5]:

```
df_transactions.head()
```

[5]:

```
                             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
```

[6]:

```
# Function to check and fix duplicates and null values for a given primary key
def remove_null_and_duplicate(df, primary_key):

    # Print the number of rows and columns
    print("Shape of DataFrame:", df.shape)

    # Print count of missing values in each column
    print("\nMissing Values:\n", df.isnull().sum())
    # Print the number of unique values in each column
    print("\nUnique Values Count:\n", df.nunique())

    # Drop rows where primary key column has missing (NaN) values
    df = df.dropna(subset=[primary_key])

    # Check if primary key column has duplicate values
    if df[primary_key].duplicated().any():
        print(f"\nWarning: Duplicate values found in '{primary_key}'. Keeping␣
  ↪only the first occurrence.")

        # Keep only the first occurrence of each duplicate
        df = df.drop_duplicates(subset=[primary_key])

    return df
```

[7]:

```
# Run the function to remove_null_and_duplicate (Users and Product tables has␣
  ↪primary keys)
print ("Users")
df_users = remove_null_and_duplicate(df_users, 'ID')
print("--------------------------")
print ("Products")
df_products = remove_null_and_duplicate(df_products, 'BARCODE')
print("--------------------------")

print ("Transactions")
# Print the number of rows and columns
print("Shape of DataFrame:", df_transactions.shape)

# Print count of missing values in each column
print("\nMissing Values:\n", df_transactions.isnull().sum())

# Print the number of unique values in each column
print("\nUnique Values Count:\n", df_transactions.nunique())
```

```
Users
Shape of DataFrame: (100000, 6)

Missing Values:
 ID                    0
CREATED_DATE         0
BIRTH_DATE        3675
STATE             4812
LANGUAGE         30508
GENDER            5892
dtype: int64
```

```
Unique Values Count:
 ID              100000
CREATED_DATE     99942
BIRTH_DATE       54721
STATE               52
LANGUAGE             2
GENDER              11
dtype: int64
-------------------------
Products
Shape of DataFrame: (20773, 7)

Missing Values:
 CATEGORY_1          2
CATEGORY_2         34
CATEGORY_3       1554
CATEGORY_4      19186
MANUFACTURER     5547
BRAND            5547
BARCODE           112
dtype: int64

Unique Values Count:
 CATEGORY_1         14
CATEGORY_2         53
CATEGORY_3        202
CATEGORY_4         51
MANUFACTURER     1723
BRAND            3191
BARCODE         13590
dtype: int64

Warning: Duplicate values found in 'BARCODE'. Keeping only the first occurrence.
-------------------------
Transactions
Shape of DataFrame: (50000, 8)

Missing Values:
 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
```

```
Unique Values Count:
 RECEIPT_ID        24440
PURCHASE_DATE         89
SCAN_DATE          24440
STORE_NAME           954
USER_ID            17694
BARCODE            11027
FINAL_QUANTITY        87
FINAL_SALE          1435
dtype: int64
```

[8]:

```python
# Trim spaces from column names to ensure no extra spaces cause issues during data manipulation
df_users.columns = df_users.columns.str.strip()
df_products.columns = df_products.columns.str.strip()
df_transactions.columns = df_transactions.columns.str.strip()

# Trim spaces from all string values in the DataFrame to clean any leading or trailing spaces in the actual data
df_users = df_users.applymap(lambda x: x.strip() if isinstance(x, str) else x)
df_products = df_products.applymap(lambda x: x.strip() if isinstance(x, str) else x)
df_transactions = df_transactions.applymap(lambda x: x.strip() if isinstance(x, str) else x)
```

[9]:

```python
# Function to check if a string contains non-ASCII characters
def contains_non_ascii(s):
    if isinstance(s, str):  # Check if it's a string
        return not all(ord(c) < 128 for c in s)
    return False  # Non-strings are considered valid as they don't contain non-ASCII characters

# Function to check all string columns for non-ASCII values
def find_non_ascii_rows(df):

    # Select only string columns
    string_cols = df.select_dtypes(include=['object'])

    # Identify non-ASCII values
    mask = string_cols.applymap(contains_non_ascii)

    # Get rows where any string column has a non-ASCII value
    non_ascii_rows = df[mask.any(axis=1)]

    # Print the column names where non-ASCII values are found
    if not non_ascii_rows.empty:
        affected_columns = mask.any(axis=0)
        print("Non-ASCII values found in columns in :",␣
 ↪list(affected_columns[affected_columns].index))
    else:
        print("No Non-ASCII values found")

    return non_ascii_rows
```

[10]:

```
# Find and print rows with non-ASCII characters
print ("Users")
non_ascii_rows = find_non_ascii_rows(df_users)
print ("\nProducts")
non_ascii_rows = find_non_ascii_rows(df_products)
print ("\nTransactions")
non_ascii_rows = find_non_ascii_rows(df_transactions)
```

```
Users
No Non-ASCII values found

Products
Non-ASCII values found in columns in : ['MANUFACTURER', 'BRAND']

Transactions
Non-ASCII values found in columns in : ['STORE_NAME']
```

[11]:

```
# ============================
# Users Table Data Cleaning and Transformation
# ============================

# Convert Date  column to SQL-friendly format ('YYYY-MM-DD HH:MM:SS')
df_users['CREATED_DATE'] = pd.to_datetime(df_users['CREATED_DATE']).dt.
 ↪strftime('%Y-%m-%d %H:%M:%S')
df_users['BIRTH_DATE'] = pd.to_datetime(df_users['BIRTH_DATE']).dt.
 ↪strftime('%Y-%m-%d %H:%M:%S')

# Fill missing 'BIRTH_DATE' values with a default pseudo date ('1900-01-01 00:
 ↪00:00'), making the data consistent
df_users['BIRTH_DATE'] = df_users['BIRTH_DATE'].fillna('1900-01-01 00:00:00')

# Save the updated file
output_file = "converted_file_users.csv"
df_users.to_csv(output_file, index=False)

print(f"Successfully converted file in desired format, Saved as: {output_file}")
```

```
Successfully converted file in desired format, Saved as:
converted_file_users.csv
```

[12]:

```
# =============================
# Products Table Data Cleaning and Transformation
# =============================

# Function to replace non-ASCII characters in strings
def replace_non_ascii(s):
    if isinstance(s, str):  # Apply only to strings
        return unicodedata.normalize('NFKD', s).encode('ascii', 'ignore').
 ↪decode('ascii')
    return s   # Return non-string values as they are

# Select only string columns
string_cols = df_products.select_dtypes(include=['object'])

# Apply the replacement function to only string columns
df_products[string_cols.columns] = string_cols.applymap(replace_non_ascii)

# Save the updated file
output_file = "converted_file_products.csv"
df_products.to_csv(output_file, index=False, encoding='utf-8')

print(f"Successfully converted file in desired format, Saved as: {output_file}")
```

```
Successfully converted file in desired format, Saved as:
converted_file_products.csv
```

[13]:

```
# =============================
# Transactions Table Data Cleaning and Transformation
# =============================

# # Convert Date  column to SQL-friendly format ('YYYY-MM-DD HH:MM:SS'),␣
 ↪replace blanks with 0
df_transactions['PURCHASE_DATE'] = pd.
 ↪to_datetime(df_transactions['PURCHASE_DATE']).dt.strftime('%Y-%m-%d %H:%M:
 ↪%S')
df_transactions['SCAN_DATE'] = pd.to_datetime(df_transactions['SCAN_DATE']).dt.
 ↪strftime('%Y-%m-%d %H:%M:%S')
df_transactions['BARCODE'] = df_transactions['BARCODE'].fillna(0)
df_transactions = df_transactions[df_transactions['FINAL_QUANTITY']!= 'zero']
df_transactions = df_transactions[df_transactions['FINAL_SALE']!= '']

# Function to replace non-ASCII characters in strings
def replace_non_ascii(s):
    if isinstance(s, str):  # Apply only to strings
        return unicodedata.normalize('NFKD', s).encode('ascii', 'ignore').
 ↪decode('ascii')
    return s   # Return non-string values as they are
```

```python
# Select only string columns
string_cols = df_transactions.select_dtypes(include=['object'])

# Apply the replacement function to only string columns
df_transactions[string_cols.columns] = string_cols.applymap(replace_non_ascii)

# Save the updated file
output_file = "converted_file_transactions.csv"
df_transactions.to_csv(output_file, index=False)

print(f"Successfully converted file in desired format, Saved as: {output_file}")
```

```
Successfully converted file in desired format, Saved as:
converted_file_transactions.csv
```

[14]:
```python
df_users.head()
```

[14]:
```
                         ID         CREATED_DATE           BIRTH_DATE STATE  \
0  5ef3b4f17053ab141787697d  2020-06-24 20:17:54  2000-08-11 00:00:00    CA
1  5ff220d383fcfc12622b96bc  2021-01-03 19:53:55  2001-09-24 04:00:00    PA
2  6477950aa55bb77a0e27ee10  2023-05-31 18:42:18  1994-10-28 00:00:00    FL
3  658a306e99b40f103b63ccf8  2023-12-26 01:46:22  1900-01-01 00:00:00    NC
4  653cf5d6a225ea102b7ecdc2  2023-10-28 11:51:50  1972-03-19 00:00:00    PA

  LANGUAGE  GENDER
0   es-419  female
1       en  female
2   es-419  female
3       en     NaN
4       en  female
```

[15]:
```python
df_products.head()
```

[15]:
```
          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  7.964940e+11
1               NaN  2.327801e+10
2           ELECSOP  4.618180e+11
3           COLGATE  3.500047e+10
4  MAPLE HOLISTICS  8.068110e+11
```

[16]:
```python
df_transactions.head()
```

[16]:
```
                                 RECEIPT_ID        PURCHASE_DATE  \
25000  7b3ec72d-9d30-40b8-b185-0bfb638942a9  2024-08-20 00:00:00
25001  04869b68-29e3-4e8d-9bdb-950046fc3473  2024-08-05 00:00:00
25002  f1a96308-24a5-46a8-8d8c-285cf9dce1ba  2024-09-03 00:00:00
25003  7ee1798e-fd2e-4278-838b-f417fdcafe08  2024-08-30 00:00:00
25004  21feab39-49f2-42e9-ae69-10371e2fc0a9  2024-08-23 00:00:00

                 SCAN_DATE           STORE_NAME                   USER_ID  \
25000  2024-08-20 11:17:29  DOLLAR GENERAL STORE  60fc1e6deb7585430ff52ee7
25001  2024-08-09 16:06:00  DOLLAR GENERAL STORE  654cf234a225ea102b81072e
25002  2024-09-03 11:28:25               WALMART  63c1cb6d3d310dceeac55487
25003  2024-09-04 12:53:31  DOLLAR GENERAL STORE  65c29b137050d0a6206cd24f
25004  2024-08-27 10:45:00                TARGET  61a58ac49c135b462ccddd1c

            BARCODE FINAL_QUANTITY FINAL_SALE
25000  7.455271e+11           1.00       1.65
25001  7.455271e+11           1.00       1.65
25002  3.700083e+10           1.00      28.22
25003  1.200050e+10           1.00       5.25
25004  2.400039e+10           1.00       2.59
```

————————————————————- All 3 files are ready to be imported in SQL for further Analysis ————————————————————-

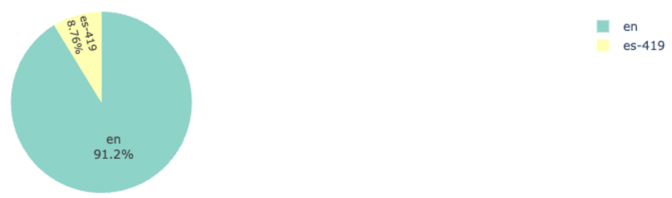
## 2 ————————————————————————————————————

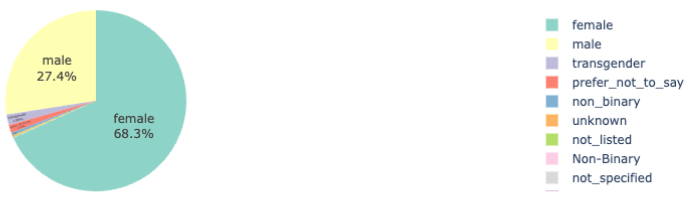
## 3 Data Visualization

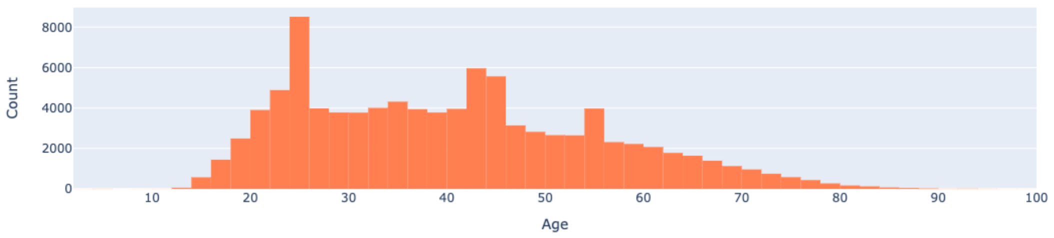
[17]:
```python
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import numpy as np
from datetime import datetime

# Read the CSV files
df_users = pd.read_csv('converted_file_users.csv')
```

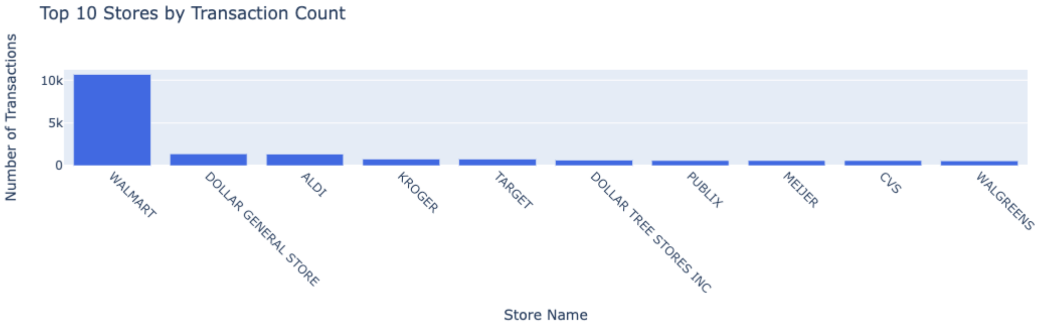
```
df_products = pd.read_csv('converted_file_products.csv')
df_transactions = pd.read_csv('converted_file_transactions.csv')

#Convert date columns to datetime
df_users['CREATED_DATE'] = pd.to_datetime(df_users['CREATED_DATE'])
df_users['BIRTH_DATE'] = pd.to_datetime(df_users['BIRTH_DATE'])
df_transactions['PURCHASE_DATE'] = pd.
 ↪to_datetime(df_transactions['PURCHASE_DATE'])
df_transactions['SCAN_DATE'] = pd.to_datetime(df_transactions['SCAN_DATE'])

# 1. User Language Distribution
def plot_language_distribution():
    lang_dist = df_users['LANGUAGE'].value_counts().reset_index()
    lang_dist.columns = ['Language', 'Count']

    fig = px.pie(lang_dist,
                 values='Count',
                 names='Language',
                 title='User Language Distribution',
                 color_discrete_sequence=px.colors.qualitative.Set3)

    fig.update_traces(textposition='inside', textinfo='percent+label')
    fig.show()

# 2. User Gender Distribution
def plot_gender_distribution():
    lang_dist = df_users['GENDER'].value_counts().reset_index()
    lang_dist.columns = ['Gender', 'Count']

    fig = px.pie(lang_dist,
                 values='Count',
                 names='Gender',
                 title='User Gender Distribution',  # Corrected title
                 color_discrete_sequence=px.colors.qualitative.Set3)

    fig.update_traces(textposition='inside', textinfo='percent+label')
    fig.show()

# 3. User Age Distribution
def plot_age_distribution():
    df_users['AGE'] = (datetime.now() - df_users['BIRTH_DATE']).dt.days / 365.25

    fig = px.histogram(df_users[df_users['AGE'] < 100], # Filter out␣
 ↪unrealistic ages
                       x='AGE',
                       nbins=50,
                       title='User Age Distribution',
                      color_discrete_sequence=['#FF7F50'])

    fig.update_layout(xaxis_title='Age',
                     yaxis_title='Count')
    fig.show()

# 4. Top Stores Analysis
def plot_top_stores():
    store_counts = df_transactions['STORE_NAME'].value_counts().head(10)

    fig = px.bar(x=store_counts.index,
                 y=store_counts.values,
                 title='Top 10 Stores by Transaction Count',
                 color_discrete_sequence=['#4169E1'])

    fig.update_layout(xaxis_title='Store Name',
                     yaxis_title='Number of Transactions',
                     xaxis_tickangle=45)
    fig.show()

# 5. Category Analysis
def plot_category_analysis():
    # Create subplots for different category levels (3 rows, 2 columns)
    fig = make_subplots(rows=3, cols=2,
                       subplot_titles=('Category 1', 'Category 2', 'Category 3',
                                      'Category 4', 'Top Brands', 'Top␣
 ↪Manufacturers'))

    # Category 1 Distribution
    cat1_counts = df_products['CATEGORY_1'].value_counts().head(10)
    fig.add_trace(go.Bar(x=cat1_counts.index, y=cat1_counts.values),
                 row=1, col=1)

    # Category 2 Distribution
    cat2_counts = df_products['CATEGORY_2'].value_counts().head(10)
    fig.add_trace(go.Bar(x=cat2_counts.index, y=cat2_counts.values),
                 row=1, col=2)

    # Category 3 Distribution
    cat3_counts = df_products['CATEGORY_3'].value_counts().head(10)
    fig.add_trace(go.Bar(x=cat3_counts.index, y=cat3_counts.values),
                 row=2, col=1)

    # Category 4 Distribution
    cat4_counts = df_products['CATEGORY_4'].value_counts().head(10)
    fig.add_trace(go.Bar(x=cat4_counts.index, y=cat4_counts.values),
                 row=2, col=2)
    # Top Brands (Top 10)
    brand_counts = df_products['BRAND'].value_counts().head(10)
    fig.add_trace(go.Bar(x=brand_counts.index, y=brand_counts.values),
                 row=3, col=1)

    # Top Manufacturers (Top 10)
    manufacturer_counts = df_products['MANUFACTURER'].value_counts().head(10)
    fig.add_trace(go.Bar(x=manufacturer_counts.index, y=manufacturer_counts.
 ↪values),
                 row=3, col=2)

    # Update the layout
    fig.update_layout(height=1000,
                     showlegend=False,
                     title_text="Product Category Analysis")

    # Show the plot
    fig.show()



# 6. Transaction Trends Over Time
def plot_transaction_trends():
    daily_transactions = df_transactions.groupby('PURCHASE_DATE').agg({
        'RECEIPT_ID': 'count',
        'FINAL_SALE': 'sum'
    }).reset_index()

    # Create figure with secondary y-axis
    fig = make_subplots(specs=[[{"secondary_y": True}]])

    fig.add_trace(
        go.Scatter(x=daily_transactions['PURCHASE_DATE'],
                  y=daily_transactions['RECEIPT_ID'],
                  name="Number of Transactions"),
        secondary_y=False,
    )

    fig.add_trace(
        go.Scatter(x=daily_transactions['PURCHASE_DATE'],
                  y=daily_transactions['FINAL_SALE'],
                  name="Total Sales"),
        secondary_y=True,
    )

    fig.update_layout(
        title_text="Daily Transaction Trends",
        xaxis_title="Date"
    )

    fig.update_yaxes(title_text="Number of Transactions", secondary_y=False)
    fig.update_yaxes(title_text="Total Sales ($)", secondary_y=True)

    fig.show()


# 7. Days Between Purchase and Scan Dates
def plot_purchase_scan_diff():
    # Calculate the difference in days between PURCHASE_DATE and SCAN_DATE
    df_transactions['DIFFERENCE_DAYS'] = (df_transactions['SCAN_DATE'] -␣
 ↪df_transactions['PURCHASE_DATE']).dt.days

    # Filter out negative or unrealistic differences (optional, depending on␣
 ↪the dataset)
    df_transactions_filtered =␣
 ↪df_transactions[df_transactions['DIFFERENCE_DAYS'] >= 0]

    # Calculate the average difference in days
    avg_days = df_transactions_filtered['DIFFERENCE_DAYS'].mean()

    # Plot the histogram of the difference in days
    fig = px.histogram(df_transactions_filtered,
                       x='DIFFERENCE_DAYS',
                       nbins=50,
                       title='Distribution of Days Between Purchase and Scan␣
 ↪Dates',
                       color_discrete_sequence=['#1f77b4'])

    # Get the min and max values of the 'DIFFERENCE_DAYS' for better tick␣
 ↪control
    min_day = int(df_transactions_filtered['DIFFERENCE_DAYS'].min())
    max_day = int(df_transactions_filtered['DIFFERENCE_DAYS'].max())

    # Update the layout to show all x-axis labels and add the average line
    fig.update_layout(
        xaxis_title='Days Difference',
        yaxis_title='Count of Transactions',
        bargap=0.2,
        xaxis=dict(
            tickmode='array',  # Set the x-axis ticks as an array
            tickvals=list(range(min_day, max_day + 1)),  # Show ticks for each␣
 ↪day from min to max
```

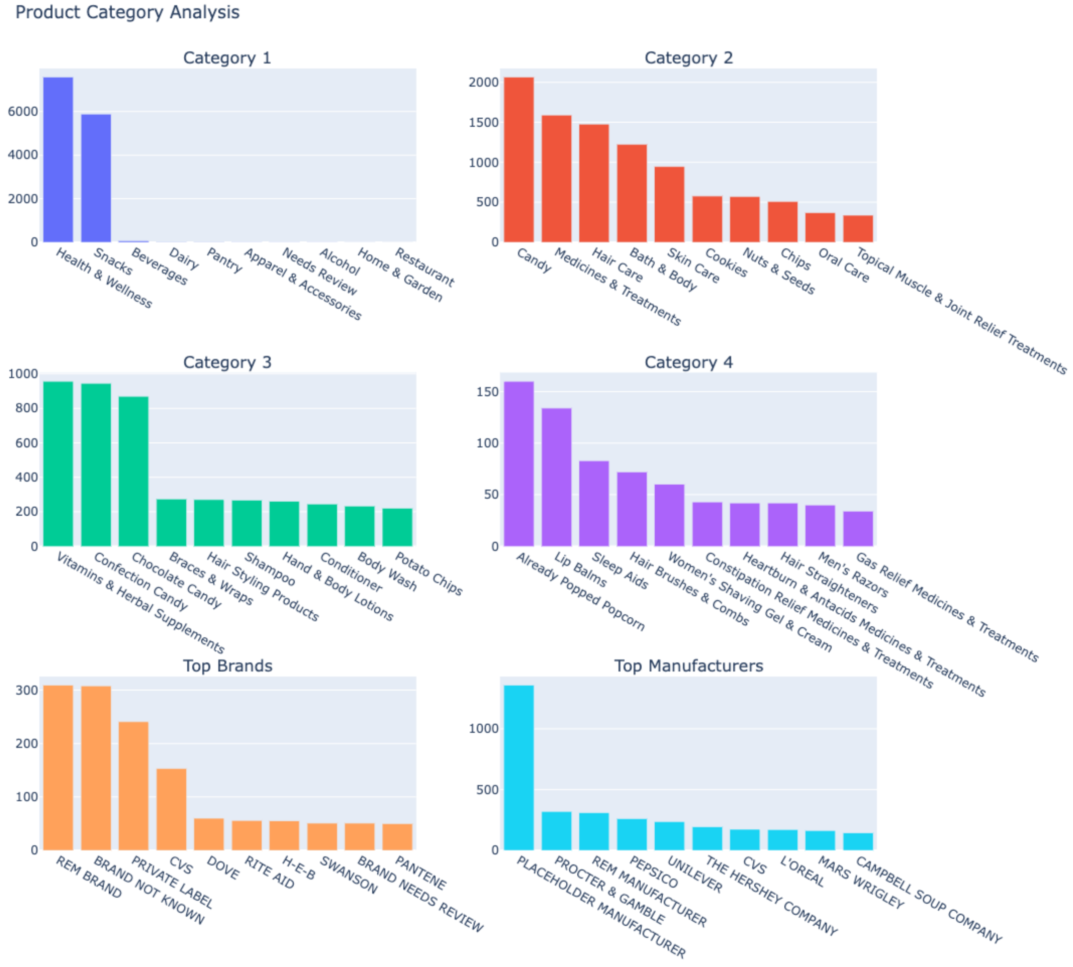
```python
            ticktext=list(map(str, range(min_day, max_day + 1)))  # Display each tick value as a string
        )
    )

    # Add vertical line for the average
    fig.add_vline(x=avg_days,
                  line=dict(color='red', dash='dash'),
                  annotation_text=f"Average: {avg_days:.2f} days",
                  annotation_position="top right")

    fig.show()

# 8. User State Distribution Map
def plot_state_distribution():
    state_counts = df_users['STATE'].value_counts().reset_index()
    state_counts.columns = ['state', 'count']

    fig = px.choropleth(state_counts,
                        locations='state',
                        locationmode="USA-states",
                        color='count',
                        scope="usa",
                        title="User Distribution by State",
                        color_continuous_scale="Viridis")

    fig.show()

# Execute all visualizations
def main():
    plot_language_distribution()
    plot_gender_distribution()
    plot_age_distribution()
    plot_top_stores()
    plot_category_analysis()
    plot_transaction_trends()
    plot_purchase_scan_diff()
    plot_state_distribution()
    print("All visualizations generated!")

if __name__ == "__main__":
    main()
```


User Language Distribution

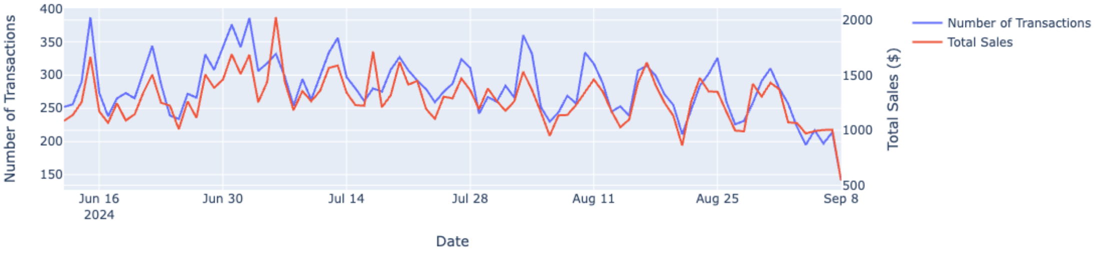

User Gender Distribution


User Age Distribution

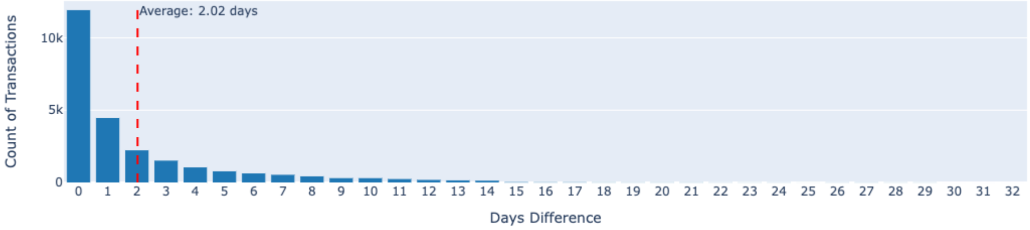

Top 10 Stores by Transaction Count
Number of Transactions
10k
5k
0
WALMART
DOLLAR GENERAL STORE
ALDI
KROGER
TARGET
DOLLAR TREE STORES INC
PUBLIX
MEIJER
CVS
WALGREENS
Store Name


Product Category Analysis
Category 1
6000
4000
2000
0
Health & Wellness
Snacks
Beverages
Dairy
Pantry
Apparel & Accessories
Needs Review
Alcohol
Home & Garden
Restaurant
Category 2
2000
1500
1000
500
0
Candy
Medicines & Treatments
Hair Care
Bath & Body
Skin Care
Cookies
Nuts & Seeds
Chips
Oral Care
Topical Muscle & Joint Relief Treatments
Category 3
1000
800
600
400
200
0
Vitamins & Herbal Supplements
Confection Candy
Chocolate Candy
Braces & Wraps
Hair Styling Products
Shampoo
Hand & Body Lotions
Conditioner
Body Wash
Potato Chips
Category 4
150
100
50
0
Already Popped Popcorn
Lip Balms
Sleep Aids
Hair Brushes & Combs
Women's Shaving Gel & Cream
Constipation Relief Medicines & Treatments
Heartburn & Antacids Medicines & Treatments
Hair Straighteners
Men's Razors
Gas Relief Medicines & Treatments
Top Brands
300
200
100
0
REM BRAND
BRAND NOT KNOWN
PRIVATE LABEL
CVS
DOVE
RITE AID
H-E-B
SWANSON
BRAND NEEDS REVIEW
PANTENE
Top Manufacturers
1000
500
0
PLACEHOLDER MANUFACTURER
PROCTER & GAMBLE
REM MANUFACTURER
PEPSICO
UNILEVER
THE HERSHEY COMPANY
CVS
L'OREAL
MARS WRIGLEY
CAMPBELL SOUP COMPANY

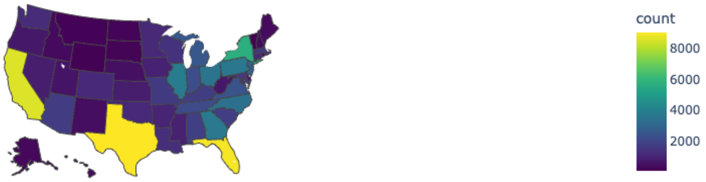
Daily Transaction Trends



Distribution of Days Between Purchase and Scan Dates



User Distribution by State



```
All visualizations generated!
```

# 4