In [1]:
```
import numpy as np
import pandas as pd
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import warnings
warnings.filterwarnings('ignore')

# Import all data files
users = pd.read_csv('./USER_TAKEHOME.csv')
txns = pd.read_csv('./TRANSACTION_TAKEHOME.csv')
products = pd.read_csv('./PRODUCTS_TAKEHOME.csv')
```

Q1: Are there any data quality issues present?

1. Duplicate Values

Duplicate Products

In [2]:
```
# Group same products by BARCODE and count occurrences
product_counts = products.groupby('BARCODE').size().reset_index(name='COUNT')

# Filter for duplicate BARCODES (more than one occurrence)
duplicate_barcodes = product_counts[product_counts['COUNT'] > 1]['BARCODE']
duplicate_products = products[products['BARCODE'].isin(duplicate_barcodes)]
duplicate_products = duplicate_products.sort_values(by='BARCODE') #Sorting to visualize same duplicate products together

print(f"Number of PRODUCT with DUPLICATE BARCODES: {len(duplicate_barcodes)}")
print(f"Number of total unique PRODUCT BARCODES: {products['BARCODE'].nunique()}")
```

```
Number of PRODUCT with DUPLICATE BARCODES: 185
Number of total unique PRODUCT BARCODES: 841342
```

In [3]:
```
# Function to highlight both duplicate row grouping and differences
def highlight_group_differences(df):
    df_highlighted = df.copy()
    df_highlighted['BARCODE'] = df_highlighted['BARCODE'].astype(str)

    # Generate unique colors for each barcode group
    unique_barcodes = df_highlighted['BARCODE'].unique()
    color_palette = px.colors.qualitative.Pastel  # Plotly's Pastel color palette
    color_palette_extended = (color_palette * (len(unique_barcodes) // len(color_palette) + 1))[:len(unique_barcodes)]
    barcode_colors = dict(zip(unique_barcodes, color_palette_extended)) # assigning a color to each of the unique barcode

    # Apply background color to entire row based on BARCODE
    row_colors = df_highlighted['BARCODE'].map(barcode_colors)
    row_styles = pd.DataFrame('', index=df.index, columns=df.columns)
    for col in df.columns:
        row_styles[col] = 'background-color: ' + row_colors

    # Compare values within groups to find differences
    for barcode, group in df.groupby('BARCODE'):
        for col in df.columns:
            if col != 'BARCODE':
                # Ignore NaN values when checking uniqueness
                if group[col].dropna().nunique() > 1:
                    row_styles.loc[group.index, col] = 'background-color: yellow'
```

```python
    return df_highlighted.style.apply(lambda x: row_styles, axis=None)

# Apply the function and display the styled dataframe
highlighted_table = highlight_group_differences(duplicate_products.head(20))
highlighted_table
```

Out[3]:

	CATEGORY_1	CATEGORY_2	CATEGORY_3	CATEGORY_4	MANUFACTURER	BRAND	BARCODE
349945	Snacks	Candy	Confection Candy	nan	MARS WRIGLEY	STARBURST	400510.0
99568	Snacks	Candy	Confection Candy	nan	MARS WRIGLEY	STARBURST	400510.0
139121	Snacks	Candy	Chocolate Candy	nan	PLACEHOLDER MANUFACTURER	BRAND NOT KNOWN	404310.0
841230	Snacks	Candy	Chocolate Candy	nan	MARS WRIGLEY	M&M'S	404310.0
274321	Snacks	Crackers	Graham Crackers	nan	TRADER JOE'S	TRADER JOE'S	438711.0
684662	Snacks	Crackers	Graham Crackers	nan	TRADER JOE'S	TRADER JOE'S	438711.0
486969	Snacks	Fruit & Vegetable Snacks	Dried Vegetables	nan	TRADER JOE'S	TRADER JOE'S	563178.0
743615	Snacks	Fruit & Vegetable Snacks	Dried Vegetables	nan	TRADER JOE'S	TRADER JOE'S	563178.0
269354	Snacks	Nuts & Seeds	Cashews	nan	TRADER JOE'S	TRADER JOE'S	603898.0
756110	Snacks	Nuts & Seeds	Cashews	nan	TRADER JOE'S	TRADER JOE'S	603898.0
610681	Snacks	Nuts & Seeds	Snack Seeds	nan	SUNRIDGE FARMS	SUNRIDGE FARMS	701983.0
645266	Snacks	Chips	Crisps	nan	TRADER JOE'S	TRADER JOE'S	701983.0
264179	Snacks	Fruit & Vegetable Snacks	Dried Vegetables	nan	TRADER JOE'S	TRADER JOE'S	853743.0
620981	Snacks	Fruit & Vegetable Snacks	Dried Vegetables	nan	TRADER JOE'S	TRADER JOE'S	853743.0
368833	Snacks	Dips & Salsa	Hummus	nan	TRADER JOE'S	TRADER JOE'S	906425.0
325056	Snacks	Dips & Salsa	Hummus	nan	TRADER JOE'S	TRADER JOE'S	906425.0
87568	Snacks	Chips	Crisps	nan	TRADER JOE'S	TRADER JOE'S	952811.0
14607	Snacks	Chips	Crisps	nan	TRADER JOE'S	TRADER JOE'S	952811.0
171015	Snacks	Nuts & Seeds	Covered Nuts	nan	TRADER JOE'S	TRADER JOE'S	969307.0
681268	Snacks	Nuts & Seeds	Almonds	nan	TRADER JOE'S	TRADER JOE'S	969307.0

📊 Insights

- **The `Products` Table contains *185 duplicate products* out of a total of *841,342 products***
- **A product with the same `BARCODE` appears multiple times with varying values in different fields such as:**
    - `BRAND` – Different brand names for the same barcode.
    - `MANUFACTURER` – Different manufacturer details for the same barcode.
    - `CATEGORY` – Products categorized differently under the same barcode.

Duplicate Transactions

In [4]:
```python
# Group transactions by RECEIPT_ID and count occurrences
receipt_counts = txns.groupby('RECEIPT_ID').size().reset_index(name='Count')

# Filter for duplicate RECEIPT_IDs (more than one occurrence)
duplicate_receipts = receipt_counts[receipt_counts['Count'] > 1]

print(f"Number of TXNS with DUPLICATE RECEIPTS: {len(duplicate_receipts)}")
print(f"Number of total unique TXN RECEIPTS: {txns['RECEIPT_ID'].nunique()}")

# Calculate the No of Unique Receipts for each Occuerence Count
occurrence_counts = receipt_counts['Count'].value_counts().reset_index()
occurrence_counts.columns = ['Number of Record in Dataset', 'Number of unique Receipts']

occurrence_counts
```

```
Number of TXNS with DUPLICATE RECEIPTS: 24440
Number of total unique TXN RECEIPTS: 24440
```

Out[4]:

	Number of Record in Dataset	Number of unique Receipts
0	2	23920
1	4	488
2	6	26
3	8	5
4	12	1

In [5]:
```python
# Filter the transactions dataset for duplicate RECEIPT_IDs
duplicate_txns = txns[txns['RECEIPT_ID'].isin(duplicate_receipts['RECEIPT_ID'])]

# Group the duplicate transactions by RECEIPT_ID
grouped_duplicate_txns = duplicate_txns.groupby('RECEIPT_ID')

# Counter to track the number of printed groups
count = 0

# Iterate through the groups and print/display the first 10 duplicates
for receipt_id, group in list(grouped_duplicate_txns):
    print(f"Receipt ID: {receipt_id}")
    display(group)  # This will show the grouped transactions for the current RECEIPT_ID
    print("\n")
    count += 1
    if count == 10:  # Stop after printing 10 groups
        break
```

```
Receipt ID: 0000d256-4041-4a3e-adc4-5623fb6e0c99
```

	RECEIPT_ID	PURCHASE_DATE	SCAN_DATE	STORE_NAME	USER_ID	BARCODE	FINAL_QUANTITY	FINAL_SALE
0	0000d256-4041-4a3e-adc4-5623fb6e0c99	2024-08-21	2024-08-21 14:19:06.539 Z	WALMART	63b73a7f3d310dceeabd4758	1.530001e+10	1.00	
41567	0000d256-4041-4a3e-adc4-5623fb6e0c99	2024-08-21	2024-08-21 14:19:06.539 Z	WALMART	63b73a7f3d310dceeabd4758	1.530001e+10	1.00	1.54

```
Receipt ID: 0001455d-7a92-4a7b-a1d2-c747af1c8fd3
```

	RECEIPT_ID	PURCHASE_DATE	SCAN_DATE	STORE_NAME	USER_ID	BARCODE	FINAL_QUANTITY	FINAL_SALE
1	0001455d-7a92-4a7b-a1d2-c747af1c8fd3	2024-07-20	2024-07-20 09:50:24.206 Z	ALDI	62c08877baa38d1a1f6c211a	NaN	zero	1.49
39291	0001455d-7a92-4a7b-a1d2-c747af1c8fd3	2024-07-20	2024-07-20 09:50:24.206 Z	ALDI	62c08877baa38d1a1f6c211a	NaN	1.00	1.49

```
Receipt ID: 00017e0a-7851-42fb-bfab-0baa96e23586
```

	RECEIPT_ID	PURCHASE_DATE	SCAN_DATE	STORE_NAME	USER_ID	BARCODE	FINAL_QUANTITY	FINAL_SALE
2	00017e0a-7851-42fb-bfab-0baa96e23586	2024-08-18	2024-08-19 15:38:56.813 Z	WALMART	60842f207ac8b7729e472020	7.874223e+10	1.00	
25928	00017e0a-7851-42fb-bfab-0baa96e23586	2024-08-18	2024-08-19 15:38:56.813 Z	WALMART	60842f207ac8b7729e472020	7.874223e+10	1.00	2.54

```
Receipt ID: 000239aa-3478-453d-801e-66a82e39c8af
```

	RECEIPT_ID	PURCHASE_DATE	SCAN_DATE	STORE_NAME	USER_ID	BARCODE	FINAL_QUANTITY	FINAL_SALE
3	000239aa-3478-453d-801e-66a82e39c8af	2024-06-18	2024-06-19 11:03:37.468 Z	FOOD LION	63fcd7cea4f8442c3386b589	7.833997e+11	zero	3.49
41475	000239aa-3478-453d-801e-66a82e39c8af	2024-06-18	2024-06-19 11:03:37.468 Z	FOOD LION	63fcd7cea4f8442c3386b589	7.833997e+11	1.00	3.49

```
Receipt ID: 00026b4c-dfe8-49dd-b026-4c2f0fd5c6a1
```

	RECEIPT_ID	PURCHASE_DATE	SCAN_DATE	STORE_NAME	USER_ID	BARCODE	FINAL_QUANTITY	FINAL_SALE
4	00026b4c-dfe8-49dd-b026-4c2f0fd5c6a1	2024-07-04	2024-07-05 15:56:43.549 Z	RANDALLS	6193231ae9b3d75037b0f928	4.790050e+10	1.00	
43233	00026b4c-dfe8-49dd-b026-4c2f0fd5c6a1	2024-07-04	2024-07-05 15:56:43.549 Z	RANDALLS	6193231ae9b3d75037b0f928	4.790050e+10	1.00	5.29

```
Receipt ID: 0002d8cd-1701-4cdd-a524-b70402e2dbc0
```

	RECEIPT_ID	PURCHASE_DATE	SCAN_DATE	STORE_NAME	USER_ID	BARCODE	FINAL_QUANTITY	FINAL_SALE
5	0002d8cd-1701-4cdd-a524-b70402e2dbc0	2024-06-24	2024-06-24 19:44:54.247 Z	WALMART	5dcc6c510040a012b8e76924	6.811314e+11	zero	1.46
40388	0002d8cd-1701-4cdd-a524-b70402e2dbc0	2024-06-24	2024-06-24 19:44:54.247 Z	WALMART	5dcc6c510040a012b8e76924	6.811314e+11	1.00	1.46

```
Receipt ID: 000550b2-1480-4c07-950f-ff601f242152
```

	RECEIPT_ID	PURCHASE_DATE	SCAN_DATE	STORE_NAME	USER_ID	BARCODE	FINAL_QUANTITY	FINAL_SALE
6	000550b2-1480-4c07-950f-ff601f242152	2024-07-06	2024-07-06 19:27:48.586 Z	WALMART	5f850bc9cf9431165f3ac175	4.920091e+10	1.00	
47862	000550b2-1480-4c07-950f-ff601f242152	2024-07-06	2024-07-06 19:27:48.586 Z	WALMART	5f850bc9cf9431165f3ac175	4.920091e+10	1.00	3.12

```
Receipt ID: 00096c49-8b04-42f9-88ce-941c5e06c4a7
```

	RECEIPT_ID	PURCHASE_DATE	SCAN_DATE	STORE_NAME	USER_ID	BARCODE	FINAL_QUANTITY	FINAL_SALE
7	00096c49-8b04-42f9-88ce-941c5e06c4a7	2024-08-19	2024-08-21 17:35:21.902 Z	TARGET	6144f4f1f3ef696919f54b5c	7.830007e+10	zero	3.59
36036	00096c49-8b04-42f9-88ce-941c5e06c4a7	2024-08-19	2024-08-21 17:35:21.902 Z	TARGET	6144f4f1f3ef696919f54b5c	7.830007e+10	1.00	3.59

```
Receipt ID: 000e1d35-15e5-46c6-b6b3-33653ed3d27e
```

	RECEIPT_ID	PURCHASE_DATE	SCAN_DATE	STORE_NAME	USER_ID	BARCODE	FINAL_QUANTITY	FINAL_SALE
8	000e1d35-15e5-46c6-b6b3-33653ed3d27e	2024-08-13	2024-08-13 18:21:07.931 Z	WALMART	61a6d926f998e47aad33db66	5.200001e+10	1.00	
41970	000e1d35-15e5-46c6-b6b3-33653ed3d27e	2024-08-13	2024-08-13 18:21:07.931 Z	WALMART	61a6d926f998e47aad33db66	5.200001e+10	1.00	0.98

```
Receipt ID: 0010d87d-1ad2-4e5e-9a25-cec736919d15
```

	RECEIPT_ID	PURCHASE_DATE	SCAN_DATE	STORE_NAME	USER_ID	BARCODE	FINAL_QUANTITY	FINAL_SALE
9	0010d87d-1ad2-4e5e-9a25-cec736919d15	2024-08-04	2024-08-04 18:01:47.787 Z	ALDI	66686fc2e04f743a096ea808	NaN	zero	2.29
40976	0010d87d-1ad2-4e5e-9a25-cec736919d15	2024-08-04	2024-08-04 18:01:47.787 Z	ALDI	66686fc2e04f743a096ea808	NaN	1.00	2.29

📊 Insights

1. **Duplicate Entries in the Transactions Table**
   The Transactions Table contains duplicate entries for every unique receipt or transaction in the dataset, with occurrences of **2, 4, 6, 8, or 12 duplicates** per transaction.

2. **Inconsistencies in Duplicate Transactions with Same RECEIPT_ID**
   For transactions with the same **RECEIPT_ID**, there are discrepancies in fields like `FINAL_QUANTITY` and `FINAL_SALE` :

   - In some cases, the `FINAL_QUANTITY` field differs between duplicates, with one entry showing the value *"zero"* (as a string), while another records are numeric value.
   - Similarly, the `FINAL_SALE` field varies, with some entries showing a valid sale amount, while others display *NA*.

---

2. Missing Values

Users Dataset

In [6]:
```
users.isna().sum()
```

Out[6]:
```
ID                0
CREATED_DATE      0
BIRTH_DATE     3675
STATE          4812
LANGUAGE      30508
GENDER         5892
dtype: int64
```

📊 Insights

- `BIRTH_DATE` : 3,675 missing values.
- `STATE` : 4,812 missing values, which accounts for a significant portion of users.
- `LANGUAGE` : 30,508 missing values—a large gap indicating issues with language preference collection.
- `GENDER` : 5,892 missing values.

Products Dataset

In [7]:
```
products.isna().sum()
```

Out[7]:
```
CATEGORY_1          111
CATEGORY_2         1424
CATEGORY_3        60566
CATEGORY_4       778093
MANUFACTURER     226474
BRAND            226472
BARCODE            4025
dtype: int64
```

📊 Insights

- `BARCODE` : 4,025 missing values (products without unique identifiers).
- `CATEGORY_1` to `CATEGORY_4` : Increasing missingness as we move deeper into categories (e.g., 778,093 missing in CATEGORY_4).
- `MANUFACTURER` and `BRAND` : Both missing for 226,474 products

The lack of barcodes and category data makes it impossible to uniquely identify a perticular product.

Transactions Dataset

In [8]:
```
txns.isna().sum()
```

Out[8]:
```
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
```

📊 Insights

- `BARCODE` : 5,762 missing values

The lack of barcodes in transactions data prevents linking a significant portion of transactions to products.

---

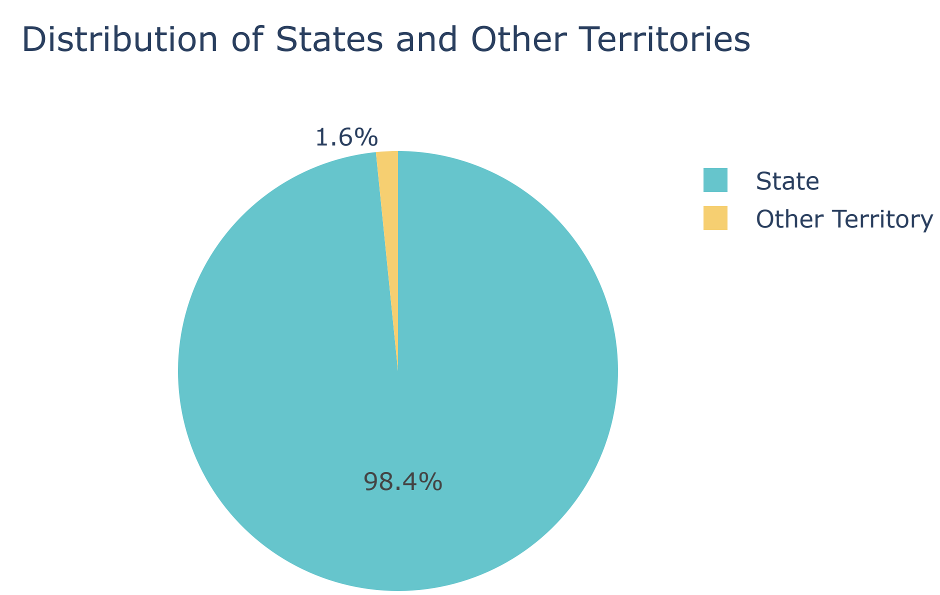
3. Inconsistent Data

STATE Field (Users Dataset)

In [9]:
```
# Get a list of unique states in the users dataset
unique_states = users[users['STATE'].notna()]['STATE'].unique()

# Number of unique states
print(f"No of Unique States in the Dataset: {len(unique_states)}")

# List of unique states
print(users[users['STATE'].notna()]['STATE'].unique())
```

```
No of Unique States in the Dataset: 52
['CA' 'PA' 'FL' 'NC' 'NY' 'IN' 'OH' 'TX' 'NM' 'PR' 'CO' 'AZ' 'RI' 'MO'
 'NJ' 'MA' 'TN' 'LA' 'NH' 'WI' 'IA' 'GA' 'VA' 'DC' 'KY' 'SC' 'MN' 'WV'
 'DE' 'MI' 'IL' 'MS' 'WA' 'KS' 'CT' 'OR' 'UT' 'MD' 'OK' 'NE' 'NV' 'AL'
 'AK' 'AR' 'HI' 'ME' 'ND' 'ID' 'WY' 'MT' 'SD' 'VT']
```

In [10]:
```python
# Create a new column to categorize as 'State' or 'Other Territory'
users['STATE_TYPE'] = users['STATE'].apply(lambda x: 'Other Territory' if x in ['PR', 'DC'] else 'State')

# Get the count of each category (State or Other Territory)
state_counts = users['STATE_TYPE'].value_counts().reset_index()
state_counts.columns = ['State/OT', 'Count']

# Create a pie chart
fig = px.pie(state_counts,
             names='State/OT',
             values='Count',
             title='Distribution of States and Other Territories',
             color_discrete_sequence=px.colors.qualitative.Pastel,
             width=500,  # Adjust width to tighten the figure
             height=400  # Adjust height accordingly
            )

# Show the pie chart
fig.show()
```



📊 Insights

- **The `STATE` attribute contains 52 unique values, including `DC` (District of Columbia) and `PR` (Puerto Rico), which are not U.S. states.**

DATE Fields (Transaction Dataset)

In [11]:
```python
txns[['RECEIPT_ID', 'STORE_NAME', 'PURCHASE_DATE', 'SCAN_DATE']].sample(2)
```

Out[11]:

	RECEIPT_ID	STORE_NAME	PURCHASE_DATE	SCAN_DATE
5975	3d057fb4-9e30-48e9-b90f-0d8dee2b2e11	COSTCO	2024-06-24	2024-07-04 10:09:09.201 Z
2213	16c9c13c-df14-4504-b802-108742e6a618	WALGREENS	2024-07-29	2024-07-29 21:06:17.211 Z

📊 Insights

- **The `PURCHASE_DATE` field uses the `MM/DD/YYYY` format, while the `SCAN_DATE` field follows the ISO-8601 format with a timestamp (e.g., `YYYY-MM-DDTHH:MM:SSZ`).**

GENDER Field (Users Table)

In [12]:
```
users['GENDER'].value_counts()
```

Out[12]:
```
GENDER
female                    64240
male                      25829
transgender                1772
prefer_not_to_say          1350
non_binary                  473
unknown                     196
not_listed                  180
Non-Binary                   34
not_specified                28
My gender isn't listed        5
Prefer not to say             1
Name: count, dtype: int64
```

📊 Insights

- **Inconsistent variations exist for the same categories, such as:**
  - *Non-Binary* → `Non-Binary`, `non_binary`, `not_listed`, and `"My gender isn't listed"`.
  - *Prefer not to say* → `Prefer not to say`, `prefer_not_to_say`.

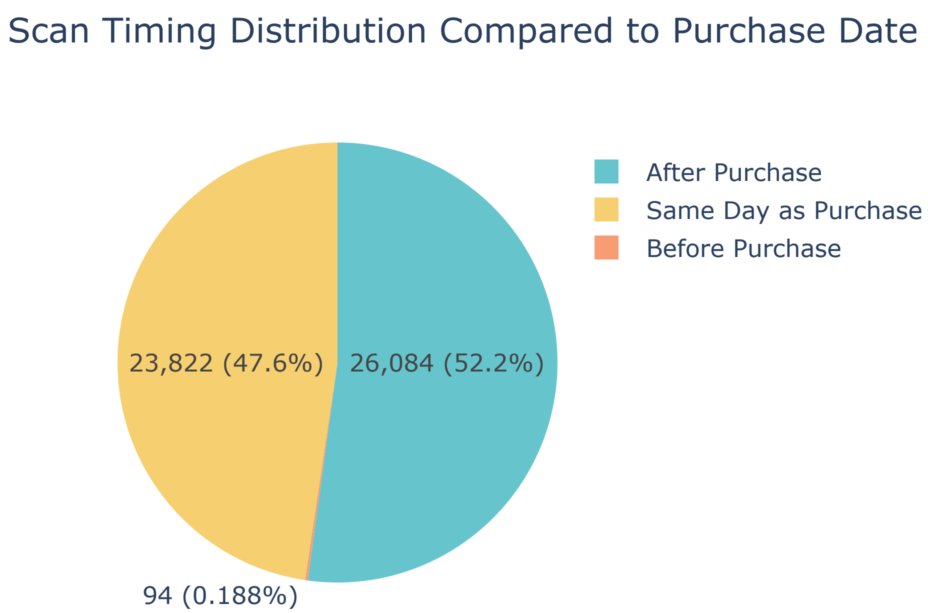
---

4. Outliers

SCAN_DATE & *PURCHASE_DATE* Fields (Transactions Dataset)

In [13]:
```
txns_cpy = txns.copy()
# Convert dates to datetime format
txns_cpy['PURCHASE_DATE'] = pd.to_datetime(txns_cpy['PURCHASE_DATE'], format='mixed')
txns_cpy['SCAN_DATE'] = pd.to_datetime(txns_cpy['SCAN_DATE'])

# Remove timezone from SCAN_DATE to make it same as PURCHASE_DATE
txns_cpy['SCAN_DATE'] = pd.to_datetime(txns['SCAN_DATE']).dt.tz_localize(None)

# Calculate the difference in days
txns_cpy['DATE_DIFFERENCE'] = (txns_cpy['SCAN_DATE'] - txns_cpy['PURCHASE_DATE']).dt.days

# Categorize the transactions
def categorize_scan(diff):
    if diff < 0:
        return 'Before Purchase'
    elif diff == 0:
        return 'Same Day as Purchase'
    else:
        return 'After Purchase'

txns_cpy['SCAN_CATEGORY'] = txns_cpy['DATE_DIFFERENCE'].apply(categorize_scan)
```

```python
# Count the occurrences of each category
scan_counts = txns_cpy['SCAN_CATEGORY'].value_counts().reset_index()
scan_counts.columns = ['Scan Category', 'Count']

# Create a pie chart
fig = px.pie(scan_counts,
             names='Scan Category',
             values='Count',
             title='Scan Timing Distribution Compared to Purchase Date',
             color_discrete_sequence=px.colors.qualitative.Pastel,
             width=500,  # Adjust width to tighten the figure
             height=400  # Adjust height accordingly
            )

# show count and percentage both in labels
fig.update_traces(textinfo='percent+value',
                  texttemplate='%{value} (%{percent})',
                  insidetextorientation='radial')

# Display the pie chart
fig.show()
```



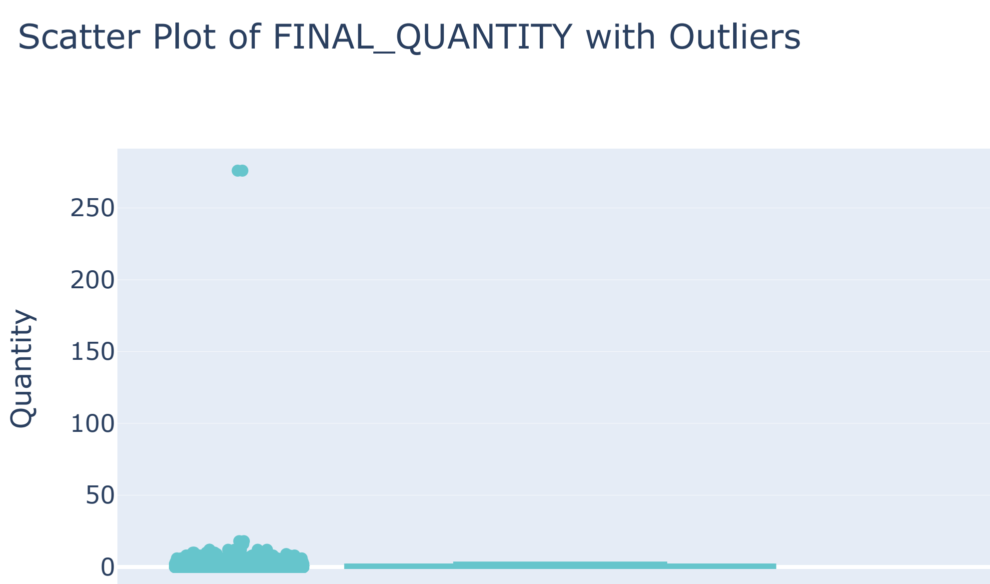
📊 Insights

- There are **94 records** (0.188% of the dataset) where the `SCAN_DATE` occurs **before** the `PURCHASE_DATE`. This inconsistency is unusual, as a product is generally expected to be scanned only after it is purchased.

FINAL_QUANTITY Field (Transactions Dataset)

In [14]:
```python
# Convert 'zero' string to 0 in FINAL_QUANITY field of Transactions dataset
txns_copy = txns.copy()
txns_copy['FINAL_QUANTITY'] = txns_copy['FINAL_QUANTITY'].replace("zero", 0)
txns_copy['FINAL_QUANTITY'] = txns_copy['FINAL_QUANTITY'].astype(float)
# Box plot to visualize the outliers in FINAL_QUANTITY
fig = px.box(txns_copy,
             y='FINAL_QUANTITY',
```

```
            title="Scatter Plot of FINAL_QUANTITY with Outliers",
            labels={'FINAL_QUANTITY': 'Quantity'},
            color_discrete_sequence=px.colors.qualitative.Pastel,
            points='all',
            width=600,  # Adjust width to tighten the figure
            height=400  # Adjust height accordingly
           )

fig.show()
```



In [15]:
```
# Display Transactions with Quanity 276 (i.e Outliers)
txns[txns['FINAL_QUANTITY']=='276.00'][["RECEIPT_ID","STORE_NAME","USER_ID", "BARCODE", "FINAL_QUANTITY", "FINAL_SALE"]]
```

Out[15]:

	RECEIPT_ID	STORE_NAME	USER_ID	BARCODE	FINAL_QUANTITY	FINAL_SALE
24810	fe0780d1-2d02-4822-8f12-7056b1814f17	MAIN STREET MARKET	5d197f9dd08976510c49d0e6	4.800135e+10	276.00	
42410	fe0780d1-2d02-4822-8f12-7056b1814f17	MAIN STREET MARKET	5d197f9dd08976510c49d0e6	4.800135e+10	276.00	5.89

📊 Insights

- There are **two identical transactions** at **MAIN STREET MARKET** with unusually high `FINAL_QUANTITY` values of **276**, which appear to be potential **outliers**.

Q2: Are there any fields that are challenging to understand?

In [16]:
```
# Display transactions with "zero" FINAL_QUANTITY
txns[txns['FINAL_QUANTITY']=="zero"].head(2)
```

Out[16]:

	RECEIPT_ID	PURCHASE_DATE	SCAN_DATE	STORE_NAME	USER_ID	BARCODE	FINAL_QUANTITY	FINAL_SALE
1	0001455d-7a92-4a7b-a1d2-c747af1c8fd3	2024-07-20	2024-07-20 09:50:24.206 Z	ALDI	62c08877baa38d1a1f6c211a	NaN	zero	1.49
3	000239aa-3478-453d-801e-66a82e39c8af	2024-06-18	2024-06-19 11:03:37.468 Z	FOOD LION	63fcd7cea4f8442c3386b589	7.833997e+11	zero	3.49

In [17]:

```
users[['ID','LANGUAGE']].head()
```

Out[17]:

	ID	LANGUAGE
0	5ef3b4f17053ab141787697d	es-419
1	5ff220d383fcfc12622b96bc	en
2	6477950aa55bb77a0e27ee10	es-419
3	658a306e99b40f103b63ccf8	en
4	653cf5d6a225ea102b7ecdc2	en

📊 Insights

Some fields in the dataset have logical inconsistencies that make them challenging to understand. For example:

- **Transactions where `FINAL_QUANTITY = 0` but `FINAL_SALE` is non-zero** are illogical, as a non-zero sale amount should typically correspond to a positive quantity.
- **The `LANGUAGE` field** in **USERS dataset** contains codes such as `en`, `es-419`, but the context is unclear. It is not clear if this represents the preferred language of users, the language in which receipts are submitted, or something else.

One Interesting Insights

Month-over-Month (MoM) growth of active user

In [18]:

```
txns['SCAN_DATE'] = pd.to_datetime(txns['SCAN_DATE'], format='mixed')

#  Extract Year and Month from PURCHASE_DATE
txns['YEAR_MONTH'] = txns['SCAN_DATE'].dt.to_period('M')

# Group by YEAR_MONTH and count distinct active users
active_users_per_month = (
    txns.groupby('YEAR_MONTH')['USER_ID']
    .nunique()
    .reset_index()
    .rename(columns={'USER_ID': 'ACTIVE_USERS'})
)

# Calculate Month-over-Month (MoM) Growth
active_users_per_month['PREVIOUS_MONTH_USERS'] = active_users_per_month['ACTIVE_USERS'].shift(1)
active_users_per_month['MOM_GROWTH_PERCENTAGE'] = (
    (active_users_per_month['ACTIVE_USERS'] - active_users_per_month['PREVIOUS_MONTH_USERS'])
    / active_users_per_month['PREVIOUS_MONTH_USERS'] * 100
)

# Drop rows without growth data (first month will have NaN for growth percentage)
```

```python
# active_users_per_month = active_users_per_month.dropna(subset=['MOM_GROWTH_PERCENTAGE'])

# Sort results by most recent month
active_users_per_month = active_users_per_month.sort_values(by='YEAR_MONTH')

active_users_per_month
```

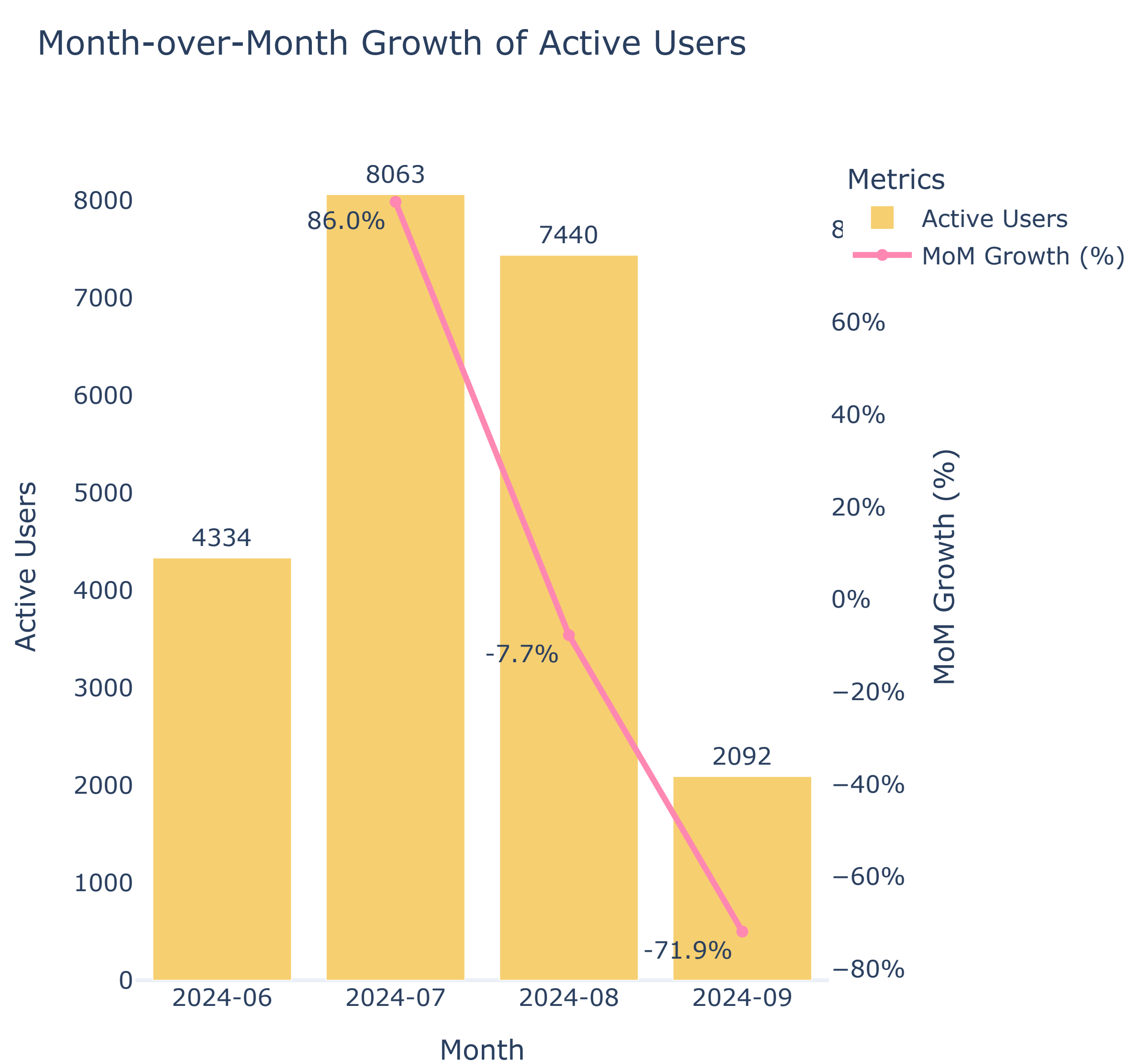
Out[18]:

	YEAR_MONTH	ACTIVE_USERS	PREVIOUS_MONTH_USERS	MOM_GROWTH_PERCENTAGE
0	2024-06	4334	NaN	NaN
1	2024-07	8063	4334.0	86.040609
2	2024-08	7440	8063.0	-7.726653
3	2024-09	2092	7440.0	-71.881720

In [19]:

```python
# Convert YEAR_MONTH from Period to timestamp and then to string for visualization
active_users_per_month['YEAR_MONTH'] = active_users_per_month['YEAR_MONTH'].dt.to_timestamp().dt.strftime('%Y-%m')

# Create figure with dual y-axes
fig = go.Figure()

# Add bar chart for total active users on left y-axis
fig.add_trace(go.Bar(
    x=active_users_per_month['YEAR_MONTH'],
    y=active_users_per_month['ACTIVE_USERS'],
    name='Active Users',
    marker_color=px.colors.qualitative.Pastel[1],
    text=active_users_per_month['ACTIVE_USERS'],
    textposition='outside',
    yaxis='y1'  # Assign to primary y-axis (left)
))

# Add line chart for MoM growth percentage on right y-axis
fig.add_trace(go.Scatter(
    x=active_users_per_month['YEAR_MONTH'],
    y=active_users_per_month['MOM_GROWTH_PERCENTAGE'],
    name='MoM Growth (%)',
    mode='lines+markers+text',
    text=[f'{x:.1f}%' for x in active_users_per_month['MOM_GROWTH_PERCENTAGE']],
    textposition='bottom left',
    line=dict(color=px.colors.qualitative.Pastel[6], width=3),
    yaxis='y2'  # Assign to secondary y-axis (right)
))

# Add labels and format the graph
fig.update_layout(
    title='Month-over-Month Growth of Active Users',
    height = 600,
    width = 600,
    xaxis=dict(
        title='Month',
        type='category',
        categoryorder='array',
        categoryarray=active_users_per_month['YEAR_MONTH'].tolist()
    ),
    yaxis=dict(
        title='Active Users',
        side='left',
        showgrid=False
```

```
    ),
    yaxis2=dict(
        title='MoM Growth (%)',
        side='right',
        overlaying='y',
        showgrid=False,
        ticksuffix='%',
        zeroline=False
    ),
    legend=dict(title='Metrics'),
    template='plotly_white'
)

fig.show()
```



📊 Insights

1. Strong Growth in July:

- Active users surged by ***86%***, from **4,334** to **8,063**

2. Moderate Decline in August:

- User count dropped by ***7.7%***

3. Sharp Drop in September:

- A significant ***71.9%*** decline to **2,092** users