FORAGE

datalab

PART I: Transaction Exploration Data Analysis

```
import pandas as pd
import os

# Check if the file exists before attempting to read
file_path = 'transaction.xlsx'
transaction = pd.read_excel(file_path)
# Display the first few rows of the dataframe
transaction.head()
```

i...	DATE	STOR...	LYLTY_CARD_NBR	T...	PR...	PROD_NAME
0	2018-10-17T00:00:00.000	1	1000	1	5	Natural Chip
1	2019-05-14T00:00:00.000	1	1307	348	66	CCs Nacho C
2	2019-05-20T00:00:00.000	1	1343	383	61	Smiths Crinkl
3	2018-08-17T00:00:00.000	2	2373	974	69	Smiths Chip T
4	2018-08-18T00:00:00.000	2	2426	1038	108	Kettle Tortilla

Rows: 5 Expand

```
# Check for null values in each column of the transaction dataframe
transaction.isnull().sum()
transaction.shape
```

```
(264836, 8)
```

```
# Strip leading and trailing spaces from all string columns in the transaction dataframe
str_cols = transaction.select_dtypes(include=['object']).columns
transaction[str_cols] = transaction[str_cols].apply(lambda x: x.str.strip())
transaction.head()
```

i...	DATE	STOR...	LYLTY_CARD_NBR	T...	PR...	PROD_NAME
0	2018-10-17T00:00:00.000	1	1000	1	5	Natural Chip
1	2019-05-14T00:00:00.000	1	1307	348	66	CCs Nacho C
2	2019-05-20T00:00:00.000	1	1343	383	61	Smiths Crinkl
3	2018-08-17T00:00:00.000	2	2373	974	69	Smiths Chip T
4	2018-08-18T00:00:00.000	2	2426	1038	108	Kettle Tortilla

Rows: 5 Expand

```
# Display the unique values in the 'PROD_NAME' column
#transaction['PROD_NAME'].unique()
```

```
# Change all column names in the transaction dataframe to lowercase
transaction.columns = transaction.columns.str.lower()
transaction.head()
```

i...	date	store...	lylty_card_nbr	t...	pro...	prod_name
0	2018-10-17T00:00:00.000	1	1000	1	5	Natural Chip
1	2019-05-14T00:00:00.000	1	1307	348	66	CCs Nacho C
2	2019-05-20T00:00:00.000	1	1343	383	61	Smiths Crinkl
3	2018-08-17T00:00:00.000	2	2373	974	69	Smiths Chip T
4	2018-08-18T00:00:00.000	2	2426	1038	108	Kettle Tortilla

Rows: 5 Expand

```
#Rename column
transaction = transaction.rename(columns={
    "date": "date",
    "store_nbr": "store_number",
    "lylty_card_nbr": "loyalty_card_number",
    "txn_id": "transaction_id",
    "prod_nbr": "product_number",
    "prod_name": "product",
    "prod_qty": "quantity",
    "tot_sales": "total_sales"
})
transaction.head()
```

	date	store_n...	loyalty_card_number	transaction...	product_nu...
0	2018-10-17T00:00:00.000	1	1000	1	5
1	2019-05-14T00:00:00.000	1	1307	348	66
2	2019-05-20T00:00:00.000	1	1343	383	61
3	2018-08-17T00:00:00.000	2	2373	974	69
4	2018-08-18T00:00:00.000	2	2426	1038	108

Rows: 5 Expand

```
transaction['date'] = pd.to_datetime(transaction['date'])
transaction.dtypes
```

```
date                  datetime64[ns]
store_number                   int64
loyalty_card_number            int64
transaction_id                 int64
product_number                 int64
product                       object
quantity                       int64
total_sales                  float64
dtype: object
```

1 hidden cell

```
import re
from rapidfuzz import process, fuzz

# Step 1: remove pack size, normalize casing
transaction["pack_size"] = transaction["product"].str.extract(r"(\d+\s?[gG])")
transaction["product"] = (
    transaction["product"]
    .str.replace(r"\d+\s?[gG]", "", regex=True)
    .str.replace(r"\s+", " ", regex=True)
    .str.strip()
    .str.title()
)

# Step 6: final tidy
transaction["product"] = transaction["product"].str.replace(r"\s*&\s*", " & ", regex=True)
transaction["product"] = transaction["product"].str.replace(r"\s+", " ", regex=True).str.strip()
```

```
transaction.head()
```

	date	store_...	loyalty_card_numb...	transactio...	product_n...	product
0	2018-10-17T00:00:00.000	1	1000	1	5	Natural C
1	2019-05-14T00:00:00.000	1	1307	348	66	Ccs Nach
2	2019-05-20T00:00:00.000	1	1343	383	61	Smiths Cr
3	2018-08-17T00:00:00.000	2	2373	974	69	Smiths Cl
4	2018-08-18T00:00:00.000	2	2426	1038	108	Kettle Tor

Rows: 5 Expand

```
#transaction["product"].unique()
```

```
# Step 1: lowercase and strip
transaction["product"] = transaction["product"].str.lower().str.strip()
transaction["pack_size"] = transaction["pack_size"].str.lower().str.strip()
```

1 hidden cell

```
transaction.head()
```

	date	store...	loyalty_card_nu...	transact...	product...	product
0	2018-10-17T00:00:00.000	1	1000	1	5	Natural Chip Comp
1	2019-05-14T00:00:00.000	1	1307	348	66	CCs Nacho Chees
2	2019-05-20T00:00:00.000	1	1343	383	61	Smiths Crinkle Cut
3	2018-08-17T00:00:00.000	2	2373	974	69	Smiths Chip Thinly
4	2018-08-18T00:00:00.000	2	2426	1038	108	Kettle Tortilla Chip

Rows: 5 Expand

PART 1: Data cleaning and standardization Normalize product names by lowercasing and stripping spaces.

- Use a mapping dictionary to replace messy and abbreviated names with clean, standardized product names (e.g. "dorito corn chp supreme" becomes "Doritos Corn Chips Supreme").
- Clean the pack_size column in the same way (lowercase + strip).
- Final cleanup pass on product names to ensure consistency.
- **Purpose:** This step ensures all product names consistent and standardised before analysis.

PART 2: Product categorisation

- Define a type_map dictionary that links keywords (e.g. "doritos",
- "cheezels", "salsa") to broader product categories (e.g. "Corn Chips", "Snacks", "Salsa").
- Create a detect_type() function that:
- Handles special cases for salsa/dip (distinguishes Tomato Salsa vs Other Salsa).
- Otherwise checks product names against type_map keywords.
- Defaults to "Other" if no match is found. Apply detect_type() to the cleaned product column to create a new **"category"** column in the dataset.
- **Purpose:** This step classifies each product into a broader category so analysis can be done at both product and category levels.

```
cols = list(transaction.columns)
# Remove 'category' and 'pack_size' from their current positions
cols.remove('brand')
cols.remove('pack_size')
# Find the index of 'product'
product_idx = cols.index('product')
# Insert 'category' and 'pack_size' after 'product'
cols = cols[:product_idx+1] + ['type', 'brand' , 'pack_size'] + cols[product_idx+1:]
transaction[cols].head(10)
```

	date	store...	loyalty_card_nu...	transact...	product...	product
0	2018-10-17T00:00:00.000	1	1000	1	5	Natural Chip Comp
1	2019-05-14T00:00:00.000	1	1307	348	66	CCs Nacho Chees
2	2019-05-20T00:00:00.000	1	1343	383	61	Smiths Crinkle Cut
3	2018-08-17T00:00:00.000	2	2373	974	69	Smiths Chip Thinly
4	2018-08-18T00:00:00.000	2	2426	1038	108	Kettle Tortilla Chip
5	2019-05-19T00:00:00.000	4	4074	2982	57	Old El Paso Salsa D
6	2019-05-16T00:00:00.000	4	4149	3333	16	Smiths Crinkle Cut
7	2019-05-16T00:00:00.000	4	4196	3539	24	Grain Waves Swee
8	2018-08-20T00:00:00.000	5	5026	4525	42	Doritos Corn Chips
9	2018-08-18T00:00:00.000	7	7150	6900	52	Grain Waves Sour

Rows: 10 Expand

```
# Create the 'price' column as total_sales divided by quantity
transaction['price'] = transaction['total_sales'] / transaction['quantity']

# Reorder columns: insert 'price' before 'quantity'
cols = list(transaction.columns)
cols.remove('price')
quantity_idx = cols.index('quantity')
cols = cols[:quantity_idx] + ['price'] + cols[quantity_idx:]

transaction[cols].head(10)
```

	date	store...	loyalty_card_nu...	transact...	product...	product
0	2018-10-17T00:00:00.000	1	1000	1	5	Natural Chip Comp
1	2019-05-14T00:00:00.000	1	1307	348	66	CCs Nacho Chees
2	2019-05-20T00:00:00.000	1	1343	383	61	Smiths Crinkle Cut
3	2018-08-17T00:00:00.000	2	2373	974	69	Smiths Chip Thinly
4	2018-08-18T00:00:00.000	2	2426	1038	108	Kettle Tortilla Chip
5	2019-05-19T00:00:00.000	4	4074	2982	57	Old El Paso Salsa D
6	2019-05-16T00:00:00.000	4	4149	3333	16	Smiths Crinkle Cut
7	2019-05-16T00:00:00.000	4	4196	3539	24	Grain Waves Swee
8	2018-08-20T00:00:00.000	5	5026	4525	42	Doritos Corn Chips
9	2018-08-18T00:00:00.000	7	7150	6900	52	Grain Waves Sour

Rows: 10 Expand

```
import pandas as pd

# Example: your transaction DataFrame
df = transaction.copy()

def flag_outliers(series):
    Q1 = series.quantile(0.25)
    Q3 = series.quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5*IQR
    upper = Q3 + 1.5*IQR
    return (series < lower) | (series > upper)

df['outlier_quantity'] = flag_outliers(df['quantity'])
df['outlier_price'] = flag_outliers(df['price'])
df['outlier_sales'] = flag_outliers(df['total_sales'])

# final flag: any of the three
df['is_outlier'] = df[['outlier_quantity','outlier_price','outlier_sales']].any(axis=1)

# show strong outliers, ordered by quantity, price, total_sales descending
strong_outliers = df[df['is_outlier']]
strong_outliers_sorted = strong_outliers.sort_values(
    by=['quantity', 'price', 'total_sales'], ascending=[False, False, False]
)
strong_outliers_sorted[['product','quantity','price','total_sales']].head()
```

index	product	quantity
69762	Doritos Corn Chips Supreme	
69763	Doritos Corn Chips Supreme	
5179	Smiths Crinkle Cut Original Big Bag	
55558	Smiths Crinkle Cut Original Big Bag	
69496	Smiths Crinkle Cut Original Big Bag	

Rows: 5 Expand

```
# Fix: Check if 'is_outlier' exists, if not, create it (example: using z-score on 'total_sales')
import numpy as np

if 'is_outlier' not in transaction.columns:
    # Example: mark as outlier if total_sales is more than 3 std from mean
    z_scores = (transaction['total_sales'] - transaction['total_sales'].mean()) / transaction['total_sales'].std()
    transaction['is_outlier'] = np.abs(z_scores) > 3

outliers = transaction[transaction['is_outlier']]
top_outliers = outliers.sort_values('total_sales', ascending=False).head(10)
top_outliers[['product','quantity','price','total_sales']]
```

index	product	quantity
69762	Doritos Corn Chips Supreme	
69763	Doritos Corn Chips Supreme	
117850	Smiths Crinkle Cut Original Big Bag	
5179	Smiths Crinkle Cut Original Big Bag	
69496	Smiths Crinkle Cut Original Big Bag	
55558	Smiths Crinkle Cut Original Big Bag	
171815	Smiths Crinkle Cut Original Big Bag	
150683	Smiths Crinkle Cut Original Big Bag	
184969	Smiths Crinkle Cut Original Big Bag	
117917	Smiths Crinkle Cut Chips Salt & Vinegar	

Rows: 10 Expand

1 value in particular is anormal **200** as it is very high compare to other quantity

```
# Calculate the median of total_sales
median = transaction['total_sales'].median()
# Calculate how many times each top_outlier's total_sales is to the median, rounded to 2 decimals
top_outliers = top_outliers.copy() #top 10 outlier
top_outliers['relative_to_median'] = (top_outliers['total_sales'] / median).round(2)
# Add a column with the median value for reference, rounded to 2 decimals
top_outliers['median'] = round(median, 2)
# Also round total_sales to 2 decimals for display
top_outliers['total_sales'] = top_outliers['total_sales'].round(2)
# Display the relevant columns
top_outliers[['product', 'total_sales', 'median', 'relative_to_median']]
```

index	product	total_sales
69762	Doritos Corn Chips Supreme	65
69763	Doritos Corn Chips Supreme	65
117850	Smiths Crinkle Cut Original Big Bag	29
5179	Smiths Crinkle Cut Original Big Bag	29
69496	Smiths Crinkle Cut Original Big Bag	29
55558	Smiths Crinkle Cut Original Big Bag	29
171815	Smiths Crinkle Cut Original Big Bag	29
150683	Smiths Crinkle Cut Original Big Bag	29
184969	Smiths Crinkle Cut Original Big Bag	29
117917	Smiths Crinkle Cut Chips Salt & Vinegar	28

Rows: 10 Expand

```
# Group by product and keep max relative_to_median
summary = (
    top_outliers.groupby(['product', 'quantity', 'price'])
    .agg({'total_sales': 'max', 'median': 'max', 'relative_to_median': 'max'})
    .reset_index()
    .sort_values('relative_to_median', ascending=False)
)

summary[['product', 'quantity', 'price', 'total_sales', 'median', 'relative_to_median']]
```

index	product	quantity	price	total_sales
0	Doritos Corn Chips Supreme	200	3.25	
2	Smiths Crinkle Cut Original Big Bag	5	5.9	
1	Smiths Crinkle Cut Chips Salt & Vinegar	5	5.7	

Rows: 3 Expand

This table is concise and shows exactly what you need:

- One extreme anomaly (Doritos at 650 sales, ~88× median).
- Three moderate anomalies (Smiths and Cheezels at ~29 sales, ~4× median).
- Doritos seems to be extremly high so we drop that column

```
# Remove rows where quantity == 200 (the outlier) and drop the 'date' and 'is_outlier' columns
filtered_transaction = transaction[transaction['quantity'] != 200].drop(columns='is_outlier')
filtered_transaction.to_csv('transaction.csv',index=False)
```

PART 2: Customer Behaviour EDA

```
import pandas as pd
behaviour = pd.read_csv('behaviour.csv')
behaviour.head()
```

	LYLTY_C...	LIFESTAGE	PREMIUM_...
0	1000	YOUNG SINGLES/COUPLES	Premium
1	1002	YOUNG SINGLES/COUPLES	Mainstream
2	1003	YOUNG FAMILIES	Budget
3	1004	OLDER SINGLES/COUPLES	Mainstream
4	1005	MIDAGE SINGLES/COUPLES	Mainstream

Rows: 5 Expand

```
# Change all column names in the transaction dataframe to lowercase
behaviour.columns = behaviour.columns.str.lower()
# Rename 'premium_customer' to 'customer_segment' and 'lylty_card_nbr' to 'loyalty_card_number' for clarity
behaviour = behaviour.rename(columns={
    'premium_customer': 'customer_segment',
    'lylty_card_nbr': 'loyalty_card_number'
})
behaviour.head()
```

	loyalty_card_nu...	lifestage	customer_s...
0	1000	YOUNG SINGLES/COUPLES	Premium
1	1002	YOUNG SINGLES/COUPLES	Mainstream
2	1003	YOUNG FAMILIES	Budget
3	1004	OLDER SINGLES/COUPLES	Mainstream
4	1005	MIDAGE SINGLES/COUPLES	Mainstream

Rows: 5 Expand

```
def map_situation(lifestage):
    if "SINGLES/COUPLES" in lifestage:
        return "Singles/Couples"
    elif "FAMILIES" in lifestage:
        return "Families"
    elif "RETIREES" in lifestage:
        return "Retirees"
    else:
        return "Other"

def map_age_category(lifestage):
    if "YOUNG" in lifestage or "NEW FAMILIES" in lifestage:
        return "Young"
    elif "MIDAGE" in lifestage:
        return "Midage"
    elif "OLDER" in lifestage:
        return "Older"
    elif "RETIREES" in lifestage:
        return "Retirees"
    else:
        return "Other"

behaviour['age_category'] = behaviour['lifestage'].apply(map_age_category)
behaviour['household_type'] = behaviour['lifestage'].apply(map_situation)
behaviour.head()
```

	loyalty_card_nu...	lifestage	customer_s...	age_...	household...
0	1000	YOUNG SINGLES/COUPLES	Premium	Young	Singles/Couples
1	1002	YOUNG SINGLES/COUPLES	Mainstream	Young	Singles/Couples
2	1003	YOUNG FAMILIES	Budget	Young	Families
3	1004	OLDER SINGLES/COUPLES	Mainstream	Older	Singles/Couples
4	1005	MIDAGE SINGLES/COUPLES	Mainstream	Midage	Singles/Couples

Rows: 5 Expand

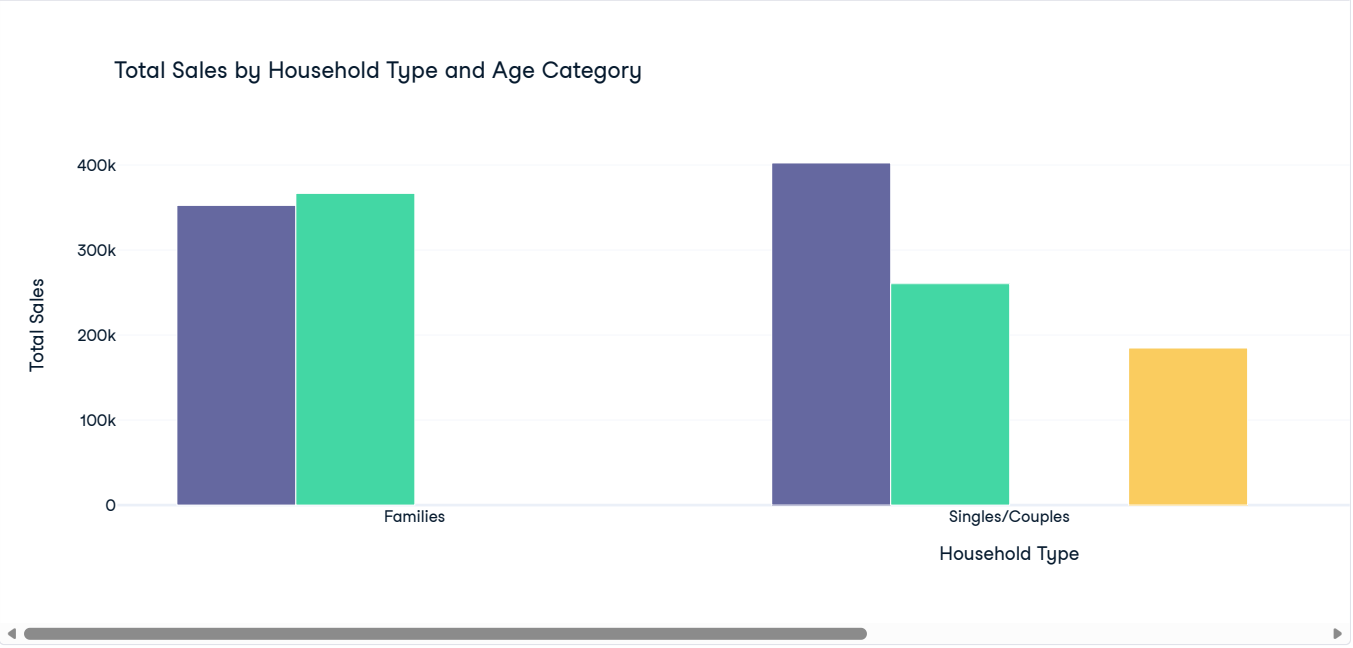
```
behaviour.drop(columns=['lifestage']).to_csv('customers.csv', index=False)
```

```
# Read in the transaction data
transactions = pd.read_csv('transaction.csv')

# Read in the customers data (created in previous steps)
customers = pd.read_csv('customers.csv')

# Ensure the join column is lowercase in both dataframes
transactions.columns = transactions.columns.str.lower()
if 'lylty_card_nbr' in transactions.columns:
    transactions = transactions.rename(columns={'lylty_card_nbr': 'loyalty_card_number'})

# Perform a left join on 'loyalty_card_number'
merged_df = transactions.merge(customers, on='loyalty_card_number', how='left')

# Display the first few rows of the merged dataframe
merged_df.head()
```

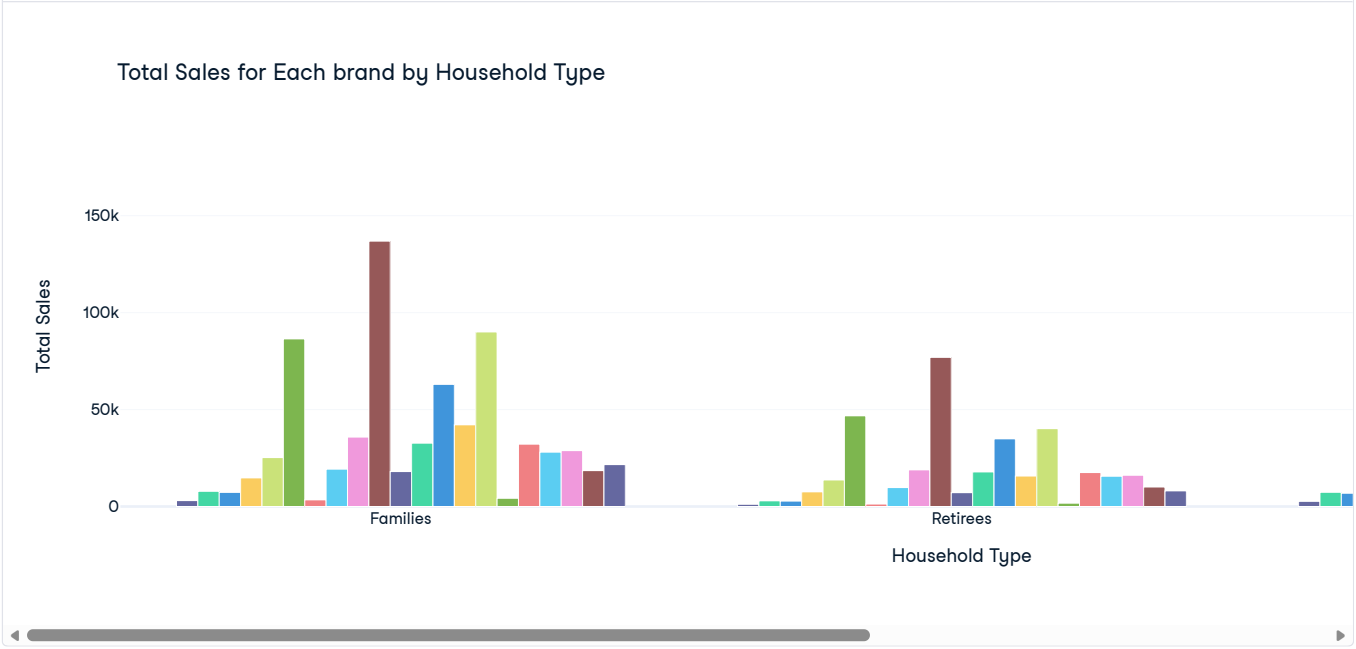
	d...	store...	loyalty_card_nu...	transact...	product...	product
0	2018-10-17	1	1000	1	5	Natural Chip Company Sea Salt
1	2019-05-14	1	1307	348	66	CCs Nacho Cheese
2	2019-05-20	1	1343	383	61	Smiths Crinkle Cut Chips Chicken
3	2018-08-17	2	2373	974	69	Smiths Chip Thinly Sour Cream & Onion
4	2018-08-18	2	2426	1038	108	Kettle Tortilla Chips Honey & Jalapeno C

Rows: 5 Expand

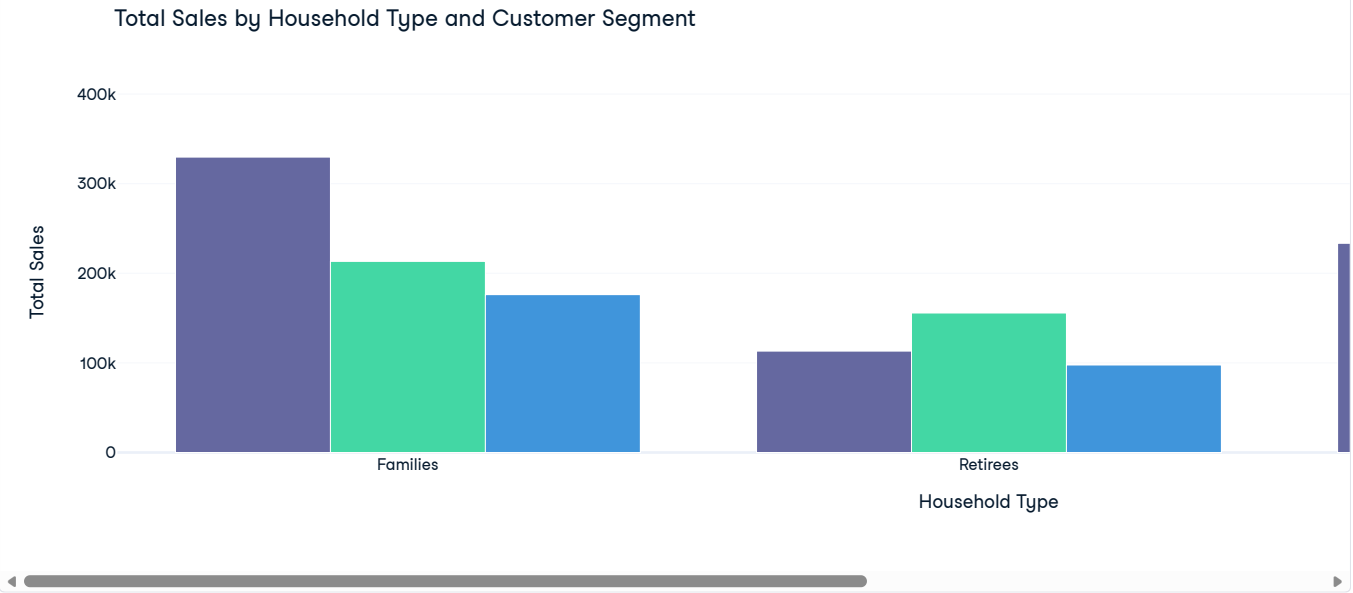
```
import plotly.express as px

# Group by 'household_type' and 'age_category', then sum 'total_sales'
sales_by_household_age = merged_df.groupby(['household_type', 'age_category'])['total_sales'].sum().reset_index()

# Create an interactive bar chart
fig = px.bar(
    sales_by_household_age,
    x='household_type',
    y='total_sales',
    color='age_category',
    barmode='group',
    title='Total Sales by Household Type and Age Category',
    labels={'total_sales': 'Total Sales', 'household_type': 'Household Type', 'age_category': 'Age Category'}
)

fig.show()
```



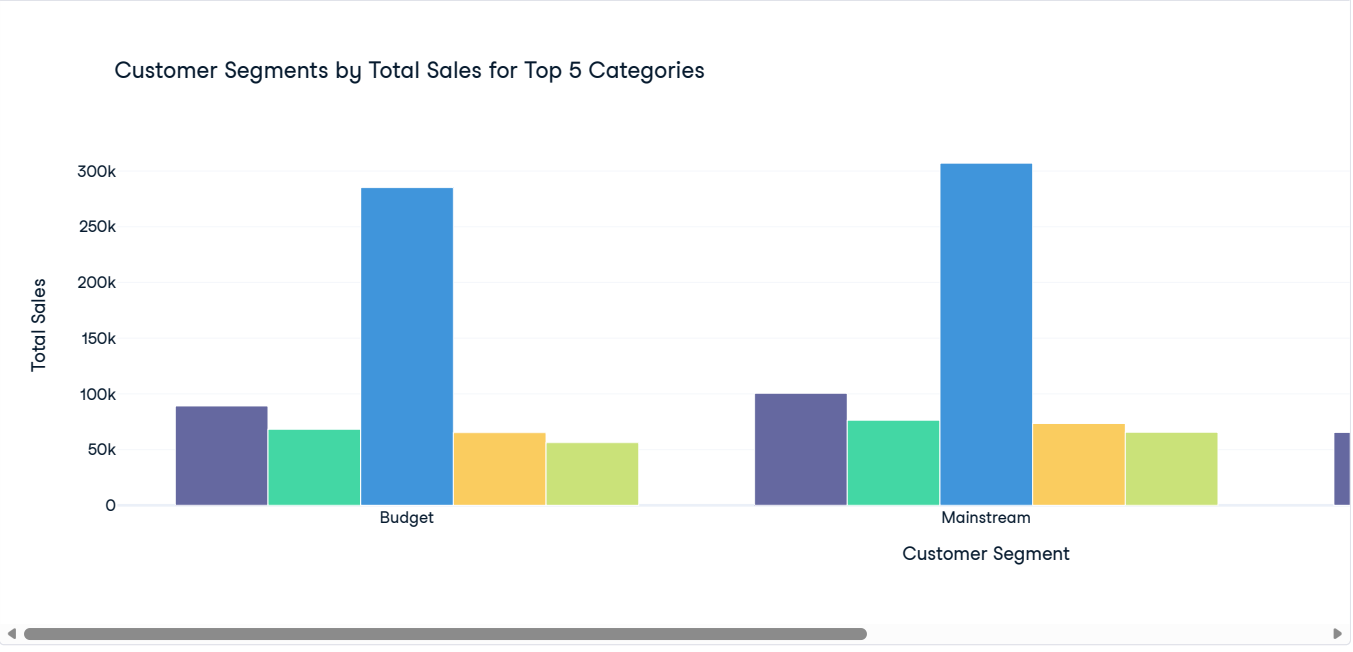
```
import plotly.express as px

# Group by 'household_type' and 'category', then sum 'total_sales'
sales_by_household_category = (
    merged_df.groupby(['household_type', 'brand'])['total_sales']
    .sum()
    .reset_index()
)

# Create an interactive bar chart
fig = px.bar(
    sales_by_household_category,
    x='household_type',
    y='total_sales',
    color='brand',
    barmode='group',
    title='Total Sales for Each brand by Household Type',
    labels={
        'household_type': 'Household Type',
        'total_sales': 'Total Sales',
        'brand': 'Brand'
    }
)

fig.show()
```



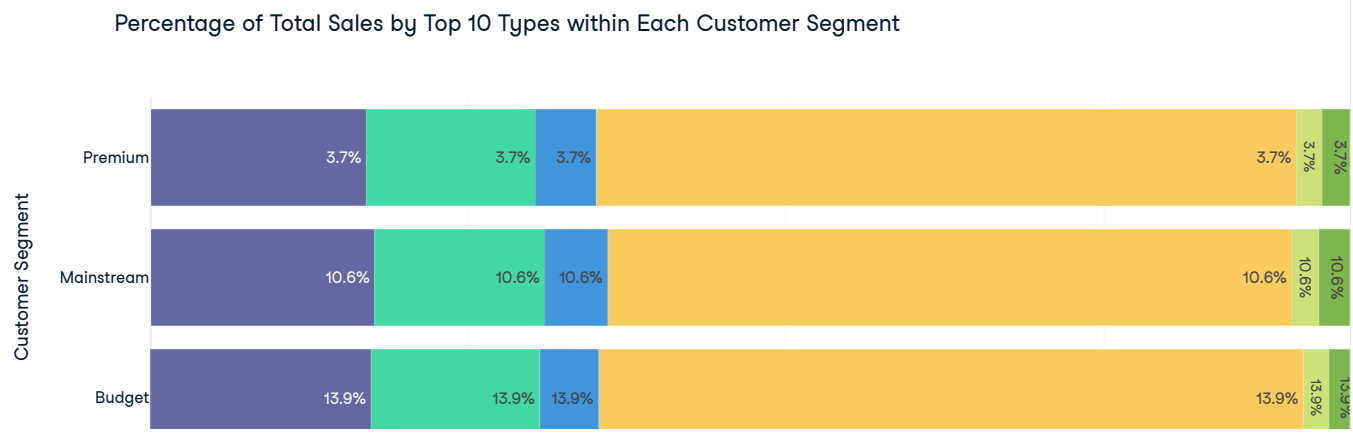
```python
import plotly.express as px

# Group by 'household_type' and 'customer_segment', then sum 'total_sales'
sales_by_household_segment = (
    merged_df.groupby(['household_type', 'customer_segment'])['total_sales']
    .sum()
    .reset_index()
)

# Create an interactive bar chart
fig = px.bar(
    sales_by_household_segment,
    x='household_type',
    y='total_sales',
    color='customer_segment',
    barmode='group',
    title='Total Sales by Household Type and Customer Segment',
    labels={
        'household_type': 'Household Type',
        'total_sales': 'Total Sales',
        'customer_segment': 'Customer Segment'
    }
)

fig.show()
```

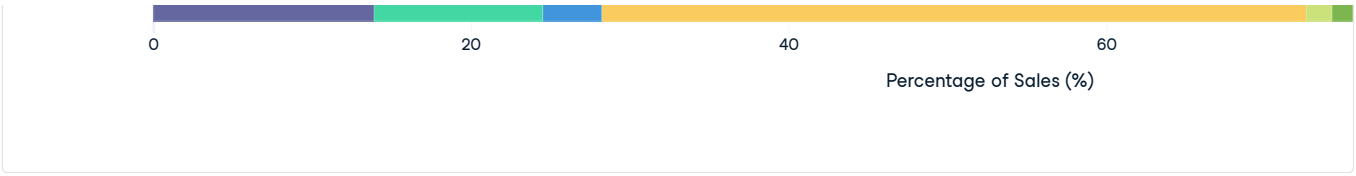


```python
import plotly.express as px

# Recalculate: Get the top 5 'type' (categories) by total_sales (across all customer_segments)
top5_types = (
    merged_df
    .groupby('type')['total_sales']
    .sum()
    .nlargest(5)
    .index
)

# Filter merged_df to only include these top 5 types
filtered_df = merged_df[merged_df['type'].isin(top5_types)]

# For each of the top 5 types, get total_sales by customer_segment
top5_sales_by_segment_category = (
    filtered_df
    .groupby(['type', 'customer_segment'])['total_sales']
    .sum()
    .reset_index()
)

# Create an interactive bar chart
fig = px.bar(
    top5_sales_by_segment_category,
    x='customer_segment',
    y='total_sales',
    color='type',
    barmode='group',
    title='Customer Segments by Total Sales for Top 5 Categories',
    labels={
        'customer_segment': 'Customer Segment',
        'total_sales': 'Total Sales',
        'type': 'Category'
    }
)

fig.show()
```

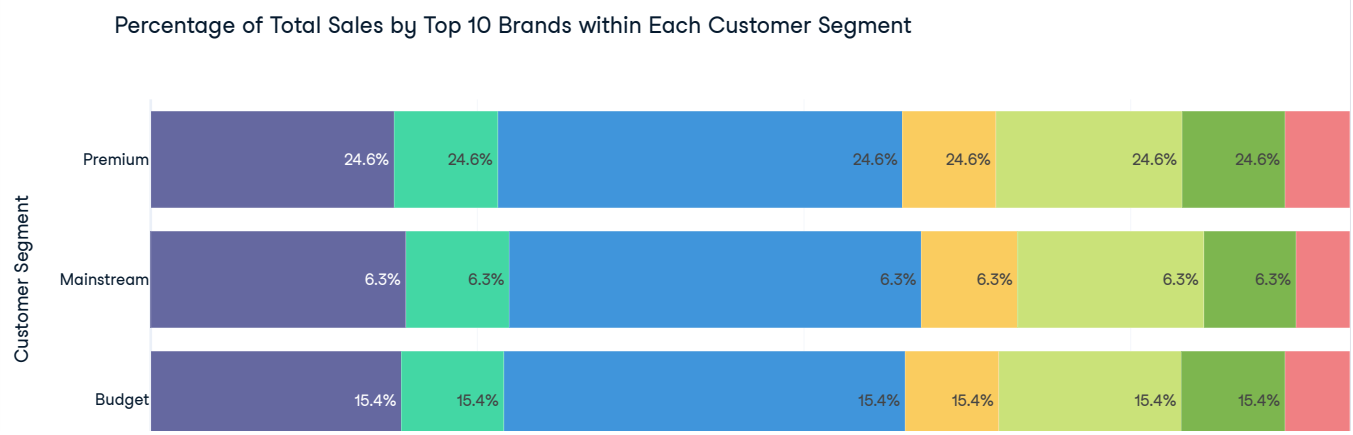


```
import pandas as pd
import plotly.express as px

# Step 1: Aggregate sales by customer segment and type
sales_by_segment_type = (
    merged_df.groupby(['customer_segment', 'type'])['total_sales']
    .sum()
    .reset_index()
)

# Step 2: Get top 10 types by total sales across all segments
top10_types = (
    sales_by_segment_type.groupby('type')['total_sales']
    .sum()
    .nlargest(10)
    .index
)

# Step 3: Filter to only top 10 types
top10_sales_type = sales_by_segment_type[
    sales_by_segment_type['type'].isin(top10_types)
].copy()

# Step 4: Calculate percentage of sales within each customer segment
top10_sales_type['percent_sales'] = (
    top10_sales_type.groupby('customer_segment')['total_sales']
    .transform(lambda x: x / x.sum() * 100)
)

# Step 5: Create horizontal stacked bar chart
fig = px.bar(
    top10_sales_type,
    y='customer_segment',
    x='percent_sales',
    color='type',
    orientation='h',
    barmode='stack',
    title='Percentage of Total Sales by Top 10 Types within Each Customer Segment',
    labels={
        'customer_segment': 'Customer Segment',
        'percent_sales': 'Percentage of Sales (%)',
        'type': 'Type'
    },
    hover_data=['type', 'customer_segment', 'total_sales', 'percent_sales']
)

# Step 6: Add percentage labels inside each bar segment
fig.update_traces(
    text=top10_sales_type['percent_sales'].round(1).astype(str) + '%',
    textposition='inside'
)

fig.update_layout(
    xaxis_title='Percentage of Sales (%)',
    legend_title='Type'
)

fig.show()
```




0
20
40
60
Percentage of Sales (%)

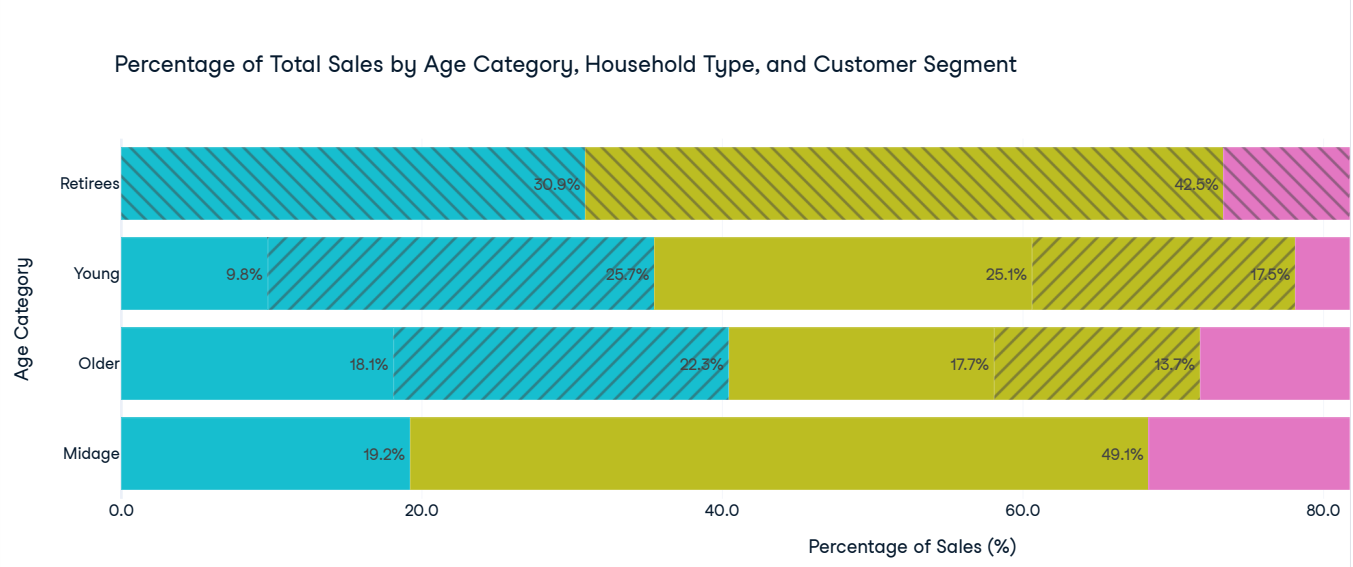
```
import pandas as pd
import plotly.express as px

# Step 1: Aggregate sales by customer segment and brand
sales_by_segment_brand = (
    merged_df.groupby(['customer_segment', 'brand'])['total_sales']
    .sum()
    .reset_index()
)

# Step 2: Get top 10 brands by total sales across all segments
top10_brands = (
    sales_by_segment_brand.groupby('brand')['total_sales']
    .sum()
    .nlargest(10)
    .index
)

# Step 3: Filter to only top 10 brands
top10_sales_brand = sales_by_segment_brand[
    sales_by_segment_brand['brand'].isin(top10_brands)
].copy()

# Step 4: Calculate percentage of sales within each customer segment
top10_sales_brand['percent_sales'] = (
    top10_sales_brand.groupby('customer_segment')['total_sales']
    .transform(lambda x: x / x.sum() * 100)
)

# Step 5: Create horizontal stacked bar chart
fig = px.bar(
    top10_sales_brand,
    y='customer_segment',
    x='percent_sales',
    color='brand',
    orientation='h',
    barmode='stack',
    title='Percentage of Total Sales by Top 10 Brands within Each Customer Segment',
    labels={
        'customer_segment': 'Customer Segment',
        'percent_sales': 'Percentage of Sales (%)',
        'brand': 'Brand'
    },
    hover_data=['brand', 'customer_segment', 'total_sales', 'percent_sales']
)

# Step 6: Add percentage labels inside each bar segment
fig.update_traces(
    text=top10_sales_brand['percent_sales'].round(1).astype(str) + '%',
    textposition='inside'
)

fig.update_layout(
    xaxis_title='Percentage of Sales (%)',
    legend_title='Brand'
)

fig.show()
```



customer segment

- Budget → These shoppers usually choose the cheapest options. They're focused on price and savings.
- Mainstream → These shoppers go for mid-range products. They balance cost and quality, often picking familiar brands.
- Premium → These shoppers are willing to spend more for higher quality or brand prestige. They care less about price and more about value or status.

```
# Group by 'age_category', 'household_type', and 'customer_segment', then sum 'total_sales'
grouped = (
    merged_df.groupby(['age_category', 'household_type', 'customer_segment'])['total_sales']
    .sum()
    .reset_index()
)

# Calculate percentage of total_sales within each age_category
grouped['percent_sales'] = (
    grouped.groupby('age_category')['total_sales']
    .transform(lambda x: x / x.sum() * 100)
)

# Define new custom colors for customer segments
custom_colors = {
    'Premium': '#e377c2',   # pink
    'Budget': '#17becf',    # teal
    'Mainstream': '#bcbd22' # olive
}

# Create horizontal stacked bar chart with new custom colors
fig = px.bar(
    grouped,
    y='age_category',
    x='percent_sales',
    color='customer_segment',
    pattern_shape='household_type',
    barmode='stack',
    orientation='h',
    title='Percentage of Total Sales by Age Category, Household Type, and Customer Segment',
    labels={
        'age_category': 'Age Category',
        'percent_sales': 'Percentage of Sales (%)',
        'customer_segment': 'Customer Segment',
        'household_type': 'Household Type'
    },
    hover_data={
        'customer_segment': True,
        'household_type': True,
        'percent_sales': ':.2f',
        'total_sales': True
    },
    text='percent_sales',
    color_discrete_map=custom_colors  # apply new custom colors
)

fig.update_traces(texttemplate='%{text:.1f}%', textposition='inside')

fig.update_layout(
    xaxis_tickformat='.1f',
    xaxis_title='Percentage of Sales (%)',
    legend_title_text='Customer Segment / Household Type'
)

fig.show()
```



Insight

- Midage Singles/Couples spend the most in Mainstream and also buy a lot of Premium. They are less focused on Budget.
- Older Families lean heavily toward Budget products. Premium is the smallest share here.
- Older Singles/Couples are balanced — they split their spending fairly evenly across Budget, Mainstream, and Premium.
- Retirees spend the most in Mainstream, but Budget and Premium are also important.
- Young Families prefer Budget products, showing they are more price-sensitive.
- Young Singles/Couples spend mostly in Mainstream, with little in Budget or Premium.

Summary

- Midage Singles/Couples → spend 49% Mainstream, 32% Premium, 19% Budget. They lean toward higher-quality and branded products.
- Older Families → spend 22% Budget, 14% Mainstream, 11% Premium. They are the most price-sensitive group.
- Older Singles/Couples → almost equal split: 18% Budget, 18% Mainstream, 18% Premium. No clear preference, very balanced.
- Retirees → spend 42% Mainstream, 31% Budget, 27% Premium. They mix mid-range with both cheaper and premium options.
- Young Families → spend 26% Budget, 18% Mainstream, 15% Premium. Strong tilt toward Budget choices.
- Young Singles/Couples → spend 25% Mainstream, 10% Budget, 7% Premium. They prefer mainstream, with little interest in Budget or Premium.

** Key takeaway**

- Families skew Budget (Older Families 22%, Young Families 26%).
- Singles/Couples skew Mainstream or Premium (Midage Singles/Couples 49% Mainstream, 32% Premium).
- Retirees skew Mainstream (42%) but still buy across all segments.
- This makes it clear who is price-sensitive (families), who is brand/quality-driven (midage singles/couples), and who is balanced (older singles/couples, retirees).

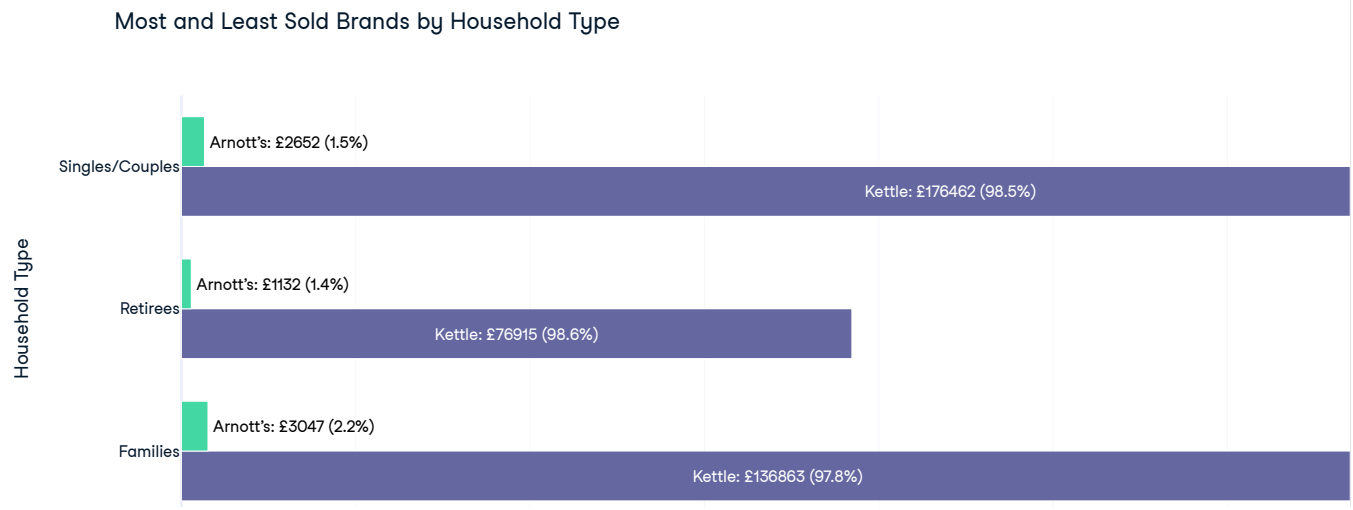
Actions

- *Targeted promotions*

- Offer Budget bundles (multi-packs, family sizes) to Young and Older Families, since they lean toward cheaper options.
- Run Premium product campaigns for Midage Singles/Couples and Retirees, who show higher willingness to spend.

```
# Step 1: Aggregate sales by household_type and brand
sales_by_segment_brand = (
    merged_df.groupby(['household_type', 'brand'])['total_sales']
    .sum()
    .reset_index()
)

# Step 2: For each household_type, find the brand with max and min sales
max_sales = sales_by_segment_brand.loc[
    sales_by_segment_brand.groupby('household_type')['total_sales'].idxmax()
]

min_sales = sales_by_segment_brand.loc[
    sales_by_segment_brand.groupby('household_type')['total_sales'].idxmin()
]

# Step 3: Combine results
brand_extremes = pd.concat([max_sales.assign(extreme='Most Sold'),
                            min_sales.assign(extreme='Least Sold')])
```

```
import plotly.express as px

# Prepare data for plotting
brand_extremes_plot = brand_extremes.copy()
brand_extremes_plot['extreme'] = brand_extremes_plot['extreme'].map({'Most Sold': 'Most Sold Brand', 'Least Sold': 'Least Sold
Brand'})
# Calculate total sales per household_type for percentage calculation
total_sales_by_household = brand_extremes_plot.groupby('household_type')['total_sales'].transform('sum')
brand_extremes_plot['percent_share'] = (brand_extremes_plot['total_sales'] / total_sales_by_household * 100).round(1)
# Create label for inside bar:
brand_extremes_plot['bar_label'] = (
    brand_extremes_plot['brand'] + ': £' +
    brand_extremes_plot['total_sales'].round(0).astype(int).astype(str) +
    ' (' + brand_extremes_plot['percent_share'].astype(str) + '%)'
)

fig = px.bar(
    brand_extremes_plot,
    y='household_type',
    x='total_sales',
    color='extreme',
    orientation='h',
    barmode='group',
    text='bar_label',
    title='Most and Least Sold Brands by Household Type',
    labels={'total_sales': 'Total Sales', 'household_type': 'Household Type', 'extreme': 'Brand Type'},
    hover_data=['brand', 'total_sales', 'percent_share']
)

# Update traces separately: inside for Most Sold, outside for Least Sold
for i, trace in enumerate(fig.data):
    if 'Most Sold' in trace.name:
        fig.data[i].textposition = 'inside'
        fig.data[i].insidetextanchor = 'middle'
    else:
        fig.data[i].textposition = 'outside'
        # optional: add a small offset so labels don't overlap bars
        fig.data[i].textfont = dict(color='black', size=12)

fig.update_layout(
    yaxis_title='Household Type',
    xaxis_title='Total Sales',
    xaxis_tickformat=',',
    legend_title_text='Brand Type',
    bargap=0.3,
    height=500,
    legend=dict(
        orientation="v",
        yanchor="top",
        y=1,
        xanchor="left",
        x=1.05
    )
)

fig.show()
```



Key insight

Across all household types, Kettle is the most sold brand:

- Families: £136,863
- Retirees: £76,915
- Singles/Couples: £176,462

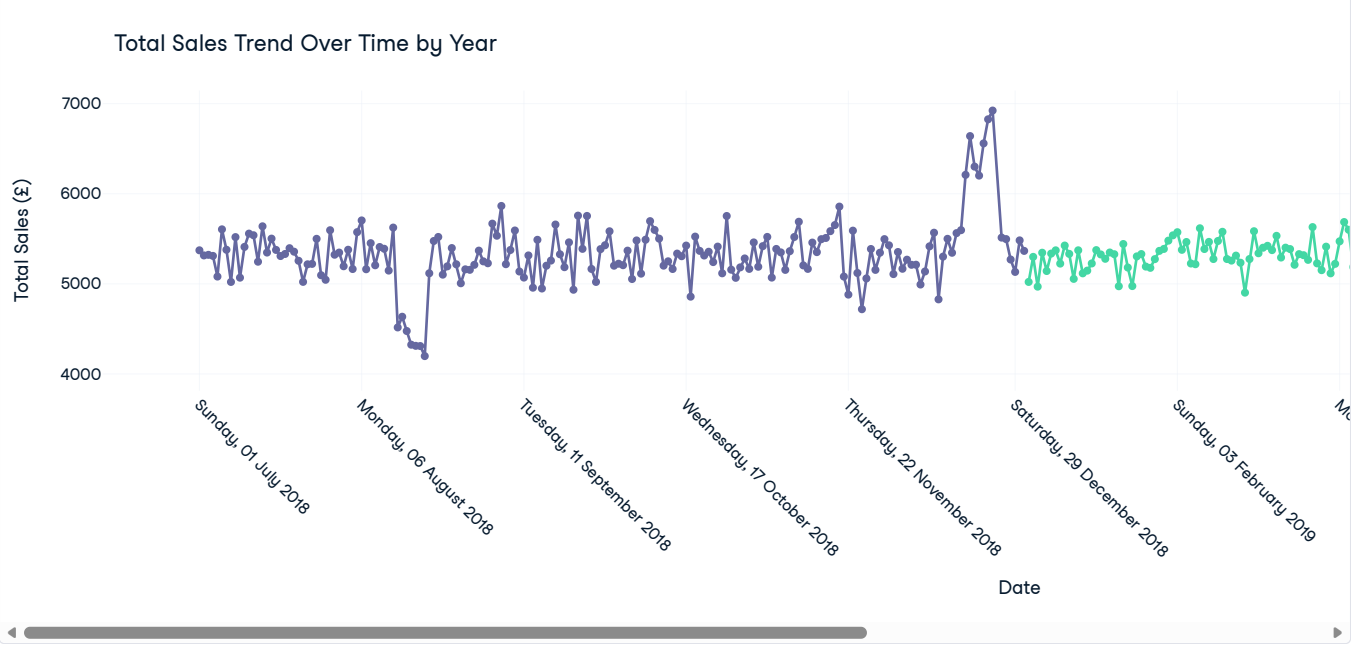
Across all household types, Arnott's is the least sold brand:

- Families: £3,048
- Retirees: £1,132
- Singles/Couples: £2,652

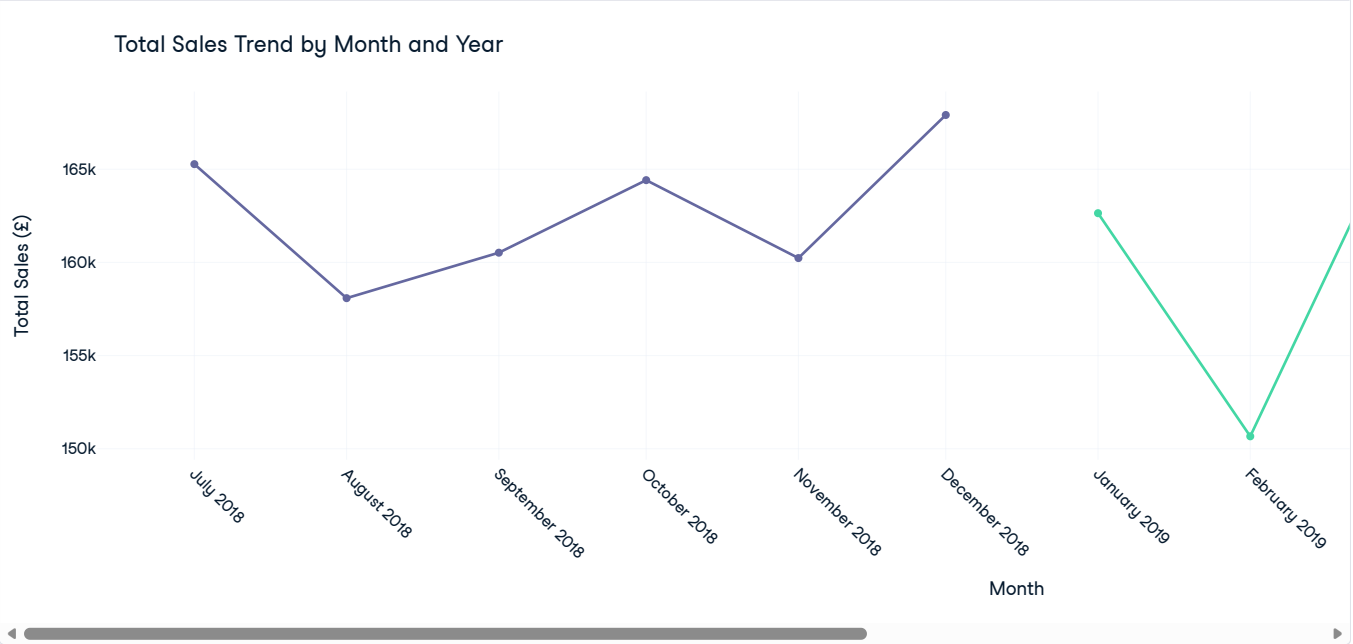
Actions & Recommendations

- Double down on Kettle: Feature it in promotions and bundles across all household types.
- Reassess Arnott's: Investigate why sales are low

```
import plotly.express as px
import pandas as pd

# Ensure 'date' is in datetime format
merged_df['date'] = pd.to_datetime(merged_df['date'])

# Extract year for separation
merged_df['year'] = merged_df['date'].dt.year

# Group by year and date, sum total sales
sales_trend = merged_df.groupby(['year', 'date'])['total_sales'].sum().reset_index()

# Format date as full date name (e.g., 'Monday, 01 January 2024')
sales_trend['date_label'] = sales_trend['date'].dt.strftime('%A, %d %B %Y')

fig = px.line(
    sales_trend,
    x='date',
    y='total_sales',
    color='year',
    markers=True,
    title='Total Sales Trend Over Time by Year',
    labels={
        'date': 'Date',
        'total_sales': 'Total Sales (£)',
        'year': 'Year'
    },
    hover_data={'date_label': True, 'date': False, 'year': True}
)

fig.update_traces(hovertemplate='<b>%{customdata[0]}</b><br>Year: %{customdata[1]}<br>Total Sales (£): %{y:,.2f}')
fig.update_layout(
    xaxis_title='Date',
    yaxis_title='Total Sales (£)',
    xaxis=dict(
        tickmode='array',
        tickvals=sales_trend['date'][::max(1, len(sales_trend)//10)],
        ticktext=sales_trend['date_label'][::max(1, len(sales_trend)//10)],
        tickangle=45
    ),
    hoverlabel=dict(bgcolor="white"),
    margin=dict(l=40, r=40, t=60, b=80)
)

fig.show()
```



```
import plotly.express as px
import pandas as pd

# Ensure 'date' is in datetime format
merged_df['date'] = pd.to_datetime(merged_df['date'])

# Create a 'month' column (first day of each month)
merged_df['month'] = merged_df['date'].dt.to_period('M').dt.to_timestamp()

# Extract year for separation
merged_df['year'] = merged_df['date'].dt.year

# Group by year and month, sum total sales
sales_trend_monthly = merged_df.groupby(['year', 'month'])['total_sales'].sum().reset_index()

# Format month as 'Month YYYY' (e.g., 'January 2024')
sales_trend_monthly['month_label'] = sales_trend_monthly['month'].dt.strftime('%B %Y')

fig = px.line(
    sales_trend_monthly,
    x='month',
    y='total_sales',
    color='year',
    markers=True,
    title='Total Sales Trend by Month and Year',
    labels={
        'month': 'Month',
        'total_sales': 'Total Sales (£)',
        'year': 'Year'
    },
    hover_data={'month_label': True, 'month': False, 'year': True}
)

fig.update_traces(hovertemplate='<b>%{customdata[0]}</b><br>Year: %{customdata[1]}<br>Total Sales (£): %{y:,.2f}')
fig.update_layout(
    xaxis_title='Month',
    yaxis_title='Total Sales (£)',
    xaxis=dict(
        tickmode='array',
        tickvals=sales_trend_monthly['month'][::max(1, len(sales_trend_monthly)//10)],
        ticktext=sales_trend_monthly['month_label'][::max(1, len(sales_trend_monthly)//10)],
        tickangle=45
    ),
    hoverlabel=dict(bgcolor="white"),
    margin=dict(l=40, r=40, t=60, b=80)
)

fig.show()
```



Analysis:

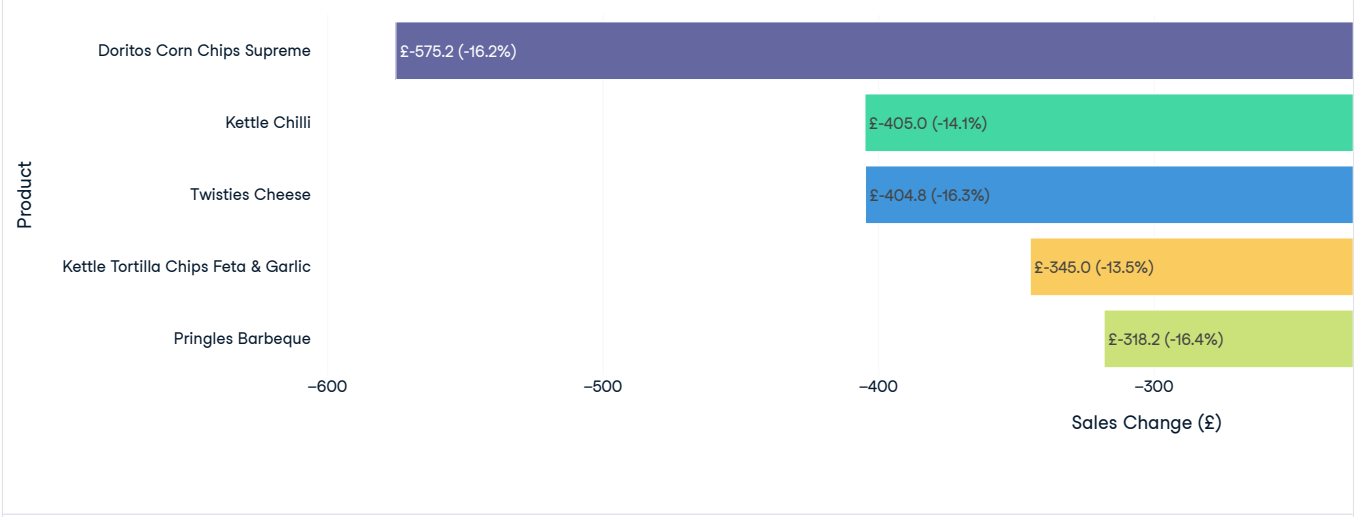
The code analyses monthly sales trends, identifying the largest month-to-month sales declines each year. For each year's biggest drop, it highlights the top 5 declines at the product, brand, and store+product levels, visualizing these with clear bar charts. This helps pinpoint which products, brands, or store/product combinations contributed most to sales downturns.

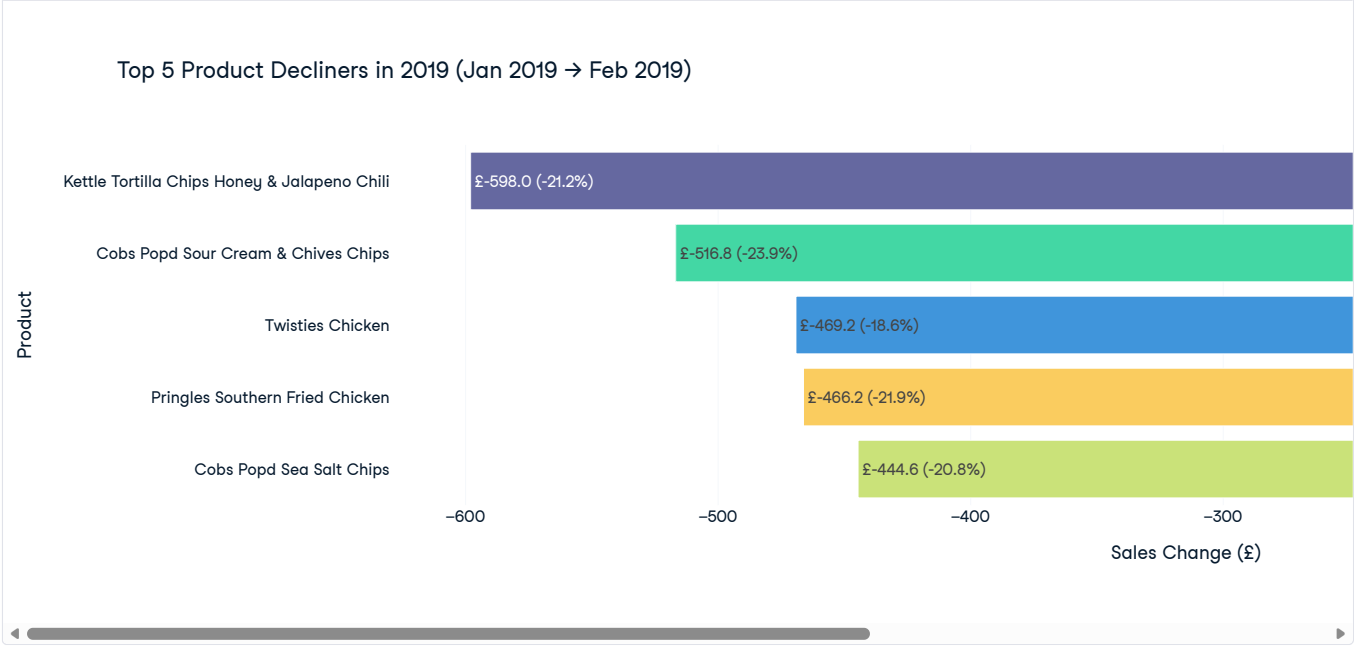
Recommended Actions:

- Investigate the top declining products, brands, and store/product pairs during the identified periods for root causes (e.g., supply issues, promotions ending, competitor actions).
- Develop targeted strategies to address these declines, such as marketing campaigns, stock adjustments, or pricing reviews for the affected items or locations.
- Monitor these segments in future months to assess the impact of any interventions.

```
import pandas as pd
import plotly.express as px

# Step 1: Ensure month and year columns exist
merged_df['month'] = merged_df['date'].dt.to_period('M').dt.to_timestamp()
merged_df['year'] = merged_df['date'].dt.year

# Step 2: Aggregate total sales by year and month
monthly_totals = (
    merged_df.groupby(['year','month'])['total_sales']
    .sum()
    .reset_index()
    .sort_values(['year','month'])
)

# Step 3: Calculate month-to-month change within each year
monthly_totals['change'] = monthly_totals.groupby('year')['total_sales'].diff()
monthly_totals['percent_change'] = (
    monthly_totals['change'] / monthly_totals.groupby('year')['total_sales'].shift(1) * 100
).round(1)

# Step 4: Find largest decline per year
largest_decline_per_year = monthly_totals.loc[
    monthly_totals.groupby('year')['change'].idxmin()
]

# Step 5: For each year, get product-level declines in that period
for _, row in largest_decline_per_year.iterrows():
    year = row['year']
    drop_month = row['month']
    prev_month = monthly_totals.loc[
        (monthly_totals['year']==year) & (monthly_totals['month']<drop_month),
        'month'
    ].max()

    # Product-level sales for those two months
    sales_by_product = (
        merged_df[merged_df['month'].isin([prev_month, drop_month])]
        .groupby(['product','month'])['total_sales']
        .sum()
        .reset_index()
        .pivot(index='product', columns='month', values='total_sales')
        .fillna(0)
    )

    # Calculate change
    sales_by_product['change'] = sales_by_product[drop_month] - sales_by_product[prev_month]
    sales_by_product['percent_change'] = (
        sales_by_product['change'] / sales_by_product[prev_month].replace(0, pd.NA) * 100
    ).round(1)

    # Top 5 decliners
    top5_decliners = sales_by_product[sales_by_product['change']<0] \
        .sort_values('change') \
        .head(5) \
        .reset_index()

    # Plot horizontal bar chart with number inside the bar
    fig = px.bar(
        top5_decliners,
        y='product',
        x='change',
        orientation='h',
        color='product',  # Different color for each product
        text=top5_decliners.apply(lambda r: f"£{r['change']:.1f} ({r['percent_change']}%)", axis=1),
        title=f"Top 5 Product Decliners in {year} ({prev_month.strftime('%b %Y')} → {drop_month.strftime('%b %Y')})",
        labels={'change':'Sales Change (£)','product':'Product'}
    )

    fig.update_traces(textposition='inside')
    fig.update_layout(xaxis_title='Sales Change (£)', yaxis_title='Product', showlegend=False)
    fig.show()
```

Top 5 Product Decliners in 2018 (Jul 2018 → Aug 2018)



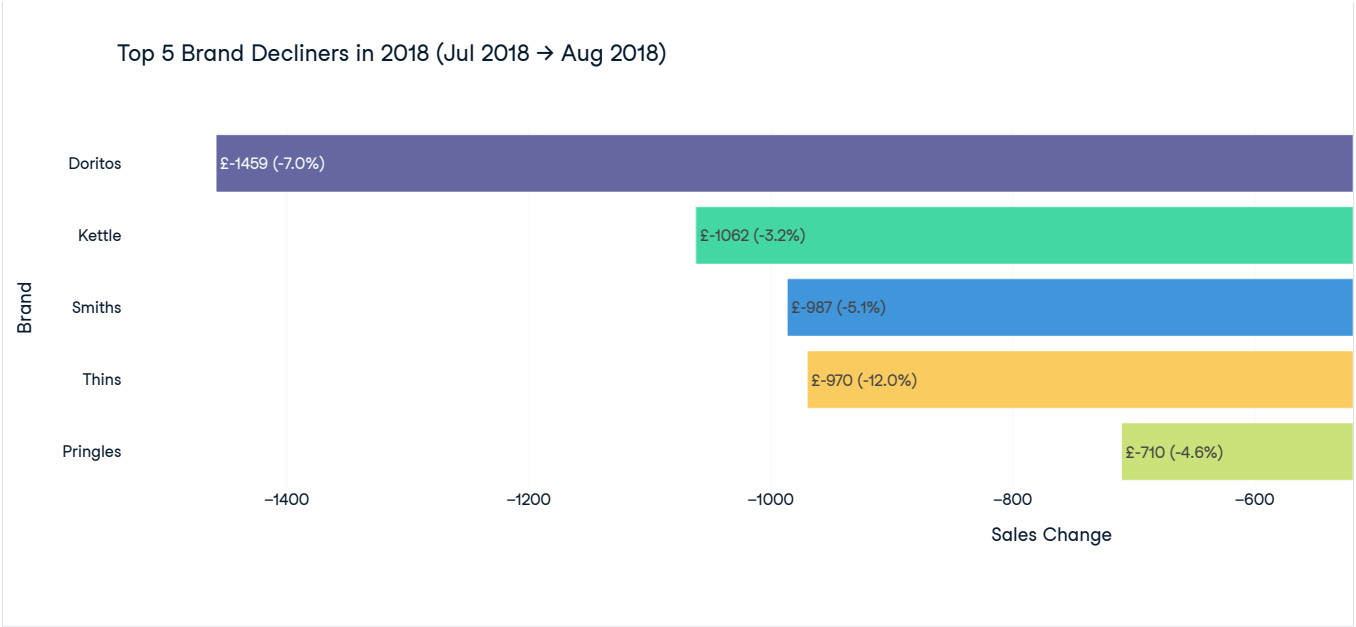


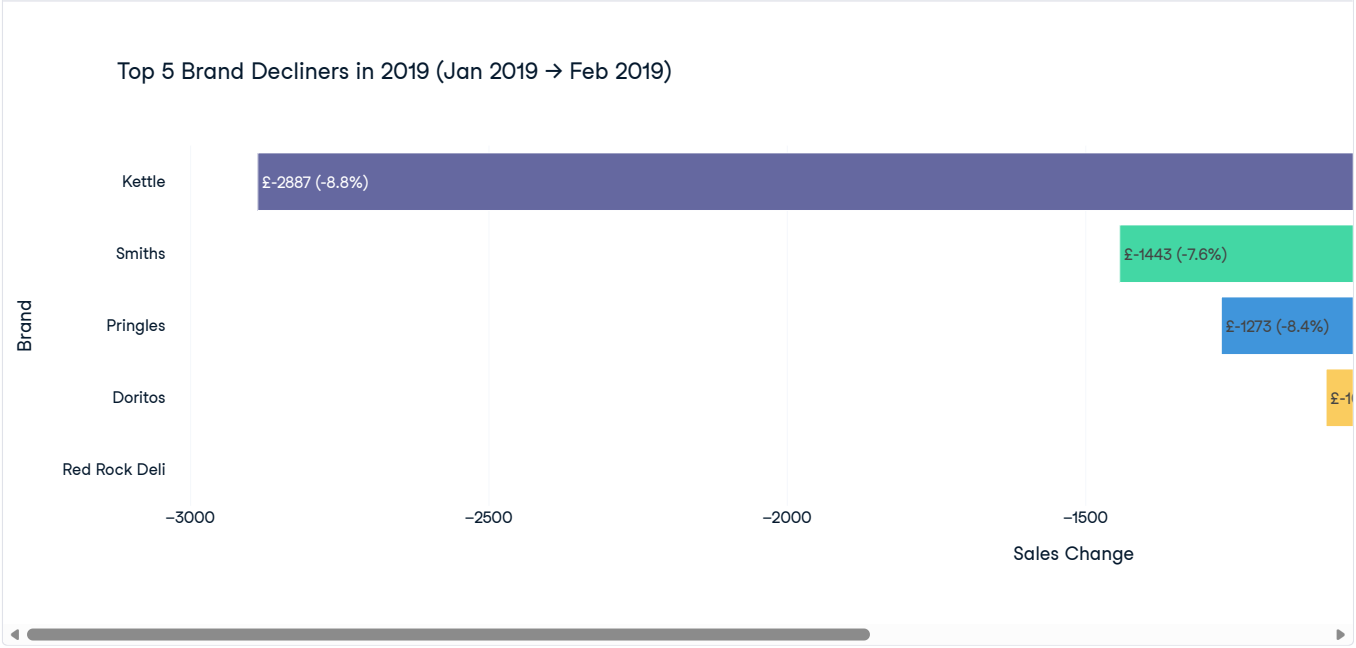
top5_decliners

	product	2019-01-01 00:00...	2019-02-01 00:00...		percent...
0	Kettle Tortilla Chips Honey & Jalapeno Chili	2819.8	2221.8	-598	-21.2
1	Cobs Popd Sour Cream & Chives Chips	2166	1649.2	-516.8	-23.9
2	Twisties Chicken	2525.4	2056.2	-469.2	-18.6
3	Pringles Southern Fried Chicken	2131.2	1665	-466.2	-21.9
4	Cobs Popd Sea Salt Chips	2135.6	1691	-444.6	-20.8

Rows: 5 Expand

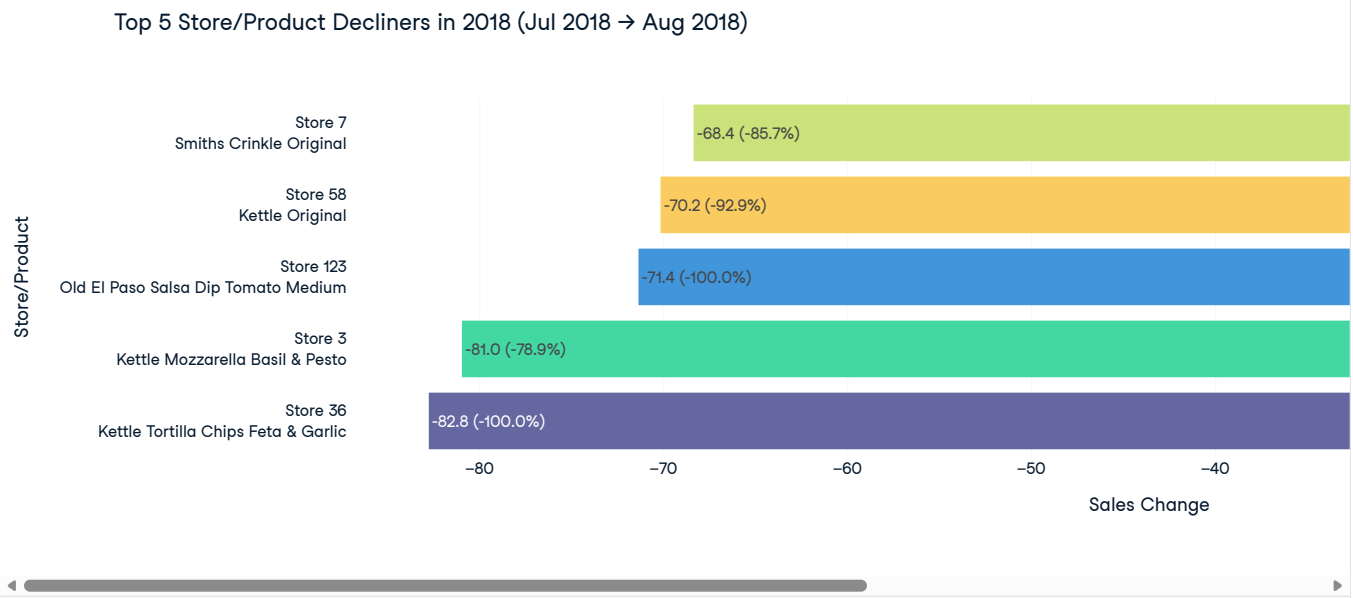
```
import pandas as pd
import plotly.express as px

# Step 1: Ensure month and year columns exist
merged_df['month'] = merged_df['date'].dt.to_period('M').dt.to_timestamp()
merged_df['year'] = merged_df['date'].dt.year

# Step 2: Aggregate total sales by year and month
monthly_totals = (
    merged_df.groupby(['year','month'])['total_sales']
    .sum()
    .reset_index()
    .sort_values(['year','month'])
)

# Step 3: Calculate month-to-month change within each year
monthly_totals['change'] = monthly_totals.groupby('year')['total_sales'].diff()
monthly_totals['percent_change'] = (
    monthly_totals['change'] / monthly_totals.groupby('year')['total_sales'].shift(1) * 100
).round(1)

# Step 4: Find largest decline per year
largest_decline_per_year = monthly_totals.loc[
    monthly_totals.groupby('year')['change'].idxmin()
]

# Step 5: For each year, get brand-level declines in that period
for _, row in largest_decline_per_year.iterrows():
    year = row['year']
    drop_month = row['month']
    prev_month = monthly_totals.loc[
        (monthly_totals['year']==year) & (monthly_totals['month']<drop_month),
        'month'
    ].max()

    # Brand-level sales for those two months
    sales_by_brand = (
        merged_df[merged_df['month'].isin([prev_month, drop_month])]
        .groupby(['brand','month'])['total_sales']
        .sum()
        .reset_index()
        .pivot(index='brand', columns='month', values='total_sales')
        .fillna(0)
    )

    # Calculate change
    sales_by_brand['change'] = sales_by_brand[drop_month] - sales_by_brand[prev_month]
    sales_by_brand['percent_change'] = (
        sales_by_brand['change'] / sales_by_brand[prev_month].replace(0, pd.NA) * 100
    ).round(1)

    # Top 5 decliners
    top5_decliners = sales_by_brand[sales_by_brand['change']<0] \
        .sort_values('change') \
        .head(5) \
        .reset_index()

    # Format the change as £ and percent for inside the bar
    top5_decliners['text_label'] = top5_decliners.apply(
        lambda r: f"£{r['change']:.0f} ({r['percent_change']}%)", axis=1
    )

    # Plot horizontal bar chart with different color for each brand
    fig = px.bar(
        top5_decliners,
        y='brand',
        x='change',
        orientation='h',
        color='brand',  # Different color for each brand
        text='text_label',
        title=f"Top 5 Brand Decliners in {year} ({prev_month.strftime('%b %Y')} → {drop_month.strftime('%b %Y')})",
        labels={'change':'Sales Change','brand':'Brand'}
    )

    fig.update_traces(textposition='inside')
    fig.update_layout(xaxis_title='Sales Change', yaxis_title='Brand', showlegend=False)
    fig.show()
```

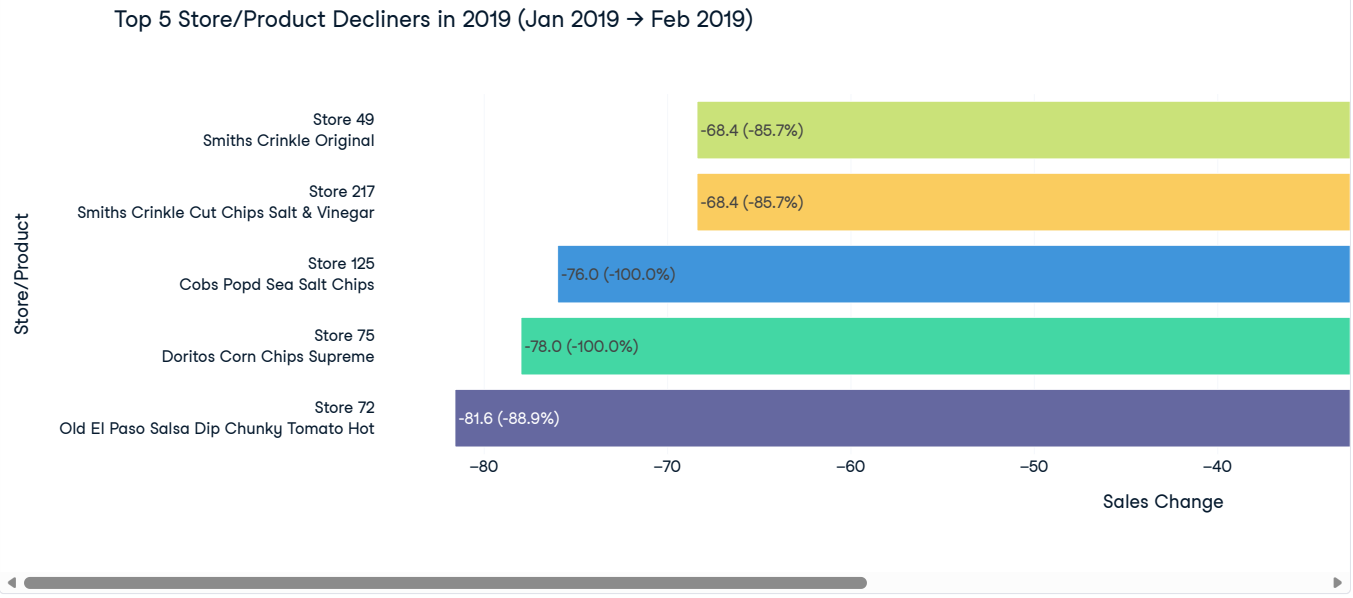

Top 5 Brand Decliners in 2018 (Jul 2018 → Aug 2018)
Brand
Doritos
Kettle
Smiths
Thins
Pringles
£-1459 (-7.0%)
£-1062 (-3.2%)
£-987 (-5.1%)
£-970 (-12.0%)
£-710 (-4.6%)
−1400
−1200
−1000
−800
−600
Sales Change


Top 5 Brand Decliners in 2019 (Jan 2019 → Feb 2019)
Brand
Kettle
Smiths
Pringles
Doritos
Red Rock Deli
£-2887 (-8.8%)
£-1443 (-7.6%)
£-1273 (-8.4%)
−3000
−2500
−2000
−1500
Sales Change

```
import pandas as pd
import plotly.express as px

# Step 1: Ensure month and year columns exist
merged_df['month'] = merged_df['date'].dt.to_period('M').dt.to_timestamp()
merged_df['year'] = merged_df['date'].dt.year

# Step 2: Aggregate monthly totals per year
monthly_totals = (
    merged_df.groupby(['year','month'])['total_sales']
    .sum()
    .reset_index()
    .sort_values(['year','month'])
)

# Step 3: Calculate month-to-month change within each year
monthly_totals['change'] = monthly_totals.groupby('year')['total_sales'].diff().round(1)
monthly_totals['percent_change'] = (
    monthly_totals['change'] / monthly_totals.groupby('year')['total_sales'].shift(1) * 100
).round(1)

# Step 4: Find largest decline per year
largest_decline_per_year = monthly_totals.loc[
    monthly_totals.groupby('year')['change'].idxmin()
]

# Step 5: For each year, get product+store declines in that period
for _, row in largest_decline_per_year.iterrows():
    year = row['year']
    drop_month = row['month']
    prev_month = monthly_totals.loc[
        (monthly_totals['year']==year) & (monthly_totals['month']<drop_month),
        'month'
    ].max()

    # Product + store sales for those two months
    sales_by_store_product = (
        merged_df[merged_df['month'].isin([prev_month, drop_month])]
        .groupby(['store_number','product','month'])['total_sales']
        .sum()
        .reset_index()
        .pivot(index=['store_number','product'], columns='month', values='total_sales')
        .fillna(0)
    )

    # Calculate change
    sales_by_store_product['change'] = (sales_by_store_product[drop_month] - sales_by_store_product[prev_month]).round(1)
    sales_by_store_product['percent_change'] = (
        sales_by_store_product['change'] / sales_by_store_product[prev_month].replace(0, pd.NA) * 100
    ).round(1)

    # Top 5 decliners
    top5_decliners = sales_by_store_product[sales_by_store_product['change']<0] \
        .sort_values('change') \
        .head(5) \
        .reset_index()

    # Treat store_number as text for coloring
    top5_decliners['store_number'] = top5_decliners['store_number'].astype(str)

    # Combine store_number + product with line break for readability
    top5_decliners['store_product'] = (
        "Store " + top5_decliners['store_number'] + "<br>" + top5_decliners['product']
    )

    # Plot horizontal bar chart with distinct colors per store
    fig = px.bar(
        top5_decliners,
        y='store_product',
        x='change',
        color='store_number',   # color by store, treated as text
        orientation='h',
        text=top5_decliners.apply(lambda r: f"{round(r['change'],1):.1f} ({round(r['percent_change'],1)}%)", axis=1),
        title=f"Top 5 Store/Product Decliners in {year} ({prev_month.strftime('%b %Y')} → {drop_month.strftime('%b %Y')})",
        labels={'change':'Sales Change','store_product':'Store/Product','store_number':'Store'}
    )
```

```
    fig.update_traces(textposition='inside')
    fig.update_layout(
        xaxis_title='Sales Change',
        yaxis_title='Store/Product',
        margin=dict(l=250)   # increase left margin for long labels
    )
    fig.show()
```





Executive Summary: July 2018 → August 2018

- Sales declines in this period were driven primarily by store-level execution issues, with several products experiencing 100% drops in specific stores, indicating delistings or stock-outs rather than true demand loss. The most severe cases were Kettle Tortilla Chips Feta & Garlic (Store 36) and Old El Paso Salsa Dip Tomato Medium (Store 123).
- At brand and product level, declines were broad-based but less extreme, with major brands such as Doritos and Kettle contributing the largest value losses and most core SKUs declining by ~14–16%. This consistency suggests temporary category softness rather than isolated product failures.

Executive Summary: Jan 2019 → Feb 2019

- Sales declines in early 2019 were broad and demand-led, with the top five declining products all falling by ~19–24% month-on-month and no evidence of full delistings or stock-outs.
- The largest value loss came from Kettle Tortilla Chips Honey & Jalapeno Chili (£–598, –21.2%), while Cobs Pop'd appeared twice among the top decliners, indicating brand-level weakness rather than isolated SKU issues. Similar-sized declines across Twisties and Pringles further reinforce a category-wide slowdown, likely linked to post-holiday seasonality or reduced promotional activity.

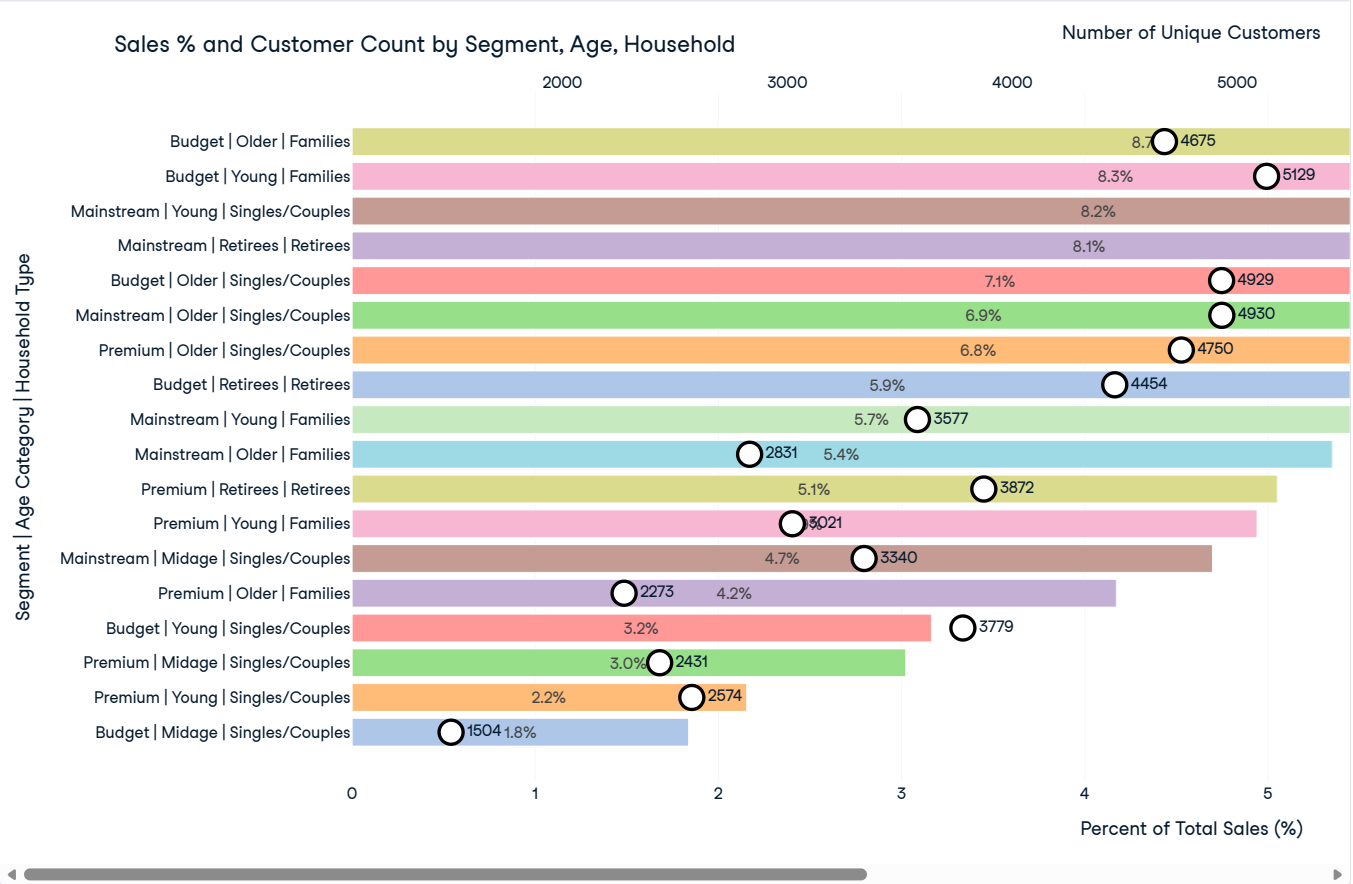
"SKU-level decline", we mean the sales drop of that exact product, not the whole brand

```
# Get the unique list of brands from the 'brand' column
unique_brands = merged_df['brand'].unique()
# Create a mapping from brand to affluence (for example, just mapping brand to itself as a placeholder)
brand_to_affluence = {brand: brand for brand in unique_brands}
merged_df['affluence'] = merged_df['brand'].map(brand_to_affluence)
```

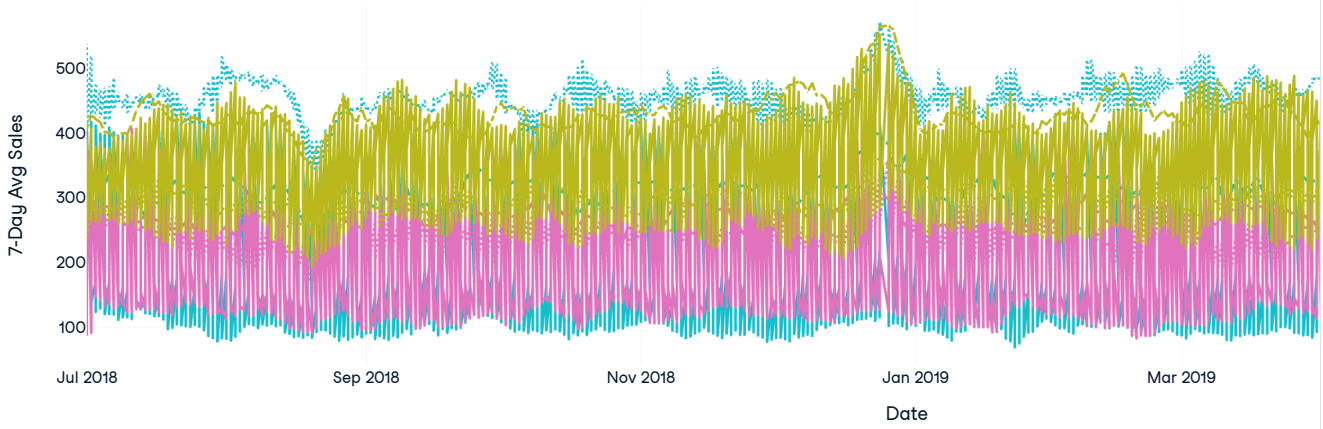
```
import pandas as pd
import plotly.express as px

# 1. Find top 5 brands overall
top5_brands = (
    merged_df.groupby('brand', as_index=False)['total_sales']
    .sum()
    .sort_values('total_sales', ascending=False)
    .head(5)['brand']
    .tolist()
)

# 2. Filter dataset to top 5 brands
filtered_df = merged_df[merged_df['brand'].isin(top5_brands)]

# 3. Aggregate sales by customer segment + age category + brand
segment_sales = (
    filtered_df
    .groupby(['customer_segment','age_category','brand'], as_index=False)['total_sales']
    .sum()
)

# 4. Calculate percent share within each segment+age group
segment_sales['percent_share'] = (
    segment_sales.groupby(['customer_segment','age_category'])['total_sales']
    .transform(lambda x: x / x.sum() * 100)
).round(1)

# 5. Plot stacked bar chart
fig = px.bar(
    segment_sales,
    y='customer_segment',
    x='percent_share',
    color='brand',
    orientation='h',
    facet_col='age_category',
    text='percent_share',
    title='Top 5 Chip Brands by Customer Segment and Age Category',
    labels={'percent_share':'% of Sales','customer_segment':'Customer Segment','brand':'Brand'}
)

fig.update_traces(textposition='inside')
fig.update_layout(barmode='stack', height=600)
fig.show()
```

```
import plotly.graph_objects as go
import pandas as pd

# --- Recalculate agg_df inside this cell ---
grouped = merged_df.groupby(['customer_segment', 'age_category', 'household_type'])
agg_df = grouped.agg(
    total_sales=('total_sales', 'sum'),
    num_customers=('loyalty_card_number', 'nunique')
).reset_index()
total_sales_all = agg_df['total_sales'].sum()
agg_df['percent_total_sales'] = 100 * agg_df['total_sales'] / total_sales_all
agg_df['label'] = (
    agg_df['customer_segment'] + ' | ' +
    agg_df['age_category'] + ' | ' +
    agg_df['household_type']
)
agg_df = agg_df.sort_values('percent_total_sales', ascending=True).reset_index(drop=True)

# --- Assign a different (light) color for each bar ---
from plotly.colors import qualitative

# Use a light color palette (pastel)
light_palette = [
    "#aec7e8",  # light blue
    "#ffbb78",  # light orange
    "#98df8a",  # light green
    "#ff9896",  # light red
    "#c5b0d5",  # light purple
    "#c49c94",  # light brown
    "#f7b6d2",  # light pink
    "#dbdb8d",  # light yellow
    "#9edae5",  # light cyan
    "#c7e9c0",  # extra light green
]
num_colors = len(light_palette)
bar_colors = [light_palette[i % num_colors] for i in range(len(agg_df))]

# --- Interactive Plotly Plot ---
fig = go.Figure()

# Horizontal bar for percent of total sales (no legend)
fig.add_trace(go.Bar(
    x=agg_df['percent_total_sales'],
    y=agg_df['label'],
    orientation='h',
    marker=dict(color=bar_colors),
    name='Percent of Total Sales',
    text=[f"{v:.1f}%" for v in agg_df['percent_total_sales']],
    textposition='inside',
    insidetextanchor='middle',
    hovertemplate=(
        "<b>%{y}</b><br>" +
        "Percent of Total Sales: %{x:.1f}%<br>" +
        "Number of Customers: %{customdata[0]}<br>" +
        "Total Sales: $%{customdata[1]:,.2f}<extra></extra>"
    ),
    customdata=agg_df[['num_customers', 'total_sales']].values,
    showlegend=False  # Hide legend for bar
))

# Overlay scatter for number of customers (show legend)
fig.add_trace(go.Scatter(
    x=agg_df['num_customers'],
    y=agg_df['label'],
    mode='markers+text',
    marker=dict(
        color='white',
        size=18,
        line=dict(color='black', width=2.5)
    ),
    text=agg_df['num_customers'],
    textposition='middle right',
    name='Number of Customers',
    hovertemplate=(
        "<b>%{y}</b><br>" +
        "Number of Customers: %{x}<extra></extra>"
    ),
```

```
    xaxis='x2',
    showlegend=True  # Show legend for dot
))

# Set up secondary x-axis for number of customers
fig.update_layout(
    xaxis=dict(
        title='Percent of Total Sales (%)',
        showgrid=True,
        zeroline=False,
        side='bottom'
    ),
    xaxis2=dict(
        title='Number of Unique Customers',
        overlaying='x',
        side='top',
        showgrid=False,
        zeroline=False
    ),
    yaxis=dict(
        title='Segment | Age Category | Household Type',
        automargin=True
    ),
    title='Sales % and Customer Count by Segment, Age, Household',
    legend=dict(
        orientation='h',
        yanchor='bottom',
        y=0,
        xanchor='right',
        x=1
    ),
    height=max(600, len(agg_df)*35),
    margin=dict(l=180, r=40, t=60, b=40)
)

fig.show()
```



```
import pandas as pd
import plotly.express as px

# --- Recalculate everything inside this cell ---

# Ensure date is datetime
merged_df['date'] = pd.to_datetime(merged_df['date'])

# Group by date, age_category, household_type, and customer_segment
grouped_time = (
    merged_df.groupby(['date', 'age_category', 'household_type', 'customer_segment'])['total_sales']
    .sum()
    .reset_index()
)

# Sort by date
grouped_time = grouped_time.sort_values('date')

# Add rolling average (7-day smoothing)
grouped_time['rolling_sales'] = (
    grouped_time.groupby(['age_category','household_type','customer_segment'])['total_sales']
    .transform(lambda x: x.rolling(7, min_periods=1).mean())
)

# Define custom colors for customer segments
custom_colors = {
    'Premium': '#e377c2',   # pink
    'Budget': '#17becf',    # teal
    'Mainstream': '#bcbd22' # olive
}

# Combine age category into legend for clarity
grouped_time['legend_group'] = (
    grouped_time['age_category'] + ' | ' +
    grouped_time['customer_segment'] + ' | ' +
    grouped_time['household_type']
)

# --- Plotting ---
fig = px.line(
    grouped_time,
    x='date',
    y='rolling_sales',
    color='customer_segment',       # color by segment
    line_dash='household_type',     # dash by household type
    facet_row=None,                 # no facets, all in one chart
    color_discrete_map=custom_colors,
    title='Smoothed Sales Over Time by Age, Household Type, and Segment',
    labels={
        'date': 'Date',
        'rolling_sales': '7-Day Avg Sales',
        'customer_segment': 'Customer Segment',
        'household_type': 'Household Type',
        'age_category': 'Age Category'
    },
    hover_data={
        'customer_segment': True,
        'household_type': True,
        'age_category': True,
        'rolling_sales': ':.2f',
        'date': '|%b %d, %Y'
    }
)

# Layout tweaks
fig.update_layout(
    yaxis_title='7-Day Avg Sales',
    legend_title_text='Customer Segment',
    hovermode='x unified'
)
fig.update_yaxes(tickformat=",")

fig.show()
```


7-Day Avg Sales
500
400
300
200
100
Jul 2018
Sep 2018
Nov 2018
Jan 2019
Mar 2019
Date

```
import pandas as pd
import plotly.express as px

# --- Recalculate everything inside this cell ---

# Ensure date is datetime
merged_df['date'] = pd.to_datetime(merged_df['date'])

# Create a month column (first day of each month)
merged_df['month'] = merged_df['date'].dt.to_period('M').dt.to_timestamp()

# Group by month, age_category, household_type, and customer_segment
grouped_month = (
    merged_df.groupby(['month', 'age_category', 'household_type', 'customer_segment'])['total_sales']
    .sum()
    .reset_index()
)

# Define custom colors for customer segments
custom_colors = {
    'Premium': '#e377c2',   # pink
    'Budget': '#17becf',    # teal
    'Mainstream': '#bcbd22' # olive
}

# Count unique age categories to set facet_col_wrap
n_age_categories = grouped_month['age_category'].nunique()
# Set facet_col_wrap to ceil(n_age_categories / 2) to get 2 rows
import math
facet_col_wrap = math.ceil(n_age_categories / 2) if n_age_categories > 2 else n_age_categories

# --- Plotting ---
fig = px.line(
    grouped_month,
    x='month',
    y='total_sales',
    color='customer_segment',       # color by segment
    line_dash='household_type',     # dash by household type
    facet_col='age_category',       # side-by-side facets for age categories
    facet_col_wrap=facet_col_wrap,  # wrap facets into 2 rows
    color_discrete_map=custom_colors,
    title='Monthly Sales by Age, Household Type, and Segment',
    labels={
        'month': 'Month',
        'total_sales': 'Total Sales',
        'customer_segment': 'Customer Segment',
        'household_type': 'Household Type',
        'age_category': 'Age Category'
    },
    hover_data={
        'customer_segment': True,
        'household_type': True,
        'age_category': True,
        'total_sales': ':.2f',
        'month': '|%b %Y'   # show month + year in hover
    }
)

# Layout tweaks
fig.update_layout(
    yaxis_title='Total Sales',
    legend_title_text='Customer Segment',
    hovermode='x unified',
    height=500 + 250 * (math.ceil(n_age_categories / facet_col_wrap) - 1)  # increase height for more rows
)

# Format x-axis ticks to show months only
fig.update_xaxes(
    dtick="M1",           # one tick per month
    tickformat="%b %Y"    # e.g., Jan 2025
)

fig.update_yaxes(tickformat=",")   # thousands separator for sales

fig.show()
```