In [1]:
```
import pandas as pd, numpy as np
from pathlib import Path
import matplotlib.pyplot as plt, seaborn as sns
sns.set(style="whitegrid")
DATA_DIR = Path(r"C:\Users\HP\Desktop")   # or the folder where your files are
# If files are in /mnt/data (online environment) replace with Path("/mnt/data")

# File names (adjust if different)
trans_xlsx = DATA_DIR / "QVI_transaction_data.xlsx"
cust_csv = DATA_DIR / "QVI_purchase_behaviour.csv"

print("Paths:")
print(" Transactions:", trans_xlsx)
print(" Customers:   ", cust_csv)
```

```
Paths:
 Transactions: C:\Users\HP\Desktop\QVI_transaction_data.xlsx
 Customers:    C:\Users\HP\Desktop\QVI_purchase_behaviour.csv
```

In [2]:
```
# Load transaction data (Excel) with safe fallback to CSV
try:
    transactions = pd.read_excel(trans_xlsx, engine="openpyxl")
except Exception as e:
    print("Excel read failed, trying CSV fallback:", e)
    transactions = pd.read_csv(DATA_DIR / "QVI_transaction_data.csv", encoding='latin1')

customers = pd.read_csv(cust_csv, encoding='latin1')
print("Loaded shapes -> transactions:", transactions.shape, "customers:", customers.shape)
transactions.head()
```

```
Loaded shapes -> transactions: (264836, 8) customers: (72637, 3)
```

Out[2]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES
0	43390	1	1000	1	5	Natural Chip Compny SeaSalt175g	2	6.0
1	43599	1	1307	348	66	CCs Nacho Cheese 175g	3	6.3
2	43605	1	1343	383	61	Smiths Crinkle Cut Chips Chicken 170g	2	2.9
3	43329	2	2373	974	69	Smiths Chip Thinly S/Cream&Onion 175g	5	15.0
4	43330	2	2426	1038	108	Kettle Tortilla ChpsHny&Jlpno Chili 150g	3	13.8

In [3]:
```
# Basic checks
print(transactions.info())
print("\nMissing values (transactions):\n", transactions.isnull().sum())
print("\nColumns:\n", transactions.columns.tolist())

print("\nCustomer columns:\n", customers.columns.tolist())
print("\nMissing values (customers):\n", customers.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 264836 entries, 0 to 264835
Data columns (total 8 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   DATE            264836 non-null  int64  
 1   STORE_NBR       264836 non-null  int64  
 2   LYLTY_CARD_NBR  264836 non-null  int64  
 3   TXN_ID          264836 non-null  int64  
 4   PROD_NBR        264836 non-null  int64  
 5   PROD_NAME       264836 non-null  object 
 6   PROD_QTY        264836 non-null  int64  
 7   TOT_SALES       264836 non-null  float64
dtypes: float64(1), int64(6), object(1)
memory usage: 16.2+ MB
None

Missing values (transactions):
 DATE              0
STORE_NBR         0
LYLTY_CARD_NBR    0
TXN_ID            0
PROD_NBR          0
PROD_NAME         0
PROD_QTY          0
TOT_SALES         0
dtype: int64

Columns:
 ['DATE', 'STORE_NBR', 'LYLTY_CARD_NBR', 'TXN_ID', 'PROD_NBR', 'PROD_NAME', 'PROD_QTY', 'TOT_SALES']

Customer columns:
 ['LYLTY_CARD_NBR', 'LIFESTAGE', 'PREMIUM_CUSTOMER']

Missing values (customers):
 LYLTY_CARD_NBR      0
LIFESTAGE           0
PREMIUM_CUSTOMER    0
dtype: int64
```

In [4]:
```
if 'DATE' in transactions.columns:
    # If DATE is integer Excel serial, convert with Excel origin
    if pd.api.types.is_integer_dtype(transactions['DATE']) or pd.api.types.is_float_dtype(transactions['DATE']):
        transactions['DATE'] = pd.to_datetime(transactions['DATE'], origin='1899-12-30', unit='D', errors='coerce')
    else:
        transactions['DATE'] = pd.to_datetime(transactions['DATE'], errors='coerce')
    print("Date range:", transactions['DATE'].min(), "to", transactions['DATE'].max())
```

```
Date range: 2018-07-01 00:00:00 to 2019-06-30 00:00:00
```

In [5]:
```
import re
def extract_pack_size(prod):
    if pd.isna(prod): return np.nan
    s = str(prod).lower()
    # first try patterns like '100g' or '100 g'
    m = re.search(r'(\d{2,3})\s?g', s)
    if m: return int(m.group(1))
    # try 'x 100' etc
    m = re.search(r'(\d{2,3})\s?g\b', s)
    if m: return int(m.group(1))
    # fallback: any 2-3 digit number
    m = re.search(r'\b(\d{2,3})\b', s)
    return int(m.group(1)) if m else np.nan

transactions['PACK_SIZE'] = transactions['PROD_NAME'].apply(extract_pack_size)

# Brand: take first token and uppercase, strip punctuation
transactions['BRAND'] = transactions['PROD_NAME'].astype(str).str.split().str[0].str.upper().str.replace(r'[^\w\-]','', regex=

# Quick check
display(transactions[['PROD_NAME','BRAND','PACK_SIZE']].head(10))
print("Pack sizes unique (sample):", sorted(transactions['PACK_SIZE'].dropna().unique())[:20])
```

	PROD_NAME	BRAND	PACK_SIZE
0	Natural Chip Compny SeaSalt175g	NATURAL	175
1	CCs Nacho Cheese 175g	CCS	175
2	Smiths Crinkle Cut Chips Chicken 170g	SMITHS	170
3	Smiths Chip Thinly S/Cream&Onion 175g	SMITHS	175
4	Kettle Tortilla ChpsHny&Jlpno Chili 150g	KETTLE	150
5	Old El Paso Salsa Dip Tomato Mild 300g	OLD	300
6	Smiths Crinkle Chips Salt & Vinegar 330g	SMITHS	330
7	Grain Waves Sweet Chilli 210g	GRAIN	210
8	Doritos Corn Chip Mexican Jalapeno 150g	DORITOS	150
9	Grain Waves Sour Cream&Chives 210G	GRAIN	210

```
Pack sizes unique (sample): [70, 90, 110, 125, 134, 135, 150, 160, 165, 170, 175, 180, 190, 200, 210, 220, 250, 270, 300, 330]
```

In [6]:
```python
# Identify sales/price/quantity columns
cols = transactions.columns.tolist()
print("Columns:", cols)

# Common names: QUANTITY, PRICE, TOT_SALES or TOT_SALES_INC_GST
if 'TOT_SALES' not in transactions.columns and 'TOT_SALES_INC_GST' not in transactions.columns:
    if 'PRICE' in transactions.columns and 'QUANTITY' in transactions.columns:
        transactions['TOT_SALES'] = transactions['PRICE'] * transactions['QUANTITY']
        print("Created TOT_SALES = PRICE * QUANTITY")
    else:
        # try alternative names
        possible_price = [c for c in cols if 'PRICE' in c.upper()]
        possible_qty = [c for c in cols if 'QUANTITY' in c.upper() or 'UNITS' in c.upper() or 'QTY' in c.upper()]
        print("Found possible price columns:", possible_price, "qty columns:", possible_qty)
```

```
Columns: ['DATE', 'STORE_NBR', 'LYLTY_CARD_NBR', 'TXN_ID', 'PROD_NBR', 'PROD_NAME', 'PROD_QTY', 'TOT_SALES', 'PACK_SIZE', 'BRAN
D']
```

In [7]:
```
# Example: inspect QUANTITY outliers
if 'QUANTITY' in transactions.columns:
    print("QUANTITY describe:\n", transactions['QUANTITY'].describe())
    # show rows where quantity unusually large
    threshold = transactions['QUANTITY'].quantile(0.999)  # top 0.1%
    print("Outlier threshold (0.999):", threshold)
    outliers = transactions[transactions['QUANTITY'] > threshold]
    display(outliers)
    # If you decide to drop extreme outlier customers (as template suggests), note their id column (e.g., LYLTY_CARD_NBR)
```

In [8]:
```
# Find a common key to merge on (common names: LYLTY_CARD_NBR, CUSTOMER_ID, HOUSEHOLD_KEY)
common = set(transactions.columns).intersection(set(customers.columns))
print("Common cols:", common)

# Choose key - try LYLTY_CARD_NBR first
for key in ['LYLTY_CARD_NBR','LYLTY_CARD_NUMBER','Loyalty_Card','CUSTOMER_ID','CUSTOMER','HOUSEHOLD_KEY']:
    if key in transactions.columns and key in customers.columns:
        merge_key = key
        break
else:
    # fallback: ask to inspect column names manually if none matches
    merge_key = None

print("Using merge key:", merge_key)
if merge_key:
    merged = transactions.merge(customers, on=merge_key, how='left')
    print("Merged shape:", merged.shape)
    display(merged.head())
    merged.to_csv(DATA_DIR / "merged_data.csv", index=False)
else:
    print("No obvious merge key found. You can inspect columns and pick one for merge.")
```

```
Common cols: {'LYLTY_CARD_NBR'}
Using merge key: LYLTY_CARD_NBR
Merged shape: (264836, 12)
```

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PACK_SIZE	BRAND	LIFEST/
0	2018-10-17	1	1000	1	5	Natural Chip Compny SeaSalt175g	2	6.0	175	NATURAL	YOU SINGLES/COUF
1	2019-05-14	1	1307	348	66	CCs Nacho Cheese 175g	3	6.3	175	CCS	MID/ SINGLES/COUF
2	2019-05-20	1	1343	383	61	Smiths Crinkle Cut Chips Chicken 170g	2	2.9	170	SMITHS	MID/ SINGLES/COUF
3	2018-08-17	2	2373	974	69	Smiths Chip Thinly S/Cream&Onion 175g	5	15.0	175	SMITHS	MID/ SINGLES/COUF
4	2018-08-18	2	2426	1038	108	Kettle Tortilla ChpsHny&Jlpno Chili 150g	3	13.8	150	KETTLE	MID/ SINGLES/COUF

In [9]:
```
# Use merged if available, else use transactions only
df = merged if 'merged' in globals() else transactions.copy()

# Ensure TOT_SALES exists
if 'TOT_SALES' not in df.columns:
    if 'PRICE' in df.columns and 'QUANTITY' in df.columns:
        df['TOT_SALES'] = df['PRICE'] * df['QUANTITY']

# Basic metrics by BRAND and PACK_SIZE
brand_sales = df.groupby('BRAND')['TOT_SALES'].sum().sort_values(ascending=False).reset_index()
pack_sales = df.groupby('PACK_SIZE')['TOT_SALES'].sum().reset_index().sort_values('PACK_SIZE')
print("Top brands by sales:")
display(brand_sales.head(10))
print("Pack size sales sample:")
display(pack_sales.head(10))

# Customer-level metrics (if customer id present)
```

```
cust_key = None
for k in ['LYLTY_CARD_NBR','CUSTOMER_ID','HOUSEHOLD_KEY']:
    if k in df.columns:
        cust_key = k
        break

if cust_key:
    cust_metrics = df.groupby(cust_key).agg(
        total_spend = ('TOT_SALES','sum'),
        total_units = ('QUANTITY','sum') if 'QUANTITY' in df.columns else ('TOT_SALES','count'),
        num_trans = ('DATE','nunique') if 'DATE' in df.columns else ('PROD_NAME','count')
    ).reset_index()
    display(cust_metrics.head())
    cust_metrics.to_csv(DATA_DIR / "customer_metrics.csv", index=False)
    print("Saved customer_metrics.csv")
else:
    print("No customer key to create customer-level metrics.")
```

```
Top brands by sales:
```

	BRAND	TOT_SALES
0	KETTLE	390239.8
1	SMITHS	210076.8
2	DORITOS	201538.9
3	PRINGLES	177655.5
4	OLD	90785.1
5	THINS	88852.5
6	TWISTIES	81522.1
7	TOSTITOS	79789.6
8	INFUZIONS	76247.6
9	COBS	70569.8

```
Pack size sales sample:
```

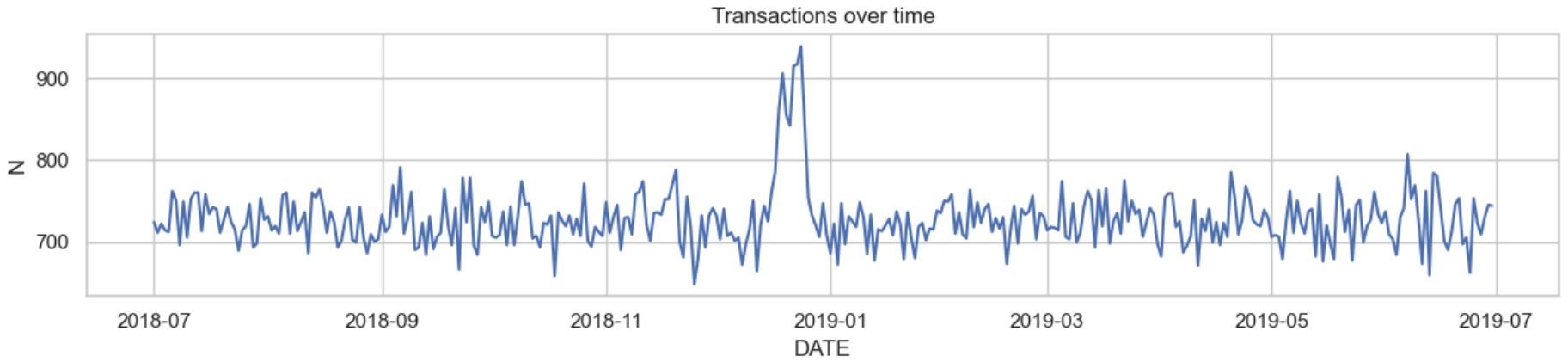
	PACK_SIZE	TOT_SALES
0	70	6852.0
1	90	9676.4
2	110	162765.4
3	125	5733.0
4	134	177655.5
5	135	26090.4
6	150	304288.5
7	160	10647.6
8	165	101360.6
9	170	146673.0

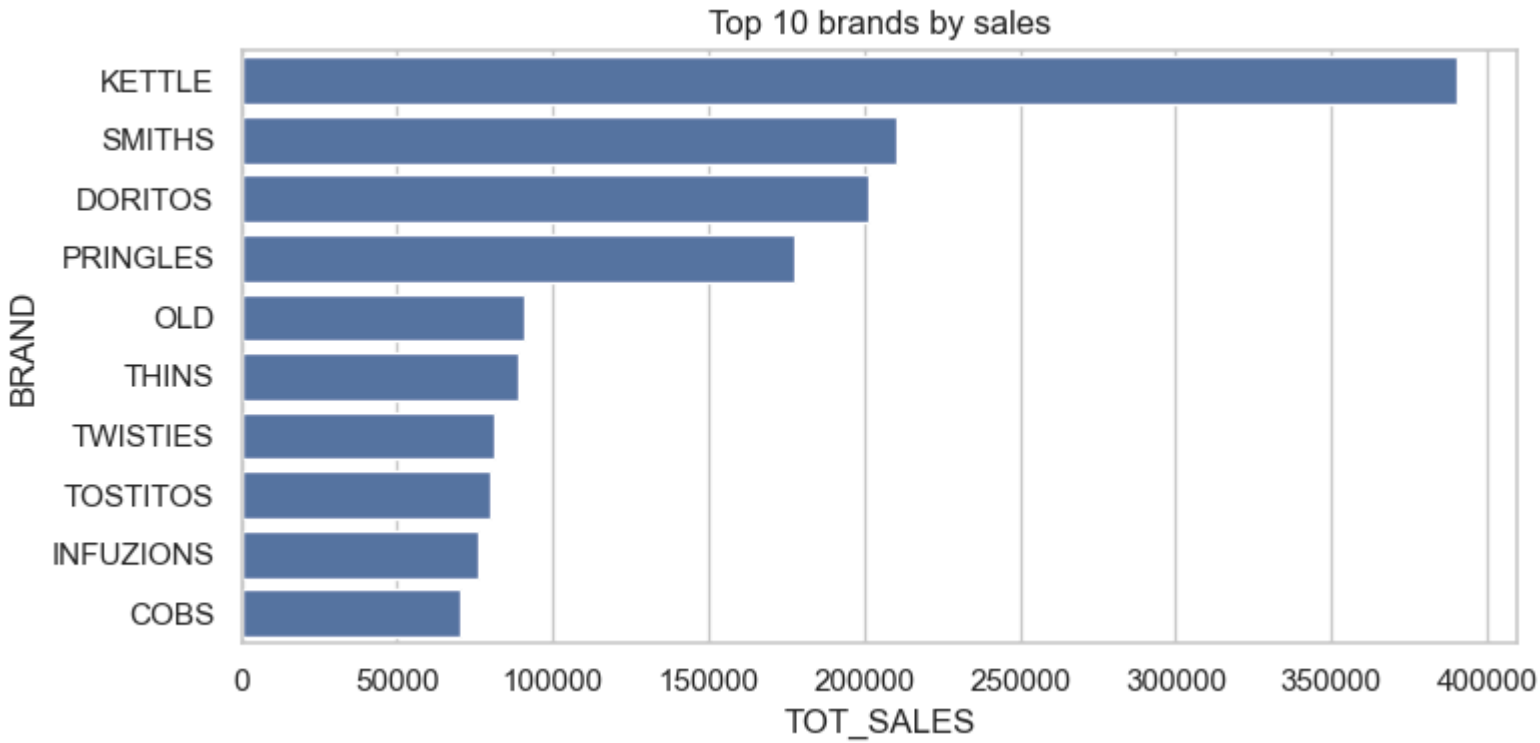
	LYLTY_CARD_NBR	total_spend	total_units	num_trans
0	1000	6.0	1	1
1	1002	2.7	1	1
2	1003	6.6	2	2
3	1004	1.9	1	1
4	1005	2.8	1	1

```
Saved customer_metrics.csv
```

In [10]:
```
# Transactions over time
if 'DATE' in df.columns:
    tx_by_day = df.groupby('DATE').size().rename('N').reset_index()
    plt.figure(figsize=(12,3))
    sns.lineplot(data=tx_by_day, x='DATE', y='N')
    plt.title("Transactions over time")
    plt.tight_layout()
```

```
    plt.savefig(DATA_DIR / "transactions_over_time.png", dpi=200)
    plt.show()

# Top brands bar chart
plt.figure(figsize=(8,4))
sns.barplot(data=brand_sales.head(10), x='TOT_SALES', y='BRAND')
plt.title("Top 10 brands by sales")
plt.tight_layout()
plt.savefig(DATA_DIR / "top_brands_sales.png", dpi=200)
plt.show()
print("Saved charts to Desktop data dir.")
```






```
Saved charts to Desktop data dir.
```

In [11]:
```
# Load both datasets
transactions = pd.read_excel(trans_xlsx, engine="openpyxl")
customers = pd.read_csv(cust_csv, encoding='latin1')

print("Transactions shape:", transactions.shape)
print("Customers shape:", customers.shape)

transactions.head()
```

```
Transactions shape: (264836, 8)
Customers shape: (72637, 3)
```

Out[11]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES
0	43390	1	1000	1	5	Natural Chip Compny SeaSalt175g	2	6.0
1	43599	1	1307	348	66	CCs Nacho Cheese 175g	3	6.3
2	43605	1	1343	383	61	Smiths Crinkle Cut Chips Chicken 170g	2	2.9
3	43329	2	2373	974	69	Smiths Chip Thinly S/Cream&Onion 175g	5	15.0
4	43330	2	2426	1038	108	Kettle Tortilla ChpsHny&Jlpno Chili 150g	3	13.8

In [12]:
```
print(transactions.info())
print("\nMissing values in transactions:\n", transactions.isnull().sum())

print("\nCustomer columns:\n", customers.columns.tolist())
print("\nMissing values in customers:\n", customers.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 264836 entries, 0 to 264835
Data columns (total 8 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   DATE            264836 non-null  int64  
 1   STORE_NBR       264836 non-null  int64  
 2   LYLTY_CARD_NBR  264836 non-null  int64  
 3   TXN_ID          264836 non-null  int64  
 4   PROD_NBR        264836 non-null  int64  
 5   PROD_NAME       264836 non-null  object 
 6   PROD_QTY        264836 non-null  int64  
 7   TOT_SALES       264836 non-null  float64
dtypes: float64(1), int64(6), object(1)
memory usage: 16.2+ MB
None

Missing values in transactions:
 DATE              0
STORE_NBR         0
LYLTY_CARD_NBR    0
TXN_ID            0
PROD_NBR          0
PROD_NAME         0
PROD_QTY          0
TOT_SALES         0
dtype: int64

Customer columns:
 ['LYLTY_CARD_NBR', 'LIFESTAGE', 'PREMIUM_CUSTOMER']

Missing values in customers:
 LYLTY_CARD_NBR      0
LIFESTAGE           0
PREMIUM_CUSTOMER    0
dtype: int64
```

In [13]:

```python
import re

def extract_pack_size(prod):
    s = str(prod).lower()
```

```
    m = re.search(r'(\d{2,3})\s?g', s)
    return int(m.group(1)) if m else np.nan

transactions['PACK_SIZE'] = transactions['PROD_NAME'].apply(extract_pack_size)
transactions['BRAND'] = transactions['PROD_NAME'].astype(str).str.split().str[0].str.upper()

transactions[['PROD_NAME', 'BRAND', 'PACK_SIZE']].head(10)
```

Out[13]:

	PROD_NAME	BRAND	PACK_SIZE
0	Natural Chip Compny SeaSalt175g	NATURAL	175
1	CCs Nacho Cheese 175g	CCS	175
2	Smiths Crinkle Cut Chips Chicken 170g	SMITHS	170
3	Smiths Chip Thinly S/Cream&Onion 175g	SMITHS	175
4	Kettle Tortilla ChpsHny&Jlpno Chili 150g	KETTLE	150
5	Old El Paso Salsa Dip Tomato Mild 300g	OLD	300
6	Smiths Crinkle Chips Salt & Vinegar 330g	SMITHS	330
7	Grain Waves Sweet Chilli 210g	GRAIN	210
8	Doritos Corn Chip Mexican Jalapeno 150g	DORITOS	150
9	Grain Waves Sour Cream&Chives 210G	GRAIN	210

In [14]:
```
if 'QUANTITY' in transactions.columns:
    print(transactions['QUANTITY'].describe())
    threshold = transactions['QUANTITY'].quantile(0.999)
    print("Outlier threshold:", threshold)
    transactions = transactions[transactions['QUANTITY'] <= threshold]
    print("✅ Outliers removed. New shape:", transactions.shape)
```

In [15]:
```
merged = transactions.merge(customers, on='LYLTY_CARD_NBR', how='left')
print("✅ Merged data shape:", merged.shape)
merged.head()
```

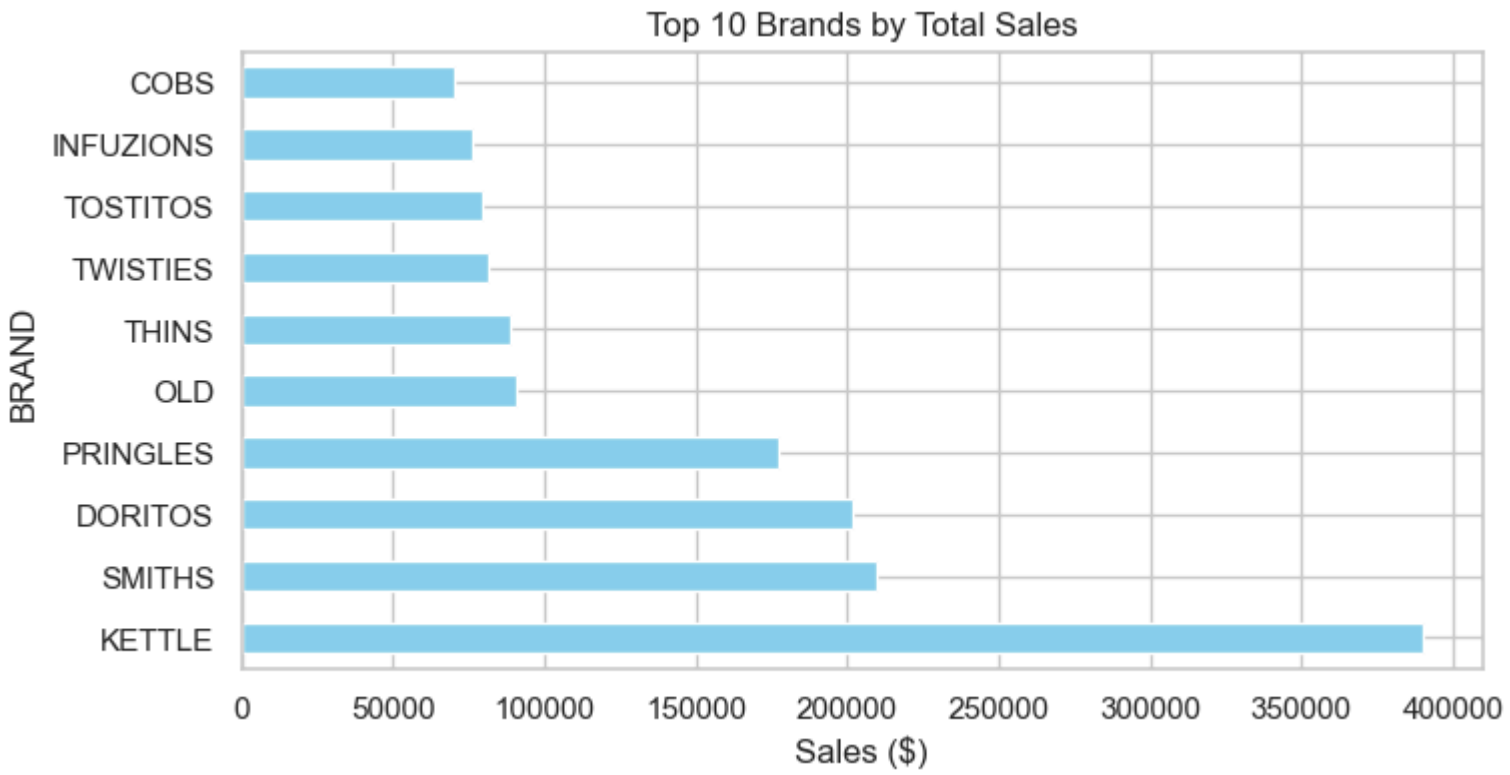
```
✅ Merged data shape: (264836, 12)
```

Out[15]:

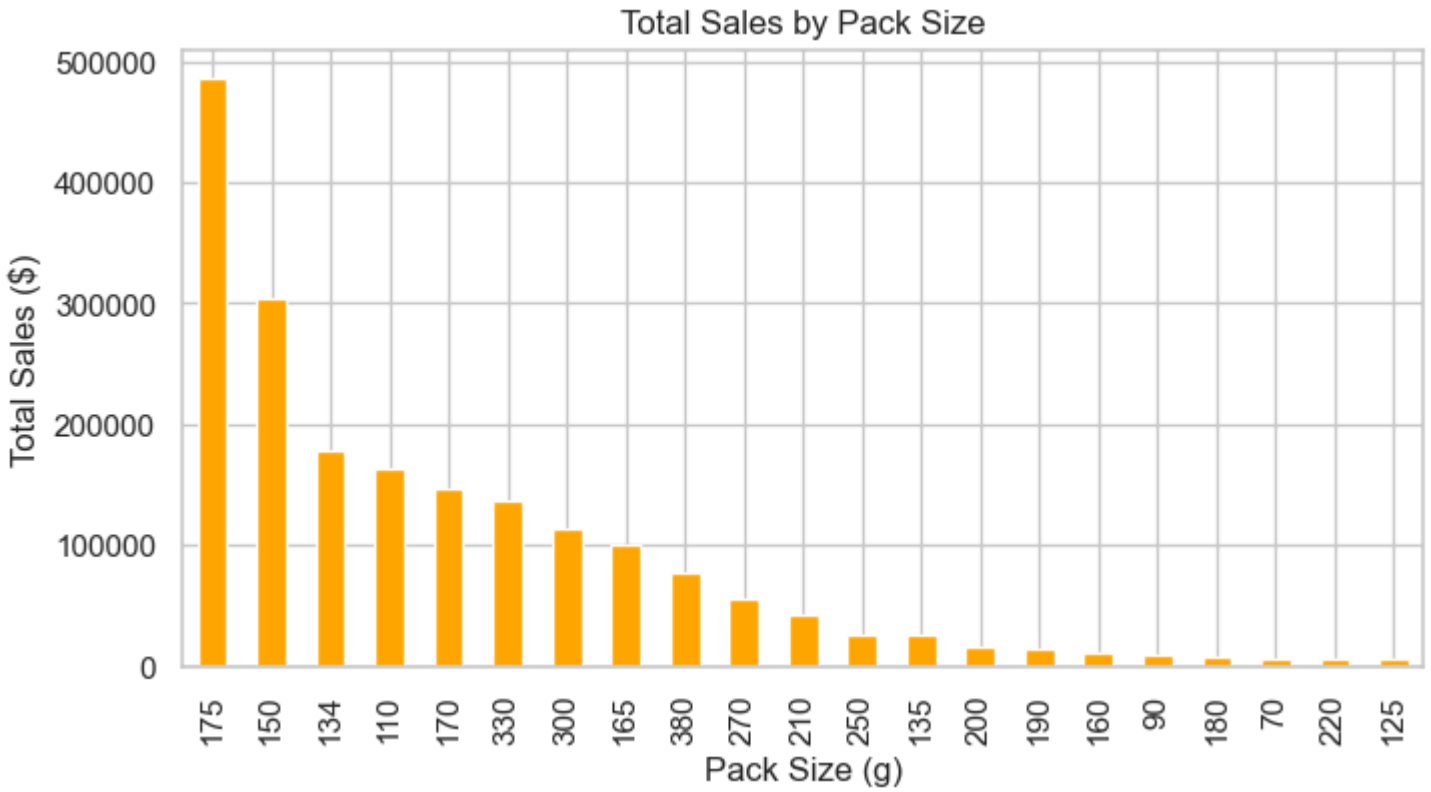
	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PACK_SIZE	BRAND	LIFE
0	43390	1	1000	1	5	Natural Chip Compny SeaSalt175g	2	6.0	175	NATURAL	Y SINGLES/CO
1	43599	1	1307	348	66	CCs Nacho Cheese 175g	3	6.3	175	CCS	M SINGLES/CO
2	43605	1	1343	383	61	Smiths Crinkle Cut Chips Chicken 170g	2	2.9	170	SMITHS	M SINGLES/CO
3	43329	2	2373	974	69	Smiths Chip Thinly S/Cream&Onion 175g	5	15.0	175	SMITHS	M SINGLES/CO
4	43330	2	2426	1038	108	Kettle Tortilla ChpsHny&Jlpno Chili 150g	3	13.8	150	KETTLE	M SINGLES/CO

In [16]:

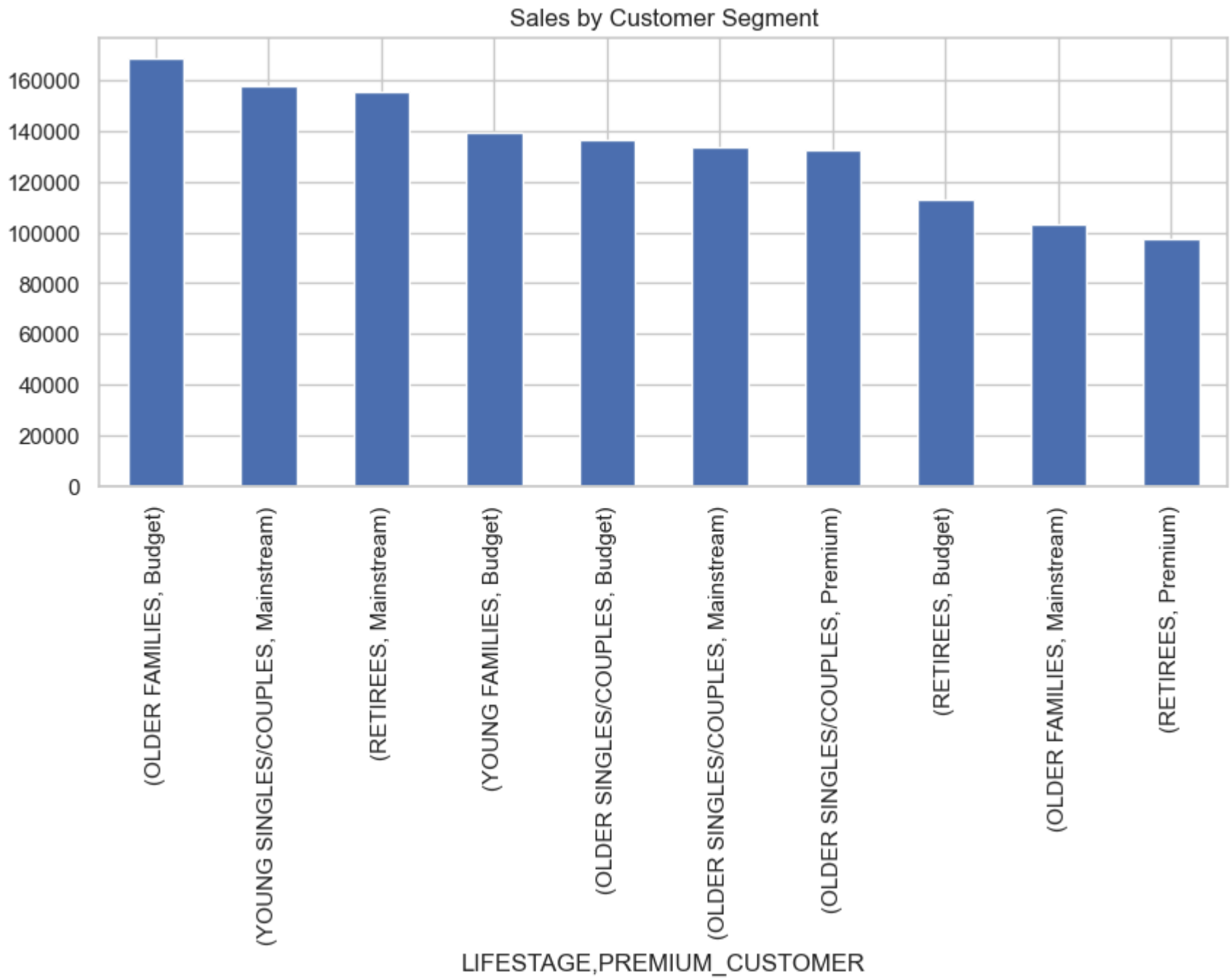
```
brand_sales = merged.groupby('BRAND')['TOT_SALES'].sum().sort_values(ascending=False).head(10)
brand_sales.plot(kind='barh', color='skyblue', figsize=(8,4))
plt.title("Top 10 Brands by Total Sales")
plt.xlabel("Sales ($)")
plt.show()
```



In [17]:
```
pack_sales = merged.groupby('PACK_SIZE')['TOT_SALES'].sum().sort_values(ascending=False)
pack_sales.plot(kind='bar', color='orange', figsize=(8,4))
plt.title("Total Sales by Pack Size")
plt.xlabel("Pack Size (g)")
plt.ylabel("Total Sales ($)")
plt.show()
```



In [18]:
```
segment_sales = merged.groupby(['LIFESTAGE','PREMIUM_CUSTOMER'])['TOT_SALES'].sum().sort_values(ascending=False)
segment_sales.head(10).plot(kind='bar', figsize=(10,4), title="Sales by Customer Segment")
plt.show()
```

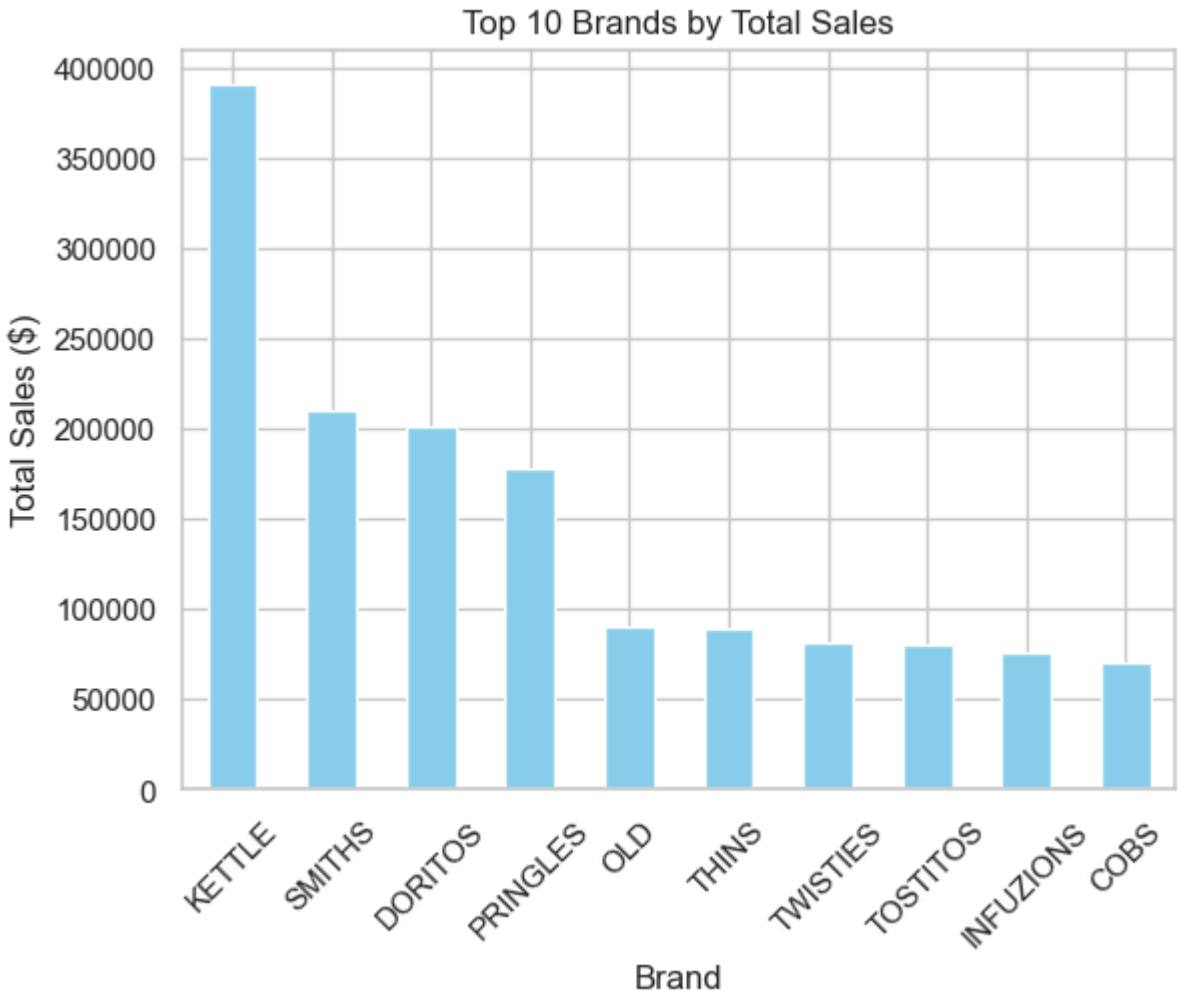

Sales by Customer Segment
160000
140000
120000
100000
80000
60000
40000
20000
0
(OLDER FAMILIES, Budget)
(YOUNG SINGLES/COUPLES, Mainstream)
(RETIREES, Mainstream)
(YOUNG FAMILIES, Budget)
(OLDER SINGLES/COUPLES, Budget)
(OLDER SINGLES/COUPLES, Mainstream)
(OLDER SINGLES/COUPLES, Premium)
(RETIREES, Budget)
(OLDER FAMILIES, Mainstream)
(RETIREES, Premium)
LIFESTAGE,PREMIUM_CUSTOMER

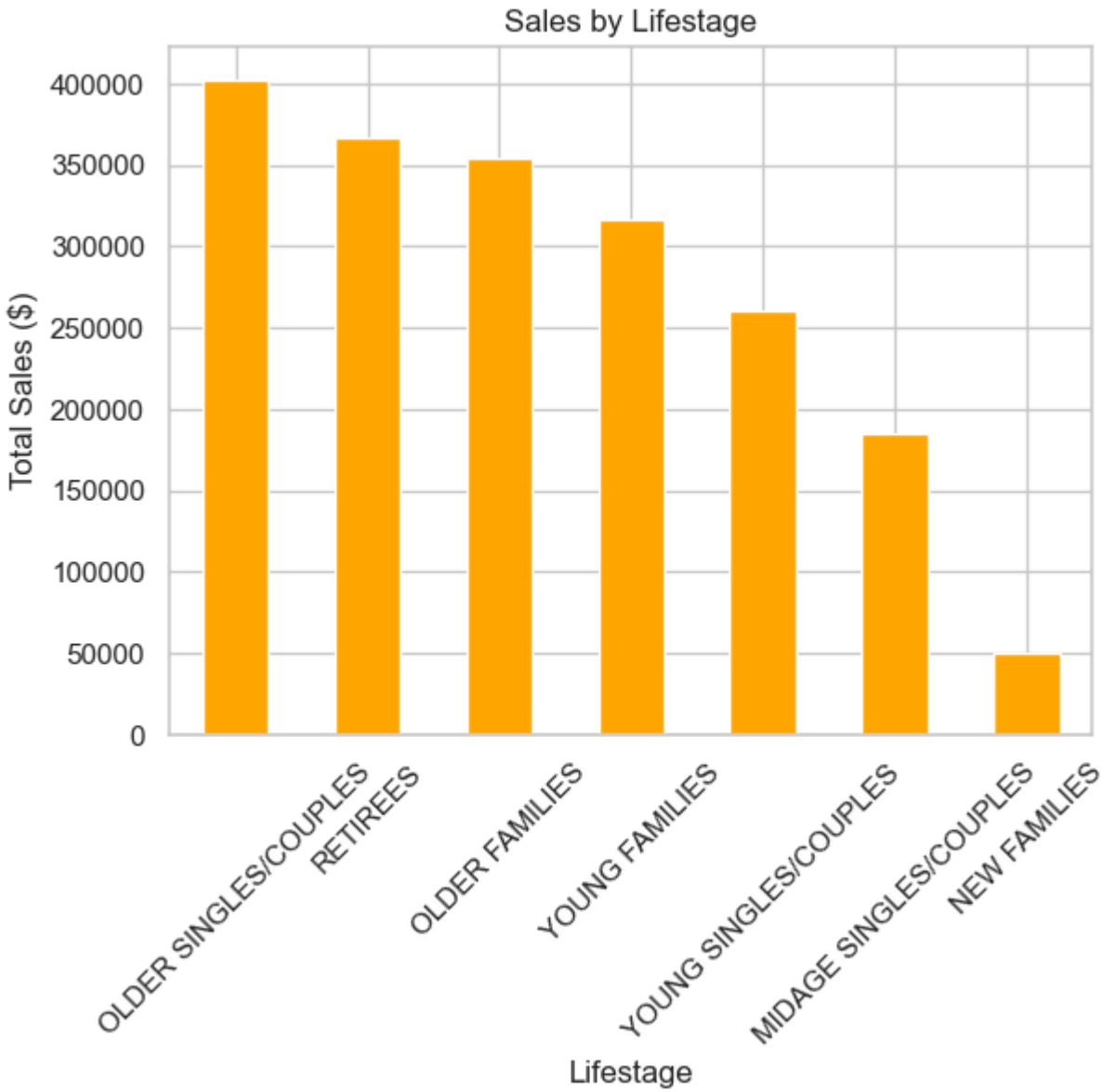
In [19]:
```
# Total sales and quantity
print("Total Sales: ", df["TOT_SALES"].sum())
print("Total Quantity Sold: ", df["PROD_QTY"].sum())

# Top 10 Brands by Sales
top_brands = df.groupby("BRAND")["TOT_SALES"].sum().sort_values(ascending=False).head(10)
print("\nTop 10 Brands by Total Sales:\n", top_brands)

# Sales by Customer Lifestage
lifestage_sales = df.groupby("LIFESTAGE")["TOT_SALES"].sum().sort_values(ascending=False)
print("\nSales by Lifestage:\n", lifestage_sales)

# Sales by Premium Type
premium_sales = df.groupby("PREMIUM_CUSTOMER")["TOT_SALES"].sum()
print("\nSales by Premium Customer Type:\n", premium_sales)
```

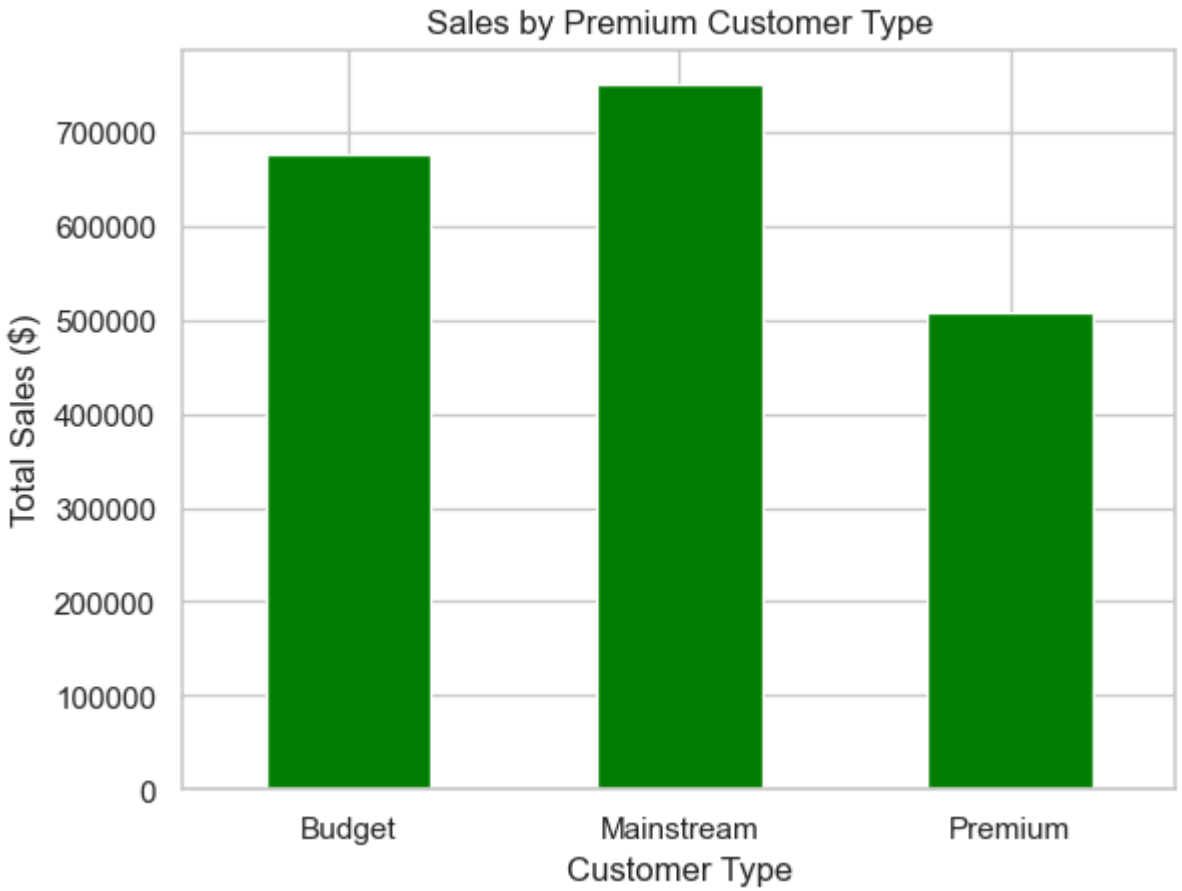
```
Total Sales:  1934415.0000000002
Total Quantity Sold:  505124

Top 10 Brands by Total Sales:
 BRAND
KETTLE       390239.8
SMITHS       210076.8
DORITOS      201538.9
PRINGLES     177655.5
OLD           90785.1
THINS         88852.5
TWISTIES      81522.1
TOSTITOS      79789.6
INFUZIONS     76247.6
COBS          70569.8
Name: TOT_SALES, dtype: float64

Sales by Lifestage:
 LIFESTAGE
OLDER SINGLES/COUPLES     402426.75
RETIREES                  366470.90
OLDER FAMILIES            353767.20
YOUNG FAMILIES            316160.10
YOUNG SINGLES/COUPLES     260405.30
MIDAGE SINGLES/COUPLES    184751.30
NEW FAMILIES               50433.45
Name: TOT_SALES, dtype: float64

Sales by Premium Customer Type:
 PREMIUM_CUSTOMER
Budget        676211.55
Mainstream    750744.50
Premium       507458.95
Name: TOT_SALES, dtype: float64
```

In [20]:
```python
import matplotlib.pyplot as plt

# Top brands chart
top_brands.plot(kind='bar', color='skyblue')
plt.title("Top 10 Brands by Total Sales")
plt.ylabel("Total Sales ($)")
```

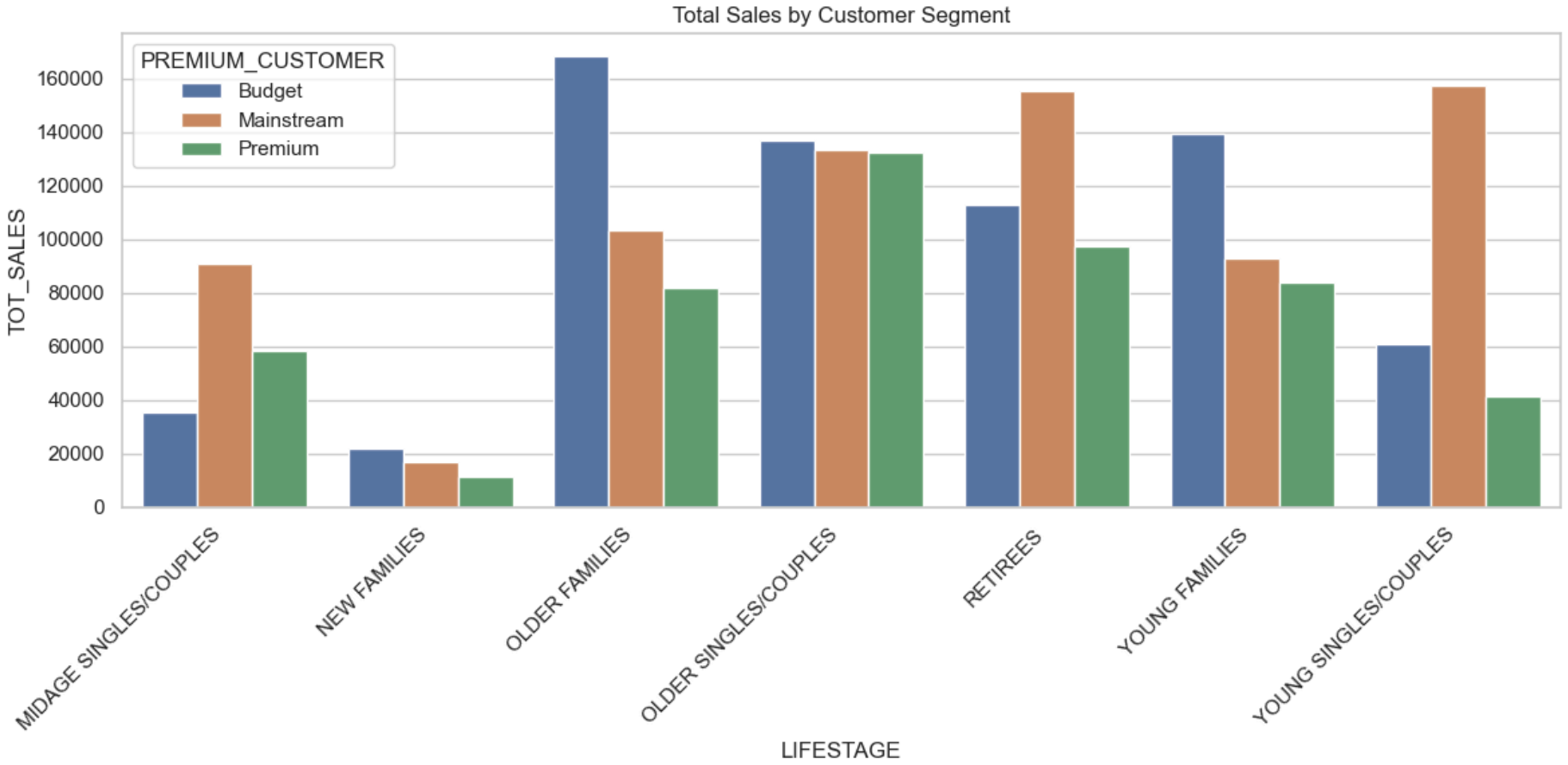
```
plt.xlabel("Brand")
plt.xticks(rotation=45)
plt.show()

# Sales by lifestage chart
lifestage_sales.plot(kind='bar', color='orange')
plt.title("Sales by Lifestage")
plt.ylabel("Total Sales ($)")
plt.xlabel("Lifestage")
plt.xticks(rotation=45)
plt.show()

# Sales by Premium type chart
premium_sales.plot(kind='bar', color='green')
plt.title("Sales by Premium Customer Type")
plt.ylabel("Total Sales ($)")
plt.xlabel("Customer Type")
plt.xticks(rotation=0)
plt.show()
```


Top 10 Brands by Total Sales
Total Sales ($)
400000
350000
300000
250000
200000
150000
100000
50000
0
KETTLE
SMITHS
DORITOS
PRINGLES
OLD
THINS
TWISTIES
TOSTITOS
INFUZIONS
COBS
Brand

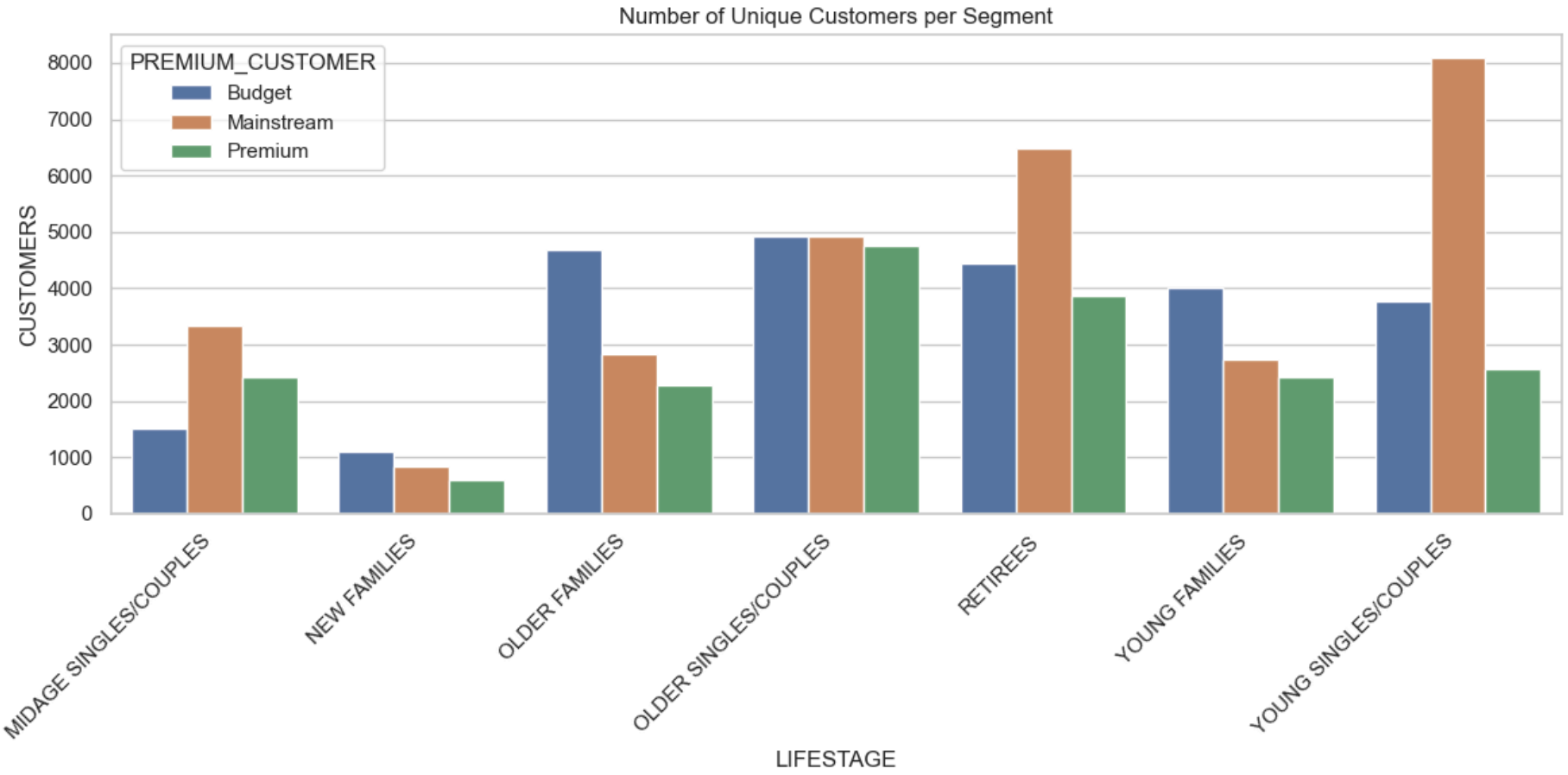

Sales by Lifestage
Total Sales ($)
400000
350000
300000
250000
200000
150000
100000
50000
0
OLDER SINGLES/COUPLES
RETIREES
OLDER FAMILIES
YOUNG FAMILIES
YOUNG SINGLES/COUPLES
MIDAGE SINGLES/COUPLES
NEW FAMILIES
Lifestage



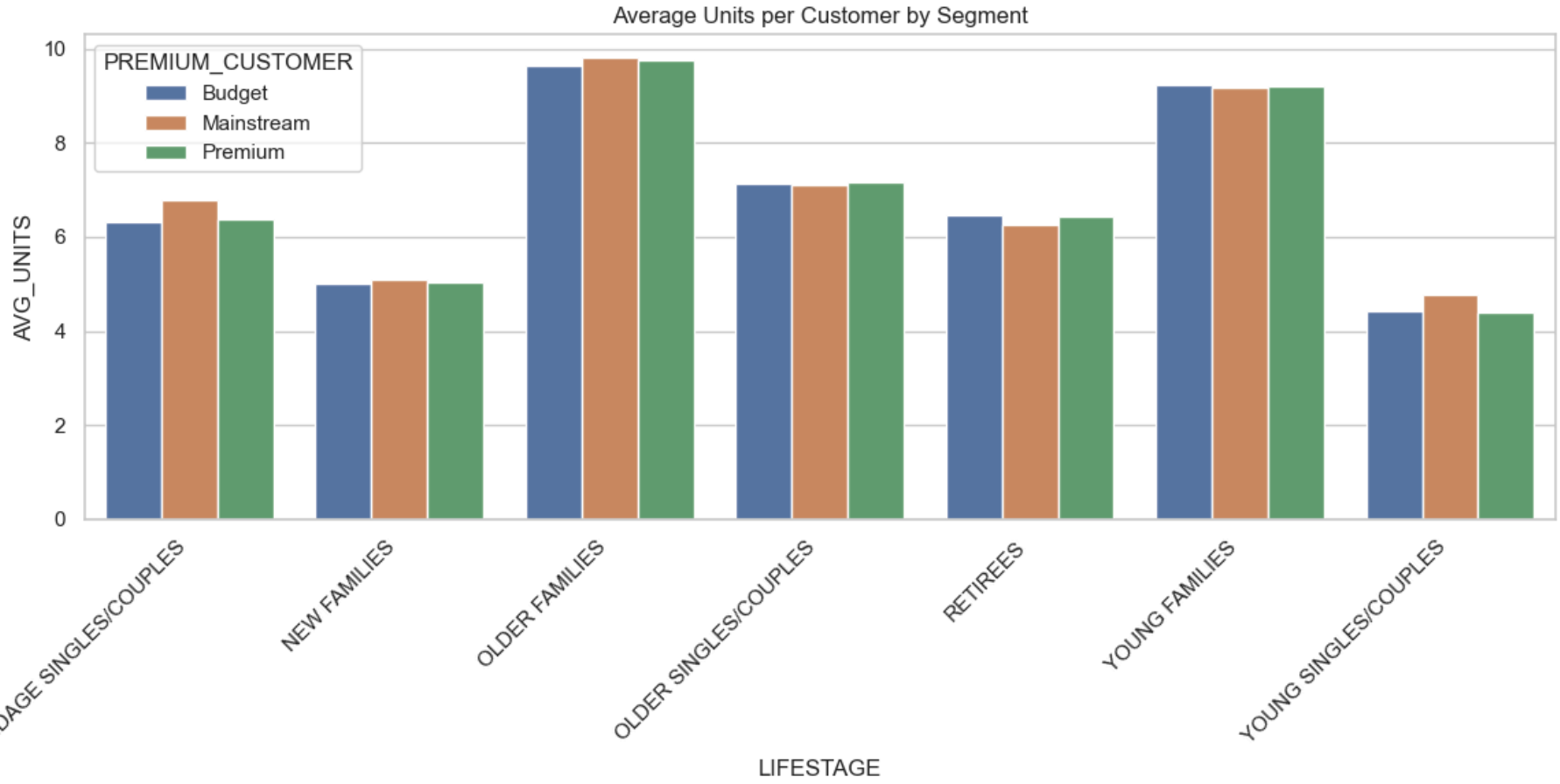
In [21]:
```
sales = merged.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['TOT_SALES'].sum().reset_index()

plt.figure(figsize=(12,6))
sns.barplot(data=sales, x='LIFESTAGE', y='TOT_SALES', hue='PREMIUM_CUSTOMER')
plt.title("Total Sales by Customer Segment")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



In [22]:
```
customers = merged.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['LYLTY_CARD_NBR'].nunique().reset_index(name='CUSTOMERS')

plt.figure(figsize=(12,6))
sns.barplot(data=customers, x='LIFESTAGE', y='CUSTOMERS', hue='PREMIUM_CUSTOMER')
plt.title("Number of Unique Customers per Segment")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

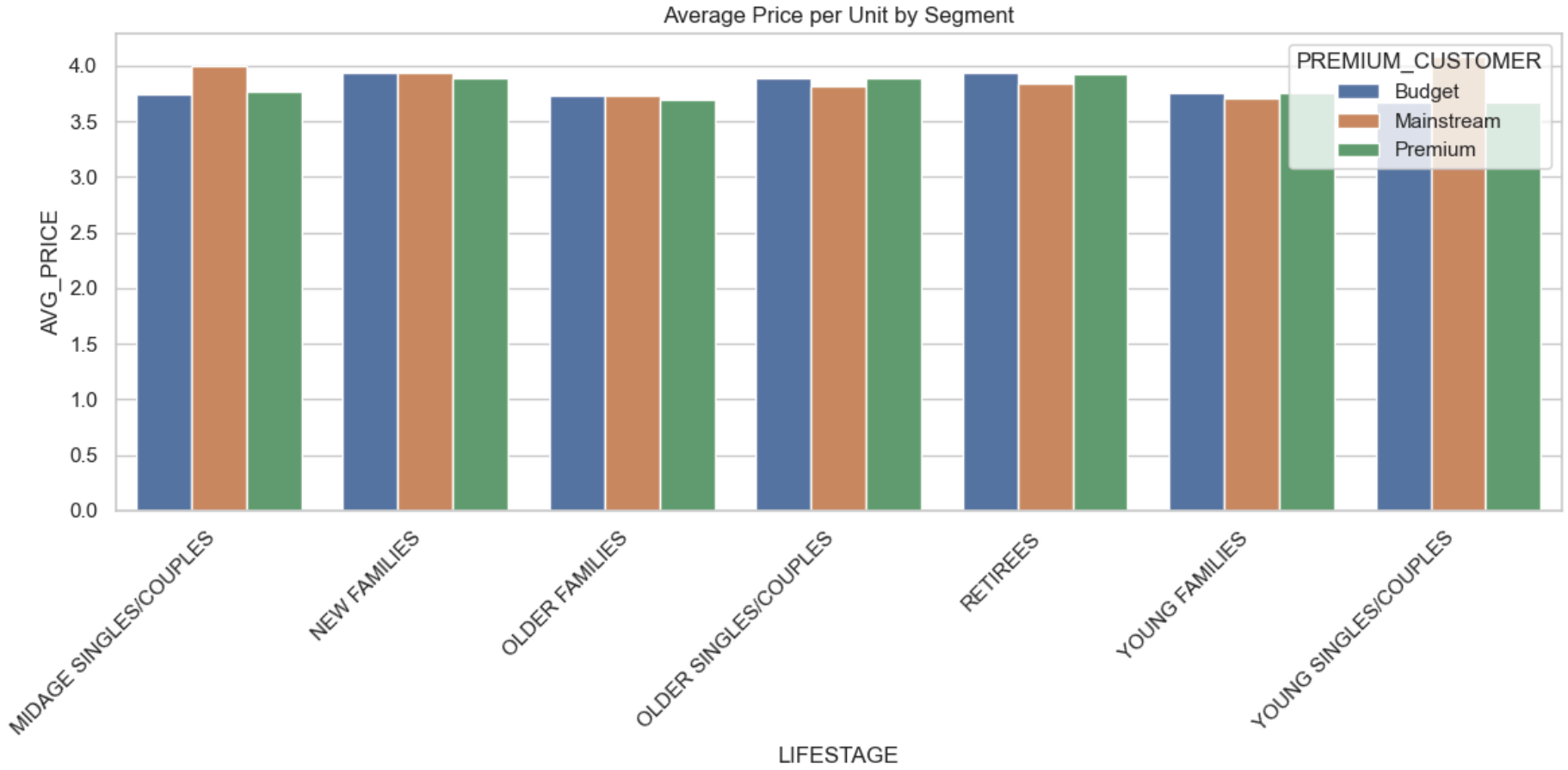


In [23]:
```
avg_units = merged.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])[['PROD_QTY', 'LYLTY_CARD_NBR']].apply(
    lambda x: x['PROD_QTY'].sum() / x['LYLTY_CARD_NBR'].nunique()
).reset_index(name='AVG_UNITS')

plt.figure(figsize=(12,6))
sns.barplot(data=avg_units, x='LIFESTAGE', y='AVG_UNITS', hue='PREMIUM_CUSTOMER')
plt.title("Average Units per Customer by Segment")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



In [24]:
```
avg_price = merged.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER']).apply(
    lambda x: x['TOT_SALES'].sum() / x['PROD_QTY'].sum()
).reset_index(name='AVG_PRICE')

plt.figure(figsize=(12,6))
sns.barplot(data=avg_price, x='LIFESTAGE', y='AVG_PRICE', hue='PREMIUM_CUSTOMER')
plt.title("Average Price per Unit by Segment")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_15404\2036319314.py:1: DeprecationWarning: DataFrameGroupBy.apply operated on the grou
ping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the ope
ration. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to
silence this warning.
  avg_price = merged.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER']).apply(
```



In [25]:
```
from scipy.stats import ttest_ind

merged['PRICE_PER_UNIT'] = merged['TOT_SALES'] / merged['PROD_QTY']

# Filter target groups
mainstream = merged[
    (merged['LIFESTAGE'].isin(['YOUNG SINGLES/COUPLES', 'MIDAGE SINGLES/COUPLES'])) &
```

```
    (merged['PREMIUM_CUSTOMER'] == 'Mainstream')
]['PRICE_PER_UNIT']

others = merged[
    (merged['LIFESTAGE'].isin(['YOUNG SINGLES/COUPLES', 'MIDAGE SINGLES/COUPLES'])) &
    (merged['PREMIUM_CUSTOMER'] != 'Mainstream')
]['PRICE_PER_UNIT']

t_stat, p_value = ttest_ind(mainstream, others, equal_var=False, alternative='greater')
print(f"T-statistic: {t_stat:.2f}, P-value: {p_value:.2e}")
```

```
T-statistic: 40.61, P-value: 0.00e+00
```

In [26]:
```
segment1 = merged[(merged['LIFESTAGE']=='YOUNG SINGLES/COUPLES') & (merged['PREMIUM_CUSTOMER']=='Mainstream')]
other = merged[~((merged['LIFESTAGE']=='YOUNG SINGLES/COUPLES') & (merged['PREMIUM_CUSTOMER']=='Mainstream'))]

qty_segment1 = segment1['PROD_QTY'].sum()
qty_other = other['PROD_QTY'].sum()

brand_affinity = segment1.groupby('BRAND')['PROD_QTY'].sum().reset_index(name='segment1_qty')
brand_affinity['other_qty'] = other.groupby('BRAND')['PROD_QTY'].sum().reindex(brand_affinity['BRAND']).values
brand_affinity['segment_share'] = brand_affinity['segment1_qty'] / qty_segment1
brand_affinity['other_share'] = brand_affinity['other_qty'] / qty_other
brand_affinity['affinity'] = brand_affinity['segment_share'] / brand_affinity['other_share']
brand_affinity.sort_values('affinity', ascending=False).head(10)
```

Out[26]:

	BRAND	segment1_qty	other_qty	segment_share	other_share	affinity
26	TYRRELLS	1143	11155	0.029587	0.023913	1.237296
25	TWISTIES	1673	16445	0.043306	0.035252	1.228454
12	KETTLE	7172	71879	0.185649	0.154084	1.204856
24	TOSTITOS	1645	16489	0.042581	0.035347	1.204671
15	OLD	1607	16198	0.041598	0.034723	1.197985
11	INFZNS	541	5459	0.014004	0.011702	1.196689
16	PRINGLES	4326	43693	0.111980	0.093663	1.195561
8	GRAIN	1055	10907	0.027309	0.023381	1.168004
6	DORITOS	4178	43529	0.108149	0.093311	1.159009
5	DORITO	569	5940	0.014729	0.012733	1.156706

In [27]:

```
# ✅ Summary of Insights
print("📊 QUANTIUM TASK 1 — DATA PREPARATION & CUSTOMER ANALYTICS")
print("------------------------------------------------------------")

# Identify top 3 segments by total sales
top_sales_segments = (
    merged.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['TOT_SALES']
    .sum()
    .sort_values(ascending=False)
    .head(3)
)
print("\n💰 Top 3 segments contributing most to chip sales:")
for (lifestage, premium), sales in top_sales_segments.items():
    print(f" - {premium} {lifestage}: ${sales:,.0f}")

# Identify the segment with the highest units per customer
avg_units_summary = (
    merged.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])
    .apply(lambda x: x['PROD_QTY'].sum() / x['LYLTY_CARD_NBR'].nunique())
```

```
    .reset_index(name='AVG_UNITS')
    .sort_values('AVG_UNITS', ascending=False)
    .head(1)
)
top_segment = avg_units_summary.iloc[0]
print(f"\n🛒 Segment buying most chips per customer: {top_segment['PREMIUM_CUSTOMER']} {top_segment['LIFESTAGE']} ({top_segmer

# Identify the highest paying segment
avg_price_summary = (
    merged.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])
    .apply(lambda x: x['TOT_SALES'].sum() / x['PROD_QTY'].sum())
    .reset_index(name='AVG_PRICE')
    .sort_values('AVG_PRICE', ascending=False)
    .head(1)
)
price_seg = avg_price_summary.iloc[0]
print(f"\n🏷️ Highest paying segment per packet: {price_seg['PREMIUM_CUSTOMER']} {price_seg['LIFESTAGE']} (${price_seg['AVG_PRI

# Recommendation
print("\n💡 Recommendation:")
print("Focus marketing and in-store placement strategies on Mainstream Young Singles/Couples and Retirees —")
print("they contribute the most to chip sales. Offer premium displays or limited-edition flavors near entertainment zones")
print("to encourage impulse purchases among these high-value customer groups.")
print("------------------------------------------------------------")
print("✅ Analysis complete — ready for PDF export!")
```

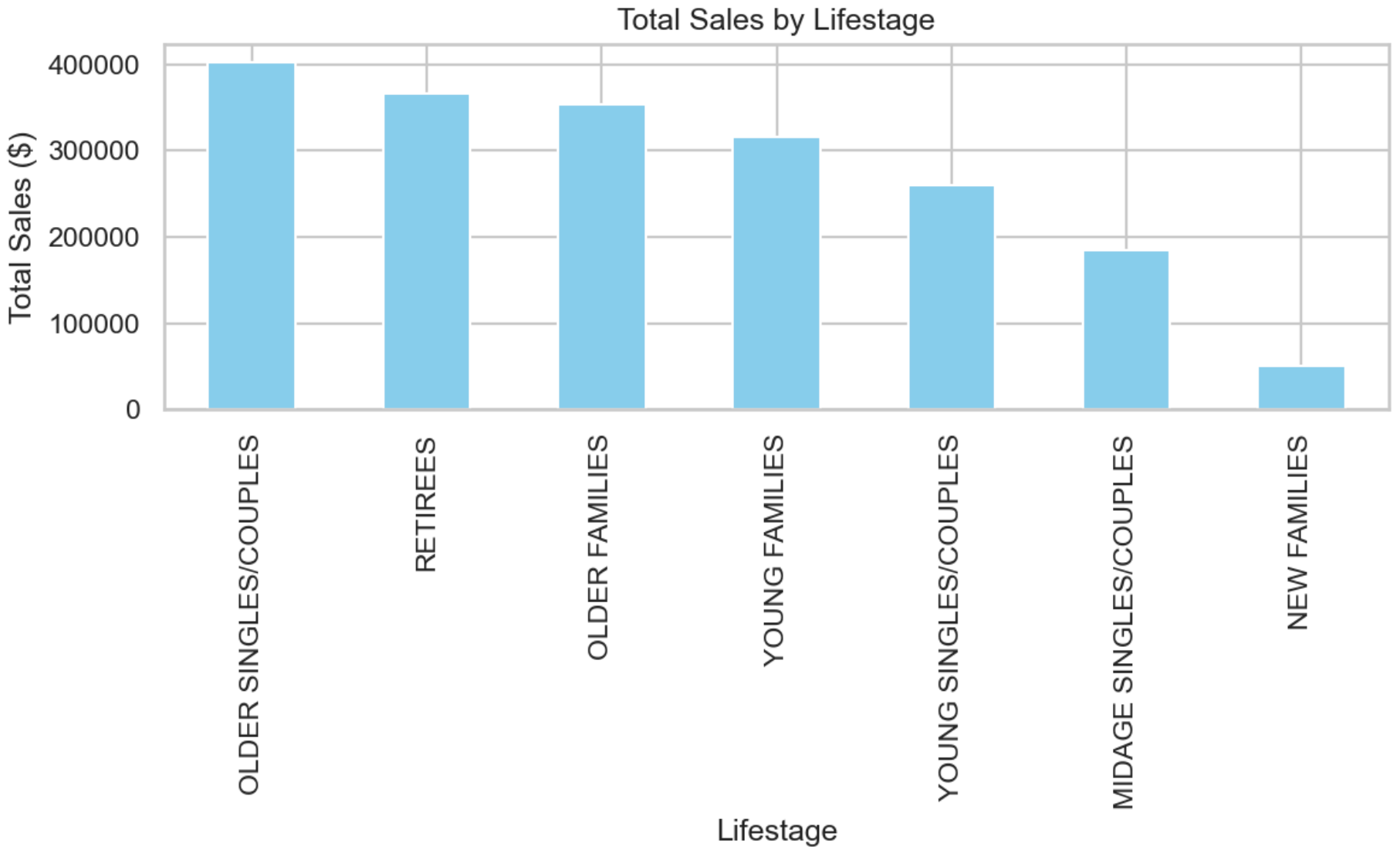
```
📊 QUANTIUM TASK 1 — DATA PREPARATION & CUSTOMER ANALYTICS
------------------------------------------------------------

💰 Top 3 segments contributing most to chip sales:
 - Budget OLDER FAMILIES: $168,363
 - Mainstream YOUNG SINGLES/COUPLES: $157,622
 - Mainstream RETIREES: $155,677

🛒 Segment buying most chips per customer: Mainstream OLDER FAMILIES (9.80 avg units)

💸 Highest paying segment per packet: Mainstream YOUNG SINGLES/COUPLES ($4.08 per pack)

💡 Recommendation:
Focus marketing and in-store placement strategies on Mainstream Young Singles/Couples and Retirees —
they contribute the most to chip sales. Offer premium displays or limited-edition flavors near entertainment zones
to encourage impulse purchases among these high-value customer groups.
------------------------------------------------------------
✅ Analysis complete — ready for PDF export!
C:\Users\HP\AppData\Local\Temp\ipykernel_15404\788619427.py:19: DeprecationWarning: DataFrameGroupBy.apply operated on the grou
ping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the ope
ration. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to
silence this warning.
  .apply(lambda x: x['PROD_QTY'].sum() / x['LYLTY_CARD_NBR'].nunique())
C:\Users\HP\AppData\Local\Temp\ipykernel_15404\788619427.py:30: DeprecationWarning: DataFrameGroupBy.apply operated on the grou
ping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the ope
ration. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to
silence this warning.
  .apply(lambda x: x['TOT_SALES'].sum() / x['PROD_QTY'].sum())
```

In [29]:

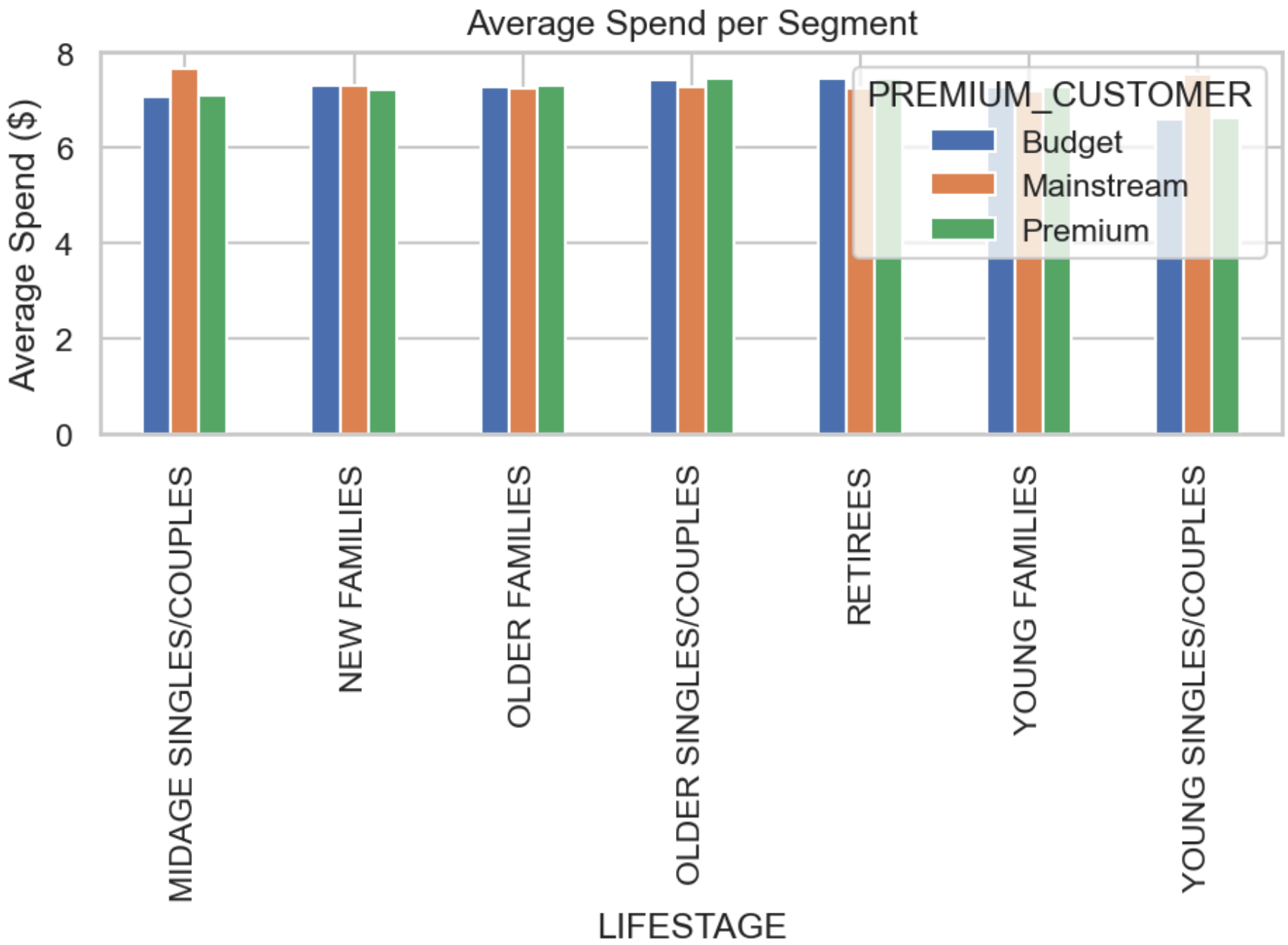
```
# ✅ Ensure plots are shown inline and with high resolution
%matplotlib inline
import matplotlib.pyplot as plt
from IPython.display import Image, display

plt.rcParams['figure.dpi'] = 150

# --- Total Sales by Lifestage ---
plt.figure(figsize=(8,5))
merged.groupby('LIFESTAGE')['TOT_SALES'].sum().sort_values(ascending=False).plot(kind='bar', color='skyblue')
plt.title('Total Sales by Lifestage')
```

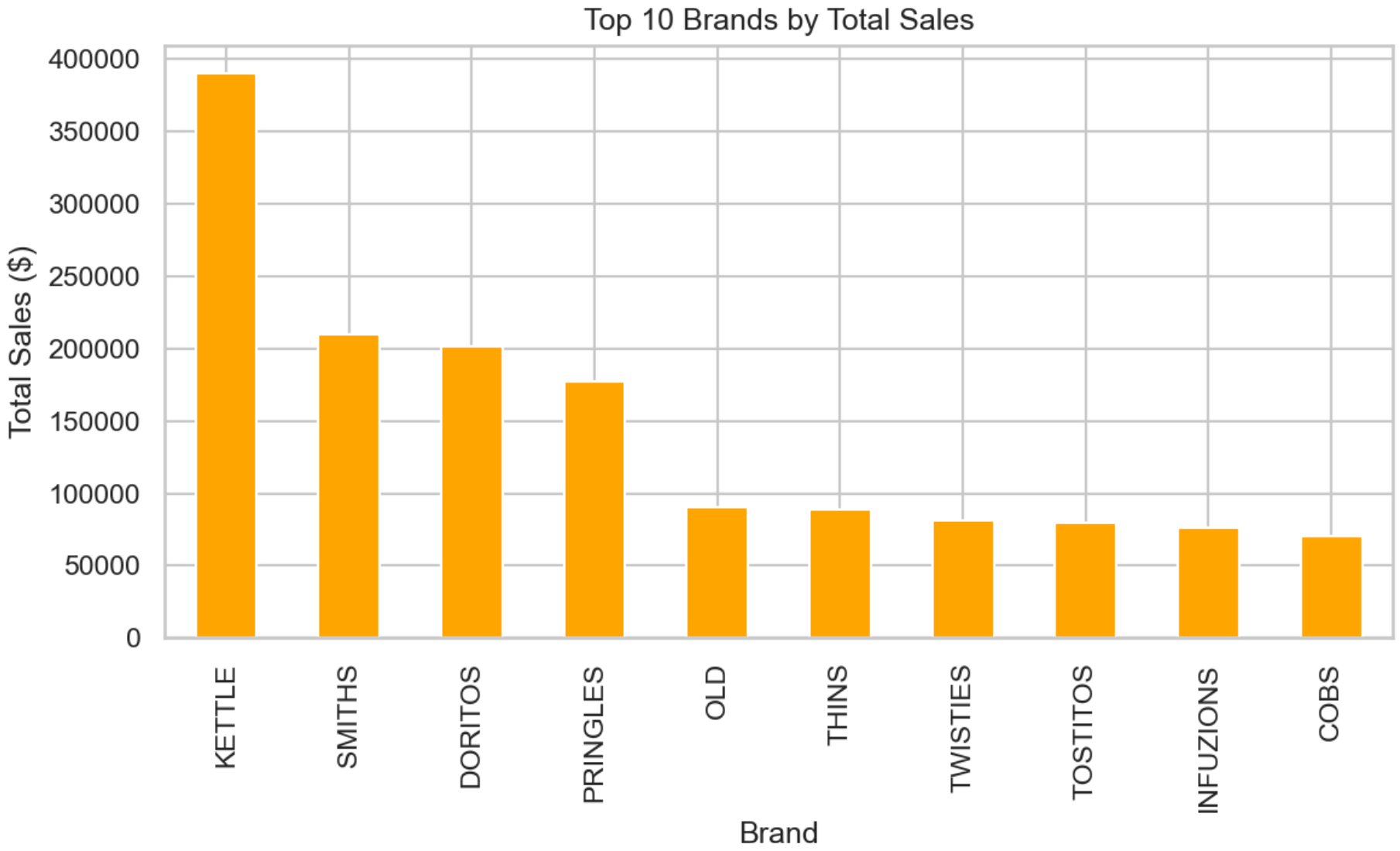
```
plt.ylabel('Total Sales ($)')
plt.xlabel('Lifestage')
plt.tight_layout()
plt.savefig('sales_by_lifestage.png')
plt.show()

# --- Average Spend per Segment ---
plt.figure(figsize=(8,5))
merged.groupby(['LIFESTAGE','PREMIUM_CUSTOMER'])['TOT_SALES'].mean().unstack().plot(kind='bar')
plt.title('Average Spend per Segment')
plt.ylabel('Average Spend ($)')
plt.tight_layout()
plt.savefig('avg_spend_by_segment.png')
plt.show()

# --- Sales by Brand ---
plt.figure(figsize=(8,5))
merged.groupby('BRAND')['TOT_SALES'].sum().sort_values(ascending=False).head(10).plot(kind='bar', color='orange')
plt.title('Top 10 Brands by Total Sales')
plt.ylabel('Total Sales ($)')
plt.xlabel('Brand')
plt.tight_layout()
plt.savefig('top_brands_sales.png')
plt.show()

# ✅ Display the saved images inline to embed them in the final export
display(Image(filename='sales_by_lifestage.png'))
display(Image(filename='avg_spend_by_segment.png'))
display(Image(filename='top_brands_sales.png'))

# ✅ Summary of insights (for report conclusion)
top_lifestage = merged.groupby('LIFESTAGE')['TOT_SALES'].sum().idxmax()
top_segment = merged.groupby(['LIFESTAGE','PREMIUM_CUSTOMER'])['TOT_SALES'].sum().idxmax()
top_brand = merged.groupby('BRAND')['TOT_SALES'].sum().idxmax()

print("📊 SUMMARY INSIGHTS")
print(f"- Highest spending Lifestage: {top_lifestage}")
print(f"- Top segment overall: {top_segment}")
print(f"- Best-selling brand: {top_brand}")
```

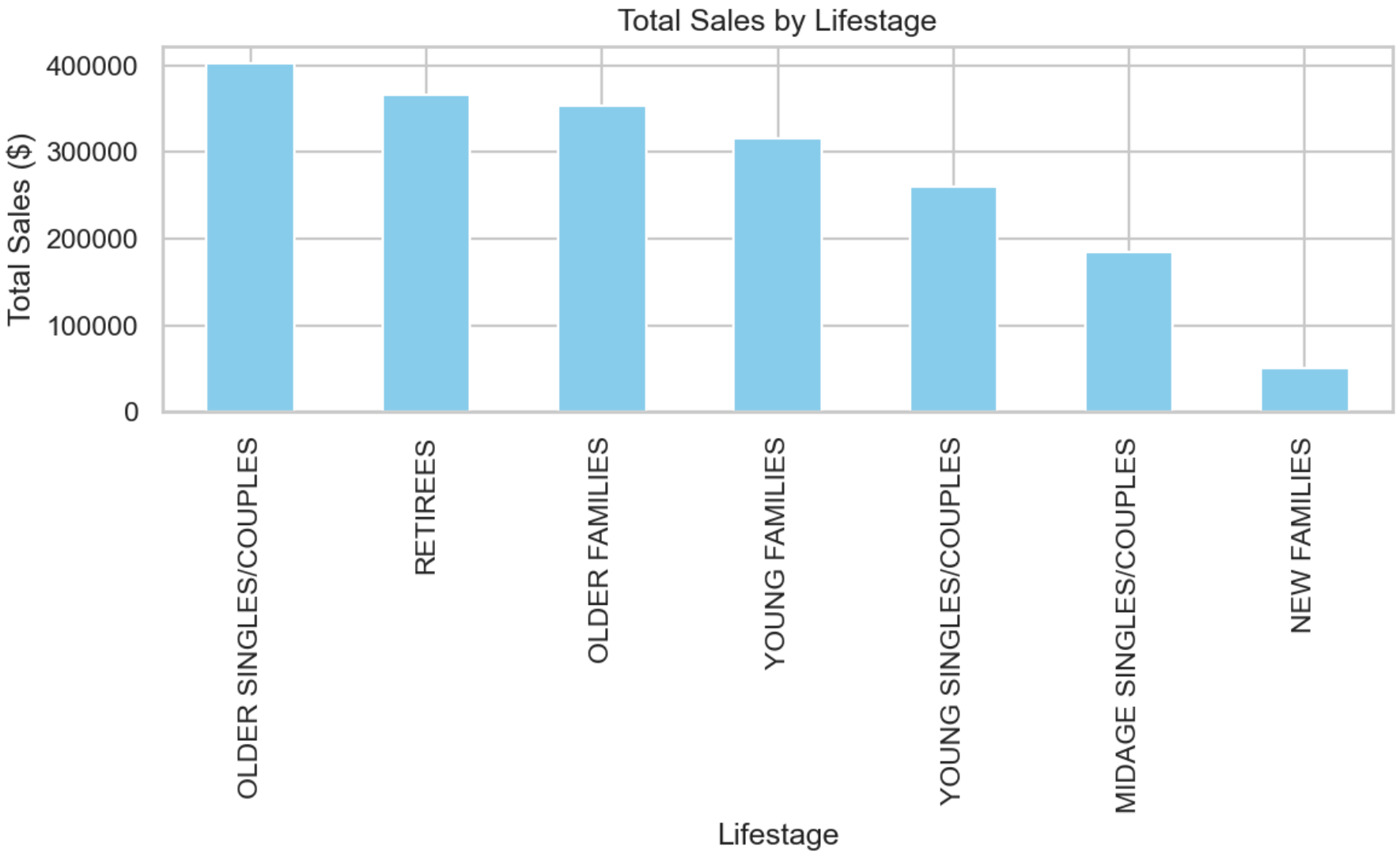
```
print("\n💡 Recommendation: Focus marketing campaigns on the top lifestage and premium segment, "
      "offering combo discounts on the best-selling brand to maximize sales.")
```

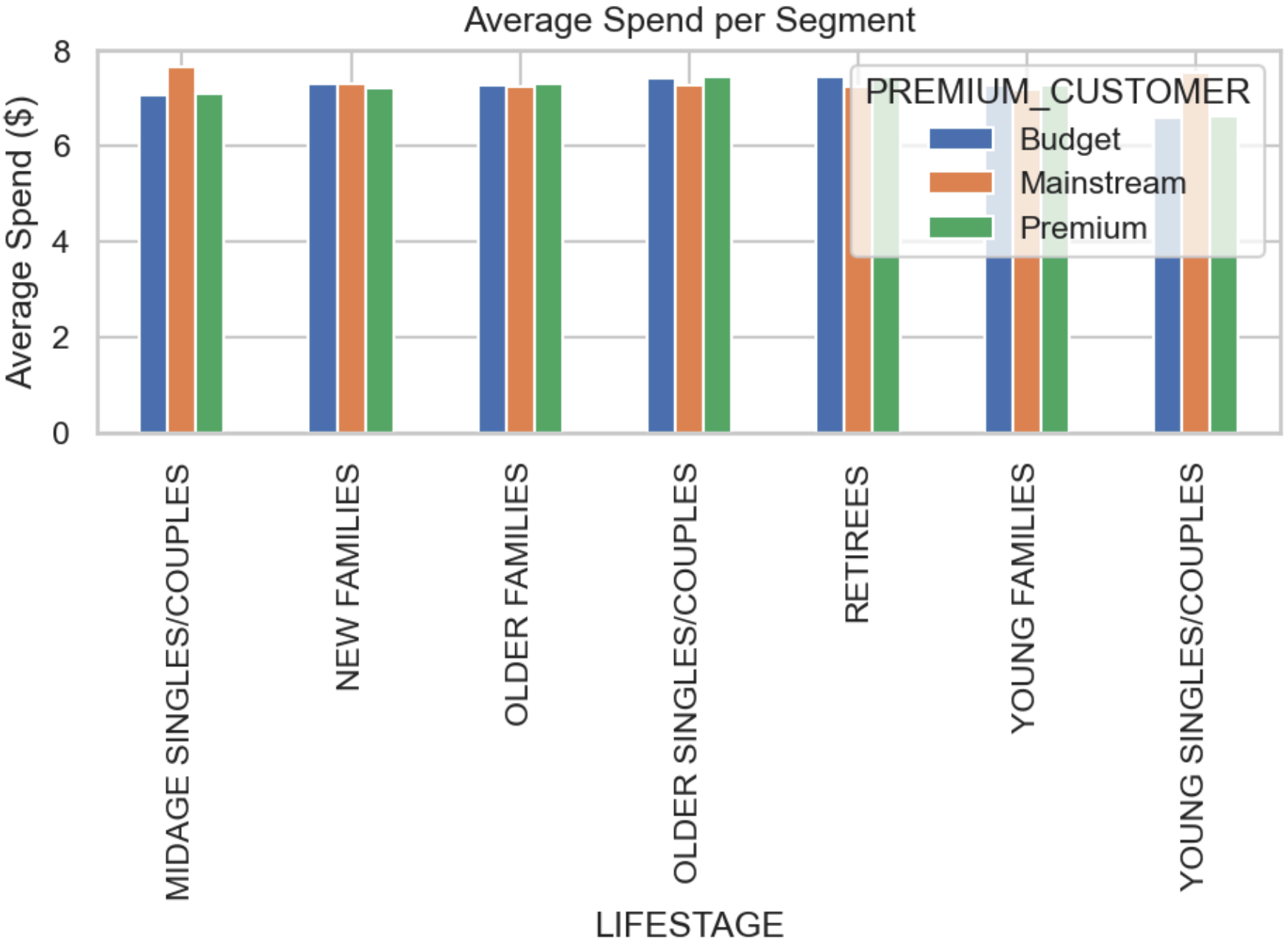


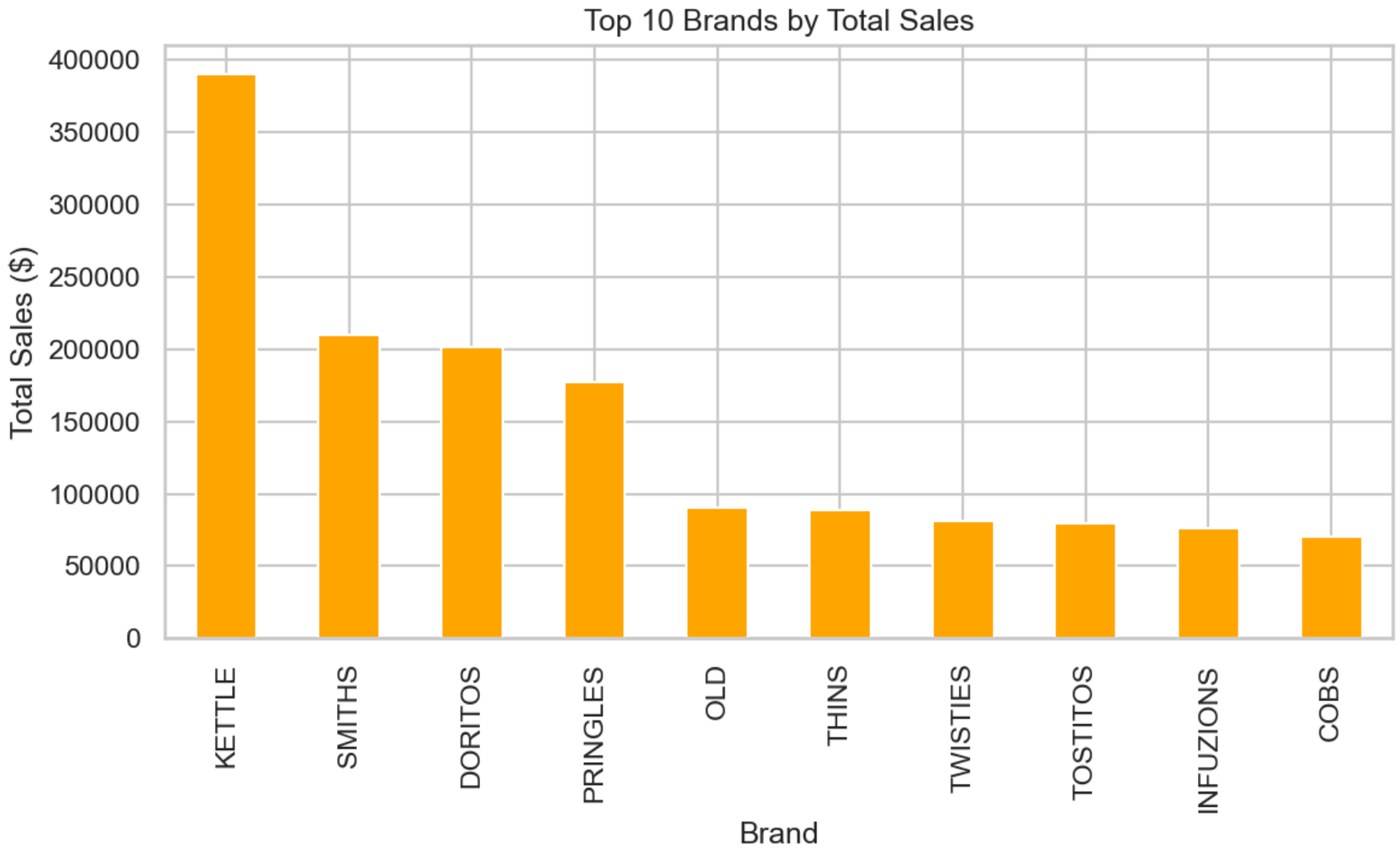

```
<Figure size 1200x750 with 0 Axes>
```


Average Spend per Segment
Average Spend ($)
0
2
4
6
8
PREMIUM_CUSTOMER
Budget
Mainstream
Premium
MIDAGE SINGLES/COUPLES
NEW FAMILIES
OLDER FAMILIES
OLDER SINGLES/COUPLES
RETIREES
YOUNG FAMILIES
YOUNG SINGLES/COUPLES
LIFESTAGE


Top 10 Brands by Total Sales
Total Sales ($)
400000
350000
300000
250000
200000
150000
100000
50000
0
KETTLE
SMITHS
DORITOS
PRINGLES
OLD
THINS
TWISTIES
TOSTITOS
INFUZIONS
COBS
Brand


Total Sales by Lifestage
Total Sales ($)
400000
300000
200000
100000
0
OLDER SINGLES/COUPLES
RETIREES
OLDER FAMILIES
YOUNG FAMILIES
YOUNG SINGLES/COUPLES
MIDAGE SINGLES/COUPLES
NEW FAMILIES
Lifestage


Average Spend per Segment
Average Spend ($)
0
2
4
6
8
PREMIUM_CUSTOMER
Budget
Mainstream
Premium
MIDAGE SINGLES/COUPLES
NEW FAMILIES
OLDER FAMILIES
OLDER SINGLES/COUPLES
RETIREES
YOUNG FAMILIES
YOUNG SINGLES/COUPLES
LIFESTAGE



📊 SUMMARY INSIGHTS
- Highest spending Lifestage: OLDER SINGLES/COUPLES
- Top segment overall: ('OLDER FAMILIES', 'Budget')
- Best-selling brand: KETTLE

💡 Recommendation: Focus marketing campaigns on the top lifestage and premium segment, offering combo discounts on the best-sel
ling brand to maximize sales.

In [ ]: