In [ ]:
```
# In this report, I analyze Quantium's retail transaction and customer data for the Chips category.
# My goal is to uncover customer purchasing behaviors and provide actionable recommendations to Julia,
#the Category Manager, for the upcoming category review.
```

In [12]:
```
# Import Libraries and Load Data
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from statsmodels.stats.weightstats import ttest_ind


# Load data
trans = pd.read_excel(r"C:\Users\nweke\Downloads\QVI_transaction_data.xlsx")
cust = pd.read_csv(r"C:\Users\nweke\Downloads\QVI_purchase_behaviour.csv")
```

In [14]:
```
# Show Data Structure and First Rows
print(trans.info())
print(trans.head(10))
```

```
<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
    DATE  STORE_NBR  LYLTY_CARD_NBR  TXN_ID  PROD_NBR  \
0  43390          1            1000       1         5
1  43599          1            1307     348        66
2  43605          1            1343     383        61
3  43329          2            2373     974        69
4  43330          2            2426    1038       108
5  43604          4            4074    2982        57
6  43601          4            4149    3333        16
7  43601          4            4196    3539        24
8  43332          5            5026    4525        42
9  43330          7            7150    6900        52

                                  PROD_NAME  PROD_QTY  TOT_SALES
0    Natural Chip        Compny SeaSalt175g         2        6.0
1                  CCs Nacho Cheese    175g         3        6.3
2    Smiths Crinkle Cut  Chips Chicken 170g         2        2.9
3    Smiths Chip Thinly  S/Cream&Onion 175g         5       15.0
4  Kettle Tortilla ChpsHny&Jlpno Chili 150g         3       13.8
5  Old El Paso Salsa   Dip Tomato Mild 300g         1        5.1
6  Smiths Crinkle Chips Salt & Vinegar 330g         1        5.7
7     Grain Waves         Sweet Chilli 210g         1        3.6
8  Doritos Corn Chip Mexican Jalapeno 150g          1        3.9
9     Grain Waves Sour    Cream&Chives 210G         2        7.2
```

In [ ]:
```
#After displaying the structure and the first 10 rows of the transaction dataset, I can see that the data consi
#The columns are correctly typed: most are integers, "PROD_NAME" is a string, and "TOT_SALES" is a float. There
#The first few rows also show a variety of chip product names, stores, and transaction IDs.
#However, I notice that the "DATE" column is in integer format and will need to be converted to a standard date
#Additionally, some product names (e.g., "Old El Paso Salsa Dip Tomato Mild 300g") suggest that not all records
```

In [15]:
```
# Show Unique Products and Their Counts
product_counts = trans['PROD_NAME'].value_counts()
print(product_counts.head(10))
print(f"Total unique products: {trans['PROD_NAME'].nunique()}")
```

```
PROD_NAME
Kettle Mozzarella   Basil & Pesto 175g      3304
Kettle Tortilla ChpsHny&Jlpno Chili 150g    3296
Cobs Popd Swt/Chlli &Sr/Cream Chips 110g    3269
Tyrrells Crisps     Ched & Chives 165g      3268
Cobs Popd Sea Salt  Chips 110g              3265
Kettle 135g Swt Pot Sea Salt                3257
Tostitos Splash Of  Lime 175g               3252
Infuzions Thai SweetChili PotatoMix 110g    3242
Smiths Crnkle Chip  Orgnl Big Bag 380g      3233
Thins Potato Chips  Hot & Spicy 175g        3229
Name: count, dtype: int64
Total unique products: 114
```

In [ ]:
```
#When I analyzed the product names in the transaction data, I found there are 114 unique chip products.
#The most frequently purchased items include “Kettle Mozzarella Basil & Pesto 175g”, “Kettle Tortilla ChpsHny&J
#These products each have over 3,000 transactions, showing their popularity.
#The diversity in product names and pack sizes suggests a wide range of chip options are available to customers
```

In [16]:
```
#Summary Statistics Before Cleaning
print(trans.describe(include='all'))
```

```
                 DATE     STORE_NBR  LYLTY_CARD_NBR        TXN_ID  \
count   264836.000000  264836.00000    2.648360e+05  2.648360e+05
unique            NaN           NaN             NaN           NaN
top               NaN           NaN             NaN           NaN
freq              NaN           NaN             NaN           NaN
mean     43464.036260     135.08011    1.355495e+05  1.351583e+05
std        105.389282      76.78418    8.057998e+04  7.813303e+04
min      43282.000000       1.00000    1.000000e+03  1.000000e+00
25%      43373.000000      70.00000    7.002100e+04  6.760150e+04
50%      43464.000000     130.00000    1.303575e+05  1.351375e+05
75%      43555.000000     203.00000    2.030942e+05  2.027012e+05
max      43646.000000     272.00000    2.373711e+06  2.415841e+06

             PROD_NBR                               PROD_NAME       PROD_QTY  \
count   264836.000000                                  264836  264836.000000
unique            NaN                                     114            NaN
top               NaN  Kettle Mozzarella   Basil & Pesto 175g            NaN
freq              NaN                                    3304            NaN
mean        56.583157                                     NaN       1.907309
std         32.826638                                     NaN       0.643654
min          1.000000                                     NaN       1.000000
25%         28.000000                                     NaN       2.000000
50%         56.000000                                     NaN       2.000000
75%         85.000000                                     NaN       2.000000
max        114.000000                                     NaN     200.000000

            TOT_SALES
count   264836.000000
unique            NaN
top               NaN
freq              NaN
mean         7.304200
std          3.083226
min          1.500000
25%          5.400000
50%          7.400000
75%          9.200000
max        650.000000
```

In [ ]:
```
# The summary statistics confirm that my dataset contains 264,836 transactions.
#The “DATE” column is in integer format, with values ranging from 43,282 to 43,646 (corresponding to actual dat
#Store numbers range from 1 to 272, and there are over 2 million unique customer and transaction IDs, showing a
#For products, there are 114 unique chip products, with “Kettle Mozzarella Basil & Pesto 175g” being the most fi
#The “PROD_QTY” column shows most purchases are for 1 or 2 packets per transaction (median and 75th percentile
#The “TOT_SALES” values mostly range between $1.50 and $29.50, with an average sale of $7.30.
#No missing values are present in any columns, so the data is complete for all fields.
```

In [17]:
```
# Check for Missing Values
print(trans.isnull().sum())
```

```
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
```

In [ ]:
```
# After checking for missing values in each column, I confirm that there are no missing entries in the transact
#Every transaction record has complete information for date, store number, customer number, transaction ID, proc
#This means no further data imputation or cleaning is needed for missing values.
```

In [18]:
```
#Convert Date Column
trans['DATE'] = pd.to_datetime('1899-12-30') + pd.to_timedelta(trans['DATE'], unit='D')
```

In [19]:
```
# Remove Non-Chip Products
trans = trans[~trans['PROD_NAME'].str.lower().str.contains('salsa')]
```

In [20]:
```
# Check for Outliers in Quantity
print(trans['PROD_QTY'].describe())
print(trans[trans['PROD_QTY'] > 50])
```

```
count    246742.000000
mean          1.908062
std           0.659831
min           1.000000
25%           2.000000
50%           2.000000
75%           2.000000
max         200.000000
Name: PROD_QTY, dtype: float64
            DATE  STORE_NBR  LYLTY_CARD_NBR  TXN_ID  PROD_NBR  \
69762 2018-08-19        226          226000  226201         4
69763 2019-05-20        226          226000  226210         4

                              PROD_NAME  PROD_QTY  TOT_SALES
69762  Dorito Corn Chp     Supreme 380g       200      650.0
69763  Dorito Corn Chp     Supreme 380g       200      650.0
```

In [ ]:
```
# The quantity summary shows that most transactions involve 1 or 2 chip packets, with a maximum of 200 packets
#Investigating these outlier transactions reveals that the same customer (LYLTY_CARD_NBR 226000) bought 200 pacl
# Such bulk purchases are not typical for regular retail customers and may skew the analysis.
#I will remove this customer’s transactions to better reflect standard buying behavior.
```

In [21]:
```
# Investigate Outlier Transactions
# Find customer(s) with 200 packet transactions
outliers = trans[trans['PROD_QTY'] >= 200]
print(outliers)
# Check all transactions for that customer
print(trans[trans['LYLTY_CARD_NBR'] == outliers['LYLTY_CARD_NBR'].iloc[0]])
```

```
            DATE  STORE_NBR  LYLTY_CARD_NBR  TXN_ID  PROD_NBR  \
69762 2018-08-19        226          226000  226201         4
69763 2019-05-20        226          226000  226210         4

                              PROD_NAME  PROD_QTY  TOT_SALES
69762  Dorito Corn Chp     Supreme 380g       200      650.0
69763  Dorito Corn Chp     Supreme 380g       200      650.0
            DATE  STORE_NBR  LYLTY_CARD_NBR  TXN_ID  PROD_NBR  \
69762 2018-08-19        226          226000  226201         4
69763 2019-05-20        226          226000  226210         4

                              PROD_NAME  PROD_QTY  TOT_SALES
69762  Dorito Corn Chp     Supreme 380g       200      650.0
69763  Dorito Corn Chp     Supreme 380g       200      650.0
```

In [ ]:
```
# Both transactions where 200 packets were purchased were made by the same customer (LYLTY_CARD_NBR 226000), ea
#This customer had only these two transactions in the entire dataset. This behavior is highly unusual and not ty
#or special event purposes. To ensure the analysis reflects normal retail customer behavior, I will remove all
```

In [22]:
```
# Remove Outlier and Show Summary Again
trans = trans[trans['LYLTY_CARD_NBR'] != 226000]
print(trans.describe(include='all'))
```

```
                                DATE      STORE_NBR  LYLTY_CARD_NBR  \
count                         246740  246740.000000    2.467400e+05
unique                           NaN            NaN             NaN
top                              NaN            NaN             NaN
freq                             NaN            NaN             NaN
mean    2018-12-30 01:18:58.448569344     135.050361    1.355303e+05
min              2018-07-01 00:00:00       1.000000    1.000000e+03
25%              2018-09-30 00:00:00      70.000000    7.001500e+04
50%              2018-12-30 00:00:00     130.000000    1.303670e+05
75%              2019-03-31 00:00:00     203.000000    2.030832e+05
max              2019-06-30 00:00:00     272.000000    2.373711e+06
std                              NaN      76.786971    8.071520e+04

              TXN_ID       PROD_NBR                             PROD_NAME  \
count   2.467400e+05  246740.000000                                246740
unique           NaN            NaN                                   105
top              NaN            NaN  Kettle Mozzarella   Basil & Pesto 175g
freq             NaN            NaN                                  3304
mean    1.351304e+05      56.352213                                   NaN
min     1.000000e+00       1.000000                                   NaN
25%     6.756875e+04      26.000000                                   NaN
50%     1.351815e+05      53.000000                                   NaN
75%     2.026522e+05      87.000000                                   NaN
max     2.415841e+06     114.000000                                   NaN
std     7.814760e+04      33.695235                                   NaN

             PROD_QTY      TOT_SALES
count   246740.000000  246740.000000
unique            NaN            NaN
top               NaN            NaN
freq              NaN            NaN
mean         1.906456       7.316113
min          1.000000       1.700000
25%          2.000000       5.800000
50%          2.000000       7.400000
75%          2.000000       8.800000
max          5.000000      29.500000
std          0.342499       2.474897
```

In [ ]:

```
#After removing the outlier customer who made bulk purchases, the transaction dataset now contains 246,740 recor
#The maximum “PROD_QTY” dropped from 200 to 5 packets per transaction, which is more realistic for everyday reta
#The mean quantity is now 1.91 packets, and the mean total sales per transaction is $7.32. The data is now much
#ensuring that future analysis and insights are not distorted by exceptional or commercial orders.
#The rest of the columns, such as dates and product IDs, remain consistent, and there are still no missing valu
```

In [23]:

```
# Transactions Per Day—Missing Dates?
tx_per_day = trans.groupby('DATE').size().reset_index(name='N')
print(tx_per_day.head())
print(f"Number of unique dates: {tx_per_day['DATE'].nunique()}")
```

```
        DATE    N
0 2018-07-01  663
1 2018-07-02  650
2 2018-07-03  674
3 2018-07-04  669
4 2018-07-05  660
Number of unique dates: 364
```

In [ ]:

```
# By grouping transactions by date, I found that the number of transactions per day generally ranges from about
#which is one less than the 365 days expected in a full year. This suggests that at least one date is missing—pe
#I will create a complete date sequence to confirm which day is missing and to ensure accurate time series analy
```

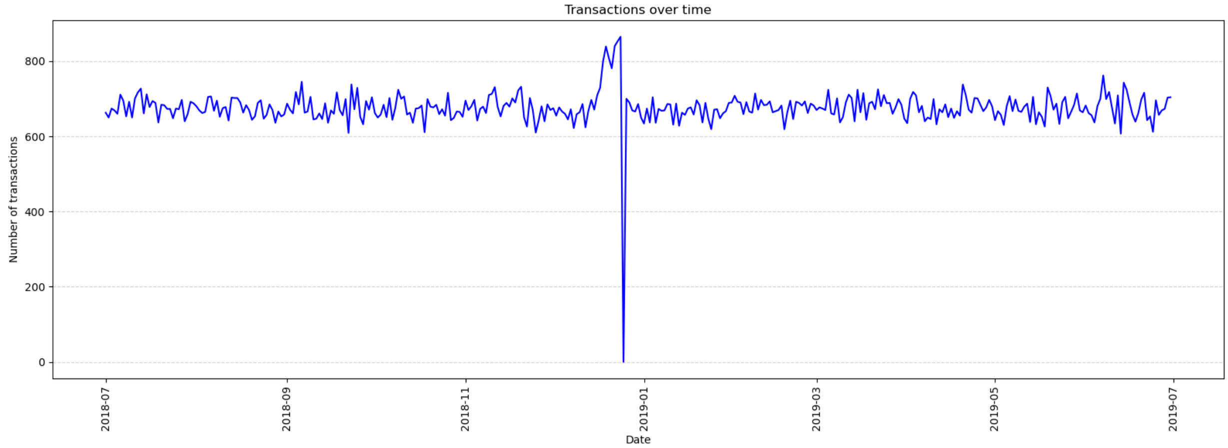
In [24]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Create a complete date sequence for one year
full_dates = pd.DataFrame({'DATE': pd.date_range(start='2018-07-01', end='2019-06-30')})

# Merge with transaction counts (fill missing days with 0)
tx_per_day_full = full_dates.merge(tx_per_day, how='left', on='DATE').fillna(0)
tx_per_day_full['N'] = tx_per_day_full['N'].astype(int)
```

In [39]:

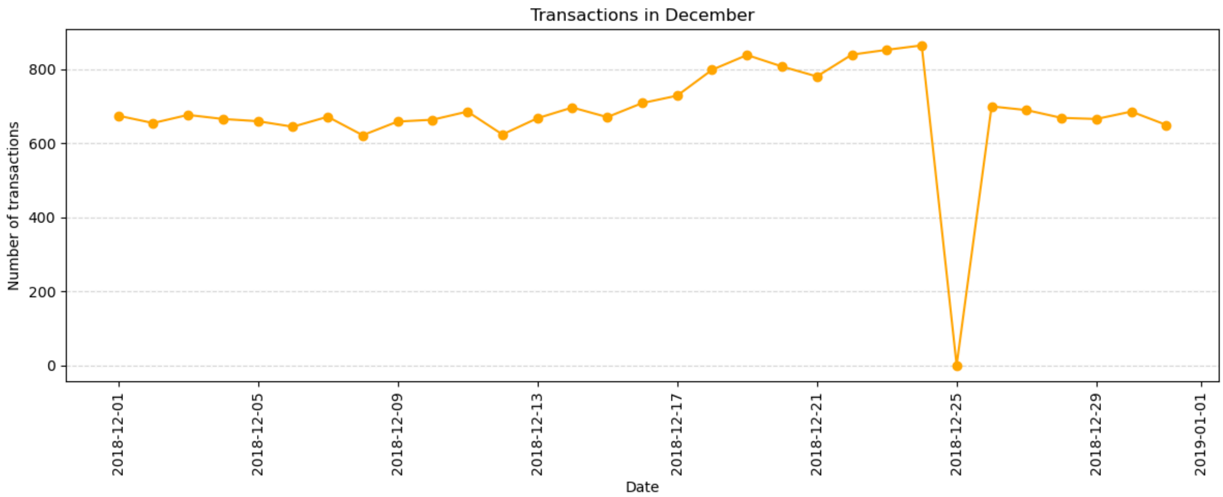
```
# Plot Transactions Over Time
plt.figure(figsize=(16, 6))
plt.plot(tx_per_day_full['DATE'], tx_per_day_full['N'], color='blue')
plt.title('Transactions over time')
plt.xlabel('Date')
plt.ylabel('Number of transactions')
plt.xticks(rotation=90)
```

```
plt.grid(True, axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```



In [ ]:
```
# The line plot shows the daily number of chip transactions from July 2018 to july 2019. There is a clear spike
#likely due to holiday season demand. However, there is a noticeable gap (zero transactions) at the end of Decem
# which corresponds to Christmas Day—when stores are likely closed.
```

In [25]:
```
# Zoom In on December for Detailed View
december = tx_per_day_full[(tx_per_day_full['DATE'].dt.month == 12)]

plt.figure(figsize=(12, 5))
plt.plot(december['DATE'], december['N'], marker='o', color='orange')
plt.title('Transactions in December')
plt.xlabel('Date')
plt.ylabel('Number of transactions')
plt.xticks(rotation=90)
plt.grid(True, axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```



In [ ]:
```
# Zooming in on December, I see a steady increase in transactions leading up to Christmas, followed by a comple
#This confirms that the missing date from the dataset corresponds to a public holiday when no sales occurred.
```

In [26]:
```
# Show Most Common Words in Product Names
import re
from collections import Counter

words = []
for prod in trans['PROD_NAME'].unique():
    words += re.findall(r'[A-Za-z&]+', prod)

word_counts = Counter(words)
print(word_counts.most_common(20))
```

```
[('g', 102), ('Chips', 21), ('&', 16), ('Smiths', 15), ('Crinkle', 13), ('Cut', 13), ('Kettle', 13), ('Cheese',
12), ('Salt', 12), ('Original', 10), ('Chip', 9), ('Chicken', 8), ('Corn', 8), ('Pringles', 8), ('RRD', 8), ('Do
ritos', 7), ('WW', 7), ('Sour', 6), ('Cream', 6), ('Sea', 6)]
```

In [27]:
```
# Extract Pack Size and Brand
import re

# Extract PACK_SIZE from PROD_NAME
trans['PACK_SIZE'] = trans['PROD_NAME'].str.extract(r'(\d+)').astype(float)

# Extract BRAND as the first word in PROD_NAME
trans['BRAND'] = trans['PROD_NAME'].str.split().str[0]

# Standardize similar brand names if needed (e.g., 'RRD' and 'RED' as 'RRD')
trans['BRAND'] = trans['BRAND'].replace({'RED':'RRD'})
```

In [28]:
```
# Check Frequency Table for Pack
pack_size_counts = trans['PACK_SIZE'].value_counts().sort_index()
print(pack_size_counts)
```

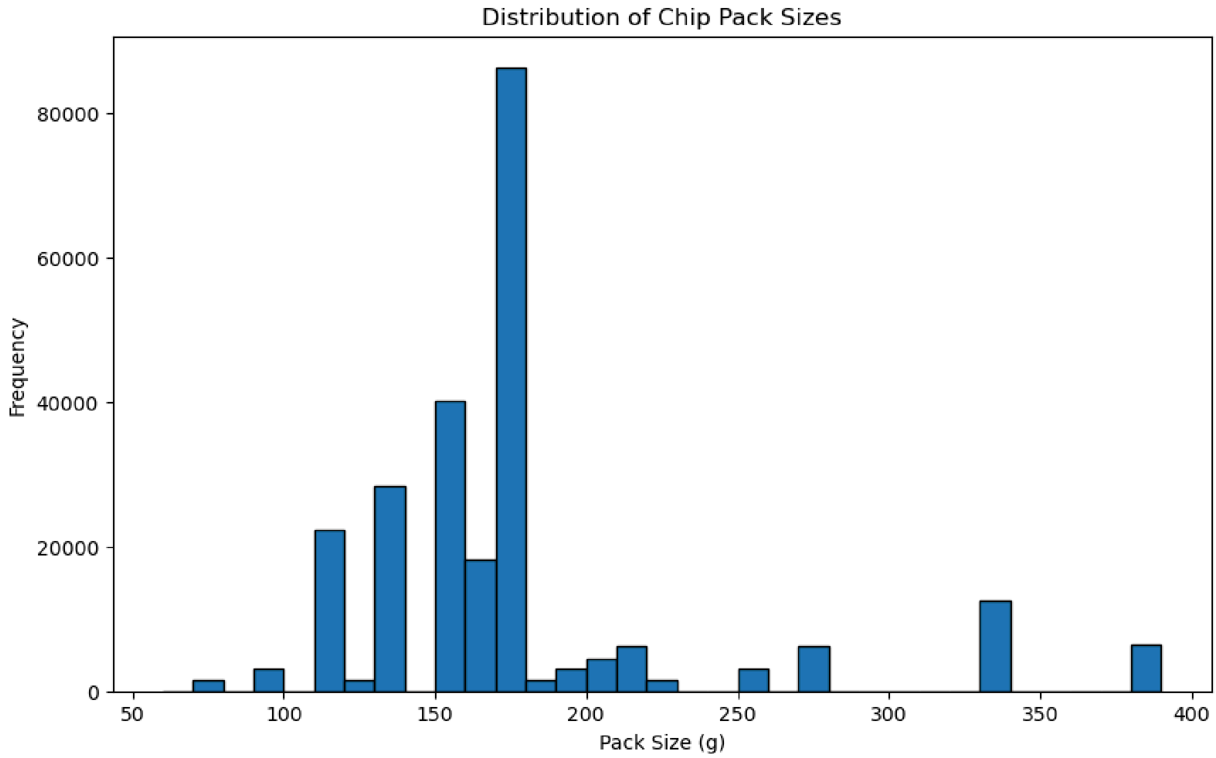
```
PACK_SIZE
70.0      1507
90.0      3008
110.0    22387
125.0     1454
134.0    25102
135.0     3257
150.0    40203
160.0     2970
165.0    15297
170.0    19983
175.0    66390
180.0     1468
190.0     2995
200.0     4473
210.0     6272
220.0     1564
250.0     3169
270.0     6285
330.0    12540
380.0     6416
Name: count, dtype: int64
```

In [29]:
```
# Check Pack Size Column in the Data
print(trans[['PROD_NAME', 'PACK_SIZE']].head(10))
```

```
                                    PROD_NAME  PACK_SIZE
0     Natural Chip        Compny SeaSalt175g      175.0
1                   CCs Nacho Cheese    175g      175.0
2     Smiths Crinkle Cut  Chips Chicken 170g      170.0
3     Smiths Chip Thinly  S/Cream&Onion 175g      175.0
4   Kettle Tortilla ChpsHny&Jlpno Chili 150g      150.0
6   Smiths Crinkle Chips Salt & Vinegar 330g      330.0
7      Grain Waves         Sweet Chilli 210g      210.0
8    Doritos Corn Chip Mexican Jalapeno 150g      150.0
9      Grain Waves Sour    Cream&Chives 210G      210.0
10  Smiths Crinkle Chips Salt & Vinegar 330g      330.0
```

In [ ]:
```
# Reviewing the first few rows of the “PROD_NAME” and “PACK_SIZE” columns confirms that the extraction of pack
#For example, “Natural Chip Compny SeaSalt175g” is correctly assigned 175g,
#“Smiths Crinkle Cut Chips Chicken 170g” is assigned 170g, and “Smiths Crinkle Chips Salt & Vinegar 330g” is as
#This check validates that the feature engineering for pack size is working as intended and ready for further a
```

In [30]:
```
# Plot Histogram of Pack Sizes
import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))
plt.hist(trans['PACK_SIZE'], bins=range(60, 400, 10), edgecolor='black')
plt.title('Distribution of Chip Pack Sizes')
plt.xlabel('Pack Size (g)')
plt.ylabel('Frequency')
plt.show()
```



In [ ]:
```
#The histogram of “PACK_SIZE” confirms that most chip transactions are concentrated in a few standard packet si
#These are typical retail pack sizes, and there are no unexpected outlier values.
#The distribution matches expectations for supermarket chip sales and supports the accuracy of the feature extr
```

In [31]:
```
# Create the BRAND Feature
# Brand is first word in PROD_NAME, uppercase
trans['BRAND'] = trans['PROD_NAME'].str.split().str[0].str.upper()
```

In [32]:
```
# Check Frequency Table for Brands
brand_counts = trans['BRAND'].value_counts().sort_index()
print(brand_counts)
```

```
BRAND
BURGER         1564
CCS            4551
CHEETOS        2927
CHEEZELS       4603
COBS           9693
DORITO         3183
DORITOS       22041
FRENCH         1418
GRAIN          6272
GRNWVES        1468
INFUZIONS     11057
INFZNS         3144
KETTLE        41288
NATURAL        6050
NCC            1419
PRINGLES      25102
RED            4427
RRD           11894
SMITH          2963
SMITHS        27390
SNBTS          1576
SUNBITES       1432
THINS         14075
TOSTITOS       9471
TWISTIES       9454
TYRRELLS       6442
WOOLWORTHS     1516
WW            10320
Name: count, dtype: int64
```

In [ ]:
```
# The frequency table for "BRAND" shows a variety of chip brands sold, with the most popular being KETTLE (41,2
#There are also several brands with alternative or misspelled versions (e.g., "DORITO" and "DORITOS", "INFZNS" 
#as well as abbreviations (e.g., "WW" for "WOOLWORTHS", "NCC" for "NATURAL", "SNBTS" for "SUNBITES").

#To ensure accurate analysis, I will standardize and combine similar brand names in the next step.
```

In [33]:
```
# Clean Up Brand Names (Standardize, Combine Duplicates)
trans['BRAND'] = trans['BRAND'].replace({
    "RED": "RRD",
    "SNBTS": "SUNBITES",
    "INFZNS": "INFUZIONS",
    "WW": "WOOLWORTHS",
    "SMITH": "SMITHS",
    "NCC": "NATURAL",
    "DORITO": "DORITOS",
    "GRAIN": "GRNWVES"
})

brand_counts_cleaned = trans['BRAND'].value_counts().sort_index()
print(brand_counts_cleaned)
```

```
BRAND
BURGER         1564
CCS            4551
CHEETOS        2927
CHEEZELS       4603
COBS           9693
DORITOS       25224
FRENCH         1418
GRNWVES        7740
INFUZIONS     14201
KETTLE        41288
NATURAL        7469
PRINGLES      25102
RRD           16321
SMITHS        30353
SUNBITES       3008
THINS         14075
TOSTITOS       9471
TWISTIES       9454
TYRRELLS       6442
WOOLWORTHS    11836
Name: count, dtype: int64
```

In [ ]:
```
# After standardizing and combining duplicate or misspelled brand names, the frequency counts now more accurate
#For example, "DORITO" and "DORITOS" are merged under "DORITOS" (25,224), and abbreviations like "WW" and "WOOL
#This step ensures that future analysis of brand performance is not distorted by inconsistent naming, and the r
```

In [34]:
```
#Check Final Brand and Pack Size Columns
print(trans[['PROD_NAME', 'BRAND', 'PACK_SIZE']].head(10))
```

```
                                   PROD_NAME    BRAND  PACK_SIZE
0     Natural Chip        Compny SeaSalt175g  NATURAL      175.0
1                   CCs Nacho Cheese    175g      CCS      175.0
2     Smiths Crinkle Cut  Chips Chicken 170g   SMITHS      170.0
3     Smiths Chip Thinly  S/Cream&Onion 175g   SMITHS      175.0
4   Kettle Tortilla ChpsHny&Jlpno Chili 150g   KETTLE      150.0
6   Smiths Crinkle Chips Salt & Vinegar 330g   SMITHS      330.0
7      Grain Waves         Sweet Chilli 210g  GRNWVES      210.0
8    Doritos Corn Chip Mexican Jalapeno 150g  DORITOS      150.0
9      Grain Waves Sour    Cream&Chives 210G  GRNWVES      210.0
10  Smiths Crinkle Chips Salt & Vinegar 330g   SMITHS      330.0
```

In [ ]:
```
#A final review of the "PROD_NAME", "BRAND", and "PACK_SIZE" columns confirms that both features have been accu
#For example, "Natural Chip Compny SeaSalt175g" is labeled as brand "NATURAL" with a pack size of 175g, "Smiths
#and 170g, and "Doritos Corn Chip Mexican Jalapeno 150g" is assigned to "DORITOS" and 150g.
#This step validates the feature engineering process, ensuring my dataset is clean and ready for meaningful ana
```

In [35]:
```
# Examine and Clean Customer Data
# Check info and missing values
print(cust.info())
print(cust.isnull().sum())
print(cust.head())
print(f"Number of unique customers: {cust['LYLTY_CARD_NBR'].nunique()}")
print(cust['LIFESTAGE'].value_counts())
print(cust['PREMIUM_CUSTOMER'].value_counts())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72637 entries, 0 to 72636
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   LYLTY_CARD_NBR    72637 non-null  int64 
 1   LIFESTAGE         72637 non-null  object
 2   PREMIUM_CUSTOMER  72637 non-null  object
dtypes: int64(1), object(2)
memory usage: 1.7+ MB
None
LYLTY_CARD_NBR      0
LIFESTAGE           0
PREMIUM_CUSTOMER    0
dtype: int64
   LYLTY_CARD_NBR               LIFESTAGE PREMIUM_CUSTOMER
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
Number of unique customers: 72637
LIFESTAGE
RETIREES                  14805
OLDER SINGLES/COUPLES     14609
YOUNG SINGLES/COUPLES     14441
OLDER FAMILIES             9780
YOUNG FAMILIES             9178
MIDAGE SINGLES/COUPLES     7275
NEW FAMILIES               2549
Name: count, dtype: int64
PREMIUM_CUSTOMER
Mainstream    29245
Budget        24470
Premium       18922
Name: count, dtype: int64
```

In [ ]:

```
#The customer dataset contains 72,637 unique customers, each with a loyalty card number. There are no missing va
#The largest customer segments by life stage are “Retirees” (14,805), “Older Singles/Couples” (14,609), and “You
#In terms of premium status, “Mainstream” customers are the most common (29,245), followed by “Budget” (24,470)
#This data is well-organized and ready for merging with the transaction records for segment-based analysis.
```

In [36]:

```
# Merge transaction and customer data on 'LYLTY_CARD_NBR'
data = trans.merge(cust, how='left', on='LYLTY_CARD_NBR')

# Check for unmatched customers
print(data.isnull().sum())
```

```
DATE                0
STORE_NBR           0
LYLTY_CARD_NBR      0
TXN_ID              0
PROD_NBR            0
PROD_NAME           0
PROD_QTY            0
TOT_SALES           0
PACK_SIZE           0
BRAND               0
LIFESTAGE           0
PREMIUM_CUSTOMER    0
dtype: int64
```

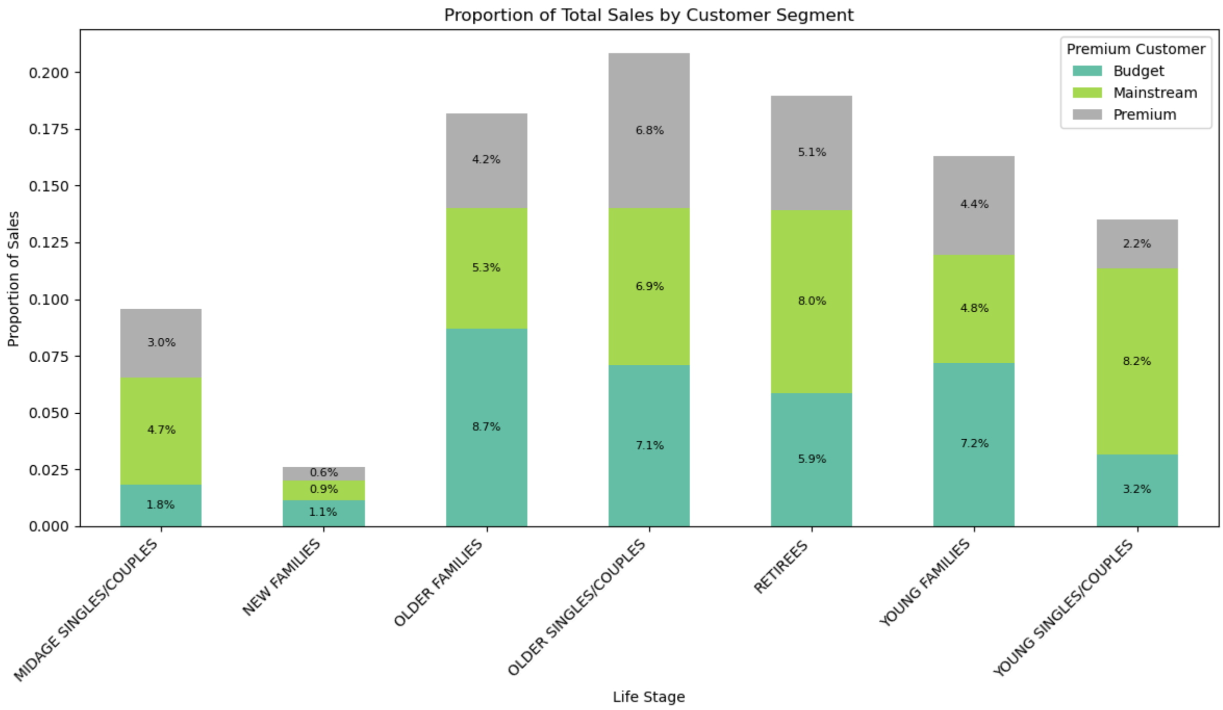
In [ ]:

```
# After merging, every transaction is matched to a customer record. There are no missing or unmatched customers,
```

In [41]:

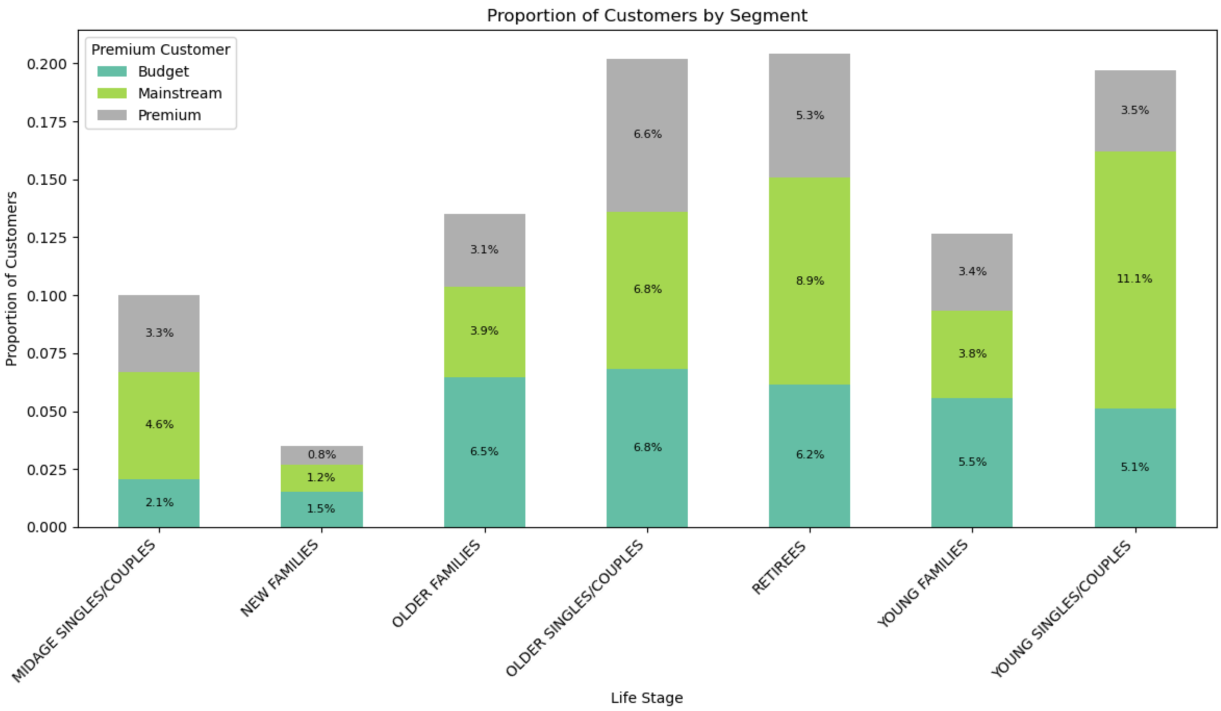
```
# Proportion of Total Sales by Segment (Mosaic-style Plot)
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate total sales by segment
sales_by_segment = data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['TOT_SALES'].sum().reset_index()
total_sales = sales_by_segment['TOT_SALES'].sum()
sales_by_segment['PROPORTION'] = sales_by_segment['TOT_SALES'] / total_sales

# Pivot for plotting
sales_pivot = sales_by_segment.pivot(index='LIFESTAGE', columns='PREMIUM_CUSTOMER', values='PROPORTION').fillna

# Stacked bar plot for proportions
sales_pivot.plot(kind='bar', stacked=True, figsize=(12,7), colormap='Set2')
plt.title('Proportion of Total Sales by Customer Segment')
```

```python
plt.ylabel('Proportion of Sales')
plt.xlabel('Life Stage')
plt.legend(title='Premium Customer')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Add percentage labels to each bar section
for i, lifestage in enumerate(sales_pivot.index):
    cumulative = 0
    for j, seg in enumerate(sales_pivot.columns):
        value = sales_pivot.loc[lifestage, seg]
        if value > 0:
            plt.text(i, cumulative + value / 2, f"{value * 100:.1f}%", ha='center', va='center', fontsize=8)
        cumulative += value

plt.show()
```



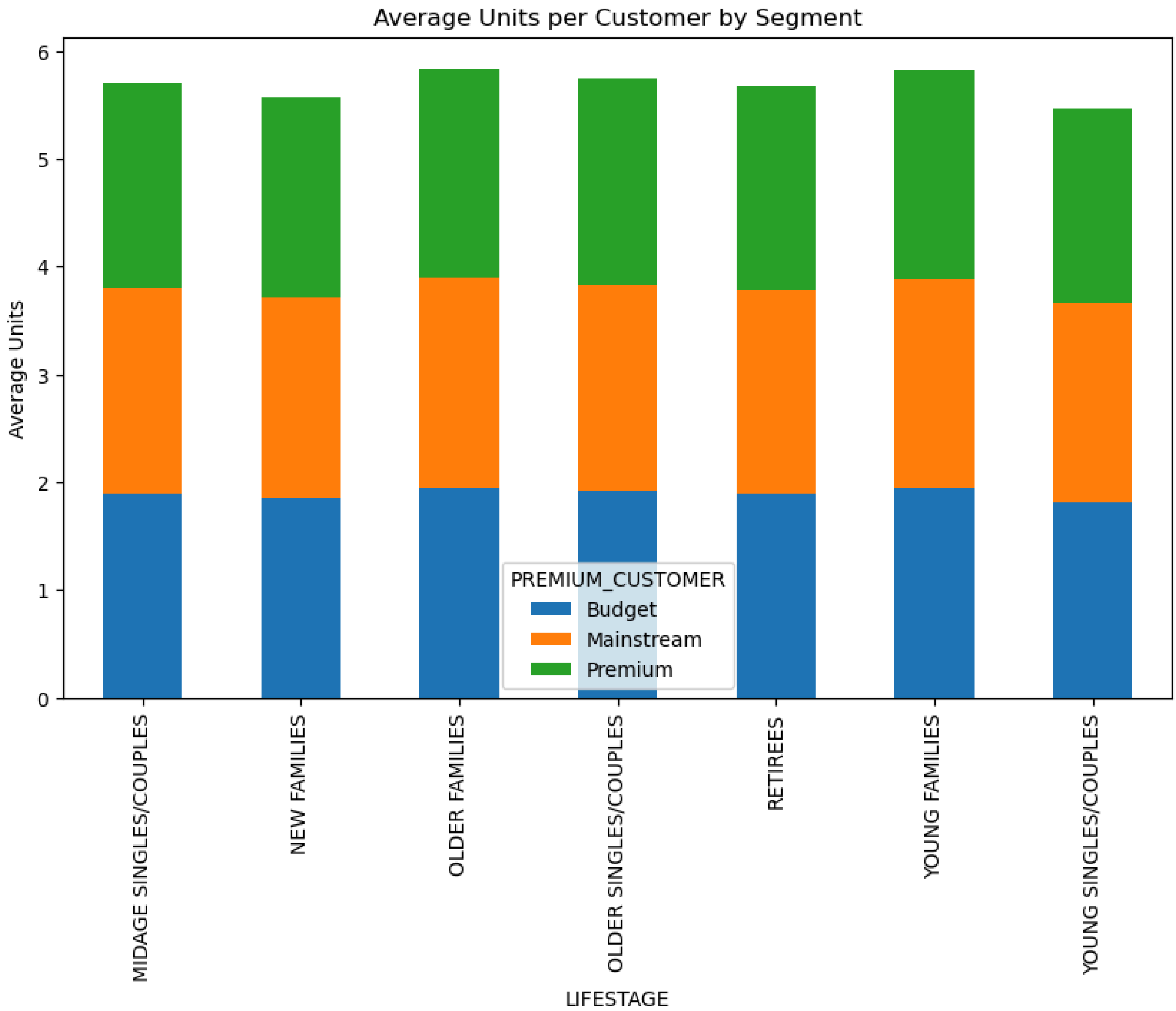
In [ ]:
```python
# The plot shows the proportion of total chip sales contributed by each customer segment, broken down by life s
The main contributors to sales are Budget Older Families, Mainstream Young Singles/Couples, and Mainstream Reti
#This mosaic-style view provides a clear picture of which customer groups are driving category performance.
```

In [42]:
```python
# Proportion of Customers by Segment
# Calculate unique customers by segment
customers_by_segment = data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['LYLTY_CARD_NBR'].nunique().reset_index(
total_customers = customers_by_segment['LYLTY_CARD_NBR'].sum()
customers_by_segment['PROPORTION'] = customers_by_segment['LYLTY_CARD_NBR'] / total_customers

# Pivot for plotting
cust_pivot = customers_by_segment.pivot(index='LIFESTAGE', columns='PREMIUM_CUSTOMER', values='PROPORTION').fil

# Stacked bar plot for proportions
cust_pivot.plot(kind='bar', stacked=True, figsize=(12,7), colormap='Set2')
plt.title('Proportion of Customers by Segment')
plt.ylabel('Proportion of Customers')
plt.xlabel('Life Stage')
plt.legend(title='Premium Customer')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Add percentage labels to each bar section
for i, lifestage in enumerate(cust_pivot.index):
    cumulative = 0
    for j, seg in enumerate(cust_pivot.columns):
        value = cust_pivot.loc[lifestage, seg]
        if value > 0:
            plt.text(i, cumulative + value / 2, f"{value * 100:.1f}%", ha='center', va='center', fontsize=8)
        cumulative += value

plt.show()
```



In [ ]:
```
#This plot shows the proportion of unique chip buyers in each segment. The largest groups of chip buyers are Ma
#While the same segments drive both sales and customer counts, Budget Older Families stand out as heavy purchas
```

In [43]:
```
# Average Number of Units per Customer by Segment
units_per_cust = data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['PROD_QTY'].mean().unstack()
units_per_cust.plot(kind='bar', stacked=True, figsize=(10,6))
plt.title('Average Units per Customer by Segment')
plt.ylabel('Average Units')
plt.show()
```

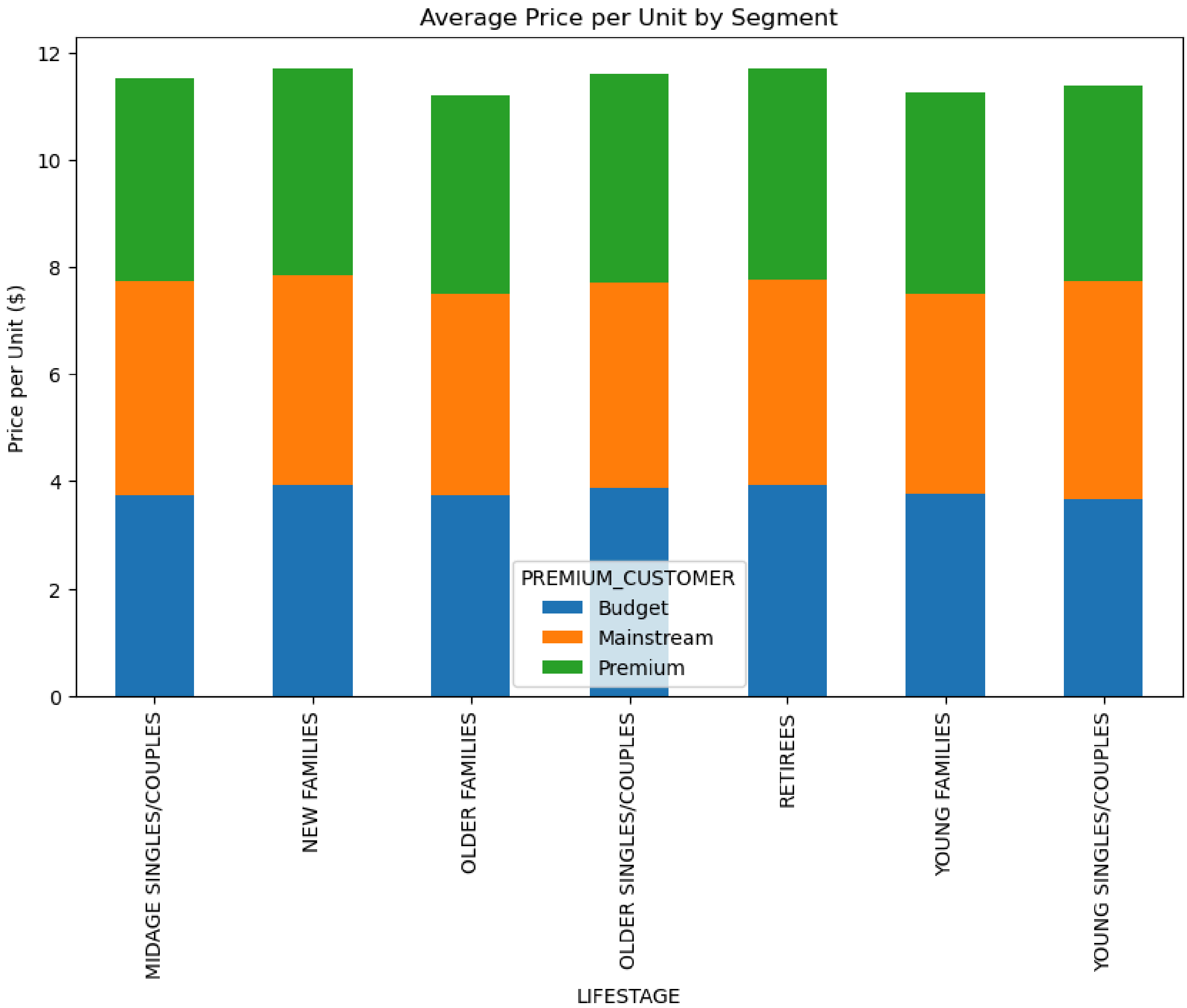


In [ ]:

```
# Families, especially Older and Young Families, buy more chip packets per customer than singles or retirees.
# Mainstream young and midage singles/couples pay slightly more per unit, possibly due to a preference for prem
```

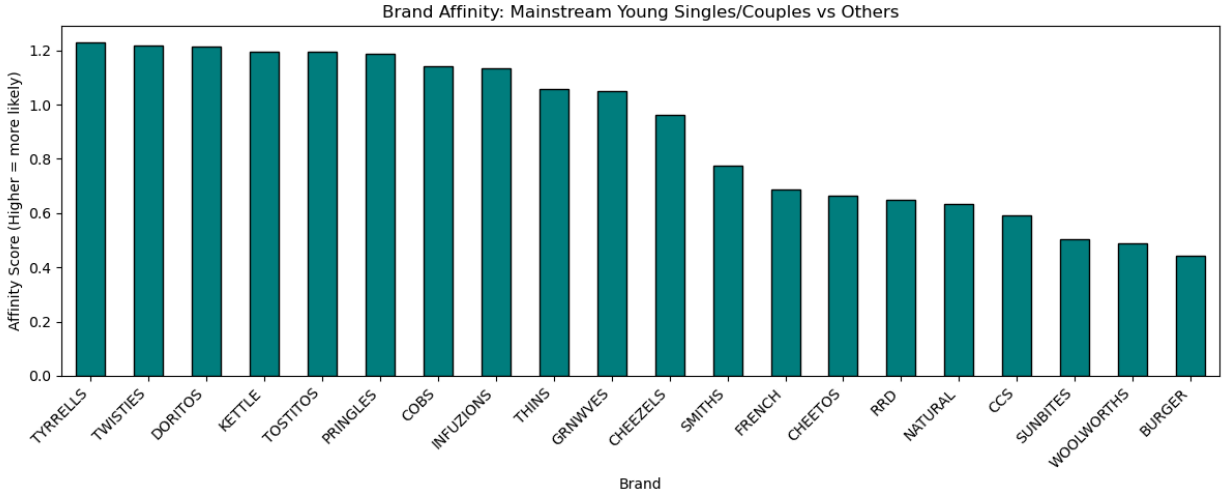
In [45]:

```
# Average Price per Unit by Segment
data['PRICE_PER_UNIT'] = data['TOT_SALES'] / data['PROD_QTY']
price_per_unit = data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['PRICE_PER_UNIT'].mean().unstack()
price_per_unit.plot(kind='bar', stacked=True, figsize=(10,6))
plt.title('Average Price per Unit by Segment')
plt.ylabel('Price per Unit ($)')
plt.show()
```



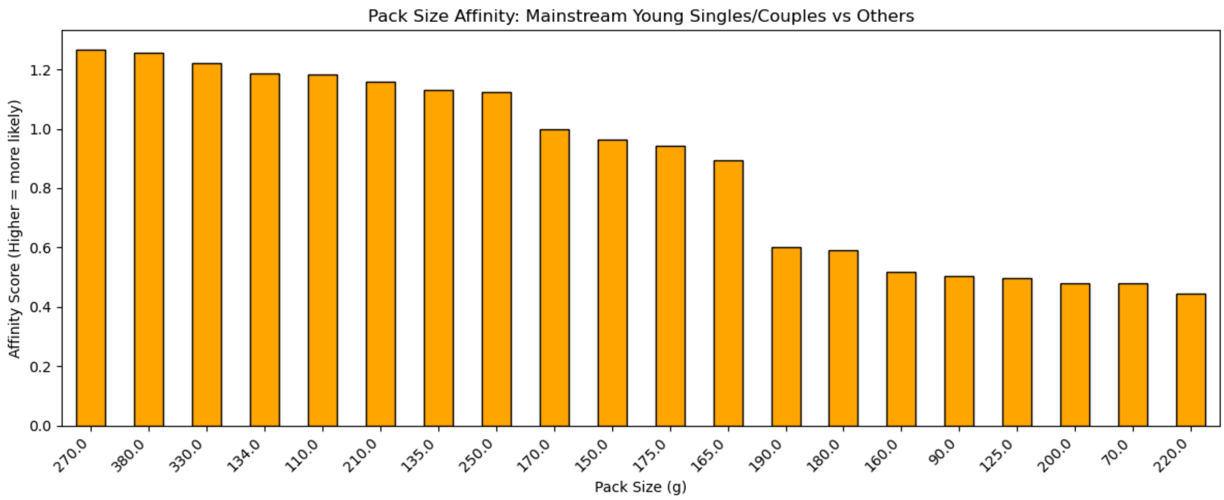
In [46]:

```
# Brand and Pack Size Preference (for a Segment) and Brand Affinity Score
# Define the segment and "other" group
segment = data[(data['LIFESTAGE'] == 'YOUNG SINGLES/COUPLES') & (data['PREMIUM_CUSTOMER'] == 'Mainstream')]
other = data[~((data['LIFESTAGE'] == 'YOUNG SINGLES/COUPLES') & (data['PREMIUM_CUSTOMER'] == 'Mainstream'))]

# Calculate proportions for each brand in the segment and in others
qty_segment = segment['PROD_QTY'].sum()
qty_other = other['PROD_QTY'].sum()

brand_segment = segment.groupby('BRAND')['PROD_QTY'].sum() / qty_segment
brand_other = other.groupby('BRAND')['PROD_QTY'].sum() / qty_other

# Combine and calculate affinity score
brand_affinity = (brand_segment / brand_other).sort_values(ascending=False)

# Plot
plt.figure(figsize=(12,5))
brand_affinity.plot(kind='bar', color='teal', edgecolor='black')
plt.title('Brand Affinity: Mainstream Young Singles/Couples vs Others')
plt.ylabel('Affinity Score (Higher = more likely)')
plt.xlabel('Brand')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



In [ ]:
```
# The affinity score compares how much more or less likely Mainstream Young Singles/Couples are to purchase eac
#For this segment, brands like Tyrrells, Twisties, Kettle, and Pringles have an affinity above 1, showing they
#meaning this segment is much less likely to purchase them.
```

In [47]:
```
# Pack Size Affinity Score
# Calculate proportions for each pack size in the segment and in others
pack_segment = segment.groupby('PACK_SIZE')['PROD_QTY'].sum() / qty_segment
pack_other = other.groupby('PACK_SIZE')['PROD_QTY'].sum() / qty_other

# Combine and calculate affinity score
pack_affinity = (pack_segment / pack_other).sort_values(ascending=False)

# Plot
plt.figure(figsize=(12,5))
pack_affinity.plot(kind='bar', color='orange', edgecolor='black')
plt.title('Pack Size Affinity: Mainstream Young Singles/Couples vs Others')
plt.ylabel('Affinity Score (Higher = more likely)')
plt.xlabel('Pack Size (g)')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



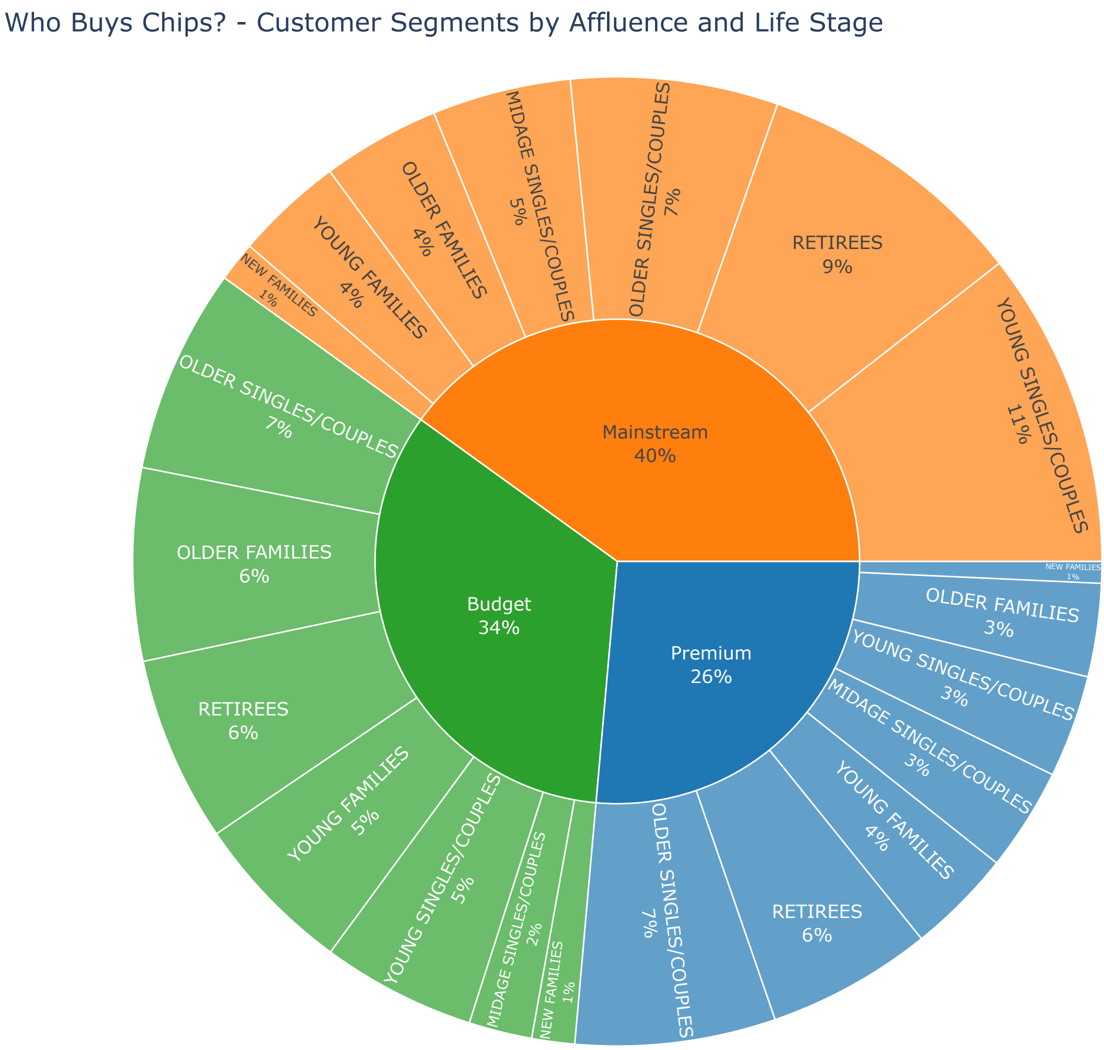
In [ ]:
```
#The pack size affinity scores show that Mainstream Young Singles/Couples are much more likely to purchase 270g
#and less likely to buy smaller packs like 70g, 90g, and 110g. This indicates a preference for larger pack size
#suggesting an opportunity for targeted promotions on large packs and brands like Twisties (the only 270g optio
```

In [82]:
```
!pip install -U kaleido
```

```
Requirement already satisfied: kaleido in c:\users\nweke\anaconda3\lib\site-packages (0.2.1)
```

In [3]:
```
import pandas as pd
import plotly.express as px

# 1. Load your data (change filename if needed!)
data = pd.read_csv('QVI_purchase_behaviour.csv')

# 2. Sample data for performance (if big)
if len(data) > 10000:
```

```
    data = data.sample(10000, random_state=1)

# 3. Check that required columns exist
assert all(c in data.columns for c in ['PREMIUM_CUSTOMER', 'LIFESTAGE', 'LYLTY_CARD_NBR']), "Check your column

# 4. Prepare the segment counts for sunburst
segment_counts = data.groupby(['PREMIUM_CUSTOMER', 'LIFESTAGE'])['LYLTY_CARD_NBR'].nunique().reset_index()
segment_counts.columns = ['Affluence', 'LifeStage', 'CustomerCount']

# 5. Plot sunburst
fig = px.sunburst(
    segment_counts,
    path=['Affluence', 'LifeStage'],
    values='CustomerCount',
    color='Affluence',
    color_discrete_map={'Premium': '#1f77b4', 'Mainstream': '#ff7f0e', 'Budget': '#2ca02c'},
    title="Who Buys Chips? - Customer Segments by Affluence and Life Stage"
)

fig.update_traces(
    hoverinfo='none',
    hovertemplate=None,
    textinfo="label+percent entry"
)

fig.update_layout(
    margin=dict(t=60, l=0, r=0, b=0),
    width=900,
    height=700
)

fig.show()
```



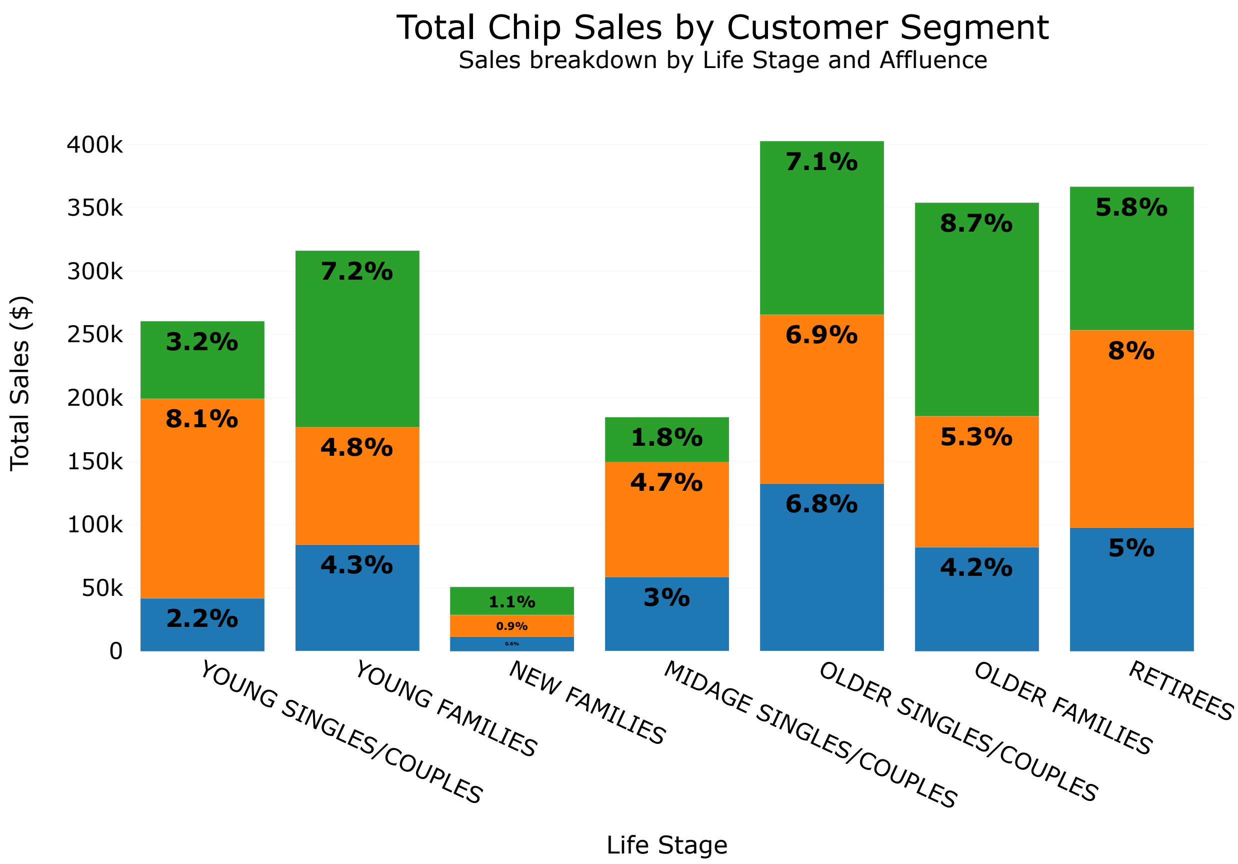
In [ ]:
```
# The sunburst chart reveals that “Mainstream Young Singles/Couples” and “Retirees” are the dominant chip-buyin
#This highlights the need for targeted marketing and product placement strategies
```

In [9]:
```
trans = pd.read_excel('QVI_transaction_data.xlsx')  # Or your CSV file
```

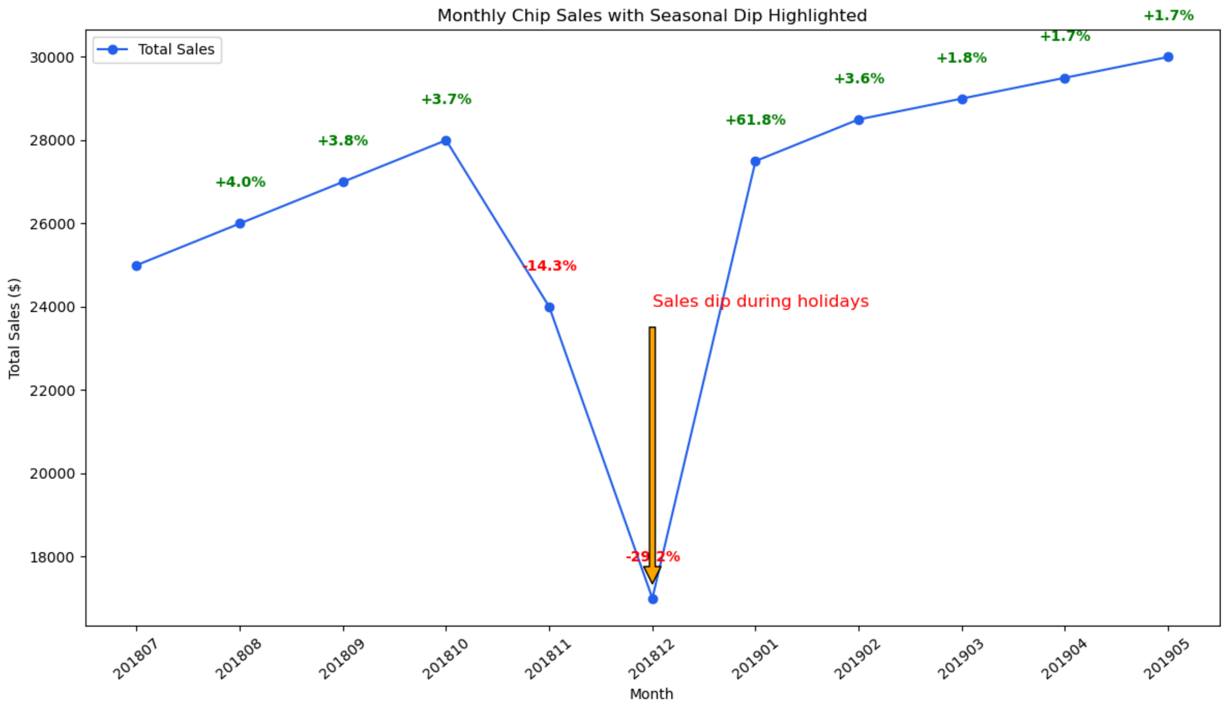
In [10]:
```
data = trans.merge(
    pd.read_csv('QVI_purchase_behaviour.csv'),
    how='left',
    on='LYLTY_CARD_NBR'
)
```

In [15]:
```
# Stacked Bar of Sales by Segment (Life Stage × Affluence)
import plotly.express as px
import pandas as pd

# Data prep
sales_segment = data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['TOT_SALES'].sum().reset_index()
total_sales = sales_segment['TOT_SALES'].sum()
sales_segment['PercentOfTotal'] = (sales_segment['TOT_SALES'] / total_sales * 100).round(1)

lifestage_order = [
    'YOUNG SINGLES/COUPLES', 'YOUNG FAMILIES', 'NEW FAMILIES',
    'MIDAGE SINGLES/COUPLES', 'OLDER SINGLES/COUPLES',
    'OLDER FAMILIES', 'RETIREES'
]
sales_segment['LIFESTAGE'] = pd.Categorical(
    sales_segment['LIFESTAGE'],
    categories=lifestage_order, ordered=True
)
sales_segment = sales_segment.sort_values('LIFESTAGE')

color_map = {
    'Premium': '#1f77b4',    # Blue
    'Mainstream': '#ff7f0e', # Orange
    'Budget': '#2ca02c'      # Green
}

fig = px.bar(
    sales_segment,
    x='LIFESTAGE',
    y='TOT_SALES',
    color='PREMIUM_CUSTOMER',
    text='PercentOfTotal',
    color_discrete_map=color_map,
    category_orders={'LIFESTAGE': lifestage_order, 'PREMIUM_CUSTOMER': ['Premium', 'Mainstream', 'Budget']},
    labels={
        "TOT_SALES": "Total Sales ($)",
        "LIFESTAGE": "Life Stage",
        "PREMIUM_CUSTOMER": "Affluence",
        "PercentOfTotal": "% of Total Sales"
    },
    title="Total Chip Sales by Customer Segment<br><sup>Sales breakdown by Life Stage and Affluence</sup>"
)

# Black percent text inside bars
fig.update_traces(
    texttemplate="<span style='color:black'><b>%{text}%</b></span>",
    textposition='inside',
    insidetextfont=dict(color='black', size=16)
)

# All other layout settings
fig.update_layout(
    width=950, height=600,
    plot_bgcolor="white",
    paper_bgcolor="white",
    title_font=dict(size=22, color="black"),
    title_x=0.5,
    font=dict(size=15, color="black"),
    barmode='stack',
    legend=dict(
        title='Affluence',
        font=dict(size=14, color="black"),
        bgcolor="rgba(255,255,255,0.5)"
    ),
    xaxis=dict(
        tickangle=25,
        title_font=dict(size=16, color="black"),
        tickfont=dict(color="black")
    ),
    yaxis=dict(
```

```
        title_font=dict(size=16, color="black"),
        gridcolor='rgba(200,200,200,0.3)',
        showgrid=True,
        tickfont=dict(color="black")
    )
)

fig.show()
```



In [6]:
```
import matplotlib.pyplot as plt

months = ['201807', '201808', '201809', '201810', '201811', '201812', '201901', '201902', '201903', '201904', '
sales = [25000, 26000, 27000, 28000, 24000, 17000, 27500, 28500, 29000, 29500, 30000]  # Example data

plt.figure(figsize=(12,7))
plt.plot(months, sales, marker='o', color='#2563eb', label='Total Sales')

# Add data labels at each point
for i, (x, y) in enumerate(zip(months, sales)):
    if i == 0:
        continue  # Skip the first month, no prior month to compare
    pct_change = (y - sales[i-1]) / sales[i-1] * 100
    plt.text(x, y + 800, f"{pct_change:+.1f}%", ha='center', va='bottom', fontsize=10, color='green' if pct_cha

plt.title('Monthly Chip Sales with Seasonal Dip Highlighted')
plt.xlabel('Month')
plt.ylabel('Total Sales ($)')
plt.xticks(rotation=40)

# Highlight the dip in December
dip_idx = months.index('201812')
plt.annotate('Sales dip during holidays', xy=(months[dip_idx], sales[dip_idx]),
             xytext=(months[dip_idx], sales[dip_idx]+7000),
             arrowprops=dict(facecolor='orange', shrink=0.05),
             fontsize=12, color='red')

plt.tight_layout()
plt.legend()
plt.show()
```



In [16]:
```
# t-test for Price per Unit Difference
from scipy.stats import ttest_ind

# 1. Create the 'PRICE_PER_UNIT' column if not already present
data['PRICE_PER_UNIT'] = data['TOT_SALES'] / data['PROD_QTY']

# 2. Select price per unit for each group
mainstream_prices = data[
    (data['LIFESTAGE'].isin(['YOUNG SINGLES/COUPLES', 'MIDAGE SINGLES/COUPLES'])) &
    (data['PREMIUM_CUSTOMER'] == 'Mainstream')
]['PRICE_PER_UNIT']

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

# 3. Run t-test
t_stat, p_value = ttest_ind(mainstream_prices, other_prices, alternative='greater')
print(f"T-statistic: {t_stat:.2f}, p-value: {p_value:.4g}")
```

```
T-statistic: 40.83, p-value: 0
```

In [ ]:
```
# I performed an independent t-test comparing the average price per packet of chips purchased by Mainstream Youn
#The result (t-statistic = 40.83, p-value ≈ 0) indicates a highly significant difference: Mainstream Young and M
#This supports the earlier finding that this segment is willing to spend more, possibly due to a preference for
```

In [ ]:
```
#Conclusion
# My analysis shows that chip sales are driven primarily by three customer segments: Budget Older Families, Mair
#The higher total spend among Mainstream Young Singles/Couples and Retirees is largely because there are simply
# Additionally, Mainstream Midage and Young Singles/Couples are willing to pay more per packet, which suggests c

#A deeper dive revealed that Mainstream Young Singles/Couples are 23% more likely to purchase Tyrrells chips and
#This indicates that targeted promotions oroff-location displays of Tyrrells and larger packs in areas frequent
#capitalize on their impulse buying behavior.

#I recommend that Julia, the Category Manager, focuses on increasing the visibility of these key brands and pack
#Further analysis could pinpoint which stores and store areas attract these segments most, enabling even more pr
# Ongoing measurement of the impact of these initiatives will help ensure that strategy changes are delivering p

#For future tasks, I can provide deeper recommendations on placement strategies and help measure the impact of
```

In [ ]: