1.LOAD REQUIRED LIBRARIES AND DATASETS

In [78]:
```
# Load required libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import re

import scipy.stats as stats

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Load both datasets
transaction_data = pd.read_csv('QVI_transaction_data.csv')
customer_data = pd.read_csv('QVI_purchase_behaviour.csv')
```

In [3]:
```
transaction_data.head(5)
```

Out[3]:

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

In [4]:
```
customer_data.head(5)
```

Out[4]:

	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

2. EXPLORATORY DATA ANALYSIS (EDA)

1. DATA CLEANING AND EXPLORATION (Transaction_data)

In [5]:

```
# examine transaction data
transaction_data.info()
print(transaction_data.head())
```

```
<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
    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   

                                  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  
```

Convert Date Column to Datetime Format

In [6]:

```
# Convert the integer format to a date format
# A search online shows that CSV and Excel integer dates begin on 30 Dec 1899
```

```
transaction_data['DATE'] = pd.to_datetime(transaction_data['DATE'], origin='1899-12
```

In [7]:
```
transaction_data['DATE'].head()
```

Out[7]:
```
0    2018-10-17
1    2019-05-14
2    2019-05-20
3    2018-08-17
4    2018-08-18
Name: DATE, dtype: datetime64[ns]
```

Summary Statistics of Transaction Data

In [8]:
```
#Generate a summary of the PROD_NAME column.
print(transaction_data['PROD_NAME'].describe())
```

```
count                                       264836
unique                                         114
top       Kettle Mozzarella   Basil & Pesto 175g
freq                                          3304
Name: PROD_NAME, dtype: object
```

In [9]:
```
# summary statistics
transaction_data.describe()
```

Out[9]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR
count	264836	264836.00000	2.648360e+05	2.648360e+05	264836.000000
mean	2018-12-30 00:52:12.879215616	135.08011	1.355495e+05	1.351583e+05	56.583157
min	2018-07-01 00:00:00	1.00000	1.000000e+03	1.000000e+00	1.000000
25%	2018-09-30 00:00:00	70.00000	7.002100e+04	6.760150e+04	28.000000
50%	2018-12-30 00:00:00	130.00000	1.303575e+05	1.351375e+05	56.000000
75%	2019-03-31 00:00:00	203.00000	2.030942e+05	2.027012e+05	85.000000
max	2019-06-30 00:00:00	272.00000	2.373711e+06	2.415841e+06	114.000000
std	NaN	76.78418	8.057998e+04	7.813303e+04	32.826638

Text Analysis of PROD_NAME

We are only interested in words that will tell us if the product is chips

In [10]:
```
product_words = pd.DataFrame(transaction_data['PROD_NAME'].str.split(expand=True).s
```

In [11]:
```
#remove all words with digits and special characters such as '&' from our set of pr
#Remove digits, and special characters, and then sort the distinct words by frequen
product_words['word'] = product_words['word'].str.replace(r'\d+', '', regex=True)
product_words['word'] = product_words['word'].str.replace(r'\W+', '', regex=True)
```

In [12]:
```
# Perform text analysis to ensure all products are chips
#Find the most common words by counting the number of times a word appears and sort
product_words = pd.DataFrame(transaction_data['PROD_NAME'].str.split().explode().va
product_words = product_words[~product_words.index.str.contains(r'\d|[^a-zA-Z\s]')]
```

In [72]:
```
print(product_words)
```

```
           count
PROD_NAME
Chips      49770
Kettle     41288
Smiths     28860
Salt       27976
Cheese     27890
...          ...
Sunbites    1432
Pc          1431
NCC         1419
Garden      1419
Fries       1418

[168 rows x 1 columns]
```

Clean and Filter the Data

Filter out any product names that contain the word "salsa," as the analysis focuses on chips

In [14]:
```
# Remove salsa products
transaction_data['SALSA'] = transaction_data['PROD_NAME'].str.contains('salsa', cas
transaction_data = transaction_data[transaction_data['SALSA'] == False].drop(column
```

In [15]:
```
transaction_data
```

Out[15]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD
0	2018-10-17	1	1000	1	5	Natural Chip Compny SeaSalt175g	
1	2019-05-14	1	1307	348	66	CCs Nacho Cheese 175g	
2	2019-05-20	1	1343	383	61	Smiths Crinkle Cut Chips Chicken 170g	
3	2018-08-17	2	2373	974	69	Smiths Chip Thinly S/Cream&Onion 175g	
4	2018-08-18	2	2426	1038	108	Kettle Tortilla ChpsHny&Jlpno Chili 150g	
...	...	...	...	...	...	...	
264831	2019-03-09	272	272319	270088	89	Kettle Sweet Chilli And Sour Cream 175g	
264832	2018-08-13	272	272358	270154	74	Tostitos Splash Of Lime 175g	
264833	2018-11-06	272	272379	270187	51	Doritos Mexicana 170g	
264834	2018-12-27	272	272379	270188	42	Doritos Corn Chip Mexican Jalapeno 150g	
264835	2018-09-22	272	272380	270189	74	Tostitos Splash Of Lime 175g	

246742 rows × 8 columns

In [16]:

```
# Summarise the data to check for nulls and possible outliers
print(transaction_data.describe())
print(transaction_data.isnull().sum())
```

```
                                DATE      STORE_NBR  LYLTY_CARD_NBR  \
count                         246742  246742.000000    2.467420e+05
mean   2018-12-30 01:19:01.211467520     135.051098    1.355310e+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    2.030840e+05
max              2019-06-30 00:00:00     272.000000    2.373711e+06
std                              NaN      76.787096    8.071528e+04

             TXN_ID       PROD_NBR       PROD_QTY      TOT_SALES
count  2.467420e+05  246742.000000  246742.000000  246742.000000
mean   1.351311e+05      56.351789       1.908062       7.321322
min    1.000000e+00       1.000000       1.000000       1.700000
25%    6.756925e+04      26.000000       2.000000       5.800000
50%    1.351830e+05      53.000000       2.000000       7.400000
75%    2.026538e+05      87.000000       2.000000       8.800000
max    2.415841e+06     114.000000     200.000000     650.000000
std    7.814772e+04      33.695428       0.659831       3.077828
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
```

Check for Outliers

Summarize the transaction data to detect potential outliers and look for transactions where an unusually large quantity of products was purchased like 200 packets

In [17]:

```python
# Filter the dataset to find the outlier
outliers = transaction_data[transaction_data['PROD_QTY'] == 200]
outliers
```

Out[17]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QT
69762	2018-08-19	226	226000	226201	4	Dorito Corn Chp Supreme 380g	2(
69763	2019-05-20	226	226000	226210	4	Dorito Corn Chp Supreme 380g	2(

In [18]:

```python
# Check other transactions made by the outlier customer.
outlier_customer = outliers['LYLTY_CARD_NBR'].iloc[0]
customer_outliers = transaction_data[transaction_data['LYLTY_CARD_NBR'] == outlier_
print(customer_outliers)
```

```
             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 [19]:
```python
# If the customer is identified as an outlier, remove their transactions from the d
transaction_data = transaction_data[transaction_data['LYLTY_CARD_NBR'] != outlier_c
```

In [20]:
```python
# Re-examine transaction data
transaction_data.describe()
```

Out[20]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR
count	246740	246740.000000	2.467400e+05	2.467400e+05	246740.000000
mean	2018-12-30 01:18:58.448569344	135.050361	1.355303e+05	1.351304e+05	56.352213
min	2018-07-01 00:00:00	1.000000	1.000000e+03	1.000000e+00	1.000000
25%	2018-09-30 00:00:00	70.000000	7.001500e+04	6.756875e+04	26.000000
50%	2018-12-30 00:00:00	130.000000	1.303670e+05	1.351815e+05	53.000000
75%	2019-03-31 00:00:00	203.000000	2.030832e+05	2.026522e+05	87.000000
max	2019-06-30 00:00:00	272.000000	2.373711e+06	2.415841e+06	114.000000
std	NaN	76.786971	8.071520e+04	7.814760e+04	33.695235

Analyze Transaction Trends Over Time

In [21]:
```python
# Count the number of transactions by date
transactions_by_day = transaction_data.groupby('DATE').size().reset_index(name='tra
transactions_by_day
```

Out[21]:

	DATE	transaction_count
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
...	...	...
359	2019-06-26	657
360	2019-06-27	669
361	2019-06-28	673
362	2019-06-29	703
363	2019-06-30	704

364 rows × 2 columns

In [22]:
```
# Create a sequence of dates and join this to the count of transactions by date
date_range = pd.date_range(start='2018-07-01', end='2019-06-30')
all_dates = pd.DataFrame(date_range, columns=['DATE'])
transactions_by_day = pd.merge(all_dates, transactions_by_day, how='left', on='DATE
transactions_by_day['transaction_count'] = transactions_by_day['transaction_count']
```

Plot Transactions Over Time

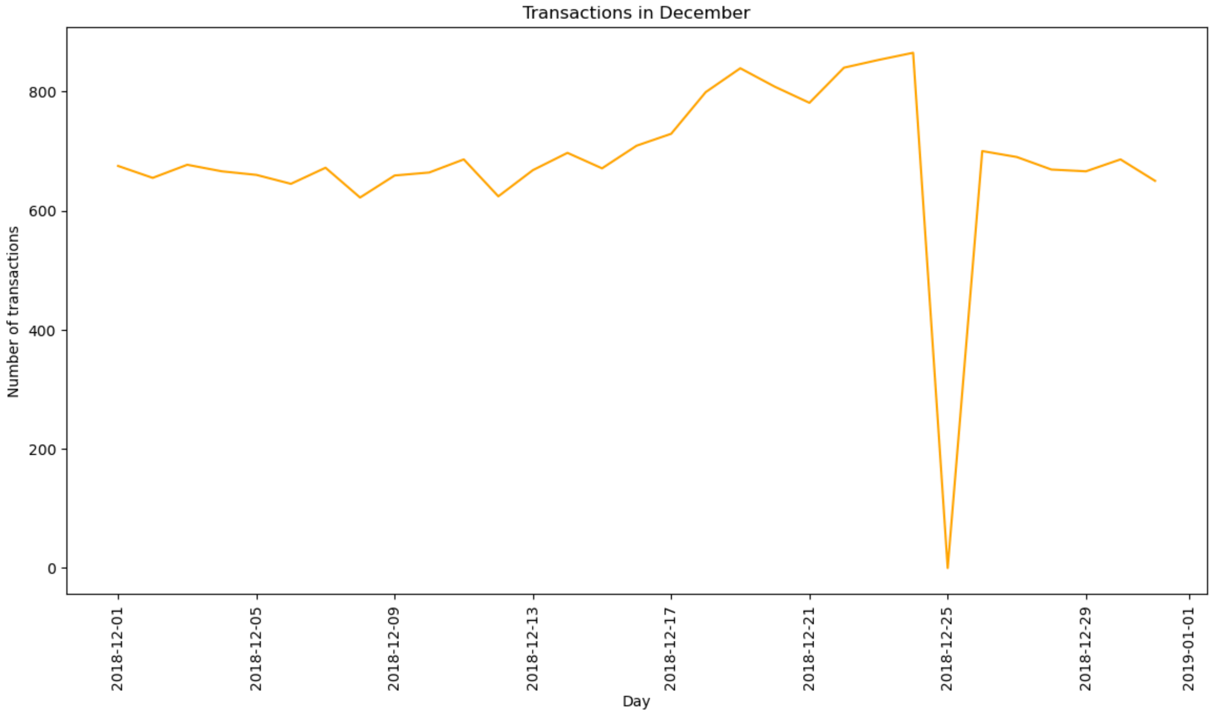
In [23]:
```
# Plot a line chart of the number of transactions over time to visually inspect tre
plt.figure(figsize=(14, 6))
sns.lineplot(data=transactions_by_day, x='DATE', y='transaction_count', color='purp
plt.title("Transactions over time")
plt.xlabel("Day")
plt.ylabel("Number of transactions")
plt.xticks(rotation=90)
plt.show()
```

In [24]:
```
# Create a sequence of dates and join this to the count of transactions by date
date_range = pd.date_range(start='2018-07-01', end='2019-06-30')
all_dates = pd.DataFrame(date_range, columns=['DATE'])
transactions_by_day = pd.merge(all_dates, transactions_by_day, how='left', on='DATE
transactions_by_day['transaction_count'] = transactions_by_day['transaction_count']
```

December Data

In [25]:
```
# Focus on December to check for any specific trends, such as increased sales aroun
December_data = transactions_by_day[(transactions_by_day['DATE'] >= '2018-12-01') &
                                    (transactions_by_day['DATE'] <= '2018-12-31')]
```

In [26]:
```
# plot transactions made in December
plt.figure(figsize=(14, 7))
sns.lineplot(data=December_data, x='DATE', y='transaction_count', color='orange')
plt.title("Transactions in December")
plt.xlabel("Day")
plt.ylabel("Number of transactions")
plt.xticks(rotation=90)
plt.show()
```



FEATURE ENGINEERING

Extract Pack Size from PROD_NAME

Extract pack size information from the product names using regular expressions, hence, analyzing different pack sizes of chips.

In [27]:
```
transaction_data['PACK_SIZE'] = transaction_data['PROD_NAME'].str.extract(r'(\d+)')
transaction_data['PACK_SIZE'].describe()
```

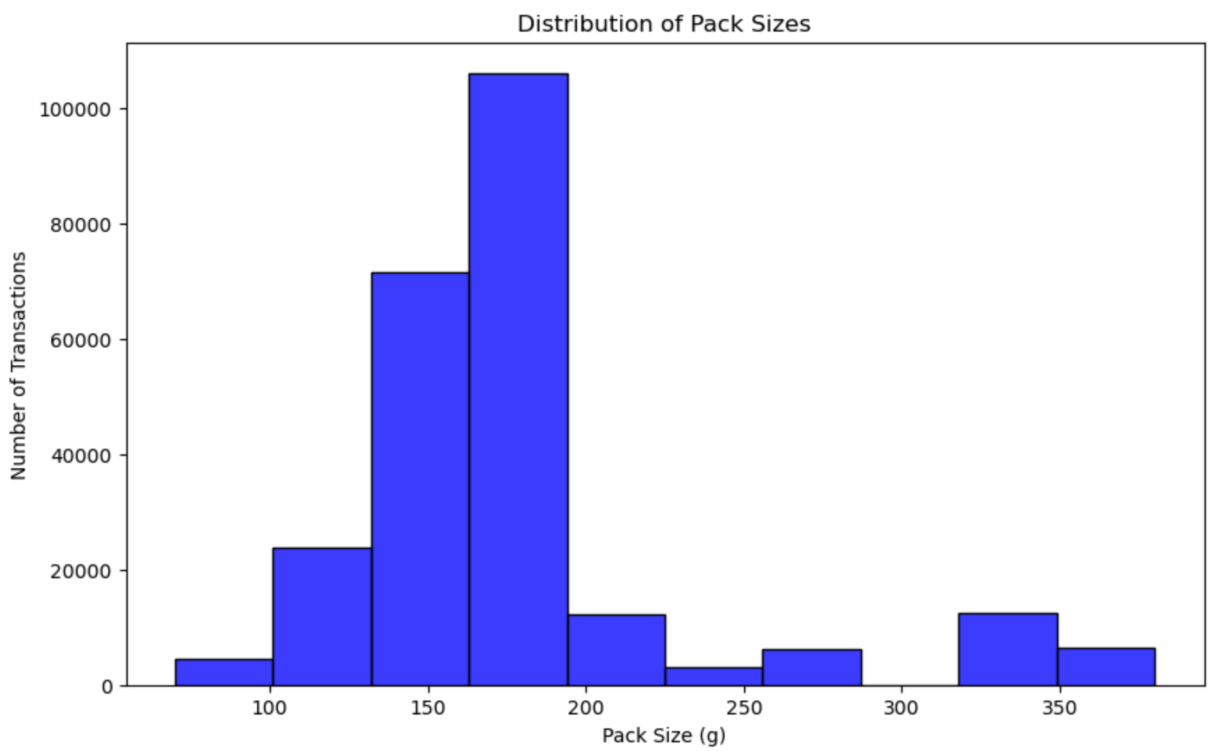
Out[27]:
```
count    246740.000000
mean        175.583521
std          59.432118
min          70.000000
25%         150.000000
50%         170.000000
75%         175.000000
max         380.000000
Name: PACK_SIZE, dtype: float64
```

In [28]:
```
# Check if the pack sizes look sensible
transaction_data['PACK_SIZE'].value_counts().sort_index()
```

Out[28]:
```
PACK_SIZE
70      1507
90      3008
110    22387
125     1454
134    25102
135     3257
150    40203
160     2970
165    15297
170    19983
175    66390
180     1468
190     2995
200     4473
210     6272
220     1564
250     3169
270     6285
330    12540
380     6416
Name: count, dtype: int64
```

Plot a Histogram of Pack Sizes

In [29]:
```python
# Visualize the distribution of pack sizes by plotting a histogram to identify the
plt.figure(figsize=(10, 6))
sns.histplot(transaction_data['PACK_SIZE'], bins=10, kde=False, color='blue')
plt.title("Distribution of Pack Sizes")
plt.xlabel("Pack Size (g)")
plt.ylabel("Number of Transactions")
plt.show()
```



Extract Brand Names from PROD_NAME

Extract this information to analyze brand performance.

In [30]:
```
# Create a column which contains the brand of the product
transaction_data['BRAND'] = transaction_data['PROD_NAME'].str.split().str[0]
```

In [31]:
```
# Check the results look reasonable
transaction_data['BRAND'].value_counts()
```

Out[31]:
```
BRAND
Kettle        41288
Smiths        27390
Pringles      25102
Doritos       22041
Thins         14075
RRD           11894
Infuzions     11057
WW            10320
Cobs           9693
Tostitos       9471
Twisties       9454
Tyrrells       6442
Grain          6272
Natural        6050
Cheezels       4603
CCs            4551
Red            4427
Dorito         3183
Infzns         3144
Smith          2963
Cheetos        2927
Snbts          1576
Burger         1564
Woolworths     1516
GrnWves        1468
Sunbites       1432
NCC            1419
French         1418
Name: count, dtype: int64
```

In [32]:
```
# Clean the brand names
transaction_data['BRAND'] = transaction_data['BRAND'].replace({'Red': 'RRD','Smith'
```

In [33]:
```
# Check again
print(transaction_data['BRAND'].value_counts())
```

```
BRAND
Kettle        41288
Smiths        30353
Doritos       25224
Pringles      25102
RRD           16321
Infuzions     14201
Thins         14075
WW            10320
Cobs           9693
Tostitos       9471
Twisties       9454
Tyrrells       6442
Grain          6272
Natural        6050
Cheezels       4603
CCs            4551
Sunbites       3008
Cheetos        2927
Burger         1564
Woolworths     1516
GrnWves        1468
NCC            1419
French         1418
Name: count, dtype: int64
```

2. DATA CLEANING AND EXPLORATION (Customer_data)

In [34]:
```python
# examine customer data
customer_data.info()
customer_data.head()
```

```
<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
```

Out[34]:

	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

In [35]:
```
# customer summary statistics
customer_data.describe()
```

Out[35]:

	LYLTY_CARD_NBR
count	7.263700e+04
mean	1.361859e+05
std	8.989293e+04
min	1.000000e+03
25%	6.620200e+04
50%	1.340400e+05
75%	2.033750e+05
max	2.373711e+06

3. MERGE TRANSACTION DATA WITH CUSTOMER DATA

In [36]:
```
# Merge transaction data to customer data
QVI_data = pd.merge(transaction_data, customer_data, how='left', on='LYLTY_CARD_NBR
```

Check for Missing Customer Details

Ensure that all transactions have corresponding customer data by checking for nulls.

In [37]:
```
# Check that no duplicates were created
print(QVI_data.shape)
print(transaction_data.shape) # The number of rows in `QVI_data` should match `tran
```

```
(246740, 12)
(246740, 10)
```

In [38]:
```
missing_customers = QVI_data['LIFESTAGE'].isnull().sum()
print(f"Missing customers: {missing_customers}")
```

```
Missing customers: 0
```

In [39]:
```
# Check for missing customer details
missing_customers = QVI_data.isnull().sum()
print(missing_customers)
```

```
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
```

No missing customer details, so all transactions are accounted for.

In [40]:
```
QVI_data.to_csv("QVI_data.csv", index=False)
```

In [41]:
```
QVI_data.head()
```

Out[41]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY
0	2018-10-17	1	1000	1	5	Natural Chip Compny SeaSalt175g	2
1	2019-05-14	1	1307	348	66	CCs Nacho Cheese 175g	3
2	2019-05-20	1	1343	383	61	Smiths Crinkle Cut Chips Chicken 170g	2
3	2018-08-17	2	2373	974	69	Smiths Chip Thinly S/Cream&Onion 175g	5
4	2018-08-18	2	2426	1038	108	Kettle Tortilla ChpsHny&Jlpno Chili 150g	3

DATA ANALYSIS ON CUSTOMER SEGMENTS

Total Sales by Lifestage and Premium Customer

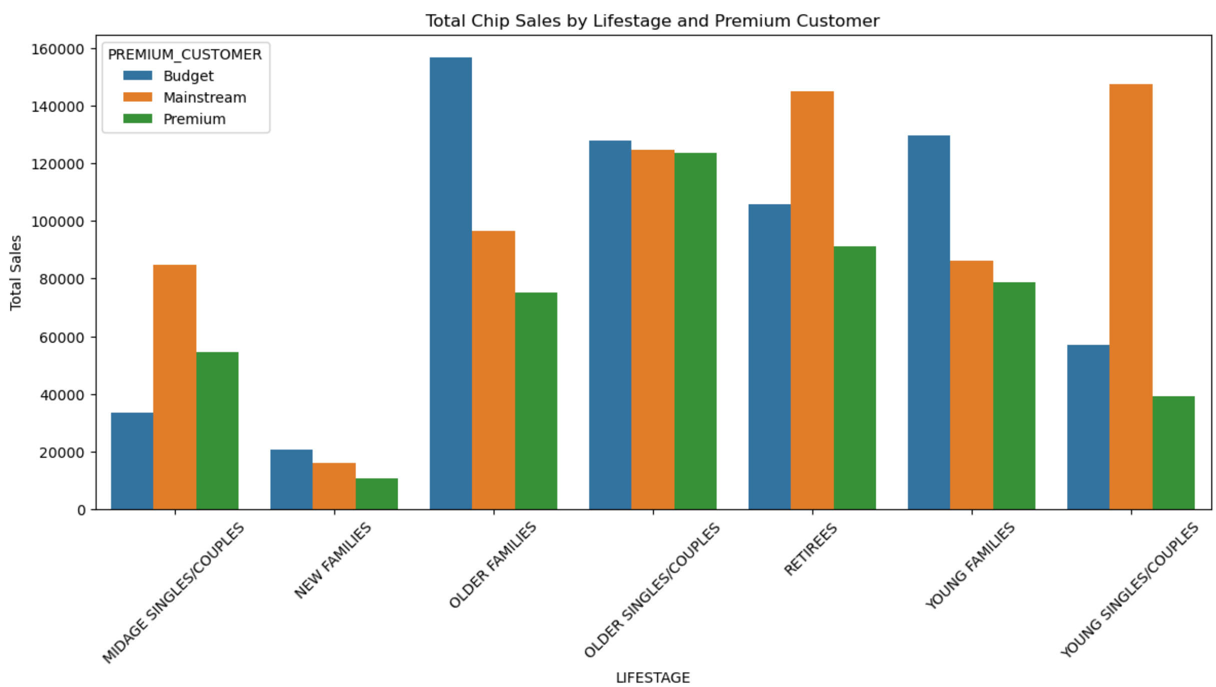
In [42]:
```
# Calculate the Total sales by LIFESTAGE and PREMIUM_CUSTOMER
total_sales = QVI_data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['TOT_SALES'].sum(
```

In [95]:
```
# Rename the columns for clarity
total_sales.columns = ['LIFESTAGE', 'PREMIUM_CUSTOMER', 'TOT_SALES']
```

```
# Display the results
print(total_sales)
```

```
                 LIFESTAGE PREMIUM_CUSTOMER  TOT_SALES
0   MIDAGE SINGLES/COUPLES           Budget   33345.70
1   MIDAGE SINGLES/COUPLES       Mainstream   84734.25
2   MIDAGE SINGLES/COUPLES          Premium   54443.85
3             NEW FAMILIES           Budget   20607.45
4             NEW FAMILIES       Mainstream   15979.70
5             NEW FAMILIES          Premium   10760.80
6           OLDER FAMILIES           Budget  156863.75
7           OLDER FAMILIES       Mainstream   96413.55
8           OLDER FAMILIES          Premium   75242.60
9    OLDER SINGLES/COUPLES           Budget  127833.60
10   OLDER SINGLES/COUPLES       Mainstream  124648.50
11   OLDER SINGLES/COUPLES          Premium  123537.55
12                RETIREES           Budget  105916.30
13                RETIREES       Mainstream  145168.95
14                RETIREES          Premium   91296.65
15          YOUNG FAMILIES           Budget  129717.95
16          YOUNG FAMILIES       Mainstream   86338.25
17          YOUNG FAMILIES          Premium   78571.70
18   YOUNG SINGLES/COUPLES           Budget   57122.10
19   YOUNG SINGLES/COUPLES       Mainstream  147582.20
20   YOUNG SINGLES/COUPLES          Premium   39052.30
```

In [96]:
```
# Plot a graph of the Total sales by LIFESTAGE and PREMIUM_CUSTOMER
plt.figure(figsize=(14, 6))
sns.barplot(x='LIFESTAGE', y='TOT_SALES', hue='PREMIUM_CUSTOMER', data=total_sales)
plt.title("Total Chip Sales by Lifestage and Premium Customer")
plt.ylabel("Total Sales")
plt.xticks(rotation=45)
plt.show()
```



Number of Customers by Lifestage and Premium Customer

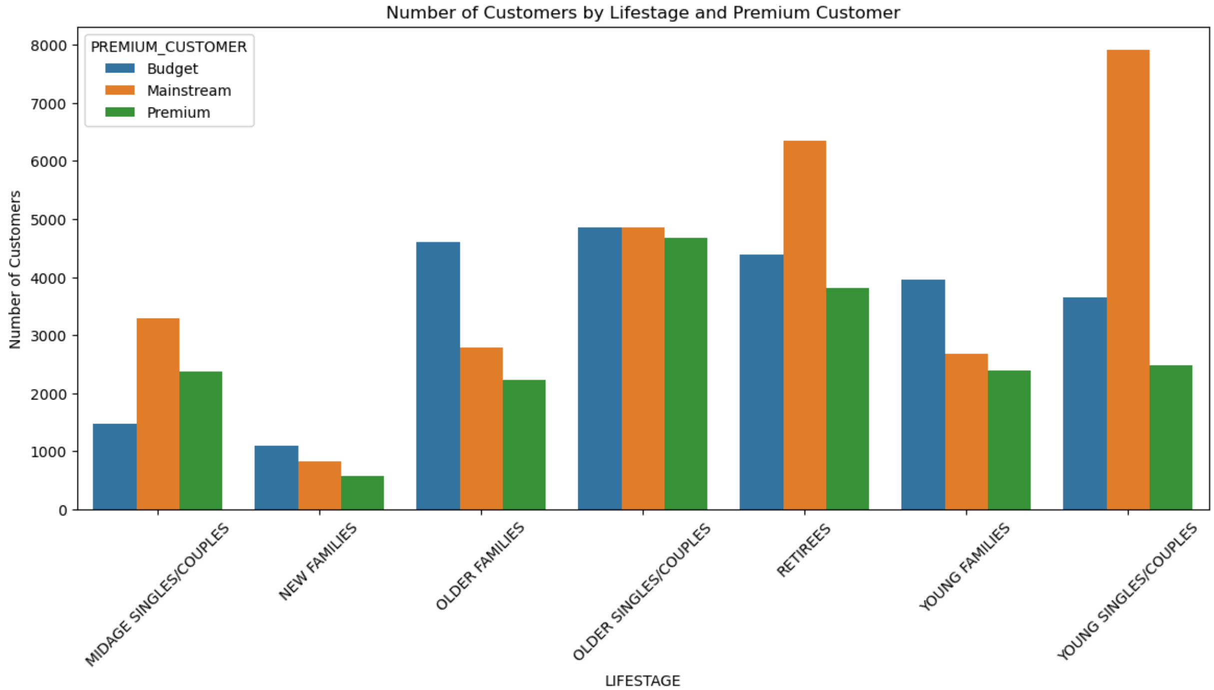
In [81]:
```
# Count the Number of customers by LIFESTAGE and PREMIUM_CUSTOMER
customer_counts = QVI_data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['LYLTY_CARD_N
```

In [82]:
```
print(customer_counts)
```

```
                 LIFESTAGE PREMIUM_CUSTOMER  LYLTY_CARD_NBR
0   MIDAGE SINGLES/COUPLES           Budget            1474
1   MIDAGE SINGLES/COUPLES       Mainstream            3298
2   MIDAGE SINGLES/COUPLES          Premium            2369
3             NEW FAMILIES           Budget            1087
4             NEW FAMILIES       Mainstream             830
5             NEW FAMILIES          Premium             575
6           OLDER FAMILIES           Budget            4611
7           OLDER FAMILIES       Mainstream            2788
8           OLDER FAMILIES          Premium            2231
9    OLDER SINGLES/COUPLES           Budget            4849
10   OLDER SINGLES/COUPLES       Mainstream            4858
11   OLDER SINGLES/COUPLES          Premium            4682
12                RETIREES           Budget            4385
13                RETIREES       Mainstream            6358
14                RETIREES          Premium            3812
15          YOUNG FAMILIES           Budget            3953
16          YOUNG FAMILIES       Mainstream            2685
17          YOUNG FAMILIES          Premium            2398
18   YOUNG SINGLES/COUPLES           Budget            3647
19   YOUNG SINGLES/COUPLES       Mainstream            7917
20   YOUNG SINGLES/COUPLES          Premium            2480
```

In [84]:
```
#Plot the Count the Number of customers by LIFESTAGE and PREMIUM_CUSTOMER
plt.figure(figsize=(14, 6))
sns.barplot(x='LIFESTAGE', y='LYLTY_CARD_NBR', hue='PREMIUM_CUSTOMER', data=custome

plt.title("Number of Customers by Lifestage and Premium Customer")
plt.ylabel("Number of Customers")
plt.xticks(rotation=45)
plt.show()
```



Number of chips are bought per customer by segment

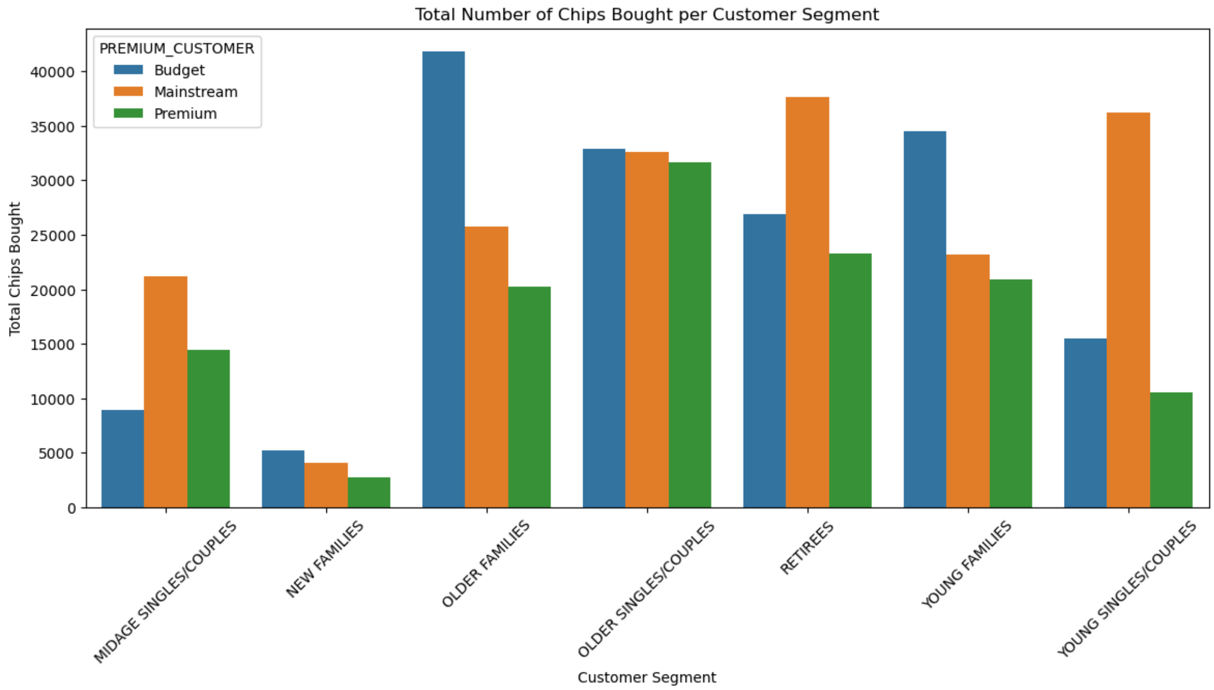
In [73]:
```
# Group by LIFESTAGE and PREMIUM_CUSTOMER and sum the PROD_QTY
chips_per_customer_segment = QVI_data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['P
```

In [74]:
```
print(chips_per_customer_segment)
```

```
                 LIFESTAGE PREMIUM_CUSTOMER  PROD_QTY
0   MIDAGE SINGLES/COUPLES           Budget      8883
1   MIDAGE SINGLES/COUPLES       Mainstream     21213
2   MIDAGE SINGLES/COUPLES          Premium     14400
3             NEW FAMILIES           Budget      5241
4             NEW FAMILIES       Mainstream      4060
5             NEW FAMILIES          Premium      2769
6           OLDER FAMILIES           Budget     41853
7           OLDER FAMILIES       Mainstream     25804
8           OLDER FAMILIES          Premium     20239
9    OLDER SINGLES/COUPLES           Budget     32883
10   OLDER SINGLES/COUPLES       Mainstream     32607
11   OLDER SINGLES/COUPLES          Premium     31695
12                RETIREES           Budget     26932
13                RETIREES       Mainstream     37677
14                RETIREES          Premium     23266
15          YOUNG FAMILIES           Budget     34482
16          YOUNG FAMILIES       Mainstream     23194
17          YOUNG FAMILIES          Premium     20901
18   YOUNG SINGLES/COUPLES           Budget     15500
19   YOUNG SINGLES/COUPLES       Mainstream     36225
20   YOUNG SINGLES/COUPLES          Premium     10575
```

In [85]:
```
# Plot the total number of chips bought per customer segment
plt.figure(figsize=(14, 6))
sns.barplot(data=chips_per_customer_segment, x='LIFESTAGE', y='PROD_QTY', hue='PREM
plt.title("Total Number of Chips Bought per Customer Segment")
```

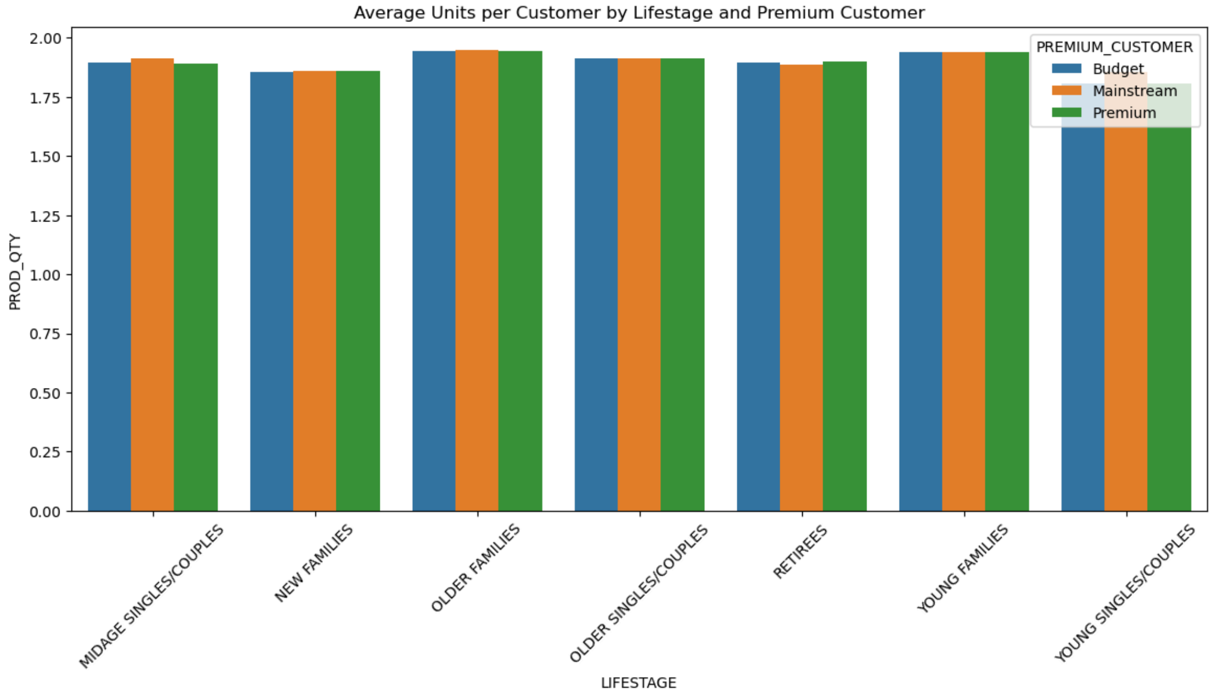
```
plt.xticks(rotation=45)
plt.ylabel("Total Chips Bought")
plt.xlabel("Customer Segment")
plt.show()
```



Average Units per Customer by Segment

In [46]:
```
units_per_customer =QVI_data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['PROD_QTY']
```

In [47]:
```
# Analyze how many units of chips each customer segment typically buys
plt.figure(figsize=(14, 6))
sns.barplot(data=units_per_customer, x='LIFESTAGE', y='PROD_QTY', hue='PREMIUM_CUST
plt.title("Average Units per Customer by Lifestage and Premium Customer")
plt.xticks(rotation=45)
plt.show()
```



Average Price per Unit by Segment

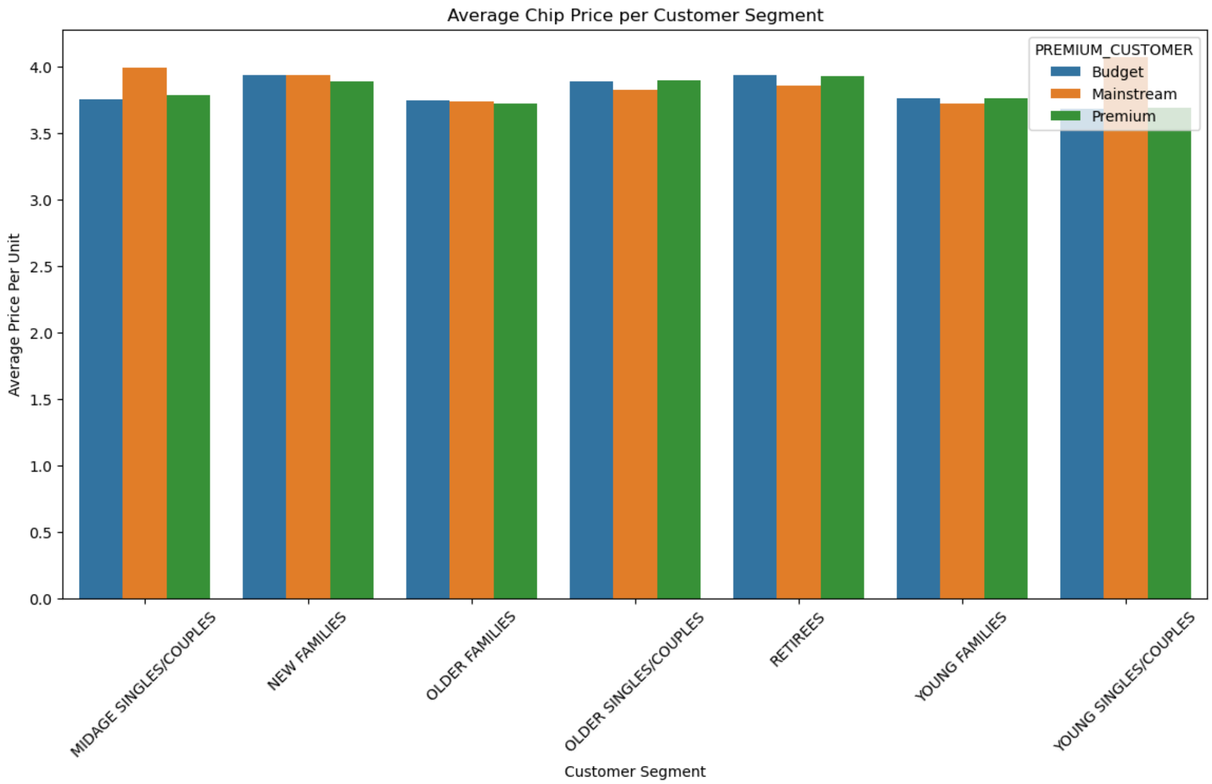
In [92]:
```
# Calculate total sales and total quantity by segment
avg_price_per_segment = QVI_data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER']).apply(l
```

In [93]:
```
# Rename the columns for clarity
avg_price_per_segment.columns = ['LIFESTAGE', 'PREMIUM_CUSTOMER', 'Avg_Price_Per_Un

# Display the results
print(avg_price_per_segment)
```

```
                 LIFESTAGE PREMIUM_CUSTOMER  Avg_Price_Per_Unit
0   MIDAGE SINGLES/COUPLES           Budget            3.753878
1   MIDAGE SINGLES/COUPLES       Mainstream            3.994449
2   MIDAGE SINGLES/COUPLES          Premium            3.780823
3             NEW FAMILIES           Budget            3.931969
4             NEW FAMILIES       Mainstream            3.935887
5             NEW FAMILIES          Premium            3.886168
6           OLDER FAMILIES           Budget            3.747969
7           OLDER FAMILIES       Mainstream            3.736380
8           OLDER FAMILIES          Premium            3.717703
9    OLDER SINGLES/COUPLES           Budget            3.887529
10   OLDER SINGLES/COUPLES       Mainstream            3.822753
11   OLDER SINGLES/COUPLES          Premium            3.897698
12                RETIREES           Budget            3.932731
13                RETIREES       Mainstream            3.852986
14                RETIREES          Premium            3.924037
15          YOUNG FAMILIES           Budget            3.761903
16          YOUNG FAMILIES       Mainstream            3.722439
17          YOUNG FAMILIES          Premium            3.759232
18   YOUNG SINGLES/COUPLES           Budget            3.685297
19   YOUNG SINGLES/COUPLES       Mainstream            4.074043
20   YOUNG SINGLES/COUPLES          Premium            3.692889
```

In [89]:
```
# Plot the average price per unit by customer segment
plt.figure(figsize=(14, 7))
sns.barplot(data=avg_price_per_segment, x='LIFESTAGE', y='Avg_Price_Per_Unit', hue=
plt.title("Average Chip Price per Customer Segment")
plt.xticks(rotation=45)
plt.ylabel("Average Price Per Unit")
plt.xlabel("Customer Segment")
plt.show()
```



STATISTICAL TESTING

Perform a t-test to see if the differences in average spending between customer segments are statistically significant.

Perform an independent t-test between mainstream vs premium and budget midage young singles and couples

The output will provide the t-statistics and p-values for each pair of comparisons. A low p-value (typically less than 0.05) indicates that there is a statistically significant difference in means between the two groups being compared.

In [50]:
```
from scipy.stats import ttest_ind
```

In [97]:
```
# Filter data for the specific segments
mainstream_young_midage = QVI_data[(QVI_data['LIFESTAGE'].isin(['YOUNG SINGLES/COUP
                                   (QVI_data['PREMIUM_CUSTOMER'] == 'Mainstream')]

premium_young_midage = QVI_data[(QVI_data['LIFESTAGE'].isin(['YOUNG SINGLES/COUPLES
                                (QVI_data['PREMIUM_CUSTOMER'] == 'Premium')]
```

```
budget_young_midage = QVI_data[(QVI_data['LIFESTAGE'].isin(['YOUNG SINGLES/COUPLES'
                            (QVI_data['PREMIUM_CUSTOMER'] == 'Budget')]
```

In [99]:
```
# Perform t-test between mainstream and premium groups
t_stat_mainstream_premium, p_val_mainstream_premium = ttest_ind(mainstream_young_mi

# Perform t-test between mainstream and budget groups
t_stat_mainstream_budget, p_val_mainstream_budget = ttest_ind(mainstream_young_mida

# Perform t-test between premium and budget groups
t_stat_premium_budget, p_val_premium_budget = ttest_ind(premium_young_midage['TOT_S
```

In [100…
```
# Output the t-statistics and p-values
print(f"T-test between Mainstream and Premium:")
print(f"T-statistic: {t_stat_mainstream_premium}, P-value: {p_val_mainstream_premiu

print(f"\nT-test between Mainstream and Budget:")
print(f"T-statistic: {t_stat_mainstream_budget}, P-value: {p_val_mainstream_budget}

print(f"\nT-test between Premium and Budget:")
print(f"T-statistic: {t_stat_premium_budget}, P-value: {p_val_premium_budget}")
```

```
T-test between Mainstream and Premium:
T-statistic: 24.77672858209525, P-value: 1.3358339199035904e-134

T-test between Mainstream and Budget:
T-statistic: 29.37968796720024, P-value: 6.642280216613805e-188

T-test between Premium and Budget:
T-statistic: 3.893400294889745, P-value: 9.908894718247683e-05
```

Interpretation: If the p_value < 0.05, there is a significant difference in price.s.

INSIGHTS INTO SOME OF THE CUSTOMER SEGMENTS

Analyze Brand Preference

In [114…
```
# Filter the dataset for "Mainstream - young singles/couples" segment
mainstream_young = QVI_data[(QVI_data['LIFESTAGE'] == 'YOUNG SINGLES/COUPLES') & (Q

# Calculate the frequency of each brand within this segment
brand_frequency = mainstream_young['BRAND'].value_counts(normalize=True) * 100

# Calculate the frequency of each brand in the rest of the population
overall_brand_frequency = QVI_data['BRAND'].value_counts(normalize=True) * 100
```

In [115…
```
# Compare the frequencies
brand_comparison = pd.DataFrame({ 'Mainstream Young Singles/Couples (%)': brand_fre
```

In [116…
```
# Sort the brands by their preference in the Mainstream Young Singles/Couples segme
brand_comparison = brand_comparison.sort_values(by='Mainstream Young Singles/Couple
```

In [118…
```
brand_comparison
```

Out[118…

BRAND	Mainstream Young Singles/Couples (%)	Overall Population (%)
Kettle	19.668440	16.733404
Doritos	12.172534	10.222907
Pringles	11.845068	10.173462
Smiths	9.829104	12.301613
Infuzions	6.395825	5.755451
Thins	5.966025	5.704385
Twisties	4.604994	3.831564
Tostitos	4.553827	3.838453
RRD	4.477077	6.614655
Cobs	4.420794	3.928427
Tyrrells	3.167212	2.610845
Grain	2.947196	2.541947
WW	2.164347	4.182540
Cheezels	1.770364	1.865526
Natural	1.642448	2.451974
CCs	1.135898	1.844452
Cheetos	0.849366	1.186269
Sunbites	0.654932	1.219097
French	0.399099	0.574694
NCC	0.373516	0.575099
GrnWves	0.358166	0.594958
Burger	0.317233	0.633866
Woolworths	0.286533	0.614412

Insights of the above

1. Preference for Premium Brands:

Kettle is the top brand for "Mainstream Young Singles/Couples," representing 19.67% of their purchases, which is higher than the overall population's preference at 16.73%. This indicates a strong inclination toward premium, gourmet-style chips within this segment.

2. Popularity of Doritos and Pringles:

Both Doritos (12.17%) and Pringles (11.85%) are more popular among this segment compared to the overall population (10.22% and 10.17%, respectively). These brands are known for their bold flavors, suggesting that this group may prefer more intense or diverse taste experiences.

3. Underrepresentation of Traditional Brands:

Brands like Smiths and RRD are less favored by this segment (9.83% and 4.48%, respectively) compared to their overall popularity (12.30% and 6.61%). This could indicate that "Mainstream Young Singles/Couples" are less interested in traditional or classic chip brands, possibly preferring newer or more trendy options.

4. Strong Affinity for Niche or Health-Conscious Brands:

Brands like Infuzions (6.40%) and Cobs (4.42%) have a higher purchase rate among this segment than the overall population. These brands often market themselves as healthier or more unique alternatives, suggesting that health-conscious or novelty-seeking behaviors are more prevalent in this group.

5. Lower Purchase of Supermarket Brands:

WW (Woolworths) and Woolworths brand chips have significantly lower purchase rates in this segment (2.16% and 0.29%) compared to the overall population (4.18% and 0.61%). This might suggest that "Mainstream Young Singles/Couples" prefer branded products over private label or store brands.

6. Potential Targets for Marketing:

The brands that are underrepresented in this segment, such as Smiths, RRD, and WW, could be targeted with marketing strategies tailored to appeal more to "Mainstream Young Singles/Couples." For instance, these brands could introduce new flavors, packaging, or promotional campaigns that resonate with the preferences of this demographic.

Conclusion: "Mainstream Young Singles/Couples" tend to prefer premium, bold-flavored, and health-conscious chip brands over traditional or supermarket brands. Marketing strategies targeting this segment should focus on enhancing the appeal of niche, gourmet, or innovative products. Brands that are underperforming in this segment could benefit from repositioning or launching targeted campaigns to capture their interest.

Analyze Pack Preference

In [120…

```
# Calculate the distribution of pack sizes for the target segment and overall popul
overall_pack_size_dist = QVI_data['PACK_SIZE'].value_counts(normalize=True).sort_in
young_pack_size_dist = mainstream_young['PACK_SIZE'].value_counts(normalize=True).s
```

In [121…

```
# Combine the distributions into a single DataFrame for comparison
pack_size_comparison = pd.DataFrame({'Mainstream Young Singles/Couples (%)': young_
```

In [122…

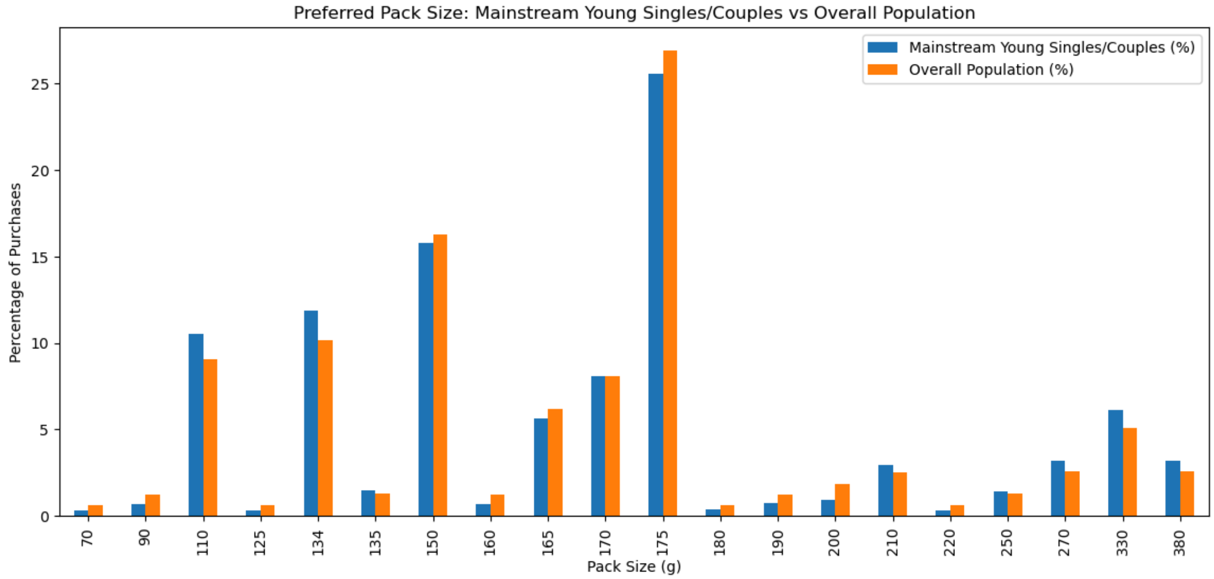
```
pack_size_comparison
```

Out[122…

PACK_SIZE	Mainstream Young Singles/Couples (%)	Overall Population (%)
70	0.322350	0.610764
90	0.654932	1.219097
110	10.494269	9.073113
125	0.301883	0.589284
134	11.845068	10.173462
135	1.483831	1.320013
150	15.759312	16.293669
160	0.654932	1.203696
165	5.638559	6.199643
170	8.058739	8.098808
175	25.567949	26.906866
180	0.358166	0.594958
190	0.757266	1.213828
200	0.915882	1.812839
210	2.947196	2.541947
220	0.317233	0.633866
250	1.432665	1.284348
270	3.172329	2.547216
330	6.114409	5.082273
380	3.203029	2.600308

In [124…

```
# Plot the comparison
pack_size_comparison.plot(kind='bar', figsize=(14, 6))
plt.title('Preferred Pack Size: Mainstream Young Singles/Couples vs Overall Populat
plt.xlabel('Pack Size (g)')
plt.ylabel('Percentage of Purchases')
plt.show()
```



Insights

Mainstream Young Singles/Couples and overall population: they prefer pack sizes of 175g

What is the preffered pack size

SUMMARY

Total Sales: Calculated by customer segments (LIFESTAGE and PREMIUM_CUSTOMER).

Number of Customers: Analyzed the distribution of customers by segments.

Average Units per Customer: Explored the average number of chip units purchased by segment.

Average Price per Unit: Examined the average price paid per unit by different customer segments.

T-test: Tested if the price difference between segments is statistically significant.

More Analysis: Focused on the preferences and behaviors of the "Mainstream - Young Singles/Couples" segment, specifically in terms of brand and pack size preferences.

In [ ]: