Title : "Quantium Virtual Internship - Retail Strategy and Analytics - Task 1"

Load required Libraries.

In [84]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import re
from collections import Counter
```

Import Dataset

In [20]:

```
# Install dependencies as needed:
# pip install kagglehub[pandas-datasets]
import kagglehub
from kagglehub import KaggleDatasetAdapter

# Set the path to the file you'd like to load
file_path = "QVI_transaction_data.csv"

# Load the latest version
transaction_data = kagglehub.dataset_load(
  KaggleDatasetAdapter.PANDAS,
  "preethis14/qvi-transaction-data",
  file_path,
  # Provide any additional arguments like
  # sql_query or pandas_kwargs. See the
  # documenation for more information:
  # https://github.com/Kaggle/kagglehub/blob/main/README.md#kaggledatasetadapterpandas
)

print("First 5 records:", transaction_data.head())
```

```
First 5 records:      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
```

In [22]:

```
# Install dependencies as needed:
# pip install kagglehub[pandas-datasets]
import kagglehub
from kagglehub import KaggleDatasetAdapter

# Set the path to the file you'd like to load
file_path = "QVI_purchase_behaviour.csv"

# Load the latest version
customer_data = kagglehub.dataset_load(
  KaggleDatasetAdapter.PANDAS,
  "preethis14/qvi-purchase-behaviour",
  file_path,
  # Provide any additional arguments like
  # sql_query or pandas_kwargs. See the
  # documenation for more information:
  # https://github.com/Kaggle/kagglehub/blob/main/README.md#kaggledatasetadapterpandas
)

print("First 5 records:", customer_data.head())
```

```
First 5 records:    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
```

Examine the transaction dataset. Look for the format of each column.

In [28]:

```
data_types = transaction_data.dtypes
print(data_types)
```

```
DATE                 int64
STORE_NBR            int64
LYLTY_CARD_NBR       int64
TXN_ID               int64
PROD_NBR             int64
PROD_NAME           object
PROD_QTY             int64
TOT_SALES          float64
dtype: object
```

The dates are in Excel serial date format. Conversion to the required format.

In [63]:

```
excel_dates = transaction_data['DATE']

dates = pd.to_datetime(excel_dates, origin='1899-12-30', unit='D')

transaction_data['DATE'] = dates

print(dates)
```

```
0        2018-10-17
1        2019-05-14
2        2019-05-20
3        2018-08-17
4        2018-08-18
            ...    
264831   2019-03-09
264832   2018-08-13
264833   2018-11-06
264834   2018-12-27
264835   2018-09-22
Name: DATE, Length: 246742, dtype: datetime64[ns]
```

Summary of product name. examine the product name column.

In [40]:

```
product_name = transaction_data['PROD_NAME']
print(product_name.info())
print(product_name.describe(include = "all"))
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 264836 entries, 0 to 264835
Series name: PROD_NAME
Non-Null Count   Dtype 
--------------   ----- 
264836 non-null  object
dtypes: object(1)
memory usage: 2.0+ MB
None
count                                       264836
unique                                         114
top       Kettle Mozzarella   Basil & Pesto 175g
freq                                          3304
Name: PROD_NAME, dtype: object
```

Checking for any inconsistencies in product name column.

In [51]:

```
def extract_clean_words(name):
    cleaned_name = re.sub(r'[^a-zA-Z\s]', '', name)
    words = cleaned_name.split()
    return ' '.join(words)

transaction_data['CLEAN_PROD_WORDS'] = transaction_data['PROD_NAME'].apply(extract_clean_words)

print(transaction_data[['PROD_NAME', 'CLEAN_PROD_WORDS']].head())
```

```
                                  PROD_NAME  \
0    Natural Chip        Compny SeaSalt175g   
1                  CCs Nacho Cheese    175g   
2    Smiths Crinkle Cut  Chips Chicken 170g   
3    Smiths Chip Thinly  S/Cream&Onion 175g   
4  Kettle Tortilla ChpsHny&Jlpno Chili 150g   

                       CLEAN_PROD_WORDS  
0          Natural Chip Compny SeaSaltg  
1                    CCs Nacho Cheese g  
2    Smiths Crinkle Cut Chips Chicken g  
3      Smiths Chip Thinly SCreamOnion g  
4  Kettle Tortilla ChpsHnyJlpno Chili g  
```

Most common words by counting the number of times a word appears and sorting them by this frequency in order of highest to lowest frequency

In [55]:

```
all_words = ' '.join(transaction_data['CLEAN_PROD_WORDS']).split()

word_counts = Counter(all_words)

def get_word_frequency(name):
    words = name.split()
    return sorted([word_counts[word] for word in words], reverse = True)

transaction_data['WORD_FREQUENCY'] = transaction_data['CLEAN_PROD_WORDS'].apply(get_word_frequency)

print(transaction_data[['PROD_NAME', 'CLEAN_PROD_WORDS', 'WORD_FREQUENCY']].head())
```

```
                                  PROD_NAME  \
0    Natural Chip        Compny SeaSalt175g
1                  CCs Nacho Cheese    175g
2    Smiths Crinkle Cut  Chips Chicken 170g
3    Smiths Chip Thinly  S/Cream&Onion 175g
4  Kettle Tortilla ChpsHny&Jlpno Chili 150g

                       CLEAN_PROD_WORDS  \
0           Natural Chip Compny SeaSaltg
1                     CCs Nacho Cheese g
2    Smiths Crinkle Cut Chips Chicken g
3      Smiths Chip Thinly SCreamOnion g
4  Kettle Tortilla ChpsHnyJlpno Chili g

                                WORD_FREQUENCY
0                    [18645, 6050, 1468, 1468]
1                  [246628, 27890, 4658, 4551]
2  [246628, 49770, 28860, 23960, 20754, 15407]
3           [246628, 28860, 18645, 7507, 1473]
4            [246628, 41288, 9580, 3296, 3296]
```

Checking for chips and not chips

In [56]:

```
transaction_data['SALSA'] = transaction_data['PROD_NAME'].str.contains('salsa', case=False, na=False)

transaction_data = transaction_data[transaction_data['SALSA'] == False]

transaction_data = transaction_data.drop(columns=['SALSA'])

print(transaction_data.head())
```

```
    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

                       CLEAN_PROD_WORDS  \
0           Natural Chip Compny SeaSaltg
1                     CCs Nacho Cheese g
2    Smiths Crinkle Cut Chips Chicken g
3      Smiths Chip Thinly SCreamOnion g
4  Kettle Tortilla ChpsHnyJlpno Chili g

                                WORD_FREQUENCY
0                    [18645, 6050, 1468, 1468]
1                  [246628, 27890, 4658, 4551]
2  [246628, 49770, 28860, 23960, 20754, 15407]
3           [246628, 28860, 18645, 7507, 1473]
4            [246628, 41288, 9580, 3296, 3296]
```

In [77]:

```
print(transaction_data.describe())
print(transaction_data.info())
print("check for any null values :\n",transaction_data.isnull().sum())
```

```
                                DATE      STORE_NBR  LYLTY_CARD_NBR  \
count                         246742  246742.000000    2.467420e+05   
mean   2018-12-30 01:19:01.211468032     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  
<class 'pandas.core.frame.DataFrame'>
Index: 246742 entries, 0 to 264835
Data columns (total 10 columns):
 #   Column            Non-Null Count   Dtype         
---  ------            --------------   -----         
 0   DATE              246742 non-null  datetime64[ns]
 1   STORE_NBR         246742 non-null  int64         
 2   LYLTY_CARD_NBR    246742 non-null  int64         
 3   TXN_ID            246742 non-null  int64         
 4   PROD_NBR          246742 non-null  int64         
 5   PROD_NAME         246742 non-null  object        
 6   PROD_QTY          246742 non-null  int64         
 7   TOT_SALES         246742 non-null  float64       
 8   CLEAN_PROD_WORDS  246742 non-null  object        
 9   WORD_FREQUENCY    246742 non-null  object        
dtypes: datetime64[ns](1), float64(1), int64(5), object(3)
memory usage: 20.7+ MB
None
check for any null values :
 DATE                0
STORE_NBR           0
LYLTY_CARD_NBR      0
TXN_ID              0
PROD_NBR            0
PROD_NAME           0
PROD_QTY            0
TOT_SALES           0
CLEAN_PROD_WORDS    0
WORD_FREQUENCY      0
dtype: int64
```

Filter outliers

In [80]:

```python
outlier_customer = outliers['LYLTY_CARD_NBR'].unique()[0]

outliers = transaction_data[transaction_data['PROD_QTY'] == 200]
print(outliers)

transaction_data = transaction_data[transaction_data['LYLTY_CARD_NBR'] != outlier_customer]

print(transaction_data.head())
```

```
            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   

                CLEAN_PROD_WORDS                      WORD_FREQUENCY  
69762  Dorito Corn Chp Supreme g  [246628, 22063, 10963, 3185, 3185]  
69763  Dorito Corn Chp Supreme g  [246628, 22063, 10963, 3185, 3185]  
        DATE  STORE_NBR  LYLTY_CARD_NBR  TXN_ID  PROD_NBR  \
0 2018-10-17          1            1000       1         5   
1 2019-05-14          1            1307     348        66   
2 2019-05-20          1            1343     383        61   
3 2018-08-17          2            2373     974        69   
4 2018-08-18          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   

                         CLEAN_PROD_WORDS  \
0          Natural Chip Compny SeaSaltg   
1                    CCs Nacho Cheese g   
2    Smiths Crinkle Cut Chips Chicken g   
3      Smiths Chip Thinly SCreamOnion g   
4  Kettle Tortilla ChpsHnyJlpno Chili g   

                               WORD_FREQUENCY  
0                    [18645, 6050, 1468, 1468]  
1                  [246628, 27890, 4658, 4551]  
2  [246628, 49770, 28860, 23960, 20754, 15407]  
3           [246628, 28860, 18645, 7507, 1473]  
4            [246628, 41288, 9580, 3296, 3296]  
```

In [83]:

```python
transactions_counts = transaction_data.groupby('DATE').size().reset_index(name='TRANSACTION_COUNT')
print(transactions_by_date.head())
transactions_by_date.count()
```

```
        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
```

Out[83]:

```
DATE                 364
TRANSACTION_COUNT    364
dtype: int64
```

As we seein the above data, it shown that there are only 364 rows, meaning only 364 dates only which indicates two dates are missing.

A chart of number of transactions over time to find the missing date.

In [89]:

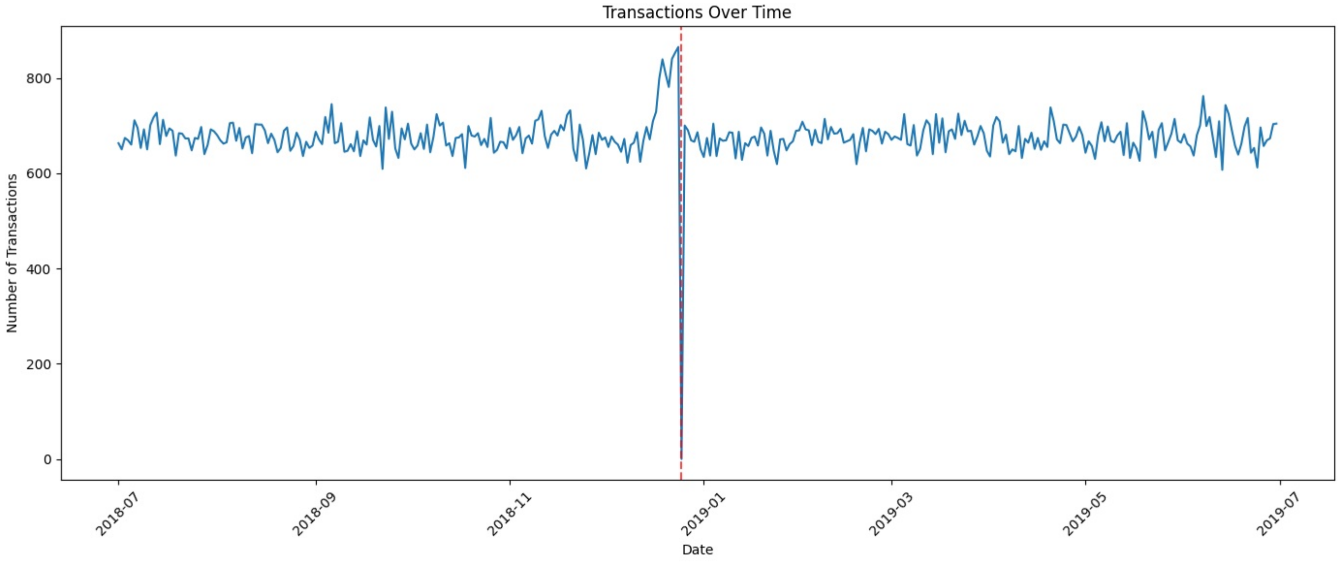
```python
full_date_range = pd.date_range(start='2018-07-01', end='2019-06-30')

transaction_counts = transaction_data.groupby('DATE').size().reset_index(name='TRANSACTION_COUNT')

transaction_data_daily = pd.DataFrame({'DATE': full_date_range})

transaction_data_daily = transaction_data_daily.merge(transaction_counts, on='DATE', how='left')

transaction_data_daily['TRANSACTION_COUNT'] = transaction_data_daily['TRANSACTION_COUNT'].fillna(0)

missing_dates = transaction_data_daily[transaction_data_daily['TRANSACTION_COUNT'] == 0]['DATE']
print("Missing date(s):")
print(missing_dates.dt.strftime('%Y-%m-%d').tolist())
```

```
Missing date(s):
['2018-12-25']
```

In [90]:

```python
plt.figure(figsize=(14, 6))
sns.lineplot(data=transaction_data_daily, x='DATE', y='TRANSACTION_COUNT')

for date in missing_dates:
    plt.axvline(x=date, color='red', linestyle='--', alpha=0.7, label='Missing Date')

plt.title('Transactions Over Time')
plt.xlabel('Date')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprec
ated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprec
ated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [91]:

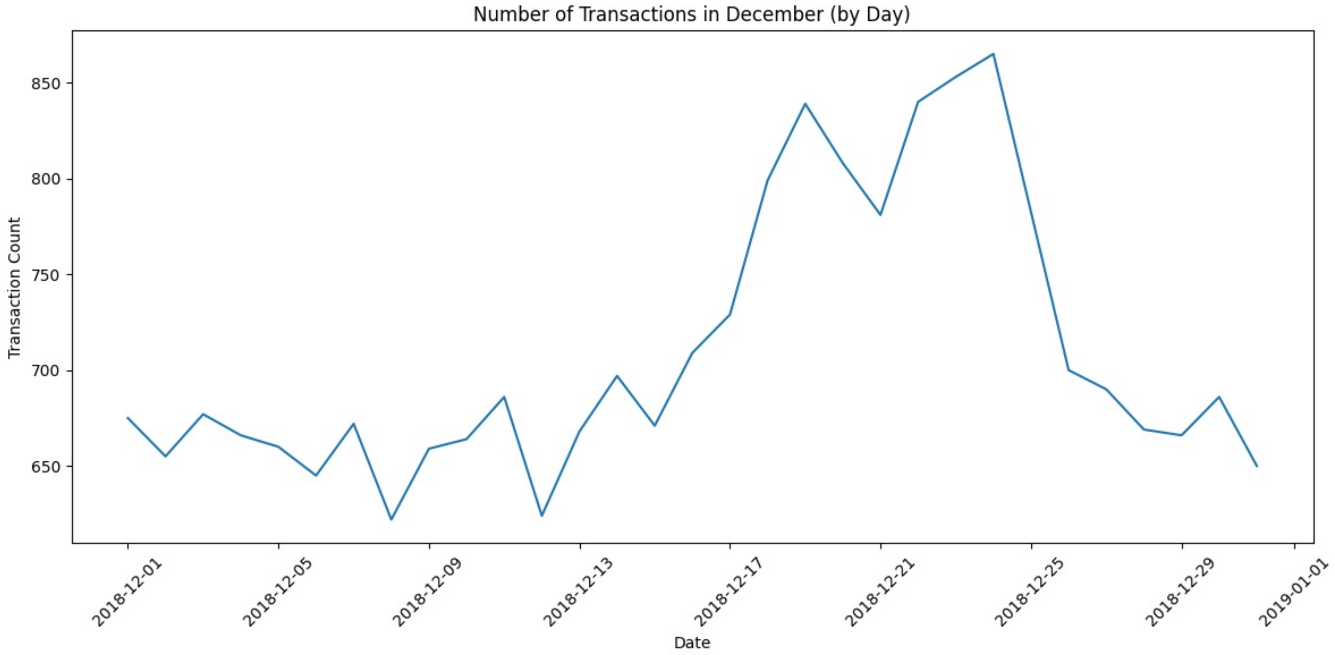
```
december_data = transaction_data[transaction_data['DATE'].dt.month == 12]

december_by_day = december_data.groupby('DATE').size().reset_index(name='TRANSACTION_COUNT')

print(december_by_day.head())
```

```
        DATE  TRANSACTION_COUNT
0 2018-12-01                675
1 2018-12-02                655
2 2018-12-03                677
3 2018-12-04                666
4 2018-12-05                660
```

In [92]:

```
plt.figure(figsize=(12, 6))
sns.lineplot(data=december_by_day, x='DATE', y='TRANSACTION_COUNT')
plt.title('Number of Transactions in December (by Day)')
plt.xlabel('Date')
plt.ylabel('Transaction Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprec
ated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprec
ated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

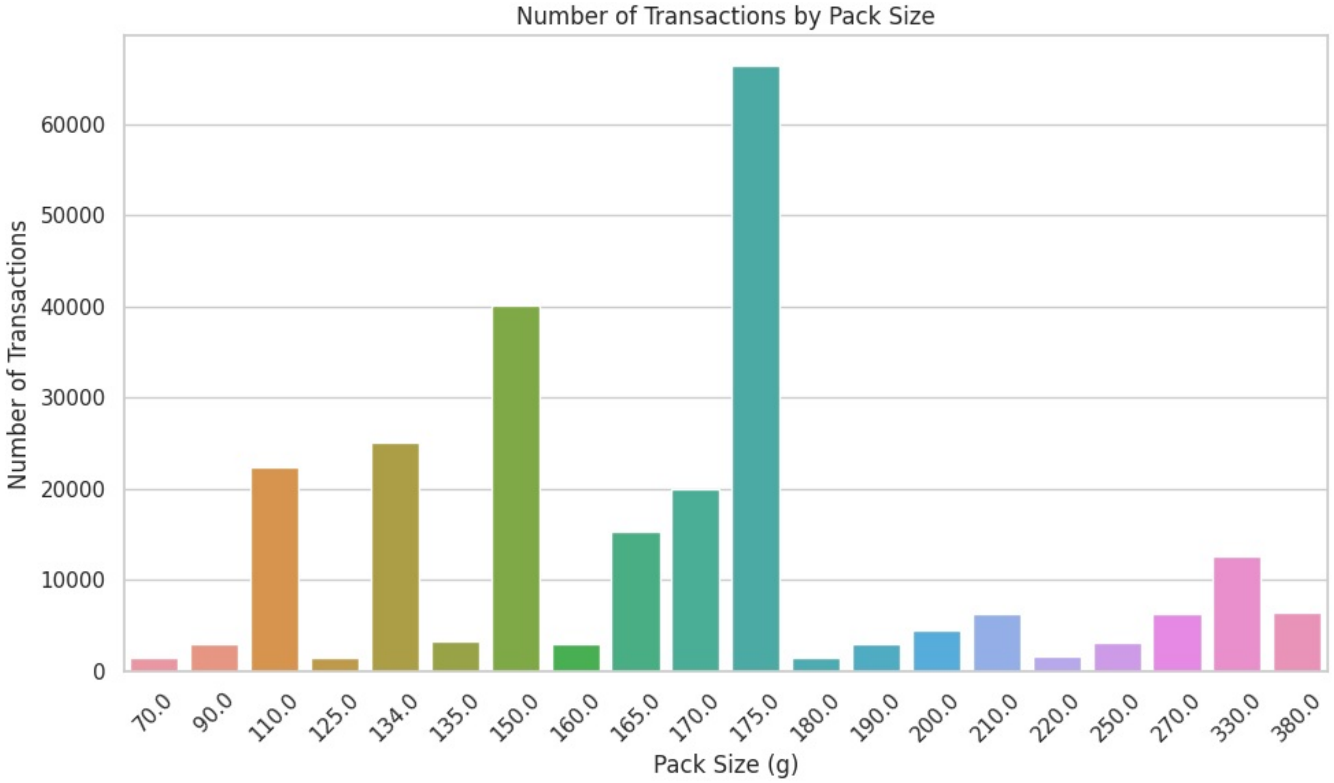


From this it is clear that the increase in the sale in lead-up to christmas and that too zero sales on the christmas day itself. This is mainly due to shops being closed on christmas day.

In [94]:
```
transaction_data['PACK_SIZE'] = transaction_data['PROD_NAME'].str.extract(r'(\d+)\s*[gG]')[0].astype(float)

pack_size_summary = transaction_data['PACK_SIZE'].value_counts().sort_index()
print(pack_size_summary)
```

```
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 [95]:
```
sns.set(style="whitegrid")

plt.figure(figsize=(10, 6))
sns.countplot(data=transaction_data, x='PACK_SIZE', order=sorted(transaction_data['PACK_SIZE'].dropna().unique(
plt.title('Number of Transactions by Pack Size')
plt.xlabel('Pack Size (g)')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [96]:
```
transaction_data['BRAND'] = transaction_data['PROD_NAME'].str.split().str[0]
```

In [97]:
```
transaction_data['BRAND'] = transaction_data['BRAND'].replace({
    'RED': 'RRD',
    'RRD': 'RRD',
    'Infzns': 'Infuzions',
    'Infzn': 'Infuzions',
    'Snbts': 'Sunbites',
    'Dorito': 'Doritos',
    'WW': 'Woolworths',
    'Smith': 'Smiths'
    # You can add more replacements here if needed
})
```

In [98]:

```
print(transaction_data[['PROD_NAME', 'BRAND']].head(10))
print("\nBrand counts:\n")
print(transaction_data['BRAND'].value_counts())
```

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

Brand counts:

BRAND
Kettle        41288
Smiths        30353
Doritos       25224
Pringles      25102
Infuzions     14201
Thins         14075
RRD           11894
Woolworths    11836
Cobs           9693
Tostitos       9471
Twisties       9454
Tyrrells       6442
Grain          6272
Natural        6050
Cheezels       4603
CCs            4551
Red            4427
Sunbites       3008
Cheetos        2927
Burger         1564
GrnWves        1468
NCC            1419
French         1418
Name: count, dtype: int64
```

In [102…

```
customer_data.info()

print("LIFESTAGE Distribution:\n")
print(customer_data['LIFESTAGE'].value_counts())

print("\nPREMIUM_CUSTOMER Distribution:\n")
print(customer_data['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
LIFESTAGE Distribution:

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 Distribution:

PREMIUM_CUSTOMER
Mainstream    29245
Budget        24470
Premium       18922
Name: count, dtype: int64
```

In [104…

```
data = transaction_data.merge(customer_data, on='LYLTY_CARD_NBR', how='left')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 246740 entries, 0 to 246739
Data columns (total 14 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   DATE              246740 non-null  datetime64[ns]
 1   STORE_NBR         246740 non-null  int64
 2   LYLTY_CARD_NBR    246740 non-null  int64
 3   TXN_ID            246740 non-null  int64
 4   PROD_NBR          246740 non-null  int64
 5   PROD_NAME         246740 non-null  object
 6   PROD_QTY          246740 non-null  int64
 7   TOT_SALES         246740 non-null  float64
 8   CLEAN_PROD_WORDS  246740 non-null  object
 9   WORD_FREQUENCY    246740 non-null  object
 10  PACK_SIZE         246740 non-null  float64
 11  BRAND             246740 non-null  object
 12  LIFESTAGE         246740 non-null  object
 13  PREMIUM_CUSTOMER  246740 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(5), object(6)
memory usage: 26.4+ MB
```

In [108…

```
print(data.isnull().sum())
print(data[data[['LIFESTAGE', 'PREMIUM_CUSTOMER']].isnull().any(axis=1)])
```

```
DATE                0
STORE_NBR           0
LYLTY_CARD_NBR      0
TXN_ID              0
PROD_NBR            0
PROD_NAME           0
PROD_QTY            0
TOT_SALES           0
CLEAN_PROD_WORDS    0
WORD_FREQUENCY      0
PACK_SIZE           0
BRAND               0
LIFESTAGE           0
PREMIUM_CUSTOMER    0
dtype: int64
Empty DataFrame
Columns: [DATE, STORE_NBR, LYLTY_CARD_NBR, TXN_ID, PROD_NBR, PROD_NAME, PROD_QTY, TOT_SALES, CLEAN_PROD_WORDS,
WORD_FREQUENCY, PACK_SIZE, BRAND, LIFESTAGE, PREMIUM_CUSTOMER]
Index: []
```

In [109…

```
data.to_csv("QVI_data.csv", index=False)
```

- Who spends the most on chips (total sales), describing customers by lifestage and how premium their general purchasing behaviour is
- How many customers are in each segment
- How many chips are bought per customer by segment
- What's the average chip price by customer segment

In [110…

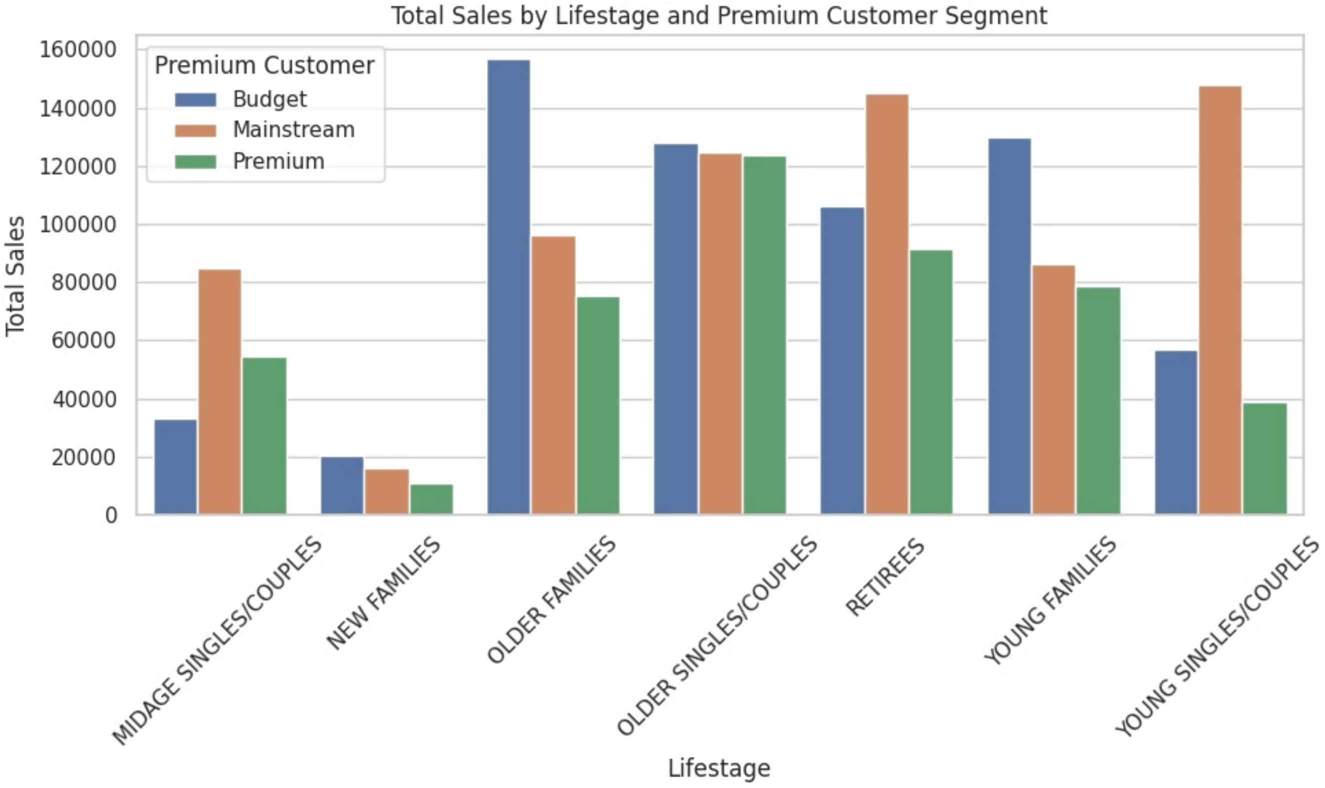
```
total_sales_by_segment = data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['TOT_SALES'].sum().reset_index()

print(total_sales_by_segment)
```

```
                 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 [111…

```
# Plotting total sales by LIFESTAGE and PREMIUM_CUSTOMER
plt.figure(figsize=(10, 6))
sns.barplot(data=total_sales_by_segment, x='LIFESTAGE', y='TOT_SALES', hue='PREMIUM_CUSTOMER')

plt.xlabel('Lifestage')
plt.ylabel('Total Sales')
```

```python
plt.title('Total Sales by Lifestage and Premium Customer Segment')
plt.xticks(rotation=45)
plt.tight_layout()
plt.legend(title='Premium Customer')
plt.show()
```



Sales are coming mainly from Budget - older families, Mainstream - retirees, and Mainstream - young single / couples.

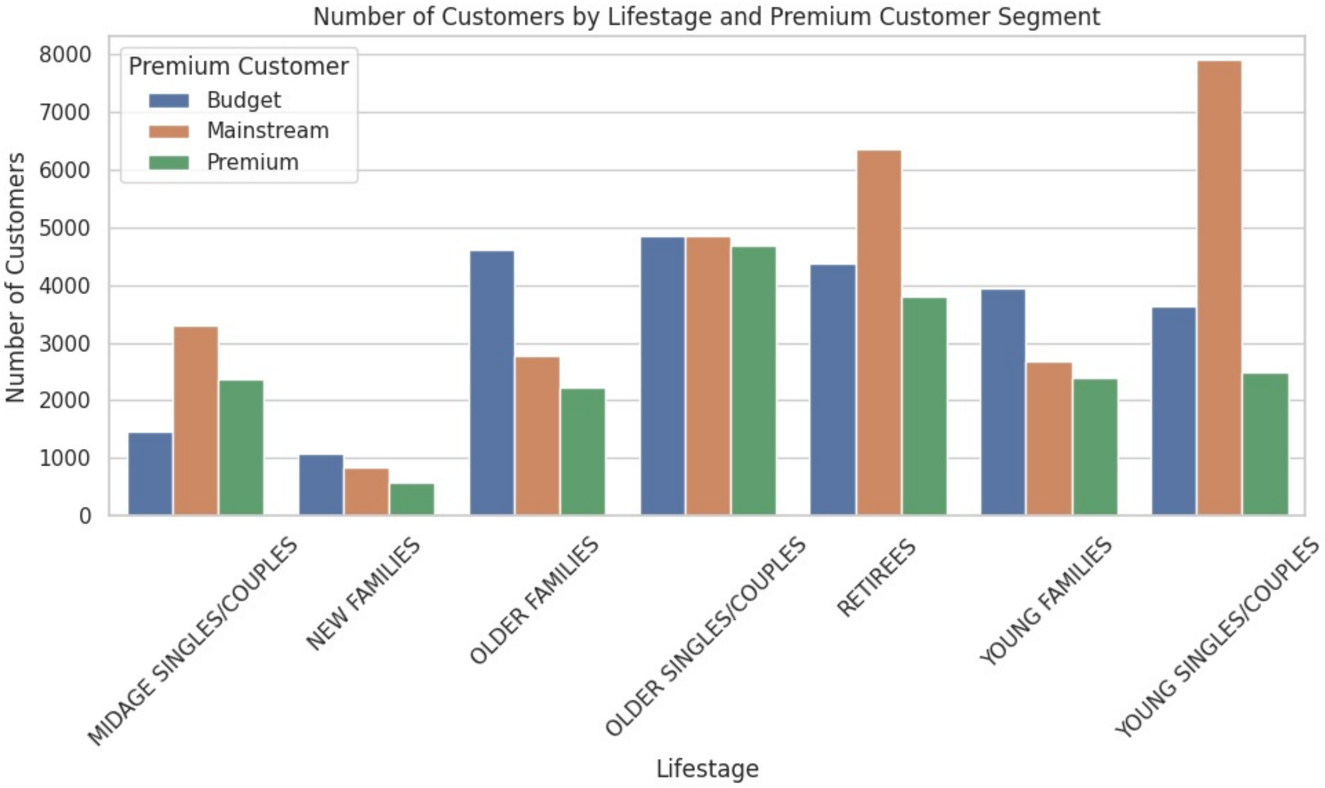
In [112…

```python
num_customers_by_segment = data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['LYLTY_CARD_NBR'].nunique().reset_in

print(num_customers_by_segment)
```

```
                 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 [113…

```python
# Plotting the number of customers by LIFESTAGE and PREMIUM_CUSTOMER
plt.figure(figsize=(10, 6))
sns.barplot(data=num_customers_by_segment, x='LIFESTAGE', y='LYLTY_CARD_NBR', hue='PREMIUM_CUSTOMER')

plt.xlabel('Lifestage')
plt.ylabel('Number of Customers')
plt.title('Number of Customers by Lifestage and Premium Customer Segment')
plt.xticks(rotation=45)
plt.tight_layout()
plt.legend(title='Premium Customer')
plt.show()
```



There are more Mainstream - young singles/couples and Mainstream - retirees who buy chips. This contributes to there being more sales to these customer segments but this is not a major driver for the Budget - Older families segment.

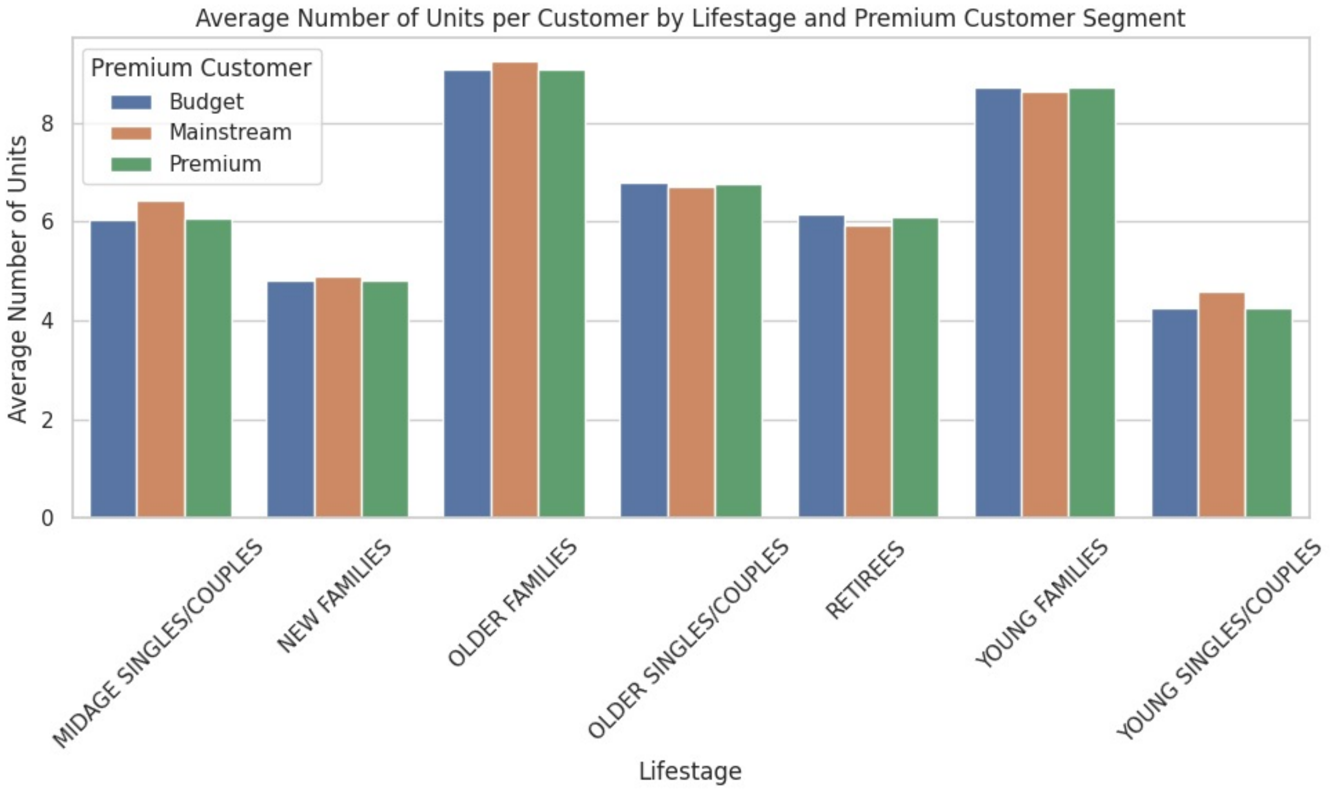
In [114…

```
total_units_per_customer = data.groupby(['LYLTY_CARD_NBR', 'LIFESTAGE', 'PREMIUM_CUSTOMER'])['PROD_QTY'].sum().

avg_units_per_customer = total_units_per_customer.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['PROD_QTY'].mean()

print(avg_units_per_customer)
```

```
                 LIFESTAGE PREMIUM_CUSTOMER  PROD_QTY
0   MIDAGE SINGLES/COUPLES           Budget  6.026459
1   MIDAGE SINGLES/COUPLES       Mainstream  6.432080
2   MIDAGE SINGLES/COUPLES          Premium  6.078514
3             NEW FAMILIES           Budget  4.821527
4             NEW FAMILIES       Mainstream  4.891566
5             NEW FAMILIES          Premium  4.815652
6           OLDER FAMILIES           Budget  9.076773
7           OLDER FAMILIES       Mainstream  9.255380
8           OLDER FAMILIES          Premium  9.071717
9    OLDER SINGLES/COUPLES           Budget  6.781398
10   OLDER SINGLES/COUPLES       Mainstream  6.712021
11   OLDER SINGLES/COUPLES          Premium  6.769543
12                RETIREES           Budget  6.141847
13                RETIREES       Mainstream  5.925920
14                RETIREES          Premium  6.103358
15          YOUNG FAMILIES           Budget  8.722995
16          YOUNG FAMILIES       Mainstream  8.638361
17          YOUNG FAMILIES          Premium  8.716013
18   YOUNG SINGLES/COUPLES           Budget  4.250069
19   YOUNG SINGLES/COUPLES       Mainstream  4.575597
20   YOUNG SINGLES/COUPLES          Premium  4.264113
```

In [115…

```
# Plotting the average number of units per customer by LIFESTAGE and PREMIUM_CUSTOMER
plt.figure(figsize=(10, 6))
sns.barplot(data=avg_units_per_customer, x='LIFESTAGE', y='PROD_QTY', hue='PREMIUM_CUSTOMER')

plt.xlabel('Lifestage')
plt.ylabel('Average Number of Units')
plt.title('Average Number of Units per Customer by Lifestage and Premium Customer Segment')
plt.xticks(rotation=45)
plt.tight_layout()
plt.legend(title='Premium Customer')
plt.show()
```



Older families and young families in general buy more chips per customer.

In [116…

```
data['PRICE_PER_UNIT'] = data['TOT_SALES'] / data['PROD_QTY']

print(data[['LYLTY_CARD_NBR', 'PROD_QTY', 'TOT_SALES', 'PRICE_PER_UNIT']].head())
```

```
   LYLTY_CARD_NBR  PROD_QTY  TOT_SALES  PRICE_PER_UNIT
0            1000         2        6.0            3.00
1            1307         3        6.3            2.10
2            1343         2        2.9            1.45
3            2373         5       15.0            3.00
4            2426         3       13.8            4.60
```

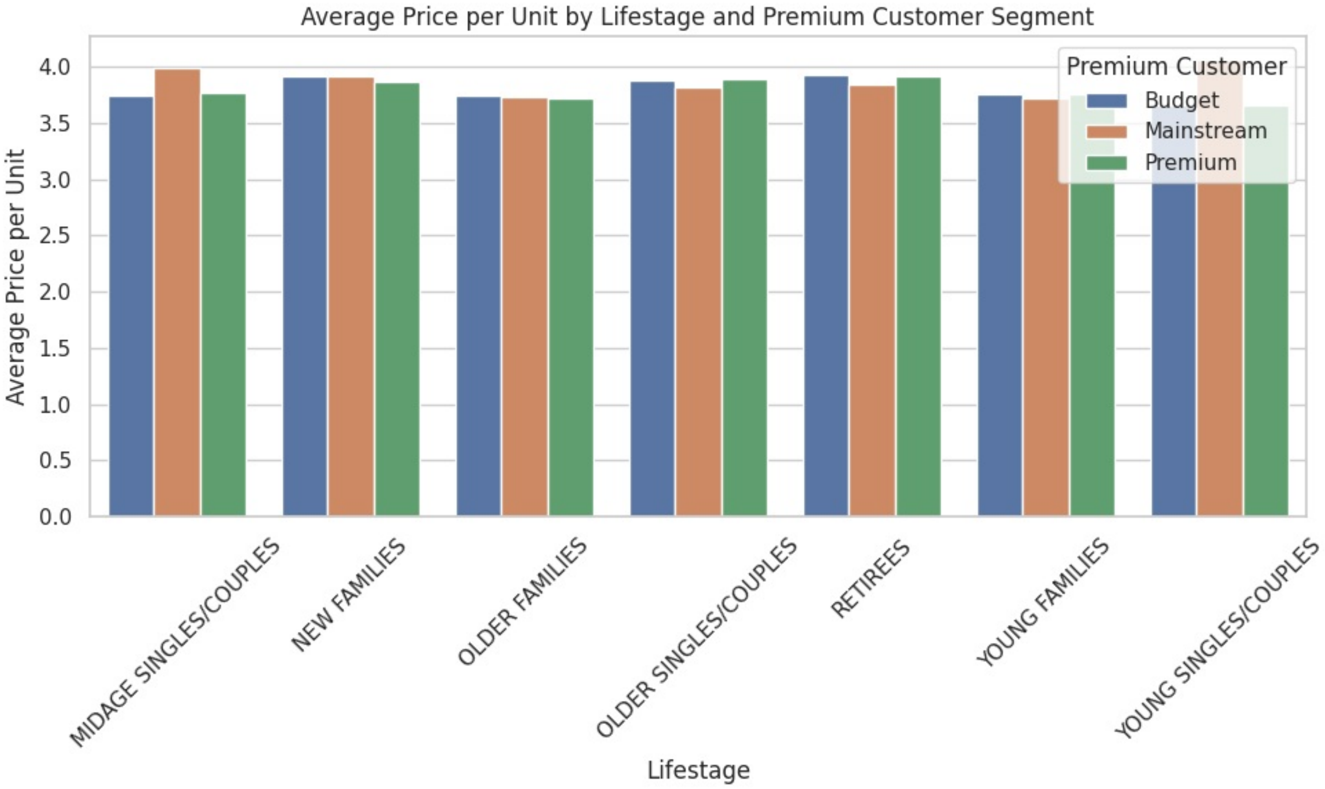
In [117…

```
avg_price_per_unit = data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['PRICE_PER_UNIT'].mean().reset_index()

print(avg_price_per_unit)
```

```
                 LIFESTAGE PREMIUM_CUSTOMER  PRICE_PER_UNIT
0   MIDAGE SINGLES/COUPLES           Budget        3.743328
1   MIDAGE SINGLES/COUPLES       Mainstream        3.994241
2   MIDAGE SINGLES/COUPLES          Premium        3.770698
3             NEW FAMILIES           Budget        3.917688
4             NEW FAMILIES       Mainstream        3.916133
5             NEW FAMILIES          Premium        3.872110
6           OLDER FAMILIES           Budget        3.745340
7           OLDER FAMILIES       Mainstream        3.737077
8           OLDER FAMILIES          Premium        3.717000
9    OLDER SINGLES/COUPLES           Budget        3.882096
10   OLDER SINGLES/COUPLES       Mainstream        3.814665
11   OLDER SINGLES/COUPLES          Premium        3.893182
12                RETIREES           Budget        3.924404
13                RETIREES       Mainstream        3.844294
14                RETIREES          Premium        3.920942
15          YOUNG FAMILIES           Budget        3.760737
16          YOUNG FAMILIES       Mainstream        3.724533
17          YOUNG FAMILIES          Premium        3.762150
18   YOUNG SINGLES/COUPLES           Budget        3.657366
19   YOUNG SINGLES/COUPLES       Mainstream        4.065642
20   YOUNG SINGLES/COUPLES          Premium        3.665414
```

In [118…

```
# Plotting the average price per unit by LIFESTAGE and PREMIUM_CUSTOMER
plt.figure(figsize=(10, 6))
sns.barplot(data=avg_price_per_unit, x='LIFESTAGE', y='PRICE_PER_UNIT', hue='PREMIUM_CUSTOMER')

plt.xlabel('Lifestage')
plt.ylabel('Average Price per Unit')
plt.title('Average Price per Unit by Lifestage and Premium Customer Segment')
plt.xticks(rotation=45)
plt.tight_layout()
plt.legend(title='Premium Customer')
plt.show()
```



Mainstream midage and young singles and couples are more willing to pay more per packet of chips compared to their budget and premium counterparts. This may be due to premium shoppers being more likely to buy healthy snacks and when they buy chips, this is mainly for entertainment purposes rather than their own consumption. This is also supported by there being fewer premium midage and young singles and couples buying chips compared to their mainstream counterparts.

In [119…

```
from scipy.stats import ttest_ind

mainstream_data = data[data['PREMIUM_CUSTOMER'] == 'Mainstream']['PRICE_PER_UNIT']
premium_data = data[data['PREMIUM_CUSTOMER'] == 'Premium']['PRICE_PER_UNIT']

t_stat, p_value = ttest_ind(mainstream_data, premium_data, equal_var=False)  # Welch's t-test

print(f"T-statistic: {t_stat}")
print(f"P-value: {p_value}")
```

```
T-statistic: 11.05723574336515
P-value: 2.078836404116925e-28
```

In [ ]: