Import required libraries and dataframe

In [1]:
```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
```

In [2]:
```
# Load the data
transaction_data = pd.read_excel('resources/QVI_transaction_data.xlsx')
customer_data = pd.read_csv('resources/QVI_purchase_behaviour.csv')
```

Exploratory data analysis

The first step in any analysis is to first understand the data. Let's take a look at each of the datasets provided.

Examine transaction_data

In [3]:
```
# Print data type info of transaction_data columns
transaction_data.info()
```

```
<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
```

We can see that the date column is in an integer format. Let's change this to a date format

In [4]:
```
# Change the DATE column to datetime format
transaction_data['DATE'] = pd.to_datetime(pd.to_numeric(transaction_data['DATE'], errors='coerce'), unit
```

We should check that we are looking at the right products by examining PROD_NAME.

In [5]:
```
print(transaction_data['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
                            ...
264831     Kettle Sweet Chilli And Sour Cream 175g
264832               Tostitos Splash Of  Lime 175g
264833                    Doritos Mexicana    170g
264834     Doritos Corn Chip Mexican Jalapeno 150g
264835               Tostitos Splash Of  Lime 175g
Name: PROD_NAME, Length: 264836, dtype: object
```

As we are only interested in words that will tell us if the product is chips or not, let's remove all words with digits and special characters such as '&' from our set of product words.

In [6]:
```
# Examine the words in PROD_NAME to see if there are any incorrect entries such as products that are not
product_words = pd.Series(' '.join(transaction_data['PROD_NAME'].unique()).split()).value_counts()
product_words
```

```
Out[6]: 175g              26
        Chips             21
        150g              19
        &                 17
        Smiths            16
                          ..
        Fig                1
        Mac                1
        N                  1
        Seasonedchicken    1
        Bolognese          1
        Name: count, Length: 220, dtype: int64
```

```
In [7]: # Remove all words with digits and special characters such as ‘&’ from our set of product words.
        product_words = product_words[~product_words.index.str.contains("\\d|[^\\w&]", regex=True)]

        # Look at the most common words by counting the number of times a word appears and sorting them by this j
        product_words = product_words.sort_values(ascending=False)
        product_words
```

```
Out[7]: Chips         21
        &             17
        Smiths        16
        Crinkle       14
        Cut           14
                      ..
        Snag&Sauce     1
        Whlegrn        1
        Hrb&Spce       1
        Sunbites       1
        Bolognese      1
        Name: count, Length: 185, dtype: int64
```

```
In [8]: # There are salsa products in the dataset but we are only interested in the chips category, so let’s rem
        transaction_data = transaction_data[~transaction_data['PROD_NAME'].str.lower().str.contains('salsa')]
```

```
In [9]: # Summarise the data to check for nulls and possible outliers
        transaction_data.describe()
```

Out[9]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_QTY	TOT_S
count	246742	246742.000000	2.467420e+05	2.467420e+05	246742.000000	246742.000000	246742.0(
mean	2018-12-30 01:19:01.211467520	135.051098	1.355310e+05	1.351311e+05	56.351789	1.908062	7.3:
min	2018-07-01 00:00:00	1.000000	1.000000e+03	1.000000e+00	1.000000	1.000000	1.7(
25%	2018-09-30 00:00:00	70.000000	7.001500e+04	6.756925e+04	26.000000	2.000000	5.8(
50%	2018-12-30 00:00:00	130.000000	1.303670e+05	1.351830e+05	53.000000	2.000000	7.4(
75%	2019-03-31 00:00:00	203.000000	2.030840e+05	2.026538e+05	87.000000	2.000000	8.8(
max	2019-06-30 00:00:00	272.000000	2.373711e+06	2.415841e+06	114.000000	200.000000	650.0(
std	NaN	76.787096	8.071528e+04	7.814772e+04	33.695428	0.659831	3.0

There are no nulls in the columns but product quantity appears to have an outlier which we should investigate further. Let’s investigate further the case where 200 packets of chips are bought in one transaction.

```
In [10]: # Filter the dataset to find the outlier
         outlier_transactions = transaction_data[transaction_data['PROD_QTY'] == 200]
         outlier_transactions
```

Out[10]:

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

In [11]:

```
# See if the customer has had other transactions
outlier_transactions = transaction_data[transaction_data['LYLTY_CARD_NBR'] == 226000]
outlier_transactions
```

Out[11]:

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

It looks like this customer has only had the two transactions over the year and is not an ordinary retail customer. The customer might be buying chips for commercial purposes instead. We'll remove this loyalty card number from further analysis.

In [12]:

```
# Filter transaction_data to remove LYLTY_CARD_NBR 226000
transaction_data = transaction_data[~transaction_data['LYLTY_CARD_NBR'].isin(outlier_transactions['LYLTY_
```

In [13]:

```
# Reexamine transaction_data
transaction_data.describe()
```

Out[13]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_QTY	TOT_S
count	246740	246740.000000	2.467400e+05	2.467400e+05	246740.000000	246740.000000	246740.0
mean	2018-12-30 01:18:58.448569344	135.050361	1.355303e+05	1.351304e+05	56.352213	1.906456	7.3
min	2018-07-01 00:00:00	1.000000	1.000000e+03	1.000000e+00	1.000000	1.000000	1.7
25%	2018-09-30 00:00:00	70.000000	7.001500e+04	6.756875e+04	26.000000	2.000000	5.8
50%	2018-12-30 00:00:00	130.000000	1.303670e+05	1.351815e+05	53.000000	2.000000	7.4
75%	2019-03-31 00:00:00	203.000000	2.030832e+05	2.026522e+05	87.000000	2.000000	8.8
max	2019-06-30 00:00:00	272.000000	2.373711e+06	2.415841e+06	114.000000	5.000000	29.5
std	NaN	76.786971	8.071520e+04	7.814760e+04	33.695235	0.342499	2.4

Now, let's look at the number of transaction lines over time to see if there are any obvious data issues such as missing data.

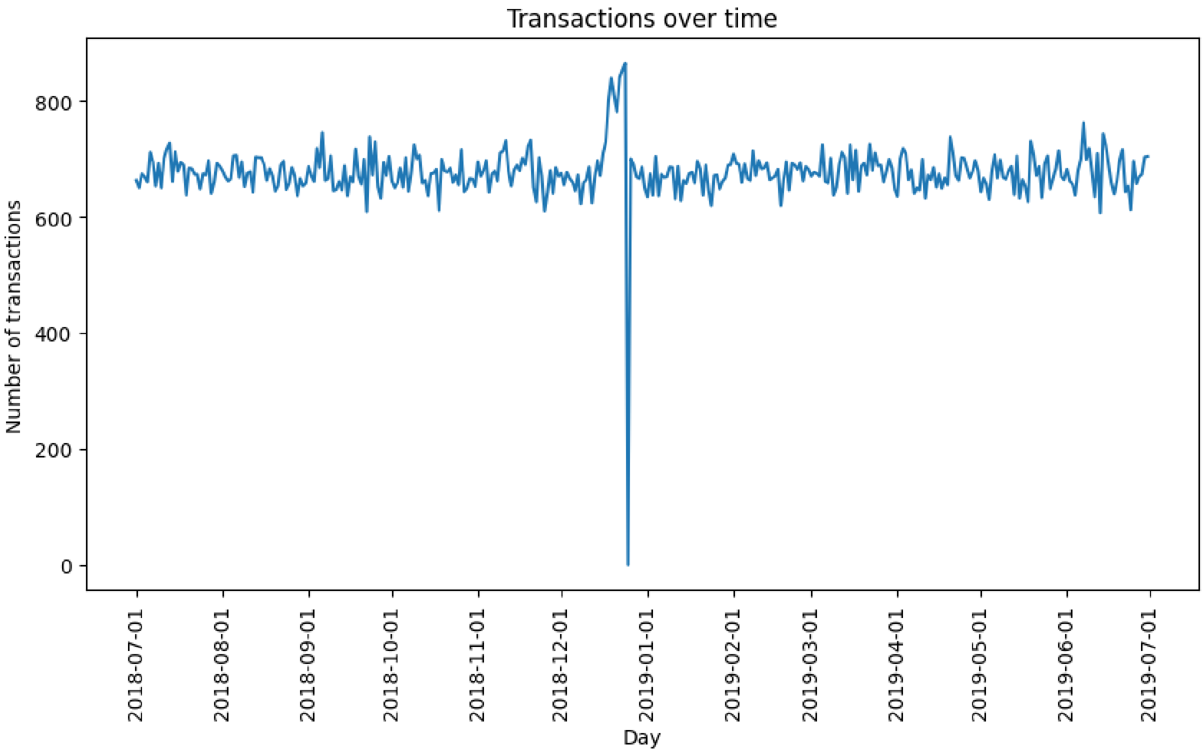
In [14]:

```
# Count the number of transactions by date
transactions_by_date = transaction_data['DATE'].value_counts().sort_index()
transactions_by_date
```

```
Out[14]: DATE
2018-07-01    663
2018-07-02    650
2018-07-03    674
2018-07-04    669
2018-07-05    660
             ...
2019-06-26    657
2019-06-27    669
2019-06-28    673
2019-06-29    703
2019-06-30    704
Name: count, Length: 364, dtype: int64
```

There's only 364 rows, meaning only 364 dates which indicates a missing date. Let's create a sequence of dates from 1 Jul 2018 to 30 Jun 2019 and use this to create a chart of number of transactions over time to find the missing date.

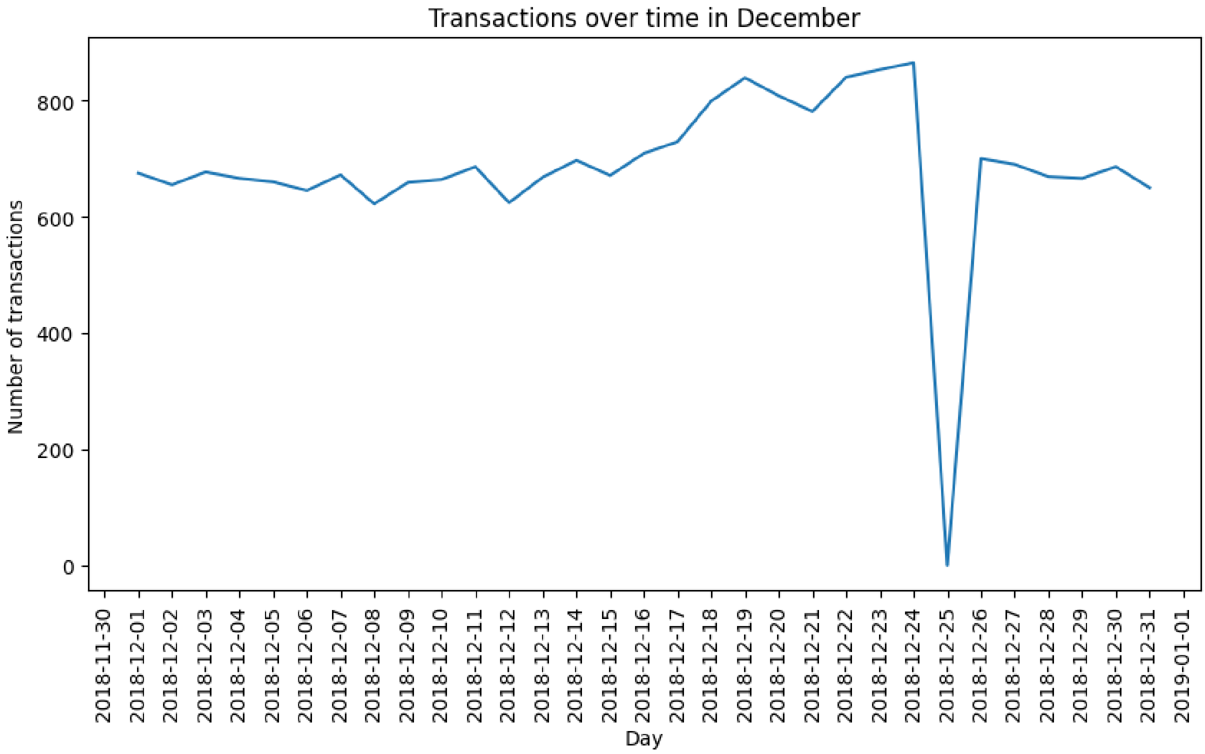
```
In [15]: # Create a sequence of dates from 1 Jul 2018 to 30 Jun 2019
all_dates = pd.date_range(start='2018-07-01', end='2019-06-30')
transactions_by_day = transactions_by_date.reindex(all_dates, fill_value=0)

# Plot transactions over time
plt.figure(figsize=(10,5))
plt.plot(transactions_by_day)
plt.title('Transactions over time')
plt.xlabel('Day')
plt.ylabel('Number of transactions')
plt.xticks(rotation=90)
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
plt.show()
```



We can see that there is an increase in purchases in December and a break in late December. Let's zoom in on this.

```
In [16]: # Filter to December and look at individual days
december_transactions = transactions_by_day[transactions_by_day.index.month == 12]

# Plot transactions over time in December
plt.figure(figsize=(10,5))
plt.plot(december_transactions)
plt.title('Transactions over time in December')
plt.xlabel('Day')
plt.ylabel('Number of transactions')
plt.xticks(rotation=90)
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
```

```
plt.gca().xaxis.set_major_locator(mdates.DayLocator())
plt.show()
```



We can see that the increase in sales occurs in the lead-up to Christmas and that there are zero sales on Christmas day itself. This is due to shops being closed on Christmas day. Now that we are satisfied that the data no longer has outliers, we can move on to creating other features such as brand of chips or pack size from PROD_NAME. We will start with pack size.

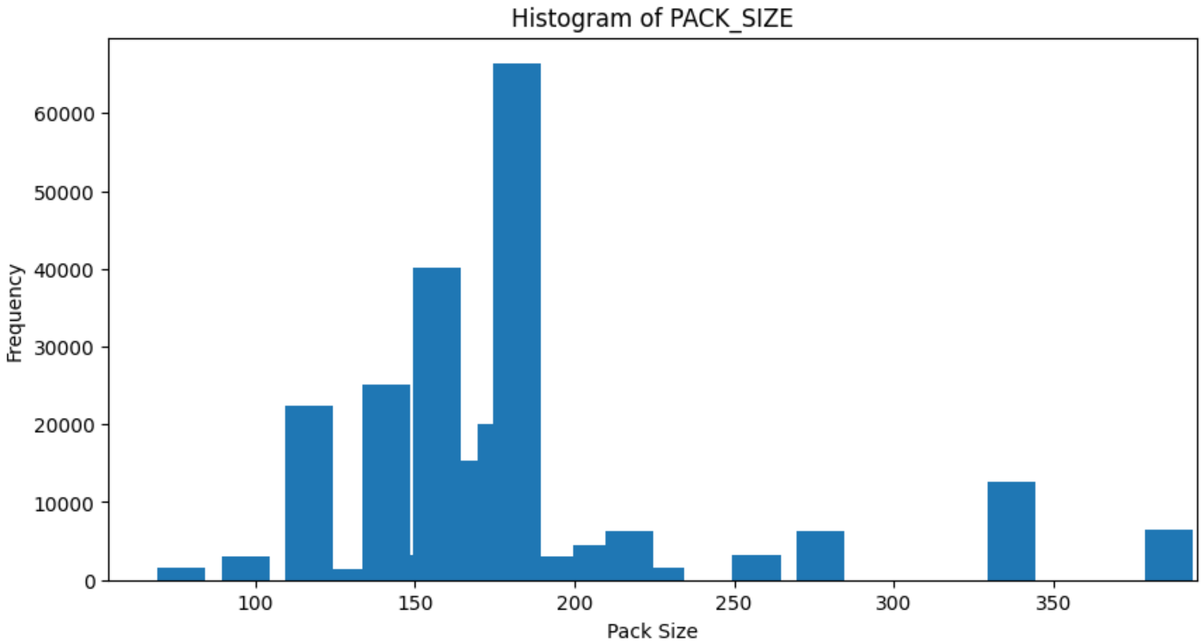
In [17]:
```
# We can work this out by taking the digits that are in PROD_NAME
transaction_data['PACK_SIZE'] = transaction_data['PROD_NAME'].str.extract(r'(\d+)').astype(int)

# Let's check if the pack sizes look sensible
pack_sizes = transaction_data['PACK_SIZE'].value_counts().sort_index()
print(pack_sizes)
```

```
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
```

The largest size is 380g and the smallest size is 70g - seems sensible! Let's plot a histogram of PACK_SIZE since we know that it is a categorical variable and not a continuous variable even though it is numeric.

In [18]:
```
# Plot a histogram of PACK_SIZE
plt.figure(figsize=(10,5))
plt.hist(transaction_data['PACK_SIZE'], bins=np.arange(transaction_data['PACK_SIZE'].min(), transaction_
```

```
plt.title('Histogram of PACK_SIZE')
plt.xlabel('Pack Size')
plt.ylabel('Frequency')
plt.show()
```



Pack sizes created look reasonable and now to create brands, we can use the first word in PROD_NAME to work out the brand name

In [19]:
```
# We can use the first word in PROD_NAME to work out the brand name
transaction_data['BRAND'] = transaction_data['PROD_NAME'].apply(lambda x: x.split(' ')[0].upper())

# Checking brands
brands = transaction_data['BRAND'].value_counts()
print(brands)
```

```
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 [20]:
```
# Some of the brand names look like they are of the same brands - such as RED and RRD, which are both Re
brand_name_corrections = {
    "RED": "RRD",
    "SNBTS": "SUNBITES",
```

```
    "INFZNS": "INFUZIONS",
    "WW": "WOOLWORTHS",
    "SMITH": "SMITHS",
    "NCC": "NATURAL",
    "DORITO": "DORITOS",
    "GRAIN": "GRNWVES"
}

# Replace brand names
transaction_data['BRAND'] = transaction_data['BRAND'].replace(brand_name_corrections)

# Check again
brands = transaction_data['BRAND'].value_counts()
print(brands)
```

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

Examining customer data

In [21]:
```
# Examining customer_data
customer_data.info()
```

```
<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
```

In [22]:
```
# Examining customer_data
customer_data.describe()
```

Out[22]:

	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

Let's have a closer look at the LIFESTAGE and PREMIUM_CUSTOMER columns

In [23]:
```
# Examining the values of lifestage and premium_customer
lifestage_values = customer_data['LIFESTAGE'].value_counts()
print(lifestage_values)
```

```
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
```

As there do not seem to be any issues with the customer data, we can now go ahead and join the transaction and customer data sets together

In [24]:
```
# Merge transaction data to customer data
data = pd.merge(transaction_data, customer_data, how='left', on='LYLTY_CARD_NBR')
```

As the number of rows in data is the same as that of transactionData, we can be sure that no duplicates were created. This is because we created data by setting all.x = TRUE (in other words, a left join) which means take all the rows in transactionData and find rows with matching values in shared columns and then joining the details in these rows to the x or the first mentioned table.

Let's also check if some customers were not matched on by checking for nulls.

In [25]:
```
data.isnull().any()
```

Out[25]:
```
DATE                False
STORE_NBR           False
LYLTY_CARD_NBR      False
TXN_ID              False
PROD_NBR            False
PROD_NAME           False
PROD_QTY            False
TOT_SALES           False
PACK_SIZE           False
BRAND               False
LIFESTAGE           False
PREMIUM_CUSTOMER    False
dtype: bool
```

Great, there are no nulls! So all our customers in the transaction data has been accounted for in the customer dataset.

In [26]:
```
# Export merged data to csv
data.to_csv('QVI_data.csv', index = False)
```

Data exploration is now complete!

Data analysis on customer segments

Now that the data is ready for analysis, we can define some metrics of interest to the client:

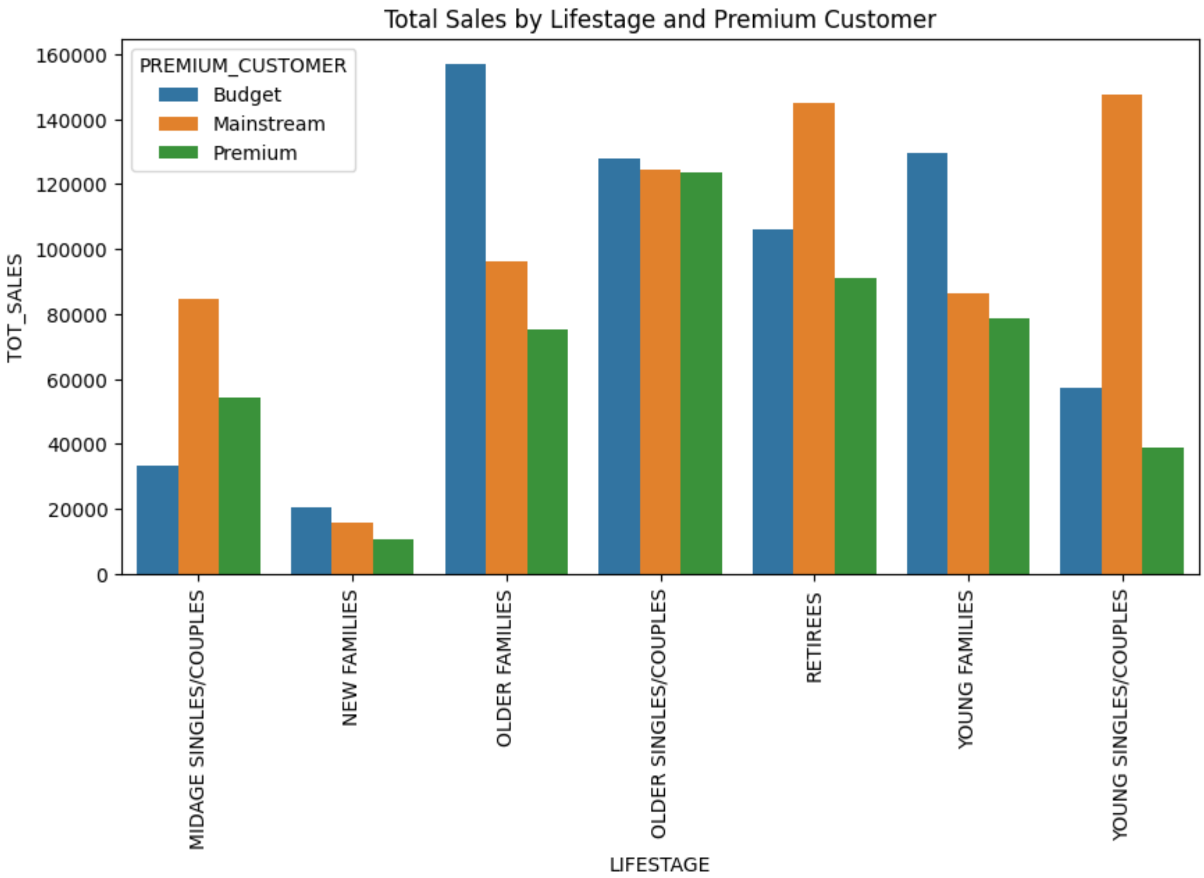
- Who spends the most on chips (total sales), describing customers by lifestage and how premium their general purchasing behaviour is
- How many customers are in each segment
- How many chips are bought per customer by segment
- What's the average chip price by customer segment

We could also ask our data team for more information. Examples are:

- The customer's total spend over the period and total spend for each transaction to understand what proportion of their grocery spend is on chips
- Proportion of customers in each customer segment overall to compare against the mix of customers who purchase chips

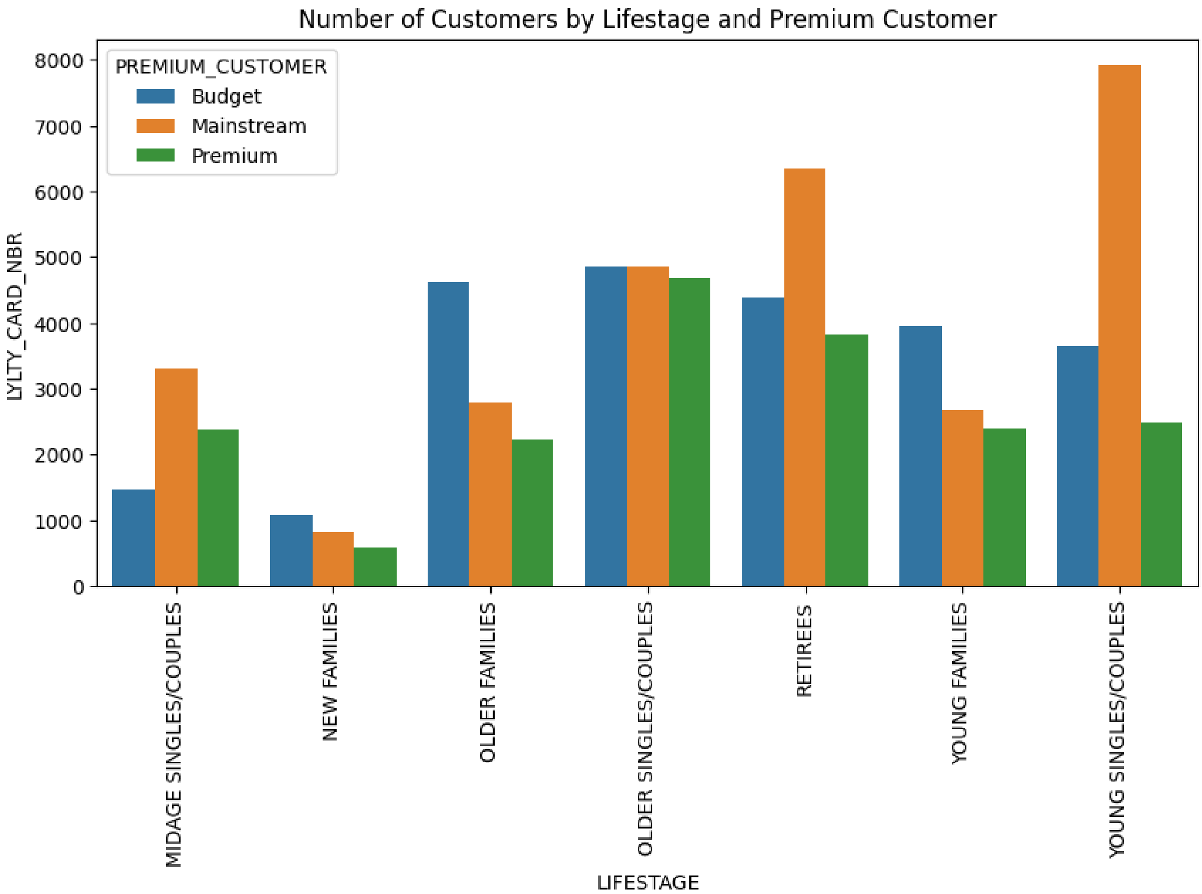
Let's start with calculating total sales by LIFESTAGE and PREMIUM_CUSTOMER and plotting the split by these segments to describe which customer segment contribute most to chip sales.

In [27]:
```
# Total sales by LIFESTAGE and PREMIUM_CUSTOMER
sales = data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['TOT_SALES'].sum().reset_index()

# Create plot
plt.figure(figsize=(10,5))
sns.barplot(x='LIFESTAGE', y='TOT_SALES', hue='PREMIUM_CUSTOMER', data=sales)
plt.title('Total Sales by Lifestage and Premium Customer')
plt.xticks(rotation=90)
plt.show()
```



Sales are coming mainly from Budget - older families, Mainstream - young singles/couples, and Mainstream retirees. Let's see if the higher sales are due to there being more customers who buy chips.

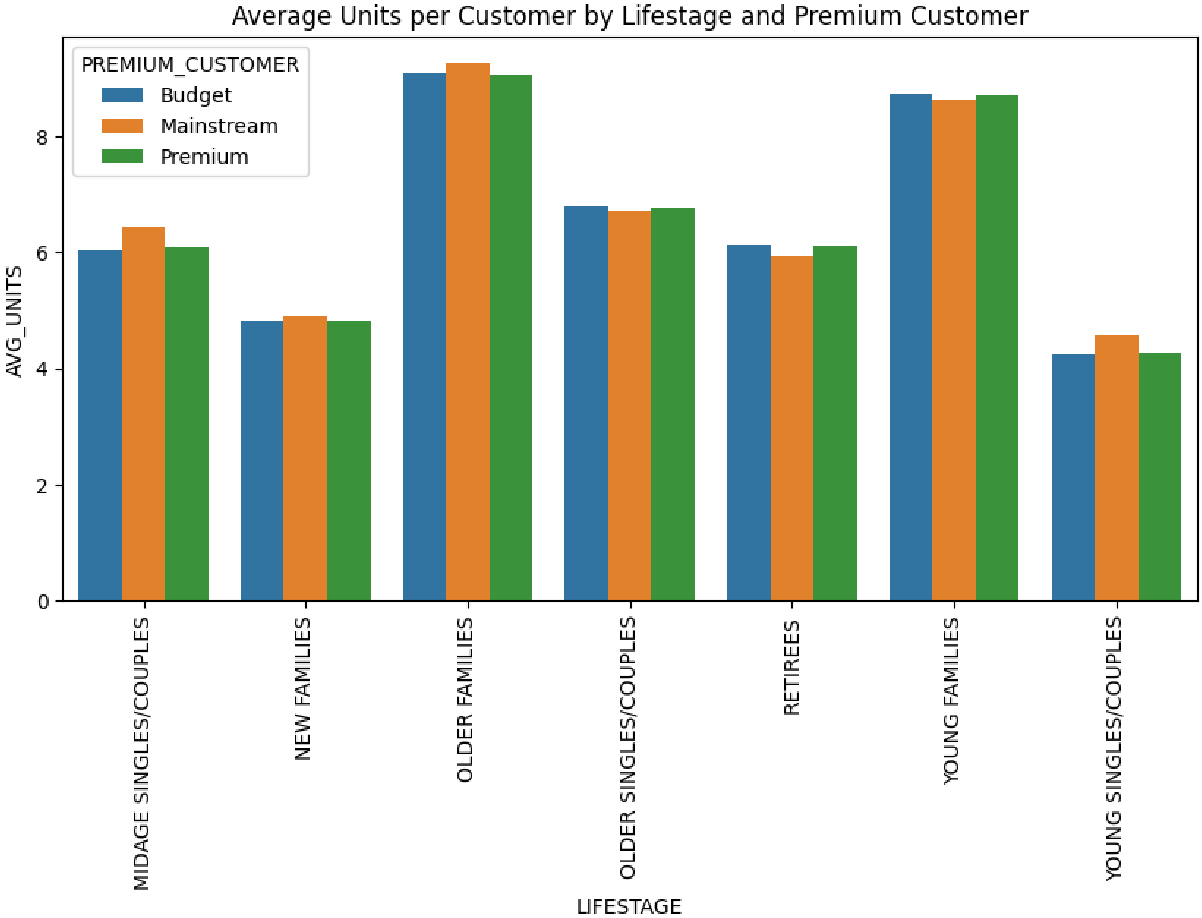
In [28]:
```
# Number of customers by LIFESTAGE and PREMIUM_CUSTOMER
customers = data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['LYLTY_CARD_NBR'].nunique().reset_index()

# Create plot
plt.figure(figsize=(10,5))
sns.barplot(x='LIFESTAGE', y='LYLTY_CARD_NBR', hue='PREMIUM_CUSTOMER', data=customers)
plt.title('Number of Customers by Lifestage and Premium Customer')
plt.xticks(rotation=90)
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. Higher sales may also be driven by more units of chips being bought per customer. Let's have a look at this next.

In [29]:

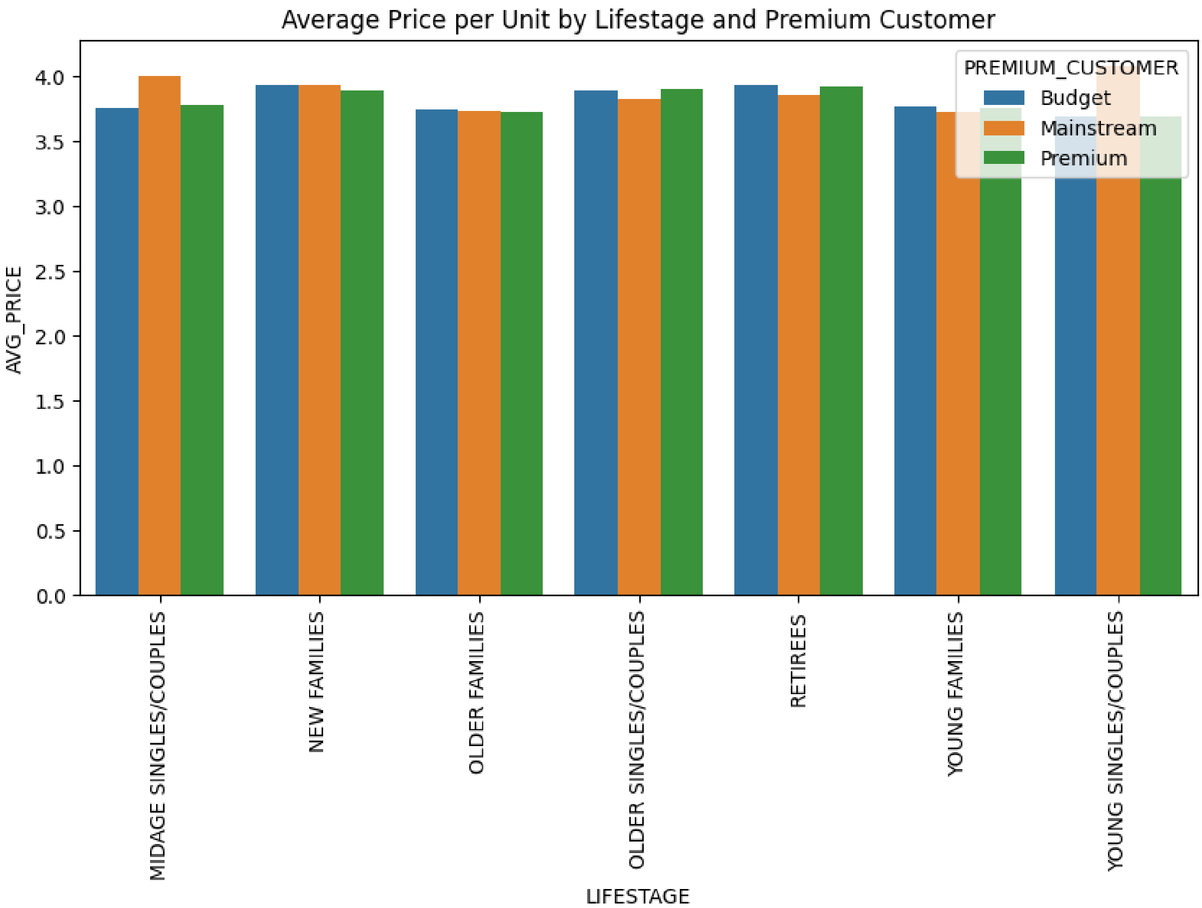
```
# Average number of units per customer by LIFESTAGE and PREMIUM_CUSTOMER
avg_units = data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['PROD_QTY'].sum() / data.groupby(['LIFESTAGE
avg_units = avg_units.reset_index().rename(columns={0: 'AVG_UNITS'})

# Create plot
plt.figure(figsize=(10,5))
sns.barplot(x='LIFESTAGE', y='AVG_UNITS', hue='PREMIUM_CUSTOMER', data=avg_units)
plt.title('Average Units per Customer by Lifestage and Premium Customer')
plt.xticks(rotation=90)
plt.show()
```



Older families and young families in general buy more chips per customer Let's also investigate the average price per unit chips bought for each customer segment as this is also a driver of total sales.

In [30]:

```
# Average price per unit by LIFESTAGE and PREMIUM_CUSTOMER
avg_price = data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['TOT_SALES'].sum() / data.groupby(['LIFESTAG
avg_price = avg_price.reset_index().rename(columns={0: 'AVG_PRICE'})

# Create plot
plt.figure(figsize=(10,5))
sns.barplot(x='LIFESTAGE', y='AVG_PRICE', hue='PREMIUM_CUSTOMER', data=avg_price)
plt.title('Average Price per Unit by Lifestage and Premium Customer')
plt.xticks(rotation=90)
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.

As the difference in average price per unit isn't large, we can check if this difference is statistically different.

In [31]:
```
# Perform an independent t-test between mainstream vs premium and budget midage and young singles and co
from scipy.stats import ttest_ind

# Create a new column 'price' which is the price per unit
data['price'] = data['TOT_SALES'] / data['PROD_QTY']

# Filter data for mainstream young singles/couples and midage singles/couples
mainstream = data[(data['LIFESTAGE'].isin(['YOUNG SINGLES/COUPLES', 'MIDAGE SINGLES/COUPLES'])) & (data[

# Filter data for non-mainstream young singles/couples and midage singles/couples
non_mainstream = data[(data['LIFESTAGE'].isin(['YOUNG SINGLES/COUPLES', 'MIDAGE SINGLES/COUPLES'])) & (d

# Perform t-test
t_stat, p_val = ttest_ind(mainstream, non_mainstream, alternative='greater')

print(f'T-statistic: {t_stat}')
print(f'P-value: {p_val}')
```

```
T-statistic: 37.83196107667815
P-value: 1.11782280577468e-309
```

The t-test results in a p-value < 2.2e-16, i.e. the unit price for mainstream, young and mid-age singles and couples are significantly higher than that of budget or premium, young and midage singles and couples.

Deep dive into specific customer segments for insights

We have found quite a few interesting insights that we can dive deeper into.

We might want to target customer segments that contribute the most to sales to retain them or further increase sales. Let's look at Mainstream - young singles/couples. For instance, let's find out if they tend to buy a particular brand of chips.

In [32]:
```
# Deep dive into Mainstream, young singles/couples
segment1 = data[(data['LIFESTAGE'] == 'YOUNG SINGLES/COUPLES') & (data['PREMIUM_CUSTOMER'] == 'Mainstrea
other = data[~((data['LIFESTAGE'] == 'YOUNG SINGLES/COUPLES') & (data['PREMIUM_CUSTOMER'] == 'Mainstream

# Brand affinity compared to the rest of the population
quantity_segment1 = segment1['PROD_QTY'].sum()
quantity_other = other['PROD_QTY'].sum()

quantity_segment1_by_brand = segment1.groupby('BRAND')['PROD_QTY'].sum() / quantity_segment1
quantity_other_by_brand = other.groupby('BRAND')['PROD_QTY'].sum() / quantity_other

brand_proportions = pd.DataFrame({'target_segment': quantity_segment1_by_brand, 'other': quantity_other_
brand_proportions['affinity_to_brand'] = brand_proportions['target_segment'] / brand_proportions['other'

brand_proportions = brand_proportions.sort_values(by='affinity_to_brand', ascending=False)
brand_proportions
```

Out[32]:

BRAND	target_segment	other	affinity_to_brand
TYRRELLS	0.031553	0.025692	1.228095
TWISTIES	0.046184	0.037877	1.219319
DORITOS	0.122761	0.101075	1.214553
KETTLE	0.197985	0.165553	1.195897
TOSTITOS	0.045411	0.037978	1.195713
PRINGLES	0.119420	0.100635	1.186670
COBS	0.044638	0.039049	1.143124
INFUZIONS	0.064679	0.057065	1.133435
THINS	0.060373	0.056986	1.059423
GRNWVES	0.032712	0.031188	1.048873
CHEEZELS	0.017971	0.018647	0.963753
SMITHS	0.096370	0.124584	0.773536
FRENCH	0.003948	0.005758	0.685569
CHEETOS	0.008033	0.012067	0.665733
RRD	0.043810	0.067494	0.649091
NATURAL	0.019600	0.030854	0.635241
CCS	0.011180	0.018896	0.591677
SUNBITES	0.006349	0.012580	0.504698
WOOLWORTHS	0.024099	0.049427	0.487573
BURGER	0.002926	0.006596	0.443597

We can see that:

- Mainstream young singles/couples are 23% more likely to purchase Tyrrells chips compared to the rest of the population
- Mainstream young singles/couples are 56% less likely to purchase Burger Rings compared to the rest of the population Let's also find out if our target segment tends to buy larger packs of chips

In [33]:
```
# Preferred pack size compared to the rest of the population
quantity_segment1_by_pack = segment1.groupby('PACK_SIZE')['PROD_QTY'].sum() / quantity_segment1
quantity_other_by_pack = other.groupby('PACK_SIZE')['PROD_QTY'].sum() / quantity_other
```

```
pack_proportions = pd.DataFrame({'target_segment': quantity_segment1_by_pack, 'other': quantity_other_by
pack_proportions['affinity_to_pack'] = pack_proportions['target_segment'] / pack_proportions['other']

pack_proportions = pack_proportions.sort_values(by='affinity_to_pack', ascending=False)
pack_proportions
```

Out[33]:

PACK_SIZE	target_segment	other	affinity_to_pack
270	0.031829	0.025096	1.268287
380	0.032160	0.025584	1.257030
330	0.061284	0.050162	1.221717
134	0.119420	0.100635	1.186670
110	0.106280	0.089791	1.183637
210	0.029124	0.025121	1.159318
135	0.014769	0.013075	1.129511
250	0.014355	0.012781	1.123166
170	0.080773	0.080986	0.997370
150	0.157598	0.163421	0.964372
175	0.254990	0.270007	0.944382
165	0.055652	0.062268	0.893757
190	0.007481	0.012442	0.601271
180	0.003589	0.006067	0.591538
160	0.006404	0.012373	0.517616
90	0.006349	0.012580	0.504698
125	0.003009	0.006037	0.498442
200	0.008972	0.018656	0.480899
70	0.003037	0.006322	0.480292
220	0.002926	0.006596	0.443597

It looks like Mainstream young singles/couples are 27% more likely to purchase a 270g pack of chips compared to the rest of the population but let's dive into what brands sell this pack size.

In [34]:

```
# Get unique product names for pack size 270
unique_prod_names = data[data['PACK_SIZE'] == 270]['PROD_NAME'].unique()
unique_prod_names
```

Out[34]:

```
array(['Twisties Cheese     270g', 'Twisties Chicken270g'], dtype=object)
```

Twisties are the only brand offering 270g packs and so this may instead be reflecting a higher likelihood of purchasing Twisties.

Conclusion

Sales have mainly been due to Budget - older families, Mainstream - young singles/couples, and Mainstream retirees shoppers. We found that the high spend in chips for mainstream young singles/couples and retirees is due to there being more of them than other buyers. Mainstream, midage and young singles and couples are also more likely to pay more per packet of chips. This is indicative of impulse buying behaviour. We've also found that Mainstream young singles and couples are 23% more likely to purchase Tyrrells chips compared to the rest of the population. The Category Manager may want to increase the category's performance by off-locating some Tyrrells and smaller packs of chips in discretionary space near segments where young singles and couples frequent more often to increase visibilty and impulse behaviour.

Quantium can help the Category Manager with recommendations of where these segments are and further help them with measuring the impact of the changed placement. We'll work on measuring the impact of trials in the next task and putting all these together in the third task.