Quantium Virtual Internship - Retail Strategy and Analytics - Task 1

```
# Loading required libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import datetime
import xlrd
import re
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
from sklearn.preprocessing import OneHotEncoder

# Read data files into data frames
customerdata = pd.read_csv('QVI_purchase_behaviour.csv')
transactiondata = pd.read_excel('QVI_transaction_data.xlsx')
```

Exploratory Data Analysis

First, we want to examine the data and make sure that it is in a usable form for our analysis.

```
# Examining the transaction data - view a summary of the table
trans_df = transactiondata.copy() # Keep a copy for a quick reset
trans_df
```

```
         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
...       ...        ...             ...     ...       ...
264831  43533        272          272319  270088        89
264832  43325        272          272358  270154        74
264833  43410        272          272379  270187        51
264834  43461        272          272379  270188        42
264835  43365        272          272380  270189        74

                                       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
...                                      ...       ...        ...
264831   Kettle Sweet Chilli And Sour Cream 175g        2       10.8
264832             Tostitos Splash Of  Lime 175g        1        4.4
264833                  Doritos Mexicana    170g        2        8.8
264834   Doritos Corn Chip Mexican Jalapeno 150g        2        7.8
264835             Tostitos Splash Of  Lime 175g        2        8.8

[264836 rows x 8 columns]
```

We can see that the date is in an integer format; change to DD/MM/YYYY format.

```
# Change date from xls integer dates to date format in customer data
trans_df['DATE'] = pd.to_datetime(trans_df['DATE'], unit='D',
origin='1899-12-30')
print(trans_df['DATE'].dtype) # check format of replacement date
column

datetime64[ns]
```

Then we want to ensure that we are only examining chip purchases.

```
# View all unique entries in the product name column
trans_df['PROD_NAME'].unique()

array(['Natural Chip        Compny SeaSalt175g',
       'CCs Nacho Cheese    175g',
       'Smiths Crinkle Cut  Chips Chicken 170g',
       'Smiths Chip Thinly  S/Cream&Onion 175g',
       'Kettle Tortilla ChpsHny&Jlpno Chili 150g',
       'Old El Paso Salsa   Dip Tomato Mild 300g',
       'Smiths Crinkle Chips Salt & Vinegar 330g',
       'Grain Waves         Sweet Chilli 210g',
       'Doritos Corn Chip Mexican Jalapeno 150g',
       'Grain Waves Sour    Cream&Chives 210G',
       'Kettle Sensations   Siracha Lime 150g',
       'Twisties Cheese     270g', 'WW Crinkle Cut      Chicken 175g',
       'Thins Chips Light&  Tangy 175g', 'CCs Original 175g',
       'Burger Rings 220g', 'NCC Sour Cream &    Garden Chives 175g',
       'Doritos Corn Chip Southern Chicken 150g',
```

```
       'Cheezels Cheese Box 125g', 'Smiths Crinkle      Original
330g',
       'Infzns Crn Crnchers Tangy Gcamole 110g',
       'Kettle Sea Salt     And Vinegar 175g',
       'Smiths Chip Thinly  Cut Original 175g', 'Kettle Original
175g',
       'Red Rock Deli Thai  Chilli&Lime 150g',
       'Pringles Sthrn FriedChicken 134g', 'Pringles Sweet&Spcy BBQ
134g',
       'Red Rock Deli SR    Salsa & Mzzrlla 150g',
       'Thins Chips         Originl saltd 175g',
       'Red Rock Deli Sp    Salt & Truffle 150G',
       'Smiths Thinly       Swt Chli&S/Cream175G', 'Kettle Chilli
175g',
       'Doritos Mexicana    170g',
       'Smiths Crinkle Cut  French OnionDip 150g',
       'Natural ChipCo      Hony Soy Chckn175g',
       'Dorito Corn Chp     Supreme 380g', 'Twisties Chicken270g',
       'Smiths Thinly Cut   Roast Chicken 175g',
       'Smiths Crinkle Cut  Tomato Salsa 150g',
       'Kettle Mozzarella   Basil & Pesto 175g',
       'Infuzions Thai SweetChili PotatoMix 110g',
       'Kettle Sensations   Camembert & Fig 150g',
       'Smith Crinkle Cut   Mac N Cheese 150g',
       'Kettle Honey Soy    Chicken 175g',
       'Thins Chips Seasonedchicken 175g',
       'Smiths Crinkle Cut  Salt & Vinegar 170g',
       'Infuzions BBQ Rib   Prawn Crackers 110g',
       'GrnWves Plus Btroot & Chilli Jam 180g',
       'Tyrrells Crisps     Lightly Salted 165g',
       'Kettle Sweet Chilli And Sour Cream 175g',
       'Doritos Salsa       Medium 300g', 'Kettle 135g Swt Pot Sea
Salt',
       'Pringles SourCream  Onion 134g',
       'Doritos Corn Chips  Original 170g',
       'Twisties Cheese     Burger 250g',
       'Old El Paso Salsa   Dip Chnky Tom Ht300g',
       'Cobs Popd Swt/Chlli &Sr/Cream Chips 110g',
       'Woolworths Mild     Salsa 300g',
       'Natural Chip Co     Tmato Hrb&Spce 175g',
       'Smiths Crinkle Cut  Chips Original 170g',
       'Cobs Popd Sea Salt  Chips 110g',
       'Smiths Crinkle Cut  Chips Chs&Onion170g',
       'French Fries Potato Chips 175g',
       'Old El Paso Salsa   Dip Tomato Med 300g',
       'Doritos Corn Chips  Cheese Supreme 170g',
       'Pringles Original   Crisps 134g',
       'RRD Chilli&         Coconut 150g',
       'WW Original Corn    Chips 200g',
```

```
       'Thins Potato Chips  Hot & Spicy 175g',
       'Cobs Popd Sour Crm  &Chives Chips 110g',
       'Smiths Crnkle Chip  Orgnl Big Bag 380g',
       'Doritos Corn Chips  Nacho Cheese 170g',
       'Kettle Sensations   BBQ&Maple 150g',
       'WW D/Style Chip     Sea Salt 200g',
       'Pringles Chicken    Salt Crips 134g',
       'WW Original Stacked Chips 160g',
       'Smiths Chip Thinly  CutSalt/Vinegr175g', 'Cheezels Cheese
330g',
       'Tostitos Lightly    Salted 175g',
       'Thins Chips Salt &  Vinegar 175g',
       'Smiths Crinkle Cut  Chips Barbecue 170g', 'Cheetos Puffs
165g',
       'RRD Sweet Chilli &  Sour Cream 165g',
       'WW Crinkle Cut      Original 175g',
       'Tostitos Splash Of  Lime 175g', 'Woolworths Medium   Salsa
300g',
       'Kettle Tortilla ChpsBtroot&Ricotta 150g',
       'CCs Tasty Cheese    175g', 'Woolworths Cheese   Rings 190g',
       'Tostitos Smoked     Chipotle 175g', 'Pringles Barbeque
134g',
       'WW Supreme Cheese   Corn Chips 200g',
       'Pringles Mystery    Flavour 134g',
       'Tyrrells Crisps     Ched & Chives 165g',
       'Snbts Whlgrn Crisps Cheddr&Mstrd 90g',
       'Cheetos Chs & Bacon Balls 190g', 'Pringles Slt Vingar 134g',
       'Infuzions SourCream&Herbs Veg Strws 110g',
       'Kettle Tortilla ChpsFeta&Garlic 150g',
       'Infuzions Mango     Chutny Papadums 70g',
       'RRD Steak &         Chimuchurri 150g',
       'RRD Honey Soy       Chicken 165g',
       'Sunbites Whlegrn    Crisps Frch/Onin 90g',
       'RRD Salt & Vinegar  165g', 'Doritos Cheese      Supreme 330g',
       'Smiths Crinkle Cut  Snag&Sauce 150g',
       'WW Sour Cream &OnionStacked Chips 160g',
       'RRD Lime & Pepper   165g',
       'Natural ChipCo Sea  Salt & Vinegr 175g',
       'Red Rock Deli Chikn&Garlic Aioli 150g',
       'RRD SR Slow Rst     Pork Belly 150g', 'RRD Pc Sea Salt
165g',
       'Smith Crinkle Cut   Bolognese 150g', 'Doritos Salsa Mild
300g'],
      dtype=object)
```

While it looks like we have chips, we want to check that the products are only chips by counting the word frequencies in the product names. To make this process clearer, we can remove the digits and symbols from the names.

```python
# Remove digits from the product names
prod_name = trans_df['PROD_NAME'].str.replace(r'[0-9]+[gG]','');

# Remove & characters from the product names and replace with a space
to separate flavours
prod_name = prod_name.str.replace(r'&',' ');

# Count the frequencies of words in product names and display counts
in descending order
word_counts = pd.Series(' '.join(prod_name).split()).value_counts()

with pd.option_context('display.max_rows', None): # show all rows
  display(word_counts)
```

Chips	49770
Kettle	41288
Smiths	28860
Salt	27976
Cheese	27890
Pringles	25102
Doritos	24962
Crinkle	23960
Corn	22063
Original	21560
Cut	20754
Chip	18645
Chicken	18577
Salsa	18094
Chilli	15390
Sea	14145
Thins	14075
Sour	13882
Crisps	12607
Vinegar	12402
RRD	11894
Sweet	11060
Infuzions	11057
Supreme	10963
Chives	10951
Cream	10723
WW	10320
Cobs	9693
Popd	9693
Tortilla	9580
Tostitos	9471
Twisties	9454
BBQ	9434
Sensations	9429
Lime	9347
Dip	9324

Paso	9324
Old	9324
El	9324
Tomato	7669
Thinly	7507
Tyrrells	6442
And	6373
Tangy	6332
SourCream	6296
Waves	6272
Grain	6272
Salted	6248
Lightly	6248
Soy	6121
Onion	6116
Natural	6050
Mild	6048
Rock	5885
Red	5885
Deli	5885
Thai	4737
Burger	4733
Swt	4718
Honey	4661
Nacho	4658
Potato	4647
Cheezels	4603
Garlic	4572
CCs	4551
Woolworths	4437
Pesto	3304
Mozzarella	3304
Basil	3304
ChpsHny	3296
Jlpno	3296
Chili	3296
Swt/Chlli	3269
Sr/Cream	3269
Ched	3268
Pot	3257
Of	3252
Splash	3252
SweetChili	3242
PotatoMix	3242
Bag	3233
Crnkle	3233
Big	3233
Orgnl	3233
Hot	3229

Spicy	3229
Camembert	3219
Fig	3219
Barbeque	3210
Jalapeno	3204
Mexican	3204
Light	3188
Chp	3185
Dorito	3185
Spcy	3177
Rib	3174
Crackers	3174
Prawn	3174
Southern	3172
Crm	3159
Ricotta	3146
ChpsBtroot	3146
Chipotle	3145
Smoked	3145
Crnchers	3144
Gcamole	3144
Crn	3144
Infzns	3144
ChpsFeta	3138
Herbs	3134
Strws	3134
Veg	3134
Siracha	3127
Chnky	3125
Ht	3125
Tom	3125
Mexicana	3115
Mystery	3114
Seasonedchicken	3114
Med	3114
Flavour	3114
Crips	3104
Vingar	3095
Slt	3095
Sthrn	3083
FriedChicken	3083
Maple	3083
Rings	3080
ChipCo	3010
SR	2984
Smith	2963
Chs	2960
S/Cream	2934
Cheetos	2927

Medium	2879
French	2856
Cheddr	1576
Snbts	1576
Whlgrn	1576
Mstrd	1576
Hrb	1572
Tmato	1572
Co	1572
Spce	1572
Vinegr	1550
Tasty	1539
Slow	1526
Belly	1526
Rst	1526
Pork	1526
Roast	1519
Mac	1512
N	1512
Mango	1507
Papadums	1507
Chutny	1507
Coconut	1506
Sauce	1503
Snag	1503
Truffle	1498
Sp	1498
Barbecue	1489
Stacked	1487
OnionStacked	1483
Balls	1479
Bacon	1479
Pepper	1473
D/Style	1469
SeaSalt	1468
Btroot	1468
Jam	1468
Plus	1468
Compny	1468
GrnWves	1468
Chli	1461
Hony	1460
Chckn	1460
Mzzrlla	1458
Chimuchurri	1455
Steak	1455
Box	1454
Bolognese	1451
Puffs	1448

```
saltd                   1441
Originl                 1441
CutSalt/Vinegr          1440
OnionDip                1438
Aioli                   1434
Chikn                   1434
Frch/Onin               1432
Sunbites                1432
Whlegrn                 1432
Pc                      1431
NCC                     1419
Garden                  1419
Fries                   1418
dtype: int64
```

Some entries in our data are salsas; we want to remove these.

```
# Remove salsas from the dataset
trans_df = trans_df[trans_df['PROD_NAME'].str.contains(r"[Ss]alsa") ==
False]
trans_df.shape # check for a reduction in no of rows

(246742, 8)
```

Now we can create summaries of the data (eg min, max, mean) to see if there are any obvious outliers in the data and if there are any nulls in any of the columns.

```
# Create summaries of the transaction data
trans_df.describe()

           STORE_NBR  LYLTY_CARD_NBR        TXN_ID       PROD_NBR  \
count  246742.000000    2.467420e+05  2.467420e+05  246742.000000
mean      135.051098    1.355310e+05  1.351311e+05      56.351789
std        76.787096    8.071528e+04  7.814772e+04      33.695428
min         1.000000    1.000000e+03  1.000000e+00       1.000000
25%        70.000000    7.001500e+04  6.756925e+04      26.000000
50%       130.000000    1.303670e+05  1.351830e+05      53.000000
75%       203.000000    2.030840e+05  2.026538e+05      87.000000
max       272.000000    2.373711e+06  2.415841e+06     114.000000

            PROD_QTY      TOT_SALES
count  246742.000000  246742.000000
mean        1.908062       7.321322
std         0.659831       3.077828
min         1.000000       1.700000
25%         2.000000       5.800000
50%         2.000000       7.400000
75%         2.000000       8.800000
max       200.000000     650.000000
```

```
# Check if there are any nans in the dataset
trans_df.isnull().values.any()
```

```
False
```

From the summary, there is at least one transaction with 200 packets. Let's investigate this purchase further.

```
# Filter the entries that have 200 packets.
trans_df.loc[trans_df['PROD_QTY'] == 200.0]
```

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

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

The same customer has made these transactions. They could have been for commercial purposes so we can check to see if they made any other purchases.

```
# Filter the entires by the customer
trans_df.loc[trans_df['LYLTY_CARD_NBR'] == 226000]
```

```
             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
```

It looks like this is the only purchase they have made so we will remove these transactions from the dataset.

```
# Remove the transactions
trans_df = trans_df[trans_df['LYLTY_CARD_NBR'] != 226000]
trans_df.shape # check for a reduction of 2 rows (i.e. 246740 rows)
```

```
(246740, 8)
```

```
# Recheck the data summary
trans_df.describe()
```

```
           STORE_NBR  LYLTY_CARD_NBR        TXN_ID       PROD_NBR  \
count  246740.000000    2.467400e+05  2.467400e+05  246740.000000
mean      135.050361    1.355303e+05  1.351304e+05      56.352213
std        76.786971    8.071520e+04  7.814760e+04      33.695235
min         1.000000    1.000000e+03  1.000000e+00       1.000000
```

```
25%        70.000000    7.001500e+04  6.756875e+04      26.000000
50%       130.000000    1.303670e+05  1.351815e+05      53.000000
75%       203.000000    2.030832e+05  2.026522e+05      87.000000
max       272.000000    2.373711e+06  2.415841e+06     114.000000

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

The summaries now look reasonable. Now look at the number of transaction lines over time to see if there are any obvious data issues such as missing data from particular days.

```
# Count transactions by date to see if there are any missing days
count =
trans_df.groupby(trans_df['DATE'].dt.date).size().reset_index(name =
'COUNT')
count.shape

(364, 2)

# There is one day of data missing. First check the range of dates by
sorting in time order.
trans_df.sort_values(by='DATE')

             DATE  STORE_NBR  LYLTY_CARD_NBR  TXN_ID  PROD_NBR  \
9161   2018-07-01         88           88140   86914        25
155442 2018-07-01         60           60276   57330         3
181349 2018-07-01        199          199014  197623       104
229948 2018-07-01         35           35052   31630        11
104647 2018-07-01         72           72104   71038        20
...           ...        ...             ...     ...       ...
10254  2019-06-30        112          112141  114611        98
113220 2019-06-30        207          207155  205513        99
229182 2019-06-30         10           10140    9882        12
229015 2019-06-30          6            6258    6047        29
262768 2019-06-30        183          183196  185975        22

                                       PROD_NAME  PROD_QTY  TOT_SALES

9161                 Pringles SourCream  Onion 134g         2        7.4

155442  Kettle Sensations   Camembert & Fig 150g         2        9.2

181349  Infuzions Thai SweetChili PotatoMix 110g         2        7.6
```

```
229948                      RRD Pc Sea Salt     165g          1         3.0
104647             Doritos Cheese      Supreme 330g          2        11.4
...                                             ...         ...         ...
10254       NCC Sour Cream &    Garden Chives 175g          2         6.0
113220             Pringles Sthrn FriedChicken 134g          2         7.4
229182    Natural Chip Co      Tmato Hrb&Spce 175g          2         6.0
229015               French Fries Potato Chips 175g          1         3.0
262768     Thins Chips          Originl saltd 175g          2         6.6

[246740 rows x 8 columns]
```

We can see that the dates range from 1 Jul 2018 to 30 Jun 2019. Now we want to check through the year of dates to see which day the data is missing.

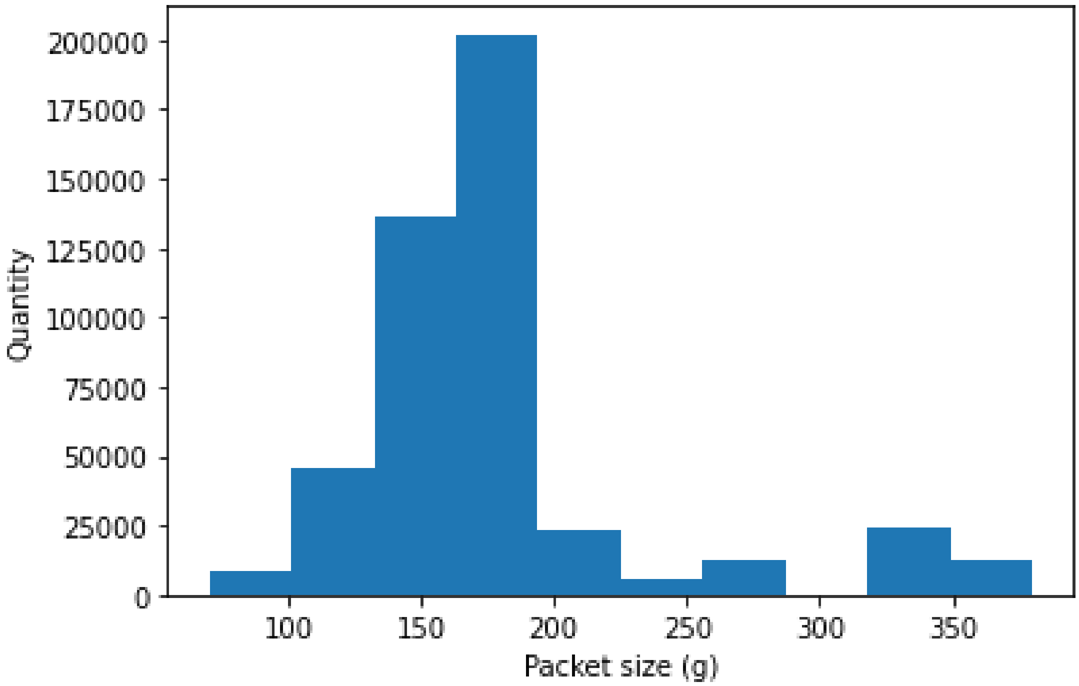
```
# Generate a list of dates with transactions in ascending order
date_counts = trans_df.groupby('DATE').size()

# Then compare to a full list of dates within the same range to find
differences between them
pd.date_range(start = '2018-07-01', end = '2019-06-
30' ).difference(date_counts.index)

DatetimeIndex(['2018-12-25'], dtype='datetime64[ns]', freq=None)
```

The missing date is Christmas day, a public holiday, so it is expected that there are no sales on this day. Now we move onto creating other features such as the pack size, and checking this for any outliers.

```
# Add a new column to data with packet sizes and extract sizes from
product name column
trans_df.insert(8, "PACK_SIZE", trans_df['PROD_NAME'].str.extract('(\
d+)').astype(float), True)

# Sort by packet sizes to check for outliers
trans_df.sort_values(by='PACK_SIZE')

             DATE  STORE_NBR  LYLTY_CARD_NBR  TXN_ID  PROD_NBR  \
40783  2018-09-25         97           97067   96696        38
42461  2019-05-05        110          110030  111890        38
176183 2018-12-30         82           82183   81660        38
227309 2018-12-03        236          236091  239098        38
```

```
42418  2018-11-05        109          109217  111470        38
...           ...        ...             ...     ...       ...
192034 2019-03-12        100          100121   99145         4
255797 2019-01-19        235          235098  238018         4
233814 2019-01-24        151          151102  149810         4
131573 2018-07-09        213          213087  212416         4
102409 2019-05-08         43           43184   39874         4

                                      PROD_NAME  PROD_QTY 
TOT_SALES  \
40783   Infuzions Mango     Chutny Papadums 70g         2        4.8

42461   Infuzions Mango     Chutny Papadums 70g         2        4.8

176183  Infuzions Mango     Chutny Papadums 70g         2        4.8

227309  Infuzions Mango     Chutny Papadums 70g         2        4.8

42418   Infuzions Mango     Chutny Papadums 70g         2        4.8

...                                         ...       ...        ...

192034         Dorito Corn Chp     Supreme 380g         2       13.0

255797         Dorito Corn Chp     Supreme 380g         2       13.0

233814         Dorito Corn Chp     Supreme 380g         1        6.5

131573         Dorito Corn Chp     Supreme 380g         2       13.0

102409         Dorito Corn Chp     Supreme 380g         2       13.0


        PACK_SIZE
40783        70.0
42461        70.0
176183       70.0
227309       70.0
42418        70.0
...           ...
192034      380.0
255797      380.0
233814      380.0
131573      380.0
102409      380.0

[246740 rows x 9 columns]

# Minimum packet size is 70g while max is 380g - this is reasonable.
# Plot a histogram to visualise distribution of pack sizes.
plt.hist(trans_df['PACK_SIZE'], weights=trans_df['PROD_QTY']);
```

```
plt.xlabel('Packet size (g)');
plt.ylabel('Quantity');
```



Now that the pack size looks reasonable, we can create the brand names using the first word of each product name.

```
# Add a column to extract the first word of each product name to.
trans_df.insert(9,
"BRAND_NAME",trans_df['PROD_NAME'].str.split().str.get(0), True)
trans_df
```

```
              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
...            ...        ...             ...     ...       ...
264831 2019-03-09        272          272319  270088        89
264832 2018-08-13        272          272358  270154        74
264833 2018-11-06        272          272379  270187        51
264834 2018-12-27        272          272379  270188        42
264835 2018-09-22        272          272380  270189        74

                                       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
...                                            ...         ...        ...
264831     Kettle Sweet Chilli And Sour Cream 175g          2        10.8
264832               Tostitos Splash Of  Lime 175g          1         4.4
264833                    Doritos Mexicana    170g          2         8.8
264834     Doritos Corn Chip Mexican Jalapeno 150g          2         7.8
264835               Tostitos Splash Of  Lime 175g          2         8.8

        PACK_SIZE BRAND_NAME
0           175.0    Natural
1           175.0        CCs
2           170.0     Smiths
3           175.0     Smiths
4           150.0     Kettle
...           ...        ...
264831      175.0     Kettle
264832      175.0   Tostitos
264833      170.0    Doritos
264834      150.0    Doritos
264835      175.0   Tostitos

[246740 rows x 10 columns]
```

```
# Then print all unique entries to check the brand names created
trans_df["BRAND_NAME"].unique()
```

```
array(['Natural', 'CCs', 'Smiths', 'Kettle', 'Grain', 'Doritos',
       'Twisties', 'WW', 'Thins', 'Burger', 'NCC', 'Cheezels',
'Infzns',
       'Red', 'Pringles', 'Dorito', 'Infuzions', 'Smith', 'GrnWves',
       'Tyrrells', 'Cobs', 'French', 'RRD', 'Tostitos', 'Cheetos',
       'Woolworths', 'Snbts', 'Sunbites'], dtype=object)
```

Some brand names have been doubled up. Replace all contractions and double ups with their full name.

```
# Create a function to identify the string replacements needed.
def replace_brandname(line):
    name = line['BRAND_NAME']
    if name == "Infzns":
        return "Infuzions"
    elif name == "Red":
        return "Red Rock Deli"
    elif name == "RRD":
        return "Red Rock Deli"
    elif name == "Grain":
        return "Grain Waves"
    elif name == "GrnWves":
        return "Grain Waves"
    elif name == "Snbts":
        return "Sunbites"
    elif name == "Natural":
        return "Natural Chip Co"
    elif name == "NCC":
        return "Natural Chip Co"
    elif name == "WW":
        return "Woolworths"
    elif name == "Smith":
        return "Smiths"
    elif name == "Dorito":
        return "Doritos"
    else:
        return name

# Then apply the function to clean the brand names
trans_df["BRAND_NAME"] = trans_df.apply(lambda line:
replace_brandname(line), axis=1)

# Check that there are no duplicate brands
trans_df["BRAND_NAME"].unique()
```

```
array(['Natural Chip Co', 'CCs', 'Smiths', 'Kettle', 'Grain Waves',
       'Doritos', 'Twisties', 'Woolworths', 'Thins', 'Burger',
'Cheezels',
       'Infuzions', 'Red Rock Deli', 'Pringles', 'Tyrrells', 'Cobs',
       'French', 'Tostitos', 'Cheetos', 'Sunbites'], dtype=object)
```

The brand names seme reasonable, without duplicates.

Now we want to examine the customer data. We can generate summaries and check the categories in this dataset.

```
# Now examine customer data
cust_df = customerdata.copy()
cust_df.head()
```

```
   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
```

```
# Rename "PREMIUM_CUSTOMER" to "MEMBER_TYPE" for easier identification
of the column data
cust_df = cust_df.rename(columns={'PREMIUM_CUSTOMER': 'MEMBER_TYPE'})

# Check the summary of the customer data
cust_df.describe()
```

```
       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
```

```
# Check the entries in the member type and lifestage columns
cust_df["MEMBER_TYPE"].unique()
```

```
array(['Premium', 'Mainstream', 'Budget'], dtype=object)
```

```
cust_df["LIFESTAGE"].unique()
```

```
array(['YOUNG SINGLES/COUPLES', 'YOUNG FAMILIES', 'OLDER
SINGLES/COUPLES',
       'MIDAGE SINGLES/COUPLES', 'NEW FAMILIES', 'OLDER FAMILIES',
       'RETIREES'], dtype=object)
```

Now that the customer dataset looks fine, we want to add this information to the transactions dataset.

```
# Join the customer and transaction datasets, and sort transactons by
date
full_df =
trans_df.set_index('LYLTY_CARD_NBR').join(cust_df.set_index('LYLTY_CAR
D_NBR'))
full_df = full_df.reset_index()
full_df = full_df.sort_values(by='DATE').reset_index(drop=True)
full_df
```

```
        LYLTY_CARD_NBR       DATE  STORE_NBR  TXN_ID  PROD_NBR  \
0                21037 2018-07-01         21   17576        62
1                25040 2018-07-01         25   21704        87
```

```
2                59236 2018-07-01        59   55555        42
3               271083 2018-07-01       271  268688        97
4                65015 2018-07-01        65   61737        17
...                ...        ...       ...     ...       ...
246735           48160 2019-06-30        48   44051        11
246736          175371 2019-06-30       175  176890        40
246737          203312 2019-06-30       203  203610        68
246738          222003 2019-06-30       222  221524        17
246739           55142 2019-06-30        55   49322        78

                                      PROD_NAME  PROD_QTY
TOT_SALES  \
0              Pringles Mystery    Flavour 134g         2        7.4

1        Infuzions BBQ Rib   Prawn Crackers 110g         2        7.6

2        Doritos Corn Chip Mexican Jalapeno 150g         2        7.8

3                        RRD Salt & Vinegar  165g         2        6.0

4             Kettle Sensations   BBQ&Maple 150g         2        9.2

...                                         ...       ...        ...

246735                   RRD Pc Sea Salt     165g         2        6.0

246736         Thins Chips Seasonedchicken 175g         2        6.6

246737      Pringles Chicken    Salt Crips 134g         2        7.4

246738        Kettle Sensations   BBQ&Maple 150g         2        9.2

246739         Thins Chips Salt &  Vinegar 175g         2        6.6


        PACK_SIZE     BRAND_NAME               LIFESTAGE MEMBER_TYPE
0           134.0       Pringles                RETIREES  Mainstream
1           110.0      Infuzions          OLDER FAMILIES      Budget
2           150.0        Doritos   OLDER SINGLES/COUPLES      Budget
3           165.0  Red Rock Deli          YOUNG FAMILIES      Budget
4           150.0         Kettle          YOUNG FAMILIES     Premium
...           ...            ...                     ...         ...
246735      165.0  Red Rock Deli                RETIREES  Mainstream
246736      175.0          Thins   OLDER SINGLES/COUPLES      Budget
246737      134.0       Pringles  MIDAGE SINGLES/COUPLES  Mainstream
246738      150.0         Kettle                RETIREES  Mainstream
246739      175.0          Thins                RETIREES  Mainstream

[246740 rows x 12 columns]
```

```python
# Check for nulls in the full dataset
full_df.isnull().values.any()
```

```
False
```

```python
# looks like all the data is reasonable so export to CSV
full_df.to_csv('QVI_fulldata.csv')
```

Data analysis on customer segments

Now that the data has been cleaned, we want to look for interesting insights in the chip market to help recommend a business strategy.

To do so, some metrics we want to consider are:

- 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

Some more information from the data team that we could ask for, to analyse with the chip information for more insight includes

- 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.
- Spending on other snacks, such as crackers and biscuits, to determine the preference and the purchase frequency of chips compared to other snacks
- Proportion of customers in each customer segment overall to compare against the mix of customers who purchase chips

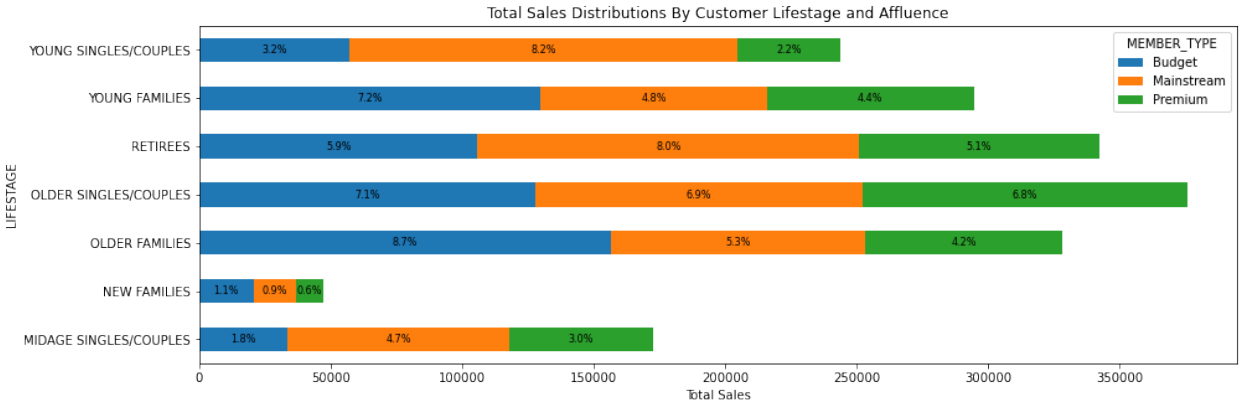
Firstly, we want to take a look at the split of the total sales by LIFESTAGE and MEMBER_TYPE.

```python
# calculate total sales by lifestage and member type and generate a
list
total_sales_cust = full_df.groupby(['LIFESTAGE','MEMBER_TYPE'],
as_index = False)['TOT_SALES'].agg(['sum'])
total_sales_cust = total_sales_cust.rename(columns={'sum':
'sum_tot_sales'})
total_sales_cust.sort_values(by = "sum_tot_sales", ascending = False)
```

LIFESTAGE	MEMBER_TYPE	sum_tot_sales
OLDER FAMILIES	Budget	156863.75
YOUNG SINGLES/COUPLES	Mainstream	147582.20
RETIREES	Mainstream	145168.95
YOUNG FAMILIES	Budget	129717.95
OLDER SINGLES/COUPLES	Budget	127833.60
	Mainstream	124648.50
	Premium	123537.55

```
RETIREES                  Budget          105916.30
OLDER FAMILIES            Mainstream       96413.55
RETIREES                  Premium          91296.65
YOUNG FAMILIES            Mainstream       86338.25
MIDAGE SINGLES/COUPLES Mainstream       84734.25
YOUNG FAMILIES            Premium          78571.70
OLDER FAMILIES            Premium          75242.60
YOUNG SINGLES/COUPLES  Budget           57122.10
MIDAGE SINGLES/COUPLES Premium          54443.85
YOUNG SINGLES/COUPLES  Premium          39052.30
MIDAGE SINGLES/COUPLES Budget           33345.70
NEW FAMILIES              Budget           20607.45
                          Mainstream       15979.70
                          Premium          10760.80
```

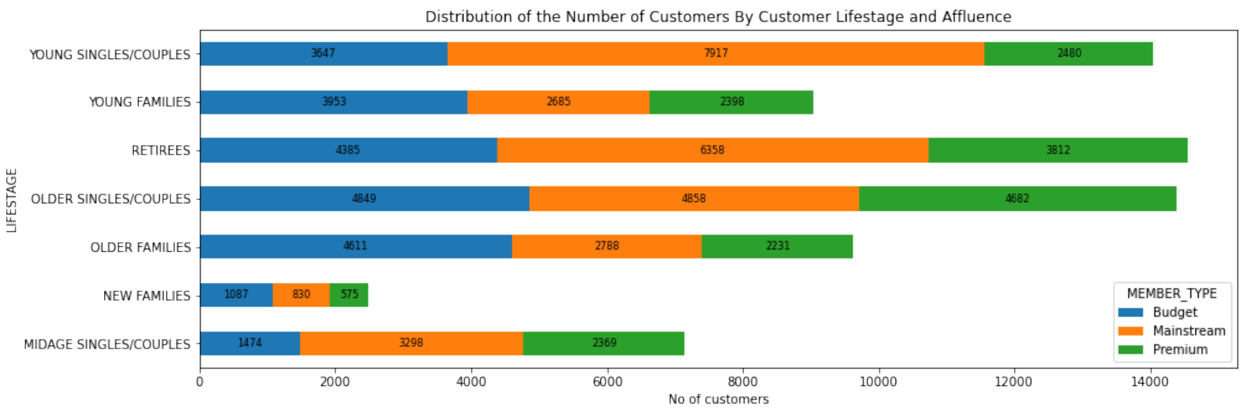
```
# Get the total sales
total_sales = full_df['TOT_SALES'].agg(['sum'])['sum']

# Plot a breakdown of the total sales by lifestage and member type
total_sales_breakdown = full_df.groupby(['LIFESTAGE','MEMBER_TYPE'],
as_index = False)['TOT_SALES'].agg(['sum',
'mean']).unstack('MEMBER_TYPE').fillna(0)
ax = total_sales_breakdown['sum'].plot(kind='barh', stacked=True,
figsize=(15, 5))

# Add percentages of the summed total sales as labels to each bar
# .patches is everything inside of the chart
for rect in ax.patches:
    # Find where everything is located
    height = rect.get_height()
    width = rect.get_width()
    label = width / total_sales * 100
    x = rect.get_x()
    y = rect.get_y()

    label_text = f'{(label):.1f}%'

    # Set label positions
    label_x = x + width / 2
    label_y = y + height / 2

    # only plot labels greater than given width
    if width > 0:
        ax.text(label_x, label_y, label_text, ha='center',
va='center', fontsize=8)

ax.set_xlabel("Total Sales")
ax.set_title('Total Sales Distributions By Customer Lifestage and
Affluence')
plt.show()
```



Here, we can see the most sales are from Older families - Budget, Young singles/couples - Mainstream and Retirees - Mainstream. We can see if this is because of the customer numbers in each segment.

```
# Check all rows are unique in customer information
len(cust_df['LYLTY_CARD_NBR'].unique()) == cust_df.shape[0]

True

# Check if all customers made chip purchases.
len(cust_df['LYLTY_CARD_NBR'].unique()) ==
len(full_df['LYLTY_CARD_NBR'].unique())

False

# Plot the numbers of customers in each segment by counting the unique
LYLTY_CARD_NBR entries
sum_customers= full_df.groupby(['LIFESTAGE','MEMBER_TYPE'])
['LYLTY_CARD_NBR'].agg('nunique').unstack('MEMBER_TYPE').fillna(0)
ax = sum_customers.plot(kind='barh', stacked=True, figsize=(15, 5))

# Add customer numbers as labels to each bar
# .patches is everything inside of the chart
for rect in ax.patches:
    # Find where everything is located
    height = rect.get_height()
    width = rect.get_width()
    x = rect.get_x()
    y = rect.get_y()

    label_text = f'{(width):.0f}'

    # Set label positions
    label_x = x + width / 2
    label_y = y + height / 2

    # only plot labels greater than given width
    if width > 0:
```

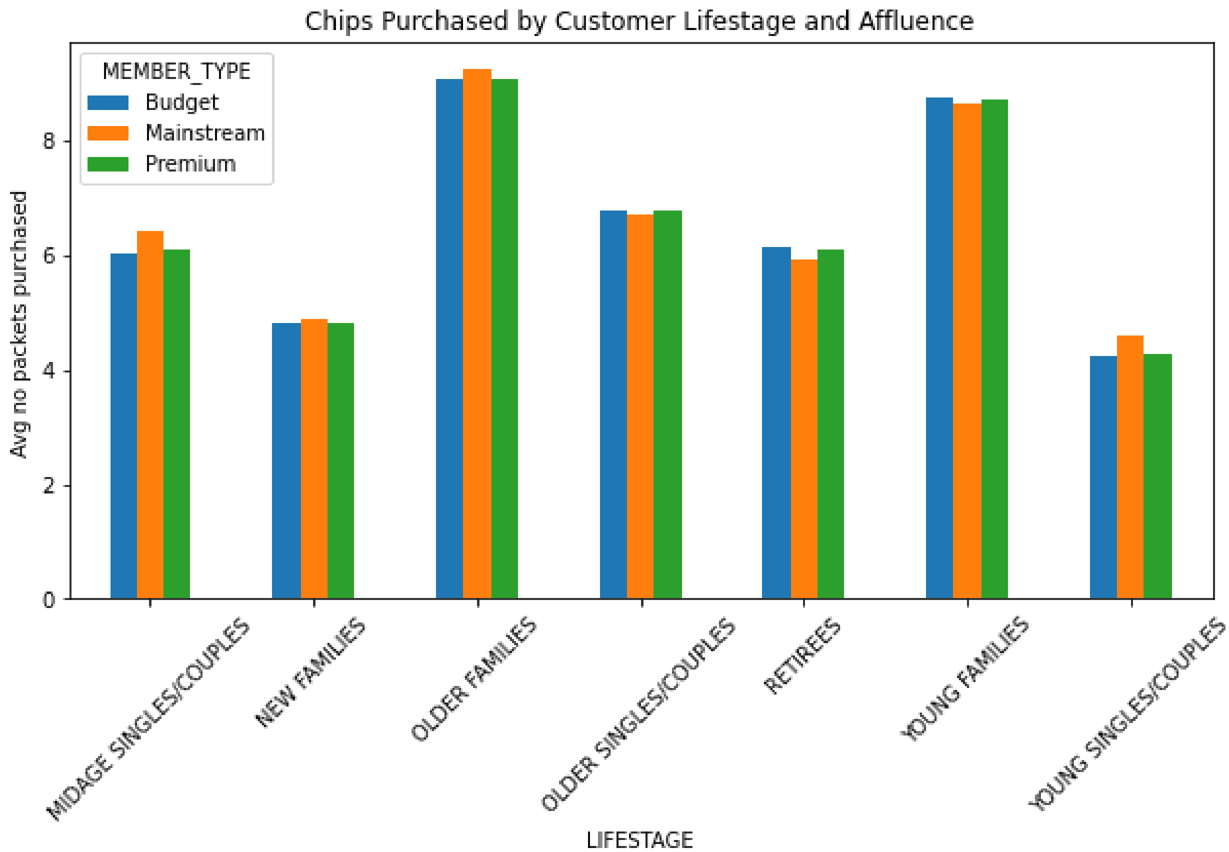
```
        ax.text(label_x, label_y, label_text, ha='center',
va='center', fontsize=8)

ax.set_xlabel("No of customers")
ax.set_title('Distribution of the Number of Customers By Customer
Lifestage and Affluence')
plt.show()
```



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

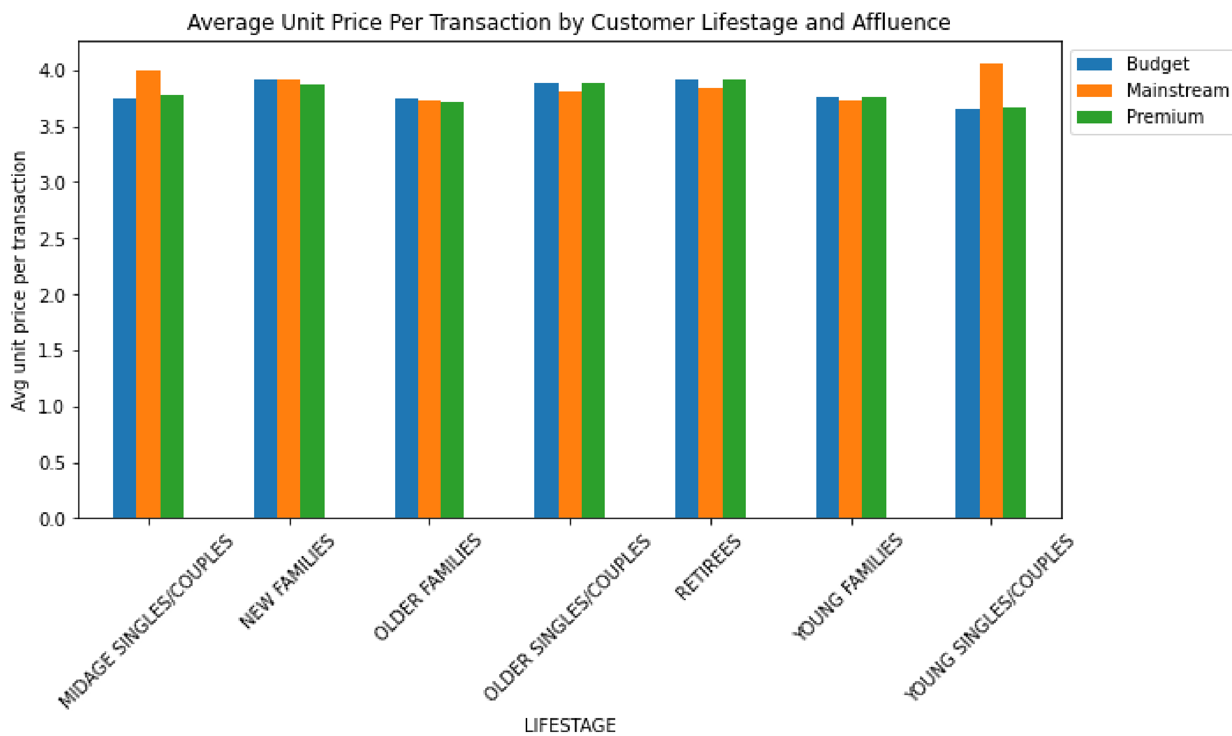
We can then take a look at the total and average units of chips bought per customer by LIFESTAGE and MEMBER_TYPE.

```
# Plot the average no of chip packets bought per customer by LIFESTAGE
and MEMBER_TYPE.
no_packets_data = full_df.groupby(['LIFESTAGE','MEMBER_TYPE'])
['PROD_QTY'].sum()/full_df.groupby(['LIFESTAGE','MEMBER_TYPE'])
['LYLTY_CARD_NBR'].nunique(0)
ax = no_packets_data.unstack('MEMBER_TYPE').fillna(0).plot.bar(stacked
= False,figsize=(10, 5))
ax.set_ylabel("Avg no packets purchased")
ax.set_title('Chips Purchased by Customer Lifestage and Affluence')
plt.xticks(rotation=45)
plt.show()
```



Older families and young families in general buy more chips per customer. We can also investigate the average price per unit sold by LIFESTAGE and MEMBER_TYPE.

```
# Create a column for the unit price of chips purchased per
transaction
full_df['UNIT_PRICE'] = full_df['TOT_SALES']/full_df['PROD_QTY']

# Plot the distribution of the average unit price per transaction by
LIFESTAGE and MEMBER_TYPE.
avg_priceperunit = full_df.groupby(['LIFESTAGE', 'MEMBER_TYPE'],
as_index = False)
['UNIT_PRICE'].agg(['mean']).unstack('MEMBER_TYPE').fillna(0)
ax = avg_priceperunit['mean'].plot.bar(stacked=False, figsize=(10, 5))
ax.set_ylabel("Avg unit price per transaction")
ax.set_title('Average Unit Price Per Transaction by Customer Lifestage
and Affluence')
plt.legend(loc = "upper left",bbox_to_anchor=(1.0, 1.0))
plt.xticks(rotation=45)
plt.show()
```



For young and midage singles/couples, the mainstream group are more willing to pay more for a packet of chips than their budget and premium counterpart. Given the total sales, as well as the number of customers buying chips, is higher in these groups compared to the non-mainstream groups, this suggests that chips may not be the choice of snack for these groups. Further information on shopping habits would be useful in this case.

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

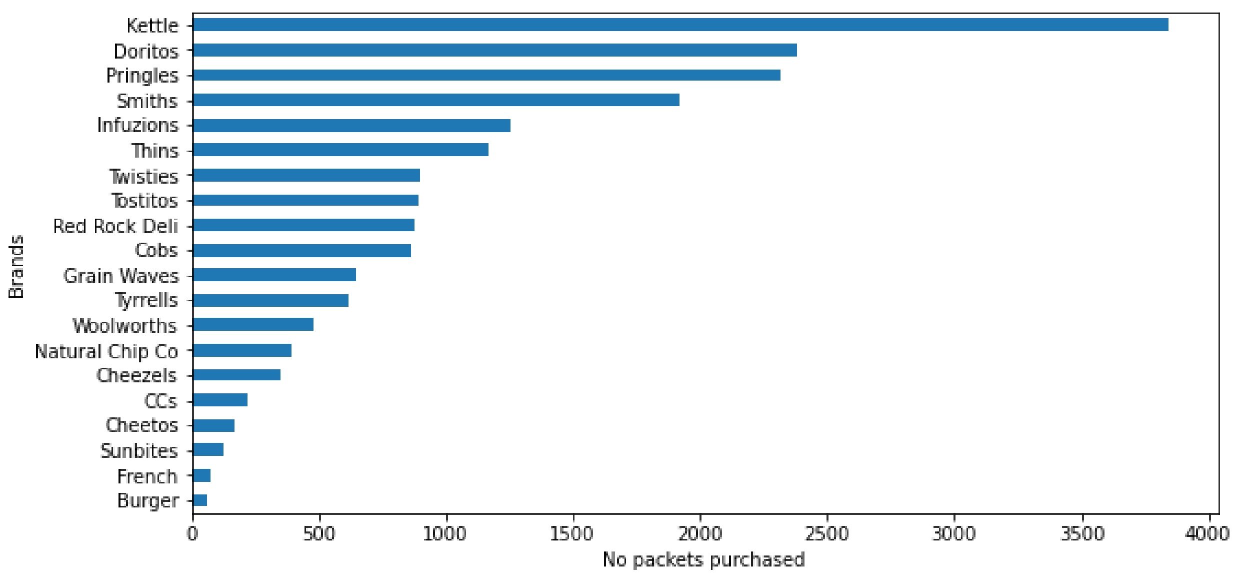
```
# Check the difference in the average price unit between the
mainstream and premium/budget groups for young/midage singles/couples
from scipy.stats import ttest_ind

# Identify the groups to test the hypthesis with
mainstream = full_df["MEMBER_TYPE"] == "Mainstream"
young_midage = (full_df["LIFESTAGE"] == "MIDAGE SINGLES/COUPLES") |
(full_df["LIFESTAGE"] == "YOUNG SINGLES/COUPLES")
premium_budget = full_df["MEMBER_TYPE"] != "Mainstream"

group1 = full_df[mainstream & young_midage]["UNIT_PRICE"]
group2 = full_df[premium_budget & young_midage]["UNIT_PRICE"]

# Generate the t-test
stat, pval = ttest_ind(group1.values, group2.values, equal_var=False)

print(pval, stat)

6.967354232991988e-306 37.6243885962296
```

The t-test results in a p-value of 6.97e-306, being close to 0, indicates that 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.

```
# Create a visual of what brands young singles/couples are purchasing the most for a general indication
young_mainstream = full_df.loc[full_df['LIFESTAGE'] == "YOUNG SINGLES/COUPLES"]
young_mainstream =
young_mainstream.loc[young_mainstream['MEMBER_TYPE'] == "Mainstream"]
ax =
young_mainstream["BRAND_NAME"].value_counts().sort_values(ascending = True).plot.barh(figsize=(10, 5))
ax.set_xlabel("No packets purchased")
ax.set_ylabel("Brands")
plt.show()
```




```
temp = full_df.copy()
temp["group"] = temp["LIFESTAGE"] + ' - ' + temp['MEMBER_TYPE']

groups = pd.get_dummies(temp["group"])
brands = pd.get_dummies(temp["BRAND_NAME"])
```

```
groups_brands = groups.join(brands)
groups_brands
```

```
        MIDAGE SINGLES/COUPLES - Budget  MIDAGE SINGLES/COUPLES - 
Mainstream  \
0                                     0                               
0
1                                     0                               
0
2                                     0                               
0
3                                     0                               
0
4                                     0                               
0
...                                 ...                               
...
246735                                0                               
0
246736                                0                               
0
246737                                0                               
1
246738                                0                               
0
246739                                0                               
0

        MIDAGE SINGLES/COUPLES - Premium  NEW FAMILIES - Budget  \
0                                      0                      0   
1                                      0                      0   
2                                      0                      0   
3                                      0                      0   
4                                      0                      0   
...                                  ...                    ...   
246735                                 0                      0   
246736                                 0                      0   
246737                                 0                      0   
246738                                 0                      0   
246739                                 0                      0   

        NEW FAMILIES - Mainstream  NEW FAMILIES - Premium  \
0                               0                       0   
1                               0                       0   
2                               0                       0   
3                               0                       0   
4                               0                       0   
...                           ...                     ...   
246735                          0                       0   
246736                          0                       0   
```

```
246737                          0                       0
246738                          0                       0
246739                          0                       0

        OLDER FAMILIES - Budget  OLDER FAMILIES - Mainstream  \
0                             0                            0
1                             1                            0
2                             0                            0
3                             0                            0
4                             0                            0
...                         ...                          ...
246735                        0                            0
246736                        0                            0
246737                        0                            0
246738                        0                            0
246739                        0                            0

        OLDER FAMILIES - Premium  OLDER SINGLES/COUPLES - Budget  ...
\
0                              0                               0  ...

1                              0                               0  ...

2                              0                               1  ...

3                              0                               0  ...

4                              0                               0  ...

...                          ...                             ...  ...

246735                         0                               0  ...

246736                         0                               1  ...

246737                         0                               0  ...

246738                         0                               0  ...

246739                         0                               0  ...


        Natural Chip Co  Pringles  Red Rock Deli  Smiths  Sunbites
Thins  \
0                     0         1              0       0         0
0
1                     0         0              0       0         0
0
2                     0         0              0       0         0
0
3                     0         0              1       0         0
```

```
0
4                      0          0              0        0          0
0
...                   ...        ...            ...      ...        ...
...
246735                 0          0              1        0          0
0
246736                 0          0              0        0          0
1
246737                 0          1              0        0          0
0
246738                 0          0              0        0          0
0
246739                 0          0              0        0          0
1

        Tostitos  Twisties  Tyrrells  Woolworths
0              0         0         0           0
1              0         0         0           0
2              0         0         0           0
3              0         0         0           0
4              0         0         0           0
...          ...       ...       ...         ...
246735         0         0         0           0
246736         0         0         0           0
246737         0         0         0           0
246738         0         0         0           0
246739         0         0         0           0

[246740 rows x 41 columns]
```

```
freq_groupsbrands = apriori(groups_brands, min_support=0.008,
use_colnames=True)
rules = association_rules(freq_groupsbrands, metric="lift",
min_threshold=0.5)
rules.sort_values('confidence', ascending = False, inplace = True)

set_temp = temp["group"].unique()
rules[rules["antecedents"].apply(lambda x: list(x)).apply(lambda x: x
in set_temp)]
```

```
                              antecedents consequents  antecedent
support  \
41   (YOUNG SINGLES/COUPLES - Mainstream)    (Kettle)
0.079209
1   (MIDAGE SINGLES/COUPLES - Mainstream)    (Kettle)
0.044966
23                    (RETIREES - Budget)    (Kettle)
0.057652
32                   (RETIREES - Premium)    (Kettle)
```

```
0.049591
13       (OLDER SINGLES/COUPLES - Budget)    (Kettle)
0.069596
21      (OLDER SINGLES/COUPLES - Premium)    (Kettle)
0.067115
27                (RETIREES - Mainstream)    (Kettle)
0.080935
17   (OLDER SINGLES/COUPLES - Mainstream)    (Kettle)
0.069146
35              (YOUNG FAMILIES - Budget)    (Kettle)
0.071991
5               (OLDER FAMILIES - Budget)    (Kettle)
0.087193
10          (OLDER FAMILIES - Mainstream)    (Kettle)
0.053664
9               (OLDER FAMILIES - Budget)    (Smiths)
0.087193
37              (YOUNG FAMILIES - Budget)    (Smiths)
0.071991
39   (YOUNG SINGLES/COUPLES - Mainstream)   (Doritos)
0.079209
19   (OLDER SINGLES/COUPLES - Mainstream)    (Smiths)
0.069146
31                (RETIREES - Mainstream)    (Smiths)
0.080935
42   (YOUNG SINGLES/COUPLES - Mainstream)  (Pringles)
0.079209
15       (OLDER SINGLES/COUPLES - Budget)    (Smiths)
0.069596
28                (RETIREES - Mainstream)  (Pringles)
0.080935
25                (RETIREES - Mainstream)   (Doritos)
0.080935
3               (OLDER FAMILIES - Budget)   (Doritos)
0.087193
6               (OLDER FAMILIES - Budget)  (Pringles)
0.087193

    consequent support   support  confidence      lift  leverage
conviction
41            0.167334  0.015579    0.196684  1.175400  0.002325
1.036537
1             0.167334  0.008657    0.192519  1.150508  0.001132
1.031190
23            0.167334  0.010505    0.182214  1.088926  0.000858
1.018196
32            0.167334  0.008981    0.181105  1.082296  0.000683
1.016816
13            0.167334  0.012422    0.178488  1.066658  0.000776
```

```
1.013578
21              0.167334  0.011944    0.177959  1.063495  0.000713
1.012925
27              0.167334  0.013723    0.169554  1.013269  0.000180
1.002674
17              0.167334  0.011490    0.166168  0.993034 -0.000081
0.998602
35              0.167334  0.011117    0.154422  0.922837 -0.000930
0.984730
5               0.167334  0.013455    0.154318  0.922216 -0.001135
0.984609
10              0.167334  0.008183    0.152481  0.911237 -0.000797
0.982475
9               0.123016  0.011948    0.137027  1.113895  0.001222
1.016236
37              0.123016  0.009459    0.131397  1.068126  0.000603
1.009648
39              0.102229  0.009642    0.121725  1.190712  0.001544
1.022198
19              0.123016  0.008389    0.121329  0.986288 -0.000117
0.998080
31              0.123016  0.009593    0.118528  0.963514 -0.000363
0.994908
42              0.101735  0.009382    0.118451  1.164310  0.001324
1.018962
15              0.123016  0.008146    0.117051  0.951509 -0.000415
0.993244
28              0.101735  0.008523    0.105308  1.035124  0.000289
1.003994
25              0.102229  0.008466    0.104607  1.023260  0.000192
1.002656
3               0.102229  0.008235    0.094450  0.923907 -0.000678
0.991410
6               0.101735  0.008089    0.092777  0.911949 -0.000781
0.990126
```

```
rules[rules['antecedents'] == {'YOUNG SINGLES/COUPLES - Mainstream'}]
```

```
                             antecedents consequents  antecedent
support  \
41  (YOUNG SINGLES/COUPLES - Mainstream)    (Kettle)
0.079209
39  (YOUNG SINGLES/COUPLES - Mainstream)   (Doritos)
0.079209
42  (YOUNG SINGLES/COUPLES - Mainstream)  (Pringles)
0.079209

    consequent support   support  confidence      lift  leverage
conviction
41              0.167334  0.015579    0.196684  1.175400  0.002325
```

```
1.036537
39            0.102229  0.009642   0.121725  1.190712  0.001544
1.022198
42            0.101735  0.009382   0.118451  1.164310  0.001324
1.018962
```

From apriori analysis, we can see that for Mainstream - young singles/couples, Kettle is the brand of choice. This is also true for most other segments. We can use the affinity index to see if there are brands this segment prefers more than the other segments to target.

```
# find the target rating proportion
target_segment =
young_mainstream["BRAND_NAME"].value_counts().sort_values(ascending =
True).rename_axis('BRANDS').reset_index(name='target')
target_segment.target /= young_mainstream["PROD_QTY"].sum()

# find the other rating proportion
not_young_mainstream = full_df.loc[full_df['LIFESTAGE'] != "YOUNG
SINGLES/COUPLES"]
not_young_mainstream =
not_young_mainstream.loc[not_young_mainstream['MEMBER_TYPE'] !=
"Mainstream"]
other =
not_young_mainstream["BRAND_NAME"].value_counts().sort_values(ascendin
g = True).rename_axis('BRANDS').reset_index(name='other')
other.other /= not_young_mainstream["PROD_QTY"].sum()

# join the two dataframes
brand_proportions =
target_segment.set_index('BRANDS').join(other.set_index('BRANDS'))
# full_df =
trans_df.set_index('LYLTY_CARD_NBR').join(cust_df.set_index('LYLTY_CAR
D_NBR'))
brand_proportions = brand_proportions.reset_index()
brand_proportions['affinity'] =
brand_proportions['target']/brand_proportions['other']
brand_proportions.sort_values(by = 'affinity', ascending = False)
```

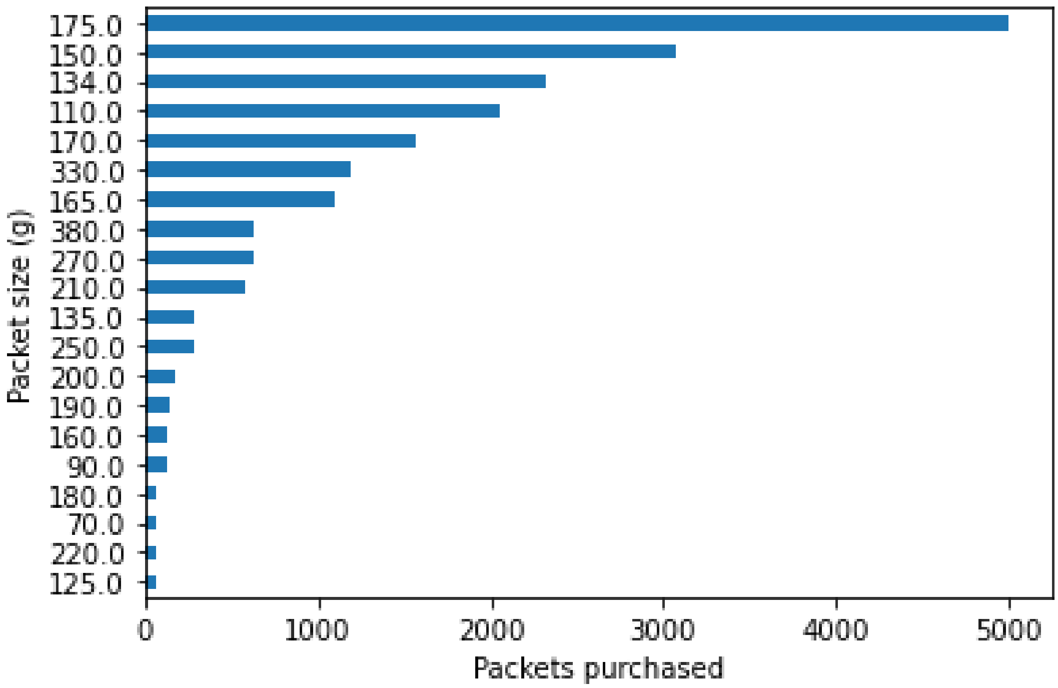
	BRANDS	target	other	affinity
8	Tyrrells	0.017088	0.013368	1.278270
13	Twisties	0.024845	0.019632	1.265496
18	Doritos	0.065673	0.052511	1.250646
12	Tostitos	0.024569	0.019944	1.231911
19	Kettle	0.106115	0.086574	1.225712
17	Pringles	0.063906	0.052477	1.217793
10	Cobs	0.023851	0.020004	1.192293
15	Infuzions	0.034507	0.029930	1.152890
9	Grain Waves	0.017833	0.016214	1.099878
14	Thins	0.032188	0.029771	1.081172

```
5           Cheezels  0.009551  0.009866  0.968161
16            Smiths  0.053030  0.064809  0.818247
3            Cheetos  0.004582  0.006139  0.746405
1             French  0.002153  0.003017  0.713793
11     Red Rock Deli  0.024155  0.035152  0.687154
6    Natural Chip Co  0.010876  0.016236  0.669883
4                CCs  0.006128  0.009668  0.633867
2           Sunbites  0.003533  0.006576  0.537349
7         Woolworths  0.013223  0.025567  0.517189
0             Burger  0.001712  0.003415  0.501180
```

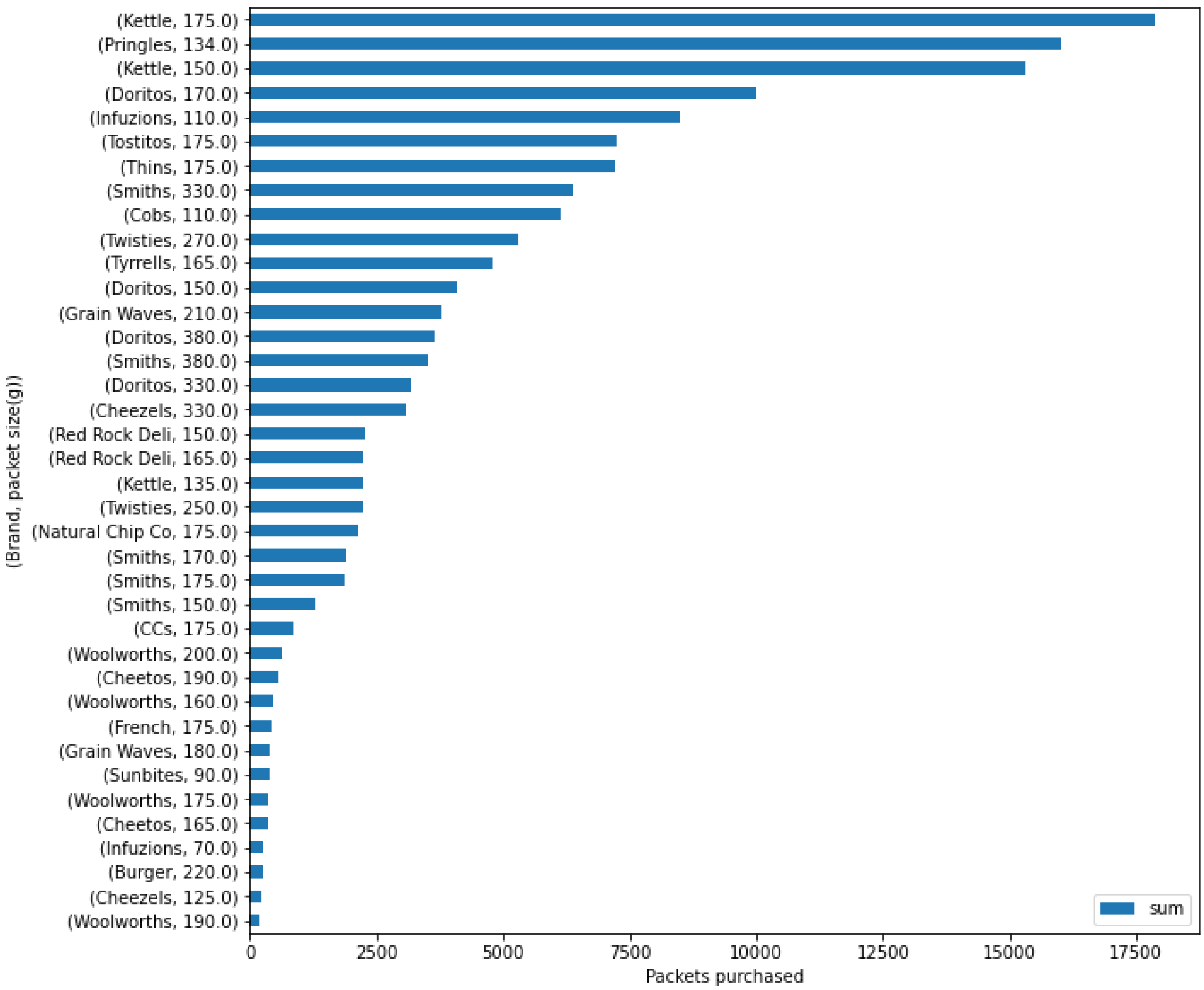
By using the affinity index, we can see that mainstream young singles/couples are 28% more likely to purcahse Tyrrells chips than the other segments. However, they are 50% less likely to purchase Burger Rings.

We also want to find out if our target segment tends to buy larger packs of chips.

```
# Plot the distribution of the packet sizes for a general indication
of what it most popular.
young_mainstream = full_df.loc[full_df['LIFESTAGE'] == "YOUNG
SINGLES/COUPLES"]
young_mainstream =
young_mainstream.loc[young_mainstream['MEMBER_TYPE'] == "Mainstream"]
ax =
young_mainstream["PACK_SIZE"].value_counts().sort_values(ascending =
True).plot.barh()
ax.set_ylabel("Packet size (g)")
ax.set_xlabel("Packets purchased")
plt.show()
```




```
# Also want to check which brands correspond to what sized packets.
brand_size = young_mainstream.groupby(['BRAND_NAME','PACK_SIZE'],
as_index = False)['TOT_SALES'].agg(['sum'])
ax = brand_size.sort_values(by = 'sum').plot.barh(y = "sum",
figsize=(10,10))
ax.set_ylabel("(Brand, packet size(g))")
ax.set_xlabel("Packets purchased")
plt.show()
```




```
groups = pd.get_dummies(temp["group"])
brands = pd.get_dummies(temp["PACK_SIZE"])
groups_brands = groups.join(brands)
groups_brands
```

```
        MIDAGE SINGLES/COUPLES - Budget  MIDAGE SINGLES/COUPLES -
Mainstream  \
0                                     0
0
1                                     0
0
2                                     0
0
3                                     0
0
4                                     0
0
...                                 ...
...
```

```
246735                                      0
0
246736                                      0
0
246737                                      0
1
246738                                      0
0
246739                                      0
0

        MIDAGE SINGLES/COUPLES - Premium  NEW FAMILIES - Budget  \
0                                      0                      0
1                                      0                      0
2                                      0                      0
3                                      0                      0
4                                      0                      0
...                                  ...                    ...
246735                                 0                      0
246736                                 0                      0
246737                                 0                      0
246738                                 0                      0
246739                                 0                      0

        NEW FAMILIES - Mainstream  NEW FAMILIES - Premium  \
0                               0                       0
1                               0                       0
2                               0                       0
3                               0                       0
4                               0                       0
...                           ...                     ...
246735                          0                       0
246736                          0                       0
246737                          0                       0
246738                          0                       0
246739                          0                       0

        OLDER FAMILIES - Budget  OLDER FAMILIES - Mainstream  \
0                             0                            0
1                             1                            0
2                             0                            0
3                             0                            0
4                             0                            0
...                         ...                          ...
246735                        0                            0
246736                        0                            0
246737                        0                            0
246738                        0                            0
246739                        0                            0
```

```
        OLDER FAMILIES - Premium  OLDER SINGLES/COUPLES - Budget  ...  
175.0  \
0                              0                               0  ...  
0
1                              0                               0  ...  
0
2                              0                               1  ...  
0
3                              0                               0  ...  
0
4                              0                               0  ...  
0
...                          ...                             ...  ...  
...
246735                         0                               0  ...  
0
246736                         0                               1  ...  
1
246737                         0                               0  ...  
0
246738                         0                               0  ...  
0
246739                         0                               0  ...  
1

        180.0  190.0  200.0  210.0  220.0  250.0  270.0  330.0  380.0  

0           0      0      0      0      0      0      0      0      0  

1           0      0      0      0      0      0      0      0      0  

2           0      0      0      0      0      0      0      0      0  

3           0      0      0      0      0      0      0      0      0  

4           0      0      0      0      0      0      0      0      0  

...       ...    ...    ...    ...    ...    ...    ...    ...    ...  

246735      0      0      0      0      0      0      0      0      0  

246736      0      0      0      0      0      0      0      0      0  

246737      0      0      0      0      0      0      0      0      0  

246738      0      0      0      0      0      0      0      0      0  

246739      0      0      0      0      0      0      0      0      0  


[246740 rows x 41 columns]
```

```
freq_groupsbrands = apriori(groups_brands, min_support=0.009,
use_colnames=True)
rules = association_rules(freq_groupsbrands, metric="lift",
min_threshold=0.5)
rules.sort_values('confidence', ascending = False, inplace = True)
set_temp = temp["group"].unique()
rules[rules["antecedents"].apply(lambda x: list(x)).apply(lambda x: x
in set_temp)]
```

```
                              antecedents consequents  antecedent
support  \
38          (YOUNG FAMILIES - Premium)     (175.0)
0.043706
34           (YOUNG FAMILIES - Budget)     (175.0)
0.071991
40    (YOUNG SINGLES/COUPLES - Budget)     (175.0)
0.034745
6        (OLDER FAMILIES - Mainstream)     (175.0)
0.053664
8           (OLDER FAMILIES - Premium)     (175.0)
0.042162
24                (RETIREES - Budget)     (175.0)
0.057652
30               (RETIREES - Premium)     (175.0)
0.049591
5            (OLDER FAMILIES - Budget)     (175.0)
0.087193
12    (OLDER SINGLES/COUPLES - Budget)     (175.0)
0.069596
21   (OLDER SINGLES/COUPLES - Premium)     (175.0)
0.067115
0   (MIDAGE SINGLES/COUPLES - Mainstream)     (175.0)
0.044966
36       (YOUNG FAMILIES - Mainstream)     (175.0)
0.048419
17  (OLDER SINGLES/COUPLES - Mainstream)     (175.0)
0.069146
29            (RETIREES - Mainstream)     (175.0)
0.080935
46  (YOUNG SINGLES/COUPLES - Mainstream)     (175.0)
0.079209
19   (OLDER SINGLES/COUPLES - Premium)     (150.0)
0.067115
3            (OLDER FAMILIES - Budget)     (150.0)
0.087193
27            (RETIREES - Mainstream)     (150.0)
0.080935
10    (OLDER SINGLES/COUPLES - Budget)     (150.0)
0.069596
22                (RETIREES - Budget)     (150.0)
```

```
0.057652
15   (OLDER SINGLES/COUPLES - Mainstream)     (150.0)
0.069146
33              (YOUNG FAMILIES - Budget)     (150.0)
0.071991
44   (YOUNG SINGLES/COUPLES - Mainstream)     (150.0)
0.079209
42   (YOUNG SINGLES/COUPLES - Mainstream)     (134.0)
0.079209

    consequent support   support  confidence      lift  leverage
conviction
38            0.269069  0.012150    0.278004  1.033210  0.000391
1.012377
34            0.269069  0.019944    0.277037  1.029613  0.000574
1.011021
40            0.269069  0.009476    0.272717  1.013558  0.000127
1.005016
6             0.269069  0.014542    0.270977  1.007091  0.000102
1.002617
8             0.269069  0.011413    0.270691  1.006030  0.000068
1.002225
24            0.269069  0.015591    0.270439  1.005094  0.000079
1.001879
30            0.269069  0.013399    0.270186  1.004154  0.000055
1.001531
5             0.269069  0.023539    0.269964  1.003327  0.000078
1.001226
12            0.269069  0.018744    0.269334  1.000985  0.000018
1.000363
21            0.269069  0.018068    0.269203  1.000499  0.000009
1.000184
0             0.269069  0.012057    0.268139  0.996544 -0.000042
0.998729
36            0.269069  0.012864    0.265673  0.987381 -0.000164
0.995376
17            0.269069  0.018339    0.265225  0.985714 -0.000266
0.994769
29            0.269069  0.021460    0.265148  0.985428 -0.000317
0.994664
46            0.269069  0.020252    0.255679  0.950239 -0.001061
0.982012
19            0.162937  0.011218    0.167150  1.025857  0.000283
1.005059
3             0.162937  0.014542    0.166775  1.023558  0.000335
1.004607
27            0.162937  0.013334    0.164747  1.011111  0.000147
1.002168
10            0.162937  0.011393    0.163697  1.004665  0.000053
```

```
1.000909
22             0.162937  0.009399    0.163023  1.000529  0.000005
1.000103
15             0.162937  0.011239    0.162534  0.997531 -0.000028
0.999520
33             0.162937  0.011599    0.161121  0.988859 -0.000131
0.997836
44             0.162937  0.012483    0.157593  0.967205 -0.000423
0.993657
42             0.101735  0.009382    0.118451  1.164310  0.001324
1.018962
```

While it appears that most segments purchase more chip packets that are 175g, which is also the size that most Kettles chips are purchased in, we can also determine whether mainstream young singles/couples have certain preferences over the other segments again using the affinity index.

```
# find the target rating proportion
target_segment =
young_mainstream["PACK_SIZE"].value_counts().sort_values(ascending =
True).rename_axis('SIZES').reset_index(name='target')
target_segment.target /= young_mainstream["PROD_QTY"].sum()

# find the other rating proportion
other =
not_young_mainstream["PACK_SIZE"].value_counts().sort_values(ascending
= True).rename_axis('SIZES').reset_index(name='other')
other.other /= not_young_mainstream["PROD_QTY"].sum()

# join the two dataframes
brand_proportions =
target_segment.set_index('SIZES').join(other.set_index('SIZES'))
brand_proportions = brand_proportions.reset_index()
brand_proportions['affinity'] =
brand_proportions['target']/brand_proportions['other']
brand_proportions.sort_values(by = 'affinity', ascending = False)
```

	SIZES	target	other	affinity
11	270.0	0.017115	0.012958	1.320826
12	380.0	0.017281	0.013375	1.291992
14	330.0	0.032988	0.026455	1.246968
10	210.0	0.015901	0.012973	1.225655
17	134.0	0.063906	0.052477	1.217793
16	110.0	0.056618	0.046653	1.213618
9	135.0	0.008006	0.006750	1.185951
8	250.0	0.007729	0.006674	1.158076
15	170.0	0.043478	0.041826	1.039502
18	150.0	0.085024	0.084969	1.000652
19	175.0	0.137943	0.141498	0.974878
13	165.0	0.030421	0.032135	0.946660

```
6   190.0  0.004086  0.006318  0.646684
3   180.0  0.001932  0.003240  0.596328
5   160.0  0.003533  0.006428  0.549720
4    90.0  0.003533  0.006576  0.537349
2    70.0  0.001739  0.003282  0.529870
0   125.0  0.001629  0.003153  0.516530
7   200.0  0.004941  0.009714  0.508695
1   220.0  0.001712  0.003415  0.501180
```

Here, we can see that mainstream young singles/couples are 32% more likely to purcahse 270g chips than the other segments. However, they are 50% less likely to purchase 220g chips. The chips that come in 270g bags are Twisties while Burger Rings come in 220g bags, which is consistent with the affinity testing for the chip brands.

Summary of Insights

The three highest contributing segments to the total sales are:

1. Older families - Budget
2. Young singles/couples - Mainstream
3. Retirees - Mainstream

The largest population group is mainstream young singles/couples, followed by mainstream retirees which explains their large total sales. While population is not a driving factor for budget older families, older families and young families in general buy more chips per customer. Furthermore, mainstream young singles/couples have the highest spend per purchase, which is statistically significant compared to the non-mainstream young singles/couples. Taking a further look at the mainstream yong singles/couples segment, we have found that they are 28% more likely to purchase Tyrells chips than the other segments. This segment does purchase the most Kettles chips, which is also consistent with most other segments. However, they are 50% less likely to purchase Burger Rings, which was also evident in the preferences for packet sizes given they are the only chips that come in 220g sizes. Mainstream young singles/couples are 32% more likely to purchase 270g chips, which is the size that Twisties come in, compare to the other segments. The packet size purchased most over many segments is 175g.

Perhaps we can use the fact that Tyrells and (the packet size of) Twisties chips are more likely to be purchased by mainstream young singles/couples and place these products where they are more likely to be seen by this segment. Furthermore, given that Kettles chips are still the most popular, if the primary target segment are mainstream young singles/couples, Tyrells and Twisties could be placed closer to the Kettles chips. This strategy, with the brands they are more likely to purchase, could also be applied to other segments that purchase the most of Kettles to increase their total sales.