In [3]:
```
import pandas as pd
import numpy as np
import seaborn as sns
```

In [15]:
```
customer_data = pd.read_csv(r"D:\Documents\PowerBi Projects\Quantim\QVI_purchase_behaviour.csv")
```

In [17]:
```
transaction_data = pd.read_excel(r"D:\Documents\PowerBi Projects\Quantim\QVI_transaction_data.xlsx")
```

In [18]:
```
customer_data.head()
```

Out[18]:

	LYLTY_CARD_NBR	LIFESTAGE	PREMIUM_CUSTOMER
0	1000	YOUNG SINGLES/COUPLES	Premium
1	1002	YOUNG SINGLES/COUPLES	Mainstream
2	1003	YOUNG FAMILIES	Budget
3	1004	OLDER SINGLES/COUPLES	Mainstream
4	1005	MIDAGE SINGLES/COUPLES	Mainstream

In [21]:
```
transaction_data.describe()
```

Out[21]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_QTY	TOT_SALES
count	264836.000000	264836.00000	2.648360e+05	2.648360e+05	264836.000000	264836.000000	264836.000000
mean	43464.036260	135.08011	1.355495e+05	1.351583e+05	56.583157	1.907309	7.304200
std	105.389282	76.78418	8.057998e+04	7.813303e+04	32.826638	0.643654	3.083226
min	43282.000000	1.00000	1.000000e+03	1.000000e+00	1.000000	1.000000	1.500000
25%	43373.000000	70.00000	7.002100e+04	6.760150e+04	28.000000	2.000000	5.400000
50%	43464.000000	130.00000	1.303575e+05	1.351375e+05	56.000000	2.000000	7.400000
75%	43555.000000	203.00000	2.030942e+05	2.027012e+05	85.000000	2.000000	9.200000
max	43646.000000	272.00000	2.373711e+06	2.415841e+06	114.000000	200.000000	650.000000

In [23]:

```
transaction_data.isnull().sum()
```

Out[23]:

```
DATE              0
STORE_NBR         0
LYLTY_CARD_NBR    0
TXN_ID            0
PROD_NBR          0
PROD_NAME         0
PROD_QTY          0
TOT_SALES         0
dtype: int64
```

In [29]:

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

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

Examine the Outlier

In [31]:
```
import matplotlib.pyplot as plt
```

In [35]:
```
import datetime
import re
```

In [43]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [47]:
```
# Read data files into data frames
customerdata = pd.read_csv(r"D:\Documents\PowerBi Projects\Quantim\QVI_purchase_behaviour.csv")
transactiondata = pd.read_excel(r"D:\Documents\PowerBi Projects\Quantim\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.

In [51]:
```
# Examining the transaction data - view a summary of the table
trans_df = transactiondata.copy() # Keep a copy for a quick reset
trans_df
```

Out[51]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES
0	43390	1	1000	1	5	Natural Chip Compny SeaSalt175g	2	6.0
1	43599	1	1307	348	66	CCs Nacho Cheese 175g	3	6.3
2	43605	1	1343	383	61	Smiths Crinkle Cut Chips Chicken 170g	2	2.9
3	43329	2	2373	974	69	Smiths Chip Thinly S/Cream&Onion 175g	5	15.0
4	43330	2	2426	1038	108	Kettle Tortilla ChpsHny&Jlpno Chili 150g	3	13.8
...	...	...	...	...	...	...	...	...
264831	43533	272	272319	270088	89	Kettle Sweet Chilli And Sour Cream 175g	2	10.8
264832	43325	272	272358	270154	74	Tostitos Splash Of Lime 175g	1	4.4
264833	43410	272	272379	270187	51	Doritos Mexicana 170g	2	8.8
264834	43461	272	272379	270188	42	Doritos Corn Chip Mexican Jalapeno 150g	2	7.8
264835	43365	272	272380	270189	74	Tostitos Splash Of Lime 175g	2	8.8

264836 rows × 8 columns

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

In [53]:

```
# 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.

In [55]:

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

```
Out[55]: 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.

In [57]:
```
# 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'&',' ');
```

In [59]:
```
# 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)
```

```
175g          60561
Chips         49770
150g          41633
Kettle        41288
Smiths        28860
Salt          27976
Cheese        27890
Pringles      25102
134g          25102
Doritos       24962
Crinkle       23960
110g          22387
Corn          22063
Original      21560
Cut           20754
Chip          18645
170g          18502
Salsa         18094
Chicken       15407
Chilli        15390
165g          15297
Sea           14145
Thins         14075
Sour          13882
Crisps        12607
330g          12540
Vinegar       12402
300g          12041
RRD           11894
Sweet         11060
Infuzions     11057
Supreme       10963
Chives        10951
Cream         10723
WW            10320
Popd           9693
Cobs           9693
Tortilla       9580
Tostitos       9471
Twisties       9454
BBQ            9434
```

```
Sensations        9429
Lime              9347
Dip               9324
Old               9324
El                9324
Paso              9324
Tomato            7669
Thinly            7507
Tyrrells          6442
380g              6418
And               6373
Tangy             6332
SourCream         6296
Grain             6272
Waves             6272
Salted            6248
Lightly           6248
Soy               6121
Natural           6050
Mild              6048
Deli              5885
Red               5885
Rock              5885
Thai              4737
Burger            4733
Swt               4718
Honey             4661
Nacho             4658
Potato            4647
Onion             4635
Cheezels          4603
Garlic            4572
CCs               4551
200g              4473
Woolworths        4437
Pesto             3304
Mozzarella        3304
Basil             3304
Jlpno             3296
Chili             3296
ChpsHny           3296
```

```
Swt/Chlli         3269
Sr/Cream          3269
Ched              3268
Pot               3257
135g              3257
Of                3252
Splash            3252
SweetChili        3242
PotatoMix         3242
Crnkle            3233
Orgnl             3233
Big               3233
Bag               3233
Hot               3229
Spicy             3229
Fig               3219
Camembert         3219
Barbeque          3210
Mexican           3204
Jalapeno          3204
Light             3188
Chp               3185
Dorito            3185
Spcy              3177
Rib               3174
Crackers          3174
Prawn             3174
Southern          3172
Chicken270g       3170
250g              3169
210g              3167
Crm               3159
Ricotta           3146
ChpsBtroot        3146
Chipotle          3145
Smoked            3145
Infzns            3144
Crn               3144
Crnchers          3144
Gcamole           3144
ChpsFeta          3138
```

```
Veg               3134
Herbs             3134
Strws             3134
Siracha           3127
Tom               3125
Chnky             3125
Ht300g            3125
270g              3115
Mexicana          3115
Flavour           3114
Mystery           3114
Seasonedchicken   3114
Med               3114
210G              3105
Crips             3104
Slt               3095
Vingar            3095
Maple             3083
Sthrn             3083
FriedChicken      3083
Rings             3080
ChipCo            3010
90g               3008
190g              2995
SR                2984
160g              2970
Smith             2963
Chs               2960
Cheetos           2927
Medium            2879
French            2856
Snbts             1576
Whlgrn            1576
Cheddr            1576
Mstrd             1576
Spce              1572
Tmato             1572
Co                1572
Hrb               1572
220g              1564
Vinegr            1550
```

```
Tasty           1539
Belly           1526
Pork            1526
Rst             1526
Slow            1526
Roast           1519
N               1512
Mac             1512
Mango           1507
70g             1507
Chutny          1507
Papadums        1507
Coconut         1506
Sauce           1503
Snag            1503
Truffle         1498
Sp              1498
150G            1498
Barbecue        1489
Stacked         1487
OnionStacked    1483
Onion170g       1481
Balls           1479
Bacon           1479
S/Cream         1473
Pepper          1473
D/Style         1469
Compny          1468
SeaSalt175g     1468
Jam             1468
GrnWves         1468
Plus            1468
Btroot          1468
180g            1468
Chli            1461
S/Cream175G     1461
Hony            1460
Chckn175g       1460
Mzzrlla         1458
Steak           1455
Chimuchurri     1455
```

```
Box                  1454
125g                 1454
Bolognese            1451
Puffs                1448
Originl              1441
saltd                1441
CutSalt/Vinegr175g   1440
OnionDip             1438
Chikn                1434
Aioli                1434
Frch/Onin            1432
Whlegrn              1432
Sunbites             1432
Pc                   1431
Garden               1419
NCC                  1419
Fries                1418
Name: count, dtype: int64
```

In [61]:

```
# 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
```

Out[61]: (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.

In [63]:

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

Out[63]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_QTY	TOT_SALES
count	246742	246742.000000	2.467420e+05	2.467420e+05	246742.000000	246742.000000	246742.000000
mean	2018-12-30 01:19:01.211467520	135.051098	1.355310e+05	1.351311e+05	56.351789	1.908062	7.321322
min	2018-07-01 00:00:00	1.000000	1.000000e+03	1.000000e+00	1.000000	1.000000	1.700000
25%	2018-09-30 00:00:00	70.000000	7.001500e+04	6.756925e+04	26.000000	2.000000	5.800000
50%	2018-12-30 00:00:00	130.000000	1.303670e+05	1.351830e+05	53.000000	2.000000	7.400000
75%	2019-03-31 00:00:00	203.000000	2.030840e+05	2.026538e+05	87.000000	2.000000	8.800000
max	2019-06-30 00:00:00	272.000000	2.373711e+06	2.415841e+06	114.000000	200.000000	650.000000
std	NaN	76.787096	8.071528e+04	7.814772e+04	33.695428	0.659831	3.077828

In [65]:
```
# Check if there are any nans in the dataset
trans_df.isnull().values.any()
```

Out[65]: False

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

In [67]:
```
# Filter the entries that have 200 packets.
trans_df.loc[trans_df['PROD_QTY'] == 200.0]
```

Out[67]:

	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

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.

In [69]:
```
# Filter the entires by the customer
trans_df.loc[trans_df['LYLTY_CARD_NBR'] == 226000]
```

Out[69]:

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

In [71]:
```
# 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)
```

Out[71]: (246740, 8)

In [73]:
```
# Recheck the data summary
trans_df.describe()
```

Out[73]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_QTY	TOT_SALES
count	246740	246740.000000	2.467400e+05	2.467400e+05	246740.000000	246740.000000	246740.000000
mean	2018-12-30 01:18:58.448569344	135.050361	1.355303e+05	1.351304e+05	56.352213	1.906456	7.316113
min	2018-07-01 00:00:00	1.000000	1.000000e+03	1.000000e+00	1.000000	1.000000	1.700000
25%	2018-09-30 00:00:00	70.000000	7.001500e+04	6.756875e+04	26.000000	2.000000	5.800000
50%	2018-12-30 00:00:00	130.000000	1.303670e+05	1.351815e+05	53.000000	2.000000	7.400000
75%	2019-03-31 00:00:00	203.000000	2.030832e+05	2.026522e+05	87.000000	2.000000	8.800000
max	2019-06-30 00:00:00	272.000000	2.373711e+06	2.415841e+06	114.000000	5.000000	29.500000
std	NaN	76.786971	8.071520e+04	7.814760e+04	33.695235	0.342499	2.474897

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.

In [75]:
```
# 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
```

Out[75]: (364, 2)

In [77]:
```
# There is one day of data missing. First check the range of dates by sorting in time order.
trans_df.sort_values(by='DATE')
```

Out[77]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES
9161	2018-07-01	88	88140	86914	25	Pringles SourCream Onion 134g	2	7.4
155442	2018-07-01	60	60276	57330	3	Kettle Sensations Camembert & Fig 150g	2	9.2
181349	2018-07-01	199	199014	197623	104	Infuzions Thai SweetChili PotatoMix 110g	2	7.6
229948	2018-07-01	35	35052	31630	11	RRD Pc Sea Salt 165g	1	3.0
104647	2018-07-01	72	72104	71038	20	Doritos Cheese Supreme 330g	2	11.4
...	...	...	...	...	...	...	...	...
10254	2019-06-30	112	112141	114611	98	NCC Sour Cream & Garden Chives 175g	2	6.0
113220	2019-06-30	207	207155	205513	99	Pringles Sthrn FriedChicken 134g	2	7.4
229182	2019-06-30	10	10140	9882	12	Natural Chip Co Tmato Hrb&Spce 175g	2	6.0
229015	2019-06-30	6	6258	6047	29	French Fries Potato Chips 175g	1	3.0
262768	2019-06-30	183	183196	185975	22	Thins Chips Originl saltd 175g	2	6.6

246740 rows × 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.

In [79]:
```
# 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)
```

Out[79]:
```
DatetimeIndex(['2018-12-25'], dtype='datetime64[ns]', freq='D')
```

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.

In [95]:
```
# Remove 'PACK_SIZE' column if it already exists
if 'PACK_SIZE' in trans_df.columns:
    trans_df.drop(columns=['PACK_SIZE'], inplace=True)

# Add the new 'PACK_SIZE' column with packet sizes extracted from 'PROD_NAME'
trans_df.insert(8, "PACK_SIZE", trans_df['PROD_NAME'].str.extract(r'(\d+)').astype(float), True)

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

In [97]:
```
trans_df
```

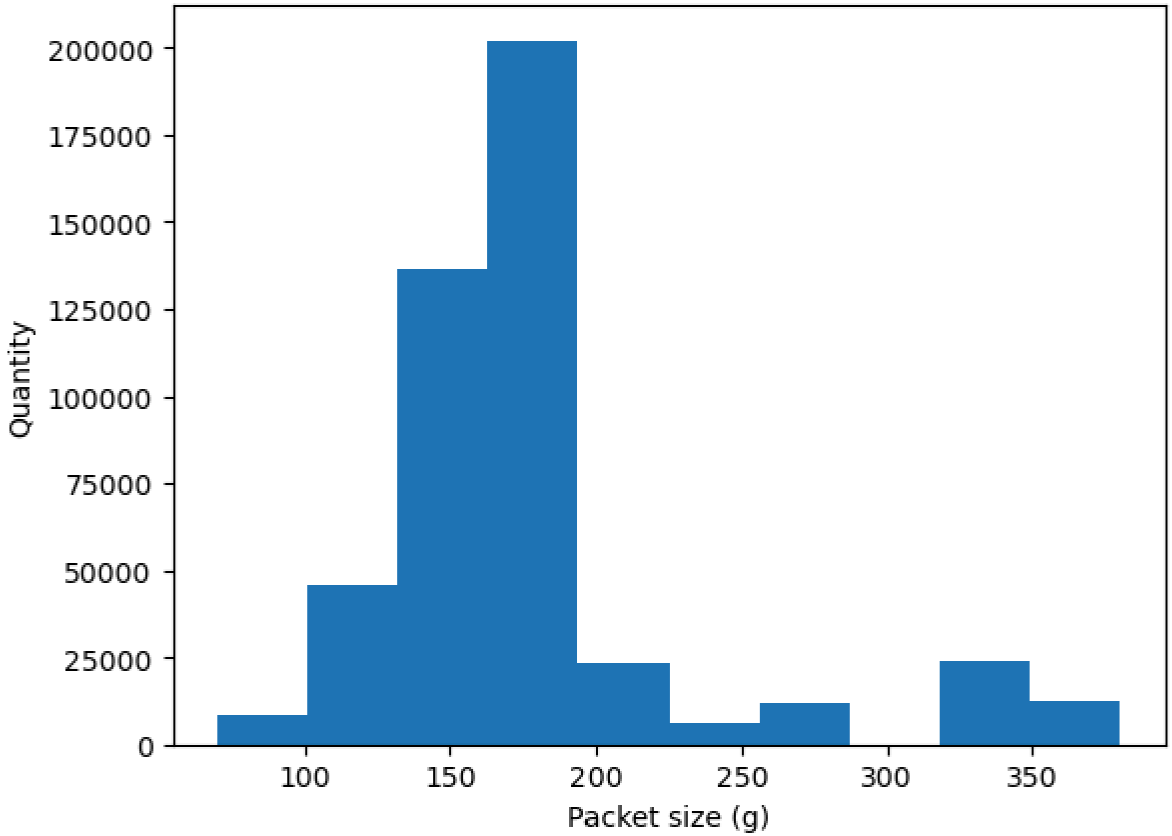
Out[97]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PACK_SIZE
40783	2018-09-25	97	97067	96696	38	Infuzions Mango Chutny Papadums 70g	2	4.8	70.0
222256	2018-12-16	114	114138	117726	38	Infuzions Mango Chutny Papadums 70g	2	4.8	70.0
261068	2019-03-23	103	103109	103191	38	Infuzions Mango Chutny Papadums 70g	1	2.4	70.0
154268	2019-04-15	47	47110	42492	38	Infuzions Mango Chutny Papadums 70g	2	4.8	70.0
76489	2018-09-29	164	164110	164477	38	Infuzions Mango Chutny Papadums 70g	2	4.8	70.0
...	...	...	...	...	...	...	...	...	...
20302	2019-04-02	104	104151	104652	14	Smiths Crnkle Chip Orgnl Big Bag 380g	2	11.8	380.0
178013	2018-12-05	119	119152	122742	4	Dorito Corn Chp Supreme 380g	2	13.0	380.0
156836	2018-08-31	78	78135	76252	4	Dorito Corn Chp Supreme 380g	2	13.0	380.0
243002	2018-09-25	59	59078	54929	4	Dorito Corn Chp Supreme 380g	2	13.0	380.0
102409	2019-05-08	43	43184	39874	4	Dorito Corn Chp Supreme 380g	2	13.0	380.0

246740 rows × 9 columns

In [99]:

```
# 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.

In [103…

```
# 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
```

Out[103…

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PACK_SIZE	BRAND_NAME
40783	2018-09-25	97	97067	96696	38	Infuzions Mango Chutny Papadums 70g	2	4.8	70.0	Infuzions
222256	2018-12-16	114	114138	117726	38	Infuzions Mango Chutny Papadums 70g	2	4.8	70.0	Infuzions
261068	2019-03-23	103	103109	103191	38	Infuzions Mango Chutny Papadums 70g	1	2.4	70.0	Infuzions
154268	2019-04-15	47	47110	42492	38	Infuzions Mango Chutny Papadums 70g	2	4.8	70.0	Infuzions
76489	2018-09-29	164	164110	164477	38	Infuzions Mango Chutny Papadums 70g	2	4.8	70.0	Infuzions
...	...	...	...	...	...	...	...	...	...	...
20302	2019-04-02	104	104151	104652	14	Smiths Crnkle Chip Orgnl Big Bag 380g	2	11.8	380.0	Smiths
178013	2018-12-05	119	119152	122742	4	Dorito Corn Chp Supreme 380g	2	13.0	380.0	Dorito

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PACK_SIZE	BRAND_NAME
156836	2018-08-31	78	78135	76252	4	Dorito Corn Chp Supreme 380g	2	13.0	380.0	Dorito
243002	2018-09-25	59	59078	54929	4	Dorito Corn Chp Supreme 380g	2	13.0	380.0	Dorito
102409	2019-05-08	43	43184	39874	4	Dorito Corn Chp Supreme 380g	2	13.0	380.0	Dorito

246740 rows × 10 columns

In [105…

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

Out[105…

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

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

In [107…

```
# 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()
```

Out[107…

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

In [109…

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

Out[109…

	LYLTY_CARD_NBR	LIFESTAGE	PREMIUM_CUSTOMER
0	1000	YOUNG SINGLES/COUPLES	Premium
1	1002	YOUNG SINGLES/COUPLES	Mainstream
2	1003	YOUNG FAMILIES	Budget
3	1004	OLDER SINGLES/COUPLES	Mainstream
4	1005	MIDAGE SINGLES/COUPLES	Mainstream

In [111…

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

In [113…

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

Out[113…

	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

In [115…

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

Out[115…

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

In [117…

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

Out[117…

```
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.

In [119…

```
# 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_CARD_NBR'))
full_df = full_df.reset_index()
full_df = full_df.sort_values(by='DATE').reset_index(drop=True)
full_df
```

Out[119…

	LYLTY_CARD_NBR	DATE	STORE_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PACK_SIZE	BRAND_NAME
0	47288	2018-07-01	47	42735	97	RRD Salt & Vinegar 165g	2	6.0	165.0	Red Rock Deli
1	112065	2018-07-01	112	114104	87	Infuzions BBQ Rib Prawn Crackers 110g	2	7.6	110.0	Infuzions
2	28024	2018-07-01	28	24672	113	Twisties Chicken270g	2	9.2	270.0	Twisties
3	186028	2018-07-01	186	188435	52	Grain Waves Sour Cream&Chives 210G	2	7.2	210.0	Grain Waves
4	219068	2018-07-01	219	218409	88	Kettle Honey Soy Chicken 175g	2	10.8	175.0	Kettle
...	...	...	...	...	...	...	...	...	...	...
246735	67038	2019-06-30	67	64039	8	Smiths Crinkle Cut Chips Original 170g	2	5.8	170.0	Smiths
246736	272074	2019-06-30	272	269737	60	Kettle Tortilla ChpsFeta&Garlic 150g	2	9.2	150.0	Kettle
246737	172204	2019-06-30	172	174020	20	Doritos Cheese Supreme 330g	2	11.4	330.0	Doritos
246738	221140	2019-06-30	221	220611	68	Pringles Chicken Salt Crips 134g	1	3.7	134.0	Pringles
246739	249288	2019-06-30	249	251195	99	Pringles Sthrn FriedChicken 134g	1	3.7	134.0	Pringles

246740 rows × 12 columns

In [121…
```
# Check for nulls in the full dataset
full_df.isnull().values.any()
```

Out[121…
```
False
```

In [123…
```
# 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.

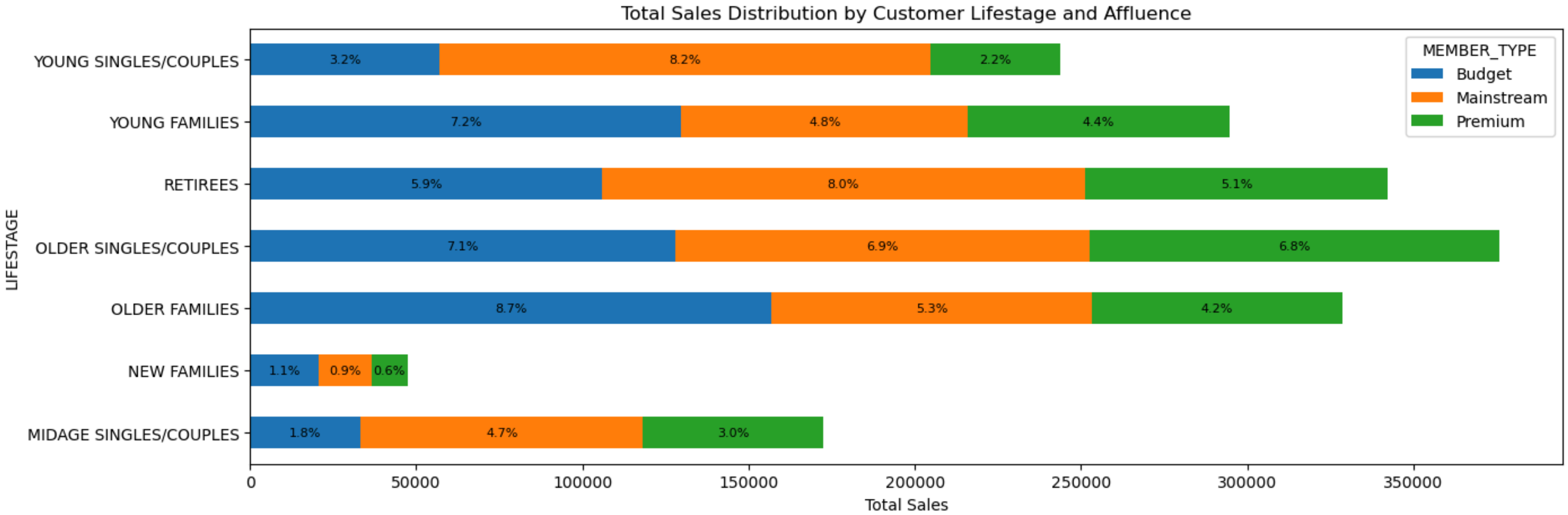
In [125…
```
# calculate total sales by lifestage and member type and generate a list
total_sales_cust = full_df.groupby(['LIFESTAGE','MEMBER_TYPE'], as_index = False)['TOT_SALES'].agg(['sum'])
total_sales_cust = total_sales_cust.rename(columns={'sum': 'sum_tot_sales'})
total_sales_cust.sort_values(by = "sum_tot_sales", ascending = False)
```

Out[125…

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

In [129…

```
# Calculate total sales
total_sales = full_df['TOT_SALES'].sum()

# Plot a breakdown of the total sales by lifestage and member type
total_sales_breakdown = (
    full_df.groupby(['LIFESTAGE', 'MEMBER_TYPE'])['TOT_SALES']
    .agg(['sum', 'mean'])
    .unstack('MEMBER_TYPE', fill_value=0)
)

# Plotting the stacked bar chart
ax = total_sales_breakdown['sum'].plot(kind='barh', stacked=True, figsize=(15, 5))

# Add percentage labels for each bar
for rect in ax.patches:
    width = rect.get_width()
    label = width / total_sales * 100  # Calculate percentage
    x = rect.get_x()
    y = rect.get_y()

    # Position the labels inside each bar
    label_text = f'{label:.1f}%'
    label_x = x + width / 2
    label_y = y + rect.get_height() / 2

    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 Distribution 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.

In [131…

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

Out[131…

```
True
```

In [133…

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

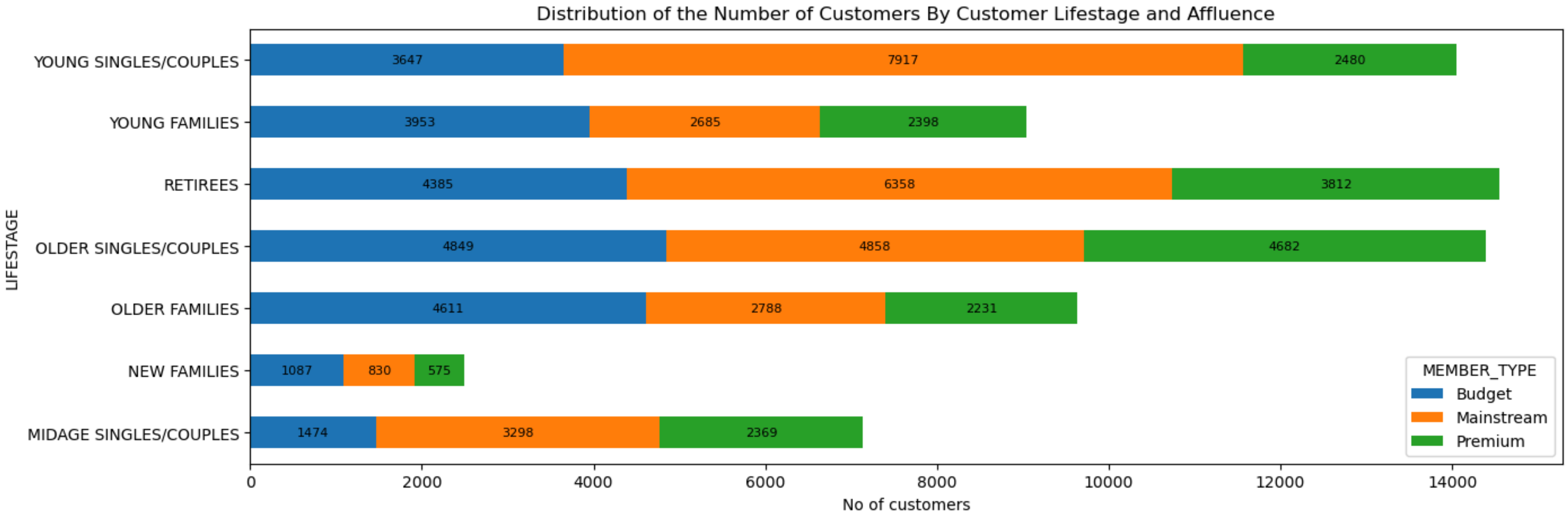
Out[133…

```
False
```

In [135…

```
# 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()
```

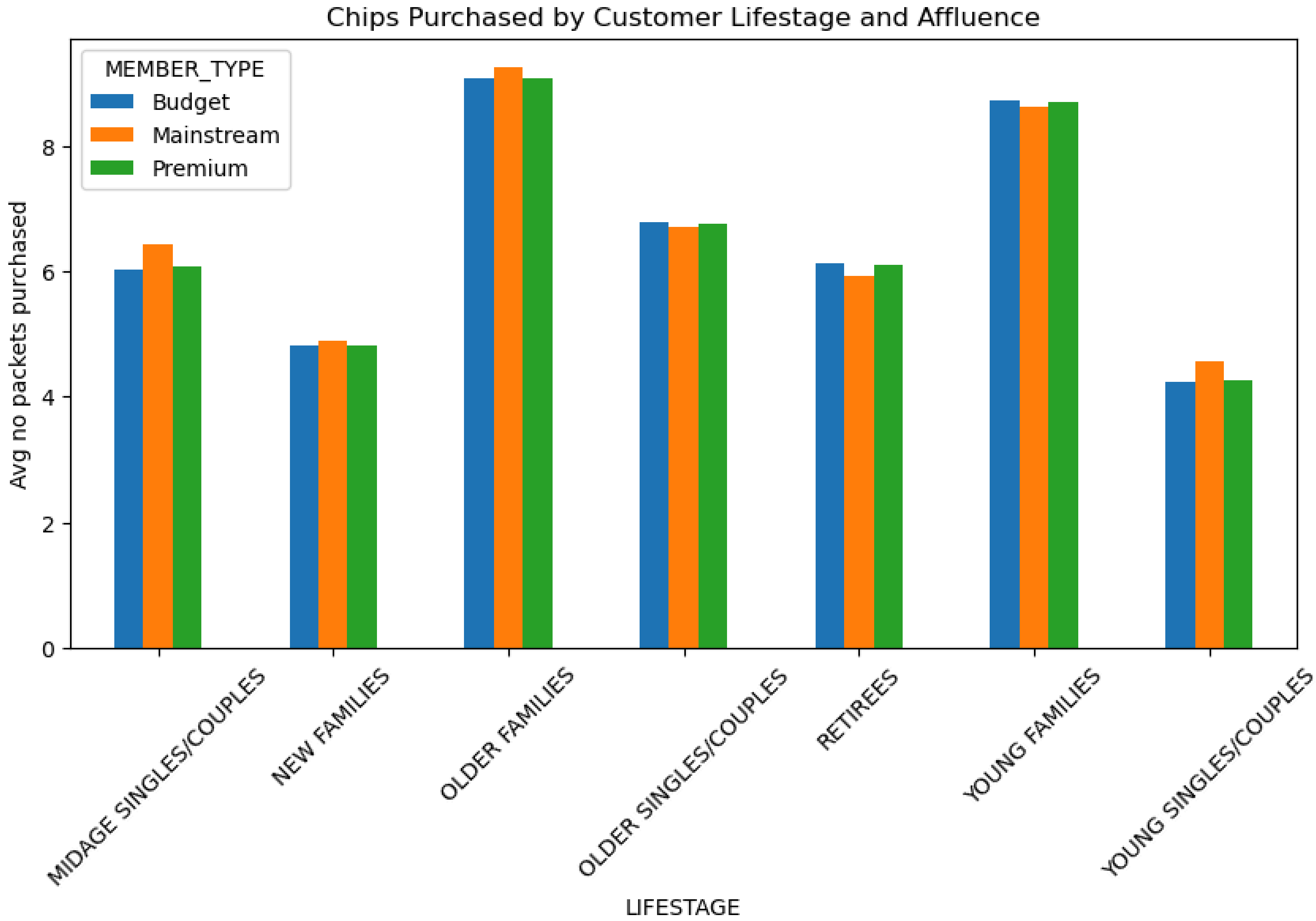


There are more Young singles/couples - mainstream and Retirees - mainstream who buy chips. This contributes to there being more sales to these customer segments but this is not a major driver for the Older families - budget segment. We can then take a look at the total and average units of chips bought per customer by LIFESTAGE and MEMBER_TYPE.

In [137…

```
# 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'])[
```

```
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()
```



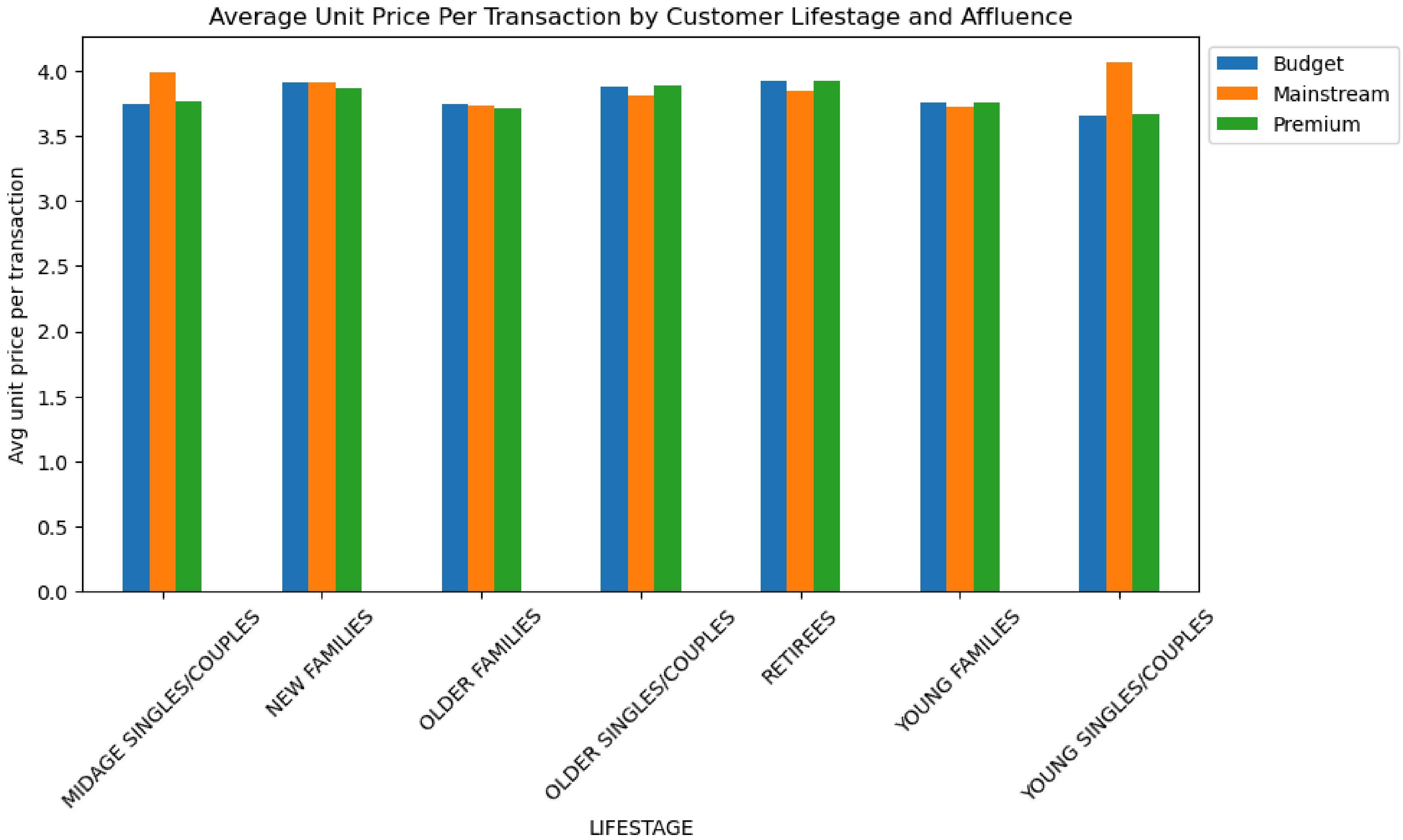
Older families and young families in general buy more chips per customer. We can also investigate the average price per unit sold by LIFESTAGE and MEMBER_TYPE.

In [139…

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

In [143…

```
# Calculate the average unit price per transaction by LIFESTAGE and MEMBER_TYPE
avg_priceperunit = (
    full_df.groupby(['LIFESTAGE', 'MEMBER_TYPE'])['UNIT_PRICE']
    .mean()
    .unstack('MEMBER_TYPE', fill_value=0)
)

# Plotting the bar chart
ax = avg_priceperunit.plot(kind='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.

In [145…

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

# Identify the groups to test the hypthesis with
```

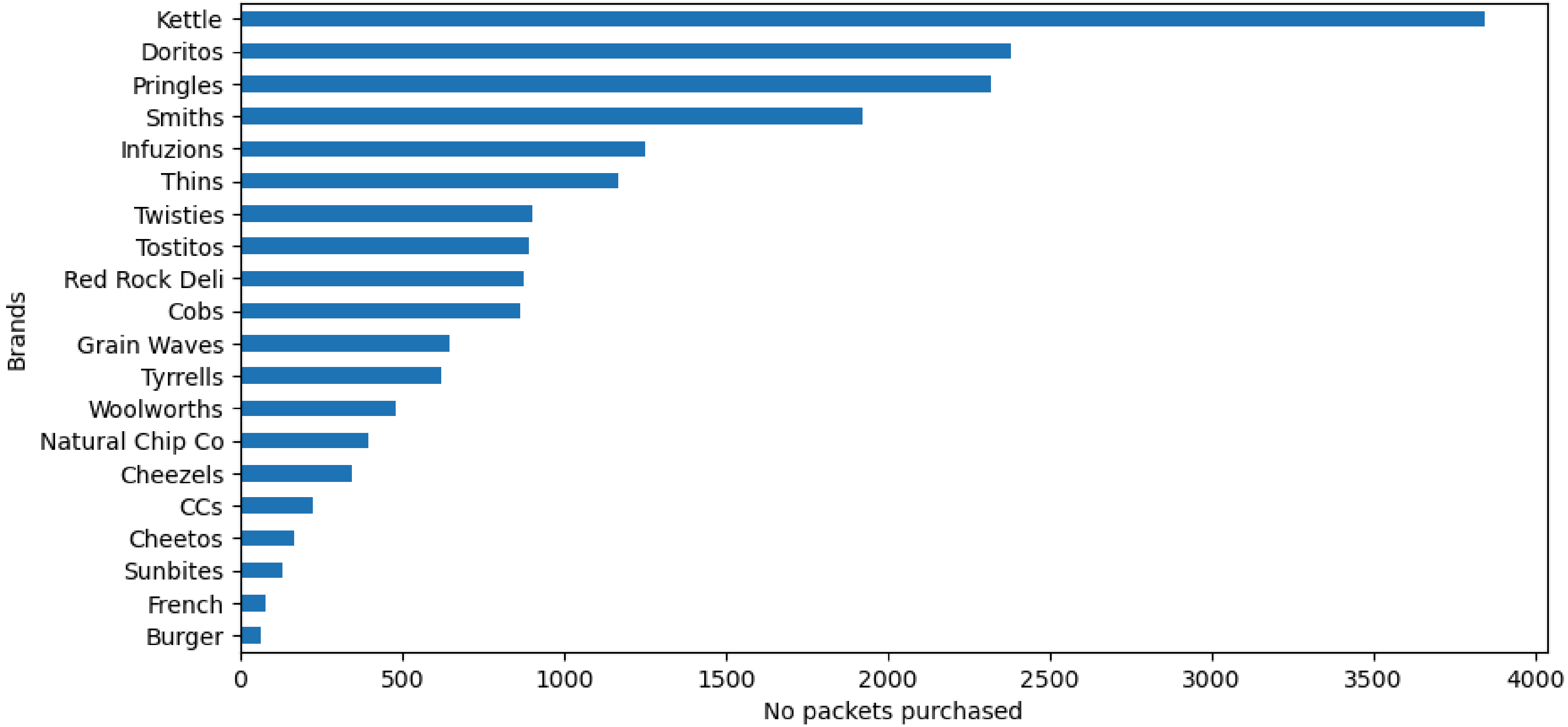
```
mainstream = full_df["MEMBER_TYPE"] == "Mainstream"
young_midage = (full_df["LIFESTAGE"] == "MIDAGE SINGLES/COUPLES") | (full_df["LIFESTAGE"] == "YOUNG SINGLES/COUPLES")
premium_budget = full_df["MEMBER_TYPE"] != "Mainstream"

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)
```

```
6.967354232991983e-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.

In [147…

```
# 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()
```



In [149…
```
temp = full_df.copy()
temp["group"] = temp["LIFESTAGE"] + ' - ' + temp['MEMBER_TYPE']
```

In [151…
```
groups = pd.get_dummies(temp["group"])
brands = pd.get_dummies(temp["BRAND_NAME"])
groups_brands = groups.join(brands)
groups_brands
```

Out[151…

	MIDAGE SINGLES/COUPLES - Budget	MIDAGE SINGLES/COUPLES - Mainstream	MIDAGE SINGLES/COUPLES - Premium	NEW FAMILIES - Budget	NEW FAMILIES - Mainstream	NEW FAMILIES - Premium	OLDER FAMILIES - Budget	OLDER FAMILIES - Mainstream	OLDER FAMILIES - Premium	SI
0	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	True	False	
2	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	...	
246735	False	False	False	False	False	False	False	False	False	
246736	False	False	False	False	False	False	False	False	False	
246737	False	False	False	False	False	False	False	True	False	
246738	False	False	False	False	False	False	False	False	False	
246739	False	False	False	False	False	False	False	False	False	

246740 rows × 41 columns

In [161…

```
import mlxtend
```

In [163…

```
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

In [165…

```
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)
```

In [167…
```
set_temp = temp["group"].unique()
rules[rules["antecedents"].apply(lambda x: list(x)).apply(lambda x: x in set_temp)]
```

Out[167…

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
40	(YOUNG SINGLES/COUPLES - Mainstream)	(Kettle)	0.079209	0.167334	0.015579	0.196684	1.175400	0.002325	1.036537	0.162062
0	(MIDAGE SINGLES/COUPLES - Mainstream)	(Kettle)	0.044966	0.167334	0.008657	0.192519	1.150508	0.001132	1.031190	0.136978
23	(RETIREES - Budget)	(Kettle)	0.057652	0.167334	0.010505	0.182214	1.088926	0.000858	1.018196	0.086660
32	(RETIREES - Premium)	(Kettle)	0.049591	0.167334	0.008981	0.181105	1.082296	0.000683	1.016816	0.080006
13	(OLDER SINGLES/COUPLES - Budget)	(Kettle)	0.069596	0.167334	0.012422	0.178488	1.066658	0.000776	1.013578	0.067167
21	(OLDER SINGLES/COUPLES - Premium)	(Kettle)	0.067115	0.167334	0.011944	0.177959	1.063495	0.000713	1.012925	0.064000
26	(RETIREES - Mainstream)	(Kettle)	0.080935	0.167334	0.013723	0.169554	1.013269	0.000180	1.002674	0.014248
16	(OLDER SINGLES/COUPLES - Mainstream)	(Kettle)	0.069146	0.167334	0.011490	0.166168	0.993034	-0.000081	0.998602	-0.007479
34	(YOUNG FAMILIES - Budget)	(Kettle)	0.071991	0.167334	0.011117	0.154422	0.922837	-0.000930	0.984730	-0.082654
4	(OLDER FAMILIES - Budget)	(Kettle)	0.087193	0.167334	0.013455	0.154318	0.922216	-0.001135	0.984609	-0.084586
10	(OLDER FAMILIES - Mainstream)	(Kettle)	0.053664	0.167334	0.008183	0.152481	0.911237	-0.000797	0.982475	-0.093327
8	(OLDER FAMILIES - Budget)	(Smiths)	0.087193	0.123016	0.011948	0.137027	1.113895	0.001222	1.016236	0.112016

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
36	(YOUNG FAMILIES - Budget)	(Smiths)	0.071991	0.123016	0.009459	0.131397	1.068126	0.000603	1.009648	0.068729
38	(YOUNG SINGLES/COUPLES - Mainstream)	(Doritos)	0.079209	0.102229	0.009642	0.121725	1.190712	0.001544	1.022198	0.173944
18	(OLDER SINGLES/COUPLES - Mainstream)	(Smiths)	0.069146	0.123016	0.008389	0.121329	0.986288	-0.000117	0.998080	-0.014715
30	(RETIREES - Mainstream)	(Smiths)	0.080935	0.123016	0.009593	0.118528	0.963514	-0.000363	0.994908	-0.039572
42	(YOUNG SINGLES/COUPLES - Mainstream)	(Pringles)	0.079209	0.101735	0.009382	0.118451	1.164310	0.001324	1.018962	0.153262
14	(OLDER SINGLES/COUPLES - Budget)	(Smiths)	0.069596	0.123016	0.008146	0.117051	0.951509	-0.000415	0.993244	-0.051929
28	(RETIREES - Mainstream)	(Pringles)	0.080935	0.101735	0.008523	0.105308	1.035124	0.000289	1.003994	0.036920
24	(RETIREES - Mainstream)	(Doritos)	0.080935	0.102229	0.008466	0.104607	1.023260	0.000192	1.002656	0.024733
2	(OLDER FAMILIES - Budget)	(Doritos)	0.087193	0.102229	0.008235	0.094450	0.923907	-0.000678	0.991410	-0.082760
6	(OLDER FAMILIES - Budget)	(Pringles)	0.087193	0.101735	0.008089	0.092777	0.911949	-0.000781	0.990126	-0.095657

In [169…

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

Out[169…

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
40	(YOUNG SINGLES/COUPLES - Mainstream)	(Kettle)	0.079209	0.167334	0.015579	0.196684	1.175400	0.002325	1.036537	0.162062
38	(YOUNG SINGLES/COUPLES - Mainstream)	(Doritos)	0.079209	0.102229	0.009642	0.121725	1.190712	0.001544	1.022198	0.173944
42	(YOUNG SINGLES/COUPLES - Mainstream)	(Pringles)	0.079209	0.101735	0.009382	0.118451	1.164310	0.001324	1.018962	0.153262

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.

In [171…

```
# find the target rating proportion
target_segment = young_mainstream["BRAND_NAME"].value_counts().sort_values(ascending = True).rename_axis('BRANDS').reset_index
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(ascending = True).rename_axis('BRANDS').reset_index(name
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_CARD_NBR'))
brand_proportions = brand_proportions.reset_index()
brand_proportions['affinity'] = brand_proportions['target']/brand_proportions['other']
brand_proportions.sort_values(by = 'affinity', ascending = False)
```

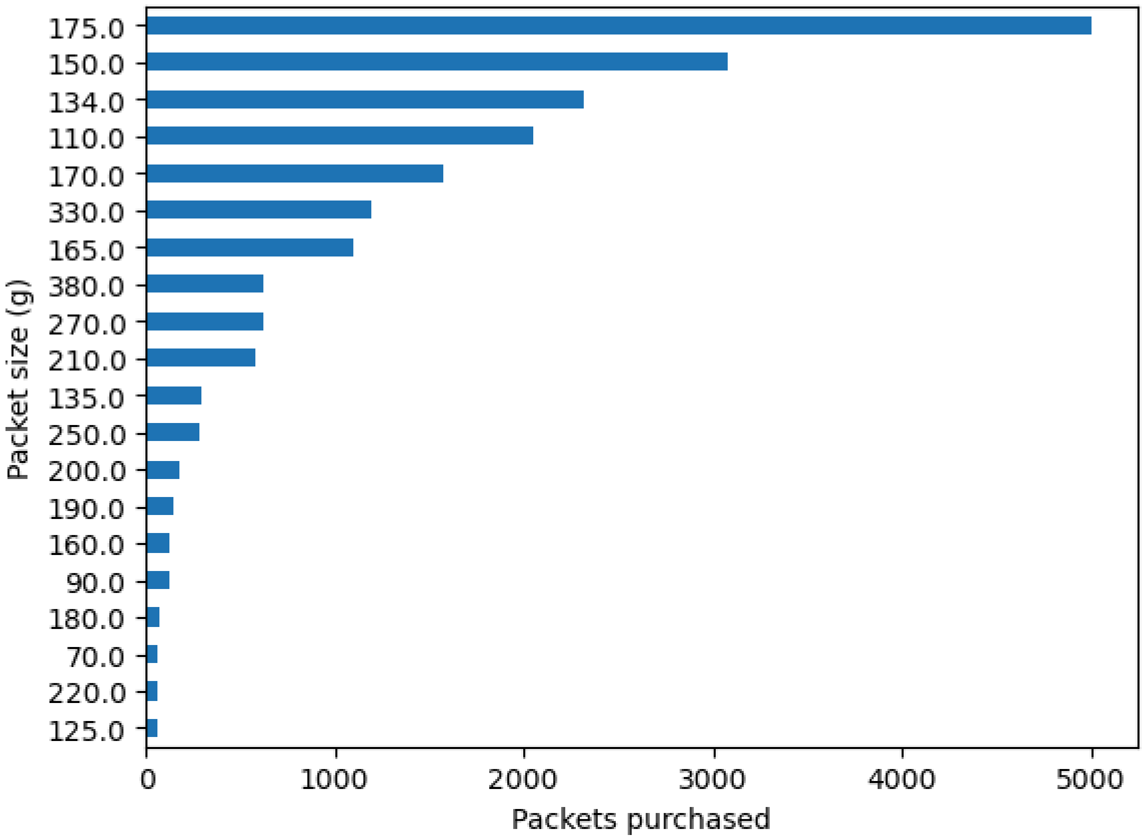
Out[171…

	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.

In [173…

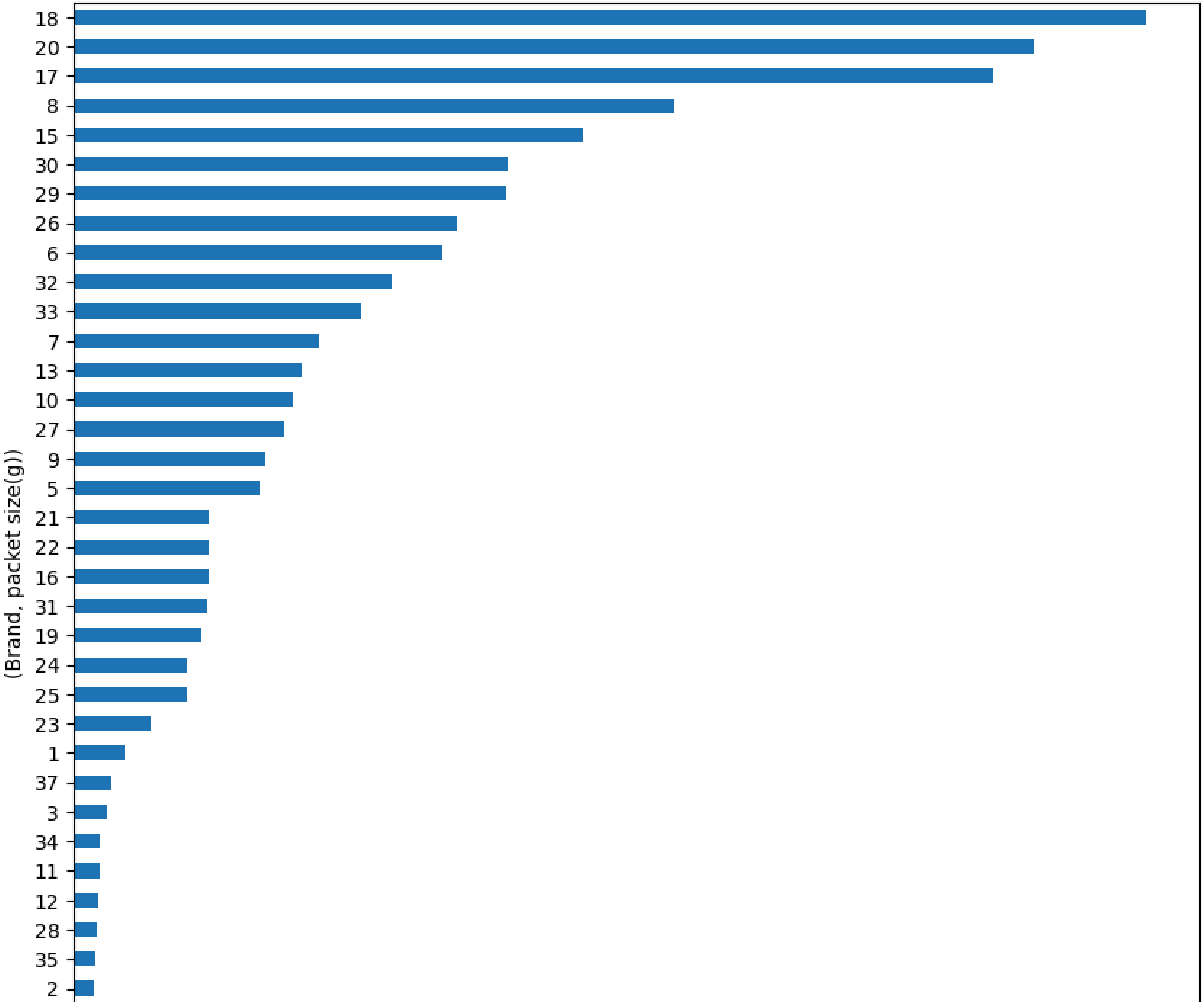
```
# Plot the distribution of the packet sizes for a general indication of what it most popular.
young_mainstream = full_df.loc[full_df['LIFESTAGE'] == "YOUNG SINGLES/COUPLES"]
young_mainstream = young_mainstream.loc[young_mainstream['MEMBER_TYPE'] == "Mainstream"]
ax = young_mainstream["PACK_SIZE"].value_counts().sort_values(ascending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

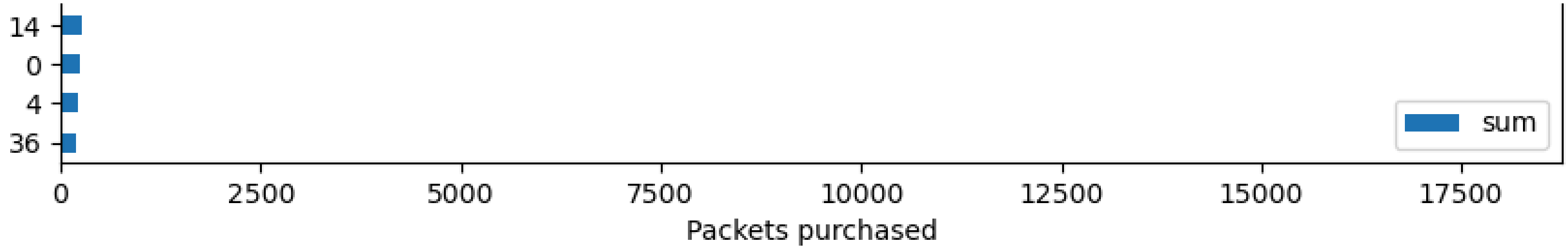


In [185…

```
# 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()
```


(Brand, packet size(g))
18
20
17
8
15
30
29
26
6
32
33
7
13
10
27
9
5
21
22
16
31
19
24
25
23
1
37
3
34
11
12
28
35
2



In [179…

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

Out[179…

	MIDAGE SINGLES/COUPLES - Budget	MIDAGE SINGLES/COUPLES - Mainstream	MIDAGE SINGLES/COUPLES - Premium	NEW FAMILIES - Budget	NEW FAMILIES - Mainstream	NEW FAMILIES - Premium	OLDER FAMILIES - Budget	OLDER FAMILIES - Mainstream	OLDER FAMILIES - Premium	SI
0	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	True	False	
2	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	...	
246735	False	False	False	False	False	False	False	False	False	
246736	False	False	False	False	False	False	False	False	False	
246737	False	False	False	False	False	False	False	True	False	
246738	False	False	False	False	False	False	False	False	False	
246739	False	False	False	False	False	False	False	False	False	

246740 rows × 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)]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.

In [191…

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

Out[191…

	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.

In [ ]:

```

```