In [81]:
```
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
```

In [3]:
```
customerdata = pd.read_csv('QVI_purchase_behaviour.csv')
transactiondata = pd.read_excel('QVI_transaction_data.xlsx')
```

In [4]:
```
trans_df = transactiondata.copy()
trans_df
```

Out[4]:

	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

In [5]:
```
trans_df['DATE'] = pd.to_datetime(trans_df['DATE'], unit='D', origin='1899-12-30')
print(trans_df['DATE'].dtype)
```

```
datetime64[ns]
```

In [6]:
```
trans_df['PROD_NAME'].unique()
```

Out[6]:
```
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',
```

In [7]:
```
prod_name = trans_df['PROD_NAME'].str.replace(r'[0-9]+[gG]','');

prod_name = prod_name.str.replace(r'&',' ');
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_12636\640731619.py:1: FutureWarning: The default va
lue of regex will change from True to False in a future version.
  prod_name = trans_df['PROD_NAME'].str.replace(r'[0-9]+[gG]','');
```

In [8]:
```
word_counts = pd.Series(' '.join(prod_name).split()).value_counts()

with pd.option_context('display.max_rows', None):
  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
Popd	9693
Cobs	9693
Tortilla	9580
Tostitos	9471
Twisties	9454
BBQ	9434
Sensations	9429
Lime	9347
Paso	9324
Dip	9324
Old	9324
El	9324
Tomato	7669
Thinly	7507
Tyrrells	6442
And	6373
Tangy	6332
SourCream	6296
Waves	6272
Grain	6272
Lightly	6248
Salted	6248
Soy	6121
Onion	6116
Natural	6050
Mild	6048
Deli	5885
Rock	5885
Red	5885
Thai	4737
Burger	4733
Swt	4718
Honey	4661
Nacho	4658
Potato	4647
Cheezels	4603
Garlic	4572
CCs	4551
Woolworths	4437
Pesto	3304
Basil	3304
Mozzarella	3304
Chili	3296
ChpsHny	3296
Jlpno	3296

Sr/Cream	3269
Swt/Chlli	3269
Ched	3268
Pot	3257
Splash	3252
Of	3252
PotatoMix	3242
SweetChili	3242
Bag	3233
Big	3233
Orgnl	3233
Crnkle	3233
Spicy	3229
Hot	3229
Camembert	3219
Fig	3219
Barbeque	3210
Jalapeno	3204
Mexican	3204
Light	3188
Chp	3185
Dorito	3185
Spcy	3177
Rib	3174
Prawn	3174
Crackers	3174
Southern	3172
Crm	3159
ChpsBtroot	3146
Ricotta	3146
Smoked	3145
Chipotle	3145
Crnchers	3144
Infzns	3144
Gcamole	3144
Crn	3144
ChpsFeta	3138
Herbs	3134
Veg	3134
Strws	3134
Siracha	3127
Chnky	3125
Tom	3125
Ht	3125
Mexicana	3115
Mystery	3114
Flavour	3114
Seasonedchicken	3114
Med	3114
Crips	3104
Slt	3095
Vingar	3095
FriedChicken	3083
Sthrn	3083
Maple	3083
Rings	3080
ChipCo	3010
SR	2984
Smith	2963
Chs	2960
S/Cream	2934
Cheetos	2927
Medium	2879
French	2856
Mstrd	1576
Cheddr	1576
Snbts	1576
Whlgrn	1576
Spce	1572
Hrb	1572
Tmato	1572
Co	1572
Vinegr	1550

```
Tasty              1539
Belly              1526
Pork               1526
Rst                1526
Slow               1526
Roast              1519
N                  1512
Mac                1512
Mango              1507
Chutny             1507
Papadums           1507
Coconut            1506
Sauce              1503
Snag               1503
Truffle            1498
Sp                 1498
Barbecue           1489
Stacked            1487
OnionStacked       1483
Bacon              1479
Balls              1479
Pepper             1473
D/Style            1469
GrnWves            1468
Compny             1468
SeaSalt            1468
Btroot             1468
Jam                1468
Plus               1468
Chli               1461
Chckn              1460
Hony               1460
Mzzrlla            1458
Steak              1455
Chimuchurri        1455
Box                1454
Bolognese          1451
Puffs              1448
Originl            1441
saltd              1441
CutSalt/Vinegr     1440
OnionDip           1438
Chikn              1434
Aioli              1434
Frch/Onin          1432
Whlegrn            1432
Sunbites           1432
Pc                 1431
NCC                1419
Garden             1419
Fries              1418
dtype: int64
```

In [9]:
```
trans_df = trans_df[trans_df['PROD_NAME'].str.contains(r"[Ss]alsa") == False]
trans_df.shape
```

Out[9]:
```
(246742, 8)
```

In [10]:
```
trans_df.describe()
```

Out[10]:

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

In [11]:
```
trans_df.isnull().sum()
```

Out[11]:
```
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 [12]:
```
trans_df.loc[trans_df['PROD_QTY'] == 200.0]
```

Out[12]:

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

In [13]:
```
trans_df.loc[trans_df['LYLTY_CARD_NBR'] == 226000]
```

Out[13]:

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

In [14]:
```
# Removing  this transaction because this is the only transaction with this high quantity
trans_df = trans_df[trans_df['LYLTY_CARD_NBR'] != 226000]
trans_df.shape
```

Out[14]:
```
(246740, 8)
```

In [15]:
```
trans_df.describe()
```

Out[15]:

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

In [16]:
```
count = trans_df.groupby(trans_df['DATE'].dt.date).size().reset_index(name = 'COUNT')
count.shape
```

Out[16]:
```
(364, 2)
```

In [17]:
```
trans_df.PROD_QTY.value_counts()
```

Out[17]:
```
2    220070
1     25476
5       415
3       408
4       371
Name: PROD_QTY, dtype: int64
```

In [18]:
```
trans_df.sort_values(by='DATE')
```

Out[18]:

	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

In [19]:
```
# 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[19]:
```
DatetimeIndex(['2018-12-25'], dtype='datetime64[ns]', freq=None)
```

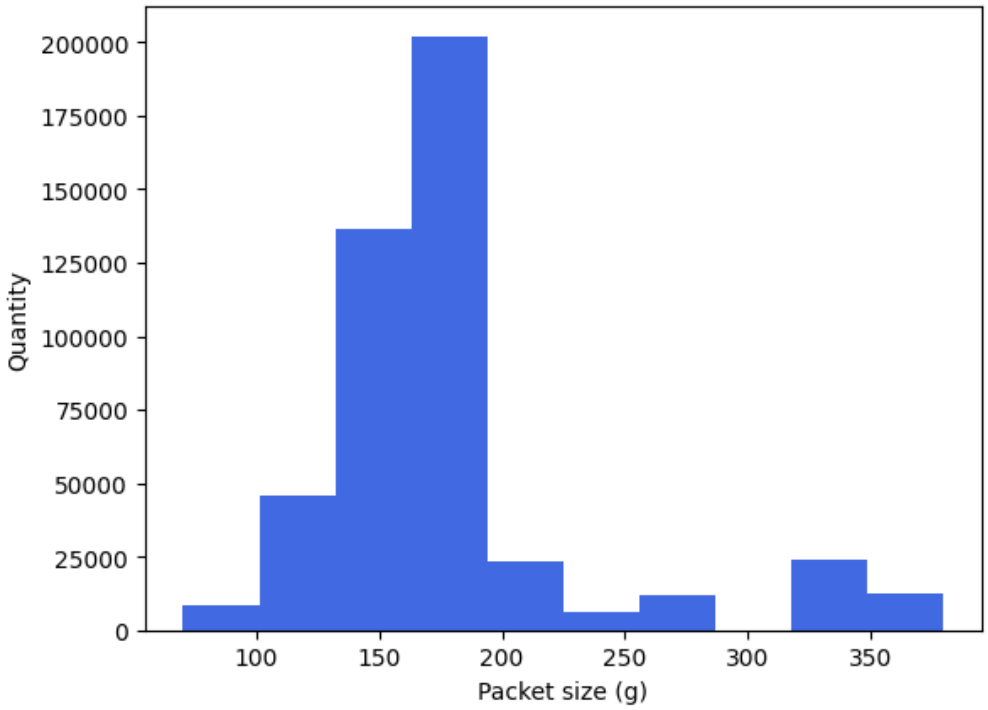
In [20]:
```
# Add a new column to data with packet sizes and extract sizes from product name column
trans_df.insert(8, "PACK_SIZE", trans_df['PROD_NAME'].str.extract('(\d+)').astype(float), True)

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

Out[20]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PACK_
40783	2018-09-25	97	97067	96696	38	Infuzions Mango Chutny Papadums 70g	2	4.8	
42461	2019-05-05	110	110030	111890	38	Infuzions Mango Chutny Papadums 70g	2	4.8	
176183	2018-12-30	82	82183	81660	38	Infuzions Mango Chutny Papadums 70g	2	4.8	
227309	2018-12-03	236	236091	239098	38	Infuzions Mango Chutny Papadums 70g	2	4.8	
42418	2018-11-05	109	109217	111470	38	Infuzions Mango Chutny Papadums 70g	2	4.8	
...	...	...	...	...	...	...	...	...	
192034	2019-03-12	100	100121	99145	4	Dorito Corn Chp Supreme 380g	2	13.0	3
255797	2019-01-19	235	235098	238018	4	Dorito Corn Chp Supreme 380g	2	13.0	3
233814	2019-01-24	151	151102	149810	4	Dorito Corn Chp Supreme 380g	1	6.5	3
131573	2018-07-09	213	213087	212416	4	Dorito Corn Chp Supreme 380g	2	13.0	3
102409	2019-05-08	43	43184	39874	4	Dorito Corn Chp Supreme 380g	2	13.0	3

246740 rows × 9 columns

In [22]:
```
# 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'], color= "royalblue");
plt.xlabel('Packet size (g)');
plt.ylabel('Quantity');
```



In [23]:
```
# 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[23]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PACK
0	2018-10-17	1	1000	1	5	Natural Chip Compny SeaSalt175g	2	6.0	
1	2019-05-14	1	1307	348	66	CCs Nacho Cheese 175g	3	6.3	
2	2019-05-20	1	1343	383	61	Smiths Crinkle Cut Chips Chicken 170g	2	2.9	
3	2018-08-17	2	2373	974	69	Smiths Chip Thinly S/Cream&Onion 175g	5	15.0	
4	2018-08-18	2	2426	1038	108	Kettle Tortilla ChpsHny&Jlpno Chili 150g	3	13.8	
...	...	...	...	...	...	...	...	...	
264831	2019-03-09	272	272319	270088	89	Kettle Sweet Chilli And Sour Cream 175g	2	10.8	
264832	2018-08-13	272	272358	270154	74	Tostitos Splash Of Lime 175g	1	4.4	
264833	2018-11-06	272	272379	270187	51	Doritos Mexicana 170g	2	8.8	
264834	2018-12-27	272	272379	270188	42	Doritos Corn Chip Mexican Jalapeno 150g	2	7.8	
264835	2018-09-22	272	272380	270189	74	Tostitos Splash Of Lime 175g	2	8.8	

246740 rows × 10 columns

In [25]:
```
# Unique Brand names
trans_df["BRAND_NAME"].unique()
```

Out[25]:
```
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.

In [26]:
```
# 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[26]:
```
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)
```

In [32]:
```
cust_df = customerdata.copy()
cust_df
```

Out[32]:

	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
...	...	...	...
72632	2370651	MIDAGE SINGLES/COUPLES	Mainstream
72633	2370701	YOUNG FAMILIES	Mainstream
72634	2370751	YOUNG FAMILIES	Premium
72635	2370961	OLDER FAMILIES	Budget
72636	2373711	YOUNG SINGLES/COUPLES	Mainstream

72637 rows × 3 columns

In [42]:
```
# Rename "PREMIUM_CUSTOMER" to "MEMBER_TYPE" for easier identification of the column data
cust_df = cust_df.rename(columns={'PREMIUM_CUSTOMER': 'MEMBER_TYPE'})
```

In [43]:
```
# Check the summary of the customer data
cust_df
```

Out[43]:

	LYLTY_CARD_NBR	LIFESTAGE	MEMBER_TYPE
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
...	...	...	...
72632	2370651	MIDAGE SINGLES/COUPLES	Mainstream
72633	2370701	YOUNG FAMILIES	Mainstream
72634	2370751	YOUNG FAMILIES	Premium
72635	2370961	OLDER FAMILIES	Budget
72636	2373711	YOUNG SINGLES/COUPLES	Mainstream

72637 rows × 3 columns

In [44]:
```
# Check the entries in the member type and lifestage columns
cust_df["MEMBER_TYPE"].unique()
```

Out[44]:
```
array(['Premium', 'Mainstream', 'Budget'], dtype=object)
```

In [45]:
```
cust_df["LIFESTAGE"].unique()
```

Out[45]:
```
array(['YOUNG SINGLES/COUPLES', 'YOUNG FAMILIES', 'OLDER SINGLES/COUPLES',
       'MIDAGE SINGLES/COUPLES', 'NEW FAMILIES', 'OLDER FAMILIES',
       'RETIREES'], dtype=object)
```

In [46]:
```
cust_df.isnull().sum()
```

Out[46]:
```
LYLTY_CARD_NBR    0
LIFESTAGE         0
MEMBER_TYPE       0
dtype: int64
```

In [47]:
```
# 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[47]:

	LYLTY_CARD_NBR	DATE	STORE_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PAC
0	21037	2018-07-01	21	17576	62	Pringles Mystery Flavour 134g	2	7.4	
1	25040	2018-07-01	25	21704	87	Infuzions BBQ Rib Prawn Crackers 110g	2	7.6	
2	59236	2018-07-01	59	55555	42	Doritos Corn Chip Mexican Jalapeno 150g	2	7.8	
3	271083	2018-07-01	271	268688	97	RRD Salt & Vinegar 165g	2	6.0	
4	65015	2018-07-01	65	61737	17	Kettle Sensations BBQ&Maple 150g	2	9.2	
...	...	...	...	...	...	...	...	...	
246735	48160	2019-06-30	48	44051	11	RRD Pc Sea Salt 165g	2	6.0	
246736	175371	2019-06-30	175	176890	40	Thins Chips Seasonedchicken 175g	2	6.6	
246737	203312	2019-06-30	203	203610	68	Pringles Chicken Salt Crips 134g	2	7.4	
246738	222003	2019-06-30	222	221524	17	Kettle Sensations BBQ&Maple 150g	2	9.2	
246739	55142	2019-06-30	55	49322	78	Thins Chips Salt & Vinegar 175g	2	6.6	

246740 rows × 12 columns

In [48]:
```
# Check for nulls in the full dataset
full_df.isnull().sum()
```

Out[48]:
```
LYLTY_CARD_NBR    0
DATE              0
STORE_NBR         0
TXN_ID            0
PROD_NBR          0
PROD_NAME         0
PROD_QTY          0
TOT_SALES         0
PACK_SIZE         0
BRAND_NAME        0
LIFESTAGE         0
MEMBER_TYPE       0
dtype: int64
```

In [49]:
```
full_df
```

Out[49]:

CARD_NBR	DATE	STORE_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PACK_SIZE	BRAND_N
21037	2018-07-01	21	17576	62	Pringles Mystery Flavour 134g	2	7.4	134.0	Pri
25040	2018-07-01	25	21704	87	Infuzions BBQ Rib Prawn Crackers 110g	2	7.6	110.0	Infu
59236	2018-07-01	59	55555	42	Doritos Corn Chip Mexican Jalapeno 150g	2	7.8	150.0	D
271083	2018-07-01	271	268688	97	RRD Salt & Vinegar 165g	2	6.0	165.0	Red Roc
65015	2018-07-01	65	61737	17	Kettle Sensations BBQ&Maple 150g	2	9.2	150.0	
...	...	...	...	...	...	...	...	...	
48160	2019-06-30	48	44051	11	RRD Pc Sea Salt 165g	2	6.0	165.0	Red Roc
175371	2019-06-30	175	176890	40	Thins Chips Seasonedchicken 175g	2	6.6	175.0	
203312	2019-06-30	203	203610	68	Pringles Chicken Salt Crips 134g	2	7.4	134.0	Pri
222003	2019-06-30	222	221524	17	Kettle Sensations BBQ&Maple 150g	2	9.2	150.0	
55142	2019-06-30	55	49322	78	Thins Chips Salt & Vinegar 175g	2	6.6	175.0	

2 columns

In [50]:
```
# 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 [51]:
```
# 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'].
total_sales_cust = total_sales_cust.rename(columns={'sum': 'sum_tot_sales'})
total_sales_cust.sort_values(by = "sum_tot_sales", ascending = False)
```

Out[51]:

		sum_tot_sales
LIFESTAGE	**MEMBER_TYPE**	
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

In [56]:
```
# Get the total sales
total_sales = full_df['TOT_SALES'].sum()
total_sales
```

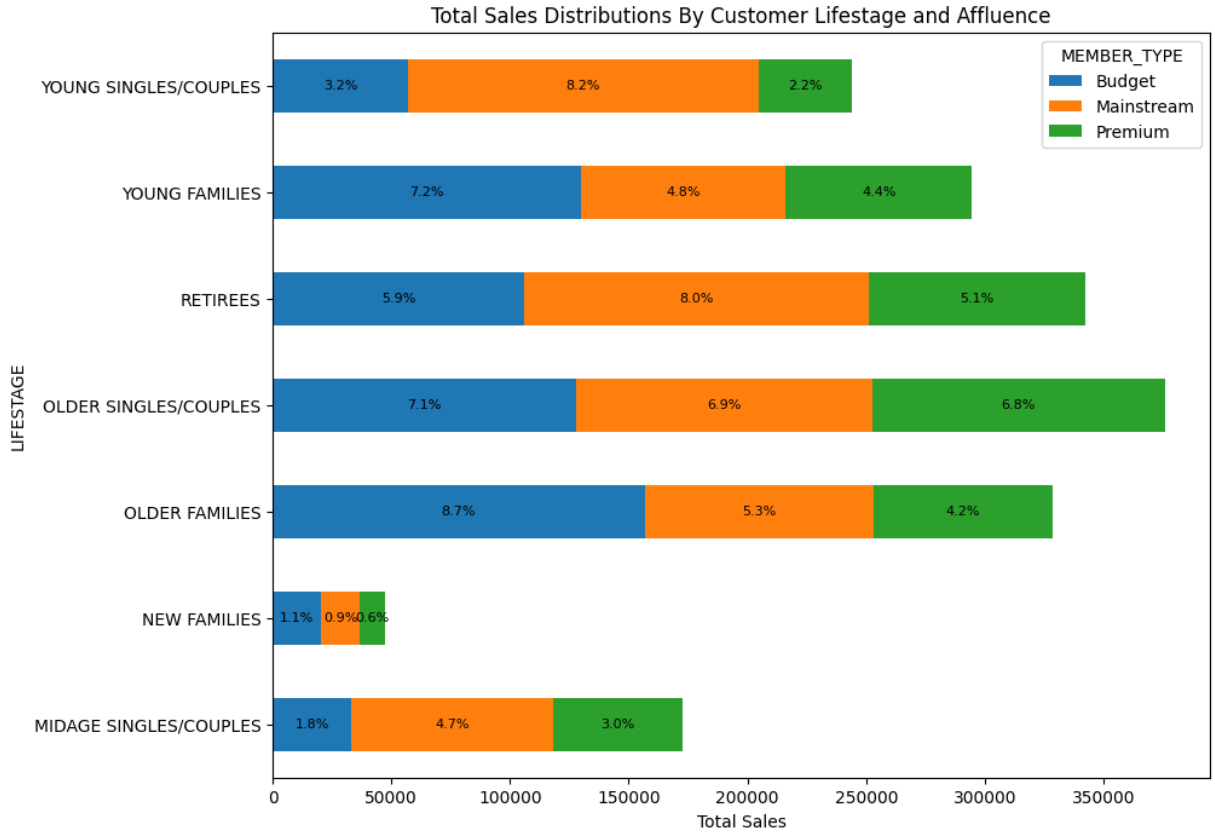
Out[56]: 1805177.7000000002

In [58]:
```
# Group by 'LIFESTAGE' and 'MEMBER_TYPE', then sum 'TOT_SALES'
sales_summary = full_df.groupby(['LIFESTAGE', 'MEMBER_TYPE'])['TOT_SALES'].sum().reset_index()
sales_summary
```

Out[58]:

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

In [62]:
```
# Pivot the DataFrame to have 'LIFESTAGE' as index, 'MEMBER_TYPE' as columns, and 'TOT_SALES' as
pivot_df = sales_summary.pivot(index='LIFESTAGE', columns='MEMBER_TYPE', values='TOT_SALES')

# Plotting
ax = pivot_df.plot(kind='barh', stacked=True, figsize=(10, 8))
ax.set_xlabel('Total Sales')
ax.set_title('Total Sales by Life Stage and Member Type')


# Iterate through the rectangles/bars in the chart
for rect in ax.patches:
    width = rect.get_width()  # Get the width of the bar
    label = (width / total_sales) * 100  # Calculate the percentage of the total sales

    # Calculate the center position of the bar for placing the label
    label_x = rect.get_x() + width / 2
    label_y = rect.get_y() + rect.get_height() / 2

    # Define the label text formatted to show only 1 decimal place
    label_text = f'{label:.1f}%'

    # Add text annotation to the bar if the width is significant enough to display the label
    if width > 0:  # You might adjust this threshold based on your data
        ax.text(label_x, label_y, label_text, ha='center', va='center', fontsize=8)

# Set labels and title
ax.set_xlabel("Total Sales")
ax.set_title('Total Sales Distributions By Customer Lifestage and Affluence')

# Display the plot
plt.show()
```



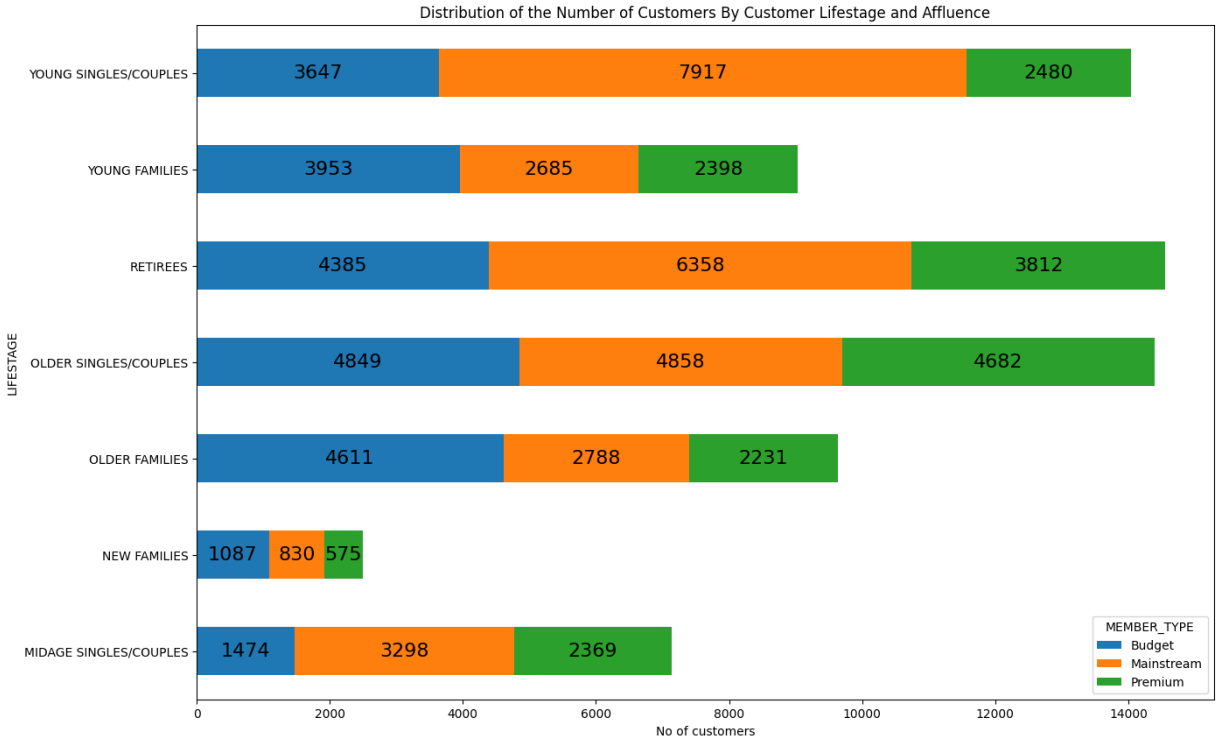
In [63]:
```
# Check all rows are unique in customer information
len(cust_df['LYLTY_CARD_NBR'].unique()) == cust_df.shape[0]
```

Out[63]: True

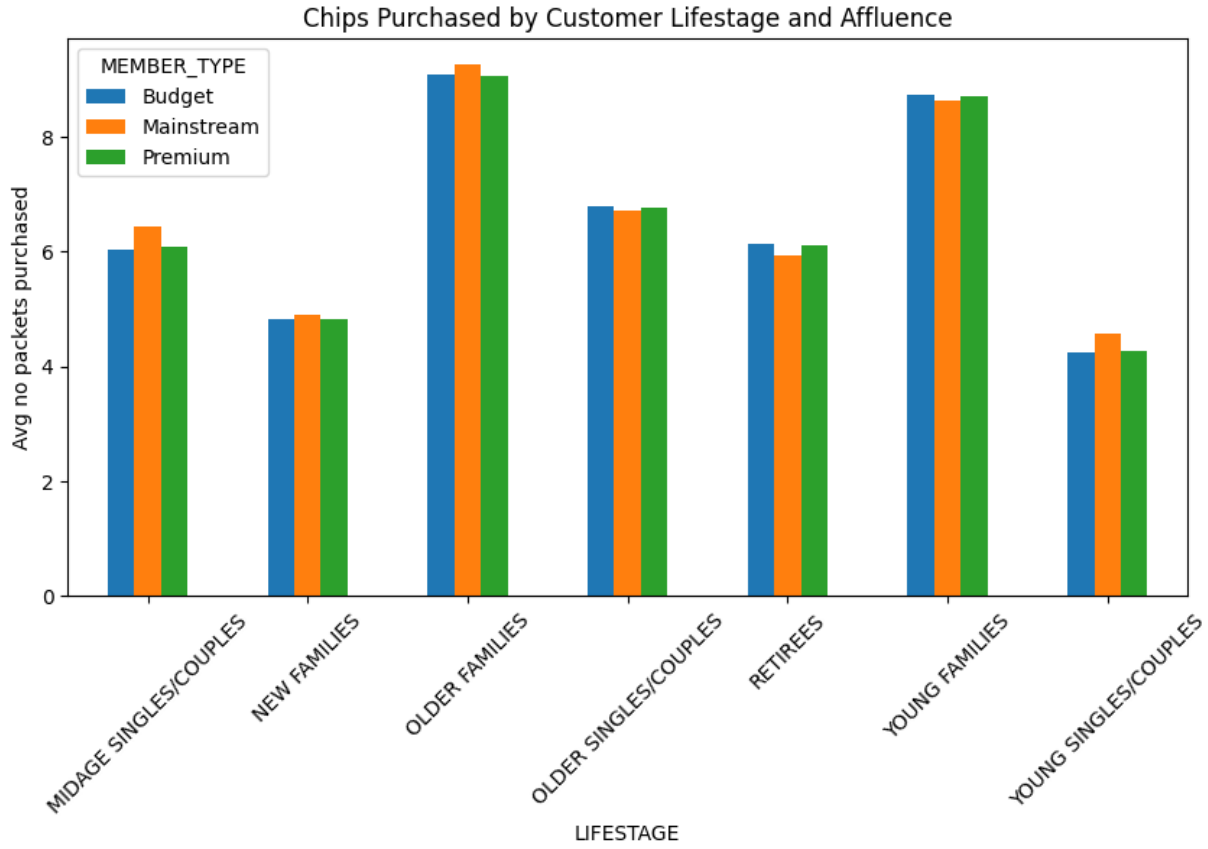
In [64]:
```
# Check if all customers made chip purchases.
len(cust_df['LYLTY_CARD_NBR'].unique()) == len(full_df['LYLTY_CARD_NBR'].unique())
```

Out[64]: False

In [70]:
```
# 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').un
ax = sum_customers.plot(kind='barh', stacked=True, figsize=(15, 10))

# 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=16)

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

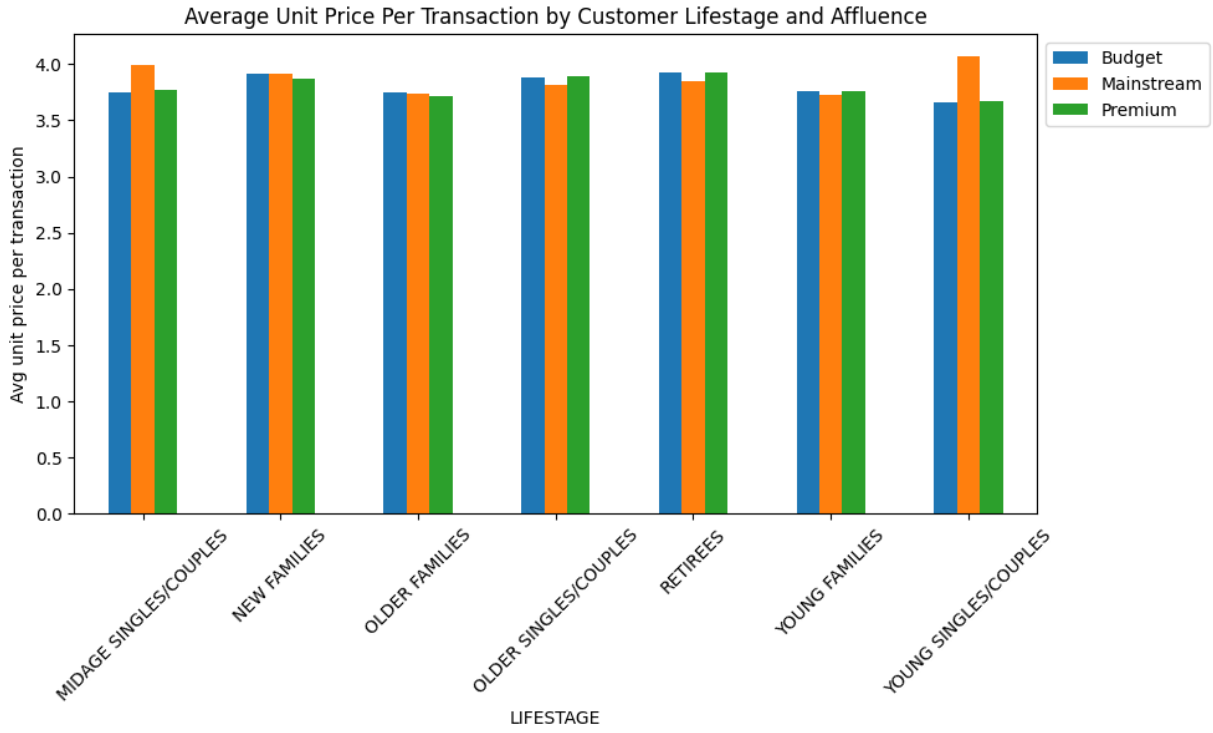


In [71]:
```
# Plot the average no of chip packets bought per customer by LIFESTAGE and MEMBER_TYPE.
no_packets_data = full_df.groupby(['LIFESTAGE','MEMBER_TYPE'])['PROD_QTY'].sum()/full_df.groupb
ax = no_packets_data.unstack('MEMBER_TYPE').fillna(0).plot.bar(stacked = False,figsize=(10, 5))
ax.set_ylabel("Avg no packets purchased")
ax.set_title('Chips Purchased by Customer Lifestage and Affluence')
plt.xticks(rotation=45)
plt.show()
```



In [72]:
```
# Create a column for the unit price of chips purchased per transaction
full_df['UNIT_PRICE'] = full_df['TOT_SALES']/full_df['PROD_QTY']
```

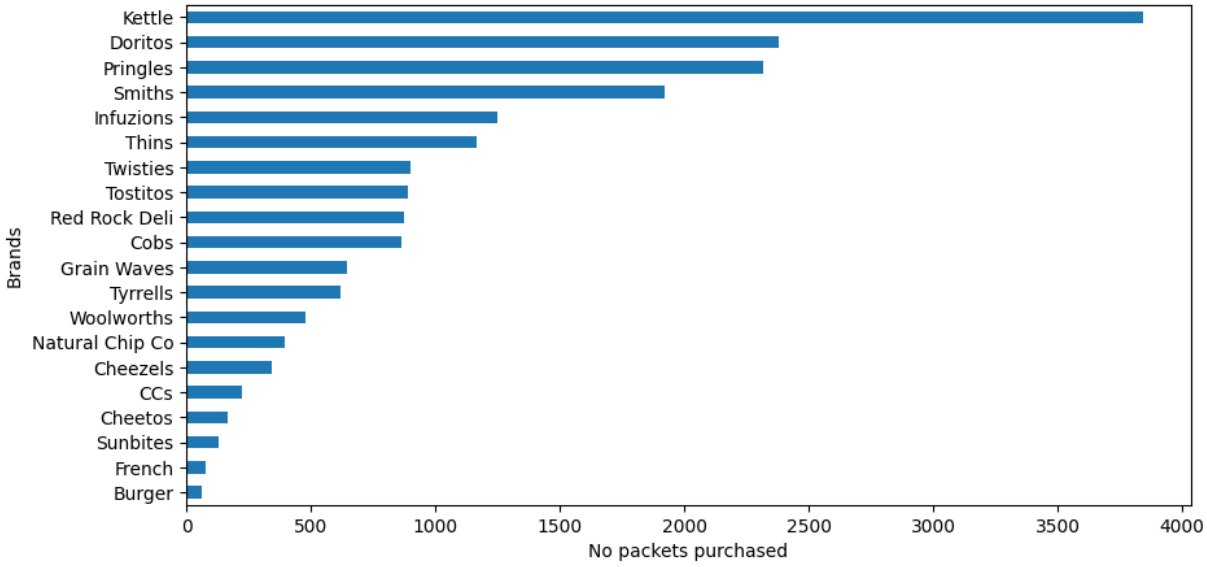
In [73]:
```
# Plot the distribution of the average unit price per transaction by LIFESTAGE and MEMBER_TYPE.
avg_priceperunit = full_df.groupby(['LIFESTAGE', 'MEMBER_TYPE'], as_index = False)['UNIT_PRICE'
ax = avg_priceperunit['mean'].plot.bar(stacked=False, figsize=(10, 5))
ax.set_ylabel("Avg unit price 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()
```



In [74]:
```
# Check the difference in the average price unit between the mainstream and premium/budget grou
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"] == "Y
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
```

In [75]:
```
# Create a visual of what brands young singles/couples are purchasing the most for a general in
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(figs
ax.set_xlabel("No packets purchased")
ax.set_ylabel("Brands")
plt.show()
```



In [76]:
```
temp = full_df.copy()
temp["group"] = temp["LIFESTAGE"] + ' - ' + temp['MEMBER_TYPE']
```

In [77]:
```
groups = pd.get_dummies(temp["group"])
brands = pd.get_dummies(temp["BRAND_NAME"])
groups_brands = groups.join(brands)
groups_brands
```

Out[77]:

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

246740 rows × 41 columns

In [82]:
```
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)
```

```
C:\Users\user\anaconda3\lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:109: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
```

In [83]:
```
set_temp = temp["group"].unique()
rules[rules["antecedents"].apply(lambda x: list(x)).apply(lambda x: x in set_temp)]
```

Out[83]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
40	(YOUNG SINGLES/COUPLES - Mainstream)	(Kettle)	0.079209	0.167334	0.015579	0.196684	1.175400	0.002325	1.036537
0	(MIDAGE SINGLES/COUPLES - Mainstream)	(Kettle)	0.044966	0.167334	0.008657	0.192519	1.150508	0.001132	1.031190
23	(RETIREES - Budget)	(Kettle)	0.057652	0.167334	0.010505	0.182214	1.088926	0.000858	1.018196
32	(RETIREES - Premium)	(Kettle)	0.049591	0.167334	0.008981	0.181105	1.082296	0.000683	1.016816
13	(OLDER SINGLES/COUPLES - Budget)	(Kettle)	0.069596	0.167334	0.012422	0.178488	1.066658	0.000776	1.013578
20	(OLDER SINGLES/COUPLES - Premium)	(Kettle)	0.067115	0.167334	0.011944	0.177959	1.063495	0.000713	1.012925
26	(RETIREES - Mainstream)	(Kettle)	0.080935	0.167334	0.013723	0.169554	1.013269	0.000180	1.002674
16	(OLDER SINGLES/COUPLES - Mainstream)	(Kettle)	0.069146	0.167334	0.011490	0.166168	0.993034	-0.000081	0.998602
34	(YOUNG FAMILIES - Budget)	(Kettle)	0.071991	0.167334	0.011117	0.154422	0.922837	-0.000930	0.984730
5	(OLDER FAMILIES - Budget)	(Kettle)	0.087193	0.167334	0.013455	0.154318	0.922216	-0.001135	0.984609
10	(OLDER FAMILIES - Mainstream)	(Kettle)	0.053664	0.167334	0.008183	0.152481	0.911237	-0.000797	0.982475
8	(OLDER FAMILIES - Budget)	(Smiths)	0.087193	0.123016	0.011948	0.137027	1.113895	0.001222	1.016236
36	(YOUNG FAMILIES - Budget)	(Smiths)	0.071991	0.123016	0.009459	0.131397	1.068126	0.000603	1.009648
39	(YOUNG SINGLES/COUPLES - Mainstream)	(Doritos)	0.079209	0.102229	0.009642	0.121725	1.190712	0.001544	1.022198
18	(OLDER SINGLES/COUPLES - Mainstream)	(Smiths)	0.069146	0.123016	0.008389	0.121329	0.986288	-0.000117	0.998080
30	(RETIREES - Mainstream)	(Smiths)	0.080935	0.123016	0.009593	0.118528	0.963514	-0.000363	0.994908
42	(YOUNG SINGLES/COUPLES - Mainstream)	(Pringles)	0.079209	0.101735	0.009382	0.118451	1.164310	0.001324	1.018962
15	(OLDER SINGLES/COUPLES - Budget)	(Smiths)	0.069596	0.123016	0.008146	0.117051	0.951509	-0.000415	0.993244
28	(RETIREES - Mainstream)	(Pringles)	0.080935	0.101735	0.008523	0.105308	1.035124	0.000289	1.003994
25	(RETIREES - Mainstream)	(Doritos)	0.080935	0.102229	0.008466	0.104607	1.023260	0.000192	1.002656
3	(OLDER FAMILIES - Budget)	(Doritos)	0.087193	0.102229	0.008235	0.094450	0.923907	-0.000678	0.991410
6	(OLDER FAMILIES - Budget)	(Pringles)	0.087193	0.101735	0.008089	0.092777	0.911949	-0.000781	0.990126

In [84]:
```
rules[rules["antecedents"]=={"YOUNG SINGLES/COUPLES - Mainstream"}]
```

Out[84]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
40	(YOUNG SINGLES/COUPLES - Mainstream)	(Kettle)	0.079209	0.167334	0.015579	0.196684	1.175400	0.002325	1.036537
39	(YOUNG SINGLES/COUPLES - Mainstream)	(Doritos)	0.079209	0.102229	0.009642	0.121725	1.190712	0.001544	1.022198
42	(YOUNG SINGLES/COUPLES - Mainstream)	(Pringles)	0.079209	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.

In [85]:
```
# find the target rating proportion
target_segment = young_mainstream["BRAND_NAME"].value_counts().sort_values(ascending = True).re
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'] != "Mainstr
other = not_young_mainstream["BRAND_NAME"].value_counts().sort_values(ascending = True).rename_
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)
```

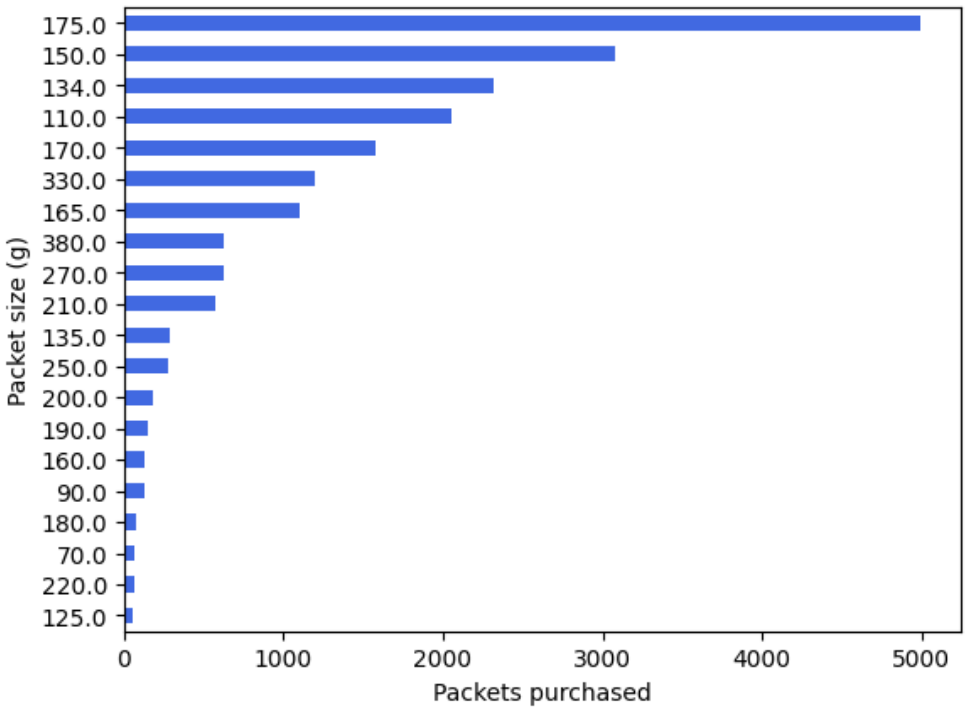
Out[85]:

	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

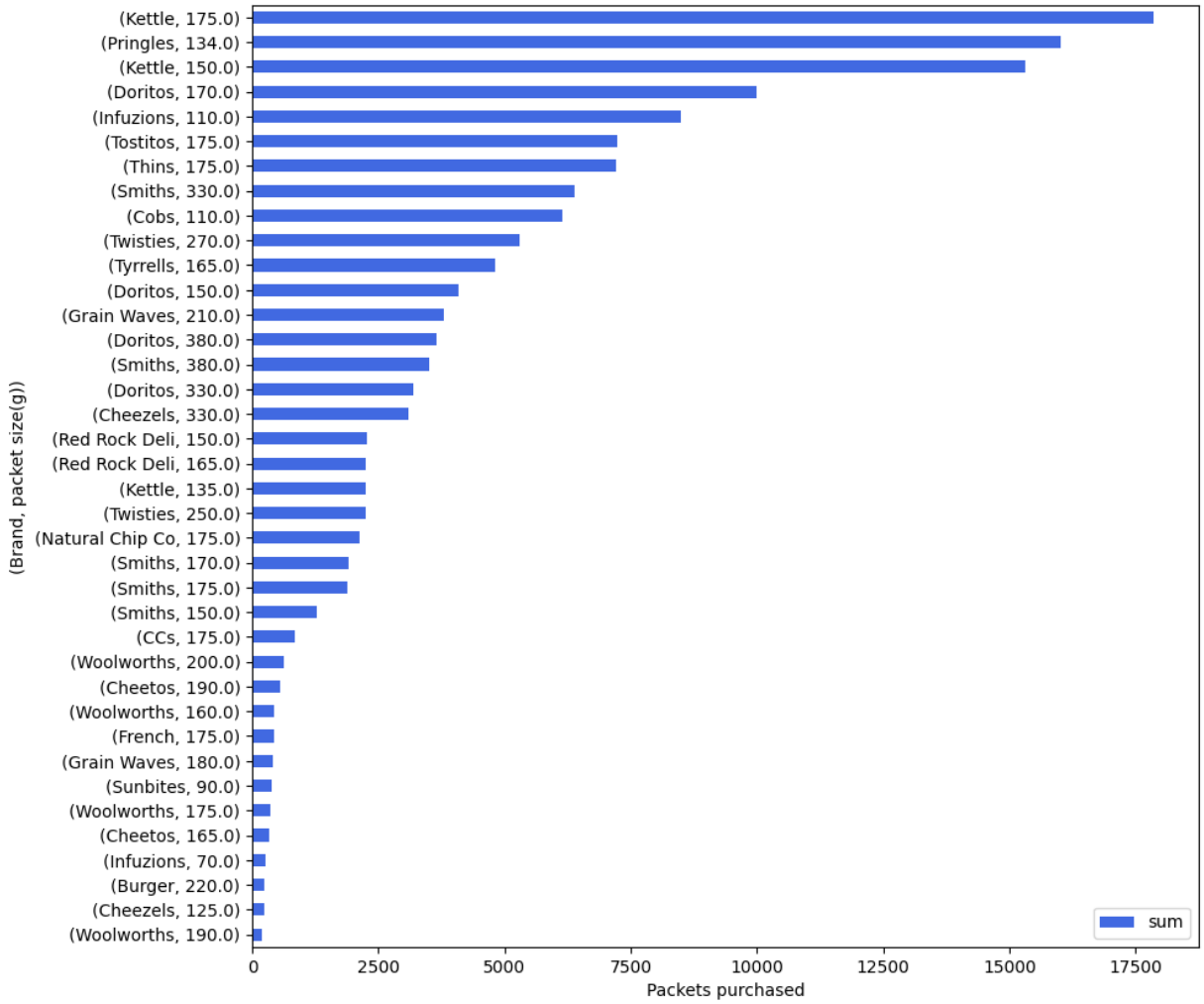
By using the affinity index, we can see that mainstream young singles/couples are 28% more likely to purcahse Tyrrells chips than the other segments. However, they are 50% less likely to purchase Burger Rings.

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

In [88]:
```
# 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(color
ax.set_ylabel("Packet size (g)")
ax.set_xlabel("Packets purchased")
plt.show()
```



In [90]:
```
# 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'
ax = brand_size.sort_values(by = 'sum').plot.barh(y = "sum", figsize=(10,10),color = "royalblue
ax.set_ylabel("(Brand, packet size(g))")
ax.set_xlabel("Packets purchased")
plt.show()
```



In [91]:
```
groups = pd.get_dummies(temp["group"])
brands = pd.get_dummies(temp["PACK_SIZE"])
groups_brands = groups.join(brands)
groups_brands
```

Out[91]:

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

246740 rows × 41 columns

In [92]:
```
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)]
```

```
C:\Users\user\anaconda3\lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:109: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
```

Out[92]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
38	(YOUNG FAMILIES - Premium)	(175.0)	0.043706	0.269069	0.012150	0.278004	1.033210	0.000391	1.012377
34	(YOUNG FAMILIES - Budget)	(175.0)	0.071991	0.269069	0.019944	0.277037	1.029613	0.000574	1.011021
40	(YOUNG SINGLES/COUPLES - Budget)	(175.0)	0.034745	0.269069	0.009476	0.272717	1.013558	0.000127	1.005016
6	(OLDER FAMILIES - Mainstream)	(175.0)	0.053664	0.269069	0.014542	0.270977	1.007091	0.000102	1.002617
8	(OLDER FAMILIES - Premium)	(175.0)	0.042162	0.269069	0.011413	0.270691	1.006030	0.000068	1.002225
25	(RETIREES - Budget)	(175.0)	0.057652	0.269069	0.015591	0.270439	1.005094	0.000079	1.001879
30	(RETIREES - Premium)	(175.0)	0.049591	0.269069	0.013399	0.270186	1.004154	0.000055	1.001531
4	(OLDER FAMILIES - Budget)	(175.0)	0.087193	0.269069	0.023539	0.269964	1.003327	0.000078	1.001226
12	(OLDER SINGLES/COUPLES - Budget)	(175.0)	0.069596	0.269069	0.018744	0.269334	1.000985	0.000018	1.000363
20	(OLDER SINGLES/COUPLES - Premium)	(175.0)	0.067115	0.269069	0.018068	0.269203	1.000499	0.000009	1.000184
0	(MIDAGE SINGLES/COUPLES - Mainstream)	(175.0)	0.044966	0.269069	0.012057	0.268139	0.996544	-0.000042	0.998729
36	(YOUNG FAMILIES - Mainstream)	(175.0)	0.048419	0.269069	0.012864	0.265673	0.987381	-0.000164	0.995376
16	(OLDER SINGLES/COUPLES - Mainstream)	(175.0)	0.069146	0.269069	0.018339	0.265225	0.985714	-0.000266	0.994769
28	(RETIREES - Mainstream)	(175.0)	0.080935	0.269069	0.021460	0.265148	0.985428	-0.000317	0.994664
46	(YOUNG SINGLES/COUPLES - Mainstream)	(175.0)	0.079209	0.269069	0.020252	0.255679	0.950239	-0.001061	0.982012
18	(OLDER SINGLES/COUPLES - Premium)	(150.0)	0.067115	0.162937	0.011218	0.167150	1.025857	0.000283	1.005059
2	(OLDER FAMILIES - Budget)	(150.0)	0.087193	0.162937	0.014542	0.166775	1.023558	0.000335	1.004607
26	(RETIREES - Mainstream)	(150.0)	0.080935	0.162937	0.013334	0.164747	1.011111	0.000147	1.002168
11	(OLDER SINGLES/COUPLES - Budget)	(150.0)	0.069596	0.162937	0.011393	0.163697	1.004665	0.000053	1.000909
23	(RETIREES - Budget)	(150.0)	0.057652	0.162937	0.009399	0.163023	1.000529	0.000005	1.000103
14	(OLDER SINGLES/COUPLES - Mainstream)	(150.0)	0.069146	0.162937	0.011239	0.162534	0.997531	-0.000028	0.999520
32	(YOUNG FAMILIES - Budget)	(150.0)	0.071991	0.162937	0.011599	0.161121	0.988859	-0.000131	0.997836
44	(YOUNG SINGLES/COUPLES - Mainstream)	(150.0)	0.079209	0.162937	0.012483	0.157593	0.967205	-0.000423	0.993657
42	(YOUNG SINGLES/COUPLES - Mainstream)	(134.0)	0.079209	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.

In [93]:
```
# find the target rating proportion
target_segment = young_mainstream["PACK_SIZE"].value_counts().sort_values(ascending = True).ren
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_a
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[93]:

	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:

Older families - Budget, Young singles/couples - Mainstream, 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 [ ]: