In [1]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import re

from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
from sklearn.preprocessing import OneHotEncoder

import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
df_trans = pd.read_excel(r'C:\Users\1100634\Downloads\Quant\QVI_transaction_data.xl
df_purchase = pd.read_csv(r'C:\Users\1100634\Downloads\Quant\QVI_purchase_behaviour
```

In [4]:
```
df_trans.head()
```

Out[4]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SA
0	2018-10-17	1	1000	1	5	Natural Chip Compny SeaSalt175g	2	
1	2019-05-14	1	1307	348	66	CCs Nacho Cheese 175g	3	
2	2019-05-20	1	1343	383	61	Smiths Crinkle Cut Chips Chicken 170g	2	
3	2018-08-17	2	2373	974	69	Smiths Chip Thinly S/Cream&Onion 175g	5	
4	2018-08-18	2	2426	1038	108	Kettle Tortilla ChpsHny&Jlpno Chili 150g	3	

In [5]:
```
df_trans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 264836 entries, 0 to 264835
Data columns (total 8 columns):
 #   Column          Non-Null Count   Dtype         
---  ------          --------------   -----         
 0   DATE            264836 non-null  datetime64[ns]
 1   STORE_NBR       264836 non-null  int64         
 2   LYLTY_CARD_NBR  264836 non-null  int64         
 3   TXN_ID          264836 non-null  int64         
 4   PROD_NBR        264836 non-null  int64         
 5   PROD_NAME       264836 non-null  object        
 6   PROD_QTY        264836 non-null  int64         
 7   TOT_SALES       264836 non-null  float64       
dtypes: datetime64[ns](1), float64(1), int64(5), object(1)
memory usage: 16.2+ MB
```

We are instructed to only analyse the chips and the sales on that

In [6]:
```
prod_name = df_trans["PROD_NAME"].str.replace(r'\d+[gG]', '', regex=True)
prod_name = prod_name.str.replace(r'&', ' ', regex=True)
```

In [7]:
```
word_counts = pd.Series(' '.join(prod_name).split()).value_counts()

word_counts = word_counts[~word_counts.index.str.contains(r'^\d+[gG]$')]

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 [8]:
```
df_trans = df_trans[df_trans["PROD_NAME"].str.contains(r'[Ss]alsa') == False]
df_trans.shape
```

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

In [9]:
```
df_trans.isnull().sum()
```

Out[9]:
```
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 [10]:
```
df_trans.describe()
```

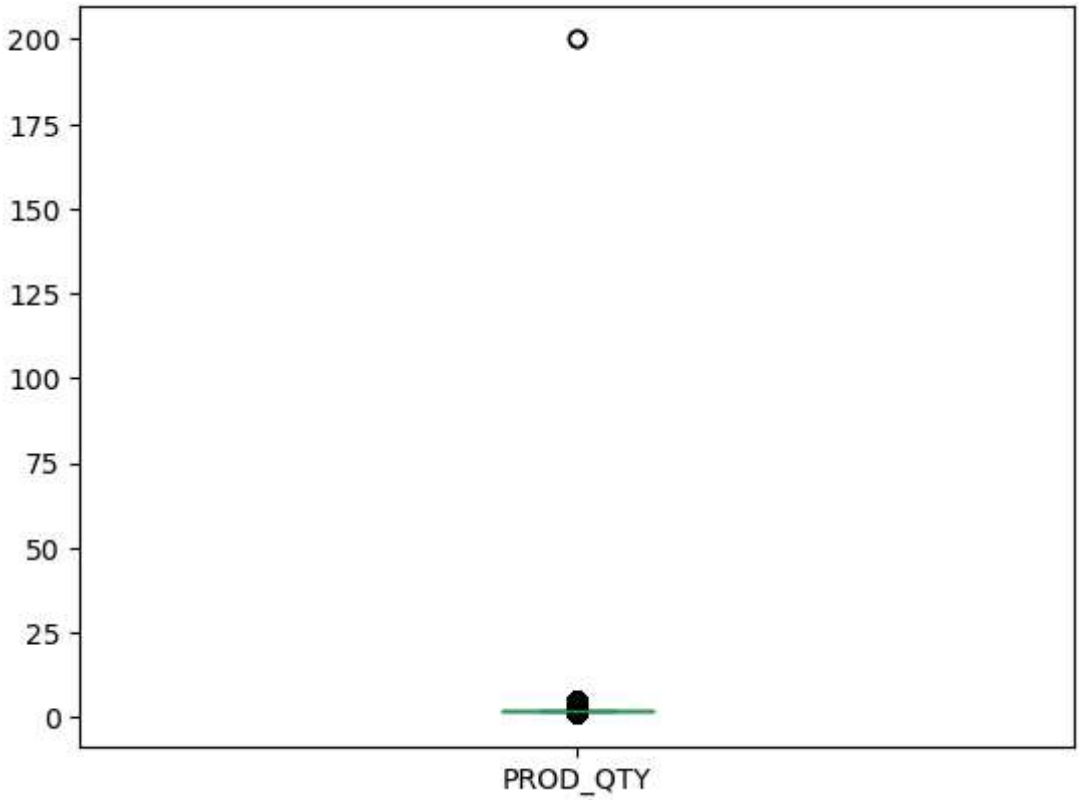
Out[10]:

	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_QTY	TOT_SALI
count	246742.000000	2.467420e+05	2.467420e+05	246742.000000	246742.000000	246742.0000(
mean	135.051098	1.355310e+05	1.351311e+05	56.351789	1.908062	7.3213:
std	76.787096	8.071528e+04	7.814772e+04	33.695428	0.659831	3.0778:
min	1.000000	1.000000e+03	1.000000e+00	1.000000	1.000000	1.7000(
25%	70.000000	7.001500e+04	6.756925e+04	26.000000	2.000000	5.8000(
50%	130.000000	1.303670e+05	1.351830e+05	53.000000	2.000000	7.4000(
75%	203.000000	2.030840e+05	2.026538e+05	87.000000	2.000000	8.8000(
max	272.000000	2.373711e+06	2.415841e+06	114.000000	200.000000	650.0000(

Lets check the qty no

In [11]:
```
df_trans["PROD_QTY"].plot(kind="box")
```

Out[11]:
```
<Axes: >
```



Its seen there is an outlier in the product qty

In [12]:
```
df_trans.loc[df_trans['PROD_QTY'] == 200]
```

Out[12]:

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

There are only two entries that to done by same customer, check once with customer id to confirm

In [13]:
```
df_trans.loc[df_trans["LYLTY_CARD_NBR"] == 226000]
```

Out[13]:

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

So, only two transactions, we can remove that

In [14]:
```
df_trans = df_trans[df_trans["LYLTY_CARD_NBR"] != 226000]
df_trans.shape
```

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

In [15]:
```
df_trans.describe().T
```

Out[15]:

	count	mean	std	min	25%	50%	75%	
STORE_NBR	246740.0	135.050361	76.786971	1.0	70.00	130.0	203.00	
LYLTY_CARD_NBR	246740.0	135530.251641	80715.196924	1000.0	70015.00	130367.0	203083.25	2.
TXN_ID	246740.0	135130.360627	78147.604242	1.0	67568.75	135181.5	202652.25	2.
PROD_NBR	246740.0	56.352213	33.695235	1.0	26.00	53.0	87.00	
PROD_QTY	246740.0	1.906456	0.342499	1.0	2.00	2.0	2.00	
TOT_SALES	246740.0	7.316113	2.474897	1.7	5.80	7.4	8.80	

In [16]:
```
#check for missing data in a date column

count =  df_trans.groupby("DATE").size().reset_index(name='COUNT')
count.shape
```

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

In [17]:
```
df_trans.sort_values(by='DATE')
```

Out[17]:

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

246740 rows × 8 columns

So the starting date is 1st july 2018, end date is 30 june 2019, so arounf 365 days, but in our dte count one is missing

In [18]:
```
counts = df_trans.groupby("DATE").size()
pd.date_range(start="2018-07-01", end="2019-06-30").difference(counts.index)
```

Out[18]:
```
DatetimeIndex(['2018-12-25'], dtype='datetime64[ns]', freq=None)
```

25th december it is the missing day

In [19]:
```
df_trans["PACK_SIZE"] = df_trans["PROD_NAME"].str.extract(r'(\d+)').astype(float)
```

In [20]:
```
df_trans.sort_values(by='PACK_SIZE').head()
```

Out[20]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TO
40783	2018-09-25	97	97067	96696	38	Infuzions Mango Chutny Papadums 70g	2	
42461	2019-05-05	110	110030	111890	38	Infuzions Mango Chutny Papadums 70g	2	
176183	2018-12-30	82	82183	81660	38	Infuzions Mango Chutny Papadums 70g	2	
227309	2018-12-03	236	236091	239098	38	Infuzions Mango Chutny Papadums 70g	2	
42418	2018-11-05	109	109217	111470	38	Infuzions Mango Chutny Papadums 70g	2	

In [21]:
```
px.histogram(
    df_trans,
    x="PACK_SIZE",
    y="PROD_QTY",
    histfunc='sum',
    nbins=20,
    title='Weighted Histogram of Pack Sizes',
    labels={"PACK_SIZE": "Pack Size (gm)", "PROD_QTY": "Total Quantity"}
)
```

Create a brand name column

In [22]:
```
df_trans["BRAND_NAME"] = df_trans["PROD_NAME"].str.split().str.get(0)
```

In [23]:
```
df_trans["BRAND_NAME"].unique()
```

Out[23]:
```
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)
```

We'll handle the duplicate names

In [24]:
```
def replace_brand(stl):
    name = stl['BRAND_NAME']
    if name == 'Dorito':
        return 'Doritos'
    elif name == 'Smith':
        return 'Smiths'
    elif name == 'WW':
        return 'Woolworths'
    elif name == 'Infzns':
        return 'Infuzions'
    elif name == 'NCC' or name == 'Natural':
        return 'Natural Chip Co'
    elif name == 'Snbts':
        return 'Sunbites'
    elif name == 'Red' or name == 'RRD':
```

```
        return 'Red Rock Deli'
    elif name == 'Grain' or name == 'GrnWves':
        return 'Grain Waves'
    else:
        return name


df_trans['BRAND_NAME'] = df_trans.apply(lambda stl: replace_brand(stl), axis=1)
df_trans["BRAND_NAME"].unique()
```

Out[24]:
```
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)
```

Check customer data

In [25]:
```
df_purchase.head()
```

Out[25]:

	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 [26]:
```
df_purchase = df_purchase.rename(columns={'PREMIUM_CUSTOMER':'MEMBER_TYPE'})
```

In [27]:
```
df_purchase['MEMBER_TYPE'].unique()
```

Out[27]:
```
array(['Premium', 'Mainstream', 'Budget'], dtype=object)
```

In [28]:
```
df_purchase['LIFESTAGE'].unique()
```

Out[28]:
```
array(['YOUNG SINGLES/COUPLES', 'YOUNG FAMILIES', 'OLDER SINGLES/COUPLES',
       'MIDAGE SINGLES/COUPLES', 'NEW FAMILIES', 'OLDER FAMILIES',
       'RETIREES'], dtype=object)
```

In [29]:
```
df = df_trans.set_index('LYLTY_CARD_NBR').join(df_purchase.set_index('LYLTY_CARD_NE
df =df.reset_index()
df = df.sort_values(by="DATE").reset_index(drop=True)
df.head()
```

Out[29]:

	LYLTY_CARD_NBR	DATE	STORE_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALI
0	21037	2018-07-01	21	17576	62	Pringles Mystery Flavour 134g	2	7
1	25040	2018-07-01	25	21704	87	Infuzions BBQ Rib Prawn Crackers 110g	2	7
2	59236	2018-07-01	59	55555	42	Doritos Corn Chip Mexican Jalapeno 150g	2	7
3	271083	2018-07-01	271	268688	97	RRD Salt & Vinegar 165g	2	6
4	65015	2018-07-01	65	61737	17	Kettle Sensations BBQ&Maple 150g	2	9

In [30]:
```
df.isnull().sum().any()
```

Out[30]:
```
False
```

In [31]:
```
# df.to_csv('QVI_cleaned_data.csv')
```

Let's dive into Data analysis by customer segments

In [32]:
```
total_sales = df.groupby(['LIFESTAGE','MEMBER_TYPE'], as_index=False)['TOT_SALES'].
total_sales = total_sales.rename(columns={'sum':'SUM_TOTAL_SALES'})
total_sales.sort_values(by='SUM_TOTAL_SALES', ascending=False)
```

Out[32]:

LIFESTAGE	MEMBER_TYPE	SUM_TOTAL_SALES
OLDER FAMILIES	Budget	156863.75
YOUNG SINGLES/COUPLES	Mainstream	147582.20
RETIREES	Mainstream	145168.95
YOUNG FAMILIES	Budget	129717.95
OLDER SINGLES/COUPLES	Budget	127833.60
	Mainstream	124648.50
	Premium	123537.55
RETIREES	Budget	105916.30
OLDER FAMILIES	Mainstream	96413.55
RETIREES	Premium	91296.65
YOUNG FAMILIES	Mainstream	86338.25
MIDAGE SINGLES/COUPLES	Mainstream	84734.25
YOUNG FAMILIES	Premium	78571.70
OLDER FAMILIES	Premium	75242.60
YOUNG SINGLES/COUPLES	Budget	57122.10
MIDAGE SINGLES/COUPLES	Premium	54443.85
YOUNG SINGLES/COUPLES	Premium	39052.30
MIDAGE SINGLES/COUPLES	Budget	33345.70
NEW FAMILIES	Budget	20607.45
	Mainstream	15979.70
	Premium	10760.80

In [33]:
```
total_saless = df["TOT_SALES"].sum()
```

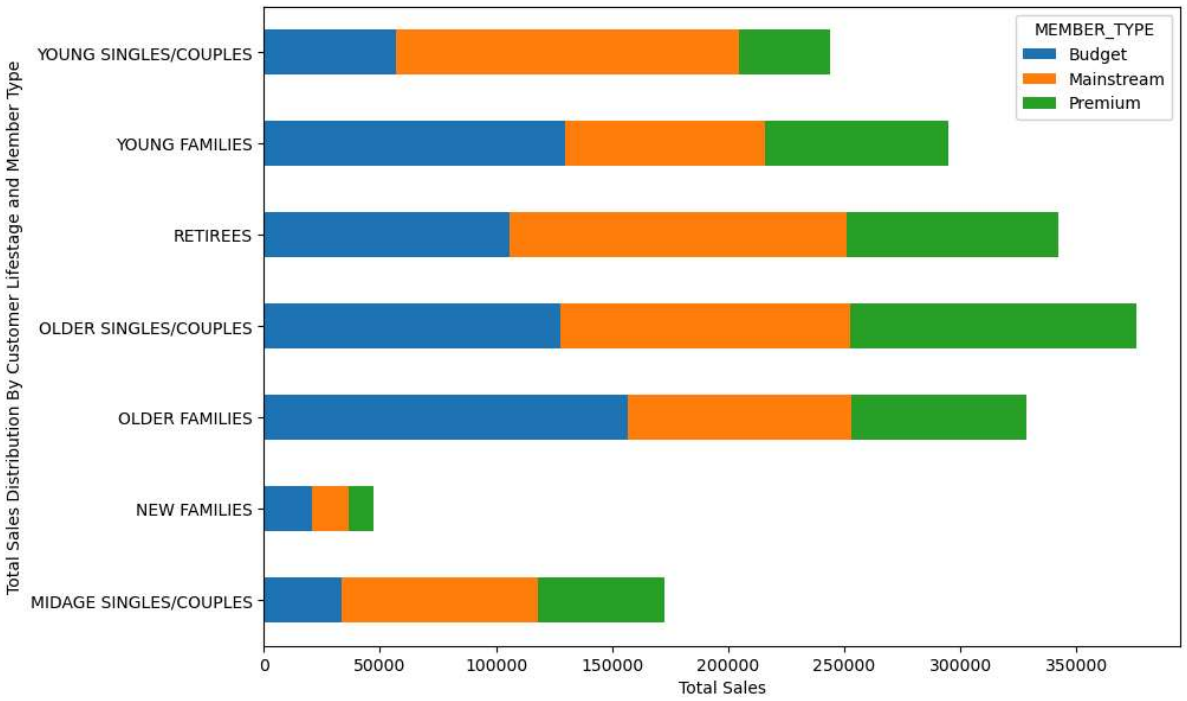
In [34]:
```
total_sales_break = df.groupby(["LIFESTAGE","MEMBER_TYPE"], as_index=True)["TOT_SAL
```

In [35]:
```
total_sales_break
```

Out[35]:

	sum			mean		
MEMBER_TYPE	Budget	Mainstream	Premium	Budget	Mainstream	Premium
LIFESTAGE						
MIDAGE SINGLES/COUPLES	33345.70	84734.25	54443.85	7.108442	7.637156	7.152371
NEW FAMILIES	20607.45	15979.70	10760.80	7.297256	7.313364	7.231720
OLDER FAMILIES	156863.75	96413.55	75242.60	7.291241	7.281440	7.232779
OLDER SINGLES/COUPLES	127833.60	124648.50	123537.55	7.444305	7.306049	7.459997
RETIREES	105916.30	145168.95	91296.65	7.445786	7.269352	7.461315
YOUNG FAMILIES	129717.95	86338.25	78571.70	7.302705	7.226772	7.285951
YOUNG SINGLES/COUPLES	57122.10	147582.20	39052.30	6.663023	7.551279	6.673325

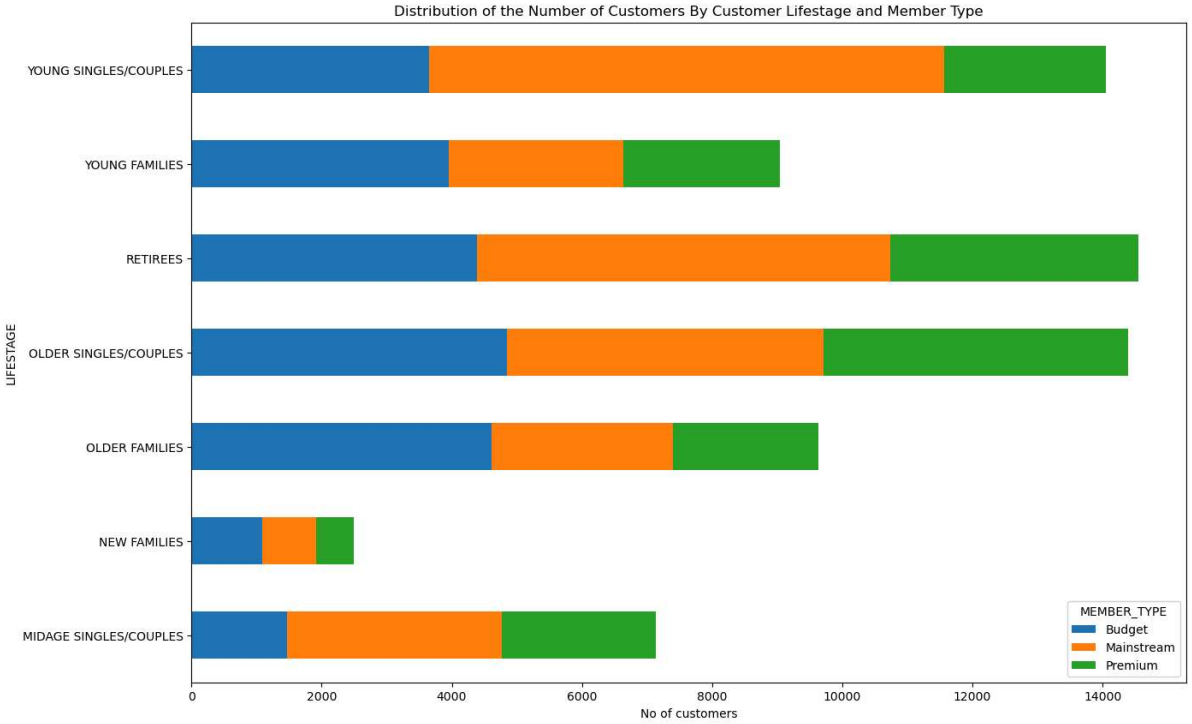
In [36]:
```
ax = total_sales_break['sum'].plot(kind='barh', stacked=True, figsize=(10, 7))
ax.set_xlabel("Total Sales")
ax.set_ylabel("Total Sales Distribution By Customer Lifestage and Member Type")
plt.show()
```



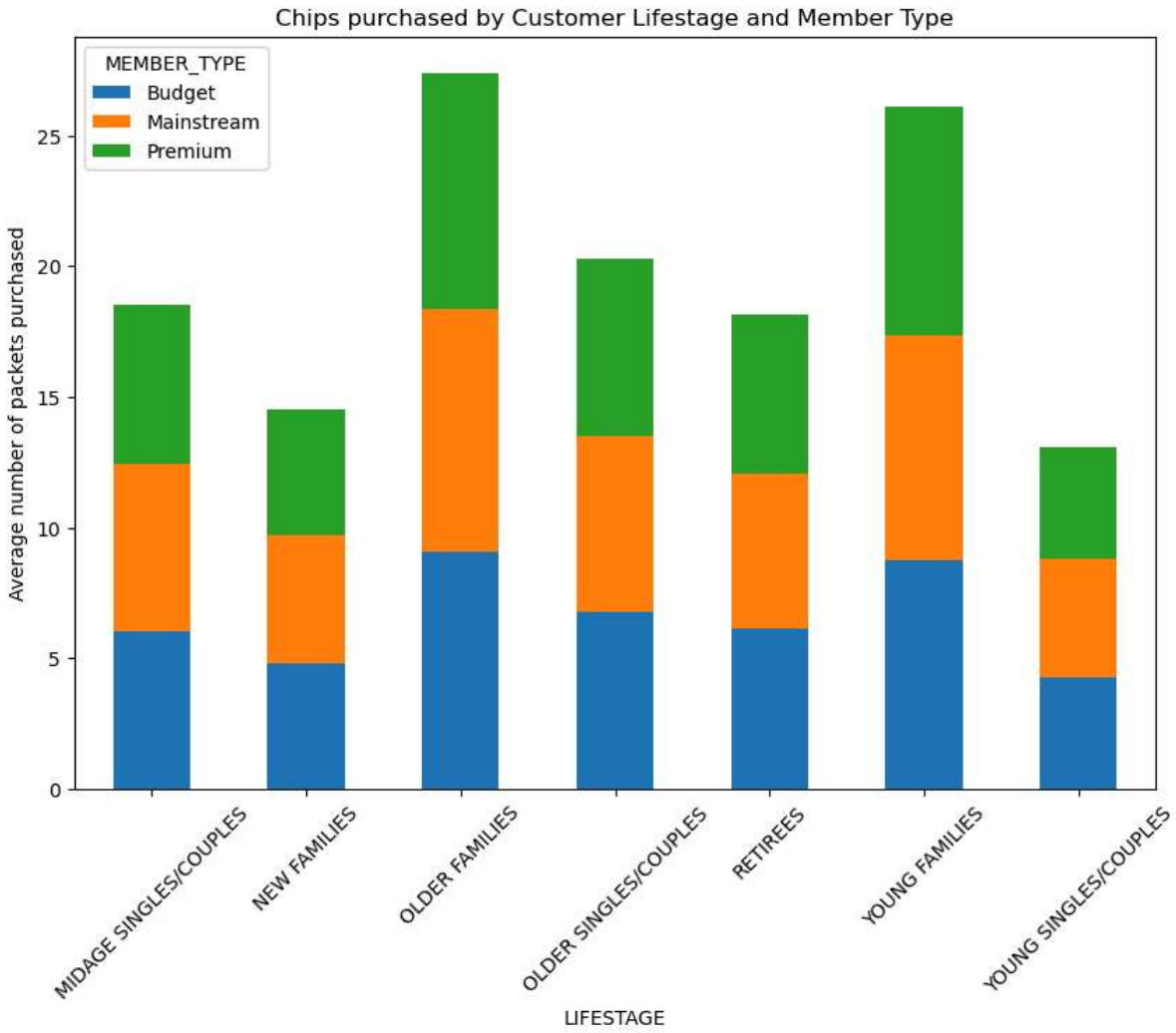
In [37]:
```
#check whether all customer made chip purchase

len(df_purchase['LYLTY_CARD_NBR'].unique()) == len(df['LYLTY_CARD_NBR'].unique())
```

Out[37]:
```
False
```

In [38]:
```
customer_count = df.groupby(['LIFESTAGE','MEMBER_TYPE'])['LYLTY_CARD_NBR'].agg('nur
ax = customer_count.plot(kind='barh', stacked=True, figsize=(15, 10))

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



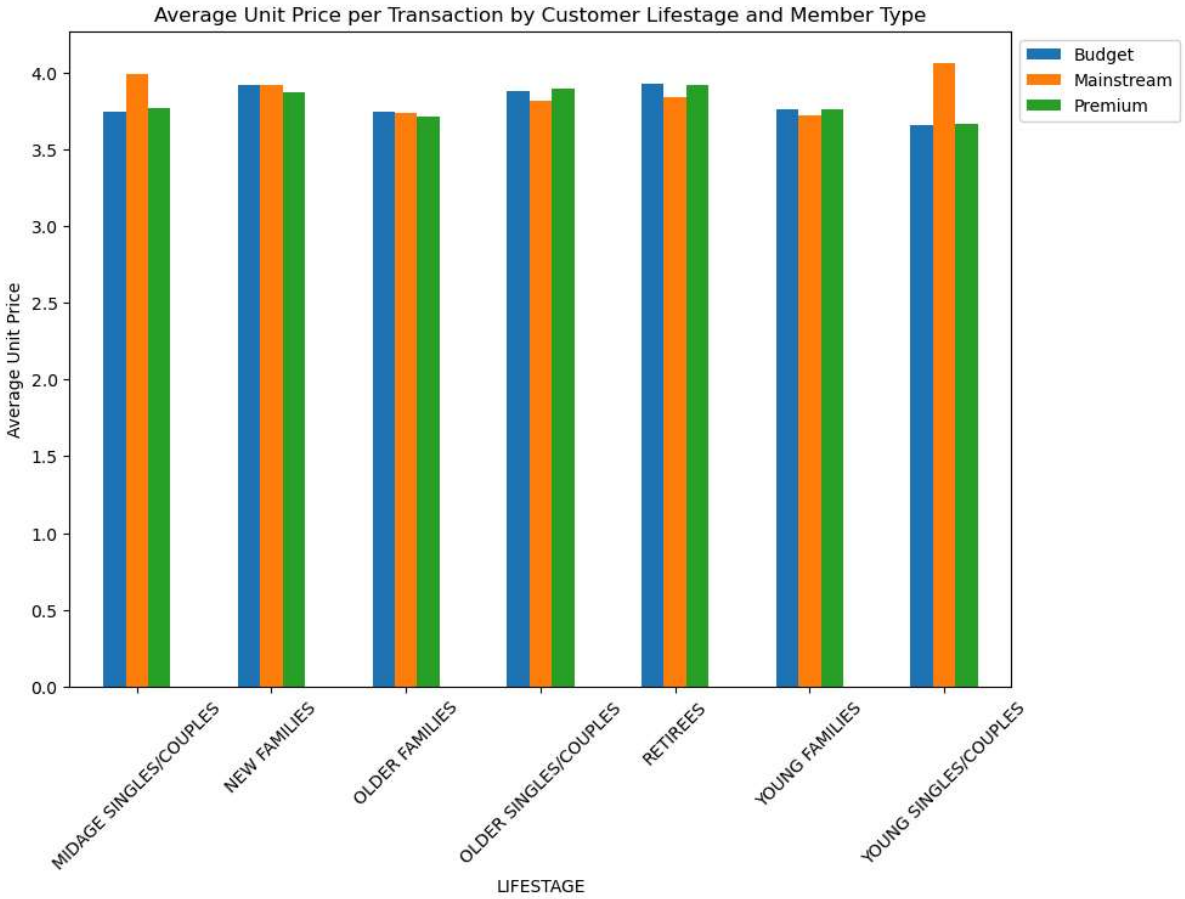
In [39]:
```
# Plot the total and average units of chips bought per customer by Lifestage and Me
num_packets_data = df.groupby(['LIFESTAGE', 'MEMBER_TYPE'])['PROD_QTY'].sum()/df.gr
ax = num_packets_data.unstack('MEMBER_TYPE').fillna(0).plot.bar(stacked = True, fig
ax.set_ylabel('Average number of packets purchased')
ax.set_title('Chips purchased by Customer Lifestage and Member Type')
plt.xticks(rotation = 45)
plt.show()
```



Older Families and Young Families buy more chips per customer.

In [40]:
```
df['UNIT_PRICE'] = df['TOT_SALES']/df['PROD_QTY']
```

In [41]:
```
# Plot the distribution of average unit price per transaction by lifestage and memb
avg_unit_price = df.groupby(['LIFESTAGE', 'MEMBER_TYPE'], as_index = True)['UNIT_PR
ax = avg_unit_price['mean'].plot.bar(stacked = False, figsize = (10, 7))
ax.set_ylabel('Average Unit Price')
ax.set_title('Average Unit Price per Transaction by Customer Lifestage and Member T
plt.legend(loc='upper left', bbox_to_anchor=(1.0, 1.0))
plt.xticks(rotation = 45)
plt.show()
```



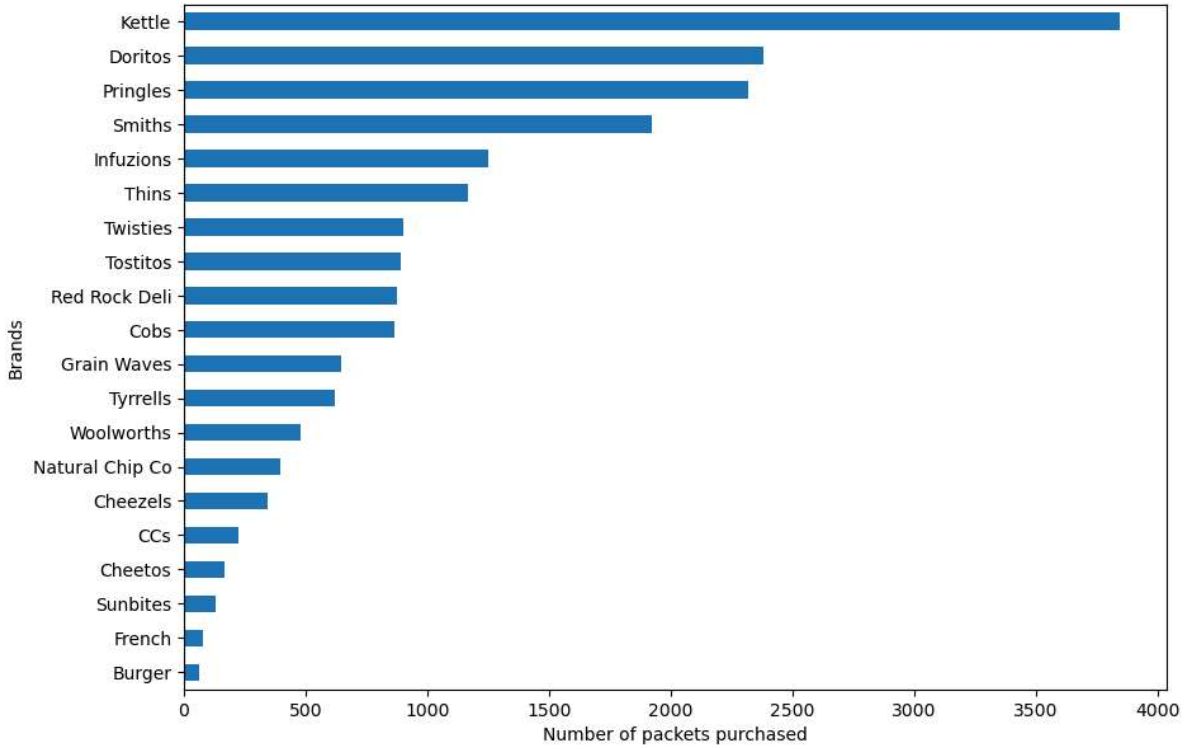
In [42]:
```
#Lets do t-test
from scipy.stats import ttest_ind
```

In [43]:
```
mainstream = df["MEMBER_TYPE"] == 'Mainstream'
young_midage = (df["LIFESTAGE"] == 'MIDAGE SINGLES/COUPLES')| (df["LIFESTAGE"] == '
premium_budget = df["MEMBER_TYPE"] != "Mainstream"

group1 = df[mainstream & young_midage]['UNIT_PRICE']
group2 = df[premium_budget & young_midage]['UNIT_PRICE']

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

print(pval, stat)
```

```
6.967354232991983e-306 37.6243885962296
```

Lets dive to specific customer segments

In [44]:
```
young_mainstream = df.loc[df['LIFESTAGE'] == "YOUNG SINGLES/COUPLES"]
young_mainstream = young_mainstream.loc[young_mainstream['MEMBER_TYPE'] == "Mainstr
ax = young_mainstream["BRAND_NAME"].value_counts().sort_values(ascending = True).pl
ax.set_xlabel("Number of packets purchased")
ax.set_ylabel("Brands")
plt.show()
```



In [45]:
```
temp = df.copy()
temp["GROUP"] = temp["LIFESTAGE"] +' - ' + temp['MEMBER_TYPE']
```

In [46]:
```
groups = pd.get_dummies(temp['GROUP'])
brands = pd.get_dummies(temp['BRAND_NAME'])
groups_brands = groups.join(brands)
groups_brands
```

Out[46]:

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

246740 rows × 41 columns

In [48]:
```
#lets use apriori algorithm for frequency

freq_groups_brands = apriori(groups_brands, min_support=0.008, use_colnames=True)
```

```
rules = association_rules(freq_groups_brands, metric='lift', min_threshold=0.5,
                          num_itemsets=freq_groups_brands['itemsets'].apply(len).ma

rules.sort_values('confidence', ascending=False, inplace=True)
```

In [50]:
```
set_temp = temp["GROUP"].unique()
rules[rules["antecedents"].apply(lambda x: list(x)).apply(lambda x: x in set_temp)]
```

Out[50]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	repr
41	(YOUNG SINGLES/COUPLES - Mainstream)	(Kettle)	0.079209	0.167334	0.015579	0.196684	1.175400	
0	(MIDAGE SINGLES/COUPLES - Mainstream)	(Kettle)	0.044966	0.167334	0.008657	0.192519	1.150508	
23	(RETIREES - Budget)	(Kettle)	0.057652	0.167334	0.010505	0.182214	1.088926	
32	(RETIREES - Premium)	(Kettle)	0.049591	0.167334	0.008981	0.181105	1.082296	
13	(OLDER SINGLES/COUPLES - Budget)	(Kettle)	0.069596	0.167334	0.012422	0.178488	1.066658	
21	(OLDER SINGLES/COUPLES - Premium)	(Kettle)	0.067115	0.167334	0.011944	0.177959	1.063495	
27	(RETIREES - Mainstream)	(Kettle)	0.080935	0.167334	0.013723	0.169554	1.013269	
17	(OLDER SINGLES/COUPLES - Mainstream)	(Kettle)	0.069146	0.167334	0.011490	0.166168	0.993034	
35	(YOUNG FAMILIES - Budget)	(Kettle)	0.071991	0.167334	0.011117	0.154422	0.922837	
4	(OLDER FAMILIES - Budget)	(Kettle)	0.087193	0.167334	0.013455	0.154318	0.922216	
11	(OLDER FAMILIES - Mainstream)	(Kettle)	0.053664	0.167334	0.008183	0.152481	0.911237	
9	(OLDER FAMILIES - Budget)	(Smiths)	0.087193	0.123016	0.011948	0.137027	1.113895	
37	(YOUNG FAMILIES - Budget)	(Smiths)	0.071991	0.123016	0.009459	0.131397	1.068126	
39	(YOUNG SINGLES/COUPLES - Mainstream)	(Doritos)	0.079209	0.102229	0.009642	0.121725	1.190712	
19	(OLDER SINGLES/COUPLES - Mainstream)	(Smiths)	0.069146	0.123016	0.008389	0.121329	0.986288	
31	(RETIREES - Mainstream)	(Smiths)	0.080935	0.123016	0.009593	0.118528	0.963514	
43	(YOUNG SINGLES/COUPLES - Mainstream)	(Pringles)	0.079209	0.101735	0.009382	0.118451	1.164310	
15	(OLDER SINGLES/COUPLES - Budget)	(Smiths)	0.069596	0.123016	0.008146	0.117051	0.951509	
29	(RETIREES - Mainstream)	(Pringles)	0.080935	0.101735	0.008523	0.105308	1.035124	

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	repr
25	(RETIREES - Mainstream)	(Doritos)	0.080935	0.102229	0.008466	0.104607	1.023260	
3	(OLDER FAMILIES - Budget)	(Doritos)	0.087193	0.102229	0.008235	0.094450	0.923907	
6	(OLDER FAMILIES - Budget)	(Pringles)	0.087193	0.101735	0.008089	0.092777	0.911949	

In [51]:
```
rules[rules['antecedents'] == {'YOUNG SINGLES/COUPLES - Mainstream'}]
```

Out[51]:

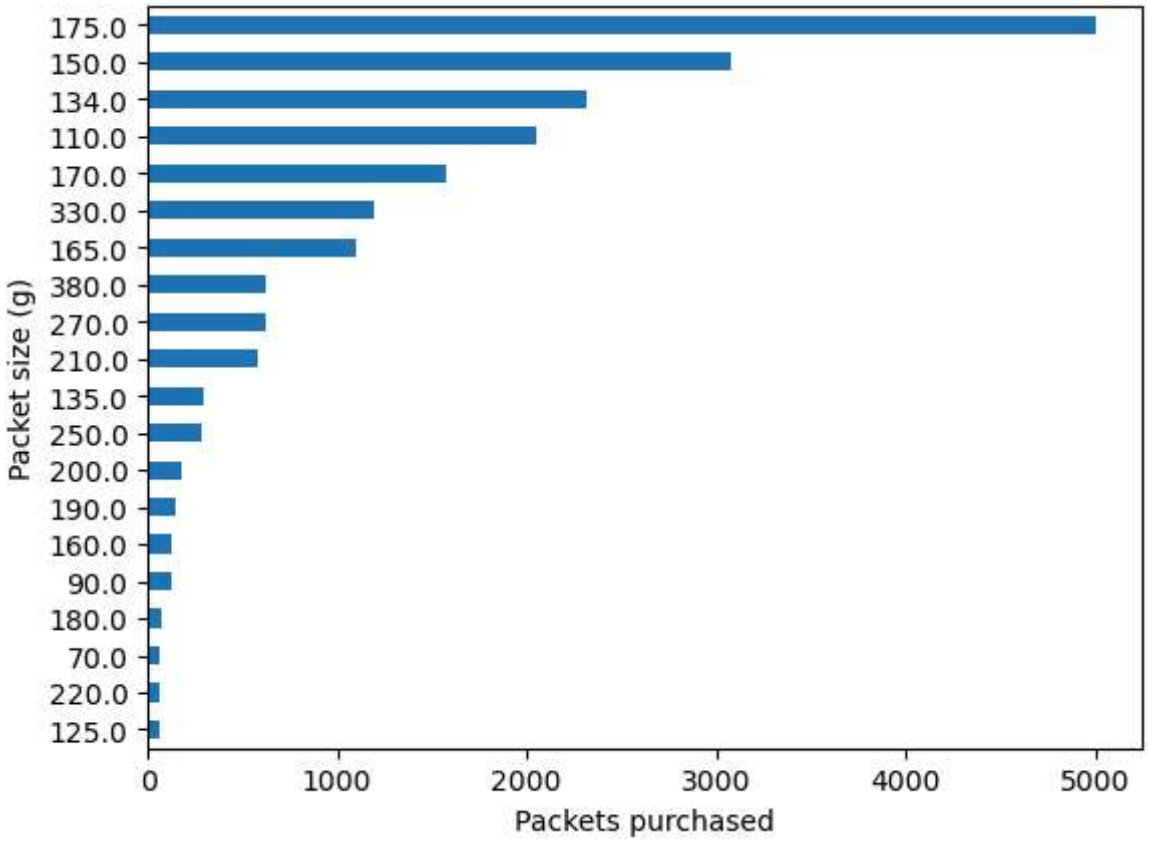
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	repr
41	(YOUNG SINGLES/COUPLES - Mainstream)	(Kettle)	0.079209	0.167334	0.015579	0.196684	1.175400	
39	(YOUNG SINGLES/COUPLES - Mainstream)	(Doritos)	0.079209	0.102229	0.009642	0.121725	1.190712	
43	(YOUNG SINGLES/COUPLES - Mainstream)	(Pringles)	0.079209	0.101735	0.009382	0.118451	1.164310	

In [53]:
```
# Find target rating proportion
target_segment = young_mainstream["BRAND_NAME"].value_counts().sort_values(ascendir
target_segment.target /= young_mainstream["PROD_QTY"].sum()

# Find other rating proportion
not_young_mainstream = df.loc[df['LIFESTAGE'] != "YOUNG SINGLES/COUPLES"]
not_young_mainstream = not_young_mainstream.loc[not_young_mainstream['MEMBER_TYPE']
other = not_young_mainstream["BRAND_NAME"].value_counts().sort_values(ascending = T
other.other /= not_young_mainstream["PROD_QTY"].sum()

brand_proportions = target_segment.set_index('BRANDS').join(other.set_index('BRANDS

brand_proportions = brand_proportions.reset_index()
brand_proportions['affinity'] = brand_proportions['target']/brand_proportions['othe
brand_proportions.sort_values(by = 'affinity', ascending = False)
```

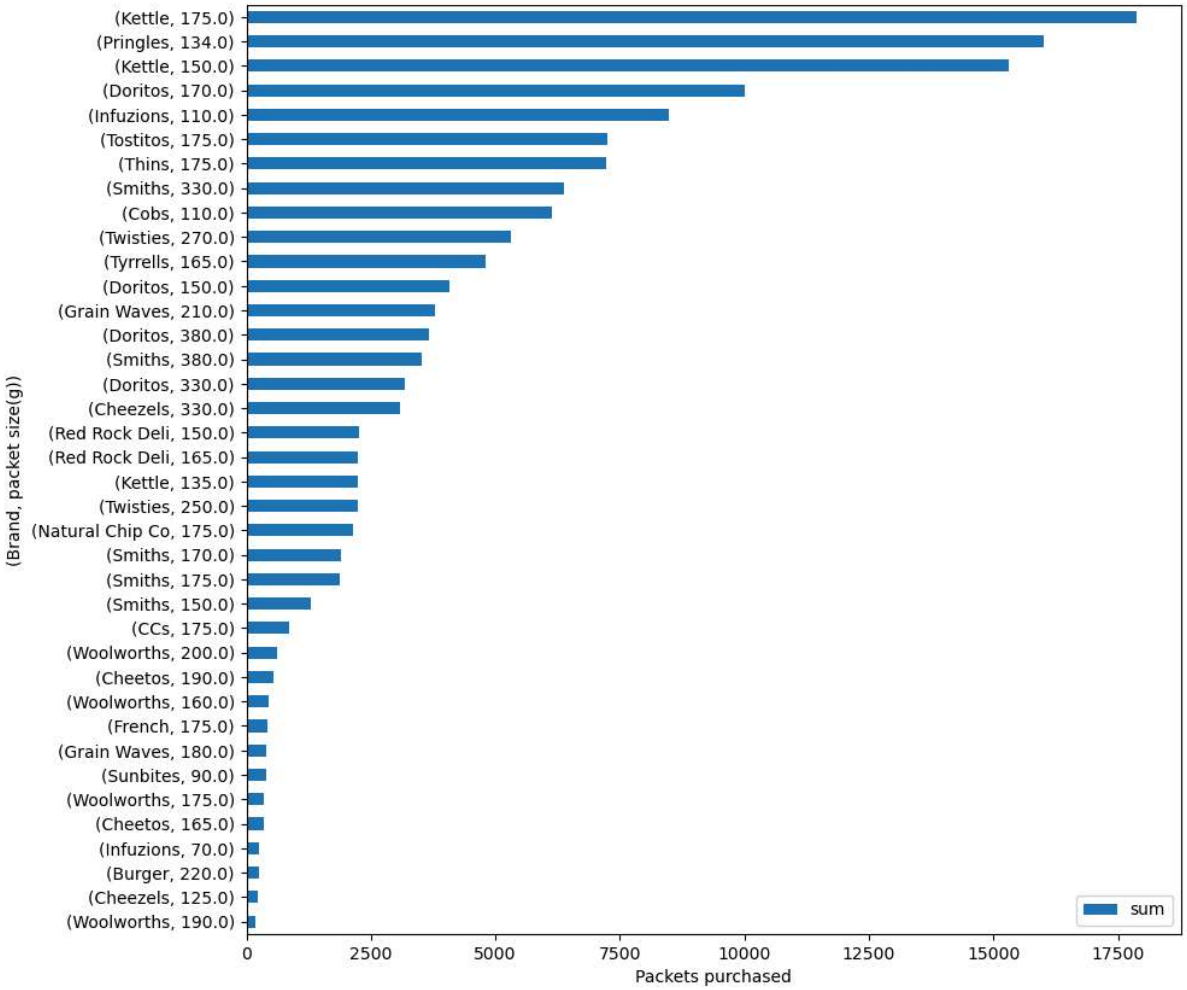
Out[53]:

	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

In [54]:
```
# Plot the distribution of the packet sizes for a general indication of popularity
young_mainstream = df.loc[df['LIFESTAGE'] == "YOUNG SINGLES/COUPLES"]
young_mainstream = young_mainstream.loc[young_mainstream['MEMBER_TYPE'] == "Mainstr
ax = young_mainstream["PACK_SIZE"].value_counts().sort_values(ascending = True).plo
ax.set_ylabel("Packet size (g)")
ax.set_xlabel("Packets purchased")
plt.show()
```



In [56]:
```
#Check which brands correspond to what sized packets.
brand_size = young_mainstream.groupby(['BRAND_NAME','PACK_SIZE'], as_index = False)
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()
```



In [57]:
```
groups = pd.get_dummies(temp["GROUP"])
brands = pd.get_dummies(temp["PACK_SIZE"])
groups_brands = groups.join(brands)
groups_brands
```

Out[57]:

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

246740 rows × 41 columns

In [58]:

```
freq_groupsbrands = apriori(groups_brands, min_support=0.009, use_colnames=True)
rules = association_rules(freq_groupsbrands, metric="lift", min_threshold=0.5, num_
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)]
```

Out[58]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	repr
38	(YOUNG FAMILIES - Premium)	(175.0)	0.043706	0.269069	0.012150	0.278004	1.033210	
34	(YOUNG FAMILIES - Budget)	(175.0)	0.071991	0.269069	0.019944	0.277037	1.029613	
40	(YOUNG SINGLES/COUPLES - Budget)	(175.0)	0.034745	0.269069	0.009476	0.272717	1.013558	
6	(OLDER FAMILIES - Mainstream)	(175.0)	0.053664	0.269069	0.014542	0.270977	1.007091	
8	(OLDER FAMILIES - Premium)	(175.0)	0.042162	0.269069	0.011413	0.270691	1.006030	
24	(RETIREES - Budget)	(175.0)	0.057652	0.269069	0.015591	0.270439	1.005094	
30	(RETIREES - Premium)	(175.0)	0.049591	0.269069	0.013399	0.270186	1.004154	
4	(OLDER FAMILIES - Budget)	(175.0)	0.087193	0.269069	0.023539	0.269964	1.003327	
13	(OLDER SINGLES/COUPLES - Budget)	(175.0)	0.069596	0.269069	0.018744	0.269334	1.000985	
20	(OLDER SINGLES/COUPLES - Premium)	(175.0)	0.067115	0.269069	0.018068	0.269203	1.000499	
0	(MIDAGE SINGLES/COUPLES - Mainstream)	(175.0)	0.044966	0.269069	0.012057	0.268139	0.996544	
37	(YOUNG FAMILIES - Mainstream)	(175.0)	0.048419	0.269069	0.012864	0.265673	0.987381	
16	(OLDER SINGLES/COUPLES - Mainstream)	(175.0)	0.069146	0.269069	0.018339	0.265225	0.985714	
28	(RETIREES - Mainstream)	(175.0)	0.080935	0.269069	0.021460	0.265148	0.985428	
46	(YOUNG SINGLES/COUPLES - Mainstream)	(175.0)	0.079209	0.269069	0.020252	0.255679	0.950239	
18	(OLDER SINGLES/COUPLES - Premium)	(150.0)	0.067115	0.162937	0.011218	0.167150	1.025857	
2	(OLDER FAMILIES - Budget)	(150.0)	0.087193	0.162937	0.014542	0.166775	1.023558	
26	(RETIREES - Mainstream)	(150.0)	0.080935	0.162937	0.013334	0.164747	1.011111	
11	(OLDER SINGLES/COUPLES - Budget)	(150.0)	0.069596	0.162937	0.011393	0.163697	1.004665	

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	repr
22	(RETIREES - Budget)	(150.0)	0.057652	0.162937	0.009399	0.163023	1.000529	
14	(OLDER SINGLES/COUPLES - Mainstream)	(150.0)	0.069146	0.162937	0.011239	0.162534	0.997531	
32	(YOUNG FAMILIES - Budget)	(150.0)	0.071991	0.162937	0.011599	0.161121	0.988859	
44	(YOUNG SINGLES/COUPLES - Mainstream)	(150.0)	0.079209	0.162937	0.012483	0.157593	0.967205	
42	(YOUNG SINGLES/COUPLES - Mainstream)	(134.0)	0.079209	0.101735	0.009382	0.118451	1.164310	

In [59]:
```
# Target rating proportion
target_segment = young_mainstream["PACK_SIZE"].value_counts().sort_values(ascending
target_segment.target /= young_mainstream["PROD_QTY"].sum()

# Other rating proportion
other = not_young_mainstream["PACK_SIZE"].value_counts().sort_values(ascending = Tr
other.other /= not_young_mainstream["PROD_QTY"].sum()

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['othe
brand_proportions.sort_values(by = 'affinity', ascending = False)
```

Out[59]:

	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

Summary Of Analysis

Key Findings and Strategic Implications

The top three customer segments contributing to total sales are:

Budget – Older Families

Mainstream – Young Singles/Couples

Mainstream – Retirees

The dominance of mainstream young singles/couples and retirees in overall sales can be attributed to their large population sizes. In contrast, the strong sales performance of budget older families is driven not by size, but by a notably higher per-customer chip consumption, suggesting deeper engagement with the category.

Among these, mainstream young singles/couples stand out with the highest average spend per transaction, a difference that is statistically significant when compared to their non-mainstream peers.

Delving deeper into this segment reveals valuable behavioral insights:

They are 28% more likely to purchase Tyrells chips compared to other segments.

While Kettles chips remain their top choice (in line with market trends), they are 50% less likely to choose Burger Rings.

These findings suggest a clear preference profile. To capitalize on this, retailers could consider strategic in-store placement—positioning Tyrells and Twisties adjacent to the popular Kettles range. This not only reinforces brand visibility but also aligns with the purchasing behavior of a key target segment.

Additionally, this approach can be adapted for other high-value segments that favor Kettles chips. By customizing shelf layouts to reflect preferred brands and packet sizes, retailers may unlock incremental gains in category sales.

In [ ]: