Background information:

To effectively advise Julia for the upcoming category review, it is essential to analyze the data and comprehend prevailing purchasing trends and behaviors. The client places specific emphasis on understanding customer segments and their buying patterns, particularly in relation to chip purchases.

Main Tasks :

Review transaction data for inconsistencies, missing entries, outliers, accurate categorization, and numeric data across all tables. Similarly, scrutinize customer data for similar issues, identify nulls, and merge transaction and customer data for analysis.

Conduct data analysis and identify customer segments by defining metrics such as total sales, sales drivers, and sources of highest sales. Explore the data, generate charts and graphs, and document noteworthy trends and insights for inclusion in the report to Julia.

Delve deeply into customer segments, formulate recommendations based on insights, specify target segments, assess the relevance of packet sizes, and draw an overall conclusion derived from the analysis.

Importing Libraries

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Mounting Google Drive

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mo
unt("/content/drive", force_remount=True).
```

In [3]:

```
df = pd.read_excel('/content/drive/MyDrive/Colab Notebooks/QVI_transaction_data.xlsx')
```

In [4]:

```
df.head()
```

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

In [5]:
```
df.describe()
```

Out[5]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_QTY	TOT
count	264836.000000	264836.00000	2.648360e+05	2.648360e+05	264836.000000	264836.000000	264836
mean	43464.036260	135.08011	1.355495e+05	1.351583e+05	56.583157	1.907309	7
std	105.389282	76.78418	8.057998e+04	7.813303e+04	32.826638	0.643654	3
min	43282.000000	1.00000	1.000000e+03	1.000000e+00	1.000000	1.000000	1
25%	43373.000000	70.00000	7.002100e+04	6.760150e+04	28.000000	2.000000	5
50%	43464.000000	130.00000	1.303575e+05	1.351375e+05	56.000000	2.000000	7
75%	43555.000000	203.00000	2.030942e+05	2.027012e+05	85.000000	2.000000	9
max	43646.000000	272.00000	2.373711e+06	2.415841e+06	114.000000	200.000000	650

In [6]:
```
df1 = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/QVI_purchase_behaviour.csv')
```

In [7]:
```
df1.head()
```

Out[7]:

	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 [8]:
```
df1.describe()
```

Out[8]:

	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 [9]:
```
df.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
```

Checking & Removing Outliers

In [10]:
```
merged_data = pd.merge(df1, df, on = 'LYLTY_CARD_NBR', how = 'right')
```

In [11]:
```
merged_data.head()
```

Out[11]:

	LYLTY_CARD_NBR	LIFESTAGE	PREMIUM_CUSTOMER	DATE	STORE_NBR	TXN_ID	PROD_NBR
0	1000	YOUNG SINGLES/COUPLES	Premium	43390	1	1	5
1	1307	MIDAGE SINGLES/COUPLES	Budget	43599	1	348	66
2	1343	MIDAGE SINGLES/COUPLES	Budget	43605	1	383	61
3	2373	MIDAGE SINGLES/COUPLES	Budget	43329	2	974	69
4	2426	MIDAGE SINGLES/COUPLES	Budget	43330	2	1038	108

In [12]:
```
print(len(merged_data))
print(len(df))
```

```
264836
264836
```

In [13]:
```
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 264836 entries, 0 to 264835
Data columns (total 10 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   LYLTY_CARD_NBR    264836 non-null  int64  
 1   LIFESTAGE         264836 non-null  object 
 2   PREMIUM_CUSTOMER  264836 non-null  object 
 3   DATE              264836 non-null  int64  
 4   STORE_NBR         264836 non-null  int64  
 5   TXN_ID            264836 non-null  int64  
 6   PROD_NBR          264836 non-null  int64  
 7   PROD_NAME         264836 non-null  object 
 8   PROD_QTY          264836 non-null  int64  
 9   TOT_SALES         264836 non-null  float64
dtypes: float64(1), int64(6), object(3)
memory usage: 22.2+ MB
```

Converting Date to Date-Time Format

In [14]:
```
from datetime import date, timedelta
start = date(1899, 12, 30)
new_date_format = []
for date in merged_data["DATE"]:
  delta = timedelta(date)
  new_date_format.append(start + delta)
```

In [15]:
```
merged_data["DATE"] = pd.to_datetime(pd.Series(new_date_format))
print(merged_data["DATE"].dtype)
```

```
datetime64[ns]
```

Product Name Columns to check all items are Chips

```
In [16]: merged_data["PROD_NAME"].unique()
```

```
Out[16]: 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)
```

In [17]:

```
split_prod = merged_data["PROD_NAME"].str.replace(r'([0-9]+[gG])','').str.replace(r'[^\w
```

```
<ipython-input-17-870fd56d7d3b>:1: FutureWarning: The default value of regex will chan
ge from True to False in a future version.
  split_prod = merged_data["PROD_NAME"].str.replace(r'([0-9]+[gG])','').str.replace(r
'[^\w]',' ').str.split()
```

In [18]:

```
word_counts = {}
def count_words(line):
  for word in line:
    if word not in word_counts:
      word_counts[word] = 1
    else:
        word_counts[word] += 1
split_prod.apply(lambda line: count_words(line))
print(pd.Series(word_counts).sort_values(ascending = False))
```

```
Chips       49770
Kettle      41288
Smiths      28860
Salt        27976
Cheese      27890
            ...  
Sunbites     1432
Pc           1431
Garden       1419
NCC          1419
Fries        1418
Length: 198, dtype: int64
```

```
In [19]: print(merged_data.describe(), '\n')
         print(merged_data.info())
```

```
       LYLTY_CARD_NBR      STORE_NBR        TXN_ID       PROD_NBR  \
count    2.648360e+05  264836.00000  2.648360e+05  264836.000000
mean     1.355495e+05     135.08011  1.351583e+05      56.583157
std      8.057998e+04      76.78418  7.813303e+04      32.826638
min      1.000000e+03       1.00000  1.000000e+00       1.000000
25%      7.002100e+04      70.00000  6.760150e+04      28.000000
50%      1.303575e+05     130.00000  1.351375e+05      56.000000
75%      2.030942e+05     203.00000  2.027012e+05      85.000000
max      2.373711e+06     272.00000  2.415841e+06     114.000000

            PROD_QTY      TOT_SALES
count  264836.000000  264836.000000
mean        1.907309       7.304200
std         0.643654       3.083226
min         1.000000       1.500000
25%         2.000000       5.400000
50%         2.000000       7.400000
75%         2.000000       9.200000
max       200.000000     650.000000

<class 'pandas.core.frame.DataFrame'>
Int64Index: 264836 entries, 0 to 264835
Data columns (total 10 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   LYLTY_CARD_NBR    264836 non-null  int64
 1   LIFESTAGE         264836 non-null  object
 2   PREMIUM_CUSTOMER  264836 non-null  object
 3   DATE              264836 non-null  datetime64[ns]
 4   STORE_NBR         264836 non-null  int64
 5   TXN_ID            264836 non-null  int64
 6   PROD_NBR          264836 non-null  int64
 7   PROD_NAME         264836 non-null  object
 8   PROD_QTY          264836 non-null  int64
 9   TOT_SALES         264836 non-null  float64
dtypes: datetime64[ns](1), float64(1), int64(5), object(3)
memory usage: 22.2+ MB
None
```

```
In [20]: merged_data["PROD_QTY"].value_counts(bins=4).sort_index()
```

```
Out[20]: (0.8, 50.75]       264834
         (50.75, 100.5]          0
         (100.5, 150.25]         0
         (150.25, 200.0]         2
         Name: PROD_QTY, dtype: int64
```

Checking description of PROD_QTY values above 50.75

```
In [21]: merged_data.sort_values(by="PROD_QTY", ascending=False).head(2)
```

Out[21]:

	LYLTY_CARD_NBR	LIFESTAGE	PREMIUM_CUSTOMER	DATE	STORE_NBR	TXN_ID	PROD_NBR
69762	226000	OLDER FAMILIES	Premium	2018-08-19	226	226201	4
69763	226000	OLDER FAMILIES	Premium	2019-05-20	226	226210	4

These two outliers of value 200 in PROD_QTY will be removed. Both entries are by the same customer(LYLTY_CARD_NBR is same) and will be examined by this customer's transactions.

In [22]:
```
merged_data = merged_data[merged_data["PROD_QTY"] < 6]
```

In [23]:
```
len(merged_data[merged_data["LYLTY_CARD_NBR"]==226000])
```

Out[23]:
```
0
```

In [24]:
```
merged_data["DATE"].describe()
```

```
<ipython-input-24-d551bd00c70c>:1: FutureWarning: Treating datetime data as categorica
l rather than numeric in `.describe` is deprecated and will be removed in a future ver
sion of pandas. Specify `datetime_is_numeric=True` to silence this warning and adopt t
he future behavior now.
  merged_data["DATE"].describe()
```

Out[24]:
```
count                  264834
unique                    364
top       2018-12-24 00:00:00
freq                      939
first     2018-07-01 00:00:00
last      2019-06-30 00:00:00
Name: DATE, dtype: object
```

There are only 364 unique values in DATE column that means 1 value is missing.

In [25]:
```
pd.date_range(start=merged_data["DATE"].min(), end=merged_data["DATE"].max()).difference
```

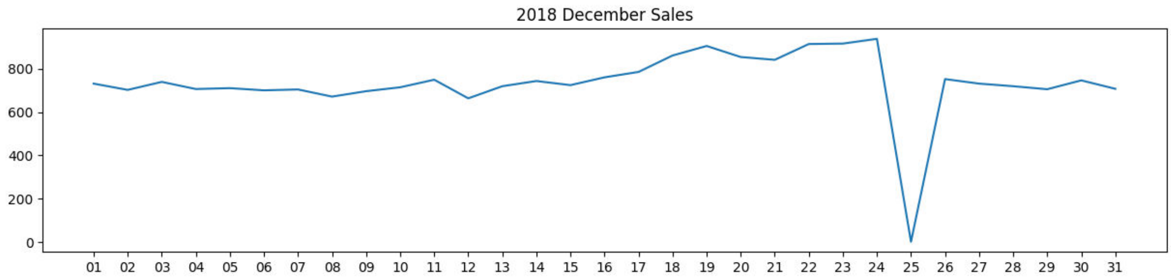
Out[25]:
```
DatetimeIndex(['2018-12-25'], dtype='datetime64[ns]', freq=None)
```

Difference method shows us that 2018-12-25 was the missing date.

In [26]:
```
check_null_date = pd.merge(pd.Series(pd.date_range(start=merged_data["DATE"].min(), end
```

In [27]:
```
trans_by_date = check_null_date["DATE"].value_counts()
dec = trans_by_date[(trans_by_date.index >= pd.datetime(2018, 12, 1)) & (trans_by_date.i
dec.index = dec.index.strftime('%d')
ax = dec.plot(figsize=(15,3))
ax.set_xticks(np.arange(len(dec)))
ax.set_xticklabels(dec.index)
plt.title("2018 December Sales")
plt.savefig("2018 December Sales.png", bbox_inches="tight")
plt.show()
```

```
<ipython-input-27-15e110b159a8>:2: FutureWarning: The pandas.datetime class is depreca
ted and will be removed from pandas in a future version. Import from datetime module i
nstead.
  dec = trans_by_date[(trans_by_date.index >= pd.datetime(2018, 12, 1)) & (trans_by_da
te.index < pd.datetime(2019, 1, 1))].sort_index()
```



In [28]:
```
check_null_date["DATE"].value_counts().sort_values().head()
```

Out[28]:
```
2018-12-25      1
2018-11-25    648
2018-10-18    658
2019-06-13    659
2019-06-24    662
Name: DATE, dtype: int64
```

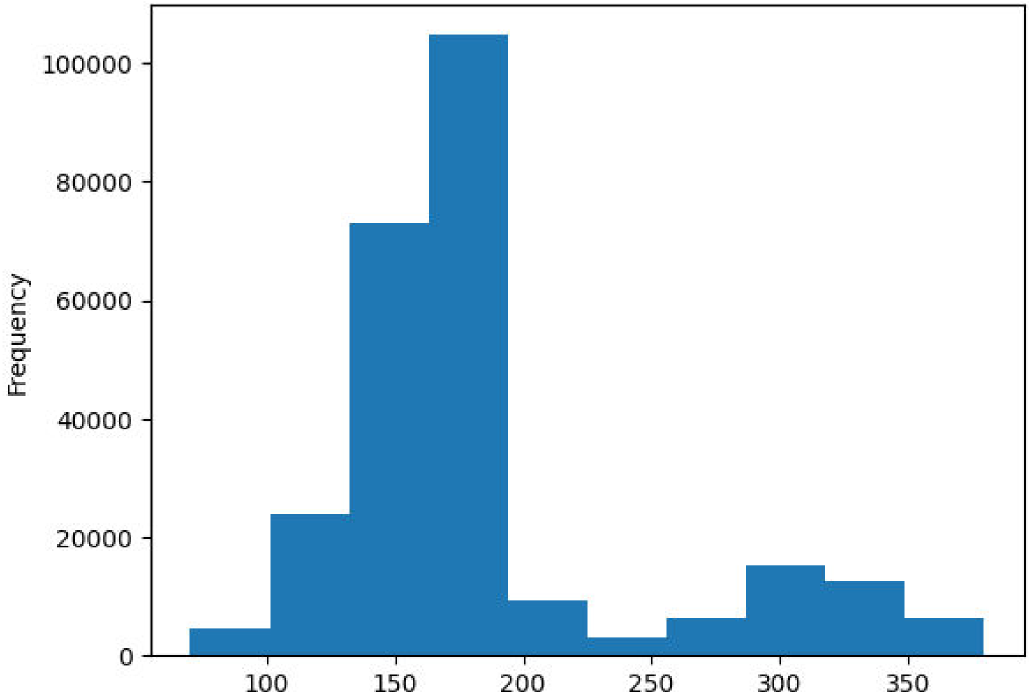
The date with no transactions is 2018-12-25 i.e. Christmas Day and hence store was closed.

Exploring Product Pack sizes.

In [29]:
```
merged_data["PROD_NAME"] = merged_data["PROD_NAME"].str.replace(r'[0-9]+(G)','g')
pack_sizes = merged_data["PROD_NAME"].str.extract(r'([0-9]+[gG])')[0].str.replace("g","
print(pack_sizes.describe())
pack_sizes.plot.hist()
```

```
<ipython-input-29-c0b8f769a815>:1: FutureWarning: The default value of regex will chan
ge from True to False in a future version.
  merged_data["PROD_NAME"] = merged_data["PROD_NAME"].str.replace(r'[0-9]+(G)','g')

count    258770.000000
mean        182.324276
std          64.955035
min          70.000000
25%         150.000000
50%         170.000000
75%         175.000000
max         380.000000
Name: 0, dtype: float64
```

Out[29]:
```
<Axes: ylabel='Frequency'>
```



Smallest pack size is 70g, and biggest pack size is 380g. Product pack size varies reasonably while highest transactions are of mid-sized pack (between 150-200g)

Exploring product brand names.

In [30]:
```
merged_data["PROD_NAME"].str.split().str[0].value_counts().sort_index()
```

Out[30]:
```
Burger         1564
CCs            4551
Cheetos        2927
Cheezels       4603
Cobs           9693
Dorito         3183
Doritos       24962
French         1418
Grain          6272
GrnWves        1468
Infuzions     11057
Infzns         3144
Kettle        41288
NCC            1419
Natural        6050
Old            9324
Pringles      25102
RRD           11894
Red            5885
Smith          2963
Smiths        28860
Snbts          1576
Sunbites       1432
Thins         14075
Tostitos       9471
Twisties       9454
Tyrrells       6442
WW            10320
Woolworths     4437
Name: PROD_NAME, dtype: int64
```

Product names have been written in multiple ways like Dorito and Doritos, Grain and GrnWves, Infuzions and Infzns, etc.

In [31]:
```
merged_data["PROD_NAME"].str.split()[merged_data["PROD_NAME"].str.split().str[0] == "Gra
```

Out[31]:
```
[Grain, Waves, Sweet, Chilli, 210g]      3167
[Grain, Waves, Sour, Cream&Chives, g]    3105
Name: PROD_NAME, dtype: int64
```

In [32]:
```
merged_data["PROD_NAME"].str.split()[merged_data["PROD_NAME"].str.split().str[0] == "Nat
```

Out[32]:
```
[Natural, Chip, Co, Tmato, Hrb&Spce, 175g]       1572
[Natural, ChipCo, Sea, Salt, &, Vinegr, 175g]    1550
[Natural, Chip, Compny, SeaSalt175g]             1468
[Natural, ChipCo, Hony, Soy, Chckn175g]          1460
Name: PROD_NAME, dtype: int64
```

In [33]:
```
merged_data["PROD_NAME"].str.split()[merged_data["PROD_NAME"].str.split().str[0] == "Rec
```

Out[33]:
```
[Red, Rock, Deli, Sp, Salt, &, Truffle, g]        1498
[Red, Rock, Deli, Thai, Chilli&Lime, 150g]        1495
[Red, Rock, Deli, SR, Salsa, &, Mzzrlla, 150g]    1458
[Red, Rock, Deli, Chikn&Garlic, Aioli, 150g]      1434
Name: PROD_NAME, dtype: int64
```

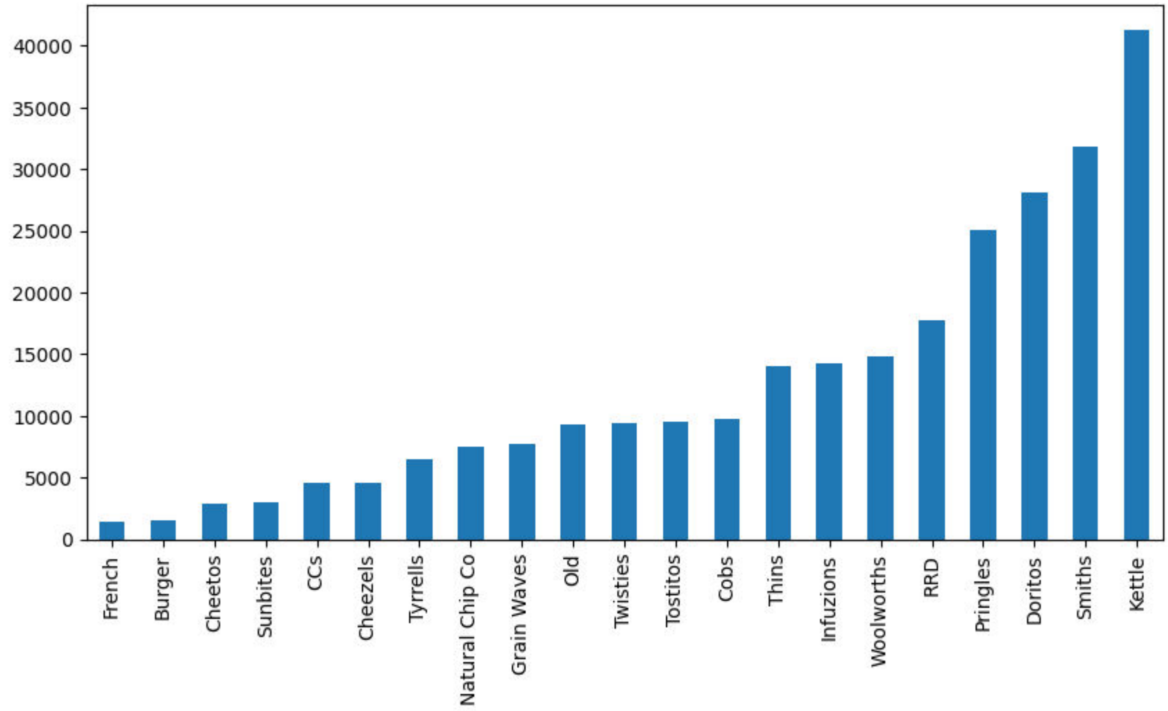
In [34]:
```
merged_data["Cleaned_Brand_Names"] = merged_data["PROD_NAME"].str.split().str[0]
```

```
In [35]: def clean_brand_names(line):
             brand = line["Cleaned_Brand_Names"]
             if brand == "Dorito":
                 return "Doritos"
             elif brand == "GrnWves" or brand == "Grain":
                 return "Grain Waves"
             elif brand == "Infzns":
                 return "Infuzions"
             elif brand == "Natural" or brand == "NCC":
                 return "Natural Chip Co"
             elif brand == "Red":
                 return "RRD"
             elif brand == "Smith":
                 return "Smiths"
             elif brand == "Snbts":
                 return "Sunbites"
             elif brand == "WW":
                 return "Woolworths"
             else:
                 return brand
```

```
In [36]: merged_data["Cleaned_Brand_Names"] = merged_data.apply(lambda line: clean_brand_names(li
```

```
In [37]: merged_data["Cleaned_Brand_Names"].value_counts(ascending=True).plot.bar(figsize=(10,5))
```

```
Out[37]: <Axes: >
```



```
In [38]: merged_data.isnull().sum()
```

```
Out[38]: LYLTY_CARD_NBR         0
         LIFESTAGE              0
         PREMIUM_CUSTOMER       0
         DATE                   0
         STORE_NBR              0
         TXN_ID                 0
         PROD_NBR               0
         PROD_NAME              0
         PROD_QTY               0
         TOT_SALES              0
         Cleaned_Brand_Names    0
         dtype: int64
```

We'll be describing customers by their lifestage and how premium their general purchasing behaviour is.

- No. of customers in each segment
- No. of chips brought/per customer in each segment
- Avg. chip price/customer segment

In [39]:
```
grouped_sales = pd.DataFrame(merged_data.groupby(["LIFESTAGE", "PREMIUM_CUSTOMER"])["TOT
grouped_sales.sort_values(ascending=False, by="sum")
```

Out[39]:

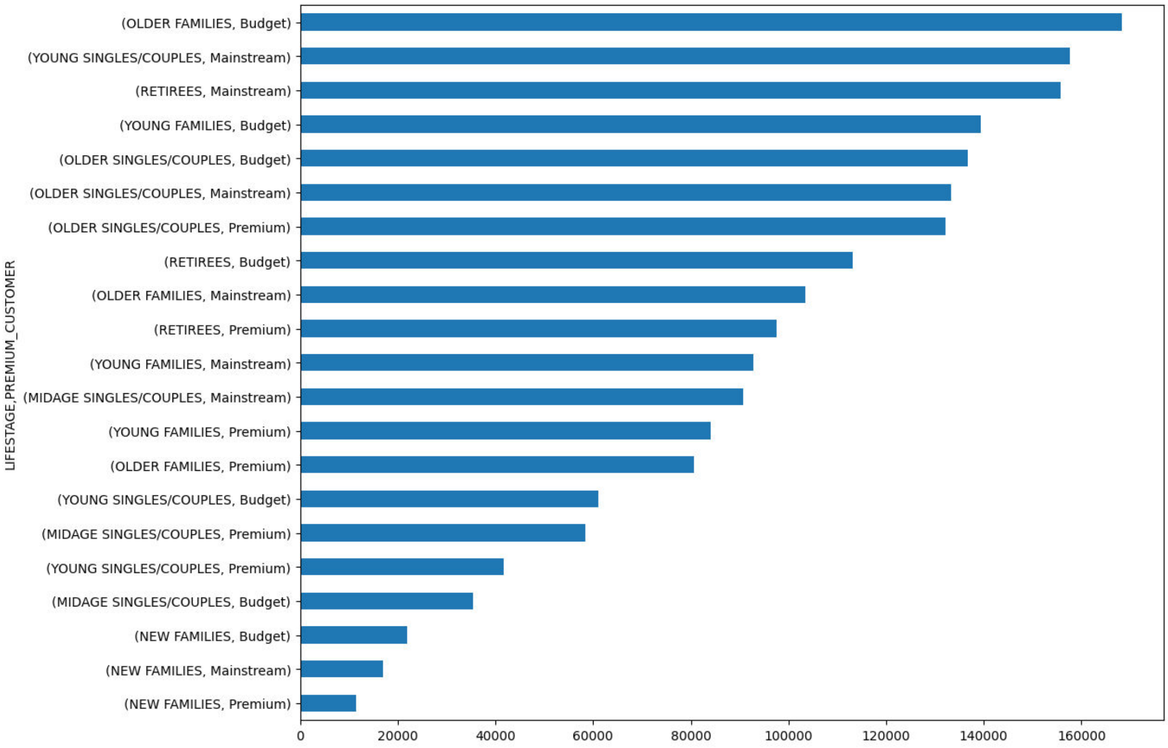
		sum	mean
LIFESTAGE	**PREMIUM_CUSTOMER**		
OLDER FAMILIES	**Budget**	168363.25	7.269570
YOUNG SINGLES/COUPLES	**Mainstream**	157621.60	7.558339
RETIREES	**Mainstream**	155677.05	7.252262
YOUNG FAMILIES	**Budget**	139345.85	7.287201
	Budget	136769.80	7.430315
OLDER SINGLES/COUPLES	**Mainstream**	133393.80	7.282116
	Premium	132263.15	7.449766
RETIREES	**Budget**	113147.80	7.443445
OLDER FAMILIES	**Mainstream**	103445.55	7.262395
RETIREES	**Premium**	97646.05	7.456174
YOUNG FAMILIES	**Mainstream**	92788.75	7.189025
MIDAGE SINGLES/COUPLES	**Mainstream**	90803.85	7.647284
YOUNG FAMILIES	**Premium**	84025.50	7.266756
OLDER FAMILIES	**Premium**	80658.40	7.208079
YOUNG SINGLES/COUPLES	**Budget**	61141.60	6.615624
MIDAGE SINGLES/COUPLES	**Premium**	58432.65	7.112056
YOUNG SINGLES/COUPLES	**Premium**	41642.10	6.629852
MIDAGE SINGLES/COUPLES	**Budget**	35514.80	7.074661
	Budget	21928.45	7.297321
NEW FAMILIES	**Mainstream**	17013.90	7.317806
	Premium	11491.10	7.231655

In [40]:
```
grouped_sales["sum"].sum()
```

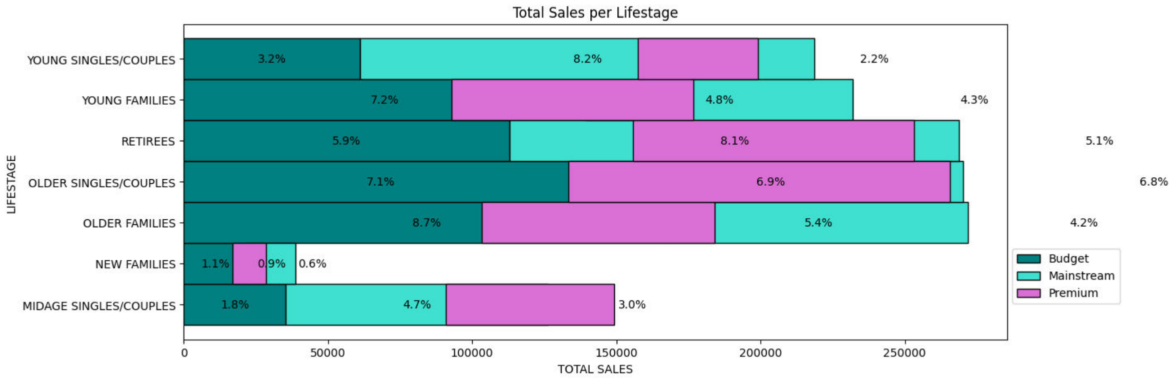
Out[40]:
```
1933115.0000000002
```

```
In [41]: grouped_sales["sum"].sort_values().plot.barh(figsize=(12,10))
```

```
Out[41]: <Axes: ylabel='LIFESTAGE,PREMIUM_CUSTOMER'>
```



```
In [42]: bars1 = grouped_sales[grouped_sales.index.get_level_values("PREMIUM_CUSTOMER") == "Budge
         bars2 = grouped_sales[grouped_sales.index.get_level_values("PREMIUM_CUSTOMER") == "Mains
         bars3 = grouped_sales[grouped_sales.index.get_level_values("PREMIUM_CUSTOMER") == "Premi

         bars1_text = (bars1 / sum(grouped_sales["sum"])).apply("{:.1%}".format)
         bars2_text = (bars2 / sum(grouped_sales["sum"])).apply("{:.1%}".format)
         bars3_text = (bars3 / sum(grouped_sales["sum"])).apply("{:.1%}".format)

         names = grouped_sales.index.get_level_values("LIFESTAGE").unique()
         r = np.arange(len(names))
         plt.figure(figsize=(13,5))

         # Create Budget bars
         budget_bar = plt.barh(r, bars1, edgecolor='black', height=1, label="Budget", color="teal
         # Create Mainstream bars
         mains_bar = plt.barh(r, bars2, left=bars1, edgecolor='black', height=1, label="Mainstrea
         # Create Premium bars
         tmp_bar = np.add(bars1, bars2)
         prem_bar = plt.barh(r, bars3, left=bars2, edgecolor='black', height=1, label="Premium", 

         for i in range(7):
             budget_width = budget_bar[i].get_width()
             budget_main_width = budget_width + mains_bar[i].get_width()
             plt.text(budget_width/2, i, bars1_text[i], va='center', ha='center', size=10)
             plt.text(budget_width + mains_bar[i].get_width()/2, i, bars2_text[i], va='center', h
             plt.text(budget_main_width + prem_bar[i].get_width()/2, i, bars3_text[i], va='center

          # For X-Axis
         plt.yticks(r, names)
         plt.ylabel("LIFESTAGE")
         plt.xlabel("TOTAL SALES")
         plt.legend(loc='center left', bbox_to_anchor=(1.0, 0.2))

         plt.title("Total Sales per Lifestage")

         plt.savefig("lifestage_sales.png", bbox_inches="tight")

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



In [43]:

```
stage_agg_prem = merged_data.groupby("LIFESTAGE")["PREMIUM_CUSTOMER"].agg(pd.Series.mode
print("Top contributor per LIFESTAGE by PREMIUM category")
print(stage_agg_prem)
```

```
Top contributor per LIFESTAGE by PREMIUM category
LIFESTAGE
NEW FAMILIES                  Budget
OLDER FAMILIES                Budget
OLDER SINGLES/COUPLES         Budget
YOUNG FAMILIES                Budget
MIDAGE SINGLES/COUPLES    Mainstream
RETIREES                  Mainstream
YOUNG SINGLES/COUPLES     Mainstream
Name: PREMIUM_CUSTOMER, dtype: object
```

In [44]:

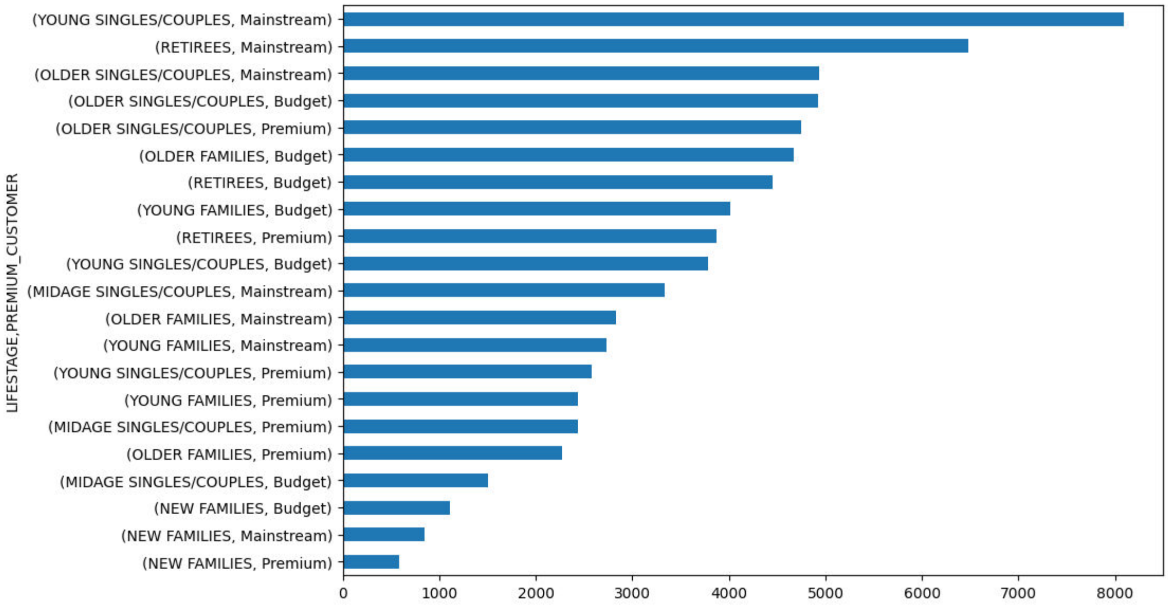
```
unique_cust = merged_data.groupby(["LIFESTAGE", "PREMIUM_CUSTOMER"])["LYLTY_CARD_NBR"].r
pd.DataFrame(unique_cust)
```

Out[44]:

		LYLTY_CARD_NBR
LIFESTAGE	PREMIUM_CUSTOMER	
YOUNG SINGLES/COUPLES	Mainstream	8088
RETIREES	Mainstream	6479
	Mainstream	4930
OLDER SINGLES/COUPLES	Budget	4929
	Premium	4750
OLDER FAMILIES	Budget	4675
RETIREES	Budget	4454
YOUNG FAMILIES	Budget	4017
RETIREES	Premium	3872
YOUNG SINGLES/COUPLES	Budget	3779
MIDAGE SINGLES/COUPLES	Mainstream	3340
OLDER FAMILIES	Mainstream	2831
YOUNG FAMILIES	Mainstream	2728
YOUNG SINGLES/COUPLES	Premium	2574
YOUNG FAMILIES	Premium	2433
MIDAGE SINGLES/COUPLES	Premium	2431
OLDER FAMILIES	Premium	2273
MIDAGE SINGLES/COUPLES	Budget	1504
	Budget	1112
NEW FAMILIES	Mainstream	849
	Premium	588

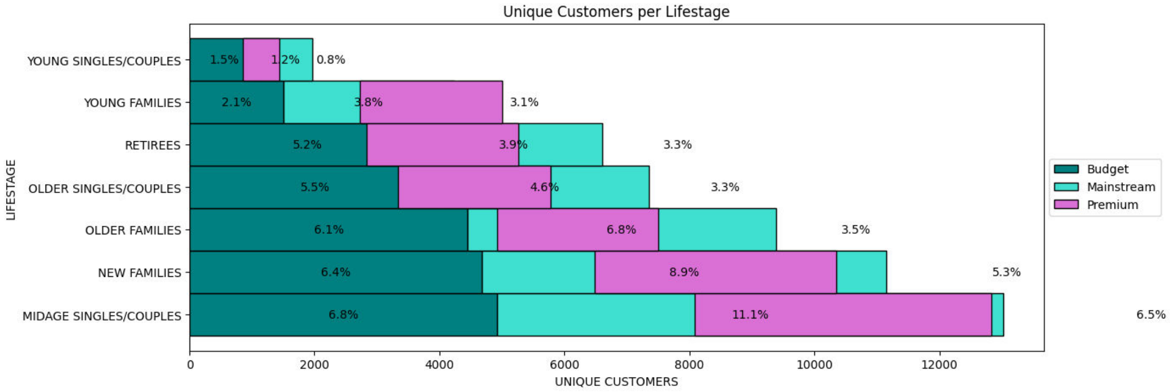
```
In [45]: unique_cust.sort_values().plot.barh(figsize=(10,7))
```

```
Out[45]: <Axes: ylabel='LIFESTAGE,PREMIUM_CUSTOMER'>
```



In [46]:

```
cust_bars1 = unique_cust[unique_cust.index.get_level_values("PREMIUM_CUSTOMER") == "Budg
cust_bars2 = unique_cust[unique_cust.index.get_level_values("PREMIUM_CUSTOMER") == "Mair
cust_bars3 = unique_cust[unique_cust.index.get_level_values("PREMIUM_CUSTOMER") == "Prem

cust_bars1_text = (cust_bars1 / sum(unique_cust)).apply("{:.1%}".format)
cust_bars2_text = (cust_bars2 / sum(unique_cust)).apply("{:.1%}".format)
cust_bars3_text = (cust_bars3 / sum(unique_cust)).apply("{:.1%}".format)

plt.figure(figsize=(13,5))

budget_bar = plt.barh(r, cust_bars1, edgecolor='black', height=1, label="Budget", color=
mains_bar = plt.barh(r, cust_bars2, left=cust_bars1, edgecolor='black', height=1, label=
prem_bar = plt.barh(r, cust_bars3, left=cust_bars2, edgecolor='black', height=1, label='

for i in range(7):
    budget_width = budget_bar[i].get_width()
    budget_main_width = budget_width + mains_bar[i].get_width()
    plt.text(budget_width/2, i, cust_bars1_text[i], va='center', ha='center', size=10)
    plt.text(budget_width + mains_bar[i].get_width()/2, i, cust_bars2_text[i], va='cente
    plt.text(budget_main_width + prem_bar[i].get_width()/2, i, cust_bars3_text[i], va='c

# Custom X axis
plt.yticks(r, names)
plt.ylabel("LIFESTAGE")
plt.xlabel("UNIQUE CUSTOMERS")
plt.legend(loc='center left', bbox_to_anchor=(1.0, 0.5))

plt.title("Unique Customers per Lifestage")

plt.savefig("lifestage_customers.png", bbox_inches="tight")

# # Show graphic
plt.show()
```



The high sales amount by segment "Young Singles/Couples - Mainstream" and "Retirees - Mainstream" have high sales amount because of their large number of unique customers but same trend is not in the "Older - Budget" segment. Next we'll explore if the "Older - Budget" segment has:

High Frequency of Purchase and Average Sales per Customer have been compared to other segments.

```
In [47]: freq_per_cust = merged_data.groupby(["LYLTY_CARD_NBR", "LIFESTAGE", "PREMIUM_CUSTOMER"])
         freq_per_cust.groupby(["LIFESTAGE", "PREMIUM_CUSTOMER"]).agg(["mean", "count"]).sort_val
```

Out[47]:

		mean	count
LIFESTAGE	PREMIUM_CUSTOMER		
OLDER FAMILIES	Mainstream	5.031438	2831
	Budget	4.954011	4675
	Premium	4.923009	2273
YOUNG FAMILIES	Budget	4.760269	4017
	Premium	4.752569	2433
	Mainstream	4.731305	2728
OLDER SINGLES/COUPLES	Premium	3.737684	4750
	Budget	3.734429	4929
	Mainstream	3.715619	4930
MIDAGE SINGLES/COUPLES	Mainstream	3.555090	3340
RETIREES	Budget	3.412887	4454
	Premium	3.382231	3872
MIDAGE SINGLES/COUPLES	Premium	3.379679	2431
	Budget	3.337766	1504
RETIREES	Mainstream	3.313166	6479
NEW FAMILIES	Mainstream	2.738516	849
	Premium	2.702381	588
	Budget	2.702338	1112
YOUNG SINGLES/COUPLES	Mainstream	2.578388	8088
	Budget	2.445621	3779
	Premium	2.440171	2574

The above table describes the "Average frequency of Purchase per segment" and "Unique customer per segment". The top 3 most frequent purchases are done by "Older Families" lifestage segment. "Older - Budget" segment contributes to high sales due to very high Unique number of customers in segment and also high purchase frequency.

```
In [48]: grouped_sales.sort_values(ascending=False, by="mean")
```

Out[48]:

		sum	mean
LIFESTAGE	**PREMIUM_CUSTOMER**		
MIDAGE SINGLES/COUPLES	**Mainstream**	90803.85	7.647284
YOUNG SINGLES/COUPLES	**Mainstream**	157621.60	7.558339
RETIREES	**Premium**	97646.05	7.456174
OLDER SINGLES/COUPLES	**Premium**	132263.15	7.449766
RETIREES	**Budget**	113147.80	7.443445
OLDER SINGLES/COUPLES	**Budget**	136769.80	7.430315
NEW FAMILIES	**Mainstream**	17013.90	7.317806
	Budget	21928.45	7.297321
YOUNG FAMILIES	**Budget**	139345.85	7.287201
OLDER SINGLES/COUPLES	**Mainstream**	133393.80	7.282116
OLDER FAMILIES	**Budget**	168363.25	7.269570
YOUNG FAMILIES	**Premium**	84025.50	7.266756
OLDER FAMILIES	**Mainstream**	103445.55	7.262395
RETIREES	**Mainstream**	155677.05	7.252262
NEW FAMILIES	**Premium**	11491.10	7.231655
OLDER FAMILIES	**Premium**	80658.40	7.208079
YOUNG FAMILIES	**Mainstream**	92788.75	7.189025
MIDAGE SINGLES/COUPLES	**Premium**	58432.65	7.112056
	Budget	35514.80	7.074661
YOUNG SINGLES/COUPLES	**Premium**	41642.10	6.629852
	Budget	61141.60	6.615624

Brand of chips which contribute in Total Sales from top 3 segments

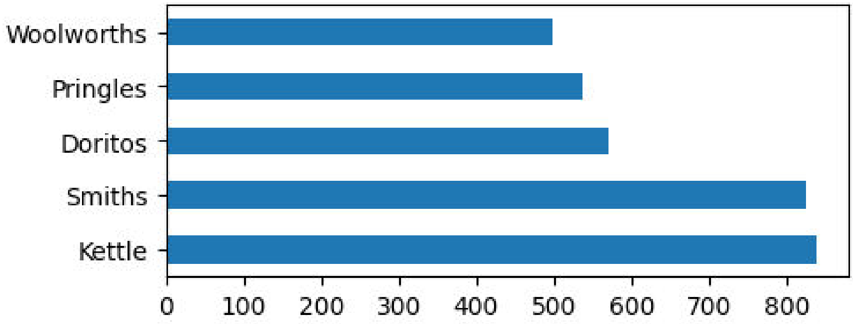
```
In [49]: merged_data.groupby(["LIFESTAGE", "PREMIUM_CUSTOMER"])["Cleaned_Brand_Names"].agg(pd.Ser
```

Out[49]:

```
LIFESTAGE               PREMIUM_CUSTOMER
MIDAGE SINGLES/COUPLES  Budget              Kettle
YOUNG FAMILIES          Premium             Kettle
                        Mainstream          Kettle
                        Budget              Kettle
RETIREES                Premium             Kettle
                        Mainstream          Kettle
                        Budget              Kettle
OLDER SINGLES/COUPLES   Premium             Kettle
YOUNG SINGLES/COUPLES   Mainstream          Kettle
OLDER SINGLES/COUPLES   Mainstream          Kettle
OLDER FAMILIES          Mainstream          Kettle
                        Budget              Kettle
NEW FAMILIES            Premium             Kettle
                        Mainstream          Kettle
                        Budget              Kettle
MIDAGE SINGLES/COUPLES  Premium             Kettle
                        Mainstream          Kettle
OLDER SINGLES/COUPLES   Budget              Kettle
YOUNG SINGLES/COUPLES   Premium             Kettle
OLDER FAMILIES          Premium             Smiths
YOUNG SINGLES/COUPLES   Budget              Smiths
Name: Cleaned_Brand_Names, dtype: object
```

```
In [50]: for stage in merged_data["LIFESTAGE"].unique():
             for prem in merged_data["PREMIUM_CUSTOMER"].unique():
                 print('|',stage, '-', prem,'|')
                 summary = merged_data[(merged_data["LIFESTAGE"] == stage) & (merged_data["PREMIU
                 print(summary)
                 plt.figure()
                 summary.plot.barh(figsize=(5,2))
                 plt.show()
```

```
| YOUNG SINGLES/COUPLES - Premium |
Kettle        838
Smiths        826
Doritos       570
Pringles      537
Woolworths    498
Name: Cleaned_Brand_Names, dtype: int64
```



"Kettle" is the most purchased brand from every segment. While "Smiths" is the second most purchased brand in all segments except "YOUNG SINGLES/COUPLES Mainstream" which had Doritos as their second most purchased brand.

In [51]:

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

temp = merged_data.reset_index().rename(columns = {"index": "transaction"})
temp["Segment"] = temp["LIFESTAGE"] + ' - ' + temp['PREMIUM_CUSTOMER']
segment_brand_encode = pd.concat([pd.get_dummies(temp["Segment"]), pd.get_dummies(temp['

frequent_sets = apriori(segment_brand_encode, min_support=0.01, use_colnames=True)
rules = association_rules(frequent_sets, metric="lift", min_threshold=1)

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

```
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: Dep
recationWarning: DataFrames with non-bool types result in worse computationalperforman
ce and their support might be discontinued in the future.Please use a DataFrame with b
ool type
  warnings.warn(
```

Out[51]:

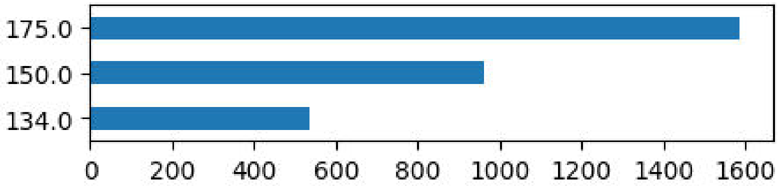
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	con
1	(OLDER FAMILIES - Budget)	(Smiths)	0.087451	0.120162	0.011679	0.133549	1.111409	0.001171	1.0
3	(OLDER SINGLES/COUPLES - Budget)	(Kettle)	0.069504	0.155901	0.011573	0.166513	1.068064	0.000738	1.0
5	(OLDER SINGLES/COUPLES - Premium)	(Kettle)	0.067038	0.155901	0.011128	0.165991	1.064716	0.000676	1.0
7	(RETIREES - Mainstream)	(Kettle)	0.081055	0.155901	0.012785	0.157738	1.011779	0.000149	1.0
8	(YOUNG SINGLES/COUPLES - Mainstream)	(Kettle)	0.078744	0.155901	0.014515	0.184329	1.182344	0.002239	1.0

By our analysis to this point, we can conclude that "Kettle" is the brand of choice for most segments.

Now, we'll find the pack size preferences among different segments.

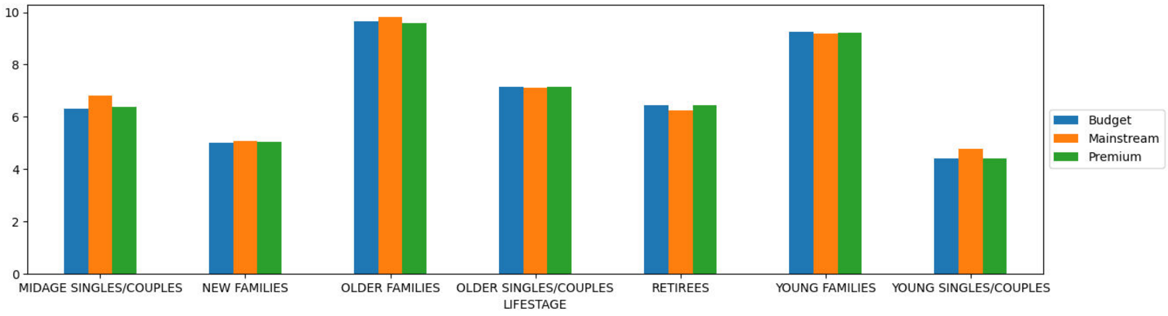
```
In [52]: merged_pack = pd.concat([merged_data, pack_sizes.rename("Pack_Size")], axis=1)

         for stage in merged_data["LIFESTAGE"].unique():
             for prem in merged_data["PREMIUM_CUSTOMER"].unique():
                 print('|',stage, '-', prem,'|')
                 summary = merged_pack[(merged_pack["LIFESTAGE"] == stage) & (merged_pack["PREMIU
                 print(summary)
                 plt.figure()
                 summary.plot.barh(figsize=(5,1))
                 plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning:
`should_run_async` will not call `transform_cell` automatically in the future. Please
pass the result to `transformed_cell` argument and any exception that happen during th
etransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)

| YOUNG SINGLES/COUPLES - Premium |
134.0     537
150.0     961
175.0    1587
Name: Pack_Size, dtype: int64
```



```
In [53]: (temp.groupby(["LIFESTAGE", "PREMIUM_CUSTOMER"])["PROD_QTY"].sum() / temp.groupby(["LIFE
         plt.legend(loc="center left", bbox_to_anchor=(1.0, 0.5))
         plt.savefig("Average purchase quantity per segment.png", bbox_inches="tight")
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning:
`should_run_async` will not call `transform_cell` automatically in the future. Please
pass the result to `transformed_cell` argument and any exception that happen during th
etransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```



Avg. Chips price per transaction by segments

In [54]:

```
temp["Unit_Price"] = temp["TOT_SALES"] / temp["PROD_QTY"]
temp.groupby(["Segment"]).mean()["Unit_Price"].sort_values(ascending=False)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning:
`should_run_async` will not call `transform_cell` automatically in the future. Please
pass the result to `transformed_cell` argument and any exception that happen during th
etransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
<ipython-input-54-e5bc58e74ee7>:2: FutureWarning: The default value of numeric_only in
DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to
False. Either specify numeric_only or select only columns which should be valid for th
e function.
  temp.groupby(["Segment"]).mean()["Unit_Price"].sort_values(ascending=False)
```

Out[54]:

```
Segment
YOUNG SINGLES/COUPLES - Mainstream     4.071485
MIDAGE SINGLES/COUPLES - Mainstream    4.000101
RETIREES - Budget                      3.924883
RETIREES - Premium                     3.921323
NEW FAMILIES - Budget                  3.919251
NEW FAMILIES - Mainstream              3.916581
OLDER SINGLES/COUPLES - Premium        3.887220
OLDER SINGLES/COUPLES - Budget         3.877022
NEW FAMILIES - Premium                 3.871743
RETIREES - Mainstream                  3.833343
OLDER SINGLES/COUPLES - Mainstream     3.803800
YOUNG FAMILIES - Budget                3.753659
MIDAGE SINGLES/COUPLES - Premium       3.752915
YOUNG FAMILIES - Premium               3.752402
OLDER FAMILIES - Budget                3.733344
MIDAGE SINGLES/COUPLES - Budget        3.728496
OLDER FAMILIES - Mainstream            3.727383
YOUNG FAMILIES - Mainstream            3.707097
OLDER FAMILIES - Premium               3.704625
YOUNG SINGLES/COUPLES - Premium        3.645518
YOUNG SINGLES/COUPLES - Budget         3.637681
Name: Unit_Price, dtype: float64
```

In [55]:

```
a = temp.groupby(["Segment", "Cleaned_Brand_Names"]).sum()["TOT_SALES"].sort_values(asce
a[a["Segment"] == "YOUNG SINGLES/COUPLES - Mainstream"]
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning:
`should_run_async` will not call `transform_cell` automatically in the future. Please
pass the result to `transformed_cell` argument and any exception that happen during th
etransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
<ipython-input-55-772218410f43>:1: FutureWarning: The default value of numeric_only in
DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to
False. Either specify numeric_only or select only columns which should be valid for th
e function.
  a = temp.groupby(["Segment", "Cleaned_Brand_Names"]).sum()["TOT_SALES"].sort_values
(ascending=False).reset_index()
```

Out[55]:

	Segment	Cleaned_Brand_Names	TOT_SALES
0	YOUNG SINGLES/COUPLES - Mainstream	Kettle	35423.6
8	YOUNG SINGLES/COUPLES - Mainstream	Doritos	21705.9
23	YOUNG SINGLES/COUPLES - Mainstream	Pringles	16006.2
24	YOUNG SINGLES/COUPLES - Mainstream	Smiths	15265.7
55	YOUNG SINGLES/COUPLES - Mainstream	Infuzions	8749.4
59	YOUNG SINGLES/COUPLES - Mainstream	Old	8180.4
65	YOUNG SINGLES/COUPLES - Mainstream	Twisties	7539.8
73	YOUNG SINGLES/COUPLES - Mainstream	Tostitos	7238.0
74	YOUNG SINGLES/COUPLES - Mainstream	Thins	7217.1
92	YOUNG SINGLES/COUPLES - Mainstream	Cobs	6144.6
124	YOUNG SINGLES/COUPLES - Mainstream	RRD	4958.1
129	YOUNG SINGLES/COUPLES - Mainstream	Tyrrells	4800.6
148	YOUNG SINGLES/COUPLES - Mainstream	Grain Waves	4201.0
189	YOUNG SINGLES/COUPLES - Mainstream	Cheezels	3318.3
246	YOUNG SINGLES/COUPLES - Mainstream	Natural Chip Co	2130.0
258	YOUNG SINGLES/COUPLES - Mainstream	Woolworths	1929.8
318	YOUNG SINGLES/COUPLES - Mainstream	Cheetos	898.8
327	YOUNG SINGLES/COUPLES - Mainstream	CCs	850.5
383	YOUNG SINGLES/COUPLES - Mainstream	French	429.0
393	YOUNG SINGLES/COUPLES - Mainstream	Sunbites	391.0
415	YOUNG SINGLES/COUPLES - Mainstream	Burger	243.8

Trends and Insights :

1. Top 3 segments in total sale contribution:

- Older families (Budget) $156,864
- Young Singles/Couples (Mainstream) $147,582
- Retirees (Mainstream) $145,169

2. Highest population - High Total Sales

- Young Singles/Couples (Mainstream)
- Retirees (Mainstream).

3. Older Families have the highest frequency of purchase which turns into high total sales while they don't have the highest population.
4. Highest avg. quantity of chips bought per purchase

- Older Families
- Young Families

5. The Mainstream category of the "Young and Midage Singles/Couples" have done the highest spend on chips per purchase.
6. "Kettle" is the most purchased brand in all segments.
7. The second most purchased brand is "Smiths" in all segments except in "Young Midage Singles/Couples" which have "Doritos" as second most purchased.
8. 175g is most purchased chips packet size followed by 150g in all segments.

General Views and Recommendations:

1. **General:** All segments have "Kettle" as the most frequently purchased brand. Chips packet size 175g (among all brands) followed by 150g are most preferred. These two insights should be considered when building plans and strategies.
2. **Older Families Segment:**

They focus on budget segment and their strenght lies in frequent purchases. Offers and incentives against number of purchases made would attract more customers.

3. **Young Singles/Couples:**

Focus on Mainstream segment. They had "Doritos" as second most purchased brand. To specifically target this segment it might be a good idea to collaborate with Doritos merchant to do some branding promotion catered to "Young Singles/Couples - Mainstream" segment. Collaborating with "Doritos" to introduce offers and promotions to attract more customers from "Young Singles/Couples". They have high population quantity and that's the strength.

4. **Retirees:**

Focus on the Mainstream segment. Their strength is also population quantity. We can make our offers and promotions reach all customers timely.