In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
behavior_df = pd.read_csv("datasets/QVI_purchase_behaviour.csv")
```

In [3]:
```
transaction_df = pd.read_excel("datasets/QVI_transaction_data.xlsx")
```

Examine Transaction Data

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

Out[4]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_
0	43390	1	1000	1	5	Natural Chip Compny SeaSalt175g	2	
1	43599	1	1307	348	66	CCs Nacho Cheese 175g	3	
2	43605	1	1343	383	61	Smiths Crinkle Cut Chips Chicken 170g	2	
3	43329	2	2373	974	69	Smiths Chip Thinly S/Cream&Onion 175g	5	
4	43330	2	2426	1038	108	Kettle Tortilla ChpsHny&Jlpno Chili 150g	3	

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

Out[5]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_QT
count	264836.000000	264836.00000	2.648360e+05	2.648360e+05	264836.000000	264836.0000(
mean	43464.036260	135.08011	1.355495e+05	1.351583e+05	56.583157	1.9073(
std	105.389282	76.78418	8.057998e+04	7.813303e+04	32.826638	0.6436
min	43282.000000	1.00000	1.000000e+03	1.000000e+00	1.000000	1.0000(
25%	43373.000000	70.00000	7.002100e+04	6.760150e+04	28.000000	2.0000(
50%	43464.000000	130.00000	1.303575e+05	1.351375e+05	56.000000	2.0000(
75%	43555.000000	203.00000	2.030942e+05	2.027012e+05	85.000000	2.0000(
max	43646.000000	272.00000	2.373711e+06	2.415841e+06	114.000000	200.0000(

In [6]:
```
transaction_df.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  int64  
 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: float64(1), int64(6), object(1)
memory usage: 16.2+ MB
```

In [7]:
```
import datetime as dt
```

In [8]:
```
# convert excel style date into datetime format
transaction_df['DATE'] = pd.TimedeltaIndex(transaction_df['DATE'],unit='d')
```

In [9]:
```
transaction_df.head()
```

Out[9]:

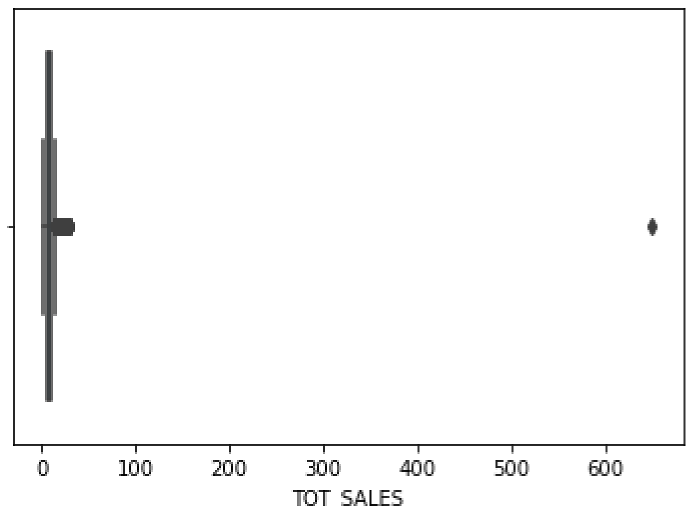
	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_
0	2018-10-17	1	1000	1	5	Natural Chip Compny SeaSalt175g	2	
1	2019-05-14	1	1307	348	66	CCs Nacho Cheese 175g	3	
2	2019-05-20	1	1343	383	61	Smiths Crinkle Cut Chips Chicken 170g	2	
3	2018-08-17	2	2373	974	69	Smiths Chip Thinly S/Cream&Onion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

In [10]:
```
# check for missing data
transaction_df.isnull().sum()
```

Out[10]:
```
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 [11]:
```
# check for outliers with a boxplot
sns.boxplot(x=transaction_df['TOT_SALES'])
```

Out[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24f34ef2e48>
```



In [12]:
```
from scipy import stats
```

In [13]:

```
# find outliers using z scores

# find the z scores for total_sales
z = np.abs(stats.zscore(transaction_df['TOT_SALES']))
# anything above and below 3 and -3 respectively will be classified as outli
threshold = 3

# check how many outliers exist for the total sales column
print(len(np.where(z > 3)[0]))

# check dataframe shape (rows and columns)
print(transaction_df.shape)

# remove the outliers
transaction_df = transaction_df[(z<3)]

# check if changes are made
print(transaction_df.shape)
```

```
439
(264836, 8)
(264397, 8)
```

In [14]:

```
transaction_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 264397 entries, 0 to 264835
Data columns (total 8 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   DATE            264397 non-null  datetime64[ns]
 1   STORE_NBR       264397 non-null  int64
 2   LYLTY_CARD_NBR  264397 non-null  int64
 3   TXN_ID          264397 non-null  int64
 4   PROD_NBR        264397 non-null  int64
 5   PROD_NAME       264397 non-null  object
 6   PROD_QTY        264397 non-null  int64
 7   TOT_SALES       264397 non-null  float64
dtypes: datetime64[ns](1), float64(1), int64(5), object(1)
memory usage: 18.2+ MB
```

In [15]:

```
# extract the 175 g from the prod_name column and place in a new column
prod_size_df = transaction_df['PROD_NAME'].str.extract("(\d+)")

# get the index number of the prod_name column
prod_name_column_loc = transaction_df.columns.get_loc("PROD_NAME")

# insert at a specific index in the dataframe
transaction_df.insert(prod_name_column_loc+1, "PROD_SIZE", prod_size_df)
```

In [16]:
```
# check if changes are made
transaction_df.head()
```

Out[16]:

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

In [17]:
```
# change datatype for PROD_NAME
transaction_df['PROD_SIZE'] = pd.to_numeric(transaction_df['PROD_SIZE'])
```

In [18]:
```
transaction_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 264397 entries, 0 to 264835
Data columns (total 9 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   DATE            264397 non-null  datetime64[ns]
 1   STORE_NBR       264397 non-null  int64
 2   LYLTY_CARD_NBR  264397 non-null  int64
 3   TXN_ID          264397 non-null  int64
 4   PROD_NBR        264397 non-null  int64
 5   PROD_NAME       264397 non-null  object
 6   PROD_SIZE       264397 non-null  int64
 7   PROD_QTY        264397 non-null  int64
 8   TOT_SALES       264397 non-null  float64
dtypes: datetime64[ns](1), float64(1), int64(6), object(1)
memory usage: 20.2+ MB
```

In [19]:
```
transaction_df.head(3)
```

Out[19]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_SIZE	PROD
0	2018-10-17	1	1000	1	5	Natural Chip Compny SeaSalt175g	175	
1	2019-05-14	1	1307	348	66	CCs Nacho Cheese 175g	175	
2	2019-05-20	1	1343	383	61	Smiths Crinkle Cut Chips Chicken 170g	170	

In [20]:
```
import re
```

In [21]:
```
transaction_df.reset_index(drop=True,inplace=True)
```

In [22]:
```
for i in range(transaction_df.shape[0]):
    edited_text = re.sub('\d\w*',"",transaction_df['PROD_NAME'][i])
    transaction_df['PROD_NAME'][i] = edited_text
```

```
C:\Users\smartestpersonalive\Anaconda3\lib\site-packages\ipykernel_launcher.py:
3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  This is separate from the ipykernel package so we can avoid doing imports unt
il
```

In [23]:
```
# save a checkpoint
transactoin_checkpoint1_df = transaction_df
```

In [24]:
```
transaction_df.head()
```

Out[24]:

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

In [25]:
```
# drop rows that contain the word "salsa" (not chips)
# some products here are not actually chips

# set all words to lower case before finding the word "salsa"
transaction_df['PROD_NAME'] = transaction_df['PROD_NAME'].apply(lambda x: x.

# select only rows that does NOT contain salsa
transaction_df = transaction_df.loc[~transaction_df['PROD_NAME'].str.contain

# change words back to upper case
# transaction_df['PROD_NAME'] = transaction_df['PROD_NAME'].apply(lambda x: 

```

In [ ]:
```

```

In [26]:
```
transaction_df.head()
```

Out[26]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_SIZE	PROD
0	2018-10-17	1	1000	1	5	natural chip compny seasalt	175	
1	2019-05-14	1	1307	348	66	ccs nacho cheese	175	
2	2019-05-20	1	1343	383	61	smiths crinkle cut chips chicken	170	
3	2018-08-17	2	2373	974	69	smiths chip thinly s/cream&onion	175	
4	2018-08-18	2	2426	1038	108	kettle tortilla chpshny&jlpno chili	150	

In [27]:
```
# check if data is recorded correctly for each product number
for i in transaction_df['PROD_NBR'].unique():
    unique_value_check = transaction_df.loc[transaction_df['PROD_NBR'] == i]
    if unique_value_check != 1:
        print(i)
```

In [28]:
```
# check date numbers
transaction_df['DATE'].nunique()
# looks like it's missing a number...
```

Out[28]:
```
364
```

In [29]:
```
# check date range
start_date = transaction_df['DATE'].min()
end_date = transaction_df['DATE'].max()
print(start_date)
print(end_date)

# check for any missing dates
pd.date_range(start = start_date, end = end_date).difference(transaction_df[
```

```
2018-07-01 00:00:00
2019-06-30 00:00:00
```

Out[29]:
```
DatetimeIndex(['2018-12-25'], dtype='datetime64[ns]', freq=None)
```

December 25 is missing from the dataset but it's also Christmas day so the stores are probably closed

In [30]:
```
# determine number of stores
transaction_df['STORE_NBR'].nunique()
```
Out[30]: 271

In [31]:
```
# check number of products
transaction_df['PROD_NBR'].nunique()
```
Out[31]: 105

In [32]:
```
# check number of different customers
transaction_df['LYLTY_CARD_NBR'].nunique()
```
Out[32]: 71253

In [33]:
```
# check prod_quantity unique values
transaction_df['PROD_QTY'].unique()
```
Out[33]: array([2, 3, 5, 1, 4], dtype=int64)

In [34]:
```
# check max and min values to see if they make sense
transaction_df.describe()
```
Out[34]:

	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_SIZE	PROD_Q
count	246331.000000	2.463310e+05	2.463310e+05	246331.000000	246331.000000	246331.000
mean	135.045573	1.355260e+05	1.351258e+05	56.357795	175.541012	1.9021
std	76.790799	8.072522e+04	7.815159e+04	33.693425	59.383908	0.3250
min	1.000000	1.000000e+03	1.000000e+00	1.000000	70.000000	1.0000
25%	70.000000	7.001400e+04	6.756150e+04	26.000000	150.000000	2.0000
50%	130.000000	1.303660e+05	1.351770e+05	53.000000	170.000000	2.0000
75%	203.000000	2.030845e+05	2.026565e+05	87.000000	175.000000	2.0000
max	272.000000	2.373711e+06	2.415841e+06	114.000000	380.000000	5.0000

In [35]:
```
# find the product that have the highest sales numbers
transaction_df.groupby(by='PROD_NBR').mean().sort_values(by='TOT_SALES', asc
```

Out[35]:

PROD_NBR	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_SIZE	PROD_QTY	TOT_SALES
4	138.061282	139191.621307	138146.721873	380.0	1.918919	12.26665€
14	133.453782	134326.274199	133572.684407	380.0	1.894180	11.175661
20	135.156229	135324.622281	135176.685893	330.0	1.909031	10.88147
7	134.931310	135102.980511	135027.150479	330.0	1.907987	10.87552
16	134.323502	135431.010668	134375.604644	330.0	1.903044	10.84734
...	...	...	...	...	...	..
105	134.209103	134356.603562	134167.830475	190.0	1.894459	3.41002€
55	135.064721	135210.847716	135172.942259	90.0	1.894670	3.22093
72	136.957447	137106.179433	136959.118440	175.0	1.890780	3.21432€
95	137.055866	138780.236732	137245.493017	90.0	1.889665	3.21243
92	134.734833	134884.648262	134794.939332	175.0	1.885481	3.205317

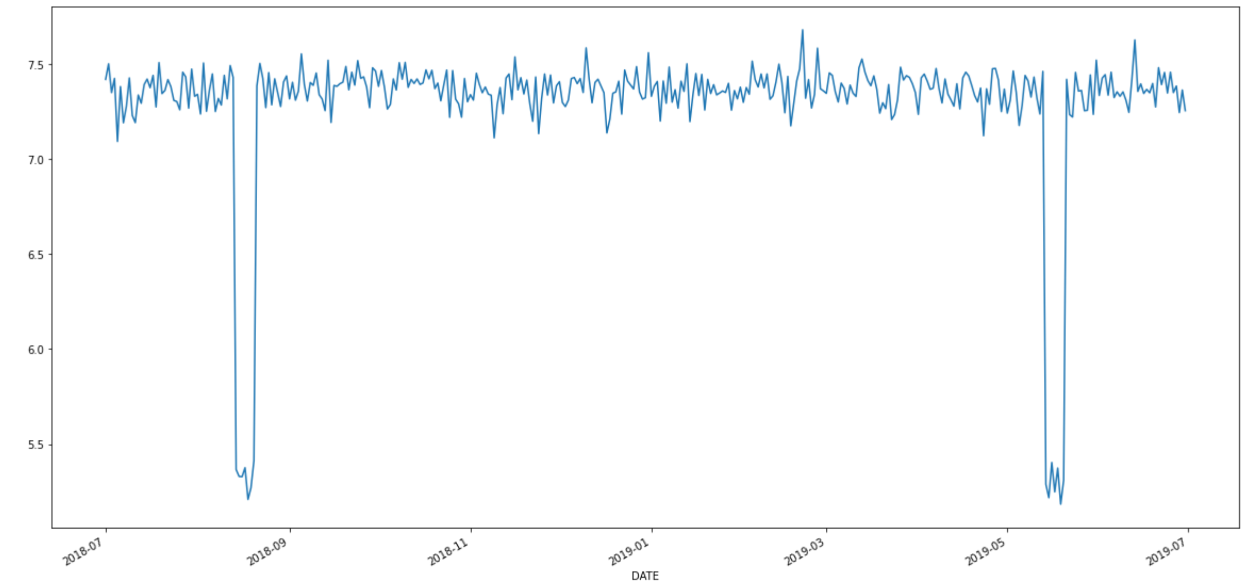
105 rows × 6 columns

check for seasonalities with sales

In [36]:
```
# find any trends
date_group = transaction_df.sort_values(by="DATE").groupby(by='DATE').mean()
date_group.head(3)
```

Out[36]:
```
DATE
2018-07-01    7.420965
2018-07-02    7.503077
2018-07-03    7.351187
Name: TOT_SALES, dtype: float64
```

In [37]:
```
plt.figure(figsize=(20,10))
date_group.plot()
plt.savefig("chip_sales.jpg")
```



In [38]:
```
some_df = date_group.reset_index()
some_df.head(3)
```

Out[38]:

	DATE	TOT_SALES
0	2018-07-01	7.420965
1	2018-07-02	7.503077
2	2018-07-03	7.351187

In [39]:
```
some_df.loc[some_df['DATE'].dt.month==8]
```

Out[39]:

	DATE	TOT_SALES
31	2018-08-01	7.342353
32	2018-08-02	7.238416
33	2018-08-03	7.507251
34	2018-08-04	7.252782
35	2018-08-05	7.364397
36	2018-08-06	7.449858
37	2018-08-07	7.251647
38	2018-08-08	7.320432
39	2018-08-09	7.285736
40	2018-08-10	7.442222
41	2018-08-11	7.318142
42	2018-08-12	7.493925
43	2018-08-13	7.431010
44	2018-08-14	5.365556
45	2018-08-15	5.329279
46	2018-08-16	5.327837
47	2018-08-17	5.375396
48	2018-08-18	5.208232
49	2018-08-19	5.268887
50	2018-08-20	5.411707
51	2018-08-21	7.386830
52	2018-08-22	7.505225
53	2018-08-23	7.425575
54	2018-08-24	7.272179
55	2018-08-25	7.456621
56	2018-08-26	7.286715
57	2018-08-27	7.423881
58	2018-08-28	7.351730
59	2018-08-29	7.277928
60	2018-08-30	7.407504
61	2018-08-31	7.439210

In [40]:
```
some_df.loc[some_df["DATE"].dt.month==5]
```

Out[40]:

	DATE	TOT_SALES
303	2019-05-01	7.243235
304	2019-05-02	7.311994
305	2019-05-03	7.465449
306	2019-05-04	7.353968
307	2019-05-05	7.178088
308	2019-05-06	7.284300
309	2019-05-07	7.441679
310	2019-05-08	7.413181
311	2019-05-09	7.327695
312	2019-05-10	7.433233
313	2019-05-11	7.322386
314	2019-05-12	7.238428
315	2019-05-13	7.463323
316	2019-05-14	5.287974
317	2019-05-15	5.217224
318	2019-05-16	5.402904
319	2019-05-17	5.247731
320	2019-05-18	5.373422
321	2019-05-19	5.183166
322	2019-05-20	5.309164
323	2019-05-21	7.421311
324	2019-05-22	7.235662
325	2019-05-23	7.222433
326	2019-05-24	7.457742
327	2019-05-25	7.359149
328	2019-05-26	7.362654
329	2019-05-27	7.255940
330	2019-05-28	7.258126
331	2019-05-29	7.444538
332	2019-05-30	7.236472
333	2019-05-31	7.522289

may 14-20 and August 14-20 have ower sales on average compared to other days

Graph number of purchases over time

In [41]:
```
transaction_df.head(3)
```

Out[41]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_SIZE	PROD_
0	2018-10-17	1	1000	1	5	natural chip compny seasalt	175	
1	2019-05-14	1	1307	348	66	ccs nacho cheese	175	
2	2019-05-20	1	1343	383	61	smiths crinkle cut chips chicken	170	

In [42]:
```
# group by date and look at count (number of purchases)
purchase_count = transaction_df.sort_values(by='DATE', ascending=True)\
                                .groupby(by='DATE')\
                                .count()['STORE_NBR']
```

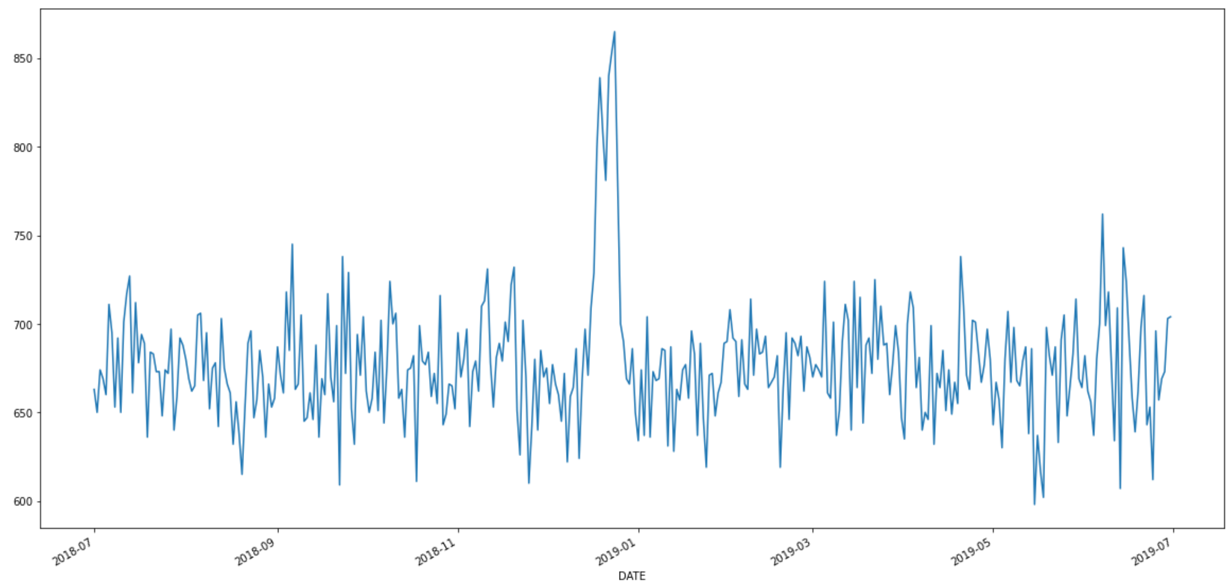
In [43]:
```
purchase_count
```

Out[43]:
```
DATE
2018-07-01    663
2018-07-02    650
2018-07-03    674
2018-07-04    669
2018-07-05    660
             ...
2019-06-26    657
2019-06-27    669
2019-06-28    673
2019-06-29    703
2019-06-30    704
Name: STORE_NBR, Length: 364, dtype: int64
```

In [44]:

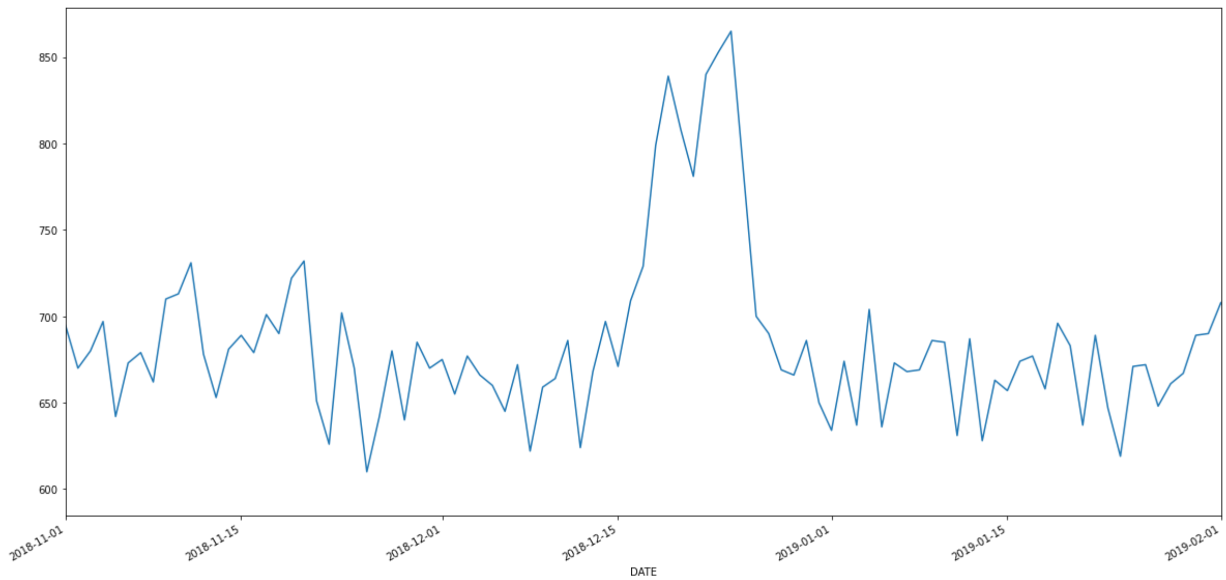
```
plt.figure(figsize=(20,10))
purchase_count.plot()
```

Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x24f3fdc2bc8>



In [45]:
```
plt.figure(figsize=(20,10))
purchase_count.plot()
plt.xlim(["2018-11-01", "2019-02-01"])
```

Out[45]: (736999.0, 737091.0)



In [46]:
```
purchase_count_df = pd.DataFrame(purchase_count)
```

In [47]:
```
purchase_count_df["2018-12-01":"2019-01-15"]

```

Out[47]:

	STORE_NBR
DATE	
2018-12-01	675
2018-12-02	655
2018-12-03	677
2018-12-04	666
2018-12-05	660
2018-12-06	645
2018-12-07	672
2018-12-08	622
2018-12-09	659
2018-12-10	664
2018-12-11	686
2018-12-12	624
2018-12-13	668
2018-12-14	697
2018-12-15	671
2018-12-16	709
2018-12-17	729
2018-12-18	799
2018-12-19	839
2018-12-20	808
2018-12-21	781
2018-12-22	840
2018-12-23	853
2018-12-24	865
2018-12-26	700
2018-12-27	690
2018-12-28	669
2018-12-29	666
2018-12-30	686
2018-12-31	650
2019-01-01	634
2019-01-02	674

	STORE_NBR
DATE	
2019-01-03	637
2019-01-04	704
2019-01-05	636
2019-01-06	673
2019-01-07	668
2019-01-08	669
2019-01-09	686
2019-01-10	685
2019-01-11	631
2019-01-12	687
2019-01-13	628
2019-01-14	663
2019-01-15	657

we can see a surge in number of purchases from December 17 to December 24. This surge could be explained by Christmas where customers want to do some last minute shopping or want to shop before the stores close on the 25th.

In [48]:

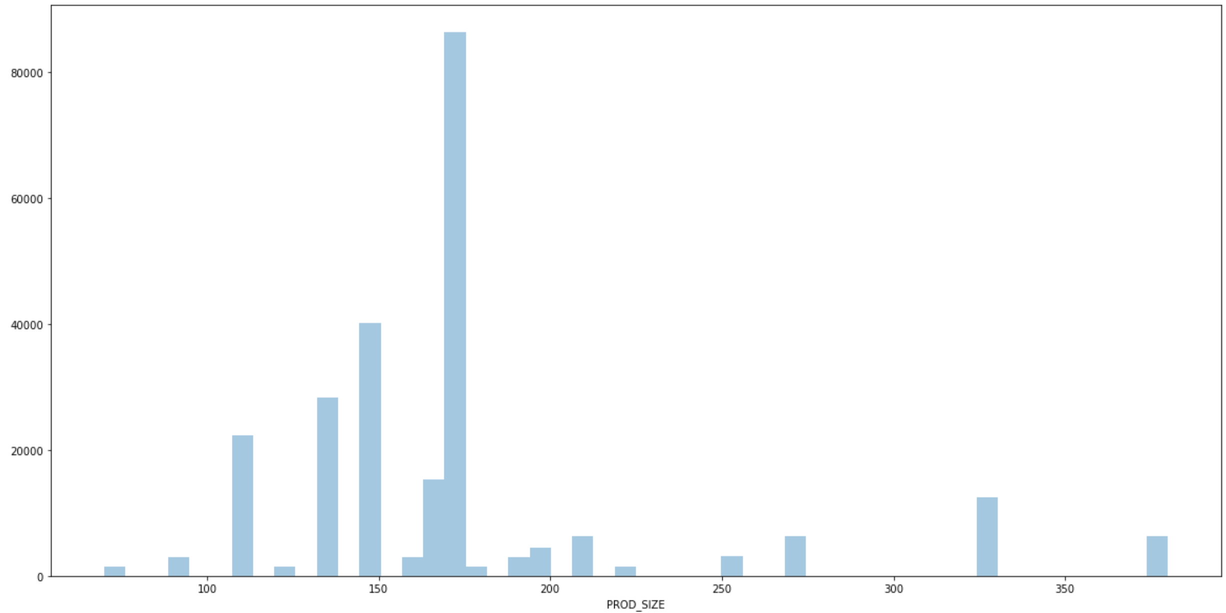
```
transaction_df.head(2)
```

Out[48]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_SIZE	PROD_
0	2018-10-17	1	1000	1	5	natural chip compny seasalt	175	
1	2019-05-14	1	1307	348	66	ccs nacho cheese	175	

In [49]:
```
plt.figure(figsize=(20,10))
sns.distplot(transaction_df['PROD_SIZE'],kde=False)
```

Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x24f414520c8>



Filter out the brands of the chips.

Brand names are just the first words in PROD_NAME

In [50]:

```
transaction_df.head(2)
```

Out[50]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_SIZE	PROD_
0	2018-10-17	1	1000	1	5	natural chip compny seasalt	175	
1	2019-05-14	1	1307	348	66	ccs nacho cheese	175	

In [57]:

```
# forgot to do this earlier when we dropped the rows
transaction_df.reset_index(inplace=True)
```

In [58]:

```
brand_names = []
for i in range(transaction_df.shape[0]):
    brand_names.append(transaction_df['PROD_NAME'][i].split()[0])

```

In [60]:

```
# get the index number of the column:"PROD_NAME"
prod_name_column_loc = transaction_df.columns.get_loc("PROD_NAME")
print(prod_name_column_loc)

transaction_df.insert(prod_name_column_loc + 1, "BRAND_NAME", brand_names)
```

6

In [62]:

```
# check if changes are made
transaction_df.head()
```

Out[62]:

	index	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	BRAND_NAI
0	0	2018-10-17	1	1000	1	5	natural chip compny seasalt	natu
1	1	2019-05-14	1	1307	348	66	ccs nacho cheese	
2	2	2019-05-20	1	1343	383	61	smiths crinkle cut chips chicken	smi
3	3	2018-08-17	2	2373	974	69	smiths chip thinly s/cream&onion	smi
4	4	2018-08-18	2	2426	1038	108	kettle tortilla chpshny&jlpno chili	ke

In [69]:
```
transaction_df['BRAND_NAME'].unique()
```

Out[69]:
```
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)
```

Brand names have inconsistent naming conventions...

- ncc and natural
- smith and smiths
- grain, grnwves and grainwaves
- ww and woolworths
- red, rrd and red rock deli
- 'infzns' and 'infuzions'
- 'snbts' and 'sunbites'
- 'smiths' and 'smith'
- 'doritos' and 'dorito'

In [70]:
```
# rename brand names for consistency
transaction_df['BRAND_NAME'].replace("ncc", "natural", inplace=True)
transaction_df['BRAND_NAME'].replace("smith", "smiths", inplace=True)
transaction_df['BRAND_NAME'].replace(['grain','grnwves'], "grainwaves", inpl
transaction_df['BRAND_NAME'].replace("ww", "woolworths", inplace=True)
transaction_df['BRAND_NAME'].replace(["red",'rrd'], "red rock deli", inplace
transaction_df['BRAND_NAME'].replace("infzns", "infuzions", inplace=True)
transaction_df['BRAND_NAME'].replace("snbts", "sunbites", inplace=True)
transaction_df['BRAND_NAME'].replace('dorito', "doritos", inplace=True)
```

```
C:\Users\smartestpersonalive\Anaconda3\lib\site-packages\pandas\core\generic.p
y:6746: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  self._update_inplace(new_data)
```

In [71]:
```
# check if changes are made
transaction_df['BRAND_NAME'].unique()
```

Out[71]:
```
array(['natural', 'ccs', 'smiths', 'kettle', 'grainwaves', 'doritos',
       'twisties', 'woolworths', 'thins', 'burger', 'cheezels',
       'infuzions', 'red rock deli', 'pringles', 'tyrrells', 'cobs',
       'french', 'tostitos', 'cheetos', 'sunbites'], dtype=object)
```

Check which brand had the most sales and purchases:

In [72]:
```
1 transaction_df.head(2)
```

Out[72]:

	index	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	BRAND_NAM
0	0	2018-10-17	1	1000	1	5	natural chip compny seasalt	natu
1	1	2019-05-14	1	1307	348	66	ccs nacho cheese	c

In [76]:
```
1 brand_group = transaction_df.groupby(by='BRAND_NAME')
```

In [82]:
```
1 brand_group.sum().sort_values(by='TOT_SALES', ascending=False)['TOT_SALES']
```

Out[82]:
```
BRAND_NAME
kettle           387471.2
doritos          225099.3
smiths           216535.9
pringles         176730.5
infuzions         98743.6
thins             88852.5
red rock deli     87607.5
twisties          80828.4
tostitos          79239.6
cobs              70284.8
grainwaves        51491.2
tyrrells          51387.0
natural           42318.0
woolworths        41059.1
cheezels          39591.0
ccs               18078.9
cheetos           16884.5
sunbites           9676.4
french             7929.0
burger             6831.0
Name: TOT_SALES, dtype: float64
```

In [83]:
```
brand_group.count().sort_values(by='TOT_SALES',ascending=False)['TOT_SALES']
```

Out[83]:
```
BRAND_NAME
kettle           41166
smiths           30311
doritos          25163
pringles         25052
red rock deli    16321
infuzions        14185
thins            14075
woolworths       11836
cobs              9678
tostitos          9443
twisties          9420
grainwaves        7733
natural           7469
tyrrells          6428
cheezels          4583
ccs               4551
sunbites          3008
cheetos           2927
burger            1564
french            1418
Name: TOT_SALES, dtype: int64
```

Kettle have the highest purchases numbers and sales. Smiths have the third highest sales numbers but second highest in purchase counts so their profit margin could be higher than Doritos

In [ ]:

In [ ]:

In [ ]:

Examine Customer Data

Lifestage: Customer attribute that identifies whether a customer has a family or not and what point in life they are at e.g. are their children in pre-school/primary/secondary school.

Premium_customer: Customer segmentation used to differentiate shoppers by the price point of products they buy and the types of products they buy. It is used to identify whether customers may spend more for quality or brand or whether they will purchase the cheapest options.

In [84]:
```
# check for null values
behavior_df.isnull().sum()
```

Out[84]:
```
LYLTY_CARD_NBR      0
LIFESTAGE           0
PREMIUM_CUSTOMER    0
dtype: int64
```

In [89]:
```
# check min max values. format to remove scientific notation
round(behavior_df.describe(),3)
```

Out[89]:

	LYLTY_CARD_NBR
count	72637.000
mean	136185.932
std	89892.932
min	1000.000
25%	66202.000
50%	134040.000
75%	203375.000
max	2373711.000

In [85]:
```
behavior_df.head()
```

Out[85]:

	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

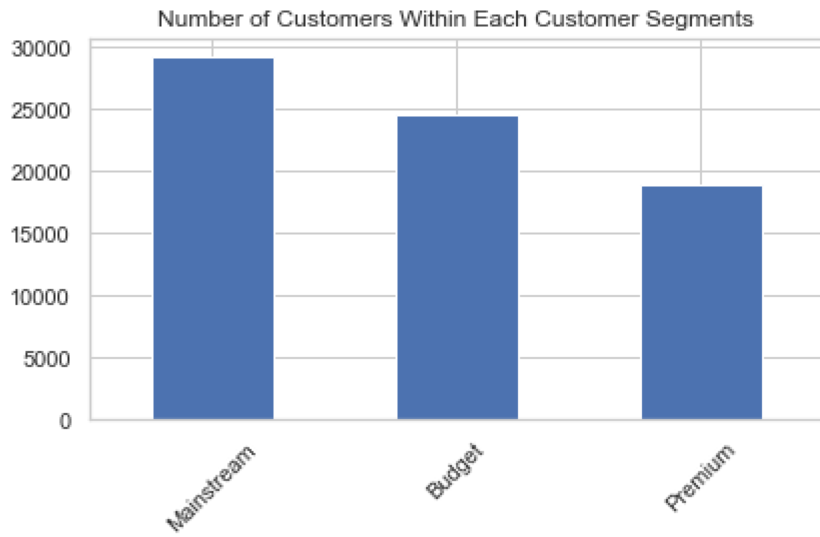
In [94]:
```
# check number of unique values for each feature
behavior_df.nunique()
```

Out[94]:
```
LYLTY_CARD_NBR      72637
LIFESTAGE               7
PREMIUM_CUSTOMER        3
dtype: int64
```

In [95]:
```
# check the different unique values for the lifestage feature
behavior_df['LIFESTAGE'].unique()

```

Out[95]:
```
array(['YOUNG SINGLES/COUPLES', 'YOUNG FAMILIES', 'OLDER SINGLES/COUPLES',
       'MIDAGE SINGLES/COUPLES', 'NEW FAMILIES', 'OLDER FAMILIES',
       'RETIREES'], dtype=object)
```

In [569]:
```
# check premium customer distribution
behavior_df['PREMIUM_CUSTOMER'].value_counts().plot(kind='bar')

# set customer behavior chart title
plt.title("Number of Customers Within Each Customer Segments")

# set x labels orientation
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig("static/analysis_pics/customerBahavior.png")
```



Merge transaction_df and behavior_df

In [102]:
```
transaction_df.shape
```

Out[102]: (246331, 11)

In [282]:
```
# merge two dataframes
complete_df = pd.merge(transaction_df, behavior_df, on="LYLTY_CARD_NBR", how
complete_df.head()
```

Out[282]:

	index	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	BRAND_NAM
0	0	2018-10-17	1	1000	1	5	natural chip compny seasalt	natu
1	1	2019-05-14	1	1307	348	66	ccs nacho cheese	c
2	202	2018-11-10	1	1307	346	96	ww original stacked chips	woolwort
3	203	2019-03-09	1	1307	347	54	ccs original	c
4	2	2019-05-20	1	1343	383	61	smiths crinkle cut chips chicken	smit

In [104]:
```
complete_df.shape
```

Out[104]:
```
(246331, 13)
```

In [105]:
```
# check for missing values
complete_df.isnull().sum()
```

Out[105]:
```
index               0
DATE                0
STORE_NBR           0
LYLTY_CARD_NBR      0
TXN_ID              0
PROD_NBR            0
PROD_NAME           0
BRAND_NAME          0
PROD_SIZE           0
PROD_QTY            0
TOT_SALES           0
LIFESTAGE           0
PREMIUM_CUSTOMER    0
dtype: int64
```

In [ ]:
```

```

Data Analysis

Analysis metrics to consider:

- who spends the most on chips (total sales), describing customers by lifestage and how premium their general purchasing behavior 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

In [106]:
```
complete_df.head()
```

Out[106]:

	index	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	BRAND_NAM
0	0	2018-10-17	1	1000	1	5	natural chip compny seasalt	natu
1	1	2019-05-14	1	1307	348	66	ccs nacho cheese	c
2	202	2018-11-10	1	1307	346	96	ww original stacked chips	woolwort
3	203	2019-03-09	1	1307	347	54	ccs original	c
4	2	2019-05-20	1	1343	383	61	smiths crinkle cut chips chicken	smit

Find out which customer segment spends the most on chips

In [468]:
```
"Find out which customer segment spends the most on chips".title()
```

Out[468]:
```
'Find Out Which Customer Segment Spends The Most On Chips'
```

In [114]:
```
# who spends the most on chips
sales_sorted = complete_df.groupby("LYLTY_CARD_NBR").sum().sort_values('TOT_
sales_sorted.head(6)
```

Out[114]:
```
LYLTY_CARD_NBR
230078    138.6
58361     124.8
63197     122.6
162039    121.6
179228    120.8
199157    118.8
Name: TOT_SALES, dtype: float64
```

In [115]:
```
complete_df.nunique()
```

Out[115]:
```
index                 246331
DATE                     364
STORE_NBR                271
LYLTY_CARD_NBR         71253
TXN_ID                244848
PROD_NBR                 105
PROD_NAME                105
BRAND_NAME                20
PROD_SIZE                 20
PROD_QTY                   5
TOT_SALES                 84
LIFESTAGE                  7
PREMIUM_CUSTOMER           3
dtype: int64
```

In [119]:
```
# group by life stage and purchasing behavior
premium_lifestage_group_df = pd.DataFrame(complete_df.groupby(['PREMIUM_CUST
```

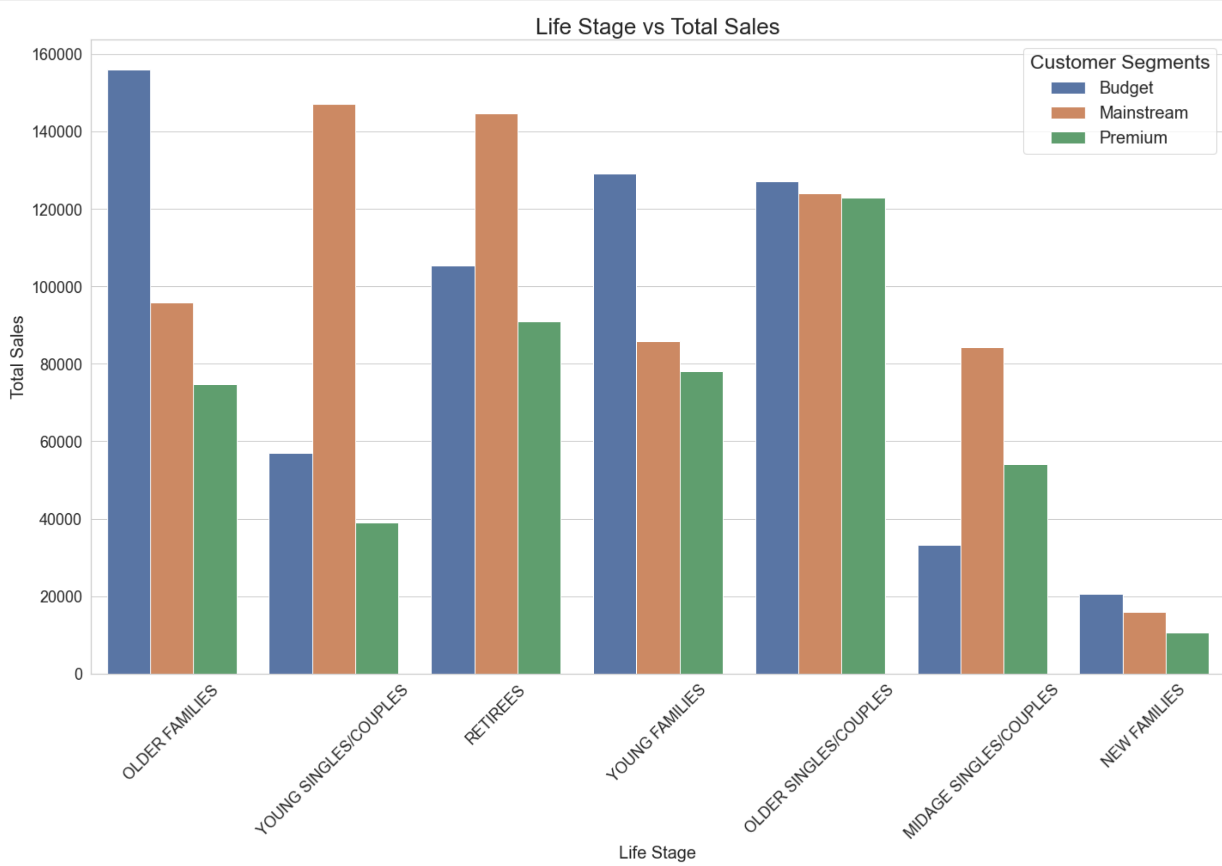
In [132]:
```
lifestyle_df = premium_lifestage_group_df.sort_values("TOT_SALES", ascending
```

In [133]:
```
lifestyle_df.head()
```

Out[133]:

	PREMIUM_CUSTOMER	LIFESTAGE	TOT_SALES
0	Budget	OLDER FAMILIES	155980.95
1	Mainstream	YOUNG SINGLES/COUPLES	147001.90
2	Mainstream	RETIREES	144686.45
3	Budget	YOUNG FAMILIES	129129.25
4	Budget	OLDER SINGLES/COUPLES	127114.00

In [573]:
```python
sns.set(style="whitegrid")
# set figure size
plt.figure(figsize=(20,14))

# create bar chart
sns.barplot(x = "LIFESTAGE",
           y = "TOT_SALES",
           hue= "PREMIUM_CUSTOMER",
           data = lifestyle_df)
# set x labels orientation
plt.xticks(rotation=45)

# set fontsize
plt.xlabel("Life Stage", fontsize= 20)
plt.ylabel("Total Sales", fontsize=20)
plt.title("Life Stage vs Total Sales", fontsize=26)
plt.tick_params(labelsize=18)

plt.legend(title = "Customer Segments",
          loc='best',
          fontsize=20,
          title_fontsize=23)

plt.tight_layout()

# save file
plt.savefig("static/analysis_pics/Lifestage_vs_totalSales.png")
```



Sales are mainly coming from Budget for OLDER FAMILIES, Mainstream for YOUNG SINGLES/COUPLES and Mainstream for RETIREES

Now check to see if the higher sales are due to there being more customers who buy chips

check how many customers there are in each customer segment

In [549]:
```
customercount_group_df = complete_df.groupby(['PREMIUM_CUSTOMER', "LIFESTAGE
```

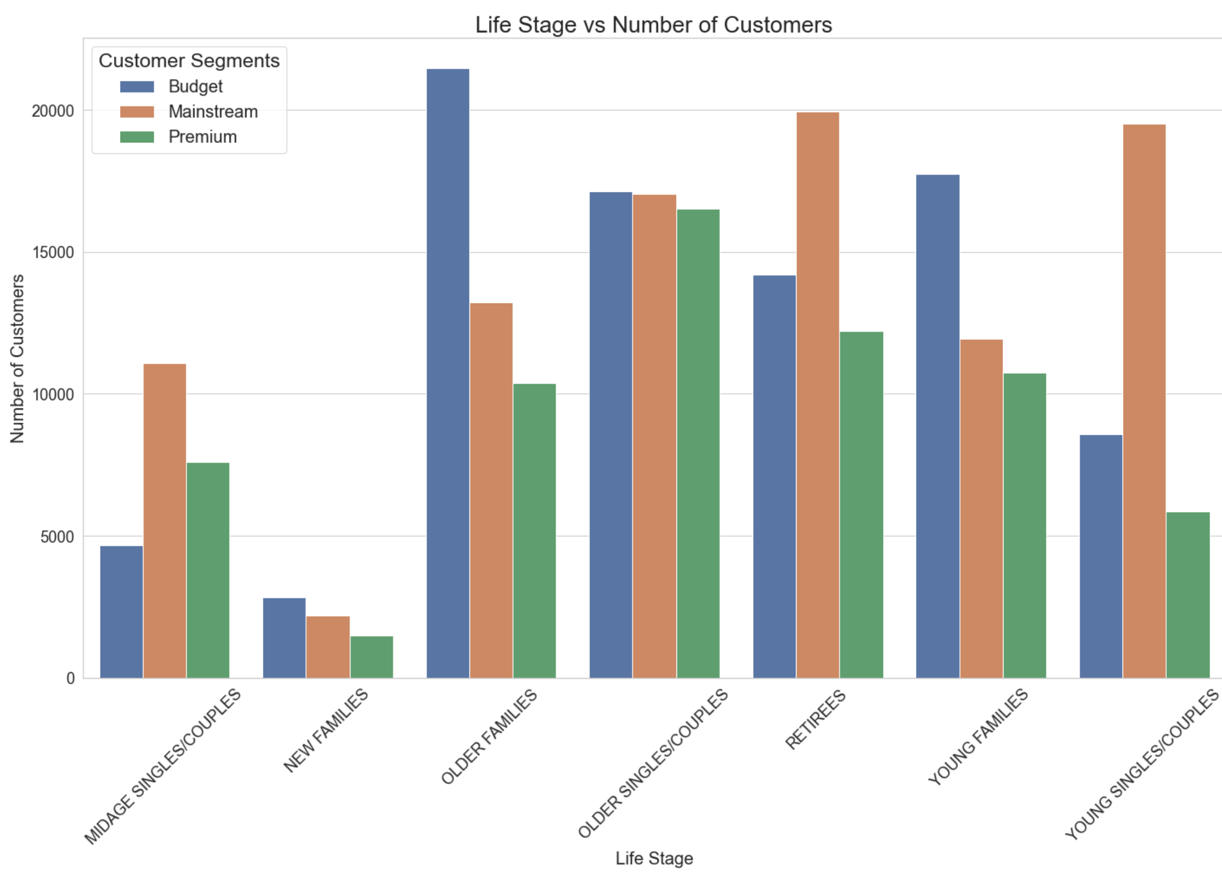
In [550]:

```
1 customercount_group_df
```

Out[550]:

	PREMIUM_CUSTOMER	LIFESTAGE	LYLTY_CARD_NBR
0	Budget	MIDAGE SINGLES/COUPLES	1474
1	Budget	NEW FAMILIES	1087
2	Budget	OLDER FAMILIES	4607
3	Budget	OLDER SINGLES/COUPLES	4847
4	Budget	RETIREES	4383
5	Budget	YOUNG FAMILIES	3952
6	Budget	YOUNG SINGLES/COUPLES	3645
7	Mainstream	MIDAGE SINGLES/COUPLES	3296
8	Mainstream	NEW FAMILIES	830
9	Mainstream	OLDER FAMILIES	2788
10	Mainstream	OLDER SINGLES/COUPLES	4853
11	Mainstream	RETIREES	6358
12	Mainstream	YOUNG FAMILIES	2684
13	Mainstream	YOUNG SINGLES/COUPLES	7907
14	Premium	MIDAGE SINGLES/COUPLES	2369
15	Premium	NEW FAMILIES	575
16	Premium	OLDER FAMILIES	2230
17	Premium	OLDER SINGLES/COUPLES	4681
18	Premium	RETIREES	3811
19	Premium	YOUNG FAMILIES	2397
20	Premium	YOUNG SINGLES/COUPLES	2479

In [570]:
```
sns.set(style="whitegrid")
# set figure size
plt.figure(figsize=(20,14))

# create bar chart
sns.barplot(x = "LIFESTAGE",
           y = "LYLTY_CARD_NBR",
           hue= "PREMIUM_CUSTOMER",
           data = customercount_group_df)
# set x labels orientation
plt.xticks(rotation=45)

# set fontsize
plt.xlabel("Life Stage", fontsize= 20)
plt.ylabel("Number of Customers", fontsize=20)
plt.title("Life Stage vs Number of Customers", fontsize=26)
plt.tick_params(labelsize=18)

plt.legend(title = "Customer Segments",
          loc='best',
          fontsize=20,
          title_fontsize=23)

plt.tight_layout()

# save file
plt.savefig("static/analysis_pics/Lifestage_vs_numberOfCustomers.png")
```



There are more Mainstream customers in "Young singles/couples" and "retirees" who buy chips. This contributes to there being more sales to these customer segments but this is not a major driver for the Budget customer segment for the "Older Families" segment.

Higher sales may be driven by more units of chips being bought per customer

calculate the number of units of chips bought per customer

In [552]:
```
# find how many customers are in each segment
num_customer_per_segment = complete_df.groupby(['PREMIUM_CUSTOMER', "LIFESTA
```

In [553]:
```
customercount_group_df['num_customers'] = num_customer_per_segment
```

In [554]:
```
customercount_group_df.head(3)
```

Out[554]:

	PREMIUM_CUSTOMER	LIFESTAGE	LYLTY_CARD_NBR	num_customers
0	Budget	MIDAGE SINGLES/COUPLES	1474	4684
1	Budget	NEW FAMILIES	1087	2822
2	Budget	OLDER FAMILIES	4607	21472

In [555]:
```
# here to ifx my mistakes lol
customercount_group_df.rename(columns={"LYLTY_CARD_NBR":"num_customers",
                                       "num_customers": "LYLTY_CARD_NBR"},
                              inplace=True)
```

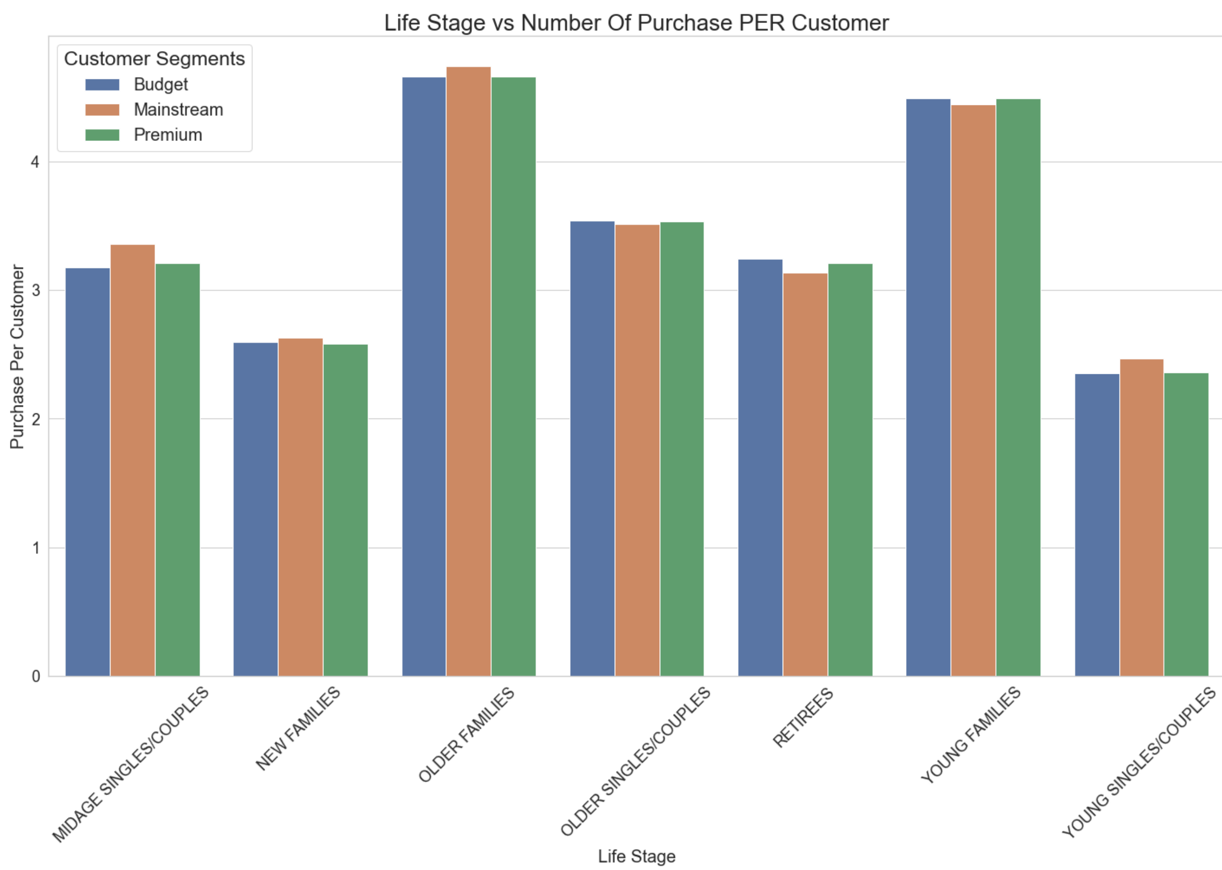
In [556]:
```
# create a variable for total number of chips bought
chips_purchased = customercount_group_df['LYLTY_CARD_NBR']

# create a varialbe for total number of customers for each segment
customers_per_segment = customercount_group_df['num_customers']

# calculate number of chips bought per customer
customercount_group_df['num_bought_percustomer'] = chips_purchased / custome
```

In [557]:
```
customercount_group_df.head(3)
```

Out[557]:

	PREMIUM_CUSTOMER	LIFESTAGE	num_customers	LYLTY_CARD_NBR	num_bought_
0	Budget	MIDAGE SINGLES/COUPLES	1474	4684	
1	Budget	NEW FAMILIES	1087	2822	
2	Budget	OLDER FAMILIES	4607	21472	

In [571]:
```python
sns.set(style="whitegrid")
# set figure size
plt.figure(figsize=(20,14))

# create bar chart
sns.barplot(x = "LIFESTAGE",
           y = "num_bought_percustomer",
           hue= "PREMIUM_CUSTOMER",
           data = customercount_group_df)
# set x labels orientation
plt.xticks(rotation=45)

# set fontsize
plt.xlabel("Life Stage", fontsize= 20)
plt.ylabel("Purchase Per Customer", fontsize=20)
plt.title("Life Stage vs Number Of Purchase PER Customer", fontsize=26)
plt.tick_params(labelsize=18)

plt.legend(title = "Customer Segments",
          loc='best',
          fontsize=20,
          title_fontsize=23)

plt.tight_layout()

# save file
plt.savefig("static/analysis_pics/Lifestage_vs_purchasePercustomer.png")
```



older families and young families in general buy more chips per customer

Find the average price per unit chips bought for each customer

In [559]:
```
# find total sales for each customer segment
total_sales = complete_df.groupby(['PREMIUM_CUSTOMER','LIFESTAGE']).sum()['T
```

In [560]:
```
# find total quantiyty purchased for each segment
total_qty = complete_df.groupby(['PREMIUM_CUSTOMER','LIFESTAGE']).sum()['PRO
```

In [561]:
```
total_qty
```

Out[561]:
```
array([ 8851,  5231, 41666, 32732, 26807, 34355, 15467, 21110,  4050,
       25681, 32471, 37571, 23105, 36101, 14352,  2760, 20142, 31564,
       23182, 20798, 10562], dtype=int64)
```

In [562]:
```
customercount_group_df['total_sales'] = total_sales
customercount_group_df['total_qty'] = total_qty
```

In [563]:
```
if 'sum_prices' in customercount_group_df.columns.values:
    customercount_group_df.drop(columns=['sum_prices'], inplace=True)
if 'average_prices' in customercount_group_df.columns.values:
    customercount_group_df.drop(columns=['average_prices'], inplace=True)
```

In [564]:
```

customercount_group_df.head()
```

Out[564]:

	PREMIUM_CUSTOMER	LIFESTAGE	num_customers	LYLTY_CARD_NBR	num_bought_per
0	Budget	MIDAGE SINGLES/COUPLES	1474	4684	
1	Budget	NEW FAMILIES	1087	2822	
2	Budget	OLDER FAMILIES	4607	21472	
3	Budget	OLDER SINGLES/COUPLES	4847	17137	
4	Budget	RETIREES	4383	14197	

In [565]:
```
some_sales = customercount_group_df['total_sales']
some_qty = customercount_group_df['total_qty']
customercount_group_df['price_per_unit'] = some_sales / some_qty
```

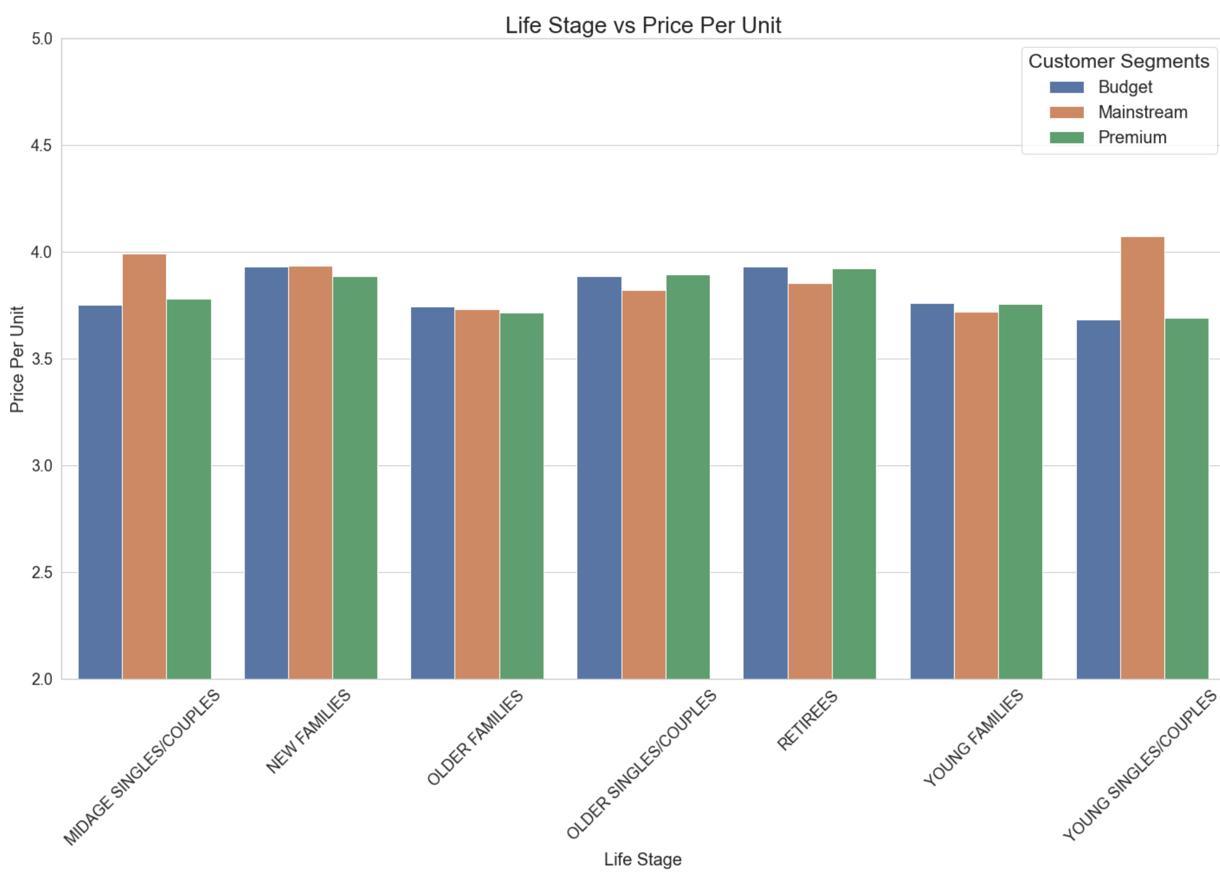
In [566]:
```
customercount_group_df.head()
```

Out[566]:

	PREMIUM_CUSTOMER	LIFESTAGE	num_customers	LYLTY_CARD_NBR	num_bought_per
0	Budget	MIDAGE SINGLES/COUPLES	1474	4684	
1	Budget	NEW FAMILIES	1087	2822	
2	Budget	OLDER FAMILIES	4607	21472	
3	Budget	OLDER SINGLES/COUPLES	4847	17137	
4	Budget	RETIREES	4383	14197	

In [572]:

```
sns.set(style="whitegrid")
# set figure size
plt.figure(figsize=(20,14))

# create bar chart
sns.barplot(x = "LIFESTAGE",
            y = "price_per_unit",
            hue= "PREMIUM_CUSTOMER",
            data = customercount_group_df)
# set x labels orientation
plt.xticks(rotation=45)

# set fontsize
plt.xlabel("Life Stage", fontsize= 20)
plt.ylabel("price per unit".title(), fontsize=20)
plt.title("Life Stage vs Price Per Unit", fontsize=26)
plt.tick_params(labelsize=18)

plt.legend(title = "Customer Segments",
           loc='best',
           fontsize=20,
           title_fontsize=23)
plt.ylim(2,5)

plt.tight_layout()

# save file
plt.savefig("static/analysis_pics/Lifestage_vs_pricePerunit.png")
```



- Young and midage singles and couples: mainstream customers are more willing to pay more per unit compared to their budget and premium counterparts.

The difference in average price per unit isn't large so we can check if this difference is statisically different

t-test between mainstream vs premium and budget for mid age and young singles/couples

In [568]:
```
from scipy.stats import ttest_ind
```

In [223]:
```
complete_df.head()
```

Out[223]:

	index	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	BRAND_NAM
0	0	2018-10-17	1	1000	1	5	natural chip compny seasalt	natu
1	1	2019-05-14	1	1307	348	66	ccs nacho cheese	c
2	202	2018-11-10	1	1307	346	96	ww original stacked chips	woolwort
3	203	2019-03-09	1	1307	347	54	ccs original	c
4	2	2019-05-20	1	1343	383	61	smiths crinkle cut chips chicken	smit

In [224]:
```
# create a duplicate of complete_df for this specific analysis
for_ttest_df = complete_df
```

In [225]:
```
for_ttest_df.head()
```

Out[225]:

E_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	BRAND_NAME	PROD_SIZE	PROD_(
1	1000	1	5	natural chip compny seasalt	natural	175	
1	1307	348	66	ccs nacho cheese	ccs	175	
1	1307	346	96	ww original stacked chips	woolworths	160	
1	1307	347	54	ccs original	ccs	175	
1	1343	383	61	smiths crinkle cut chips chicken	smiths	170	

In [226]:
```
# calculate price per unit and create a column named "priced_per_unit"
for_ttest_df['price_per_unit'] = for_ttest_df['TOT_SALES'] / for_ttest_df['P
```

In [227]:
```
for_ttest_df.head()
```

Out[227]:

CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	BRAND_NAME	PROD_SIZE	PROD_QTY	TOT_SALE
1000	1	5	natural chip compny seasalt	natural	175	2	6
1307	348	66	ccs nacho cheese	ccs	175	3	6
1307	346	96	ww original stacked chips	woolworths	160	2	3
1307	347	54	ccs original	ccs	175	1	2
1343	383	61	smiths crinkle cut chips chicken	smiths	170	2	2

In [228]:
```
for_ttest_df['PREMIUM_CUSTOMER'].unique()
```

Out[228]:
```
array(['Premium', 'Budget', 'Mainstream'], dtype=object)
```

In [229]:
```
for_ttest_df['LIFESTAGE'].unique()
```

Out[229]:
```
array(['YOUNG SINGLES/COUPLES', 'MIDAGE SINGLES/COUPLES', 'NEW FAMILIES',
       'OLDER FAMILIES', 'OLDER SINGLES/COUPLES', 'RETIREES',
       'YOUNG FAMILIES'], dtype=object)
```

In [234]:
```
# select the price per unit for mainstream customers
mainstream_ppu = for_ttest_df.loc[(for_ttest_df['PREMIUM_CUSTOMER'] == "Main
                & (for_ttest_df['LIFESTAGE'] == "YOUNG SINGLES/COUPLES")\
                | (for_ttest_df['LIFESTAGE'] == "MIDAGE SINGLES/COUPLES")\
                , 'price_per_unit']

# select the price per unit for budget and premium customers
budget_premium_ppu = for_ttest_df.loc[(for_ttest_df['PREMIUM_CUSTOMER'] != "
                & (for_ttest_df['LIFESTAGE'] == "YOUNG SINGLES/COUPLES")\
                | (for_ttest_df['LIFESTAGE'] == "MIDAGE SINGLES/COUPLES")\
                , 'price_per_unit']
```

In [235]:
```
1 mainstream_ppu
```

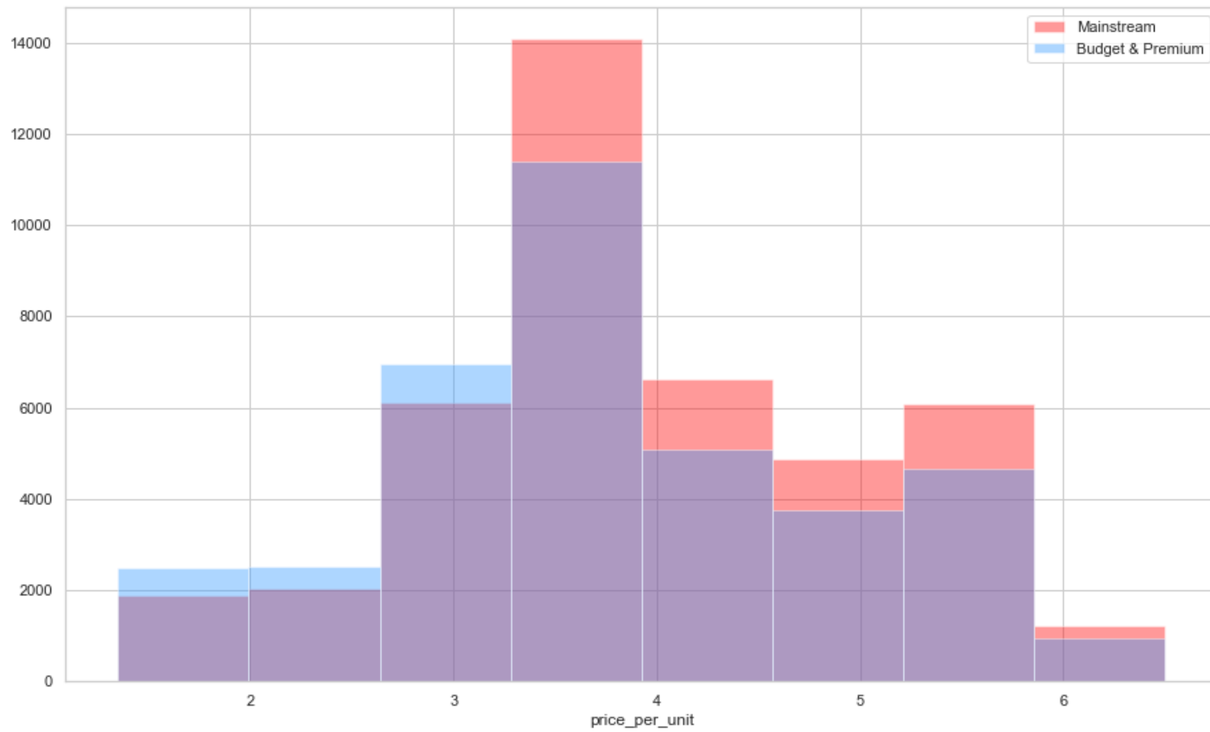
Out[235]:
```
1         2.10
2         1.90
3         2.10
4         1.45
5         3.00
          ...
240478    3.80
240479    4.60
240480    3.70
240481    3.70
240482    4.20
Name: price_per_unit, Length: 42874, dtype: float64
```

In [236]:
```
1 budget_premium_ppu
```

Out[236]:
```
0         3.00
1         2.10
2         1.90
3         2.10
4         1.45
          ...
246326    5.40
246327    4.40
246328    4.40
246329    3.90
246330    4.40
Name: price_per_unit, Length: 37774, dtype: float64
```

In [267]:
```
plt.figure(figsize=(15,9))
ax = sns.distplot(mainstream_ppu, kde=False, color="#FF0000", bins=8, label
ax1 = sns.distplot(budget_premium_ppu, kde=False, color = "#3399FF", bins=8,
plt.legend()
```

Out[267]: <matplotlib.legend.Legend at 0x24f40f1d448>



In [243]:
```
ttest_ind(mainstream_ppu, budget_premium_ppu)
```

Out[243]: Ttest_indResult(statistic=22.478937900290347, pvalue=1.4720083206181396e-111)

The larger the t score, the more difference there is between groups. The smaller the t score, the more similarity there is between groups.

A t score of 3 means that the groups are three times as different from each other as they are within each other.

p-value is the probability that the results from the sample data occurred by chance. We got a very low p-value which indicates that the data did not occur by chance. In this case, there is only a 1.47e-111% probability that the results from the data happened by chance.

The small p-value indicates that young and mid-age singles and couples are significantly higher than that of budget or premium, young and midage singles and couples

Further analyze each customer segments for insights

find out if a specific segment tend to buy a particular brand of chips

try to target customer segments that contribute the most to sales to retain them or further increase sales

In [ ]:
```
1 
```

In [268]:
```
1 complete_df.head(3)
```

Out[268]:

LTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	BRAND_NAME	PROD_SIZE	PROD_QTY	TOT_S
1000	1	5	natural chip compny seasalt	natural	175	2	
1307	348	66	ccs nacho cheese	ccs	175	3	
1307	346	96	ww original stacked chips	woolworths	160	2	

In [295]:
```
1 purchase_trend_df = pd.DataFrame(complete_df.groupby(['PREMIUM_CUSTOMER','BR
```

In [296]:
```
1 purchase_trend_df
```

Out[296]:

	PREMIUM_CUSTOMER	BRAND_NAME	LIFESTAGE	LYLTY_CARD_NBR
0	Budget	burger	MIDAGE SINGLES/COUPLES	43
1	Budget	burger	NEW FAMILIES	18
2	Budget	burger	OLDER FAMILIES	159
3	Budget	burger	OLDER SINGLES/COUPLES	110
4	Budget	burger	RETIREES	66
...	...	...	...	...
415	Premium	woolworths	OLDER FAMILIES	636
416	Premium	woolworths	OLDER SINGLES/COUPLES	701
417	Premium	woolworths	RETIREES	470
418	Premium	woolworths	YOUNG FAMILIES	565
419	Premium	woolworths	YOUNG SINGLES/COUPLES	393

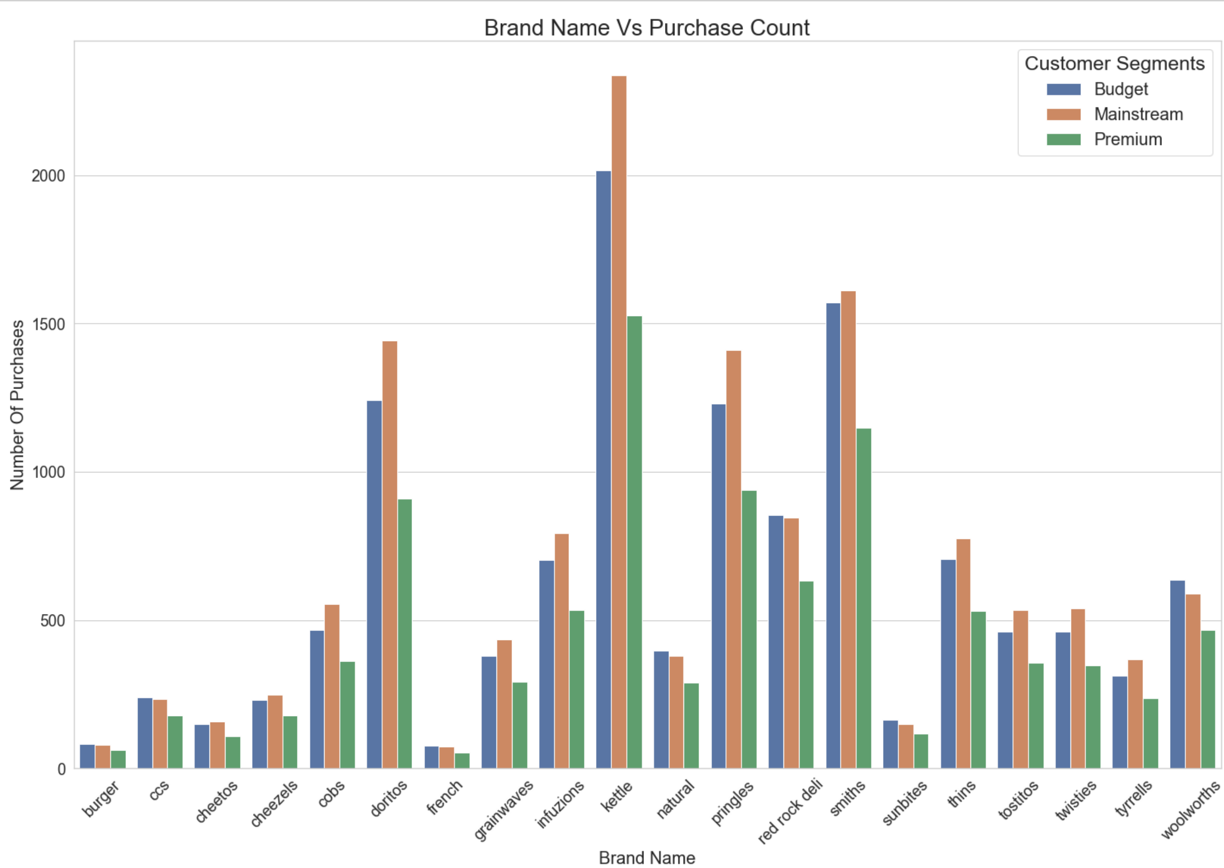
420 rows × 4 columns

In [297]:
```
1 purchase_trend_df.nunique()
```

Out[297]:
```
PREMIUM_CUSTOMER      3
BRAND_NAME           20
LIFESTAGE             7
LYLTY_CARD_NBR      353
dtype: int64
```

In [585]:

```
sns.set(style="whitegrid")
# set figure size
plt.figure(figsize=(20,14))

# create bar chart
sns.barplot(x = "BRAND_NAME",
           y = "LYLTY_CARD_NBR",
           hue= "PREMIUM_CUSTOMER",
           data = purchase_trend_df,
           ci=None)
# set x labels orientation
plt.xticks(rotation=45)

# set fontsize
plt.xlabel("Brand Name", fontsize= 20)
plt.ylabel("number of purchases".title(), fontsize=20)
plt.title("brand name vs purchase count".title(), fontsize=26)
plt.tick_params(labelsize=18)

plt.legend(title = "Customer Segments",
          loc='best',
          fontsize=20,
          title_fontsize=23)


plt.tight_layout()

# save file
# plt.savefig("static/analysis_pics/BrandName_vs_PurchaseCount.png")
```



find out how many packs of chips were purchased from each customer segment

In [579]:
```
packs_purchase_trend_df = complete_df.groupby(['PREMIUM_CUSTOMER','BRAND_NAM
```

In [581]:
```
packs_purchase_trend_df.head()
```

Out[581]:

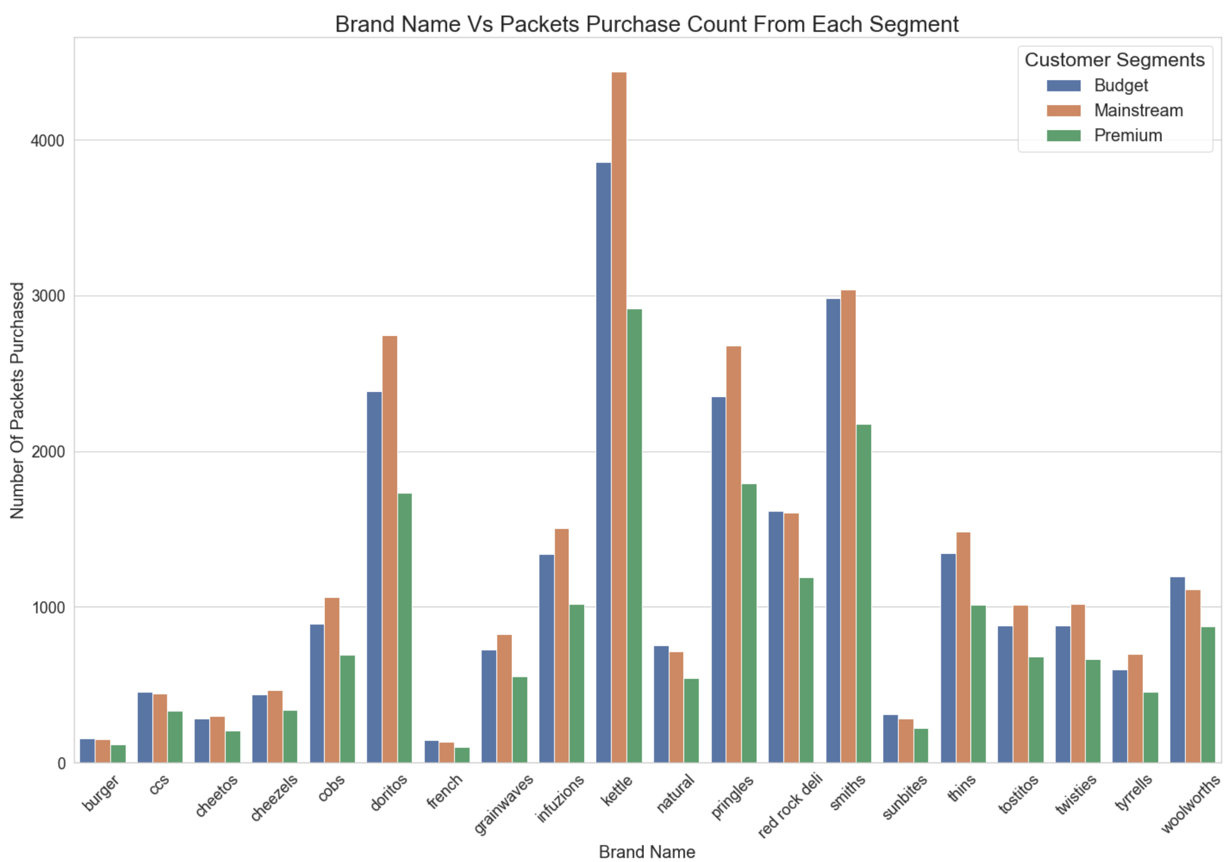
	PREMIUM_CUSTOMER	BRAND_NAME	LIFESTAGE	PROD_QTY
0	Budget	burger	MIDAGE SINGLES/COUPLES	84
1	Budget	burger	NEW FAMILIES	30
2	Budget	burger	OLDER FAMILIES	301
3	Budget	burger	OLDER SINGLES/COUPLES	209
4	Budget	burger	RETIREES	129

In [580]:
```
packs_purchase_trend_df.nunique()
```

Out[580]:
```
PREMIUM_CUSTOMER      3
BRAND_NAME           20
LIFESTAGE             7
PROD_QTY            383
dtype: int64
```

In [586]:

```python
sns.set(style="whitegrid")
# set figure size
plt.figure(figsize=(20,14))

# create bar chart
sns.barplot(x = "BRAND_NAME",
            y = "PROD_QTY",
            hue= "PREMIUM_CUSTOMER",
            data = packs_purchase_trend_df,
            ci=None)
# set x labels orientation
plt.xticks(rotation=45)

# set fontsize
plt.xlabel("Brand Name", fontsize= 20)
plt.ylabel("number of packets purchased".title(), fontsize=20)
plt.title("brand name vs packets purchase count from each segment".title(),
plt.tick_params(labelsize=18)

plt.legend(title = "Customer Segments",
          loc='best',
          fontsize=20,
          title_fontsize=23)


plt.tight_layout()
```



In [667]:
```
# for just mainstream customers:
mainstream_df = complete_df.loc[(complete_df['LIFESTAGE']=="YOUNG SINGLES/CO
```

In [669]:
```
mainstream_qty_group_df = pd.DataFrame(mainstream_df.groupby(['PREMIUM_CUSTO
```

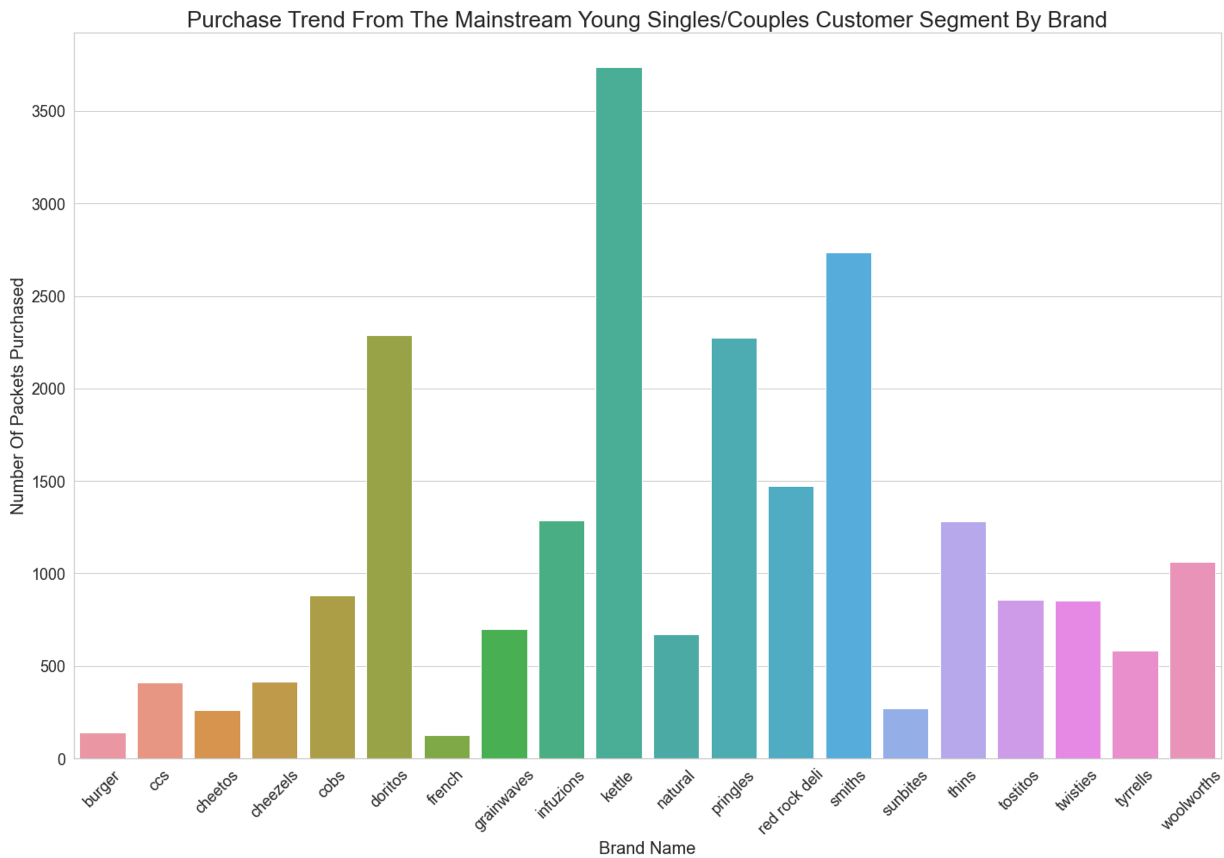
In [670]:
```
mainstream_qty_group_df.head()
```

Out[670]:

	PREMIUM_CUSTOMER	BRAND_NAME	LIFESTAGE	PROD_QTY
0	Mainstream	burger	YOUNG SINGLES/COUPLES	106
1	Mainstream	ccs	YOUNG SINGLES/COUPLES	405
2	Mainstream	cheetos	YOUNG SINGLES/COUPLES	291
3	Mainstream	cheezels	YOUNG SINGLES/COUPLES	651
4	Mainstream	cobs	YOUNG SINGLES/COUPLES	1617

In [728]:

```
sns.set(style="whitegrid")
# set figure size
plt.figure(figsize=(20,14))

# create bar chart
sns.barplot(x = "BRAND_NAME",
            y = "PROD_QTY",
#             hue= "PREMIUM_CUSTOMER",
            data = packs_purchase_trend_df,
            ci=None)
# set x labels orientation
plt.xticks(rotation=45)

# set fontsize
plt.xlabel("Brand Name", fontsize= 20)
plt.ylabel("number of packets purchased".title(), fontsize=20)
plt.title("purchase trend from the mainstream young singles/couples customer
plt.tick_params(labelsize=18)

# plt.legend(title = "Customer Segments",
#            loc='best',
#            fontsize=20,
#            title_fontsize=23)


plt.tight_layout()
plt.savefig("static/analysis_pics/purchase_trend_by_brand.png")
```



In [639]:

```

```

In [ ]:
```
1
```

In [ ]:
```
1
```

Affinity Analysis

This analysis will focus on young singles/couples because they are one of the top contributors to total sales

In [597]:
```
1 complete_df.head()
```

Out[597]:

RE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	BRAND_NAME	PROD_SIZE	PROD
1	1000	1	5	natural chip compny seasalt	natural	175	
1	1307	348	66	ccs nacho cheese	ccs	175	
1	1307	346	96	ww original stacked chips	woolworths	160	
1	1307	347	54	ccs original	ccs	175	
1	1343	383	61	smiths crinkle cut chips chicken	smiths	170	

In [673]:
```
1 mainstream_df.head()
```

Out[673]:

	index	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	BRAN
220968	237298	2018-08-16	1	1020	26	19	smiths crinkle cut snag&sauce	
220969	238065	2018-10-02	1	1020	27	7	smiths crinkle original	
220970	238066	2019-05-02	1	1020	28	84	grnwves plus btroot & chilli jam	g
220971	237299	2018-08-17	1	1163	188	46	kettle original	
220972	238079	2019-02-07	1	1163	189	12	natural chip co tmato hrb&spce	

In [674]:
```
1 mainstream_df['LYLTY_CARD_NBR'].nunique()
```

Out[674]: 7907

In [675]:
```python

market_basket_df = mainstream_df.groupby(['LYLTY_CARD_NBR', 'BRAND_NAME'])['
                    sum()\
                    .unstack()\
                    .reset_index().\
                    fillna(0).\
                    set_index("LYLTY_CARD_NBR")
```

In [676]:
```python
market_basket_df
```

Out[676]:

BRAND_NAME	burger	ccs	cheetos	cheezels	cobs	doritos	french	grainwaves	infuzions
LYLTY_CARD_NBR									
1002	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
1010	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.
1018	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.
1020	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.
1060	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	...	..
272391	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
2330041	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.

basket data: we have name of the products as columns. if the value is 0, then that means that product was not present for the customer (LYLTY_CARD_NBR). if a number under burger is 1 with the LYLTY_CARD_NBR of 1000,the customer would have purchased 1 pack of chips from the burger brand.

In [677]:
```python
from mlxtend.frequent_patterns import apriori, association_rules
```

In [678]:
```python
# make the dataframe values either 0 or 1 because the algorithmn expects 0/1
def units_purchased(x):
    if x <= 0:
        return 0
    if x >0:
        return 1
```

In [679]:
```python
# use the applymap() to run through every element of the dataframe
basket_sets = market_basket_df.applymap(units_purchased)
```

In [680]:

```
# check to see if changes are made
basket_sets.head()
```

Out[680]:

BRAND_NAME	burger	ccs	cheetos	cheezels	cobs	doritos	french	grainwaves	infuzions
LYLTY_CARD_NBR									
1002	0	0	0	0	0	0	0	0	0
1010	0	0	0	0	0	1	0	0	0
1018	0	0	0	0	0	0	0	0	1
1020	0	0	0	0	0	0	0	1	0
1060	0	0	0	0	0	1	0	0	0

In [689]:

```
# train the model
# frequent itemsets. using 0.07 support. use column names as item names
frequent_itemsets = apriori(basket_sets, min_support=0.05, use_colnames=True
```

In [737]:

```
frequent_itemsets_sorted_df = frequent_itemsets.sort_values(by='support',asc
frequent_itemsets_sorted_df
```

Out[737]:

	support	itemsets
4	0.386999	(kettle)
1	0.260402	(doritos)
5	0.255976	(pringles)
7	0.202352	(smiths)
3	0.143164	(infuzions)
8	0.136208	(thins)
10	0.107247	(twisties)
9	0.105856	(tostitos)
0	0.103959	(cobs)
6	0.094094	(red rock deli)
16	0.091564	(kettle, pringles)
13	0.090047	(kettle, doritos)
2	0.078917	(grainwaves)
11	0.076262	(tyrrells)
17	0.075629	(kettle, smiths)
14	0.062603	(doritos, pringles)
12	0.053876	(woolworths)
18	0.051347	(kettle, thins)
15	0.050967	(doritos, smiths)

In [697]:

```
# generate rules
rules = association_rules(frequent_itemsets, metric='lift')
```

In [698]:
```
rules
```

Out[698]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage
0	(kettle)	(doritos)	0.386999	0.260402	0.090047	0.232680	0.893540	-0.010729
1	(doritos)	(kettle)	0.260402	0.386999	0.090047	0.345799	0.893540	-0.010729
2	(doritos)	(pringles)	0.260402	0.255976	0.062603	0.240408	0.939183	-0.004054
3	(pringles)	(doritos)	0.255976	0.260402	0.062603	0.244565	0.939183	-0.004054
4	(doritos)	(smiths)	0.260402	0.202352	0.050967	0.195726	0.967254	-0.001725
5	(smiths)	(doritos)	0.202352	0.260402	0.050967	0.251875	0.967254	-0.001725
6	(kettle)	(pringles)	0.386999	0.255976	0.091564	0.236601	0.924312	-0.007498
7	(pringles)	(kettle)	0.255976	0.386999	0.091564	0.357708	0.924312	-0.007498
8	(kettle)	(smiths)	0.386999	0.202352	0.075629	0.195425	0.965765	-0.002681
9	(smiths)	(kettle)	0.202352	0.386999	0.075629	0.373750	0.965765	-0.002681
10	(kettle)	(thins)	0.386999	0.136208	0.051347	0.132680	0.974093	-0.001366
11	(thins)	(kettle)	0.136208	0.386999	0.051347	0.376973	0.974093	-0.001366

In [731]:
```
# sort from highest to lowest by confidence level
rules_sorted_df = rules.sort_values(by='confidence', ascending=False)
rules_sorted_df
```

Out[731]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage
11	(thins)	(kettle)	0.136208	0.386999	0.051347	0.376973	0.974093	-0.001366
9	(smiths)	(kettle)	0.202352	0.386999	0.075629	0.373750	0.965765	-0.002681
7	(pringles)	(kettle)	0.255976	0.386999	0.091564	0.357708	0.924312	-0.007498
1	(doritos)	(kettle)	0.260402	0.386999	0.090047	0.345799	0.893540	-0.010729
5	(smiths)	(doritos)	0.202352	0.260402	0.050967	0.251875	0.967254	-0.001725
3	(pringles)	(doritos)	0.255976	0.260402	0.062603	0.244565	0.939183	-0.004054
2	(doritos)	(pringles)	0.260402	0.255976	0.062603	0.240408	0.939183	-0.004054
6	(kettle)	(pringles)	0.386999	0.255976	0.091564	0.236601	0.924312	-0.007498
0	(kettle)	(doritos)	0.386999	0.260402	0.090047	0.232680	0.893540	-0.010729
4	(doritos)	(smiths)	0.260402	0.202352	0.050967	0.195726	0.967254	-0.001725
8	(kettle)	(smiths)	0.386999	0.202352	0.075629	0.195425	0.965765	-0.002681
10	(kettle)	(thins)	0.386999	0.136208	0.051347	0.132680	0.974093	-0.001366

In [685]:
```
basket_sets.shape[0]
```

Out[685]: 7907

Explanation:

- a high lift value means that it occurs more frequently than would be expected given the number of transaction and product combinations
- rule is moderately strong as we used 7907 transactions for this model.
- Confidence signifies the likelihood of item y being purchased when item X is purchased
- Lift signifies the likeihood of item y being purchased when item x is purchased while taking into account the popularity of y.
  - if the Lift value is greater than 1, it means that y is likely ot be bought with x
  - Lift value less than 1 means item y unlikely ot be bought if item x is bought

In [704]:
```
basket_sets['kettle'].sum()
# comment: everytime product A (kettle) is bought, we cna recommend product

```

Out[704]: 3060

In [701]:
```
# making recommendations
basket_sets['doritos'].sum()
```

Out[701]: 2059

In [705]:
```
# sort from highest to lowest support value
frequent_itemsets.sort_values(by='support',ascending=False).head()
```

Out[705]:

	support	itemsets
4	0.386999	(kettle)
1	0.260402	(doritos)
5	0.255976	(pringles)
7	0.202352	(smiths)
3	0.143164	(infuzions)

In [732]:
```
# save as html
rules_sorted_df.to_html("html_formatted_dataframes/brand_affinity.html")
```

In [739]:
```
frequent_itemsets_sorted_df.to_html("html_formatted_dataframes/frequent_item
```

Recommendation:

- The evidence shows a < 1 Lift ratio, which means that the rules are independent from each other, where a purchase of a brand of chips, is not a strong indicator of the the purchase of another brand.

- mainstream customers in the young singles/couples segment have low support levels and <50% confidence levels between different brands. This indicates that the evidence does not support a strong purchase relationship between different brands.
- I also looked at how much opportunity there is to use the popularity of one product to drive sales of another. For instance, 3060 kettle chips were sold but only 2059 doritos so we could drive more dorito sales through recommendations

The Apriori Algorithm calculates the frequency of occurence for each brand for the mainstream customers in the Young Singles/Couple segment which calculated to be:

- kettle 33.48%
- Doritos 22.66%
- Pringles 23.04%
- Smiths 20.24%

In [707]:

```
# Same affinity (market basket analysis) but this time on chip packet size
mainstream_df.head()
```

Out[707]:

_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	BRAND_NAME	PROD_SIZE	PROD_Q
1	1020	26	19	smiths crinkle cut snag&sauce	smiths	150	
1	1020	27	7	smiths crinkle original	smiths	330	
1	1020	28	84	grnwves plus btroot & chilli jam	grainwaves	180	
1	1163	188	46	kettle original	kettle	175	
1	1163	189	12	natural chip co tmato hrb&spce	natural	175	

In [712]:

```
market_basket_pksize_df = mainstream_df.groupby(['LYLTY_CARD_NBR','PROD_SIZE
                            sum().\
                            unstack().\
                            reset_index().\
                            fillna(0).\
                            set_index('LYLTY_CARD_NBR')

```

In [713]:
```
market_basket_pksize_df
```

Out[713]:

PROD_SIZE	70	90	110	125	134	135	150	160	165	170	175	180	190	200	210
LYLTY_CARD_NBR															
1002	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1010	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0
1018	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1020	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
1060	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	1.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
272391	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2330041	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0
2330321	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2370181	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0
2373711	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

7907 rows × 20 columns

In [715]:
```
# use predefined "units_purchased" function to turn values into either 0 or
pksize_basket_sets = market_basket_pksize_df.applymap(units_purchased)
```

In [717]:
```
# check to see if changes are made
pksize_basket_sets
```

Out[717]:

PROD_SIZE	70	90	110	125	134	135	150	160	165	170	175	180	190	200	210
LYLTY_CARD_NBR															
1002	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1010	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1018	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0
1020	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
1060	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
272391	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
2330041	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
2330321	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2370181	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
2373711	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

7907 rows × 20 columns

In [718]:
```
pksize_frequent_itemsets = apriori(pksize_basket_sets, min_support=0.05, use
```

In [719]:
```
pksize_frequent_itemsets.sort_values(by='support', ascending = False)
```

Out[719]:

	support	itemsets
5	0.458202	(175)
2	0.313773	(150)
1	0.255976	(134)
0	0.224738	(110)
4	0.176932	(170)
8	0.139244	(330)
16	0.137726	(150, 175)
3	0.127861	(165)
14	0.111167	(134, 175)
12	0.094220	(110, 175)
18	0.080435	(170, 175)
9	0.076262	(380)
13	0.074997	(150, 134)
7	0.074870	(270)
6	0.070191	(210)
11	0.065385	(150, 110)
17	0.060073	(165, 175)
19	0.059188	(330, 175)
10	0.057797	(134, 110)
15	0.054762	(170, 150)

In [721]:
```
# generate rules
pksize_rules = association_rules(pksize_frequent_itemsets, metric='lift')
```

In [722]:
```
pksize_rules.head()
```

Out[722]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage
0	(134)	(110)	0.255976	0.224738	0.057797	0.225791	1.004685	0.000270
1	(110)	(134)	0.224738	0.255976	0.057797	0.257175	1.004685	0.000270
2	(150)	(110)	0.313773	0.224738	0.065385	0.208384	0.927231	-0.005131
3	(110)	(150)	0.224738	0.313773	0.065385	0.290940	0.927231	-0.005131
4	(110)	(175)	0.224738	0.458202	0.094220	0.419246	0.914981	-0.008755

In [733]:
```
# sort from highest to lowest
pksize_rules_sorted_df = pksize_rules.sort_values(by='confidence', ascending
pksize_rules_sorted_df
```

Out[733]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage
14	(165)	(175)	0.127861	0.458202	0.060073	0.469832	1.025382	0.001487
16	(170)	(175)	0.176932	0.458202	0.080435	0.454610	0.992162	-0.000635
12	(150)	(175)	0.313773	0.458202	0.137726	0.438936	0.957954	-0.006045
8	(134)	(175)	0.255976	0.458202	0.111167	0.434289	0.947811	-0.006121
18	(330)	(175)	0.139244	0.458202	0.059188	0.425068	0.927688	-0.004614
4	(110)	(175)	0.224738	0.458202	0.094220	0.419246	0.914981	-0.008755
10	(170)	(150)	0.176932	0.313773	0.054762	0.309507	0.986405	-0.000755
13	(175)	(150)	0.458202	0.313773	0.137726	0.300580	0.957954	-0.006045
7	(134)	(150)	0.255976	0.313773	0.074997	0.292984	0.933747	-0.005321
3	(110)	(150)	0.224738	0.313773	0.065385	0.290940	0.927231	-0.005131
1	(110)	(134)	0.224738	0.255976	0.057797	0.257175	1.004685	0.000270
9	(175)	(134)	0.458202	0.255976	0.111167	0.242617	0.947811	-0.006121
6	(150)	(134)	0.313773	0.255976	0.074997	0.239017	0.933747	-0.005321
0	(134)	(110)	0.255976	0.224738	0.057797	0.225791	1.004685	0.000270
2	(150)	(110)	0.313773	0.224738	0.065385	0.208384	0.927231	-0.005131
5	(175)	(110)	0.458202	0.224738	0.094220	0.205631	0.914981	-0.008755
17	(175)	(170)	0.458202	0.176932	0.080435	0.175545	0.992162	-0.000635
11	(150)	(170)	0.313773	0.176932	0.054762	0.174526	0.986405	-0.000755
15	(175)	(165)	0.458202	0.127861	0.060073	0.131107	1.025382	0.001487
19	(175)	(330)	0.458202	0.139244	0.059188	0.129175	0.927688	-0.004614

In [735]:
```
pksize_frequet_itemsets_sorted_df = pksize_frequent_itemsets.sort_values(by=
pksize_frequet_itemsets_sorted_df.head()
```

Out[735]:

	support	itemsets
5	0.458202	(175)
2	0.313773	(150)
1	0.255976	(134)
0	0.224738	(110)
4	0.176932	(170)

In [ ]:
```

```

In [740]:
```
# save as html table
pksize_rules_sorted_df.to_html("html_formatted_dataframes/pksize_affinity.ht
```

In [741]:
```
pksize_frequet_itemsets_sorted_df.to_html("html_formatted_dataframes/pksize_
```

Recommendation:

- similar to the relationship between brands, the low support, confidence, and life values indicates a weak relationship for purchase trends between package sizes.

Using the Apriori Algorithmn, we can see the frequency of purchase for specifc package sizes for mainstream custoemrs in the young singles/couples segment.

- 45.82% for 175g
- 31.38% for 150g
- 25.60% for 134g

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

affinity (market basket) analysis

preference to a certain type of brand

- Support (Prevaluence):
  - How frequent are itemsets, or consequent and antecedent purchased together
- Condidence (Predictability):
  - given a purchase of the antecedent, how likely is a purchase of the consequent
- Lift (Interest):
  - How much more likely is this association than we would expect by chance

In [ ]:
```

```

In [306]:
```
for_ttest_df
```

Out[306]:

	index	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	BRAI
0	0	2018-10-17	1	1000	1	5	natural chip compny seasalt	
1	1	2019-05-14	1	1307	348	True	ccs nacho cheese	
2	202	2018-11-10	1	1307	346	True	ww original stacked chips	v
3	203	2019-03-09	1	1307	347	True	ccs original	
4	2	2019-05-20	1	1343	383	True	smiths crinkle cut chips chicken	
...	...	...	...	...	...	...	...	
246326	264392	2019-03-09	272	272319	270088	89	kettle sweet chilli and sour cream	
246327	264393	2018-08-13	272	272358	270154	74	tostitos splash of lime	
246328	264394	2018-11-06	272	272379	270187	51	doritos mexicana	
246329	264395	2018-12-27	272	272379	270188	42	doritos corn chip mexican jalapeno	
246330	264396	2018-09-22	272	272380	270189	74	tostitos splash of lime	

246331 rows × 14 columns

support = transactions involving certain brands / total transaction

In [308]:
```
purchase_trend_df.head(3)
```

Out[308]:

	PREMIUM_CUSTOMER	BRAND_NAME	LIFESTAGE	LYLTY_CARD_NBR
0	Budget	burger	MIDAGE SINGLES/COUPLES	43
1	Budget	burger	NEW FAMILIES	18
2	Budget	burger	OLDER FAMILIES	159

In [314]:
```
# separate mainstream segment and (Budget and Premium) segments
mainstream_trend_df = purchase_trend_df.loc[purchase_trend_df['PREMIUM_CUSTO
print(mainstream_trend_df.head())
not_mainstream_trend_df = purchase_trend_df.loc[(purchase_trend_df['PREMIUM_
                                                | (purchase_trend_df['PREMIUM_C
not_mainstream_trend_df
```

```
  PREMIUM_CUSTOMER BRAND_NAME               LIFESTAGE  LYLTY_CARD_NBR
0       Mainstream     burger  MIDAGE SINGLES/COUPLES              48
1       Mainstream     burger            NEW FAMILIES              14
2       Mainstream     burger          OLDER FAMILIES             123
3       Mainstream     burger   OLDER SINGLES/COUPLES              93
4       Mainstream     burger                RETIREES             122
```

Out[314]:

	PREMIUM_CUSTOMER	BRAND_NAME	LIFESTAGE	LYLTY_CARD_NBR
0	Budget	burger	MIDAGE SINGLES/COUPLES	43
1	Budget	burger	NEW FAMILIES	18
2	Budget	burger	OLDER FAMILIES	159
3	Budget	burger	OLDER SINGLES/COUPLES	110
4	Budget	burger	RETIREES	66
...	...	...	...	...
275	Premium	woolworths	OLDER FAMILIES	636
276	Premium	woolworths	OLDER SINGLES/COUPLES	701
277	Premium	woolworths	RETIREES	470
278	Premium	woolworths	YOUNG FAMILIES	565
279	Premium	woolworths	YOUNG SINGLES/COUPLES	393

280 rows × 4 columns

In [315]:
```
# calculate support for mainstream
mainstream_trend_df['target'] = mainstream_trend_df['LYLTY_CARD_NBR'] / main
```

In [589]:
```
# group by brand
brand_group_df = pd.DataFrame(mainstream_trend_df.groupby(by='BRAND_NAME').s
brand_group_df.sort_values('target',ascending=False)
```

Out[589]:

BRAND_NAME	target
kettle	0.172477
smiths	0.118930
doritos	0.106389
pringles	0.104175
red rock deli	0.062431
infuzions	0.058406
thins	0.057289
woolworths	0.043525
cobs	0.040922
twisties	0.039763
tostitos	0.039278
grainwaves	0.032006
natural	0.028001
tyrrells	0.027190
cheezels	0.018221
ccs	0.017189
cheetos	0.011709
sunbites	0.010981
burger	0.005775
french	0.005343

In [323]:
```
# calculate support for premium and budget
not_mainstream_trend_df['target'] =not_mainstream_trend_df['LYLTY_CARD_NBR']
```

In [590]:
```
# group by brand
not_brand_group_df = pd.DataFrame(not_mainstream_trend_df.groupby(by='BRAND_
not_brand_group_df.rename(columns={"target": "non_target"}, inplace=True)
not_brand_group_df.sort_values('non_target', ascending=False)
```

Out[590]:

BRAND_NAME	non_target
kettle	0.163758
smiths	0.125631
pringles	0.100150
doritos	0.099496
red rock deli	0.068653
infuzions	0.057071
thins	0.057045
woolworths	0.050884
cobs	0.038265
tostitos	0.037744
twisties	0.037288
natural	0.031774
grainwaves	0.031008
tyrrells	0.025409
ccs	0.019281
cheezels	0.018845
sunbites	0.012982
cheetos	0.011991
burger	0.006709
french	0.006015

In [330]:
```
# merge the target and none target dataframes together
brand_supports_df = pd.merge(brand_group_df, not_brand_group_df, how='inner'
```

In [331]:
```
brand_supports_df.head()
```

Out[331]:

	target	non_target
BRAND_NAME		
burger	0.005775	0.006709
ccs	0.017189	0.019281
cheetos	0.011709	0.011991
cheezels	0.018221	0.018845
cobs	0.040922	0.038265

In [332]:
```
brand_supports_df['affinity_to_brand'] = brand_supports_df['target'] / brand

```

In [594]:
```
brand_supports_df.sort_values("affinity_to_brand", ascending=False).head()
```

Out[594]:

	target	non_target	affinity_to_brand
BRAND_NAME			
tyrrells	0.027190	0.025409	1.070096
cobs	0.040922	0.038265	1.069429
doritos	0.106389	0.099496	1.069273
twisties	0.039763	0.037288	1.066368
kettle	0.172477	0.163758	1.053243

In [ ]:
```
brand_supports_df['contains_both']
```

Mainstream customers in general are more likely to purchase chips from tyrrells compared to other brands

In [419]:
```
# save table in html format for use later
brand_supports_df.sort_values("affinity_to_brand", ascending=False).to_html(
```

In [ ]:
```

```

Further Analyze young single/couples because they are one of the top contributors to total sales

We could retain them or further increase sales

In [340]:
```
mainstream_trend_df.head(3)
```

Out[340]:

	PREMIUM_CUSTOMER	BRAND_NAME	LIFESTAGE	LYLTY_CARD_NBR	target
0	Mainstream	burger	MIDAGE SINGLES/COUPLES	48	0.000506
1	Mainstream	burger	NEW FAMILIES	14	0.000148
2	Mainstream	burger	OLDER FAMILIES	123	0.001296

In [388]:
```
# find out the exact lifestage label for young single couples
mainstream_trend_df['LIFESTAGE'].unique()
```

Out[388]:
```
array(['MIDAGE SINGLES/COUPLES', 'NEW FAMILIES', 'OLDER FAMILIES',
       'OLDER SINGLES/COUPLES', 'RETIREES', 'YOUNG FAMILIES',
       'YOUNG SINGLES/COUPLES'], dtype=object)
```

In [389]:
```
mainstream_youngcouples_df = mainstream_trend_df.drop(columns='target')
```

In [390]:
```
# select only the young singles and couples
mainstream_youngcouples_df = mainstream_youngcouples_df.loc[mainstream_young
```

In [391]:
```
# find the support value for the affinity analysis
target = mainstream_youngcouples_df['LYLTY_CARD_NBR'].sum()
mainstream_youngcouples_df['target'] = mainstream_youngcouples_df['LYLTY_CAR
```

In [392]:
```
# set the brand name column as index
mainstream_youngcouples_df = pd.DataFrame(mainstream_youngcouples_df.set_ind
```

In [393]:
```
mainstream_youngcouples_df.head()
```

Out[393]:

BRAND_NAME	target
burger	0.003177
ccs	0.011376
cheetos	0.008506
cheezels	0.017730
cobs	0.044274

In [399]:
```
not_mainstream_youngcouples_df = not_mainstream_trend_df.drop(columns='targe
```

In [400]:
```
not_mainstream_youngcouples_df = not_mainstream_youngcouples_df.loc[not_main
```

In [401]:
```
target = not_mainstream_youngcouples_df['LYLTY_CARD_NBR'].sum()
not_mainstream_youngcouples_df['non_target'] = not_mainstream_youngcouples_d
```

In [402]:
```
# set the brand name as index and get the sum of support value
not_mainstream_youngcouples_df = pd.DataFrame(not_mainstream_youngcouples_df
```

In [403]:
```
not_mainstream_youngcouples_df
```

Out[403]:

BRAND_NAME	non_target
burger	0.008047
ccs	0.025806
cheetos	0.013736
cheezels	0.018592
cobs	0.036837
doritos	0.088172
french	0.008047
grainwaves	0.029830
infuzions	0.052931
kettle	0.141866
natural	0.036975
pringles	0.094762
red rock deli	0.078599
smiths	0.136802
sunbites	0.016164
thins	0.055012
tostitos	0.033091
twisties	0.034270
tyrrells	0.023309
woolworths	0.067152

In [405]:
```
youngcouple_analysis_df = pd.merge(mainstream_youngcouples_df, not_mainstrean
```

In [406]:
```
youngcouple_analysis_df.head()
```

Out[406]:

BRAND_NAME	target	non_target
burger	0.003177	0.008047
ccs	0.011376	0.025806
cheetos	0.008506	0.013736
cheezels	0.017730	0.018592
cobs	0.044274	0.036837

In [408]:
```
youngcouple_target = youngcouple_analysis_df['target']
youngcouple_nontarget = youngcouple_analysis_df['non_target']
youngcouple_analysis_df['affinity_to_brand'] = youngcouple_target / youngcou
```

In [413]:
```
youngcouple_sorted_df = youngcouple_analysis_df.sort_values("affinity_to_bra
youngcouple_sorted_df.head()

```

Out[413]:

BRAND_NAME	target	non_target	affinity_to_brand
kettle	0.196208	0.141866	1.383051
doritos	0.121804	0.088172	1.381433
tostitos	0.045555	0.033091	1.376669
tyrrells	0.031719	0.023309	1.360810
twisties	0.046016	0.034270	1.342751

In [635]:
```
youngcouple_sorted_df.tail()
```

Out[635]:

BRAND_NAME	target	non_target	affinity_to_brand
french	0.003997	0.008047	0.496687
ccs	0.011376	0.025806	0.440815
sunbites	0.006559	0.016164	0.405789
burger	0.003177	0.008047	0.394802
woolworths	0.024545	0.067152	0.365516

From the dataframe above, we can conclude that mainstream young singles/couples are 38% more likely to purchase Kettle branded chips compared to other brands. Although the top three choices of kettle, doritos, and tostitos are closely competing.

Brands like Sunbites, Burger, and Woolworths have more than 40% less likelihood to be the brand selection

In [415]:
```
# save as html table to use later
youngcouple_sorted_df.to_html("html_formatted_dataframes/youngcouple_affinit
```

In [ ]:
```

```

In [ ]:
```

```

Find the Affinity to pack size

Check to see which size do young singles and couples prefer

In [412]:
```
complete_df.head()
```

Out[412]:

	index	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	BRAND_NAM
0	0	2018-10-17	1	1000	1	5	natural chip compny seasalt	natu
1	1	2019-05-14	1	1307	348	66	ccs nacho cheese	c
2	202	2018-11-10	1	1307	346	96	ww original stacked chips	woolwort
3	203	2019-03-09	1	1307	347	54	ccs original	c
4	2	2019-05-20	1	1343	383	61	smiths crinkle cut chips chicken	smit

In [438]:

```
# target segment
mainstream_df = complete_df.loc[complete_df['PREMIUM_CUSTOMER'] == "Mainstre
mainstream_df = mainstream_df.groupby(['PROD_SIZE'])
mainstream_df = pd.DataFrame(mainstream_df.sum()['PROD_QTY'])
mainstream_df['target'] = mainstream_df['PROD_QTY'] / mainstream_df['PROD_QT
mainstream_df
```

Out[438]:

PROD_SIZE	PROD_QTY	target
70	988	0.005486
90	1973	0.010956
110	16990	0.094342
125	944	0.005242
134	18771	0.104232
135	2484	0.013793
150	29344	0.162942
160	1921	0.010667
165	11093	0.061597
170	14762	0.081971
175	47624	0.264447
180	976	0.005420
190	2095	0.011633
200	2933	0.016286
210	4783	0.026559
220	1040	0.005775
250	2335	0.012966
270	4815	0.026737
330	9359	0.051969
380	4859	0.026981

In [436]:

```
# non target segment
not_mainstream_df = complete_df.loc[(complete_df['PREMIUM_CUSTOMER'] == "Pre
                                     | (complete_df['PREMIUM_CUSTOMER'] == "Bu
not_mainstream_df = not_mainstream_df.groupby(['PROD_SIZE'])
not_mainstream_df = pd.DataFrame(not_mainstream_df.sum()['PROD_QTY'])
not_mainstream_df['non_target'] = not_mainstream_df['PROD_QTY'] / not_mainst
not_mainstream_df

```

Out[436]:

	PROD_QTY	non_target
PROD_SIZE		
70	1867	0.006472
90	3719	0.012892
110	25690	0.089056
125	1786	0.006191
134	28998	0.100524
135	3700	0.012826
150	47033	0.163044
160	3683	0.012767
165	17896	0.062038
170	23185	0.080373
175	78423	0.271859
180	1788	0.006198
190	3578	0.012403
200	5492	0.019038
210	7144	0.024765
220	1930	0.006690
250	3685	0.012774
270	7129	0.024713
330	14410	0.049953
380	7333	0.025420

In [440]:

```
pack_size_df = pd.merge(pd.DataFrame(mainstream_df),
                 pd.DataFrame(not_mainstream_df),
                 how='inner',
                right_index = True,
                left_index=True)
```

In [441]:
```
1 pack_size_df.head()
```

Out[441]:

	PROD_QTY_x	target	PROD_QTY_y	non_target
PROD_SIZE				
70	988	0.005486	1867	0.006472
90	1973	0.010956	3719	0.012892
110	16990	0.094342	25690	0.089056
125	944	0.005242	1786	0.006191
134	18771	0.104232	28998	0.100524

In [442]:
```
1 pack_size_df['affinity_to_packsize'] = pack_size_df['target'] / pack_size_df
```

In [444]:
```
1 pack_size_sorted_df = pack_size_df.sort_values(by='affinity_to_packsize', as
2 pack_size_sorted_df.head()
```

Out[444]:

	PROD_QTY_x	target	PROD_QTY_y	non_target	affinity_to_packsize
PROD_SIZE					
270	4815	0.026737	7129	0.024713	1.081881
135	2484	0.013793	3700	0.012826	1.075380
210	4783	0.026559	7144	0.024765	1.072435
380	4859	0.026981	7333	0.025420	1.061395
110	16990	0.094342	25690	0.089056	1.059354

In [447]:
```
1 pack_size_sorted_df.columns.values
```

Out[447]:
```
array(['PROD_QTY_x', 'target', 'PROD_QTY_y', 'non_target',
       'affinity_to_packsize'], dtype=object)
```

In [450]:
```
1 pack_size_sorted_df.drop(columns=['PROD_QTY_x', 'PROD_QTY_y'], inplace=True)
```

In [451]:
```
1 pack_size_sorted_df.head()
```

Out[451]:

	target	non_target	affinity_to_packsize
PROD_SIZE			
270	0.026737	0.024713	1.081881
135	0.013793	0.012826	1.075380
210	0.026559	0.024765	1.072435
380	0.026981	0.025420	1.061395
110	0.094342	0.089056	1.059354

result: mainstream young singles/couples are more likly to buy size 270g compared to other pack result: mainstream young singles/couples are more likely to buy size 270g compared to other pack sizes

Next we can look at which brands offer this specific pack size which is ideal for this customer segment

In [452]:
```
# save as html table
pack_size_sorted_df.to_html("html_formatted_dataframes/youngcouple_affinity_
```

In [453]:
```
complete_df.columns.values
```

Out[453]:
```
array(['index', 'DATE', 'STORE_NBR', 'LYLTY_CARD_NBR', 'TXN_ID',
       'PROD_NBR', 'PROD_NAME', 'BRAND_NAME', 'PROD_SIZE', 'PROD_QTY',
       'TOT_SALES', 'LIFESTAGE', 'PREMIUM_CUSTOMER'], dtype=object)
```

In [455]:
```
packsize_270_df = complete_df.loc[complete_df['PROD_SIZE'] == 270]
packsize_270_df.head()
```

Out[455]:

	index	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	BRAND_N
31	11	2019-05-18	9	9208	8634	15	twisties cheese	tw
76	789	2018-08-24	39	39167	35639	113	twisties chicken	tw
122	1026	2019-05-06	54	54305	48304	15	twisties cheese	tw
129	41	2019-05-20	55	55073	48887	113	twisties chicken	tw
206	1514	2019-01-01	80	80182	78980	15	twisties cheese	tw

In [456]:
```
packsize_270_df['BRAND_NAME'].unique()
```

Out[456]:
```
array(['twisties'], dtype=object)
```

Twisties is the only brand that offers the 270 pack size

In [459]:
```
packsize_top3_df = complete_df.loc[complete_df['PROD_SIZE'].isin([270, 135,
packsize_top3_df.head()
```

Out[459]:

	index	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	BRAND_NA
11	7	2019-05-16	4	4196	3539	24	grain waves sweet chilli	grainwa
20	9	2018-08-18	7	7150	6900	52	grain waves sour cream&chives	grainwa
31	11	2019-05-18	9	9208	8634	15	twisties cheese	twis
32	382	2018-12-28	9	9208	8633	24	grain waves sweet chilli	grainwa
76	789	2018-08-24	39	39167	35639	113	twisties chicken	twis

In [462]:
```
top3_brands = ','.join(packsize_top3_df['BRAND_NAME'].unique())
print(f"The brands that sell more preferable pack sizes for young couples an
```

```
The brands that sell more preferable pack sizes for young couples and singles a
re grainwaves,twisties,kettle
```

In [ ]:
```

```

In [466]:
```
transaction_df.head().to_html("html_formatted_dataframes/transaction_df.html
```

In [467]:
```
behavior_df.head().to_html("html_formatted_dataframes/customer_behavior_df.h
```

Conclusion

Sales are the highest for Budget customers in the "Older Families" segment, mainstream customers in the "young singles/couples" segment, and mainstream customers from the "Retirees" segment respectively.

I later found out that the high sales numbers for (Mainstream, young singles/couples) and (Mainstream, Retirees) can be explained by these segments having more customers. Mainstream customers from the young singles/couples segment are also likely to spend more per package of chips compared to other types of customers.

Using the Affinity (Market Basket) AnalysisI found out that the relationship between brands and package sizes in terms of purchase habits is weak, meaning that we cannot accurately assume that a customer will purchase a specific brand based on the purchase of one brand of chips. Likewise, we cannot assume that a customer will purchase a specific package size based on the previous purchase.

The client could relocate shelf placement of Kettle chips to popular shopping locations for mainstream customers, as the Kettle brand is the most popular for this customer segment. Although not very clear, there are still some noticeable shopping patterns among chip brands like Kettle and Dorito. The client could try recommending Dorito chips to shoppers looking to buy Kettle chips, by placing the two brands in the same section in a store aisle. Since this group of shoppers enjoy Kettle chips already, client could further boost sales by placing Kettle chips at eye level in the middle section of a store aisle.

In [ ]:

```

```