In [1]:
```
# python libraries

# Data Analysis and wrangling

import pandas as pd
import numpy as np
import missingno

# to load excel file

import openpyxl

# Data Visualization

import matplotlib.pyplot as plt
import seaborn as sns

#  time and dates

import datetime
from matplotlib.dates import DateFormatter

# Text analysis
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist as fdist
import re

# Statistical analysis
from scipy.stats import ttest_ind

# Remove warnings
import warnings
warnings.filterwarnings('ignore')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Rukaiyya\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
```

Load Data Transaction Data as td Customer data as cd

In [2]:
```
Cd = pd.read_csv('C:/Users/Rukaiyya/Data Science/Internship Quntium Data Analytics/QVI_
td = pd.read_excel('C:/Users/Rukaiyya/Data Science/Internship Quntium Data Analytics/QV
```

Transaction Data

In [3]:
```
td.head()
```

Out[3]:

	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

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES
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

observations from transaction data table 1) Date is not in right format 2) Transaction ID should be unique number, it will be check later if all are unique or not. 3) Product Number and product name should not be mismatch, it will check be check later.

In [4]:
```
td.shape
```

Out[4]: (264836, 8)

In [5]:
```
#checking unique transaction ID
td['TXN_ID'].nunique()
```

Out[5]: 263127

In [6]:
```
# it is seen that total number of rows in transaction data is not equal to the number o
# looking for duplicated TXN ID
td[td.duplicated(['TXN_ID'])].head()
```

Out[6]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES
42	43605	55	55073	48887	113	Twisties Chicken270g	1	4.6
377	43475	7	7364	7739	20	Doritos Cheese Supreme 330g	2	11.4
419	43391	12	12301	10982	93	Doritos Corn Chip Southern Chicken 150g	2	7.8
476	43351	16	16427	14546	81	Pringles Original Crisps 134g	1	3.7
511	43315	19	19272	16683	31	Infzns Crn Crnchers Tangy Gcamole 110g	2	7.6

In [7]:
```
# looking at 42 row have TXN ID 48887
td.loc[td['TXN_ID']== 48887,:]
```

Out[7]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES
41	43605	55	55073	48887	4	Dorito Corn Chp Supreme 380g	1	3.25
42	43605	55	55073	48887	113	Twisties Chicken270g	1	4.60

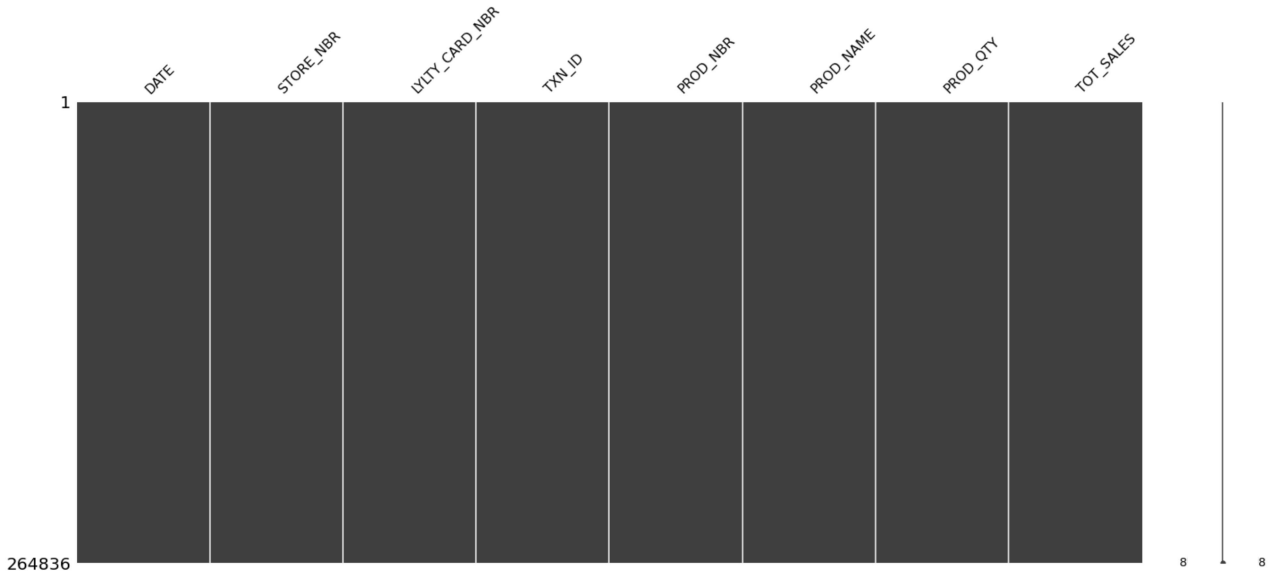
In [8]:
```
# look at STORE_NBR, LYLTY_CARD_NBR, TXN_ID all are same
```

In [9]:
```
td.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 [10]:
```
# first goal to find missing value
# for finding missing value we will plot graph

missingno.matrix(td)
```

Out[10]:
```
<AxesSubplot:>
```



as we see from above graph there is no missing value present.

In [11]:
```
#now we will see date feature
td['DATE'].head()
```

Out[11]:
```
0    43390
1    43599
2    43605
3    43329
4    43330
Name: DATE, dtype: int64
```

In [12]:
```
#DATE is not in right formate we have to convert it into right format
# we have to define function for converstion excel integer into date yyyy-mm-dd format

def xlseriesdate_to_datetime(xlseriesdate):
    excel_anchor = datetime.datetime(1900,1,1)
    if (xlseriesdate < 60) :
        delta_in_days = datetime.timedelta(days = xlseriesdate-1)
    else:
        delta_in_days = datetime.timedelta(days = xlseriesdate-2)
    converted_date = excel_anchor+ delta_in_days
    return converted_date
```

In [13]:
```
# now apply feature to change date in right format
td['DATE']= td['DATE'].apply(xlseriesdate_to_datetime)
```

In [14]:
```
# check it is in right format or not
td['DATE'].head()
```

Out[14]:
```
0   2018-10-17
1   2019-05-14
2   2019-05-20
3   2018-08-17
4   2018-08-18
Name: DATE, dtype: datetime64[ns]
```

In [15]:
```
# date is successfully changed to right format
# now check product name
td['PROD_NAME'].head()
```

Out[15]:
```
0      Natural Chip        Compny SeaSalt175g
1                    CCs Nacho Cheese    175g
2      Smiths Crinkle Cut  Chips Chicken 170g
3      Smiths Chip Thinly  S/Cream&Onion 175g
4    Kettle Tortilla ChpsHny&Jlpno Chili 150g
Name: PROD_NAME, dtype: object
```

In [16]:
```
# we see size of product is combined wiht name
# to ease the analysis we now make sepatate comumn or feature call as PACK_SIZETt
td['PACK_SIZE'] = td['PROD_NAME'].str.extract("(\d+)")
td['PACK_SIZE'] = pd.to_numeric(td['PACK_SIZE'])
td.head()
```

Out[16]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES
0	2018-10-17	1	1000	1	5	Natural Chip Compny SeaSalt175g	2	6.0
1	2019-05-14	1	1307	348	66	CCs Nacho Cheese 175g	3	6.3
2	2019-05-20	1	1343	383	61	Smiths Crinkle Cut Chips Chicken 170g	2	2.9
3	2018-08-17	2	2373	974	69	Smiths Chip Thinly S/Cream&Onion 175g	5	15.0
4	2018-08-18	2	2426	1038	108	Kettle Tortilla ChpsHny&Jlpno Chili 150g	3	13.8

In [17]:
```
# Create text cleaning function for PROD_NAME feature
def clean_text(text):
    text = re.sub('[&/]', ' ', text) # remove special characters '&' and '/'
    text = re.sub('\d\w*', ' ', text) # remove product weights
    return text

# Apply text cleaning function to PROD_NAME column
td['PROD_NAME'] = td['PROD_NAME'].apply(clean_text)
```

In [19]:
```
# Create one giant string and apply 'word_tokenize' to separate the words

cleanProdName = td['PROD_NAME']
string = ''.join(cleanProdName)
prodWord = word_tokenize(string)
```

In [20]:
```
# Apply 'fdist' function which computes the frequency of each token and put it into a d
```

```
wordFrequency = fdist(prodWord)
freq_df = pd.DataFrame(list(wordFrequency.items()), columns = ["Word", "Frequency"]).so
```

In [21]:
```
# Let's see the top 5 most frequent words

freq_df.head()
```

Out[21]:

	Word	Frequency
10	Chips	49770
16	Kettle	40739
7	Smiths	28572
6	Cheese	27890
66	Pringles	24743

In [23]:
```
# Drop rows with salsa word in PROD_NAME

td['PROD_NAME'] = td['PROD_NAME'].apply(lambda x: x.lower())
td = td[~td['PROD_NAME'].str.contains("salsa")]
td['PROD_NAME'] = td['PROD_NAME'].apply(lambda x: x.title())
```

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

Out[24]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	P/
0	2018-10-17	1	1000	1	5	Natural Chip Compny Seasalt	2	6.0	
1	2019-05-14	1	1307	348	66	Ccs Nacho Cheese	3	6.3	
2	2019-05-20	1	1343	383	61	Smiths Crinkle Cut Chips Chicken	2	2.9	
3	2018-08-17	2	2373	974	69	Smiths Chip Thinly S Cream Onion	5	15.0	
4	2018-08-18	2	2426	1038	108	Kettle Tortilla Chpshny Jlpno Chili	3	13.8	

In [25]:
```
# Value counts of PROD_QTY

td['PROD_QTY'].value_counts()
```

Out[25]:
```
2      220070
1       25476
5         415
3         408
4         371
200         2
Name: PROD_QTY, dtype: int64
```

We have two occurences of 200 in the dataset. This seems odd so let's explore further.

In [26]:
```
td.loc[td['PROD_QTY'] == 200, :]
```

Out[26]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALE
69762	2018-08-19	226	226000	226201	4	Dorito Corn Chp Supreme	200	650.
69763	2019-05-20	226	226000	226210	4	Dorito Corn Chp Supreme	200	650.

Both these transactions have been made by the same person at the same store. Let's see all the transactions this person has made by tracking his loyalty card number

In [27]:
```
td.loc[td['LYLTY_CARD_NBR'] == 226000, :]
```

Out[27]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALE
69762	2018-08-19	226	226000	226201	4	Dorito Corn Chp Supreme	200	650.
69763	2019-05-20	226	226000	226210	4	Dorito Corn Chp Supreme	200	650.

This customer has only made two transactions over the entire year so unlikely to be a retail customer. He/she is most likely purchasing for commercial purposes so it is safe for us to drop these this customer from both 'transaction Data' and 'customer Data' dataset

In [32]:
```
td.drop(td.index[td['LYLTY_CARD_NBR'] == 226000], inplace = True)
Cd.drop(Cd.index[Cd['LYLTY_CARD_NBR'] == 226000], inplace = True)
```

In [33]:
```
# Make sure it has been dropped

td.loc[td['LYLTY_CARD_NBR'] == 226000]
```

Out[33]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PA

In [34]:
```
# Now let's examine the number of transactions over time to see if there are any obviou

td['DATE'].nunique()
```

Out[34]: 364

In [35]:
```
# Look for the missing date

pd.date_range(start = '2018-07-01', end = '2019-06-30').difference(td['DATE'])
```

Out[35]: DatetimeIndex(['2018-12-25'], dtype='datetime64[ns]', freq=None)

We have a missing date on Christmas Day. This makes sense because most retail stores are closed that day.

In [37]:
```
# Create a new dataframe which contains the total sale for each date

a = pd.pivot_table(td, values = 'TOT_SALES', index = 'DATE', aggfunc = 'sum')
a.head()
```

Out[37]:

DATE	TOT_SALES
2018-07-01	4920.1
2018-07-02	4877.0
2018-07-03	4954.7
2018-07-04	4968.1
2018-07-05	4682.0

In [38]:
```
b = pd.DataFrame(index = pd.date_range(start = '2018-07-01', end = '2019-06-30'))
b['TOT_SALES'] = 0
len(b)
```

Out[38]: 365

In [39]:
```
c = a + b
c.fillna(0, inplace = True)
```

In [40]:
```
c.head()
```

Out[40]:

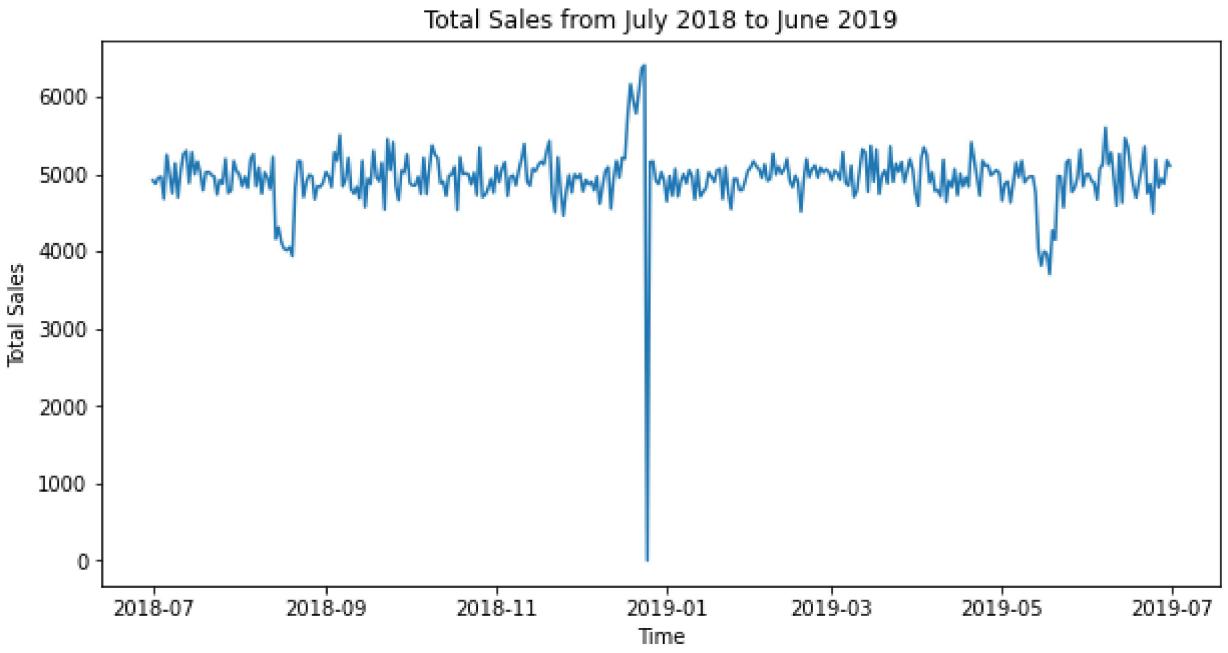
	TOT_SALES
2018-07-01	4920.1
2018-07-02	4877.0
2018-07-03	4954.7
2018-07-04	4968.1
2018-07-05	4682.0

In [41]:
```
c.index.name = 'Date'
c.rename(columns = {'TOT_SALES': 'Total Sales'}, inplace = True)
c.head()
```

Out[41]:

Date	Total Sales
2018-07-01	4920.1
2018-07-02	4877.0
2018-07-03	4954.7
2018-07-04	4968.1
2018-07-05	4682.0

In [42]:
```
timeline = c.index
graph = c['Total Sales']

fig, ax = plt.subplots(figsize = (10, 5))
ax.plot(timeline, graph)

date_form = DateFormatter("%Y-%m")
ax.xaxis.set_major_formatter(date_form)
plt.title('Total Sales from July 2018 to June 2019')
plt.xlabel('Time')
plt.ylabel('Total Sales')

plt.show()
```



We can see that sales spike up during the December month and zero sale on Christmas Day

In [43]:
```
# Confirm the date where sales count equals to zero

c[c['Total Sales'] == 0]
```

Out[43]:

Date	Total Sales
2018-12-25	0.0

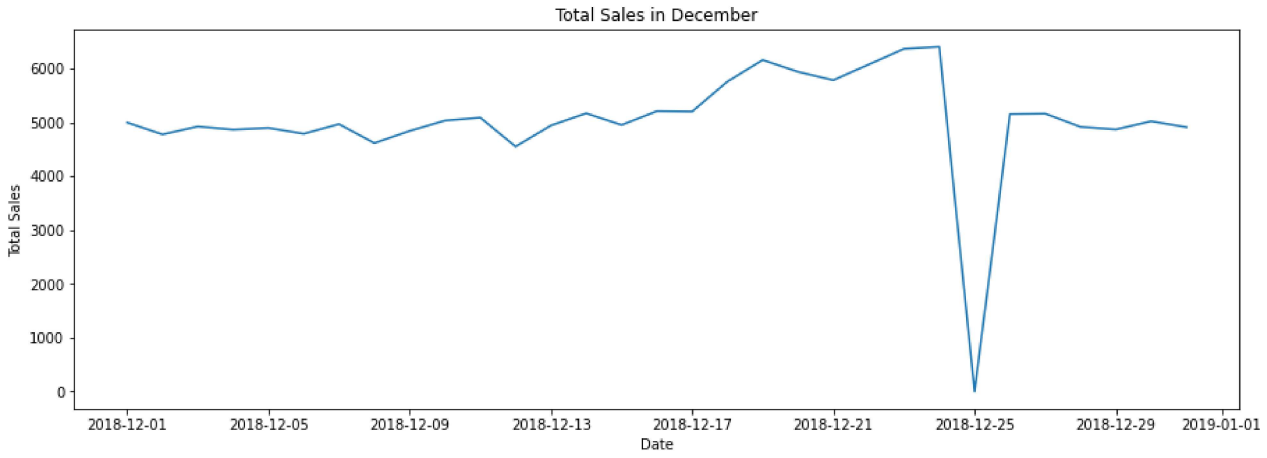
In [44]:
```
# Let's look at the December month only

c_december = c[(c.index < "2019-01-01") & (c.index > "2018-11-30")]
c_december.head()
```

Out[44]:

Date	Total Sales
2018-12-01	5000.9
2018-12-02	4781.1
2018-12-03	4927.0
2018-12-04	4869.4
2018-12-05	4900.5

In [45]:
```
plt.figure(figsize = (15, 5))
plt.plot(c_december)
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.title('Total Sales in December')
```

Out[45]:
```
Text(0.5, 1.0, 'Total Sales in December')
```



In [46]:
```
# Reset index

c_december.reset_index(drop = True, inplace = True)
c_december.head()
```

Out[46]:

	Total Sales
0	5000.9

	Total Sales
1	4781.1
2	4927.0
3	4869.4
4	4900.5

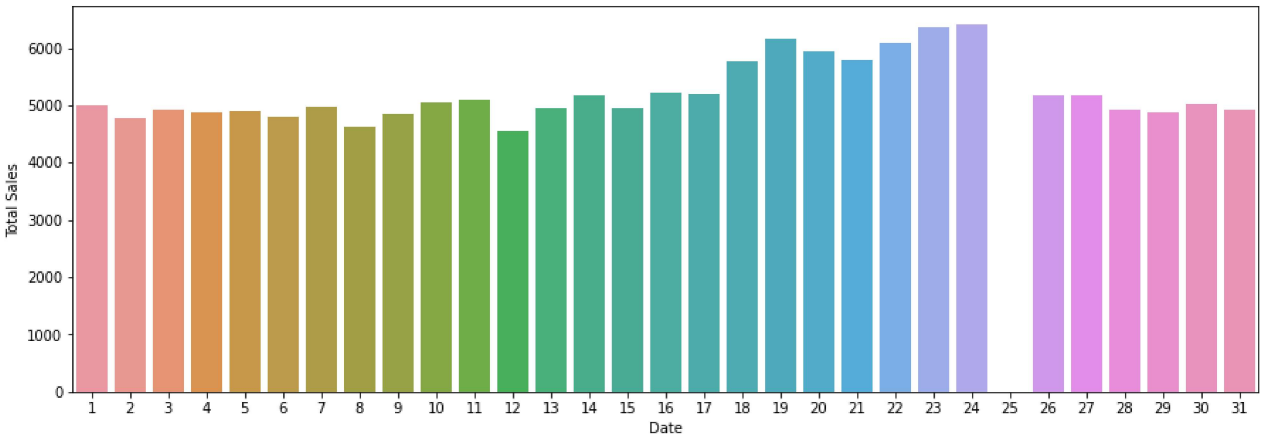
In [47]:
```
# Relabel Date

c_december['Date'] = c_december.index + 1
c_december.head()
```

Out[47]:

	Total Sales	Date
0	5000.9	1
1	4781.1	2
2	4927.0	3
3	4869.4	4
4	4900.5	5

In [48]:
```
plt.figure(figsize = (15,5))
sns.barplot(x = 'Date', y ='Total Sales', data = c_december)
```

Out[48]:
```
<AxesSubplot:xlabel='Date', ylabel='Total Sales'>
```



In [49]:
```
td['PACK_SIZE'].head()
```

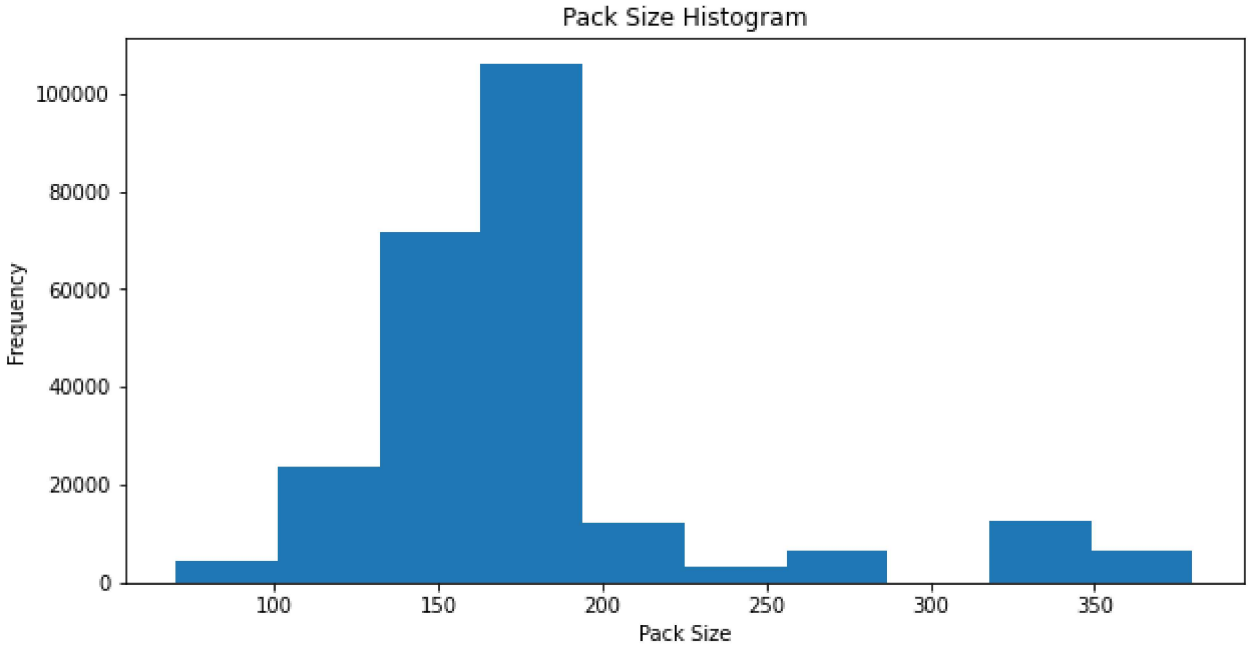
Out[49]:
```
0    175
1    175
2    170
3    175
4    150
Name: PACK_SIZE, dtype: int64
```

In [50]:
```
td['PACK_SIZE'].unique()
```

Out[50]:
```
array([175, 170, 150, 330, 210, 270, 220, 125, 110, 134, 380, 180, 165,
       135, 250, 200, 160, 190,  90,  70], dtype=int64)
```

In [51]:
```
# Check the distribution of PACK_SIZE

plt.figure(figsize = (10, 5))
plt.hist(td['PACK_SIZE'])
plt.xlabel('Pack Size')
plt.ylabel('Frequency')
plt.title('Pack Size Histogram')
```

Out[51]:
```
Text(0.5, 1.0, 'Pack Size Histogram')
```



In [52]:
```
# Extract brand name from PROD_NAME and create new column called BRAND

part = td['PROD_NAME'].str.partition()
td['BRAND'] = part[0]
td.head()
```

Out[52]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	P/
0	2018-10-17	1	1000	1	5	Natural Chip Compny Seasalt	2	6.0	
1	2019-05-14	1	1307	348	66	Ccs Nacho Cheese	3	6.3	
2	2019-05-20	1	1343	383	61	Smiths Crinkle Cut Chips Chicken	2	2.9	
3	2018-08-17	2	2373	974	69	Smiths Chip Thinly S Cream Onion	5	15.0	

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	P/
4	2018-08-18	2	2426	1038	108	Kettle Tortilla Chpshny Jlpno Chili	3	13.8	

In [53]:
```
td['BRAND'].unique()
```

Out[53]:
```
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)
```

In [54]:
```
# Rename brand names for consistency

td['BRAND'].replace('Ncc', 'Natural', inplace = True)
td['BRAND'].replace('Ccs', 'CCS', inplace = True)
td['BRAND'].replace('Smith', 'Smiths', inplace = True)
td['BRAND'].replace(['Grain', 'Grnwves'], 'Grainwaves', inplace = True)
td['BRAND'].replace('Dorito', 'Doritos', inplace = True)
td['BRAND'].replace('Ww', 'Woolworths', inplace = True)
td['BRAND'].replace('Infzns', 'Infuzions', inplace = True)
td['BRAND'].replace(['Red', 'Rrd'], 'Red Rock Deli', inplace = True)
td['BRAND'].replace('Snbts', 'Sunbites', inplace = True)

td['BRAND'].unique()
```

Out[54]:
```
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)
```

In [55]:
```
# Which brand had the most sales?

td.groupby('BRAND').TOT_SALES.sum().sort_values(ascending = False)
```

Out[55]:
```
BRAND
Kettle           390239.8
Doritos          226329.9
Smiths           217492.0
Pringles         177655.5
Infuzions         99047.6
Thins             88852.5
Red Rock Deli     87607.5
Twisties          81522.1
Tostitos          79789.6
Cobs              70569.8
Tyrrells          51647.4
Grainwaves        51617.2
Natural           42318.0
Woolworths        41059.1
Cheezels          40029.9
CCS               18078.9
Cheetos           16884.5
Sunbites           9676.4
French             7929.0
```

```
Burger             6831.0
Name: TOT_SALES, dtype: float64
```

Customer Data

In [57]:
```
list(Cd.columns)
```

Out[57]:
```
['LYLTY_CARD_NBR', 'LIFESTAGE', 'PREMIUM_CUSTOMER']
```

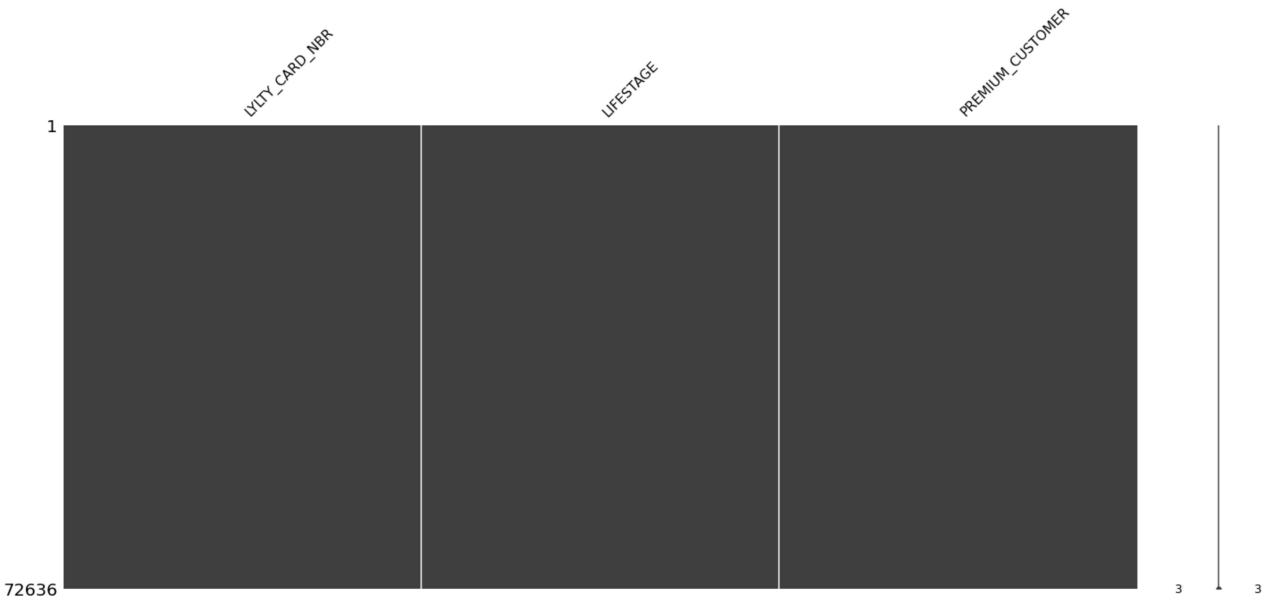
In [58]:
```
Cd.head()
```

Out[58]:

	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 [59]:
```
# Missing values in customerData

missingno.matrix(Cd)
```

Out[59]:
```
<AxesSubplot:>
```



In [60]:
```
len(Cd)
```

Out[60]:
```
72636
```

In [61]:
```
Cd['LYLTY_CARD_NBR'].nunique()
```

Out[61]: 72636

Since the number of rows in customerData is equal to number of unique loyalty card number, we conclude that loyalty card numbers are unique to each row.M

In [62]:
```
# How many unique lifestages?

Cd['LIFESTAGE'].nunique()
```

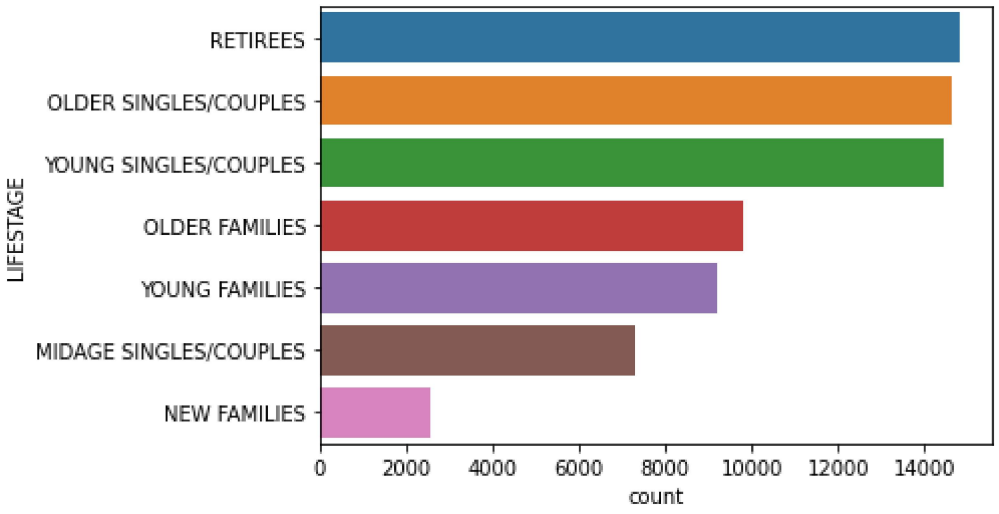
Out[62]: 7

In [63]:
```
# Value counts for lifestages

Cd['LIFESTAGE'].value_counts().sort_values(ascending = False)
```

Out[63]:
```
RETIREES                14805
OLDER SINGLES/COUPLES   14609
YOUNG SINGLES/COUPLES   14441
OLDER FAMILIES           9779
YOUNG FAMILIES           9178
MIDAGE SINGLES/COUPLES   7275
NEW FAMILIES             2549
Name: LIFESTAGE, dtype: int64
```

In [64]:
```
sns.countplot(y = Cd['LIFESTAGE'], order = Cd['LIFESTAGE'].value_counts().index)
```

Out[64]: <AxesSubplot:xlabel='count', ylabel='LIFESTAGE'>



In [65]:
```
# How many unique premium customer categories?

Cd['PREMIUM_CUSTOMER'].nunique()
```

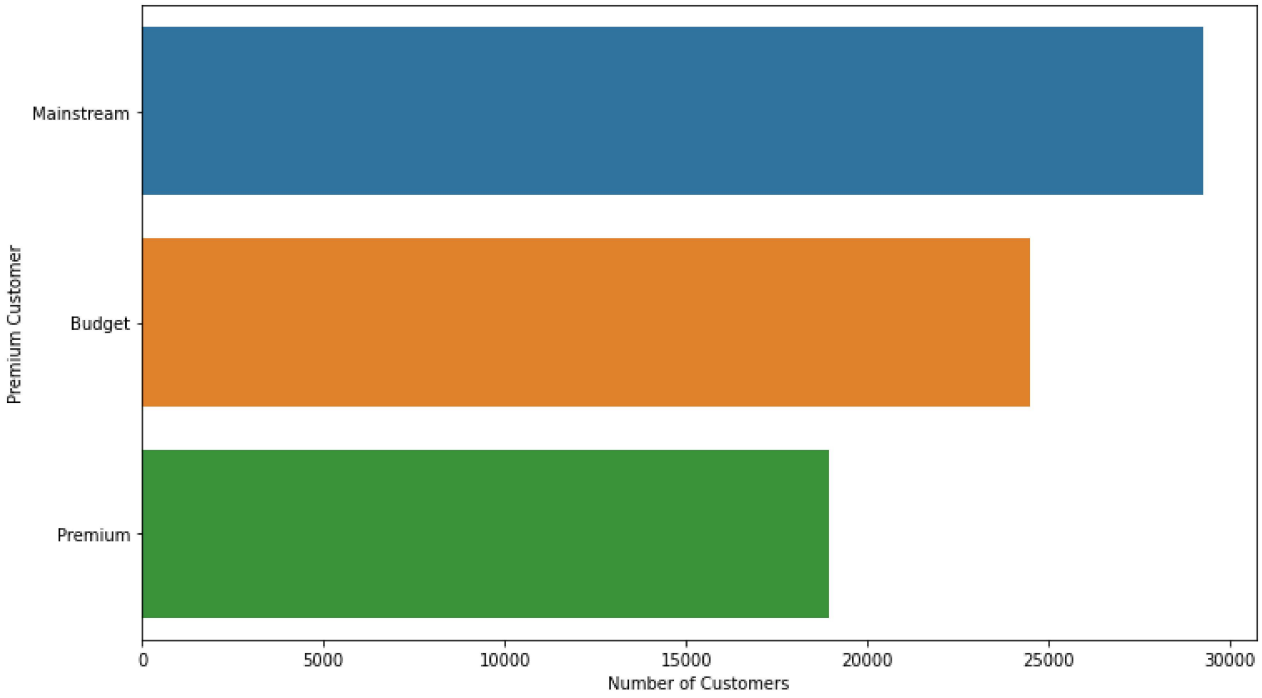
Out[65]: 3

In [66]:
```
# Value counts for each premium customer category

Cd['PREMIUM_CUSTOMER'].value_counts().sort_values(ascending = False)
```

Out[66]:
```
Mainstream    29245
Budget        24470
Premium       18921
Name: PREMIUM_CUSTOMER, dtype: int64
```

In [67]:
```
plt.figure(figsize = (12, 7))
sns.countplot(y = Cd['PREMIUM_CUSTOMER'], order = Cd['PREMIUM_CUSTOMER'].value_counts()
plt.xlabel('Number of Customers')
plt.ylabel('Premium Customer')
```

Out[67]:
```
Text(0, 0.5, 'Premium Customer')
```



In [80]:
```
# Merge transactionData and customerData together

combineData = pd.merge(td, Cd)
```

In [81]:
```
print("Transaction data shape: ", td.shape)
print("Customer data shape: ", Cd.shape)
print("Combined data shape: ", combinedData.shape)
```

```
Transaction data shape:  (246740, 10)
Customer data shape:  (246740, 12)
Combined data shape:  (246742, 12)
```

In [82]:
```
# Check for null values

combineData.isnull().sum()
```

Out[82]:
```
DATE              0
STORE_NBR         0
LYLTY_CARD_NBR    0
TXN_ID            0
PROD_NBR          0
PROD_NAME         0
```

```
PROD_QTY            0
TOT_SALES           0
PACK_SIZE           0
BRAND               0
LIFESTAGE           0
PREMIUM_CUSTOMER    0
dtype: int64
```

Data analysis on customer segments

Now that our data is ready for analysis, we can define some metrics of interest to the client:

Who spends the most on chips, describing customers by lifestage and how premium their general purchasing behaviour is How many customers are in each segment How many chips are bought per customer by segment What is the average chip price by customer segment

In [83]:
```
# Total sales by PREMIUM_CUSTOMER and LIFESTAGE

sales = pd.DataFrame(combineData.groupby(['PREMIUM_CUSTOMER', 'LIFESTAGE']).TOT_SALES.s
sales.rename(columns = {'TOT_SALES': 'Total Sales'}, inplace = True)
sales.sort_values(by = 'Total Sales', ascending = False, inplace = True)
sales
```

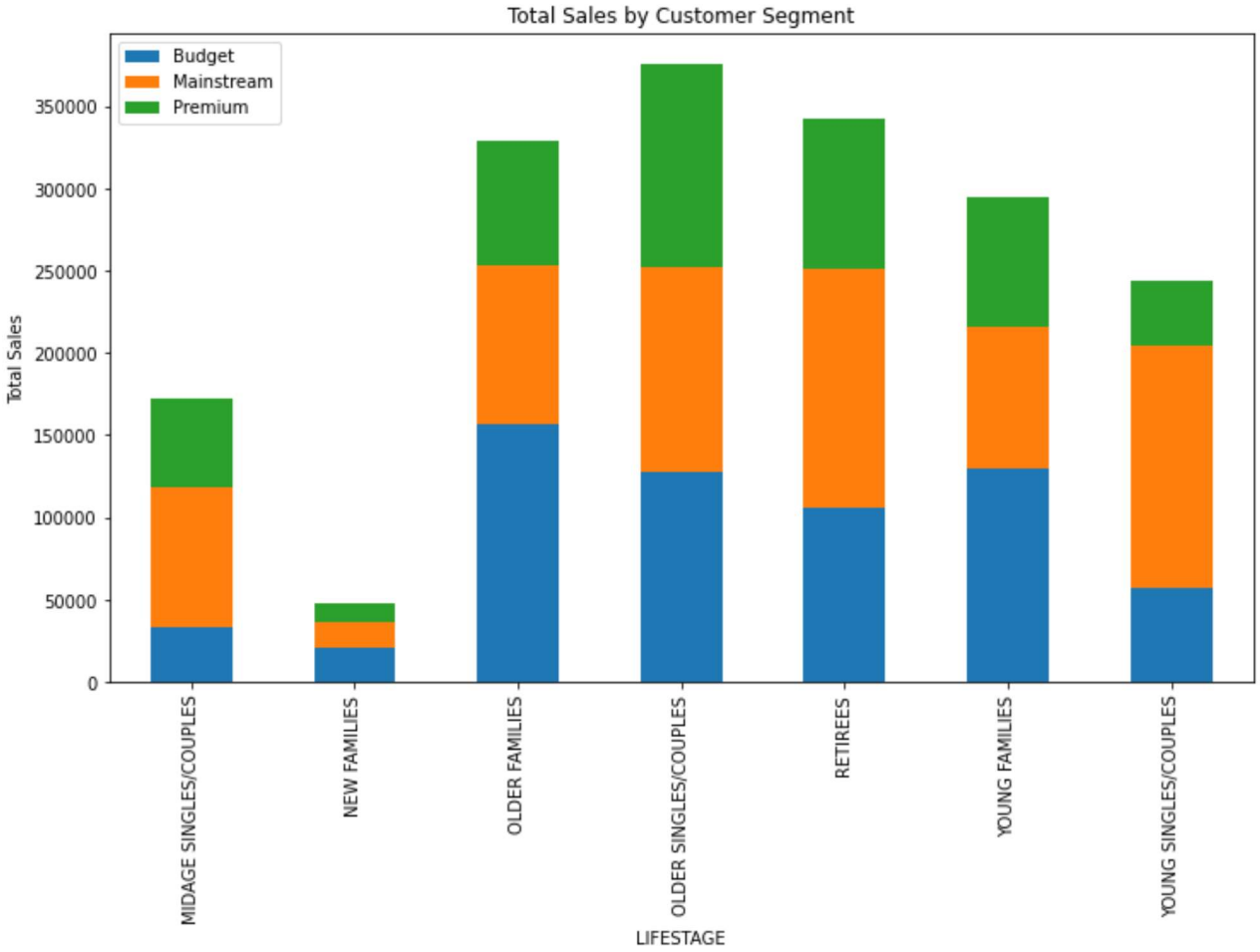
Out[83]:

		Total Sales
PREMIUM_CUSTOMER	**LIFESTAGE**	
Budget	**OLDER FAMILIES**	156863.75
Mainstream	**YOUNG SINGLES/COUPLES**	147582.20
	RETIREES	145168.95
Budget	**YOUNG FAMILIES**	129717.95
	OLDER SINGLES/COUPLES	127833.60
Mainstream	**OLDER SINGLES/COUPLES**	124648.50
Premium	**OLDER SINGLES/COUPLES**	123549.55
Budget	**RETIREES**	105916.30
Mainstream	**OLDER FAMILIES**	96413.55
Premium	**RETIREES**	91296.65
Mainstream	**YOUNG FAMILIES**	86338.25
	MIDAGE SINGLES/COUPLES	84734.25
Premium	**YOUNG FAMILIES**	78571.70
	OLDER FAMILIES	75242.60
Budget	**YOUNG SINGLES/COUPLES**	57122.10
Premium	**MIDAGE SINGLES/COUPLES**	54443.85
	YOUNG SINGLES/COUPLES	39052.30

		Total Sales
PREMIUM_CUSTOMER	**LIFESTAGE**	
Budget	**MIDAGE SINGLES/COUPLES**	33345.70
	NEW FAMILIES	20607.45
Mainstream	**NEW FAMILIES**	15979.70
Premium	**NEW FAMILIES**	10760.80

In [84]:
```
# Visualise

salesPlot = pd.DataFrame(combineData.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER']).TOT_SAL
salesPlot.unstack().plot(kind = 'bar', stacked = True, figsize = (12, 7), title = 'Tota
plt.ylabel('Total Sales')
plt.legend(['Budget', 'Mainstream', 'Premium'], loc = 2)
```

Out[84]: <matplotlib.legend.Legend at 0x22c13f75c18>



Top 3 sales come from budget older families, mainstream young singles/couples and mainstream retirees.

In [85]:
```
# Number of customers by PREMIUM_CUSTOMER and LIFESTAGE

customers = pd.DataFrame(combineData.groupby(['PREMIUM_CUSTOMER', 'LIFESTAGE']).LYLTY_C
```

```
customers.rename(columns = {'LYLTY_CARD_NBR': 'Number of Customers'}, inplace = True)
customers.sort_values(by = 'Number of Customers', ascending = False).head(10)
```

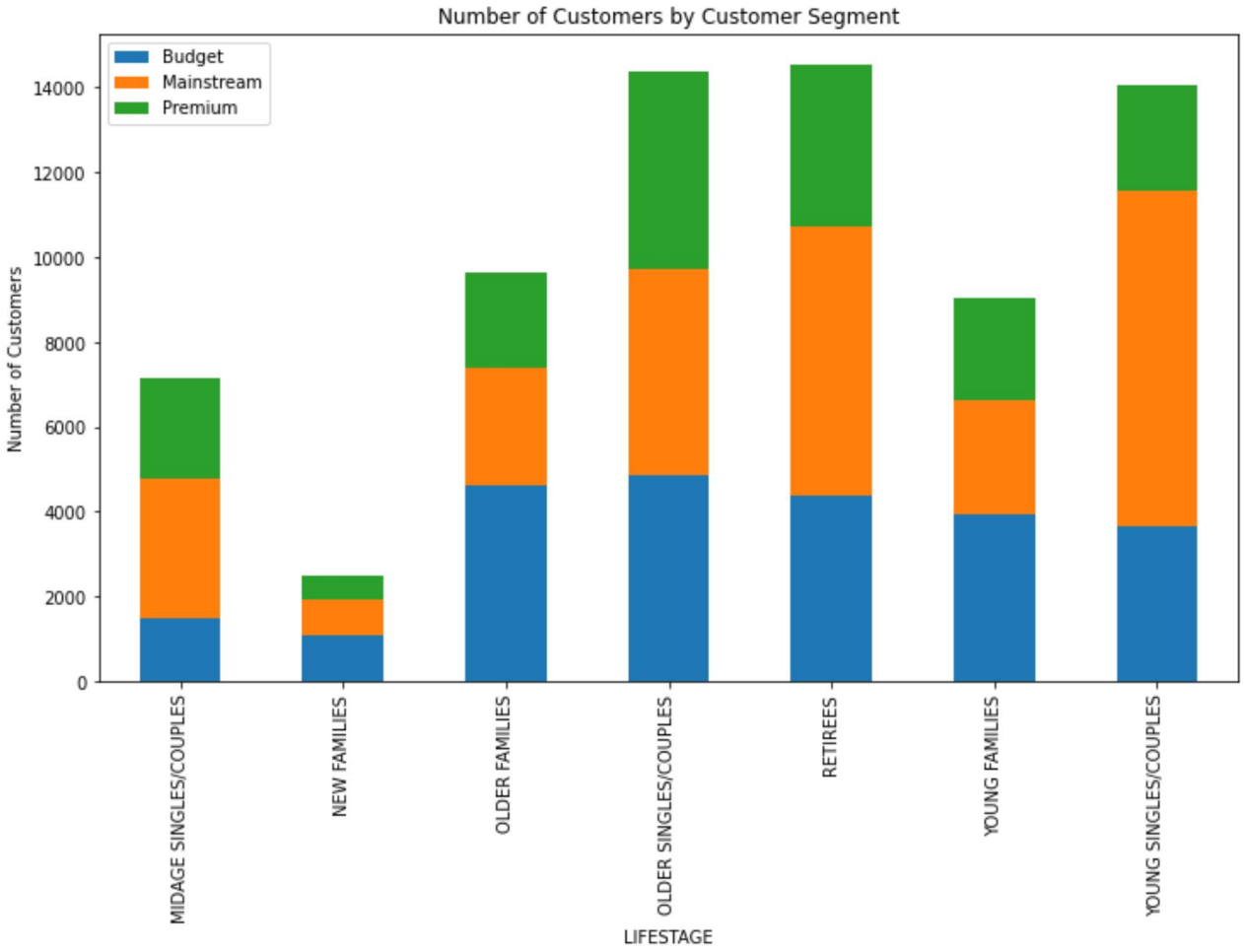
Out[85]:

		Number of Customers
PREMIUM_CUSTOMER	**LIFESTAGE**	
Mainstream	**YOUNG SINGLES/COUPLES**	7917
	RETIREES	6358
	OLDER SINGLES/COUPLES	4858
Budget	**OLDER SINGLES/COUPLES**	4849
Premium	**OLDER SINGLES/COUPLES**	4682
Budget	**OLDER FAMILIES**	4611
	RETIREES	4385
	YOUNG FAMILIES	3953
Premium	**RETIREES**	3812
Budget	**YOUNG SINGLES/COUPLES**	3647

In [86]:

```
# Visualise

customersPlot = pd.DataFrame(combineData.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER']).LYL
customersPlot.unstack().plot(kind = 'bar', stacked = True, figsize = (12, 7), title = '
plt.ylabel('Number of Customers')
plt.legend(['Budget', 'Mainstream', 'Premium'], loc = 2)
```

Out[86]: <matplotlib.legend.Legend at 0x22c13dba0b8>



There are more mainstream young singles/couples and retirees. This contributes to to more chips sales in these segments however this is not the major driver for the budget older families segment.

In [87]:
```
# Average units per customer by PREMIUM_CUSTOMER and LIFESTAGE

avg_units = combineData.groupby(['PREMIUM_CUSTOMER', 'LIFESTAGE']).PROD_QTY.sum() / com
avg_units = pd.DataFrame(avg_units, columns = {'Average Unit per Customer'})
avg_units.sort_values(by = 'Average Unit per Customer', ascending = False).head()
```

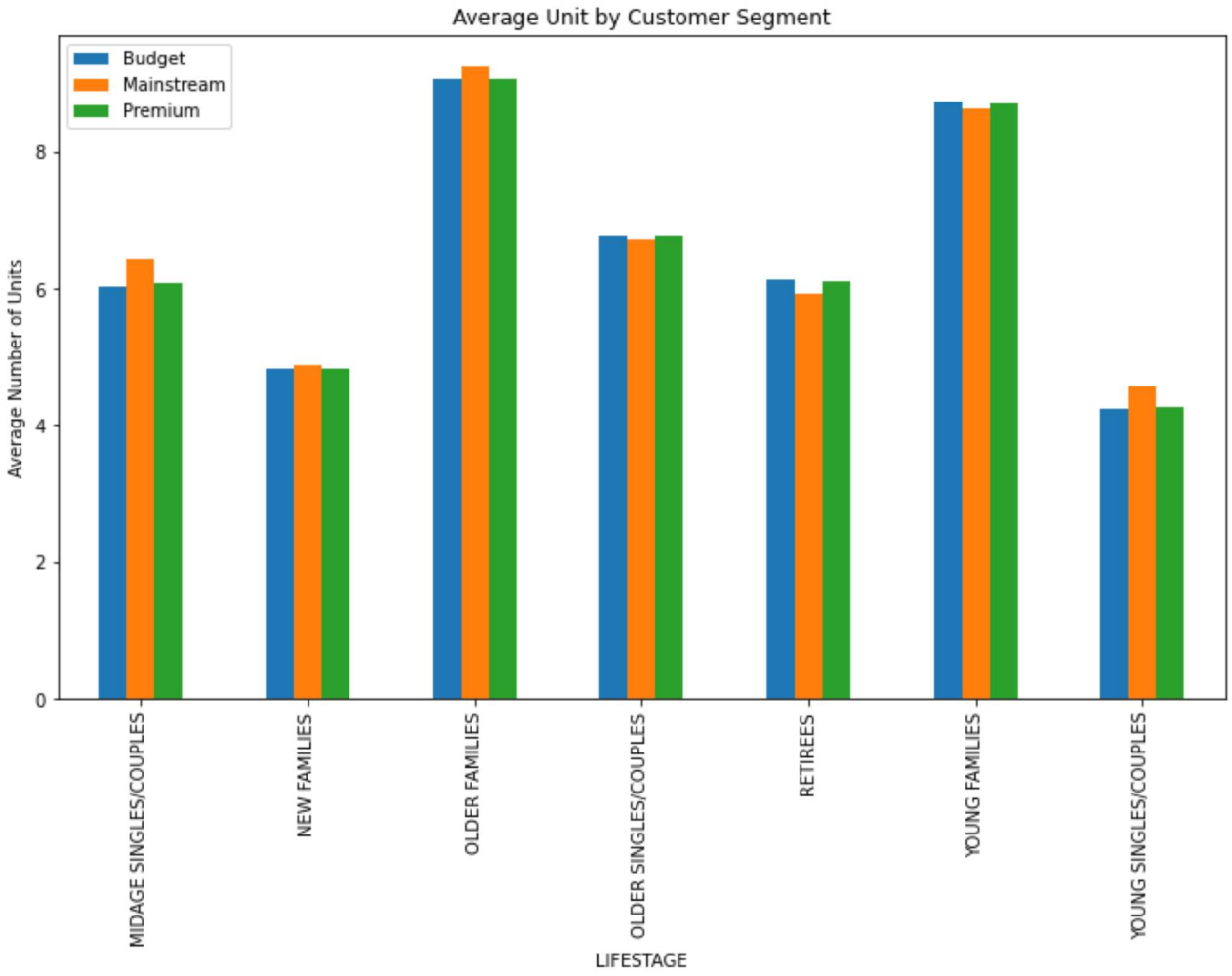
Out[87]:

		Average Unit per Customer
PREMIUM_CUSTOMER	**LIFESTAGE**	
Mainstream	**OLDER FAMILIES**	9.255380
Budget	**OLDER FAMILIES**	9.076773
Premium	**OLDER FAMILIES**	9.071717
Budget	**YOUNG FAMILIES**	8.722995
Premium	**YOUNG FAMILIES**	8.716013

In [88]:
```
# Visualise

avgUnitsPlot = pd.DataFrame(combineData.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER']).PROD
avgUnitsPlot.unstack().plot(kind = 'bar', figsize = (12, 7), title = 'Average Unit by C
```

```
plt.ylabel('Average Number of Units')
plt.legend(['Budget', 'Mainstream', 'Premium'], loc = 2)
```

Out[88]: <matplotlib.legend.Legend at 0x22c148c7748>



Older families and young families buy more chips per customer.

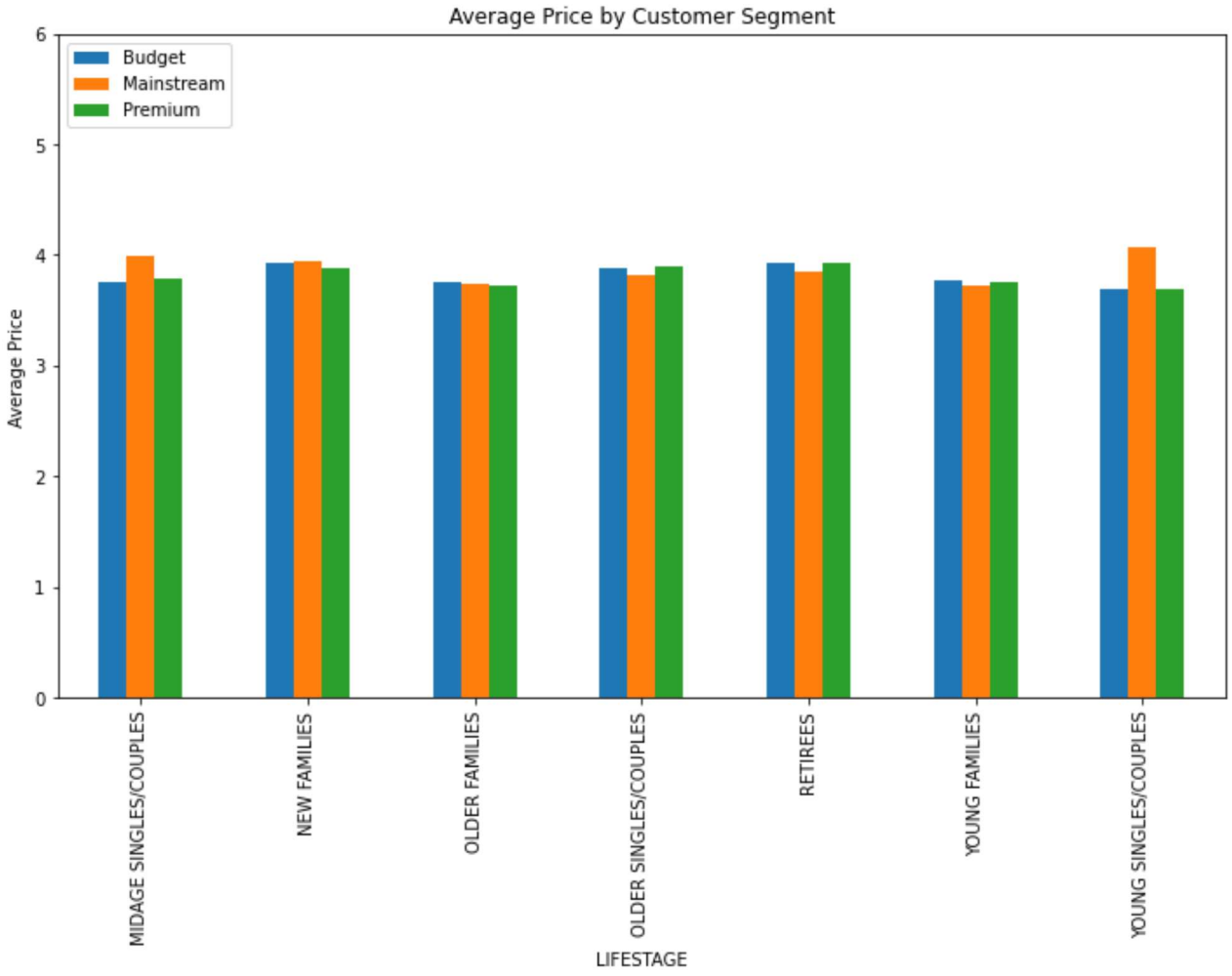
In [89]:
```
# Average price per unit by PREMIUM_CUSTOMER and LIFESTAGE

avg_price = combineData.groupby(['PREMIUM_CUSTOMER', 'LIFESTAGE']).TOT_SALES.sum() / co
avg_price = pd.DataFrame(avg_price, columns = {'Price per Unit'})
avg_price.sort_values(by = 'Price per Unit', ascending = False).head()
```

Out[89]:

		Price per Unit
PREMIUM_CUSTOMER	**LIFESTAGE**	
Mainstream	**YOUNG SINGLES/COUPLES**	4.074043
	MIDAGE SINGLES/COUPLES	3.994449
	NEW FAMILIES	3.935887
Budget	**RETIREES**	3.932731
	NEW FAMILIES	3.931969

In [90]:
```
# Visualise

avgPricePlot = pd.DataFrame(combineData.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER']).TOT_
avgPricePlot.unstack().plot(kind = 'bar', figsize = (12, 7), title = 'Average Price by
plt.ylabel('Average Price')
plt.legend(['Budget', 'Mainstream', 'Premium'], loc = 2)
```

Out[90]: <matplotlib.legend.Legend at 0x22c14978d68>



Mainstream midage and young singles and couples are more willing to pay more per packet of chips compared to their budget and premium counterparts. This may be due to premium shoppers being more likely to buy healthy snacks and when they do buy chips, it is mainly for entertainment purposes rather than their own consumption. This is also supported by there being fewer premium midage and young singles and couples buying chips compared to their mainstream counterparts.

In [91]:
```
# Perform an independent t-test between mainstream vs non-mainstream midage and young s

# Create a new dataframe pricePerUnit
pricePerUnit = combineData

# Create a new column under pricePerUnit called PRICE
pricePerUnit['PRICE'] = pricePerUnit['TOT_SALES'] / pricePerUnit['PROD_QTY']

# Let's have a look
pricePerUnit.head()
```

Out[91]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	P/
0	2018-10-17	1	1000	1	5	Natural Chip Compny Seasalt	2	6.0	
1	2019-05-14	1	1307	348	66	Ccs Nacho Cheese	3	6.3	
2	2019-05-20	1	1343	383	61	Smiths Crinkle Cut Chips Chicken	2	2.9	
3	2018-08-17	2	2373	974	69	Smiths Chip Thinly S Cream Onion	5	15.0	
4	2018-08-18	2	2426	1038	108	Kettle Tortilla Chpshny Jlpno Chili	3	13.8	

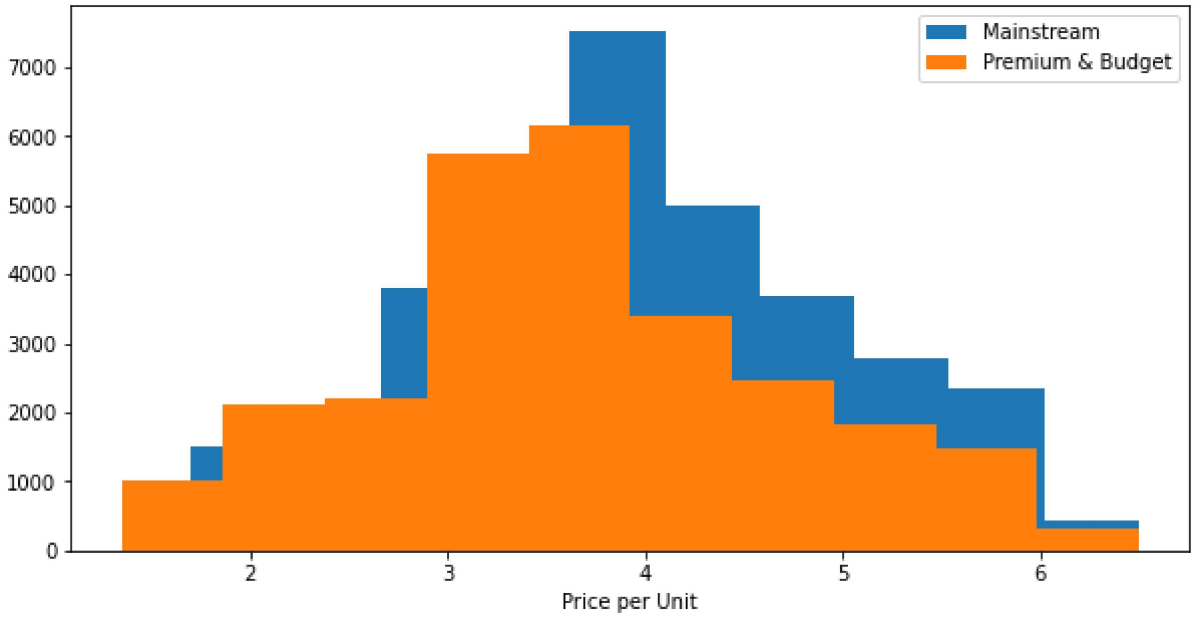
In [92]:

```
# Let's group our data into mainstream and non-mainstream

mainstream = pricePerUnit.loc[(pricePerUnit['PREMIUM_CUSTOMER'] == 'Mainstream') & ( (p
nonMainstream = pricePerUnit.loc[(pricePerUnit['PREMIUM_CUSTOMER'] != 'Mainstream') & (
```

In [93]:

```
# Compare histograms of mainstream and non-mainstream customers

plt.figure(figsize = (10, 5))
plt.hist(mainstream, label = 'Mainstream')
plt.hist(nonMainstream, label = 'Premium & Budget')
plt.legend()
plt.xlabel('Price per Unit')
```

Out[93]: Text(0.5, 0, 'Price per Unit')



In [94]:
```
print("Mainstream average price per unit: ${:.2f}".format(np.mean(mainstream)))
print("Non-mainstream average price per unit: ${:.2f}".format(np.mean(nonMainstream)))
if np.mean(mainstream) > np.mean(nonMainstream):
    print("Mainstream customers have higher average price per unit. ")
else:
    print("Non-mainstream customers have a higher average price per unit. ")
```

```
Mainstream average price per unit: $4.04
Non-mainstream average price per unit: $3.71
Mainstream customers have higher average price per unit.
```

In [95]:
```
# Perform t-test

ttest_ind(mainstream, nonMainstream)
```

Out[95]:
```
Ttest_indResult(statistic=37.83196107667815, pvalue=2.235645611549355e-309)
```

Mainstream customers have higher average price per unit than that of non-mainstream customers.

We have found quite a few interesting insights that we can dive deeper into. For example, we might want to target customers segments that contribute the most to sales to retain them to further increase sales. Let's examine mainstream young singles/couples against the rest of the cutomer segments to see if they prefer any particular brand of chips.

In [96]:
```
target = combineData.loc[(combineData['LIFESTAGE'] == 'YOUNG SINGLES/COUPLES') & (combi
nonTarget = combineData.loc[(combineData['LIFESTAGE'] != 'YOUNG SINGLES/COUPLES' ) & (c
target.head()
```

Out[96]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SAL
221347	2018-08-16	1	1020	26	19	Smiths Crinkle Cut Snag Sauce	1	2
221348	2018-08-17	1	1163	188	46	Kettle Original	1	5
221349	2018-08-14	1	1291	333	27	Ww Supreme Cheese Corn Chips	1	
221350	2019-05-15	3	3031	1227	14	Smiths Crnkle Chip Orgnl Big Bag	1	5
221351	2019-05-18	3	3118	1574	62	Pringles Mystery Flavour	1	3

Affinity to brand

In [97]:
```
# Target Segment
targetBrand = target.loc[:, ['BRAND', 'PROD_QTY']]
```

```
targetSum = targetBrand['PROD_QTY'].sum()
targetBrand['Target Brand Affinity'] = targetBrand['PROD_QTY'] / targetSum
targetBrand = pd.DataFrame(targetBrand.groupby('BRAND')['Target Brand Affinity'].sum())

# Non-target segment
nonTargetBrand = nonTarget.loc[:, ['BRAND', 'PROD_QTY']]
nonTargetSum = nonTargetBrand['PROD_QTY'].sum()
nonTargetBrand['Non-Target Brand Affinity'] = nonTargetBrand['PROD_QTY'] / nonTargetSum
nonTargetBrand = pd.DataFrame(nonTargetBrand.groupby('BRAND')['Non-Target Brand Affinit
```

In [99]:

```
# Merge the two dataframes together

brand_proportions = pd.merge(targetBrand, nonTargetBrand, left_index = True, right_inde
brand_proportions.head()
```

Out[99]:

BRAND	Target Brand Affinity	Non-Target Brand Affinity
Burger	0.002926	0.006538
CCS	0.011180	0.018444
Cheetos	0.008033	0.011759
Cheezels	0.017971	0.018904
Cobs	0.044638	0.038448

In [100…

```
brand_proportions['Affinity to Brand'] = brand_proportions['Target Brand Affinity'] / b
brand_proportions.sort_values(by = 'Affinity to Brand', ascending = False)
```

Out[100…

BRAND	Target Brand Affinity	Non-Target Brand Affinity	Affinity to Brand
Tyrrells	0.031553	0.025714	1.227044
Twisties	0.046184	0.037932	1.217524
Doritos	0.122761	0.101169	1.213415
Kettle	0.197985	0.166558	1.188685
Tostitos	0.045411	0.038350	1.184118
Pringles	0.119420	0.101109	1.181108
Cobs	0.044638	0.038448	1.160976
Infuzions	0.064679	0.057409	1.126639
Thins	0.060373	0.057158	1.056233
Grainwaves	0.032712	0.031068	1.052911
Cheezels	0.017971	0.018904	0.950667
Smiths	0.096370	0.124232	0.775728
French	0.003948	0.005707	0.691735

BRAND	Target Brand Affinity	Non-Target Brand Affinity	Affinity to Brand
Cheetos	0.008033	0.011759	0.683160
Red Rock Deli	0.043810	0.067183	0.652090
Natural	0.019600	0.030958	0.633101
CCS	0.011180	0.018444	0.606151
Sunbites	0.006349	0.012613	0.503406
Woolworths	0.024099	0.048746	0.494383
Burger	0.002926	0.006538	0.447581

Mainstream young singles/couples are more likely to purchase Tyrrells chips compared to other brands.

Affinity to pack size

In [101…

```
# Target segment
targetSize = target.loc[:, ['PACK_SIZE', 'PROD_QTY']]
targetSum = targetSize['PROD_QTY'].sum()
targetSize['Target Pack Affinity'] = targetSize['PROD_QTY'] / targetSum
targetSize = pd.DataFrame(targetSize.groupby('PACK_SIZE')['Target Pack Affinity'].sum()

# Non-target segment
nonTargetSize = nonTarget.loc[:, ['PACK_SIZE', 'PROD_QTY']]
nonTargetSum = nonTargetSize['PROD_QTY'].sum()
nonTargetSize['Non-Target Pack Affinity'] = nonTargetSize['PROD_QTY'] / nonTargetSum
nonTargetSize = pd.DataFrame(nonTargetSize.groupby('PACK_SIZE')['Non-Target Pack Affini
```

In [102…

```
# Merge the two dataframes together

pack_proportions = pd.merge(targetSize, nonTargetSize, left_index = True, right_index =
pack_proportions.head()
```

Out[102…

PACK_SIZE	Target Pack Affinity	Non-Target Pack Affinity
70	0.003037	0.006283
90	0.006349	0.012613
110	0.106280	0.089574
125	0.003009	0.005976
134	0.119420	0.101109

In [103…

```
pack_proportions['Affinity to Pack'] = pack_proportions['Target Pack Affinity'] / pack_
pack_proportions.sort_values(by = 'Affinity to Pack', ascending = False)
```

Out[103…

PACK_SIZE	Target Pack Affinity	Non-Target Pack Affinity	Affinity to Pack
270	0.031829	0.025069	1.269628
380	0.032160	0.025711	1.250846
330	0.061284	0.050974	1.202262
110	0.106280	0.089574	1.186510
134	0.119420	0.101109	1.181108
210	0.029124	0.024891	1.170038
135	0.014769	0.012931	1.142104
250	0.014355	0.012863	1.115976
170	0.080773	0.080346	1.005315
150	0.157598	0.163067	0.966463
175	0.254990	0.271469	0.939297
165	0.055652	0.061587	0.903642
190	0.007481	0.012131	0.616706
180	0.003589	0.006177	0.580952
160	0.006404	0.012222	0.524021
125	0.003009	0.005976	0.503497
90	0.006349	0.012613	0.503406
200	0.008972	0.018471	0.485718
70	0.003037	0.006283	0.483263
220	0.002926	0.006538	0.447581

It looks like mainstream singles/couples are more likely to purchase a 270g pack size compared to other pack sizes.

In [104…

```
# Which brand offers 270g pack size?

combineData.loc[combineData['PACK_SIZE'] == 270, :].head(10)
```

Out[104…

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES
11	2019-05-18	9	9208	8634	15	Twisties Cheese	2	9.2
39	2019-05-20	55	55073	48887	113	Twisties Chicken	1	4.6
64	2019-05-20	88	88320	87811	113	Twisties Chicken	2	9.2
96	2018-08-20	149	149317	149044	15	Twisties Cheese	1	4.6

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES
102	2018-08-15	153	153220	152829	113	Twisties Chicken	2	9.2
121	2018-08-18	173	173154	174551	15	Twisties Cheese	1	4.6
150	2018-08-17	217	217168	217061	113	Twisties Chicken	1	4.6
151	2019-05-18	219	219065	218396	15	Twisties Cheese	1	4.6
204	2019-04-21	1	1491	577	113	Twisties Chicken	1	4.6
252	2019-06-30	4	4106	3138	113	Twisties Chicken	2	9.2

In [105…

```
# Is Twisties the only brand who sells 270g pack size?

combineData.loc[combineData['PACK_SIZE'] == 270, 'BRAND'].unique()
```

Out[105…

```
array(['Twisties'], dtype=object)
```

Twisties is the only brand that offers 270g pack size.

Conclusion

Sales are highest for (Budget, OLDER FAMILIES), (Mainstream, YOUNG SINGLES/COUPLES) and (Mainstream, RETIREES) in which (Mainstream, YOUNG SINGLES/COUPLES) and (Mainstream, RETIREES) are in major part due to the more customers in these category. (Mainstream, YOUNG SINGLES/COUPLES) are more likely to pay more per packet of chips than their premium and budget cotegory. They are most favorable to purchase 'Tyrrells' and '270g' pack sizes than the rest of the population