Task 1: Data Preparation and Customer Analytics.

The Virtual Experience Content Was offered via Forage.

1 Problem Identification, Business Case Evaluation.

Problem Statement.

> The retail client wants to understand the behaviour of customers that purchase chips within a region and form a holistic view of the customer base. Additionally the insights will be utilised for strategy formation and decision making.

Other Details

> Industry: Retail | Audience: Management Team (Category Manager);

1.2 Approach to the scenario presented.

0. Analyse the data using know-how.
1. Compelete the solution on own and learn on the go.
2. After Compeleting and uploading solution (on forage and code to github)
    - A. Compare with solutions.
    - B. How others approached the problem.
    - C. Ideas and improvements for next project/takeaways.

1.3 Environment Setup.

- Libraries/packages
- Helper Functions

In [19]:
```python
#Data packages/libraries.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as ex
from plotly.offline import download_plotlyjs, init_notebook_mode, p
init_notebook_mode(connected=True)
from mlxtend.frequent_patterns import apriori

# Python Standard Libraries. (based on the project)
import re
import pprint
import collections
import multiprocessing
import math
from pathlib import Path
import webbrowser
import time
import scipy
import warnings


# For Easier Readability.
print_mod = lambda *x: print('\n',*x)
from typing import Union
pp = pprint.PrettyPrinter(indent=4)


# Options for libraries
# Matplotlib style

# Pandas Dataframe Style.
pd.set_option('display.max_columns',None)
warnings.filterwarnings('ignore')

# When running all cells, make a function/magic that toggles the ab
# To quicken time of all cell runnings.
# Woking with a smaller dataset, at the end working with a large da
```

In [20]:
```
### Helper Functions and classes.

# Generic
def col_value_name_change(data_frame,col_name,curr_value_name,new_v
    index = data_frame[data_frame[col_name] == curr_value_name].ind
    temp_df = data_frame[data_frame[col_name] == curr_value_name][:
    if inplace:
        data_frame.loc[index,col_name] = new_value_name
    if show_curr:
        print('-- Curr -- ')
        print(temp_df.head(n_rows))
    if show_old_new:
        print('-- Old -- ')
        print(temp_df.head(n_rows))
        print('-- New -- ')
        temp_df.loc[index,col_name] =  new_value_name
        print(temp_df.head(n_rows))

def col_stats(data_frame,col_name):
    #mean, percentiles, median('50th percentile'), mode, most frequ
    pass

# Custom to the project at hand.
```

2 Data Wrangling

2.1 Data collection

In [21]:
```
# The dataset is provided, no need to collect, extract or use other
import os

class dataset_local():
    def __init__(self,name,extension,path=''):
        self._name = name
        self._extension = extension
        self._path = path

    @property
    def extension(self):
        return self._extension

    @property
    def name(self):
        return self._name

    @property
    def path(self):
        return self._path

    @extension.setter
    def extension(self, new_value):
        self._extension = new_value

    @name.setter
    def name(self, new_value):
        self._name = new_value

    @path.setter
    def path(self, new_value):
        self._path = new_value

    def full_name_to_dataset(self):
        return f''

    def __str__(self):
        path_name = os.path.expandvars(os.path.join(self._path, self
        return f'{path_name}.{self._extension}'

    __repr__ = __str__

dir_path = '.'
qvi_customer_behaviour_dataset = dataset_local('QVI_purchase_behavio
qvi_customer_transaction_dataset = dataset_local('QVI_transaction_da
cust_behv_df = pd.read_csv(str(qvi_customer_behaviour_dataset))
cust_tran_df = pd.read_excel(str(qvi_customer_transaction_dataset))
```

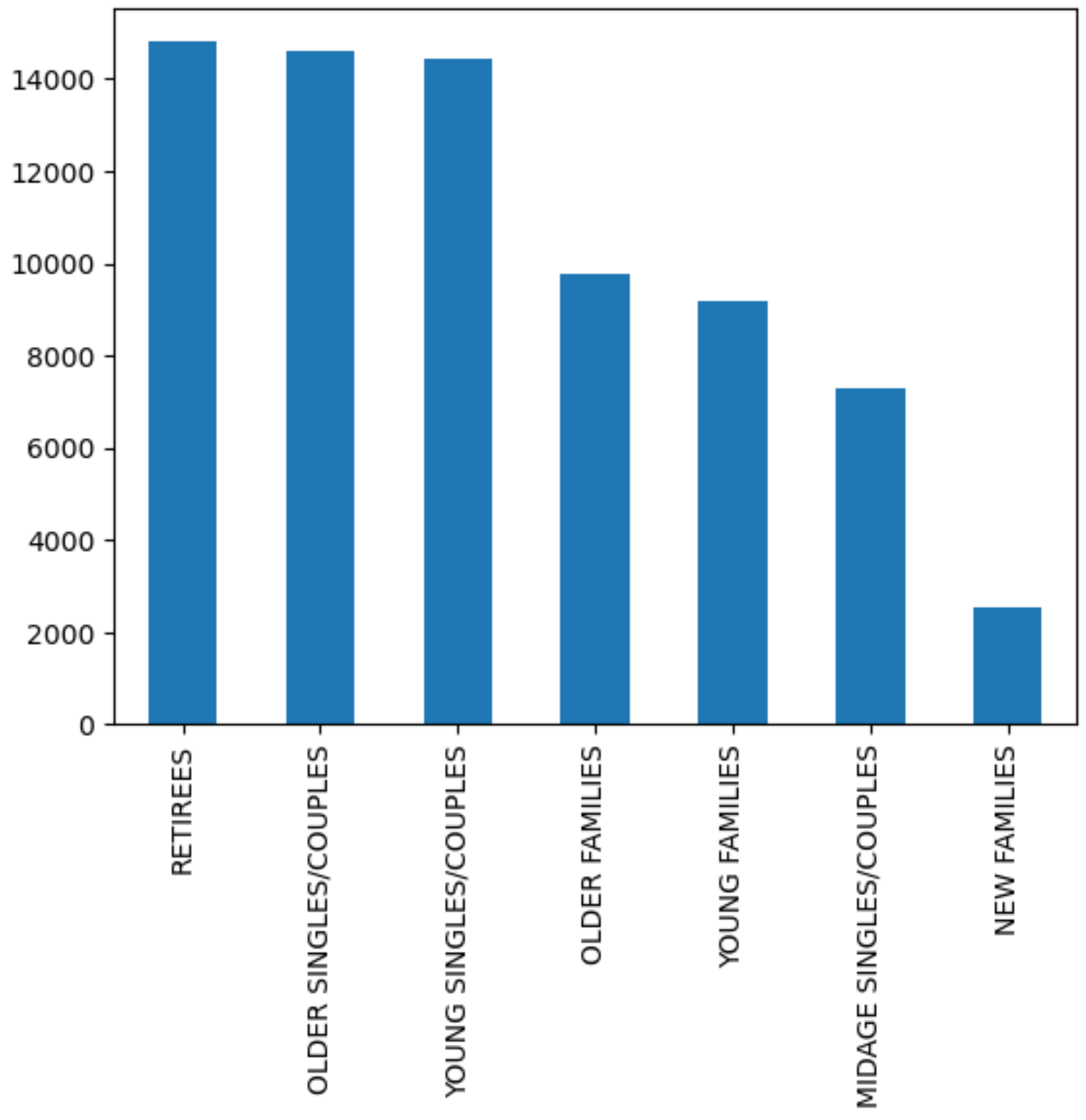
2.2 Data Exploration

```
In [22]: # Dataset 1 information
         print_mod(cust_behv_df.info())
         print_mod(cust_behv_df.dtypes)
         print_mod(cust_behv_df.head())
         [print(cust_behv_df[col_name].describe()) for col_name in cust_behv
         print_mod(cust_behv_df['LIFESTAGE'].value_counts())
         print_mod(cust_behv_df['PREMIUM_CUSTOMER'].value_counts())
         print_mod(cust_behv_df['LYLTY_CARD_NBR'].value_counts())
         print_mod(cust_behv_df['LYLTY_CARD_NBR'].value_counts()[cust_behv_d
         print_mod(cust_behv_df.groupby(['PREMIUM_CUSTOMER','LIFESTAGE']).co
         print_mod(cust_behv_df)
```

```
Name: LYLTY_CARD_NBR, Length: 72637, dtype: int64

 Series([], Name: LYLTY_CARD_NBR, dtype: int64)

                        LYLTY_CARD_NBR
PREMIUM_CUSTOMER               Budget Mainstream Premium
LIFESTAGE
MIDAGE SINGLES/COUPLES           1504       3340    2431
NEW FAMILIES                     1112        849     588
OLDER FAMILIES                   4675       2831    2274
OLDER SINGLES/COUPLES            4929       4930    4750
RETIREES                         4454       6479    3872
YOUNG FAMILIES                   4017       2728    2433
YOUNG SINGLES/COUPLES            3779       8088    2574

        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
```

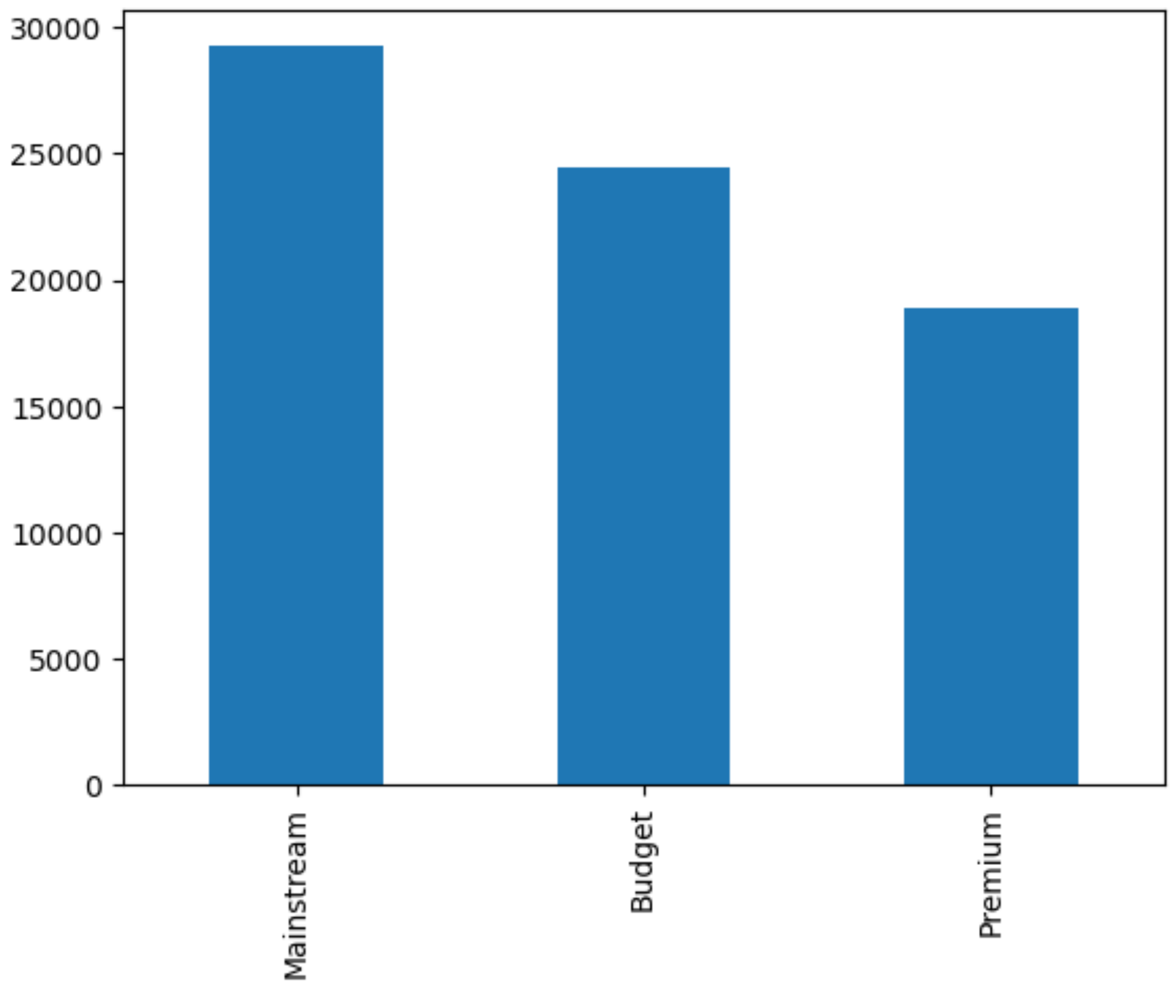
```
In [23]: # Dataset 1 exploratory visualisation.
         print_mod(cust_behv_df['LIFESTAGE'].value_counts().plot(kind='bar'))
```

```
AxesSubplot(0.125,0.11;0.775x0.77)
```



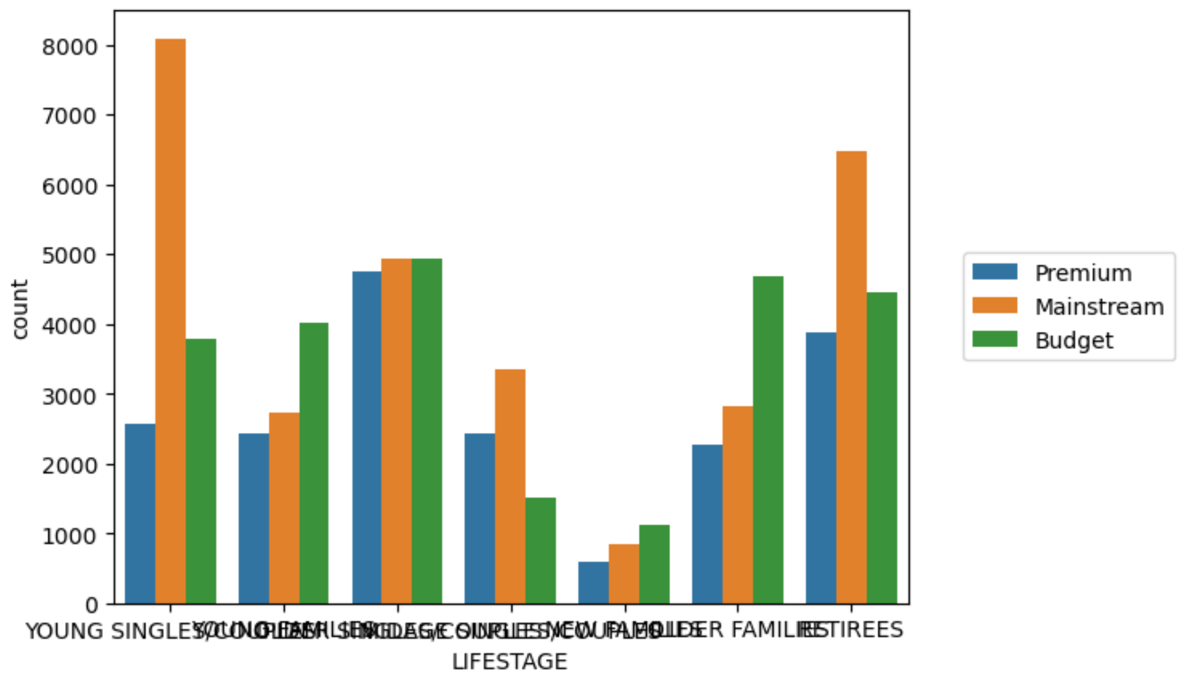
```
In [24]: print_mod(cust_behv_df['PREMIUM_CUSTOMER'].value_counts().plot(kind
```

```
AxesSubplot(0.125,0.11;0.775x0.77)
```



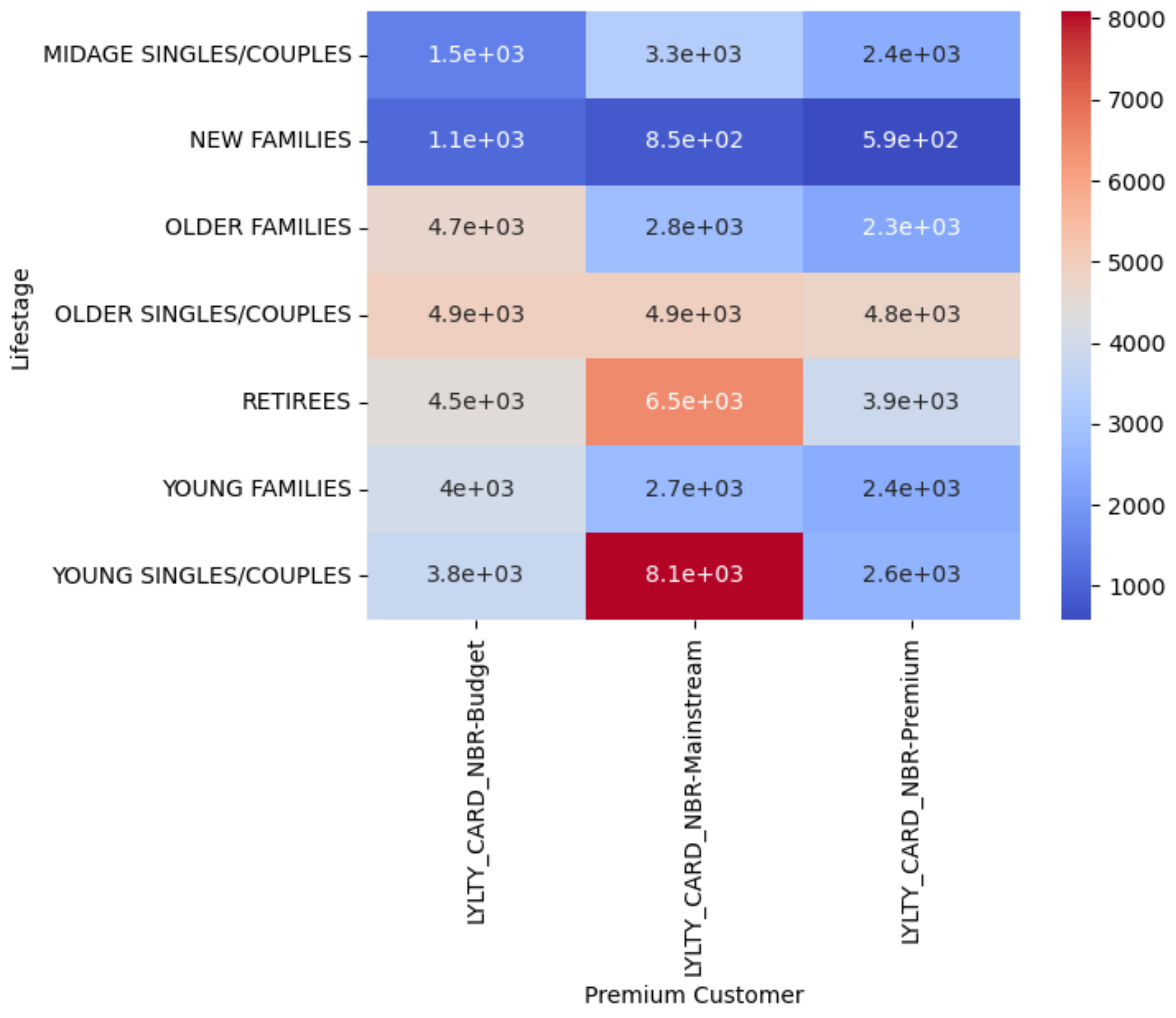
```
In [25]: sns.countplot(x=cust_behv_df['LIFESTAGE'],hue=cust_behv_df['PREMIUM
```

```
Out[25]: <matplotlib.legend.Legend at 0x7f7edb282f10>
```



```
In [26]: sns.heatmap(cust_behv_df.groupby(['PREMIUM_CUSTOMER','LIFESTAGE']).
         plt.ylabel('Lifestage')
         plt.xlabel('Premium Customer')
```

```
Out[26]: Text(0.5, 23.38159722222222, 'Premium Customer')
```



In [27]:
```
cust_tran_df.tail()
```
Out[27]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD
264831	43533	272	272319	270088	89	Kettle Sweet Chilli And Sour Cream 175g	
264832	43325	272	272358	270154	74	Tostitos Splash Of Lime 175g	
264833	43410	272	272379	270187	51	Doritos Mexicana 170g	
264834	43461	272	272379	270188	42	Doritos Corn Chip Mexican Jalapeno 150g	
264835	43365	272	272380	270189	74	Tostitos Splash Of Lime 175g	

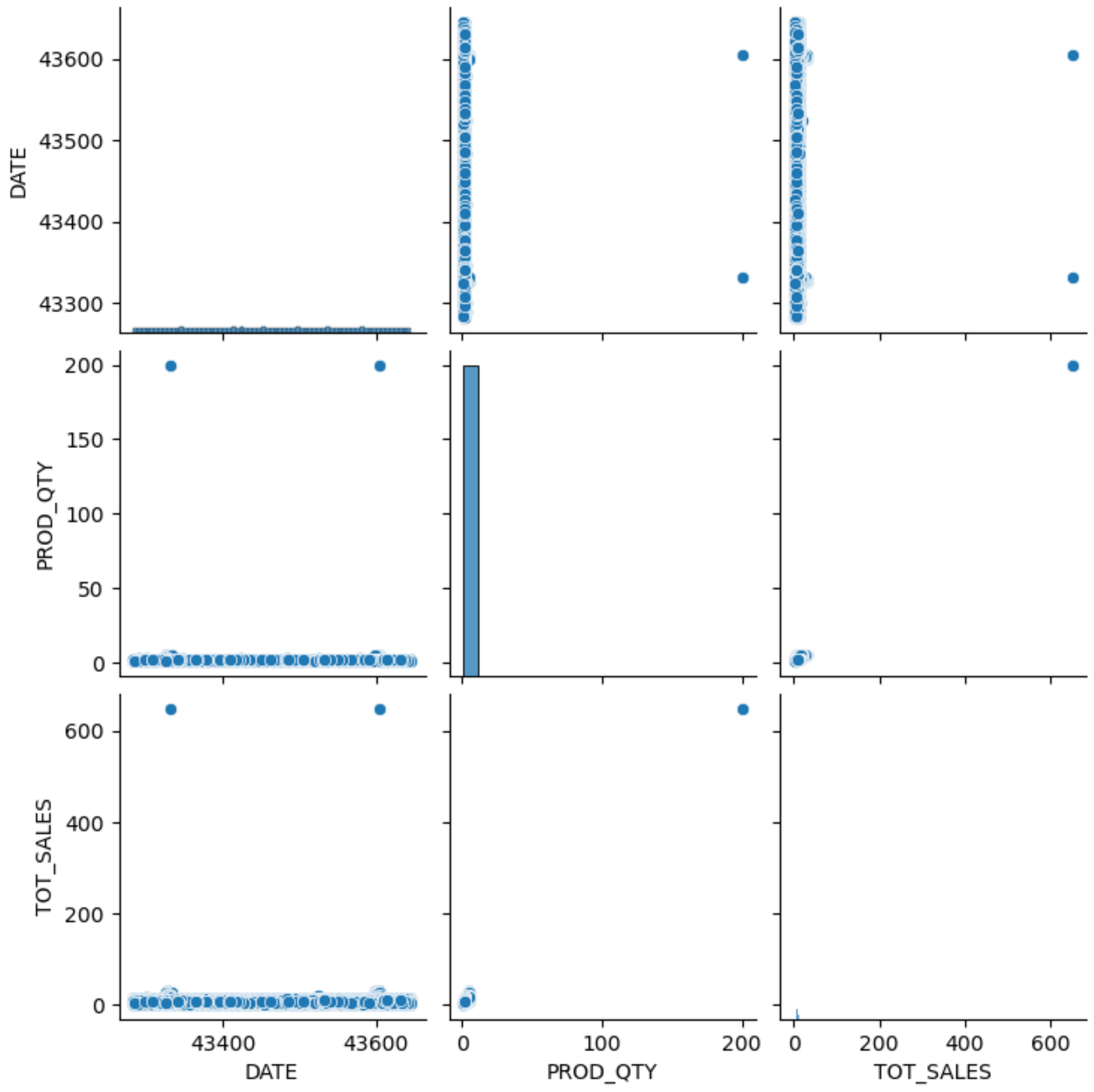
In [28]:
```
# Dataset 2
cust_tran_df.info()
cust_tran_df.dtypes
cust_tran_df.head()
cust_tran_df.describe()
```

```
<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
```

Out[28]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	P
count	264836.000000	264836.00000	2.648360e+05	2.648360e+05	264836.000000	2648
mean	43464.036260	135.08011	1.355495e+05	1.351583e+05	56.583157	
std	105.389282	76.78418	8.057998e+04	7.813303e+04	32.826638	
min	43282.000000	1.00000	1.000000e+03	1.000000e+00	1.000000	
25%	43373.000000	70.00000	7.002100e+04	6.760150e+04	28.000000	
50%	43464.000000	130.00000	1.303575e+05	1.351375e+05	56.000000	
75%	43555.000000	203.00000	2.030942e+05	2.027012e+05	85.000000	
max	43646.000000	272.00000	2.373711e+06	2.415841e+06	114.000000	2

```
In [29]: sns.pairplot(cust_tran_df[['DATE', 'PROD_QTY', 'TOT_SALES']])

Out[29]: <seaborn.axisgrid.PairGrid at 0x7f7ede99c580>
```



2.4 Data Cleaning.

```
In [30]: # Column Renaming;  Dataset 1
         print_mod(cust_behv_df.head())
         cust_behv_df.rename(columns={'LYLTY_CARD_NBR':'Loyalty_Card_Number'
                                      'LIFESTAGE':'Life_Stage',
                                      'PREMIUM_CUSTOMER':'Card_Subscription'}
         print_mod(cust_behv_df.head())
```

```
    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

    Loyalty_Card_Number              Life_Stage Card_Subscription
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 [31]: # Column Renaming;  Dataset 2
         print_mod(cust_tran_df.head())
         cust_tran_df.rename(columns={'DATE': 'Date',
                                      'STORE_NBR': 'Store_Number',
                                      'LYLTY_CARD_NBR': 'Loyalty_Card_Number
                                      'TXN_ID': 'Taxation_Id',
                                      'PROD_NAME':'Product_Name',
                                      'PROD_NBR': 'Product_Number',
                                      'PROD_QTY': 'Product_Quantity',
                                      'TOT_SALES': 'Total_Sales'},inplace=Tr
         print_mod(cust_tran_df.head())
```

```
    DATE  STORE_NBR  LYLTY_CARD_NBR  TXN_ID  PROD_NBR  \
0  43390          1            1000       1         5
1  43599          1            1307     348        66
2  43605          1            1343     383        61
3  43329          2            2373     974        69
4  43330          2            2426    1038       108

                                  PROD_NAME  PROD_QTY  TOT_SALES
0    Natural Chip        Compny SeaSalt175g         2        6.0
1                  CCs Nacho Cheese    175g         3        6.3
2    Smiths Crinkle Cut  Chips Chicken 170g         2        2.9
3    Smiths Chip Thinly  S/Cream&Onion 175g         5       15.0
4  Kettle Tortilla ChpsHny&Jlpno Chili 150g         3       13.8

    Date  Store_Number  Loyalty_Card_Number  Taxation_Id  Product
_Number  \
0  43390             1                 1000            1
5
1  43599             1                 1307          348
66
2  43605             1                 1343          383
61
3  43329             2                 2373          974
69
4  43330             2                 2426         1038
108

                               Product_Name  Product_Quantity  Tot
al_Sales
0    Natural Chip        Compny SeaSalt175g                 2
6.0
1                  CCs Nacho Cheese    175g                 3
6.3
2    Smiths Crinkle Cut  Chips Chicken 170g                 2
2.9
3    Smiths Chip Thinly  S/Cream&Onion 175g                 5
15.0
4  Kettle Tortilla ChpsHny&Jlpno Chili 150g                 3
13.8
```

In [32]:

```



# Column Creation; Dataset 1 # Not needed.
```

In [33]:

```
# Column Creation; Dataset 2
print_mod(cust_tran_df.sample(3)) # Product Name has weight specifi
# Note numbers can be in the product name: 'Smith 1st original anti
# Notice positioning of weight is at end;
# Hence, if numeric input is at end, large likelihood it's weight.
# However, in a more untidy dataset multiple weights could have bee
# So that assumption/scenarios too needs to be handeled, generally.
# Additionally approach must take weight measurement metric into ac
```

```
          Date  Store_Number  Loyalty_Card_Number  Taxation_Id  Pr
oduct_Number  \
62024   43367           118               118106       121295
18
231034  43590            67                67168        64820
90
247803  43589           124               124415       128159
15

                             Product_Name  Product_Quantity  Total
_Sales
62024      Cheetos Chs & Bacon Balls 190g                 2
6.6
231034  Tostitos Smoked     Chipotle 175g                 2
8.8
247803           Twisties Cheese     270g                 1
4.6
```

In [34]:
```
# Observing that product price can be calculated by total_sales/pro
cust_tran_df['Product_Price'] = cust_tran_df['Total_Sales'] / cust_
cust_tran_df.head()
```

Out[34]:

	Date	Store_Number	Loyalty_Card_Number	Taxation_Id	Product_Number	Product_Nam
0	43390	1	1000	1	5	Natural Ch Compr SeaSalt175
1	43599	1	1307	348	66	CCs Nach Cheese 175
2	43605	1	1343	383	61	Smiths Crink Cut Chip Chicken 170
3	43329	2	2373	974	69	Smiths Ch Thin S/Cream&Onic 175
4	43330	2	2426	1038	108	Kettle Tortil ChpsHny&Jlpr Chili 150

In [35]:
```

# The weights have been extracted; For futher Processing.
cust_tran_df['Weight_Extraction'] = cust_tran_df['Product_Name'].ap
cust_tran_df.head()
```

Out[35]:

	Date	Store_Number	Loyalty_Card_Number	Taxation_Id	Product_Number	Product_Nam
0	43390	1	1000	1	5	Natural Ch Compr SeaSalt175
1	43599	1	1307	348	66	CCs Nach Cheese 175
2	43605	1	1343	383	61	Smiths Crink Cut Chip Chicken 170
3	43329	2	2373	974	69	Smiths Ch Thin S/Cream&Onic 175
4	43330	2	2426	1038	108	Kettle Tortil ChpsHny&Jlpr Chili 150

In [36]:
```
# All weight measures are in grams.
weight_measurement_metrics = cust_tran_df['Weight_Extraction'].appl
print(weight_measurement_metrics)
```

```
['g']
```

In [37]:
```
# Ensuring a single cell contains a single value. Measurment is com
cust_tran_df['Product_Weight_Grams'] = cust_tran_df['Product_Name']
cust_tran_df.sample(5)
```

Out[37]:

	Date	Store_Number	Loyalty_Card_Number	Taxation_Id	Product_Number	Product
259381	43518	24	24056	20592	95	S Whlegrn Frch/C
247913	43614	125	125339	129650	109	I Barbequ
73106	43591	84	84299	83988	63	Kett Swt
67224	43389	223	223250	224438	112	Tyrrells Ched &
241438	43291	37	37236	33785	44	Thin Light

In [38]:
```
# Delete uneeded intermediate column: weight extraction.
cust_tran_df.drop(columns=['Weight_Extraction'],axis=1,inplace=True
cust_tran_df.head()
```

Out[38]:

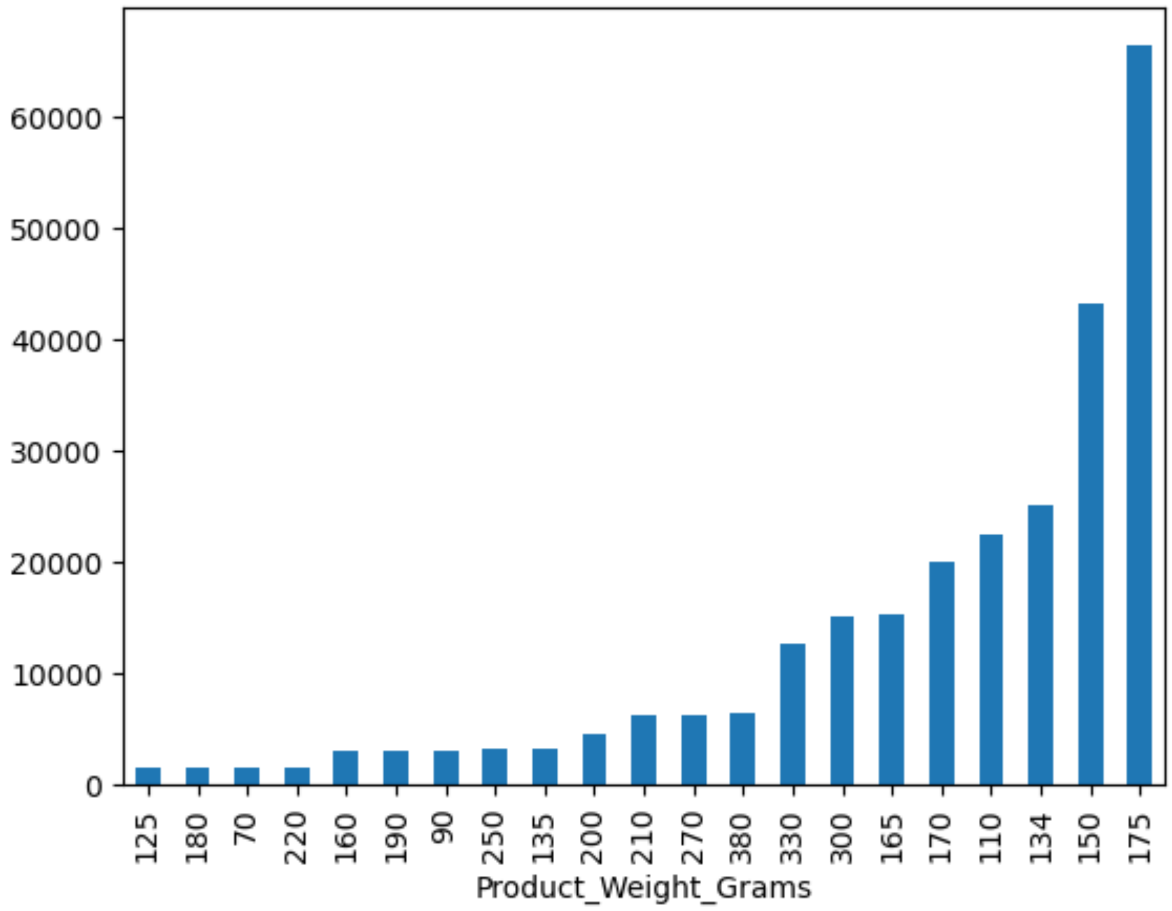
	Date	Store_Number	Loyalty_Card_Number	Taxation_Id	Product_Number	Product_Nam
0	43390	1	1000	1	5	Natural Ch Compr SeaSalt175
1	43599	1	1307	348	66	CCs Nach Cheese 175
2	43605	1	1343	383	61	Smiths Crink Cut Chip Chicken 170
3	43329	2	2373	974	69	Smiths Ch Thin S/Cream&Onio 175
4	43330	2	2426	1038	108	Kettle Tortil ChpsHny&Jlpr Chili 150

```
In [39]: # Remove weight from Product Weights from product name.
         print(cust_tran_df['Product_Name'])
         cust_tran_df['Product_Name'] = cust_tran_df['Product_Name'].apply(l
         print(cust_tran_df['Product_Name'])
```

```
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
                              ...
264831     Kettle Sweet Chilli And Sour Cream 175g
264832               Tostitos Splash Of  Lime 175g
264833                    Doritos Mexicana    170g
264834     Doritos Corn Chip Mexican Jalapeno 150g
264835               Tostitos Splash Of  Lime 175g
Name: Product_Name, Length: 264836, dtype: object
0                 Natural Chip Compny Seasalt
1                            Ccs Nacho Cheese
2            Smiths Crinkle Cut Chips Chicken
3            Smiths Chip Thinly S/Cream&Onion
4         Kettle Tortilla Chpshny&Jlpno Chili
                           ...
264831     Kettle Sweet Chilli And Sour Cream
264832                Tostitos Splash Of Lime
264833                       Doritos Mexicana
264834     Doritos Corn Chip Mexican Jalapeno
264835                Tostitos Splash Of Lime
Name: Product_Name, Length: 264836, dtype: object
```

```
In [40]: # New Column exploratory data analysis.
         cust_tran_df.groupby('Product_Weight_Grams').count().sort_values('D
         print(cust_tran_df.groupby('Product_Weight_Grams').count()['Date'].
         print('Number of Chip Weight Gram Categoires:',cust_tran_df['Produc
```

```
Product_Weight_Grams
125     1454
180     1468
70      1507
220     1564
160     2970
190     2995
90      3008
250     3169
135     3257
200     4473
210     6272
270     6285
380     6418
330    12540
300    15166
165    15297
170    19983
110    22387
134    25102
150    43131
175    66390
Name: Date, dtype: int64
Number of Chip Weight Gram Categoires: 21
```



In [41]:
```
# Lifestage (Have this column and others from breaking the column d
# Age group: Young, Midage, Old, Unknown.
# Relationship type: Family, Single/Couples, Unknown.

# Family: The definition of family Assumed to be parents/in a relat
          # But a family can be a old person e.g. Grandparent with
          # A family can be just a group of people not connected by


# Unknowns; When breaking the Life_stage Column.
# Age group
cust_behv_df[cust_behv_df['Life_Stage'] == 'New Families']
# Relationship type (A retiree can be single/couple, can be a famil
cust_behv_df[cust_behv_df['Life_Stage'] == 'Retirees']


cust_behv_df['Age_Group'] = cust_behv_df['Life_Stage'].apply(lambda
                                                      .apply(lambda
                                                      .apply(lambda

cust_behv_df['Relationship_Type'] = cust_behv_df['Life_Stage'].appl
                                                       .apply(lambd
                                                       .apply(lambd
                                                       .apply(lambd

#cust_behv_df.head()
cust_tran_df.head()
```

Out[41]:

	Date	Store_Number	Loyalty_Card_Number	Taxation_Id	Product_Number	Product_Nam
0	43390	1	1000	1	5	Natural Ch Compr Seasa
1	43599	1	1307	348	66	Ccs Nach Chees
2	43605	1	1343	383	61	Smiths Crink Cut Chip Chicke
3	43329	2	2373	974	69	Smiths Ch Thin S/Cream&Onic
4	43330	2	2426	1038	108	Kettle Tortil Chpshny&Jlpr Ch

In [42]:
```
# For product name, the Brand names, some other minute words need t
#pp.pprint()
product_names = sorted(cust_tran_df['Product_Name'].unique())


product_name_col_change = {'Cheetos Chs & Bacon Balls':'Cheetos Che
 'Cobs Popd Sour Crm &Chives Chips':'Cobs Popd Sour Cream & Chives
 'Cobs Popd Swt/Chlli &Sr/Cream Chips':'Cobs Popd Sweet Chilli & So
 'Dorito Corn Chp Supreme':'Doritos Corn Chips Supreme',
 'Doritos Corn Chip Mexican Jalapeno':'Doritos Corn Chips Mexican J
 'Doritos Corn Chip Southern Chicken':'Doritos Corn Chips Southern
```

```
'Grain Waves Sour Cream&Chives':'Grain Waves Sour Cream & Chives',
'Grnwves Plus Btroot & Chilli Jam':'Grain Waves Plus Beetroot & Ch
'Infuzions Mango Chutny Papadums':'Infuzions Mango Chutney Papadum
'Infuzions Sourcream&Herbs Veg Strws':'Infuzions Sour Cream & Herb
'Infuzions Thai Sweetchili Potatomix':'Infuzions Thai Sweet Chilli
'Infzns Crn Crnchers Tangy Gcamole':'Infuzions Corn Crunchers Tang
'Kettle Sensations Bbq&Maple': 'Kettle Sensations Barbeque & Maple
'Kettle Sensations Siracha Lime':'Kettle Sensations Sriracha Lime'
'Kettle Swt Pot Sea Salt':'Kettle Sweet Pot Sea Salt',
'Kettle Tortilla Chpsbtroot&Ricotta':'Kettle Tortilla Chips Beetro
'Kettle Tortilla Chpsfeta&Garlic': 'Kettle Tortilla Chips Feta & G
'Kettle Tortilla Chpshny&Jlpno Chili':'Kettle Tortilla Chips Honey
'Natural Chip Co Tmato Hrb&Spce':'Natural Chip Company Tomato Herb
'Natural Chip Compny Seasalt':'Natural Chip Company Sea Salt',
'Natural Chipco Hony Soy Chckn':'Natural Chip Company Honey Soy Ch
'Natural Chipco Sea Salt & Vinegr':'Natural Chip Company Sea Salt 
'Ncc Sour Cream & Garden Chives':'Natural Chip Company Sour Cream 
'Old El Paso Salsa Dip Chnky Tom Ht':'Old El Paso Salsa Dip Chunck
'Old El Paso Salsa Dip Tomato Med':'Old El Paso Salsa Dip Tomato M
'Pringles Chicken Salt Crips':'Pringles Chicken Salt Crisps',
'Pringles Slt Vingar':'Pringles Salt Vinegar',
'Pringles Sourcream Onion':'Pringles Sour Cream Onion',
'Pringles Sthrn Friedchicken':'Pringles Southern Fried Chicken',
'Pringles Sweet&Spcy Bbq':'Pringles Sweet & Spicy Barbeque',
'Red Rock Deli Chikn&Garlic Aioli':'Red Rock Deli Chicken & Garlic
'Red Rock Deli Sp Salt & Truffle':'Red Rock Deli Spicy Salt & Truf
'Red Rock Deli Sr Salsa & Mzzrlla':'Red Rock Deli Sr Salsa & Mozza
'Red Rock Deli Thai Chilli&Lime':'Red Rock Deli Thai Chilli & Lime
'Rrd Chilli& Coconut':'Red Rock Chilli & Coconut',
'Rrd Honey Soy Chicken':'Red Rock Honey Soy Chicken',
'Rrd Lime & Pepper':'Red Rock Lime & Pepper',
'Rrd Pc Sea Salt':'Red Rock Potato Chips Sea Salt',
'Rrd Salt & Vinegar':'Red Rock Salt & Vinegar',
'Rrd Sr Slow Rst Pork Belly':'Red Rock Special Reserve Slow Roast 
'Rrd Steak & Chimuchurri':'Red Rock Steak & Chimichurri',
'Rrd Sweet Chilli & Sour Cream':'Red Rock Sweet Chilli & Sour Crea
'Smiths Chip Thinly Cut Original':'Smiths Chips Thinly Cut Origina
'Smiths Chip Thinly Cutsalt/Vinegr':'Smiths Chips Thinly Cut Salt 
'Smiths Chip Thinly S/Cream&Onion':'Smiths Chips Thinly Sour Cream
'Smiths Crinkle Cut Chips Chs&Onion':'Smiths Crinkle Cut Chips Che
'Smiths Crinkle Cut French Oniondip':'Smiths Crinkle Cut French On
'Smiths Crinkle Cut Snag&Sauce':'Smiths Crinkle Cut Snag & Sauce',
'Smiths Crnkle Chip Orgnl Big Bag':'Smiths Crinkle Chips Original 
'Smiths Thinly Swt Chli&S/Cream':'Smiths Thinly Sweet Chilli & Sou
'Snbts Whlgrn Crisps Cheddr&Mstrd':'Sunbites Wholegrain Crisps Che
'Sunbites Whlegrn Crisps Frch/Onin':'Sunbites Wholegrain Crisps Fr
'Thins Chips Light& Tangy':'Thins Chips Light & Tangy',
'Thins Chips Originl Saltd':'Thins Chips Original Salted',
'Thins Chips Seasonedchicken':'Thins Chips Seasoned Chicken',
'Tyrrells Crisps Ched & Chives':'Tyrrells Crisps Cheese & Chives',
'Ww Crinkle Cut Chicken':'Woolworths Crinkle Cut Chicken',
'Ww Crinkle Cut Original':'Woolworths Crinkle Cut Original',
'Ww D/Style Chip Sea Salt':'Woolworths Deli Style Chips Sea Salt',
'Ww Original Corn Chips':'Woolworths Original Corn Chips',
'Ww Original Stacked Chips':'Woolworths Original Stacked Chips',
'Ww Sour Cream &Onionstacked Chips':'Woolworths Sour Cream & Onion
'Ww Supreme Cheese Corn Chips':'Woolworths Supreme Cheese Corn Chi
```

```
def product_name_expansion(current_product_names:list, new_expanded
    for product_name in current_product_names:
        if product_name not in new_expanded_names.keys():
            new_expanded_names[product_name] = product_name


    if len(current_product_names) == len(new_expanded_names.keys())
        if show_old_new == True and show_changed == True:
            raise Exception('Can Only Have One Argument, Either "sh
        new_val_series = cust_tran_df['Product_Name'].map(new_expan
        if show_changed:
            print('-- Changed --', new_val_series, sep='\n')
        if show_old_new:
            print('-- old --',cust_tran_df['Product_Name'], '-- new
        if in_place:
            cust_tran_df['Product_Name'] = new_val_series
        else:
            return new_val_series

product_name_expansion(product_names,product_name_col_change,show_o
```

```
-- old --
0                 Natural Chip Compny Seasalt
1                            Ccs Nacho Cheese
2            Smiths Crinkle Cut Chips Chicken
3            Smiths Chip Thinly S/Cream&Onion
4         Kettle Tortilla Chpshny&Jlpno Chili
                           ...
264831     Kettle Sweet Chilli And Sour Cream
264832               Tostitos Splash Of Lime
264833                      Doritos Mexicana
264834     Doritos Corn Chip Mexican Jalapeno
264835               Tostitos Splash Of Lime
Name: Product_Name, Length: 264836, dtype: object
-- new --
0                         Natural Chip Company Sea Salt
1                                      Ccs Nacho Cheese
2                      Smiths Crinkle Cut Chips Chicken
3                 Smiths Chips Thinly Sour Cream & Onion
4         Kettle Tortilla Chips Honey & Jalapeno Chilli
                                ...
264831               Kettle Sweet Chilli And Sour Cream
264832                         Tostitos Splash Of Lime
264833                                Doritos Mexicana
264834             Doritos Corn Chips Mexican Jalapeno
264835                         Tostitos Splash Of Lime
Name: Product_Name, Length: 264836, dtype: object
```

In [43]:
```
# Single word brand names.

def unique(array:list):
    unique = []
    for name in array:
        if name not in unique:
            unique.append(name)
    return unique
one_word_brand_names_maybe = unique([product_name.split()[0] for pr
one_word_brand_names_maybe = unique([product_name.split()[0] for pr

# Renaming Product Name.
cust_tran_df['Product_Name'] = cust_tran_df['Product_Name'].apply(l

# Used for determine brand name from product name.
pprint.pprint(sorted(cust_tran_df['Product_Name'].unique()))

brand_names_word_segment = {'Burger': 'Burger',
'Ccs': 'Ccs',
 'Cheetos': 'Cheetos',
 'Cheezels': 'Cheezels',
 'Cobs': 'Cobs',
 'Doritos': 'Doritos',
 'French': 'Unknown (french fries)',
 'Grain': 'Grain Waves',
 'Infuzions': 'Infuzions',
 'Kettle': 'Kettle',
 'Natural': 'Natural Chip Company',
 'Old': 'Old El Paso',
 'Pringles': 'Pringles',
 'Red': 'Red Rock',
 'Smiths': 'Smiths',
 'Sunbites': 'Sunbites',
 'Thins': 'Thins',
 'Tostitos': 'Tostitos',
 'Twisties': 'Twisties',
 'Tyrrells': 'Tyrrells',
 'Woolworths': 'Woolworths'}

cust_tran_df['Brand_Name'] = cust_tran_df['Product_Name'].apply(lam
cust_tran_df['Product_Name'] = cust_tran_df['Product_Name'].apply(l

# Flavour
# Remove brand, Remove words'Chips, Chip, Potato'

# The following script was used to search for the product names to
'''
search_terms = sorted(cust_tran_df['Product_Name'].unique())

iter_terms_to_sleep = iter(zip(np.random.random_sample((1,len(searc
for time_to_sleep,query, in iter_terms_to_sleep:
    time_to_sleep = time_to_sleep + np.random.randint(5,7)
    query_formatted = re.sub('\s','+',query)
    url = f"https://www.google.com/search?q={query_formatted}+&clie
    print('Executed: ',query,'| Sleep:', time_to_sleep)
    time.sleep(time_to_sleep)
    webbrowser.open_new_tab(url)
'''

not_a_chip = ['Burger Rings','Cheetos Puffs','Cheezels Cheese','Che
              'Infuzions Bbq Rib Prawn Crackers', 'Infuzions Thai S
             'Woolworths Mild Salsa']

for product in not_a_chip:
    cust_tran_df.drop(cust_tran_df[cust_tran_df['Product_Name'] ==

#for x in cust_tran_df['Product_Name'].unique():
 #   if 'mexica' in x.lower(): print(x)
#print(cust_tran_df['Product_Name'].apply(la)
# What is a chip ?
    # This is a debate topic. Hence, my prespective on it.
    # a wafer-thin slice of potato fried or baked until crisp and e
    # A chip can be classified by a multitude of attributes. One be
# Quantium suggestion, has chips in the product name: That's incorr
    # Whilst techincally incorrect as for example it gets rid of ch
        # However as it's minute detail, technically again incorrec
```

```
['Burger Rings',
 'Ccs Nacho Cheese',
 'Ccs Original',
 'Ccs Tasty Cheese',
 'Cheetos Cheese & Bacon Balls',
 'Cheetos Puffs',
 'Cheezels Cheese',
 'Cheezels Cheese Box',
 'Cobs Popd Sea Salt Chips',
 'Cobs Popd Sour Cream & Chives Chips',
 'Cobs Popd Sweet Chilli & Sour Cream Chips',
 'Doritos Cheese Supreme',
 'Doritos Corn Chips Cheese Supreme',
 'Doritos Corn Chips Mexican Jalapeno',
 'Doritos Corn Chips Nacho Cheese',
 'Doritos Corn Chips Original',
 'Doritos Corn Chips Southern Chicken',
 'Doritos Corn Chips Supreme',
 'Doritos Mexicana',
 'Doritos Salsa Medium'
```

In [44]:

```
 # Other columns. (Relevant) that can be used to analyse customer b
 # Flavour. (Can be a variable), Brand name (variable), Packet weig
 # Packet Size, Packet material, Packet air volume to chip volume,
 # Nutiritionl Metrics (Food package backside).
 # Chip Ingredients, Chip Size, Chip shape, Chip Color, Chip cut ty
 # Branding(package styling, colors, name, logo, slogan, 5 feelings
 # Partnership, Placement, Inventory Management (By store), Efficen
 # Execution ability by management.
 # Price.
 # Exclusivity Metric (Rare package, limited edition)
 # Type of customer (Health,Foodie, Age group, Type of work they do
```

In [45]:

```

# Dataset 1; Basic Cleaning.
# Titlised cell values.

cust_behv_df['Life_Stage'] = cust_behv_df['Life_Stage'].apply(lambd
cust_behv_df['Age_Group'] = cust_behv_df['Age_Group'].apply(lambda
cust_behv_df['Relationship_Type'] = cust_behv_df['Relationship_Type
```

```
In [46]: # Duplicates Dataset 1
         print('Duplicated Row: ',cust_behv_df.duplicated().sum())

         # Duplicates Rows Processing
         #cust_tran_df.drop_duplicates(inplace=True)

         #print('Duplicated Row: ',cust_tran_df.duplicated().sum())
```

```
Duplicated Row:  0
```

```
In [47]: # Duplicates Dataset 2
         print('Duplicated Row: ',cust_tran_df.duplicated().sum())
         print_mod(cust_tran_df[cust_tran_df.duplicated()])
         # Duplicates Rows Processing
         cust_tran_df.drop_duplicates(inplace=True)

         print('Duplicated Row: ',cust_tran_df.duplicated().sum())
```

```
Duplicated Row:  1

          Date  Store_Number  Loyalty_Card_Number  Taxation_Id  Pr
oduct_Number  \
124845  43374           107               107024       108462
45

                           Product_Name  Product_Quantity  Total_S
ales  \
124845  Smiths Thinly Cut Roast Chicken                 2
6.0

        Product_Price  Product_Weight_Grams Brand_Name
124845            3.0                   175     Smiths
Duplicated Row:  0
```

In [48]:
```python
# Type Casting.

def percentage_change(before,after,verbose: bool=False ) -> int or
    percentage_change = ((after - before) / before) * 100
    if verbose:
        if percentage_change > 0:
            return f'A increase by {percentage_change}%'
        elif percentage_change == 0:
            return f'No Change: {percentage_change}%'
        return f'A decrease by {abs(percentage_change)}%'
    return percentage_change

def dataset_types(data_frame) -> None:
    data_frame.info()
    print()
    for column_name in data_frame.columns:
        print(column_name, type(data_frame[column_name][0]))
    print(data_frame.head())

def mem_usage(data_frame, size: str='mb', show_out: bool=True, retu
    '''Gives the memory size of a dataframe'''
    size = size.title()
    memory_size_unit = {'Kb':10 ** 3,'Mb':10 ** 6,'Gb':10 ** 9,'Tb'
    if size not in memory_size_unit.keys():
            raise Exception(f"Available Memory Sizes: {','.join(mem
    calculation = data_frame.memory_usage().sum() / memory_size_uni
    if show_out:
        print(f'Memory Size: {calculation} {size}')
    if return_calc:
        return (calculation,size)

def numeric_cols_max_min_determiner(data_frame,dtype_info: bool=Fal
    numeric_dtypes = [float,int,np.int8,np.int16,np.int32,np.int64,

    cnt = 0
    for column in data_frame.columns:
        cnt += 1
        if type(data_frame[column].iloc[0]) in numeric_dtypes:
            if cnt == len(data_frame.columns):
                print(f"Column <{column}>\n\tmax: {data_frame[colum
            else:
                print(f"Column <{column}>\n\tmax: {data_frame[colum

    if dtype_info:
        print()
        for numeic_data_type in numeric_dtypes:
            if 'int' in str(numeic_data_type):
                print(np.iinfo(numeic_data_type))
            else:
                print(str(np.finfo(numeic_data_type)))
```

In [49]:
```
# Data Set 1
dataset_types(cust_behv_df)
before_casting_memory_usage = mem_usage(cust_behv_df)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72637 entries, 0 to 72636
Data columns (total 5 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   Loyalty_Card_Number  72637 non-null  int64 
 1   Life_Stage           72637 non-null  object
 2   Card_Subscription    72637 non-null  object
 3   Age_Group            72637 non-null  object
 4   Relationship_Type    72637 non-null  object
dtypes: int64(1), object(4)
memory usage: 2.8+ MB

Loyalty_Card_Number <class 'numpy.int64'>
Life_Stage <class 'str'>
Card_Subscription <class 'str'>
Age_Group <class 'str'>
Relationship_Type <class 'str'>
   Loyalty_Card_Number              Life_Stage Card_Subscription A
ge_Group  \
0                 1000   Young Singles/Couples           Premium 
Young   
1                 1002   Young Singles/Couples        Mainstream 
Young   
2                 1003          Young Families            Budget 
Young   
3                 1004   Older Singles/Couples        Mainstream 
Older   
4                 1005  Midage Singles/Couples        Mainstream 
Midage   

  Relationship_Type  
0   Singles/Couples  
1   Singles/Couples  
2          Families  
3   Singles/Couples  
4   Singles/Couples  
Memory Size: 2.905608 Mb
```

In [50]:
```
# Type Casting Checks # DataSet 1
# The Loyalty Card Number is an identifier, used for identifying a
numeric_cols_max_min_determiner(cust_behv_df)
```

```
Column <Loyalty_Card_Number>
        max: 2373711 | min: 1000
```

In [51]:
```
cust_behv_df['Loyalty_Card_Number'] = cust_behv_df['Loyalty_Card_Nu
```

In [52]:
```
dataset_types(cust_behv_df)
after_casting_memory_usage = mem_usage(cust_behv_df) # Notice the m
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72637 entries, 0 to 72636
Data columns (total 5 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Loyalty_Card_Number   72637 non-null  uint32
 1   Life_Stage            72637 non-null  object
 2   Card_Subscription     72637 non-null  object
 3   Age_Group             72637 non-null  object
 4   Relationship_Type     72637 non-null  object
dtypes: object(4), uint32(1)
memory usage: 2.5+ MB

Loyalty_Card_Number <class 'numpy.uint32'>
Life_Stage <class 'str'>
Card_Subscription <class 'str'>
Age_Group <class 'str'>
Relationship_Type <class 'str'>
   Loyalty_Card_Number              Life_Stage Card_Subscription A
ge_Group  \
0                 1000   Young Singles/Couples           Premium
Young
1                 1002   Young Singles/Couples        Mainstream
Young
2                 1003          Young Families            Budget
Young
3                 1004   Older Singles/Couples        Mainstream
Older
4                 1005  Midage Singles/Couples        Mainstream
Midage

  Relationship_Type
0   Singles/Couples
1   Singles/Couples
2          Families
3   Singles/Couples
4   Singles/Couples
Memory Size: 2.61506 Mb
```

In [53]:
```
print('Original:',before_casting_memory_usage,'After:',after_castin
print(percentage_change(before_casting_memory_usage[0],after_castin
```

```
Original: (2.905608, 'Mb') After: (2.61506, 'Mb')
A decrease by 9.999559472578538%
```

In [54]:
```
# Data Set 2 Basic Cleaning.
# After Further looking into the date format, excel returns a seria
# Hence they can be altered using the following.
cust_tran_df['Date'] = pd.to_datetime(cust_tran_df['Date'],unit='D'
cust_tran_df.head()
```

Out[54]:

	Date	Store_Number	Loyalty_Card_Number	Taxation_Id	Product_Number	Product_Nam
0	2018-10-17	1	1000	1	5	Natural Chi Company Se Sa
1	2019-05-14	1	1307	348	66	Ccs Nach Chees
2	2019-05-20	1	1343	383	61	Smiths Crinkl Cut Chip Chicke
3	2018-08-17	2	2373	974	69	Smiths Chip Thinly Sou Cream an Onio
4	2018-08-18	2	2426	1038	108	Kettle Tortill Chips Hone and Jalapen Chil

In [55]:
```
dataset_types(cust_tran_df)
before_casting_memory_usage = mem_usage(cust_tran_df)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 213986 entries, 0 to 264835
Data columns (total 11 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   Date                 213986 non-null  datetime64[ns]
 1   Store_Number         213986 non-null  int64
 2   Loyalty_Card_Number  213986 non-null  int64
 3   Taxation_Id          213986 non-null  int64
 4   Product_Number       213986 non-null  int64
 5   Product_Name         213986 non-null  object
 6   Product_Quantity     213986 non-null  int64
 7   Total_Sales          213986 non-null  float64
 8   Product_Price        213986 non-null  float64
 9   Product_Weight_Grams 213986 non-null  int64
 10  Brand_Name           213986 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(6), object(2)
memory usage: 19.6+ MB

Date <class 'pandas._libs.tslibs.timestamps.Timestamp'>
Store_Number <class 'numpy.int64'>
Loyalty_Card_Number <class 'numpy.int64'>
Taxation_Id <class 'numpy.int64'>
Product_Number <class 'numpy.int64'>
Product_Name <class 'str'>
Product_Quantity <class 'numpy.int64'>
Total_Sales <class 'numpy.float64'>
```

```
Product_Price <class 'numpy.float64'>
Product_Weight_Grams <class 'numpy.int64'>
Brand_Name <class 'str'>
        Date  Store_Number  Loyalty_Card_Number  Taxation_Id  Prod
uct_Number  \
0 2018-10-17             1                 1000            1
5
1 2019-05-14             1                 1307          348
66
2 2019-05-20             1                 1343          383
61
3 2018-08-17             2                 2373          974
69
4 2018-08-18             2                 2426         1038
108

                                Product_Name  Product_Quanti
ty  \
0                  Natural Chip Company Sea Salt
2
1                                Ccs Nacho Cheese
3
2               Smiths Crinkle Cut Chips Chicken
2
3         Smiths Chips Thinly Sour Cream and Onion
5
4  Kettle Tortilla Chips Honey and Jalapeno Chilli
3

   Total_Sales  Product_Price  Product_Weight_Grams            Bra
nd_Name
0          6.0           3.00                   175  Natural Chip
Company
1          6.3           2.10                   175
Ccs
2          2.9           1.45                   170
Smiths
3         15.0           3.00                   175
Smiths
4         13.8           4.60                   150
Kettle
Memory Size: 28.99684 Mb
```

In [56]:
```python
# For Ease of comparison, creating a new column: price_per_100_gram
cust_tran_df['Product_Price_Per_100_Grams'] = round(cust_tran_df['P
cust_tran_df.head()
```

Out[56]:

	Date	Store_Number	Loyalty_Card_Number	Taxation_Id	Product_Number	Product_Nam
0	2018-10-17	1	1000	1	5	Natural Chi Company Se Sa
1	2019-05-14	1	1307	348	66	Ccs Nach Chees
2	2019-05-20	1	1343	383	61	Smiths Crinkl Cut Chip Chicke
3	2018-08-17	2	2373	974	69	Smiths Chip Thinly Sou Cream an Onio
4	2018-08-18	2	2426	1038	108	Kettle Tortill Chips Hone and Jalapen Chil

In [57]:
```python
# Dataset type casting.
# Product weight grams is a string
numeric_cols_max_min_determiner(cust_tran_df)
[print(np.iinfo(x)) for x in [np.uint8,np.uint16, np.uint32]]
[print(np.finfo(x)) for x in [np.float16]]
```

```
Column <Store_Number>
        max: 272 | min: 1

Column <Loyalty_Card_Number>
        max: 2373711 | min: 1000

Column <Taxation_Id>
        max: 2415841 | min: 1

Column <Product_Number>
        max: 114 | min: 1

Column <Product_Quantity>
        max: 200 | min: 1

Column <Total_Sales>
        max: 650.0 | min: 1.7

Column <Product_Price>
        max: 6.5 | min: 1.3199999999999998

Column <Product_Weight_Grams>
        max: 380 | min: 90

Column <Product_Price_Per_100_Grams>
        max: 3.45 | min: 0.69
Machine parameters for uint8
```

```
---------------------------------------------------------------
min = 0
max = 255
---------------------------------------------------------------

Machine parameters for uint16
---------------------------------------------------------------
min = 0
max = 65535
---------------------------------------------------------------

Machine parameters for uint32
---------------------------------------------------------------
min = 0
max = 4294967295
---------------------------------------------------------------

Machine parameters for float16
---------------------------------------------------------------
precision =   3   resolution = 1.00040e-03
machep =    -10   eps =        9.76562e-04
negep =     -11   epsneg =     4.88281e-04
minexp =    -14   tiny =       6.10352e-05
maxexp =     16   max =        6.55040e+04
nexp =        5   min =        -max
---------------------------------------------------------------
```

Out[57]:
```
[None]
```

In [58]:
```
cust_tran_df['Product_Quantity'] = cust_tran_df['Product_Quantity']
cust_tran_df['Product_Number'] = cust_tran_df['Product_Number'].ast
cust_tran_df['Store_Number'] = cust_tran_df['Store_Number'].astype(
cust_tran_df['Product_Weight_Grams'] = cust_tran_df['Product_Weight
cust_tran_df['Loyalty_Card_Number'] = cust_tran_df['Loyalty_Card_Nu
cust_tran_df['Taxation_Id'] = cust_tran_df['Taxation_Id'].astype(np

# Float64 to Float16 works. however, additional decimal places are
#cust_tran_df['Product_Price_Per_100_Grams'] = cust_tran_df['Produc
#cust_tran_df['Product_Price'] = cust_tran_df['Product_Price'].appl
#cust_tran_df['Total_Sales'] = cust_tran_df['Total_Sales'].apply(la
```

In [59]:
```
dataset_types(cust_tran_df)
after_casting_memory_usage = mem_usage(cust_tran_df,show_out=False)
print()
print('Before',before_casting_memory_usage,'After',after_casting_me
print(percentage_change(before_casting_memory_usage[0],after_castin
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 213986 entries, 0 to 264835
Data columns (total 12 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   Date                       213986 non-null  datetime64[ns]
 1   Store_Number               213986 non-null  uint16
 2   Loyalty_Card_Number        213986 non-null  uint32
 3   Taxation_Id                213986 non-null  uint32
 4   Product_Number              213986 non-null  uint8
 5   Product_Name                213986 non-null  object
 6   Product_Quantity            213986 non-null  uint8
 7   Total_Sales                 213986 non-null  float64
 8   Product_Price               213986 non-null  float64
 9   Product_Weight_Grams        213986 non-null  uint16
 10  Brand_Name                  213986 non-null  object
 11  Product_Price_Per_100_Grams  213986 non-null  float64
dtypes: datetime64[ns](1), float64(3), object(2), uint16(2), uint3
2(2), uint8(2)
memory usage: 22.3+ MB

Date <class 'pandas._libs.tslibs.timestamps.Timestamp'>
Store_Number <class 'numpy.uint16'>
Loyalty_Card_Number <class 'numpy.uint32'>
Taxation_Id <class 'numpy.uint32'>
Product_Number <class 'numpy.uint8'>
Product_Name <class 'str'>
Product_Quantity <class 'numpy.uint8'>
Total_Sales <class 'numpy.float64'>
Product_Price <class 'numpy.float64'>
Product_Weight_Grams <class 'numpy.uint16'>
Brand_Name <class 'str'>
Product_Price_Per_100_Grams <class 'numpy.float64'>
        Date  Store_Number  Loyalty_Card_Number  Taxation_Id  Prod
uct_Number  \
0 2018-10-17             1                 1000            1
5
1 2019-05-14             1                 1307          348
66
2 2019-05-20             1                 1343          383
61
3 2018-08-17             2                 2373          974
69
4 2018-08-18             2                 2426         1038
108

                                     Product_Name  Product_Quanti
ty  \
0                    Natural Chip Company Sea Salt
2
1                                Ccs Nacho Cheese
3
2                  Smiths Crinkle Cut Chips Chicken
2
3          Smiths Chips Thinly Sour Cream and Onion
5
4  Kettle Tortilla Chips Honey and Jalapeno Chilli
3

   Total_Sales  Product_Price  Product_Weight_Grams            Bra
nd_Name  \
0          6.0           3.00                   175  Natural Chip
Company
1          6.3           2.10                   175
Ccs
2          2.9           1.45                   170
```

```
Smiths
3        15.0          3.00                     175
Smiths
4        13.8          4.60                     150
Kettle

   Product_Price_Per_100_Grams
0                         1.71
1                         1.20
2                         0.85
3                         1.71
4                         3.07

Before (28.99684, 'Mb') After (23.433204, 'Mb')
A decrease by 19.18704245014284%
```

2.4.1 Missing Values

In [60]:
```
print(cust_behv_df.isna().sum())
print()
print(cust_tran_df.isna().sum())
```

```
Loyalty_Card_Number    0
Life_Stage             0
Card_Subscription      0
Age_Group              0
Relationship_Type      0
dtype: int64

Date                           0
Store_Number                   0
Loyalty_Card_Number            0
Taxation_Id                    0
Product_Number                 0
Product_Name                   0
Product_Quantity               0
Total_Sales                    0
Product_Price                  0
Product_Weight_Grams           0
Brand_Name                     0
Product_Price_Per_100_Grams    0
dtype: int64
```

2.4.2 Outliers

In [61]:
```
# For customer behaviou dataset, there are no outliers.

# For the customer transaction/transfer that occured there can be o
# col -> row (numeric identifier to cat identifie) Transposed
# row transposed columns into a single column with indexes increase
# visualisable as a boxplot.
#ex.histogram(cust_tran_df['Product_Quantity'])

# Using plotly for interactive plots, as there is a considerable di
ex.box(data_frame=cust_tran_df.iloc[:,6:].T.reset_index().melt(id_v
# The Outliers can be observed, from the graph and hovered over to 
# Note not all columns have outliers.
```

In [62]:
```
print(cust_tran_df.describe())
# Chip packet points shown above as outliers, can be chip packet si
# The total sales of 650 is justified by the 200 product quantity o
# The product quantity bought seems like an outlier, but in fact a 
    # For Example: This can occur if for example a person is partic
    # They can also be running thier own store (re-selling it, and 
print(cust_tran_df[cust_tran_df['Product_Quantity'] == cust_tran_df
```

```
        Store_Number  Loyalty_Card_Number   Taxation_Id  Product_N
umber  \
count  213986.000000         2.139860e+05  2.139860e+05   213986.0
00000
mean      135.150954         1.356399e+05  1.352342e+05       55.3
82572
std        76.749305         8.081598e+04  7.812943e+04       33.3
10589
min         1.000000         1.000000e+03  1.000000e+00        1.0
00000
25%        70.000000         7.004600e+04  6.776125e+04       26.0
00000
50%       130.000000         1.303880e+05  1.352760e+05       52.0
00000
75%       203.000000         2.030940e+05  2.026985e+05       83.0
00000
max       272.000000         2.373711e+06  2.415841e+06      114.0
00000

       Product_Quantity    Total_Sales  Product_Price  Product_Wei
ght_Grams  \
count     213986.000000  213986.000000  213986.000000         2139
86.000000
mean           1.908115       7.307322       3.824852            1
73.614690
std            0.695743       3.176891       1.091602
54.903837
min            1.000000       1.700000       1.320000
90.000000
25%            2.000000       5.800000       3.000000            1
50.000000
50%            2.000000       7.400000       3.700000            1
70.000000
75%            2.000000       8.800000       4.600000            1
75.000000
max          200.000000     650.000000       6.500000            3
80.000000

       Product_Price_Per_100_Grams
count                213986.000000
mean                      2.293107
std                       0.664562
min                       0.690000
25%                       1.730000
50%                       2.510000
75%                       2.760000
max                       3.450000
            Date  Store_Number  Loyalty_Card_Number  Taxation_Id
\
69762 2018-08-19           226               226000       226201
69763 2019-05-20           226               226000       226210

       Product_Number                Product_Name  Product_Quantit
y  \
69762               4  Doritos Corn Chips Supreme              20
0
69763               4  Doritos Corn Chips Supreme              20
0
```

```
       Total_Sales  Product_Price  Product_Weight_Grams Brand_Name  \
69762        650.0           3.25                   380    Doritos
69763        650.0           3.25                   380    Doritos

       Product_Price_Per_100_Grams
69762                         0.86
69763                         0.86
```

2.4.3 Other Anomalies

In [63]:
```
# None. Not enough adequate knowdlege to do so for now.
```

In [64]:
```
cust_behv_df.head()
```

Out[64]:

	Loyalty_Card_Number	Life_Stage	Card_Subscription	Age_Group	Relationship_Type
0	1000	Young Singles/Couples	Premium	Young	Singles/Couples
1	1002	Young Singles/Couples	Mainstream	Young	Singles/Couples
2	1003	Young Families	Budget	Young	Families
3	1004	Older Singles/Couples	Mainstream	Older	Singles/Couples
4	1005	Midage Singles/Couples	Mainstream	Midage	Singles/Couples

3 Data Analysis and Data Merging.

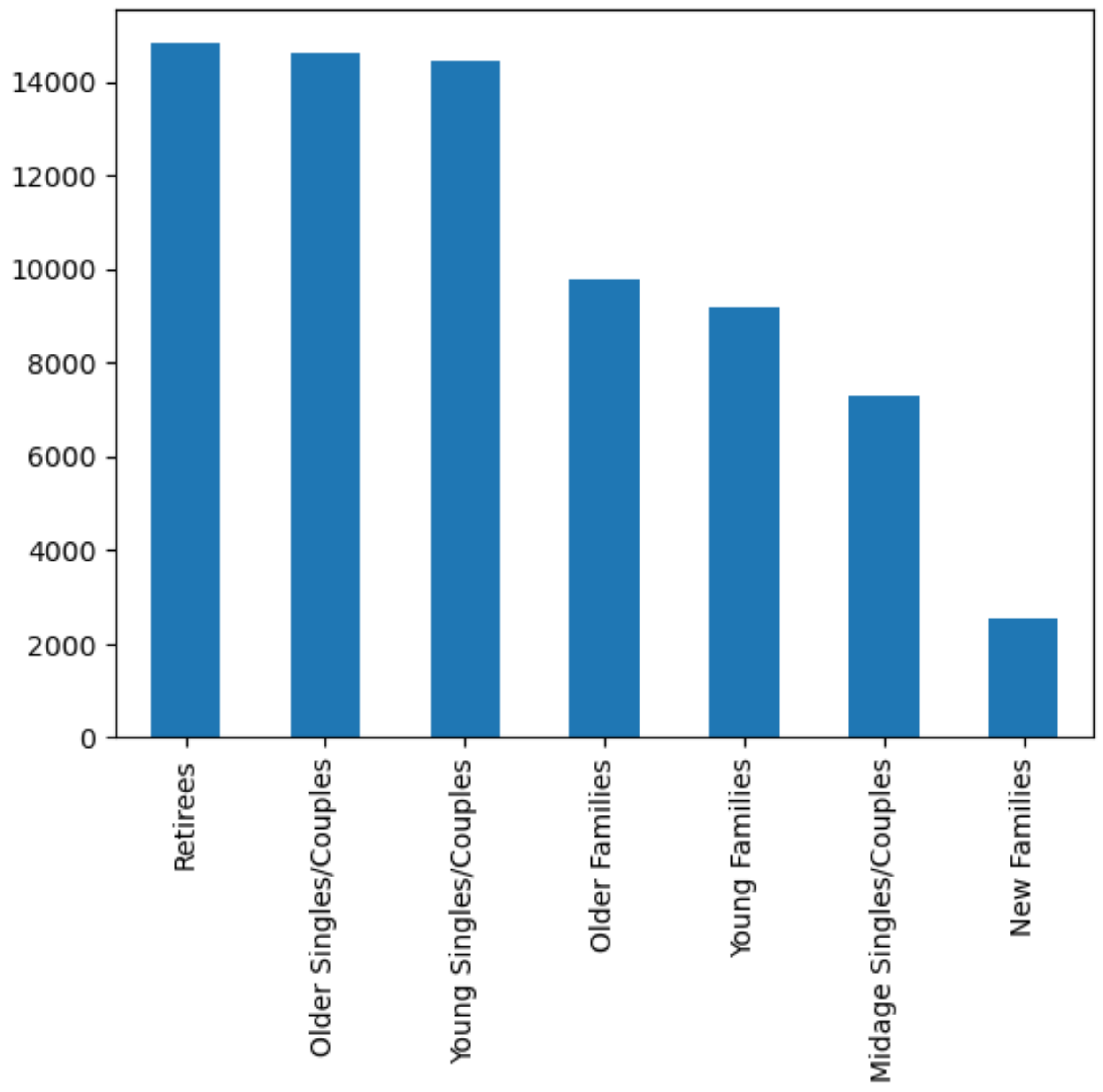
In [65]:
```

#### 3.2 Metrics Definition and analysis.
#### 3.3 More Question Synthesis and data visualisation.
```

3.1 Simple Questions.

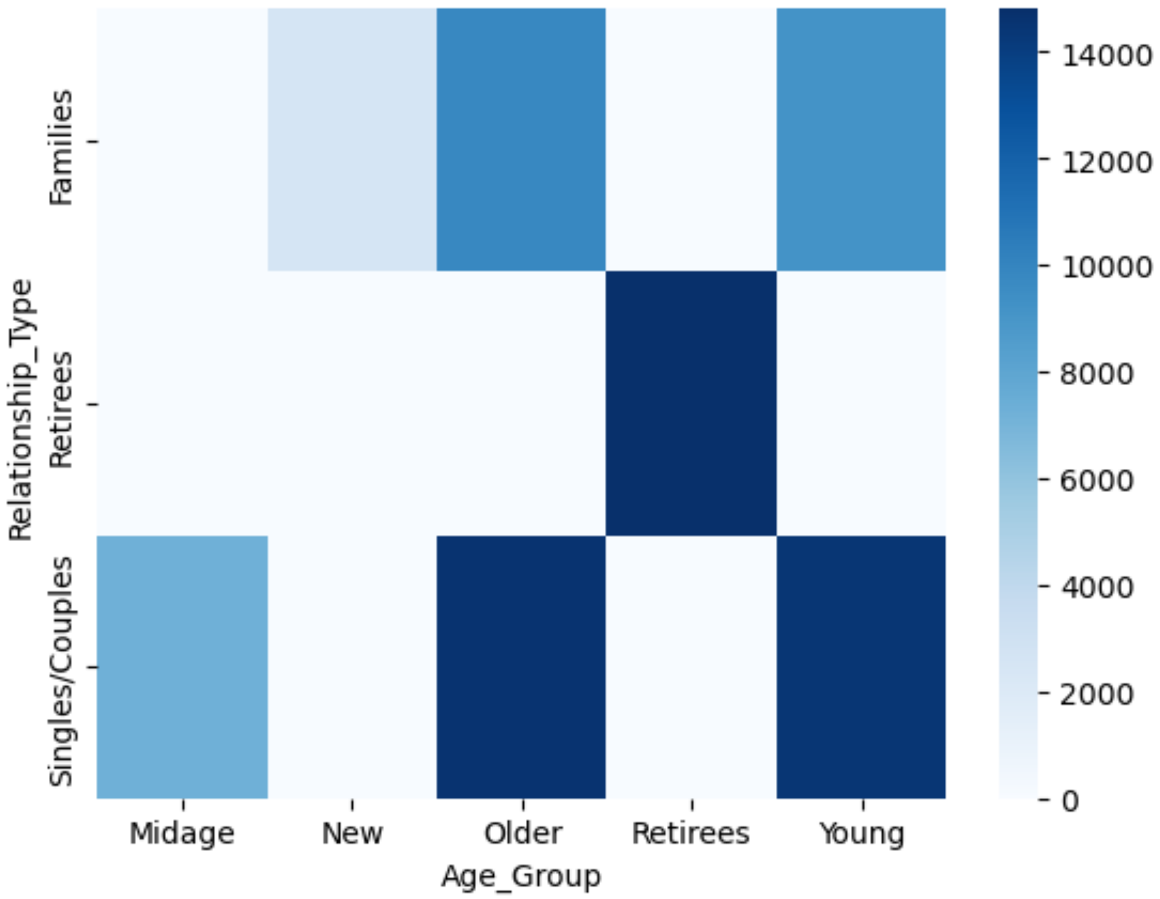
```
In [66]: # Single Dataset
             # Customer behaviour dataset.
                 # Basic visual presentation of data.
                 # Which age group has which relationship type ?
                 # How are different life stages distributed based on card s

             # Customer product dataset.
              # Individual columns.
                 # The cheapest product.
                 # The most expensive product.
                 # Which product brings the most revenue ?
                 # Which Store brings the most revenue in the supermarket ch
                 # Total revenue for chips for the supermarket per year, ove
                 # What is the chip season ? When are chips mostly bought an
                 # Which brand has the most products ? What is the products
                 # Total tax paid by the supermarket chain ? Average tax pai
                 # The most weight, the least weight.
                 # The best product for the customer based on weight. # The
                 # Which product is the most popular and which is the least
                 # Which brand is the most popular and which is the least ?
                 # Who bought more than one product type in a single order ?
                 # How many products are bought on a day on average in the s
                 # Who spends the most on chips (total sales), describing cu
                 # How premium their general purchasing behaviour is
                 # How many customers are in each segment
                 # How many chips are bought per customer by segment
                 # What's the average chip price by customer segment


         # Merged Dataset
             # How to group segements by age group ; by card subsiption; by
             # How does being part of multiple goup segements change your bu
             # How does the buying behaviour change over time ?
             # What are other patterns in buying by consumers ?
             # Who spends the most on chips (total sales), describing custom
                # how premium their general purchasing behaviour is
             # How many customers are in each segment
             # How many chips are bought per customer by segment
             # What's the average chip price by customer segment
             # The customer's total spend over the period and total spend fo
             # Proportion of customers in each customer segment overall to c
```

In [67]:
```
# Basic visual presentation of data. # use comments to select.
#cust_behv_df['Age_Group'].value_counts().plot.bar()
cust_behv_df['Life_Stage'].value_counts().plot.bar()
#cust_behv_df['Relationship_Type'].value_counts().plot.bar()
#cust_behv_df['Card_Subscription'].value_counts().plot.bar()
```

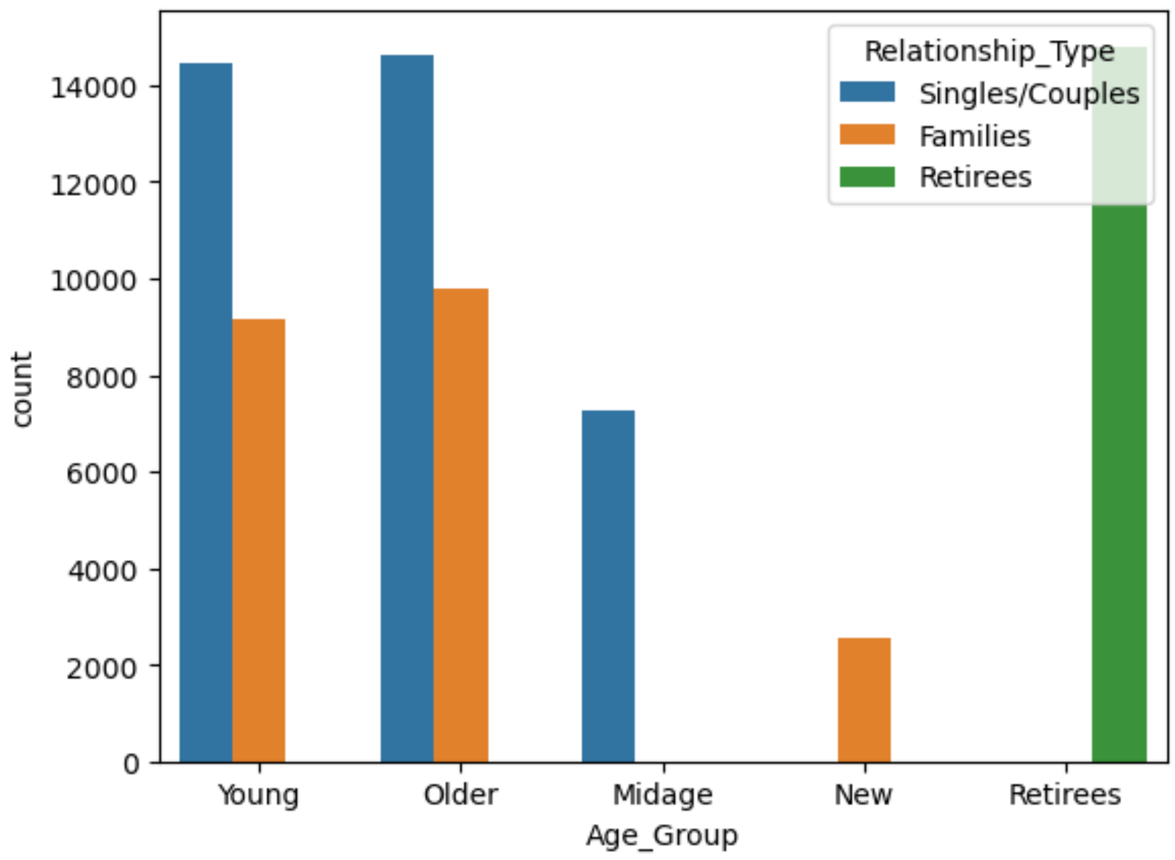
Out[67]:
```
<AxesSubplot:>
```



```
In [68]: # Which age group has which relationship type ?
         sns.heatmap(cust_behv_df.groupby(['Age_Group','Relationship_Type'])

Out[68]: <AxesSubplot:xlabel='Age_Group', ylabel='Relationship_Type'>
```



In [69]:
```
# How are different life stages distributed based on card subscript
ax = sns.countplot(data=cust_behv_df,x='Age_Group',hue='Relationshi
sns.move_legend(ax, "upper right", bbox_to_anchor=(1, 1))
```

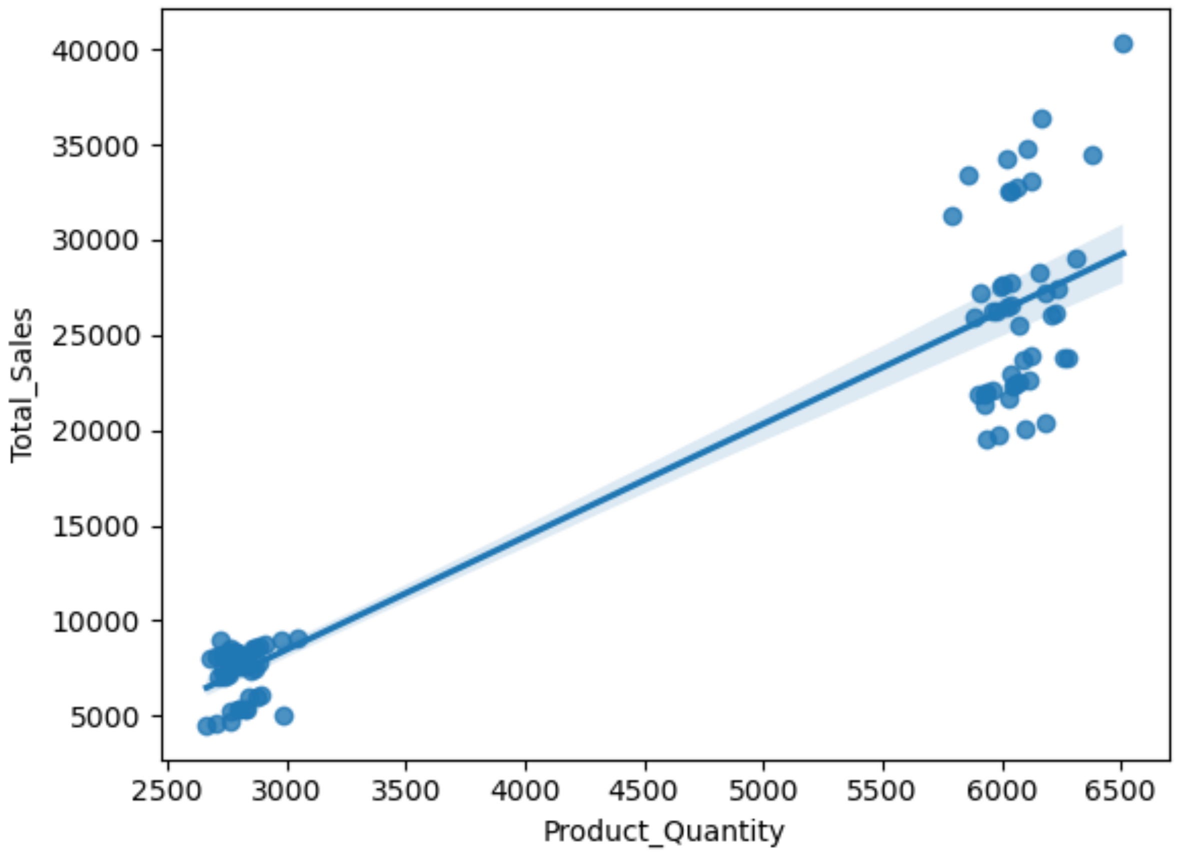


In [70]:
```
# The cheapest product.
print('Most cheapest product:')
print(cust_tran_df[cust_tran_df['Product_Price'] == cust_tran_df['P

# The most expensive product.
print('Most expensive product:')
print(cust_tran_df[cust_tran_df['Product_Price'] == cust_tran_df['P
```

```
Most cheapest product:
['Thins Chips Salt and Vinegar']        1.32
Most expensive product:
['Doritos Corn Chips Supreme']   6.5
```

```
In [71]:  # Which product brings the most revenue ?
         revenue_quantity_df = cust_tran_df.groupby('Product_Name').sum()[['
         print(revenue_quantity_df.head()) # Doritos Corn Chips Supreme brin
         # Is there a correlation between the quantity sold and most revenue
         # There is a strong positive coorelation between the two variables.
         # Hence, selling more products will yeild more revenue.
         # Note how two clusters seem to form.
          # The bottom left cluster is of products that sell amidst the 2500
          # Whereas, the top right cluster is of products that sell in the 5
         sns.regplot(data=revenue_quantity_df,y='Total_Sales',x='Product_Qua
         print(revenue_quantity_df.corr(method='pearson'))
          # Quantity sold and revenue brought
```

```
                                        Total_Sales  Product_Quanti
ty
Product_Name
Doritos Corn Chips Supreme                  40352.0            6509
.0
Smiths Crinkle Chips Original Big Bag       36367.6            6164
.0
Smiths Crinkle Chips Salt and Vinegar       34804.2            6106
.0
Kettle Mozzarella Basil and Pesto           34457.4            6381
.0
Smiths Crinkle Original                     34302.6            6018
.0
                  Total_Sales  Product_Quantity
Total_Sales          1.000000          0.940523
Product_Quantity     0.940523          1.000000
```



```
In [72]: # Which brand brings the most revenue ? Which is the best brand ?
         brand_df = cust_tran_df.groupby('Brand_Name').agg(['sum','count'])[
         print(brand_df.head())
         # Kettle is the brand that brings the most revenue.
         # This can be also looked at in the following manner:
             # How much does 1 product sold for the brand generate what amou
         # OR
             # How many products need to be sold to bring in revenue of 1 do
         brand_df['per_product_sold_revenue'] = brand_df['sum'] / brand_df['
         brand_df['products_sold_for_single_dollar'] = brand_df['count'] / b
         print(brand_df.sort_values('per_product_sold_revenue',ascending=Fal
         # Old El Paso for a single product sold brings in revenue of $9.74.
```

```
                 sum  count
Brand_Name
Kettle      390239.8  41288
Doritos     227629.9  25226
Smiths      224654.2  31822
Pringles    177655.5  25102
Red Rock     95046.0  17779
                           sum  count  per_product_sold_revenue  \
Brand_Name
Kettle                390239.8  41288                  9.451652
Doritos               227629.9  25226                  9.023622
Tostitos               79789.6   9471                  8.424623
Tyrrells               51647.4   6442                  8.017293
Cobs                   70569.8   9693                  7.280491
Pringles              177655.5  25102                  7.077344
Smiths                224654.2  31822                  7.059713
Grain Waves            51617.2   7740                  6.668889
Thins                  88852.5  14075                  6.312789
Natural Chip Company   42318.0   7469                  5.665819
Red Rock               95046.0  17779                  5.345970
Ccs                    18078.9   4551                  3.972512
Woolworths             35889.5  10320                  3.477665
Sunbites                9676.4   3008                  3.216888

                      products_sold_for_single_dollar
Brand_Name
Kettle                                       0.105802
Doritos                                      0.110820
Tostitos                                     0.118700
Tyrrells                                     0.124730
Cobs                                         0.137353
Pringles                                     0.141296
Smiths                                       0.141649
Grain Waves                                  0.149950
Thins                                        0.158409
Natural Chip Company                         0.176497
Red Rock                                     0.187057
Ccs                                          0.251730
Woolworths                                   0.287549
Sunbites                                     0.310859
```

In [73]:
```
# Which Store brings the most revenue in the supermarket chain ? #
cust_tran_df.groupby(['Store_Number']).sum()['Product_Quantity'].so
#cust_tran_df.groupby(['Store_Number']).count().sort_values(by='Dat

# Out of the 272 stores present that sell chips, for the client:
    # Store number 226 has the most total sales.
    # Reason: Poduct quantity sold.
        # This could be as a reason of location, stock availability
        # The cause behind this high product quantity being sold is
```

Out[73]:
```
Store_Number
226    3605.0
88     2972.0
93     2962.0
165    2940.0
237    2847.0
Name: Product_Quantity, dtype: float64
```

In [74]:
```
# Average Product price, quantity sold, total sales, product weight
cust_tran_df.mean()
```

Out[74]:
```
Store_Number                     135.150954
Loyalty_Card_Number           135639.861159
Taxation_Id                   135234.197078
Product_Number                    55.382572
Product_Quantity                   1.908115
Total_Sales                        7.307322
Product_Price                      3.824852
Product_Weight_Grams             173.614690
Product_Price_Per_100_Grams        2.293107
dtype: float64
```

```
In [75]: # Daily analysis on transactitons, poduct quantity and revenue.

         # Use if plotly isn't present.
         #plt.figure(figsize=(20,20))
         #graph = cust_tran_df['Date'].value_counts().plot(kind='line')
         #graph.set_xlabel('Date')
         #graph.set_ylabel('Count')
         #graph.set_title('Chips Bought on Each Day')

         ex.line(data_frame=cust_tran_df['Date'].value_counts().reset_index(
                         x='index',y='Date',labels={
                              "Date": "Count",
                              "index": "Date"}, title = 'Number of Transacti
         # Total revenue for chips for the supermarket per year, over time,
         # What is the chip season ? When are chips mostly bought and sold ?
```

In [76]:
```
ex.line(data_frame=cust_tran_df.groupby('Date').sum()['Product_Quan
                 x='Date',y='Product_Quantity', title = 'Chips Bough
```

In [77]:
```
ex.line(data_frame=cust_tran_df.groupby('Date').sum()['Total_Sales'
                x='Date',y='Total_Sales', title ='Daily Total Reven
```

In [78]:
```
# Total revenue for chips for the supermarket per year, over time, 
        # What is the chip season ? When are chips mostly bought an

cust_tran_df['Year'] = cust_tran_df['Date'].apply(lambda date: str(
cust_tran_df['Month'] = cust_tran_df['Date'].apply(lambda date: str
print(cust_tran_df['Date'].describe()) # The data provided represen

#cust_tran_df.groupby(['Year','Month']).sum()['Total_Sales'].plot.b
#cust_tran_df.groupby(['Year','Month']).sum()['Product_Quantity'].p

# From the graphs presented there is no obvious season for product 
# Chips are bought somewhat predictably and consisently, with sligh
```

```
count                  213986
unique                    364
top       2018-12-24 00:00:00
freq                      755
first     2018-07-01 00:00:00
last      2019-06-30 00:00:00
Name: Date, dtype: object
```

In [79]:
```
# Which brand has the most products ? What is the products to sales
t_df1 = cust_tran_df[['Brand_Name','Product_Name']].drop_duplicates
t_df1.rename(columns={'Product_Name':'Unique_Products'},inplace=Tru
t_df2 = cust_tran_df.groupby('Brand_Name').sum()[['Total_Sales']]
brand_unique_product_sales_df = pd.merge(left=t_df1,right=t_df2,how
# Smiths has the most unique products for customers to chose from.
print(brand_unique_product_sales_df)
brand_unique_product_sales_df['Per_Unique_Product_Sales'] = brand_u
print(brand_unique_product_sales_df.sort_values('Per_Unique_Product
# Old El Paso has 3 unique product and makes revenue of appoximatel
# Compared to Kettle has 13 uniqe products and makes revenue of app
# There is no relationsip between having more unique products and m
```

```
                      Unique_Products  Total_Sales
Brand_Name
Smiths                             18     224654.2
Kettle                             13     390239.8
Red Rock                           12      95046.0
Doritos                             8     227629.9
Pringles                            8     177655.5
Woolworths                          7      35889.5
Natural Chip Company                5      42318.0
Thins                               5      88852.5
Ccs                                 3      18078.9
Cobs                                3      70569.8
Grain Waves                         3      51617.2
Tostitos                            3      79789.6
Sunbites                            2       9676.4
Tyrrells                            2      51647.4
                      Unique_Products  Total_Sales  Per_Unique_Pro
duct_Sales
Brand_Name
Kettle                             13     390239.8             30
018.446154
Doritos                             8     227629.9             28
453.737500
Tostitos                            3      79789.6             26
596.533333
Tyrrells                            2      51647.4             25
823.700000
Cobs                                3      70569.8             23
523.266667
Pringles                            8     177655.5             22
206.937500
Thins                               5      88852.5             17
770.500000
Grain Waves                         3      51617.2             17
205.733333
Smiths                             18     224654.2             12
480.788889
Natural Chip Company                5      42318.0              8
463.600000
Red Rock                           12      95046.0              7
920.500000
Ccs                                 3      18078.9              6
026.300000
Woolworths                          7      35889.5              5
```

```
127.071429
Sunbites                                          2       9676.4               4
838.200000
```

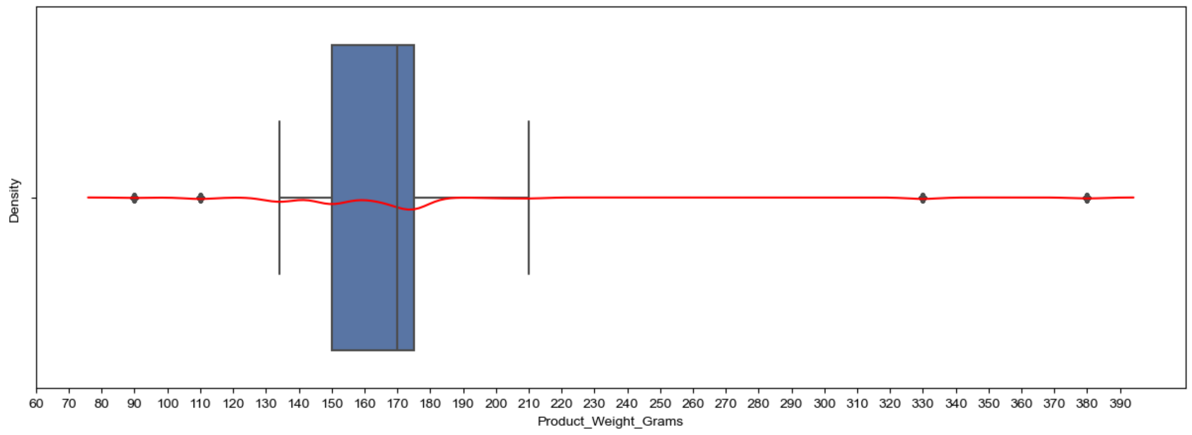
In [80]:
```
# GST assumed: 10 %.
    #E.g. if a packet of chips is sold for $4.40 (Including GST);
    # The supermarket makes: $4.00; The other 40 cents is paid as G

cust_tran_df['Tax_Paid'] = cust_tran_df['Total_Sales'].apply(lambda
# Total Tax paid by the supermarket chain.
print('Total Tax Paid', round(cust_tran_df['Tax_Paid'].sum(),2))

#Average tax paid by the store per transaction ?
cust_tran_df.groupby('Store_Number').mean()['Tax_Paid'].apply(lambd

#cust_tran_df['Total_Sales'].apply(lambda customer_paid: round(cust
```

```
Total Tax Paid 142188.12
```

Out[80]:
```
Store_Number
1      0.38
2      0.36
3      0.77
4      0.79
5      0.63
       ...
268    0.42
269    0.64
270    0.64
271    0.64
272    0.74
Name: Tax_Paid, Length: 271, dtype: float64
```

In [81]:
```
# The most weight, the least weight.
# The best product for the customer per weight.
plt.figure(figsize=(15,5))
plt.xticks(np.arange(0,400,10))
sns.set_theme(context='notebook',style='whitegrid')
sns.boxplot(cust_tran_df['Product_Weight_Grams'])
sns.kdeplot(data=cust_tran_df, x="Product_Weight_Grams",color='red'
# The most weight for a packet of chips is 380 grams.
# The least weith is 70 grams.
```

Out[81]:
```
<AxesSubplot:xlabel='Product_Weight_Grams', ylabel='Density'>
```



In [145]:
```
cust_behv_tran_df.Taxation_Id
```

Out[145]:
```
0              1
1              2
2              3
3              4
4              5
           ...  
213981    240350
213982    240378
213983    240394
213984    240480
213985    241815
Name: Taxation_Id, Length: 213986, dtype: uint32
```

In [82]:
```
# The best product for the customer based on weight and price # The
for col in ['Brand_Name','Product_Name','Product_Weight_Grams','Pro
    print(col)
    print('\t',', '.join(str(x) for x in cust_tran_df[cust_tran_df[
```

```
Brand_Name
         Grain Waves
Product_Name
         Grain Waves Sweet Chilli
Product_Weight_Grams
         210
Product_Price
         1.44
Product_Price_Per_100_Grams
         0.69
```

```
In [83]: # Which product/brnd is the most popular and which is the least ?
          # 'Popular' can be: Most sales. Most loyal customers. Most revenue
          # The interpretation of most sales and revenue isn't taken into ac
          # The most loyal customer:
                #determining loyalty in one possible way: buys again and aga
```

```
In [84]: # Which product is the most popular and which is the least ?
          # 'Popular' can be: Most sales. Most loyal customers. Most revenue
          # The interpretation of most sales and revenue isn't taken into ac
          # The most loyal customer:
                #determining loyalty in one possible way: buys again and aga

         # Repeated purchases percentage.
         t_df1 = cust_tran_df.groupby('Product_Number').count()[['Date']]
         t_df2 = (cust_tran_df.groupby(['Product_Number','Loyalty_Card_Numbe
         repeated_purchases_df = pd.merge(left = t_df1,right = t_df2, how='i
         repeated_purchases_percentage_series = repeated_purchases_df['Repea
         repeated_purchases_percentage_df = pd.DataFrame(repeated_purchases_
         product_num_brand_name_df = cust_tran_df[['Product_Number','Product

         product_repeated_purchase_df = pd.merge(left=repeated_purchases_per
         product_repeated_purchase_df.head(10)
```

Out[84]:

	Product_Number	Repeated_Purchases_Percentage	Product_Name
0	42	3.152310	Doritos Corn Chips Mexican Jalapeno
1	24	3.125987	Grain Waves Sweet Chilli
2	89	3.000000	Kettle Sweet Chilli and Sour Cream
3	33	2.967268	Cobs Popd Sweet Chilli and Sour Cream Chips
4	70	2.898551	Tyrrells Crisps Lightly Salted
5	60	2.868069	Kettle Tortilla Chips Feta and Garlic
6	39	2.857143	Smiths Crinkle Cut Tomato Salsa
7	16	2.783860	Smiths Crinkle Chips Salt and Vinegar
8	62	2.761721	Pringles Mystery Flavour
9	32	2.741885	Kettle Sea Salt and Vinegar

```
In [85]: # Which brand is the most popular and which is the least ?
          # 'Popular' can be: Most sales. Most loyal customers. Most revenue
          # The interpretation of most sales and revenue isn't taken into ac
          # The most loyal customer:
                #determining loyalty in one possible way: buys again and aga
          # Similar as above.
          # Different columns to use.
          # TO DO: Later.
```

In [86]:
```
# Who bought more than one product type in a single order ?
more_than_one_product_type_in_order_df = cust_tran_df[cust_tran_df[
print('The following customers bought more than one product/type of
more_than_one_product_type_in_order_df.head()
```

```
The following customers bought more than one product/type of chips
in a single order:
         [  7364  12301  16427 ... 248338 259056 265467]
 1049 Customers
```

Out[86]:

	Date	Store_Number	Loyalty_Card_Number	Taxation_Id	Product_Number	Product_Na
376	2019-01-10	7	7364	7739	50	Tostitos Lig Sa
377	2019-01-10	7	7364	7739	20	Doritos Che Supre
418	2018-10-18	12	12301	10982	50	Tostitos Lig Sa
419	2018-10-18	12	12301	10982	93	Doritos C Chips South Chic
475	2018-09-08	16	16427	14546	99	Pring Southern F Chic

In [87]:
```
# How many products are bought on a day on average in the store fro
avg_product_qnt_store_df = cust_tran_df.groupby('Store_Number').mea
print('Average product quantity in all stores:',list(map(int,avg_pr
avg_product_qnt_store_df
```

```
Average product quantity in all stores: [1, 2]
```

Out[87]:

	Avg_Product_Quantity_Bought_By_Customers
Store_Number	
1	1.0
2	1.0
3	2.0
4	2.0
5	2.0
...	...
268	1.0
269	2.0
270	2.0
271	2.0
272	2.0

271 rows × 1 columns

In [88]:
```
cust_behv_df.head(1)
```

Out[88]:

	Loyalty_Card_Number	Life_Stage	Card_Subscription	Age_Group	Relationship_Type
0	1000	Young Singles/Couples	Premium	Young	Singles/Couples

In [89]:
```
cust_tran_df.head(1)
```

Out[89]:

	Date	Store_Number	Loyalty_Card_Number	Taxation_Id	Product_Number	Product_Nam
0	2018-10-17	1	1000	1	5	Natural Chi Company Se Sa

In [90]:
```
# The column Loyalty_card_number is a common column.
# The Loyalty card number is a identifier as shown above, it identi
# Hence, merging will occur on this column.
# The resulting merged tabular data will showcase the transactions
# A single transation can be identified via the date, the product n
```

In [91]:
```
cust_behv_tran_df = pd.merge(cust_behv_df,cust_tran_df, on=['Loyalt
cust_behv_tran_df.head()
```

Out[91]:

	Loyalty_Card_Number	Life_Stage	Card_Subscription	Age_Group	Relationship_Type
0	1000	Young Singles/Couples	Premium	Young	Singles/Couples
1	1002	Young Singles/Couples	Mainstream	Young	Singles/Couples
2	1003	Young Families	Budget	Young	Families
3	1003	Young Families	Budget	Young	Families
4	1004	Older Singles/Couples	Mainstream	Older	Singles/Couples

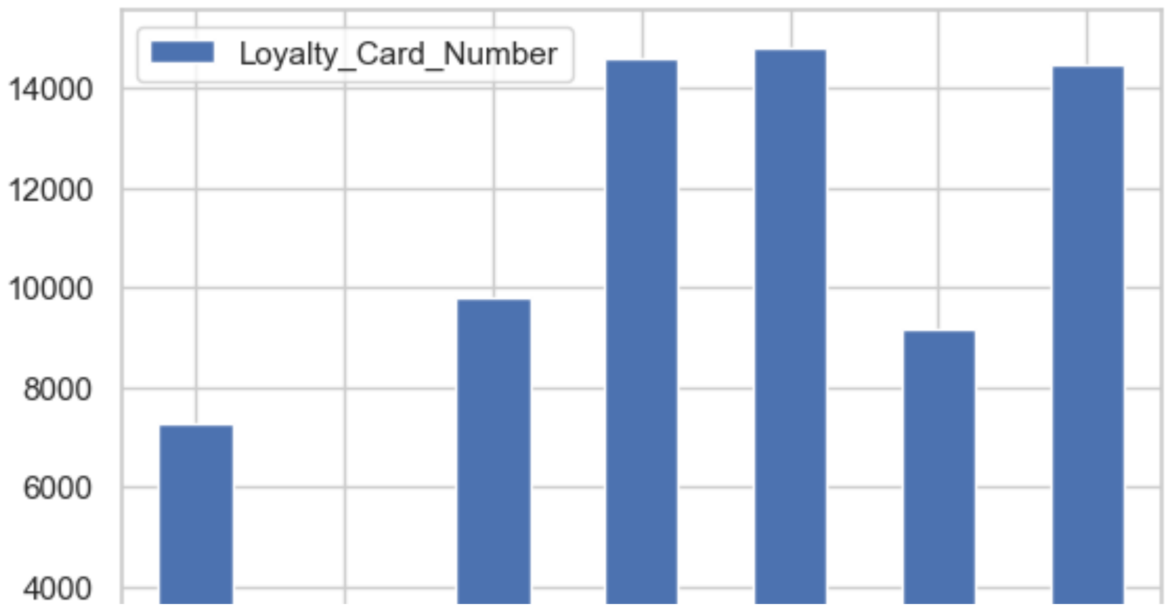
In [111]:
```
# Saving datasets cleaned.
def save_to_local_disk(path,data_frame,dir_name,file_name):
    curr_path = os.getcwd()
    path = os.path.expandvars(os.path.expanduser(path))
    if not os.path.isabs(path):
        path = os.path.abspath(path)
    if os.path.exists(path):
        os.chdir(path)
        if not os.path.exists(os.path.join(path,dir_name)):
            os.mkdir(dir_name)
        os.chdir(dir_name)
        data_frame.to_csv(f'{file_name}.csv')
        data_frame.to_json(f'{file_name}.json')
        print('Saved in directory "' + dir_name + '" at path ' + pa
    else:
        print('Path Doesn\'t exist:' , path)
    os.chdir(curr_path)

save_to_local_disk('.',cust_behv_tran_df,'quantium_dataset_processe

#cust_behv_tran_df.to_csv('')
```

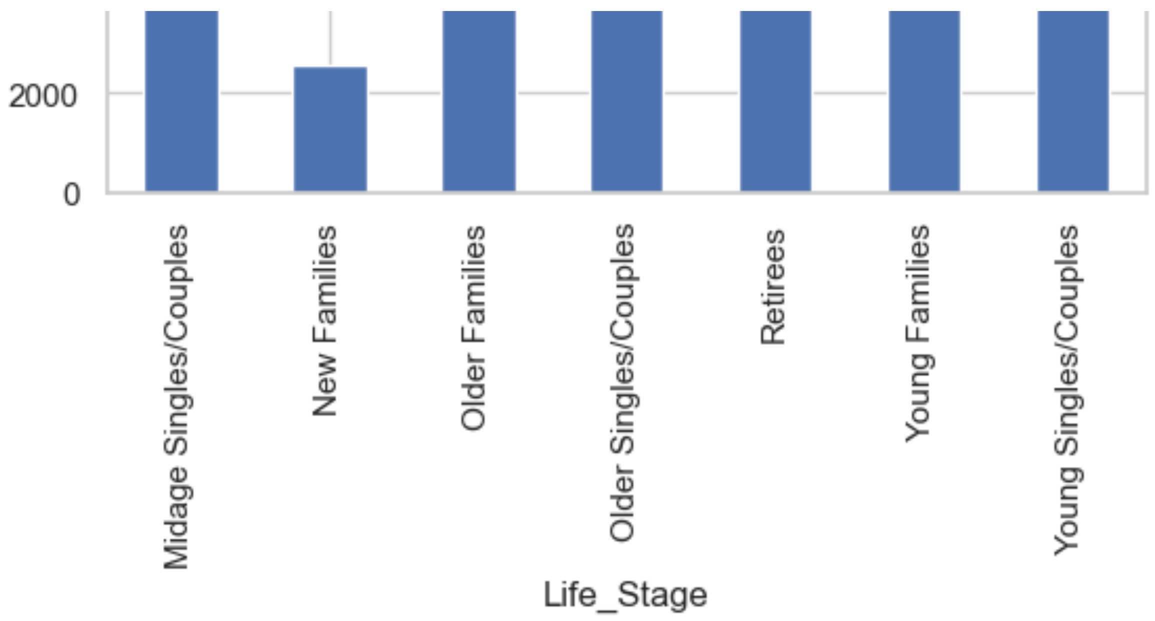
In [93]:
```

```

```
 # How do group segments by age group ; by card subscription; by re
#cust_behv_tran_df.groupby(['Life_Stage','Product_Quantity']).count
#cust_behv_tran_df.groupby(['Life_Stage','Product_Name']).count()
#cust_behv_tran_df.groupby(['Life_Stage','Brand_Name']).count()
#cust_behv_tran_df.groupby(['Life_Stage','Product_Weight_Grams']).c
#sns.boxplot(data=cust_behv_tran_df,y='Life_Stage',x='Product_Price
#sns.boxplot(data=cust_behv_tran_df,y='Card_Subscription',x='Produc
#sns.boxplot(data=cust_behv_tran_df,y='Card_Subscription',x='Produc
#cust_behv_tran_df.groupby(['Life_Stage']).sum()['Total_Sales'].plo
#cust_behv_tran_df.groupby(['Life_Stage']).sum()['Total_Sales'].plo
#cust_behv_tran_df.groupby('Date').count()['Life_Stage'].plot.line(

# How many customers are in each segment ?
    #cust_behv_tran_df[['Loyalty_Card_Number','Life_Stage']].drop_d
    # A customer can have a loyalty card but never buy a product or
    # Hence, the cust_behv_dataset will be utilised.
    # It turns out, all customers with loyalty cards bought product
num_cust_segment_df = cust_behv_df[['Loyalty_Card_Number','Life_Sta
print(num_cust_segment_df)
num_cust_segment_df.plot(kind='bar')

#cust_behv_tran_df[['Loyalty_Card_Number','Life_Stage']].drop_dupli


#.groupby(['Life_Stage'])

# Analysis, self explanatory.
```

```
                        Loyalty_Card_Number
Life_Stage
Midage Singles/Couples                 7275
New Families                           2549
Older Families                         9780
Older Singles/Couples                 14609
Retirees                              14805
Young Families                         9178
Young Singles/Couples                 14441
```

Out[93]:
```
<AxesSubplot:xlabel='Life_Stage'>
```




2000
0
Midage Singles/Couples
New Families
Older Families
Older Singles/Couples
Retirees
Young Families
Young Singles/Couples
Life_Stage

In [94]:
```
# Who drives the most sales ? # Customer lifestage and card subscri
cust_behv_tran_df.groupby(['Life_Stage', 'Card_Subscription']).sum(
```

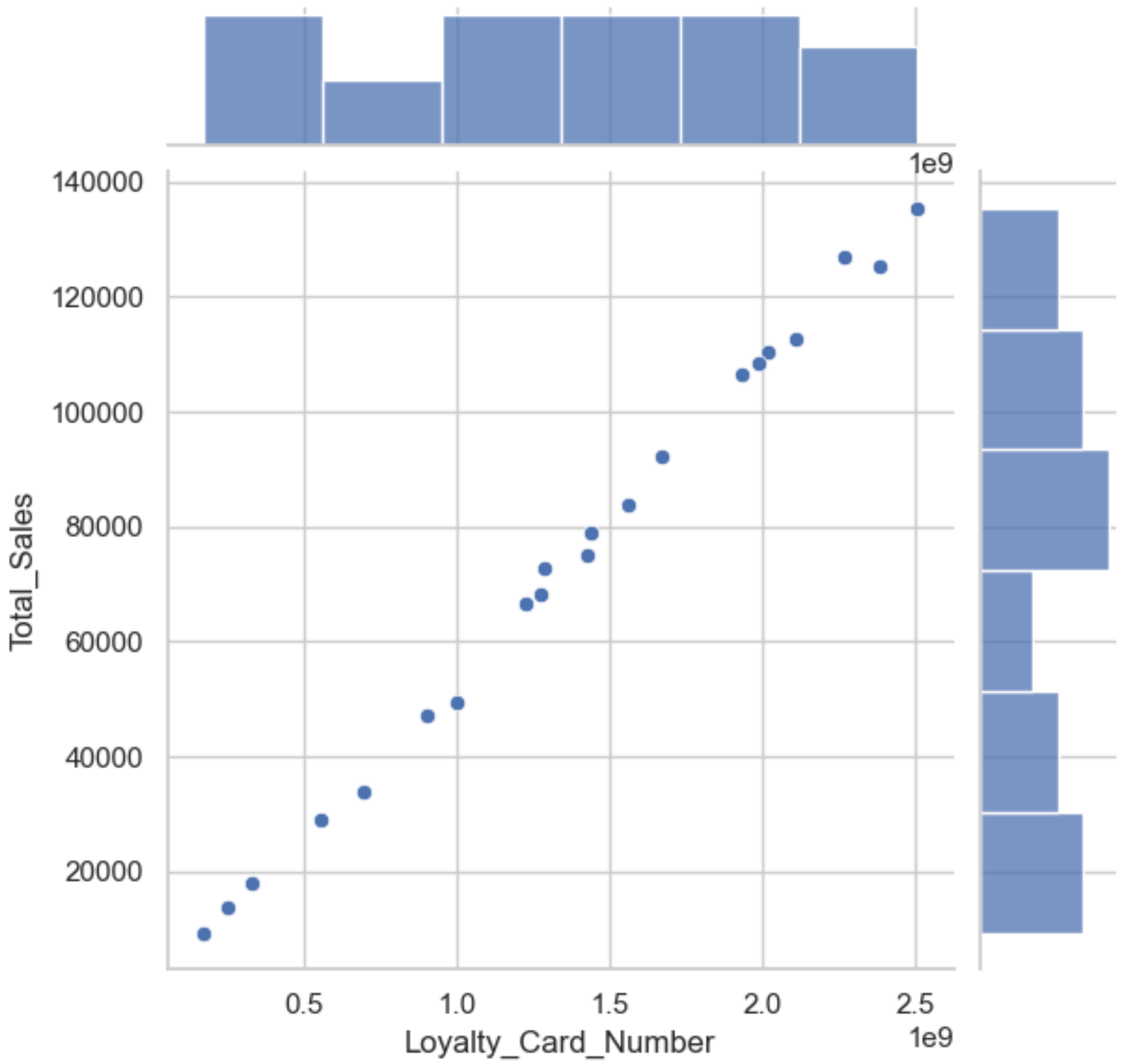
Out[94]:

		Total_Sales
Life_Stage	**Card_Subscription**	
Older Families	**Budget**	135381.45
Young Singles/Couples	**Mainstream**	126972.70
Retirees	**Mainstream**	125293.35
Young Families	**Budget**	112533.75
	Budget	110527.20
Older Singles/Couples	**Mainstream**	108546.00
	Premium	106633.95
Retirees	**Budget**	92105.20
Older Families	**Mainstream**	83877.75
Retirees	**Premium**	78942.55
Young Families	**Mainstream**	74873.05
Midage Singles/Couples	**Mainstream**	72775.35
Young Families	**Premium**	68069.40
Older Families	**Premium**	66566.70
Young Singles/Couples	**Budget**	49487.70
Midage Singles/Couples	**Premium**	47213.25
Young Singles/Couples	**Premium**	33805.20
Midage Singles/Couples	**Budget**	28865.10
	Budget	17952.85
New Families	**Mainstream**	13879.70
	Premium	9362.50

```
In [95]: #Let's see if the higher sales are due to there being more customer
         cust_chips_df = cust_behv_tran_df.groupby(['Life_Stage', 'Card_Subs
         print(cust_chips_df.head())
         sns.jointplot(data=cust_chips_df,x='Loyalty_Card_Number', y = 'Tota
         # The more the customers the more the total_sales, as illustrated b
```

```
                                              Loyalty_Card_Number  Tota
l_Sales
Life_Stage             Card_Subscription
Older Families         Budget                          2507718248    13
5381.45
Young Singles/Couples Mainstream                      2269981535    12
6972.70
Retirees               Mainstream                      2384413801    12
5293.35
Young Families         Budget                          2106929972    11
2533.75
Older Singles/Couples Budget                          2019530236    11
0527.20
```

```
Out[95]: <seaborn.axisgrid.JointGrid at 0x7f7ec704cc70>
```



```
In [96]:
```

```
# Average number of units per customer by LIFESTAGE and PREMIUM_CUS
cust_df1 = cust_behv_tran_df.groupby(['Life_Stage','Card_Subscripti
cust_df1['life_stage_card_sub'] =  [life_stage + ' ' + card_sub for
cust_df1 = cust_df1.set_index('life_stage_card_sub')[['Product_Quan

cust_df2 = cust_behv_tran_df.groupby(['Life_Stage','Card_Subscripti
cust_df2['life_stage_card_sub'] =  [life_stage + ' ' + card_sub for
cust_df2 = cust_df2.set_index('life_stage_card_sub')[['Loyalty_Card

cust_df3 = pd.merge(left=cust_df1,right=cust_df2,on='life_stage_car
cust_df3['Avg_Qnty_Per_Customer'] = round(cust_df3['Product_Quantit
cust_df3.sort_values('Avg_Qnty_Per_Customer',ascending=False)

# Older families and young families spend more per average per cust
```

Out[96]:

life_stage_card_sub	Product_Quantity	Loyalty_Card_Number	Avg_Qnty_Per_Customer
Older Families Premium	18008.0	9058	1.99
Older Families Mainstream	22443.0	11511	1.95
Young Families Premium	18151.0	9379	1.94
Young Families Mainstream	20174.0	10387	1.94
Young Families Budget	30003.0	15449	1.94
Older Families Budget	36263.0	18648	1.94
Older Singles/Couples Mainstream	28441.0	14870	1.91
Older Singles/Couples Budget	28434.0	14849	1.91
Midage Singles/Couples Mainstream	18270.0	9556	1.91
Older Singles/Couples Premium	27426.0	14327	1.91
Retirees Premium	20137.0	10592	1.90
Retirees Mainstream	32590.0	17277	1.89
Midage Singles/Couples Budget	7721.0	4080	1.89
Retirees Budget	23446.0	12379	1.89
Midage Singles/Couples Premium	12531.0	6628	1.89
New Families Premium	2405.0	1288	1.87
New Families Mainstream	3521.0	1894	1.86
New Families Budget	4555.0	2455	1.86
Young Singles/Couples	31199.0	16847	1.85

Mainstream			
Young Singles/Couples Budget	13438.0	7443	1.81
Young Singles/Couples Premium	9154.0	5069	1.81

In [114]:
```
# Let's also investigate the average price per unit chips bought fo
avg_price_df = cust_behv_tran_df.groupby(['Life_Stage','Card_Subscr
avg_price_df['Avg_Price_Packet_Chips'] = avg_price_df['Total_Sales'
avg_price_df.sort_values(by='Avg_Price_Packet_Chips',ascending=Fals
#avg_price_df.sort_values(by='Avg_Price_Packet_Chips',ascending=Fal
```

Out[114]:

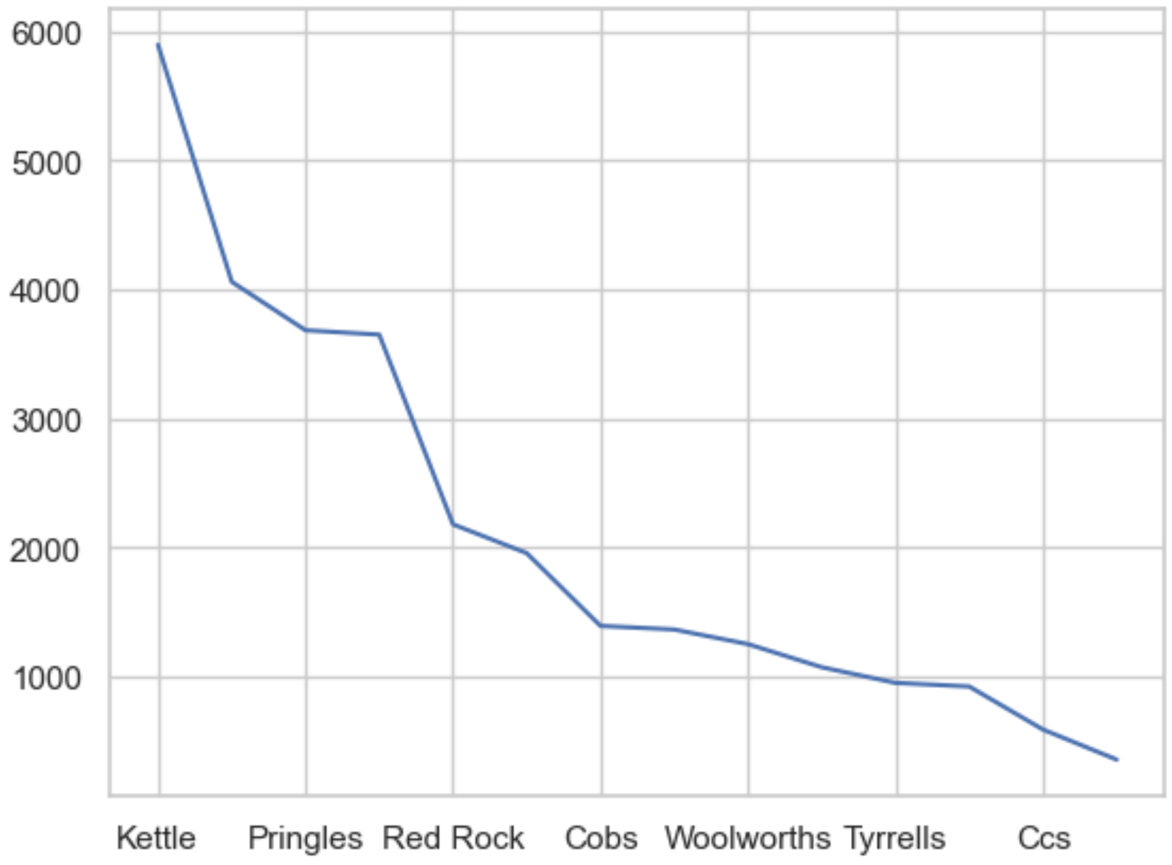
		Total_Sales	Product_Quantity	Avg_Price_Packet_Chips
Life_Stage	**Card_Subscription**			
Young Singles/Couples	Mainstream	126972.70	31199.0	4.069768
Midage Singles/Couples	Mainstream	72775.35	18270.0	3.983325
New Families	Mainstream	13879.70	3521.0	3.941977
	Budget	17952.85	4555.0	3.941350
Retirees	Budget	92105.20	23446.0	3.928397
	Premium	78942.55	20137.0	3.920274
New Families	Premium	9362.50	2405.0	3.892931
Older Singles/Couples	Premium	106633.95	27426.0	3.888061
	Budget	110527.20	28434.0	3.887149
Retirees	Mainstream	125293.35	32590.0	3.844534
Older Singles/Couples	Mainstream	108546.00	28441.0	3.816532
Midage Singles/Couples	Premium	47213.25	12531.0	3.767716
Young Families	Budget	112533.75	30003.0	3.750750
	Premium	68069.40	18151.0	3.750174
Midage Singles/Couples	Budget	28865.10	7721.0	3.738518
Older Families	Mainstream	83877.75	22443.0	3.737368
	Budget	135381.45	36263.0	3.733322
Young Families	Mainstream	74873.05	20174.0	3.711364
Older Families	Premium	66566.70	18008.0	3.696507
Young Singles/Couples	Premium	33805.20	9154.0	3.692943
	Budget	49487.70	13438.0	3.682669

In [99]:
```
# Notice that the mainstream young singles/couples and midage singl
# As seen above.
# Let's further explore this.

mainstream_midageyoung_df = cust_behv_tran_df[(cust_behv_tran_df['C
prembudg_midageyoung_df = cust_behv_tran_df[((cust_behv_tran_df['Ca
mainstream_midageyoung_df['Type'] = 'Mainstream'
prembudg_midageyoung_df['Type'] = 'Premium/Budget'
mainstream_prembudg_midageyoung_df = pd.concat([mainstream_midageyo
fig = ex.histogram(data_frame=mainstream_prembudg_midageyoung_df,x=
fig.update_layout(title_font_family="Times New Roman",title_font_si

# Use if above doesn't render. Not interactive.
#mainstream_midageyoung_Series.plot.hist()
#prembudg_midageyoung_Series.plot.hist()
```

```
In [101]: # Looking further, deep diving into a customer segment, also lookin
          # Similar process for other customer segments.
          print(cust_behv_tran_df[cust_behv_tran_df['Life_Stage'] == 'Young S
          cust_behv_tran_df[cust_behv_tran_df['Life_Stage'] == 'Young Singles
          # Kettle is the brand preffered by young singles/couples.
```

```
Kettle                    5893
Smiths                    4059
Pringles                  3684
Doritos                   3650
Red Rock                  2182
Thins                     1959
Cobs                      1396
Tostitos                  1368
Woolworths                1255
Grain Waves               1076
Tyrrells                   955
Natural Chip Company       927
Ccs                        594
Sunbites                   361
Name: Brand_Name, dtype: int64
```

```
Out[101]: <AxesSubplot:>
```



```
In [102]: # Using the aporiri algorithm for the young singles/couples to dete
          # which brand a customer of this lifestage buys frequently together
              # Step 1: Setup
              # Step 2: Aporiri algorithm.
```

```
In [103]: # Step 1: Setup
yng_df = cust_behv_tran_df[cust_behv_tran_df['Life_Stage'] == 'Youn
print('Target segment:', 'Young Singles/Couples')

# Transactions of buying a single brand, in an order.
index_1_only = yng_df.groupby(['Loyalty_Card_Number','Date'])['Bran
print('Number of transaction 1 brand :', len(index_1_only))

# Transactions of buying 2 brands together, in an order
index_2_together = yng_df.groupby(['Loyalty_Card_Number','Date'])['
print('Number of transaction 2 brands together:', len(index_2_toget

# Transactions of buying more than 2 brand together, in an order
index_2_or_more = yng_df.groupby(['Loyalty_Card_Number','Date'])['B
print('Number of transaction more than 2 brands bought together:',l

out_index_1,out_index_2  = [],[]
for index_in,index_out in [(index_1_only,out_index_1),(index_2_toge
                   for lty_num, date in index_in:
                           index_out.append(list(yng_df[(yng_df['Lo
brand_1_only_df = pd.DataFrame(data=out_index_1,columns=['brand1onl
brands_2_df = pd.DataFrame(data=sorted(out_index_2),columns=['brand

print(brand_1_only_df.head(),brands_2_df.head(),sep='\n')
```

```
Target segment: Young Singles/Couples
Number of transaction 1 brand : 29209
Number of transaction 2 brands together: 75
Number of transaction more than 2 brands bought together: 0
             brand1only
0  Natural Chip Company
1              Red Rock
2              Red Rock
3               Doritos
4                Kettle
  brand1                brand2
0    Ccs               Doritos
1    Ccs                 Thins
2    Ccs              Tyrrells
3   Cobs               Doritos
4   Cobs  Natural Chip Company
```

In [104]:
```
# Step 2: Aporiri algorithm, somewhat, used.
# Support threshold determined from prior domain knowdlege and data
# Here, support is set to: 2
print(brand_1_only_df.value_counts())

print(len(brands_2_df))
brands_2_df['i'] = 1
brands_2_df.groupby(['brand1','brand2']).count()[brands_2_df.groupb
# Strong association brands.
```

```
brand1only
Kettle                  5877
Smiths                  4037
Pringles                3669
Doritos                 3632
Red Rock                2165
Thins                   1948
Cobs                    1386
Tostitos                1361
Woolworths              1248
Grain Waves             1071
Tyrrells                 950
Natural Chip Company     919
Ccs                      588
Sunbites                 358
dtype: int64
75
```

Out[104]:

		i
brand1	**brand2**	
Smiths	**Natural Chip Company**	4.0
Doritos	**Pringles**	3.0
Kettle	**Pringles**	3.0
Red Rock	**Kettle**	3.0
Doritos	**Red Rock**	2.0
	Cobs	2.0
Kettle	**Smiths**	2.0
	Tostitos	2.0
Pringles	**Doritos**	2.0
	Thins	2.0
Red Rock	**Tostitos**	2.0
Smiths	**Pringles**	2.0
	Red Rock	2.0
Woolworths	**Red Rock**	2.0

In [105]:
```
cust_behv_tran_df
```

Out[105]:

	Loyalty_Card_Number	Life_Stage	Card_Subscription	Age_Group	Relationship_
0	1000	Young Singles/Couples	Premium	Young	Singles/Co
1	1002	Young Singles/Couples	Mainstream	Young	Singles/Co
2	1003	Young Families	Budget	Young	Far
3	1003	Young Families	Budget	Young	Far
4	1004	Older Singles/Couples	Mainstream	Older	Singles/Co
...	...	...	...	...	
213981	2370651	Midage Singles/Couples	Mainstream	Midage	Singles/Co
213982	2370701	Young Families	Mainstream	Young	Far
213983	2370751	Young Families	Premium	Young	Far
213984	2370961	Older Families	Budget	Older	Far
213985	2373711	Young Singles/Couples	Mainstream	Young	Singles/Co

213986 rows × 19 columns

In [106]:
```
# Other notions, can be also looked into for customer segment Young
yng_df['Product_Weight_Grams'].value_counts()

# Out customer segment preferes a moderate-low package size. Not la
```

Out[106]:
```
175    8759
150    5743
134    3684
170    2751
165    1890
110    1396
330    1366
380     955
210     913
200     538
135     452
160     388
90      361
180     163
Name: Product_Weight_Grams, dtype: int64
```

In [107]:
```


# Who spends the most on chips (total sales), describing customers
spend_most_chips_df = cust_behv_tran_df.groupby(['Life_Stage']).sum
print(spend_most_chips_df)
spend_most_chips_df.plot.bar()
print('\n',spend_most_chips_df.index[0] + " Spend's Most On Chips."

# and how premium their general purchasing behaviour is ?
purchasing_behv_most_spender_df = cust_behv_tran_df[cust_behv_tran_
print(purchasing_behv_most_spender_df)
purchasing_behv_most_spender_df.apply(lambda subscription_type: sub
# 32.87% premium(nearly a third) is thier total sales general purch


```

```
Life_Stage
Older Singles/Couples     325707.15
Retirees                  296341.10
Older Families            285825.90
Young Families            255476.20
Young Singles/Couples     210265.60
Midage Singles/Couples    148853.70
New Families               41195.05
Name: Total_Sales, dtype: float64

 Older Singles/Couples Spend's Most On Chips.
Card_Subscription
Budget        110527.20
Mainstream    108546.00
Premium       106633.95
Name: Total_Sales, dtype: float64
```

Out[107]:
```
Card_Subscription
Budget        33.934533
```