QuantiumPart1

August 25, 2024

1 Data Preparation

Objectives:

To perform customer segmentation on transaction and customer data about chips purchases for a client in retail.

Develop metrics and examine sales drivers to gain insights into overall sales performance.

Create visualisations and prepare findings to formulate a clear recommendation for the client's strategy.

[252]:
```
import pandas as pd
from numpy import where
from matplotlib import pyplot
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import randint
```

[260]:
```
transaction_data_df = pd.read_csv('QVI_transaction_data.csv')
purchase_behaviour_df = pd.read_csv('QVI_purchase_behaviour.csv')
```

[261]:
```
transaction_data_df.head(), purchase_behaviour_df.head()
```

[261]:
```
(    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  ,
    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)
```

[262]:
```
# Check for missing values
print(purchase_behaviour_df.info())
print(transaction_data_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72637 entries, 0 to 72636
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   LYLTY_CARD_NBR    72637 non-null  int64 
 1   LIFESTAGE         72637 non-null  object
 2   PREMIUM_CUSTOMER  72637 non-null  object
dtypes: int64(1), object(2)
memory usage: 1.7+ MB
None
<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
None
```

[263]:
```
# Checking for null values
purchase_behaviour_df.isnull().sum(), transaction_data_df.isnull().sum()

# There are no null values.
```

[263]:
```
(LYLTY_CARD_NBR      0
 LIFESTAGE           0
 PREMIUM_CUSTOMER    0
 dtype: int64,
 DATE              0
 STORE_NBR         0
```

```
LYLTY_CARD_NBR    0
TXN_ID            0
PROD_NBR          0
PROD_NAME         0
PROD_QTY          0
TOT_SALES         0
dtype: int64)
```

[264]:
```
# Convert the DATE column in the transaction_data_df dataframe to a date format
transaction_data_df['DATE'] = pd.to_datetime(transaction_data_df['DATE'],␣
↪origin='1899-12-30', unit='D')
```

[265]:
```
# Engineering a feature that takes the pack size from the product name, this␣
↪will be useful later
transaction_data_df['PACK_SIZE'] = transaction_data_df['PROD_NAME'].str.
↪extract('(\d+)').astype(float)
transaction_data_df.head()
```

```
<>:2: SyntaxWarning: invalid escape sequence '\d'
<>:2: SyntaxWarning: invalid escape sequence '\d'
C:\Users\Alden\AppData\Local\Temp\ipykernel_20268\3763230065.py:2:
SyntaxWarning: invalid escape sequence '\d'
  transaction_data_df['PACK_SIZE'] =
transaction_data_df['PROD_NAME'].str.extract('(\d+)').astype(float)
```

[265]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PACK_SIZE
0	2018-10-17	1	1000	1	5	Natural Chip Compny SeaSalt175g	2	6.0	175.0
1	2019-05-14	1	1307	348	66	CCs Nacho Cheese 175g	3	6.3	175.0
2	2019-05-20	1	1343	383	61	Smiths Crinkle Cut Chips Chicken 170g	2	2.9	170.0
3	2018-08-17	2	2373	974	69	Smiths Chip Thinly S/Cream&Onion 175g	5	15.0	175.0
4	2018-08-18	2	2426	1038	108	Kettle Tortilla ChpsHny&Jlpno Chili 150g	3	13.8	150.0

[266]:
```
# As we can see, some products are not actually chips, some of them are salsa␣
↪or dips
transaction_data_df['PROD_NAME'].value_counts()
```

[266]:
```
PROD_NAME
Kettle Mozzarella   Basil & Pesto 175g      3304
Kettle Tortilla ChpsHny&Jlpno Chili 150g    3296
Cobs Popd Swt/Chlli &Sr/Cream Chips 110g    3269
Tyrrells Crisps     Ched & Chives 165g      3268
```

```
Cobs Popd Sea Salt  Chips 110g              3265
                                           …
RRD Pc Sea Salt     165g                   1431
Woolworths Medium   Salsa 300g             1430
NCC Sour Cream &    Garden Chives 175g     1419
French Fries Potato Chips 175g             1418
WW Crinkle Cut      Original 175g          1410
Name: count, Length: 114, dtype: int64
```

[267]:
```
# Filtering out those products that are not chips

# Convert product names to lowercase for consistent text processing
transaction_data_df['PROD_NAME'] = transaction_data_df['PROD_NAME'].str.lower()

# Filter out product names that contain keywords related to chips
#non_chip_keywords = ['nuts', 'popcorn', 'chocolate', 'pretzels', 'cheese',␣
 ↪'salsa', 'corn', 'dip']

non_chip_keywords = pd.DataFrame(
    [word for name in transaction_data_df['PROD_NAME'].unique() for word in␣
 ↪name.split()],
    columns=['words'])

# Create a mask to identify non-chip products
non_chip_mask = transaction_data_df['PROD_NAME'].apply(lambda x: any(keyword in␣
 ↪x for keyword in non_chip_keywords))

# Filter out non-chip products
non_chip_products = transaction_data_df[non_chip_mask]

# Returns product names that are chips
transaction_data_df = transaction_data_df[~non_chip_mask]
transaction_data_df
```

[267]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	\
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	
…	…	…	…	…	…	
264831	2019-03-09	272	272319	270088	89	
264832	2018-08-13	272	272358	270154	74	
264833	2018-11-06	272	272379	270187	51	
264834	2018-12-27	272	272379	270188	42	
264835	2018-09-22	272	272380	270189	74	

```
                                      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
…                                         …       …         …
264831   kettle sweet chilli and sour cream 175g         2       10.8
264832             tostitos splash of  lime 175g         1        4.4
264833                  doritos mexicana    170g         2        8.8
264834   doritos corn chip mexican jalapeno 150g         2        7.8
264835             tostitos splash of  lime 175g         2        8.8

        PACK_SIZE
0           175.0
1           175.0
2           170.0
3           175.0
4           150.0
…           …
264831      175.0
264832      175.0
264833      170.0
264834      150.0
264835      175.0

[264836 rows x 9 columns]
```

[269]:
```python
# Removing special characters and numbers in prodcut names for readability

import re
from collections import Counter

# Remove digits and special characters from product names
transaction_data_df['PROD_NAME'] = transaction_data_df['PROD_NAME'].
 ↪apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x))
transaction_data_df['PROD_NAME'] = transaction_data_df['PROD_NAME'].str.
 ↪slice(0, -1)

# Split the cleaned product names into individual words
all_words = ' '.join(transaction_data_df['PROD_NAME']).split()

# Count the frequency of each word
word_freq = Counter(all_words)

# Convert the Counter object to a DataFrame for easy sorting and display
word_freq_df = pd.DataFrame(word_freq.items(), columns=['Word', 'Frequency'])
```

```
# Sort the words by frequency of occurrence
word_freq_df = word_freq_df.sort_values(by='Frequency', ascending=False)

# Display the sorted words by frequency
print(word_freq_df.head(20))
```

```
         Word  Frequency
10       chips      49770
14      kettle      41288
7       smiths      28860
6       cheese      27890
66    pringles      25102
31     doritos      24962
25        salt      24719
8      crinkle      23960
32        corn      22063
45    original      21560
9          cut      20754
1         chip      18645
11     chicken      18577
21       salsa      18094
59         sea      14145
42       thins      14075
30      chilli      13895
35        sour      13882
111     crisps      12607
26     vinegar      12402
```

[271]:
```
# The product name is now clean
transaction_data_df
```

[271]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	\
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	
…	…	…	…	…	…	
264831	2019-03-09	272	272319	270088	89	
264832	2018-08-13	272	272358	270154	74	
264833	2018-11-06	272	272379	270187	51	
264834	2018-12-27	272	272379	270188	42	
264835	2018-09-22	272	272380	270189	74	

	PROD_NAME	PROD_QTY	TOT_SALES	PACK_SIZE
0	natural chip compny seasalt	2	6.0	175.0
1	ccs nacho cheese	3	6.3	175.0

```
2        smiths crinkle cut  chips chicken          2        2.9      170.0
3          smiths chip thinly  screamonion          5       15.0      175.0
4       kettle tortilla chpshnyjlpno chili          3       13.8      150.0
…                                       …       …         …         …
264831  kettle sweet chilli and sour cream          2       10.8      175.0
264832            tostitos splash of  lime          1        4.4      175.0
264833                 doritos mexicana               2        8.8      170.0
264834  doritos corn chip mexican jalapeno          2        7.8      150.0
264835            tostitos splash of  lime          2        8.8      175.0

[264836 rows x 9 columns]
```

[272]:
```
# Looking for anomalies or outliers
transaction_data_df.describe()
```

[272]:
```
                                DATE     STORE_NBR  LYLTY_CARD_NBR  \
count                         264836  264836.00000    2.648360e+05
mean   2018-12-30 00:52:12.879215616     135.08011    1.355495e+05
min              2018-07-01 00:00:00       1.00000    1.000000e+03
25%              2018-09-30 00:00:00      70.00000    7.002100e+04
50%              2018-12-30 00:00:00     130.00000    1.303575e+05
75%              2019-03-31 00:00:00     203.00000    2.030942e+05
max              2019-06-30 00:00:00     272.00000    2.373711e+06
std                              NaN      76.78418    8.057998e+04

             TXN_ID       PROD_NBR       PROD_QTY      TOT_SALES  \
count  2.648360e+05  264836.000000  264836.000000  264836.000000
mean   1.351583e+05      56.583157       1.907309       7.304200
min    1.000000e+00       1.000000       1.000000       1.500000
25%    6.760150e+04      28.000000       2.000000       5.400000
50%    1.351375e+05      56.000000       2.000000       7.400000
75%    2.027012e+05      85.000000       2.000000       9.200000
max    2.415841e+06     114.000000     200.000000     650.000000
std    7.813303e+04      32.826638       0.643654       3.083226

          PACK_SIZE
count  264836.000000
mean      182.427004
min        70.000000
25%       150.000000
50%       170.000000
75%       175.000000
max       380.000000
std        64.327196
```

[273]:
```
# The maximum product quantity is 200 which is an outlier so we'll take a look␣
↪at that
```

```
filtered_df = transaction_data_df[transaction_data_df['PROD_QTY'] == 200]
filtered_df
```

[273]:
```
            DATE  STORE_NBR  LYLTY_CARD_NBR  TXN_ID  PROD_NBR  \
69762 2018-08-19        226          226000  226201         4
69763 2019-05-20        226          226000  226210         4

                        PROD_NAME  PROD_QTY  TOT_SALES  PACK_SIZE
69762  dorito corn chp     supreme       200      650.0      380.0
69763  dorito corn chp     supreme       200      650.0      380.0
```

[275]:
```
# Removing that purchase made by the same loyalty card number as that won't␣
 ↪affect our analysis
transaction_data_df = transaction_data_df[~((transaction_data_df['PROD_QTY'] ==␣
 ↪200) & (transaction_data_df['LYLTY_CARD_NBR'] == 226000))]
transaction_data_df.describe()

# Now, all features don't have any outliers
```

[275]:
```
                                DATE      STORE_NBR  LYLTY_CARD_NBR  \
count                         264834  264834.000000    2.648340e+05
mean   2018-12-30 00:52:10.292938240     135.079423    1.355488e+05
min              2018-07-01 00:00:00       1.000000    1.000000e+03
25%              2018-09-30 00:00:00      70.000000    7.002100e+04
50%              2018-12-30 00:00:00     130.000000    1.303570e+05
75%              2019-03-31 00:00:00     203.000000    2.030940e+05
max              2019-06-30 00:00:00     272.000000    2.373711e+06
std                              NaN      76.784063    8.057990e+04

             TXN_ID       PROD_NBR       PROD_QTY      TOT_SALES  \
count  2.648340e+05  264834.000000  264834.000000  264834.000000
mean   1.351576e+05      56.583554       1.905813       7.299346
min    1.000000e+00       1.000000       1.000000       1.500000
25%    6.760050e+04      28.000000       2.000000       5.400000
50%    1.351365e+05      56.000000       2.000000       7.400000
75%    2.026998e+05      85.000000       2.000000       9.200000
max    2.415841e+06     114.000000       5.000000      29.500000
std    7.813292e+04      32.826444       0.343436       2.527241

          PACK_SIZE
count  264834.000000
mean      182.425512
min        70.000000
25%       150.000000
50%       170.000000
75%       175.000000
max       380.000000
```

```
std        64.325148
```

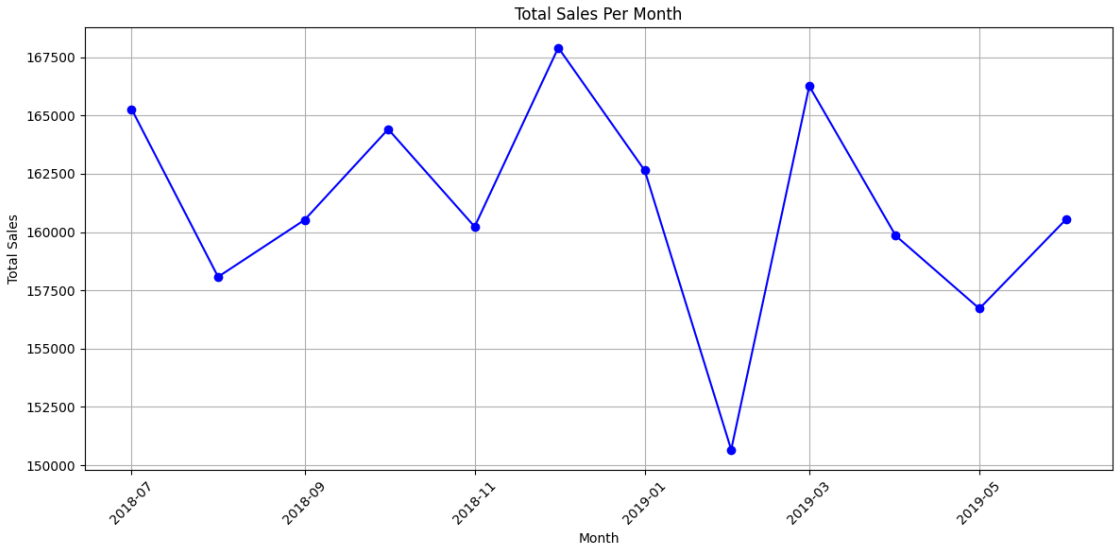
2 Data Visualisations

[277]:
```
# Looking at the date to see if there is any missing dates
start_date = '2018-07-01'
end_date = '2019-06-30'
full_date_range = pd.date_range(start=start_date, end=end_date)
full_dates_df = pd.DataFrame({'DATE': full_date_range})
full_dates_df

# There is one missing date there are only 364 observations
```

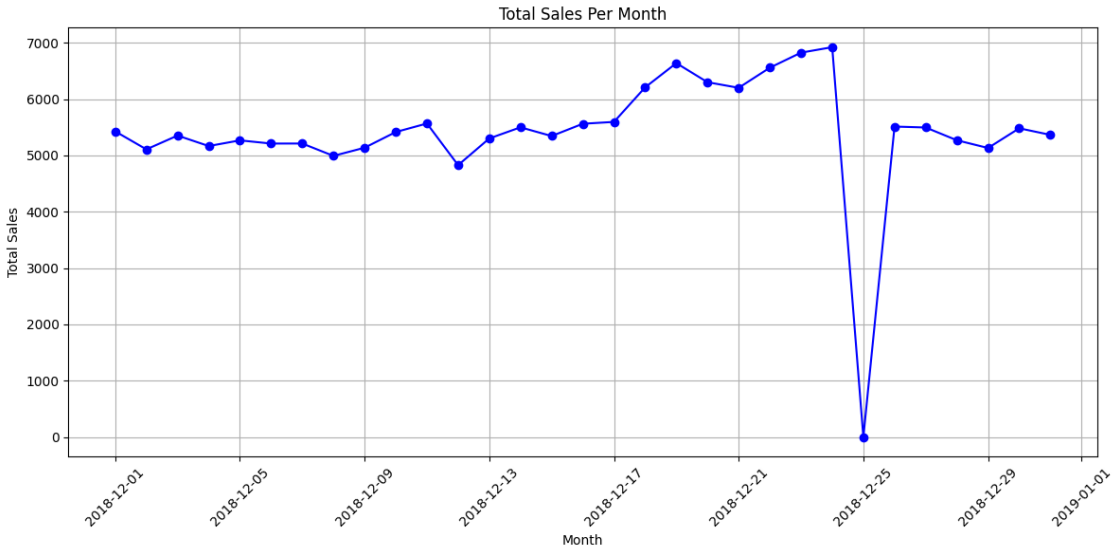
[277]:
```
           DATE
0    2018-07-01
1    2018-07-02
2    2018-07-03
3    2018-07-04
4    2018-07-05
..          …
360  2019-06-26
361  2019-06-27
362  2019-06-28
363  2019-06-29
364  2019-06-30

[365 rows x 1 columns]
```

[278]:
```
# Merging both dates from two dataframes and filling the missing dataset
new_transaction_data_df = pd.merge(full_dates_df, transaction_data_df,␣
 ↪on='DATE', how='left')
new_transaction_data_df = new_transaction_data_df.fillna(0)
new_transaction_data_df

# Plot transactions over time
total_sales_per_month = new_transaction_data_df.
 ↪groupby(new_transaction_data_df['DATE'].dt.to_period('M'))['TOT_SALES'].
 ↪sum().reset_index()
total_sales_per_month['DATE'] = total_sales_per_month['DATE'].dt.to_timestamp()
plt.figure(figsize=(12, 6))
plt.plot(total_sales_per_month['DATE'], total_sales_per_month['TOT_SALES'],␣
 ↪marker='o', linestyle='-', color='b')
plt.title('Total Sales Per Month')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.grid(True)
```

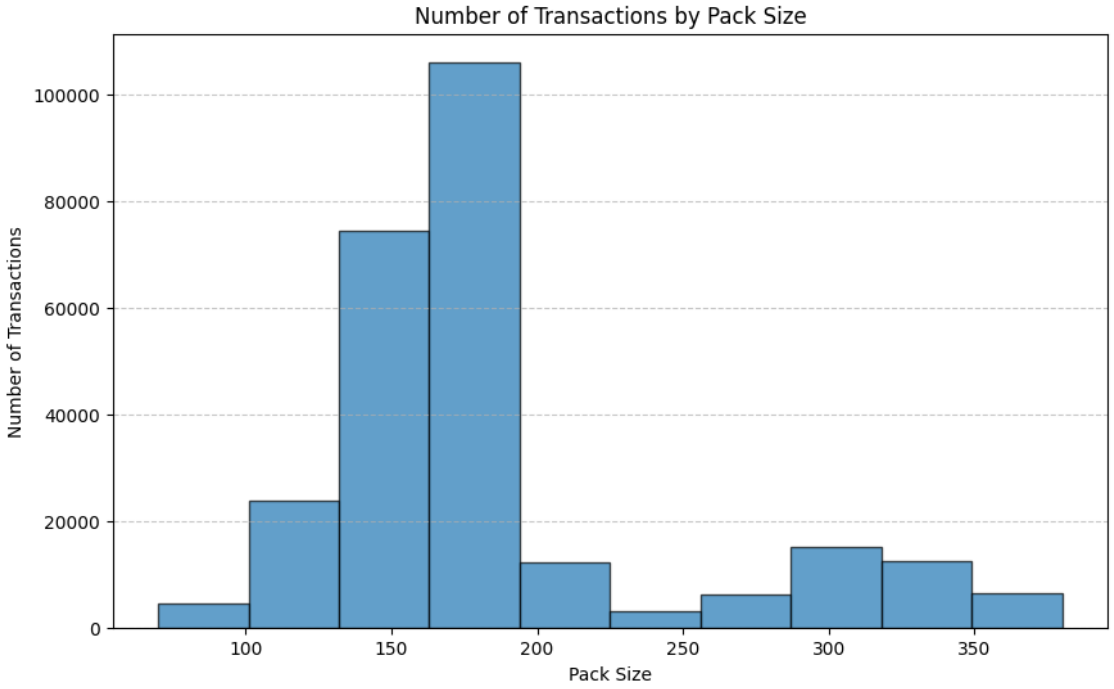
```
plt.xticks(rotation=45)
plt.tight_layout()
```



[167]:
```
# Zooming in on December to see the dip
december_df = new_transaction_data_df[new_transaction_data_df['DATE'].dt.month␣
 ↪== 12]
daily_sales_december = december_df.groupby(december_df['DATE'].dt.
 ↪to_period('D'))['TOT_SALES'].sum().reset_index()
daily_sales_december['DATE'] = daily_sales_december['DATE'].dt.to_timestamp()
plt.figure(figsize=(12, 6))
plt.plot(daily_sales_december['DATE'], daily_sales_december['TOT_SALES'],␣
 ↪marker='o', linestyle='-', color='b')
plt.title('Total Sales Per Month')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()

# The dip happened on Christmas Day which is a holiday so shops will be closed
```



[279]:
```
# Plot a histogram of the number of transactions by pack size
plt.figure(figsize=(10, 6))
plt.hist(transaction_data_df['PACK_SIZE'], bins=10, edgecolor='black', alpha=0.
↪7)
plt.title('Number of Transactions by Pack Size')
plt.xlabel('Pack Size')
plt.ylabel('Number of Transactions')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

# Seems about okay, there is a lot more purchases in pack sizes between 150-200g
```



[283]:
```
# Feature engineering a column 'BRAND' that extracts the first name from␣
 ↪product name
transaction_data_df['BRAND'] = transaction_data_df['PROD_NAME'].str.split().
 ↪str[0]
```

[284]:
```
transaction_data_df.head()
```

[284]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PACK_SIZE	BRAND
0	2018-10-17	1	1000	1	5	natural chip compny seasalt	2	6.0	175.0	natural
1	2019-05-14	1	1307	348	66	ccs nacho cheese	3	6.3	175.0	ccs
2	2019-05-20	1	1343	383	61	smiths crinkle cut chips chicken	2	2.9	170.0	
3	2018-08-17	2	2373	974	69	smiths chip thinly screamonion	5	15.0	175.0	
4	2018-08-18	2	2426	1038	108	kettle tortilla chpshnyjlpno chili	3	13.8	150.0	

```
2   smiths
3   smiths
4   kettle
```

[285]:
```
transaction_data_df['BRAND'].unique()
```

[285]:
```
array(['natural', 'ccs', 'smiths', 'kettle', 'old', 'grain', 'doritos',
       'twisties', 'ww', 'thins', 'burger', 'ncc', 'cheezels', 'infzns',
       'red', 'pringles', 'dorito', 'infuzions', 'smith', 'grnwves',
       'tyrrells', 'cobs', 'woolworths', 'french', 'rrd', 'tostitos',
       'cheetos', 'snbts', 'sunbites'], dtype=object)
```

[286]:
```
# Some brands similar to others
# For readability purposes, we will update the 'BRAND' column where the value␣
 ↪is "RED" to "RRD" for example
transaction_data_df.loc[transaction_data_df['BRAND'] == "red", 'BRAND'] = "rrd"
transaction_data_df.loc[transaction_data_df['BRAND'] == "snbts", 'BRAND'] =␣
 ↪"sunbites"
transaction_data_df.loc[transaction_data_df['BRAND'] == "infzns", 'BRAND'] =␣
 ↪"infuzions"
transaction_data_df.loc[transaction_data_df['BRAND'] == "dorito", 'BRAND'] =␣
 ↪"doritos"
```

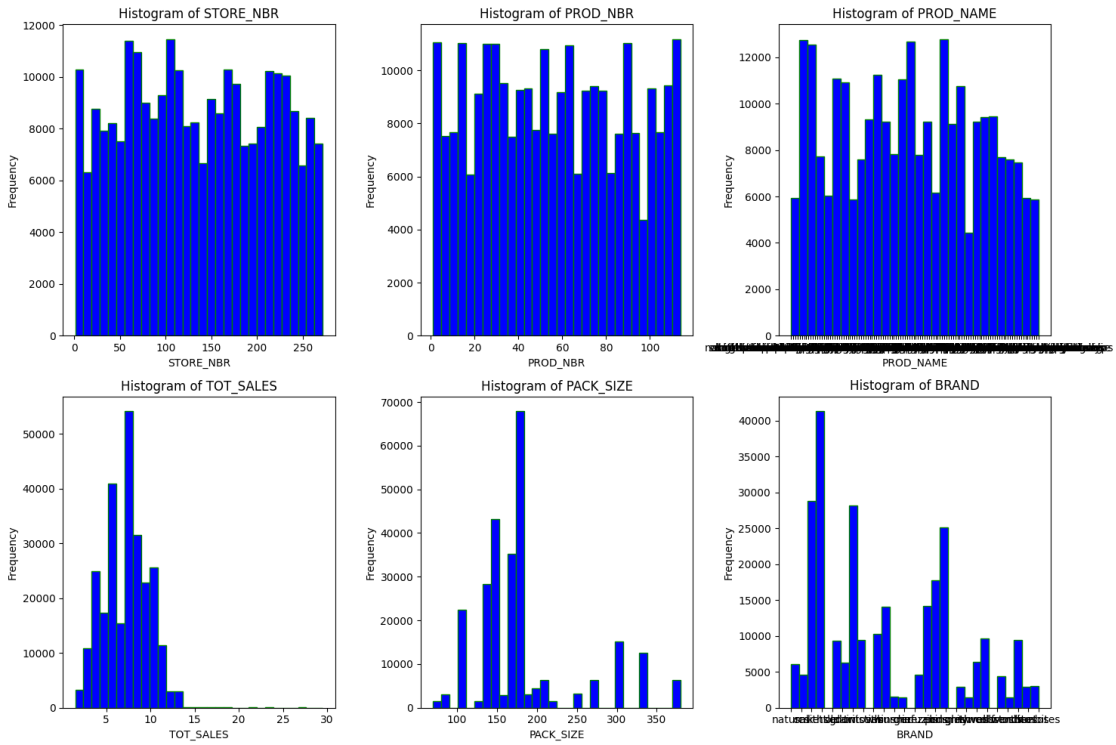
[287]:
```
transaction_data_df['BRAND'].unique()
```

[287]:
```
array(['natural', 'ccs', 'smiths', 'kettle', 'old', 'grain', 'doritos',
       'twisties', 'ww', 'thins', 'burger', 'ncc', 'cheezels',
       'infuzions', 'rrd', 'pringles', 'smith', 'grnwves', 'tyrrells',
       'cobs', 'woolworths', 'french', 'tostitos', 'cheetos', 'sunbites'],
      dtype=object)
```

3 Exploratory Data Analysis and more Visualisations

[288]:
```
# Histograms for each feature
features = ['STORE_NBR', 'PROD_NBR', 'PROD_NAME', 'TOT_SALES', 'PACK_SIZE',␣
 ↪'BRAND']
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))

for i, feature in enumerate(features):
    ax = axes[i//3, i%3]
    ax.hist(transaction_data_df[feature], bins=30, color='blue',␣
 ↪edgecolor='green')
    ax.set_title(f'Histogram of {feature}')
    ax.set_xlabel(feature)
    ax.set_ylabel('Frequency')
```

```
plt.tight_layout()
plt.show()

# They seem about right, resembling a normal distribution
```



[291]:
```
# No outliers or anomalies
transaction_data_df.describe()
```

[291]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_QTY	TOT_SALES
count	264834	264834.000000	2.648340e+05	2.648340e+05	264834.000000	264834.000000	264834.000000
mean	2018-12-30 00:52:10.292938240	135.079423	1.355488e+05	1.351576e+05	56.583554	1.905813	7.299346
min	2018-07-01 00:00:00	1.000000	1.000000e+03	1.000000e+00	1.000000	1.000000	1.500000
25%	2018-09-30 00:00:00	70.000000	7.002100e+04	6.760050e+04	28.000000	2.000000	5.400000
50%	2018-12-30 00:00:00	130.000000	1.303570e+05				
75%	2019-03-31 00:00:00	203.000000	2.030940e+05				
max	2019-06-30 00:00:00	272.000000	2.373711e+06				
std	NaN	76.784063	8.057990e+04				

```
50%    1.351365e+05      56.000000       2.000000       7.400000
75%    2.026998e+05      85.000000       2.000000       9.200000
max    2.415841e+06     114.000000       5.000000      29.500000
std    7.813292e+04      32.826444       0.343436       2.527241

           PACK_SIZE
count  264834.000000
mean      182.425512
min        70.000000
25%       150.000000
50%       170.000000
75%       175.000000
max       380.000000
std        64.325148
```

[292]:
```
# Proper types
transaction_data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 264834 entries, 0 to 264835
Data columns (total 10 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   DATE            264834 non-null  datetime64[ns]
 1   STORE_NBR       264834 non-null  int64
 2   LYLTY_CARD_NBR  264834 non-null  int64
 3   TXN_ID          264834 non-null  int64
 4   PROD_NBR        264834 non-null  int64
 5   PROD_NAME       264834 non-null  object
 6   PROD_QTY        264834 non-null  int64
 7   TOT_SALES       264834 non-null  float64
 8   PACK_SIZE       264834 non-null  float64
 9   BRAND           264834 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(5), object(2)
memory usage: 22.2+ MB
```

[295]:
```
# Merging transaction data and purchase behaviour data on loyalty card number
merged_df = pd.merge(transaction_data_df, purchase_behaviour_df,␣
 ↪on='LYLTY_CARD_NBR', how='left')
merged_df.head()
```

[295]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	\
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	

```
                              PROD_NAME  PROD_QTY  TOT_SALES  PACK_SIZE  \
0   natural chip        compny seasalt         2        6.0      175.0
1                ccs nacho cheese             3        6.3      175.0
2   smiths crinkle cut  chips chicken         2        2.9      170.0
3     smiths chip thinly  screamonion         5       15.0      175.0
4  kettle tortilla chpshnyjlpno chili         3       13.8      150.0

     BRAND              LIFESTAGE PREMIUM_CUSTOMER
0  natural   YOUNG SINGLES/COUPLES          Premium
1      ccs  MIDAGE SINGLES/COUPLES           Budget
2   smiths  MIDAGE SINGLES/COUPLES           Budget
3   smiths  MIDAGE SINGLES/COUPLES           Budget
4   kettle  MIDAGE SINGLES/COUPLES           Budget
```

[296]:
```
# No null values
merged_df.isnull().sum()
```

[296]:
```
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
```

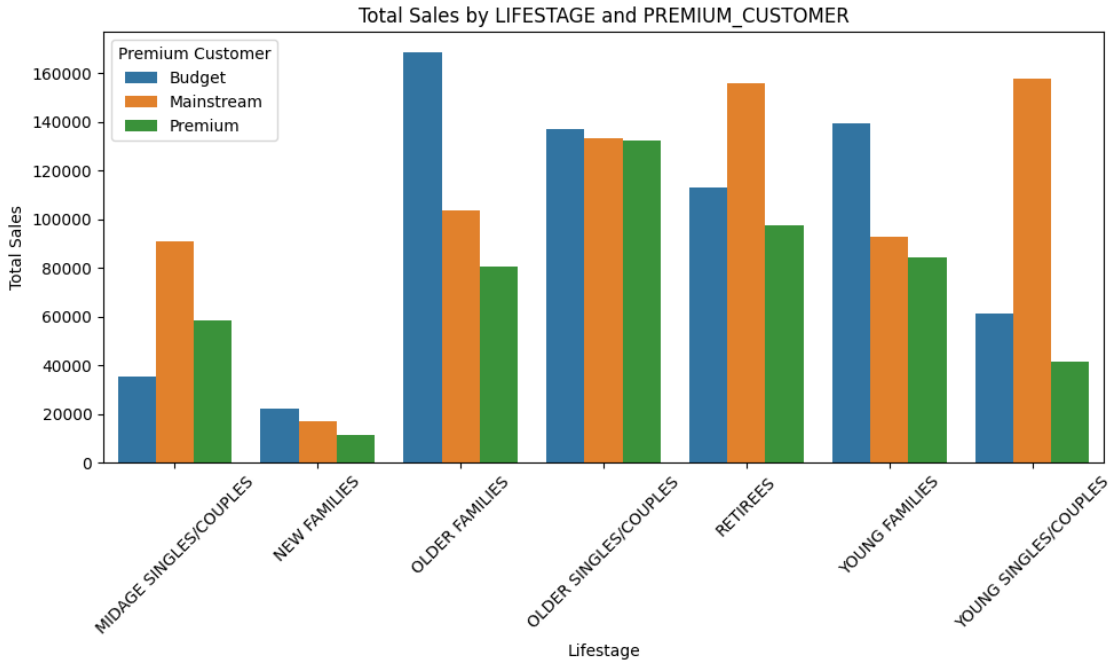
4 Data Analysis

[297]:
```
# Group by LIFESTAGE and PREMIUM_CUSTOMER, then sum the sales
total_sales = merged_df.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['TOT_SALES'].
 ↪sum().reset_index()
total_sales
```

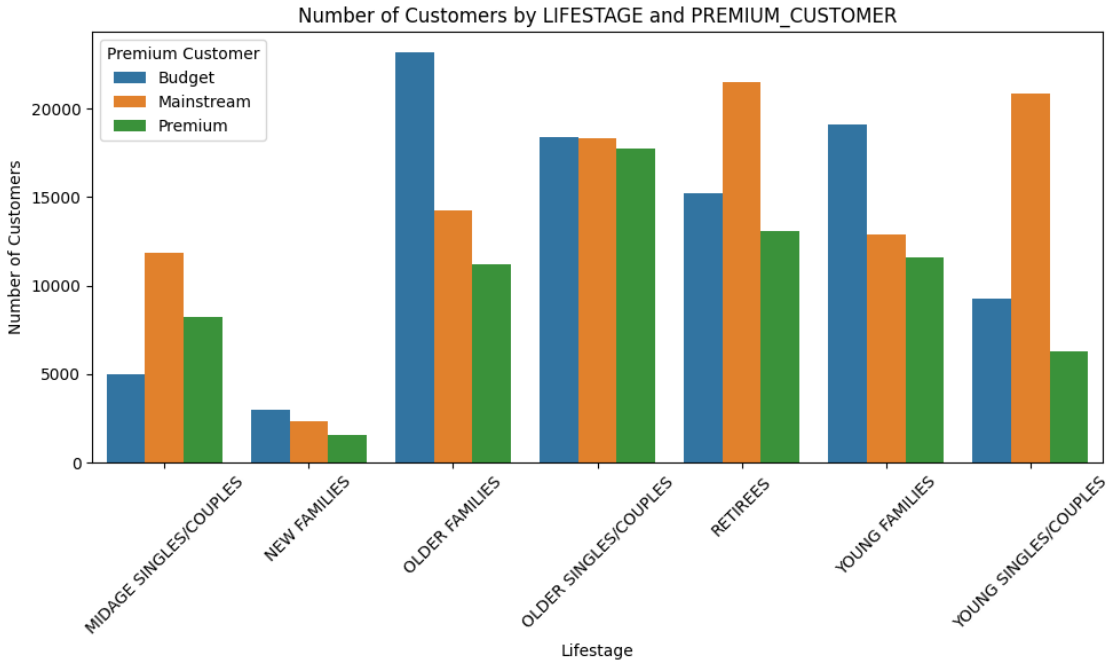
[297]:
```
                LIFESTAGE PREMIUM_CUSTOMER  TOT_SALES
0   MIDAGE SINGLES/COUPLES           Budget   35514.80
1   MIDAGE SINGLES/COUPLES       Mainstream   90803.85
2   MIDAGE SINGLES/COUPLES          Premium   58432.65
3            NEW FAMILIES           Budget   21928.45
4            NEW FAMILIES       Mainstream   17013.90
5            NEW FAMILIES          Premium   11491.10
6          OLDER FAMILIES           Budget  168363.25
```

```
7           OLDER FAMILIES       Mainstream  103445.55
8           OLDER FAMILIES          Premium   80658.40
9    OLDER SINGLES/COUPLES           Budget  136769.80
10   OLDER SINGLES/COUPLES       Mainstream  133393.80
11   OLDER SINGLES/COUPLES          Premium  132263.15
12                RETIREES           Budget  113147.80
13                RETIREES       Mainstream  155677.05
14                RETIREES          Premium   97646.05
15         YOUNG FAMILIES           Budget  139345.85
16         YOUNG FAMILIES       Mainstream   92788.75
17         YOUNG FAMILIES          Premium   84025.50
18   YOUNG SINGLES/COUPLES           Budget   61141.60
19   YOUNG SINGLES/COUPLES       Mainstream  157621.60
20   YOUNG SINGLES/COUPLES          Premium   41642.10
```

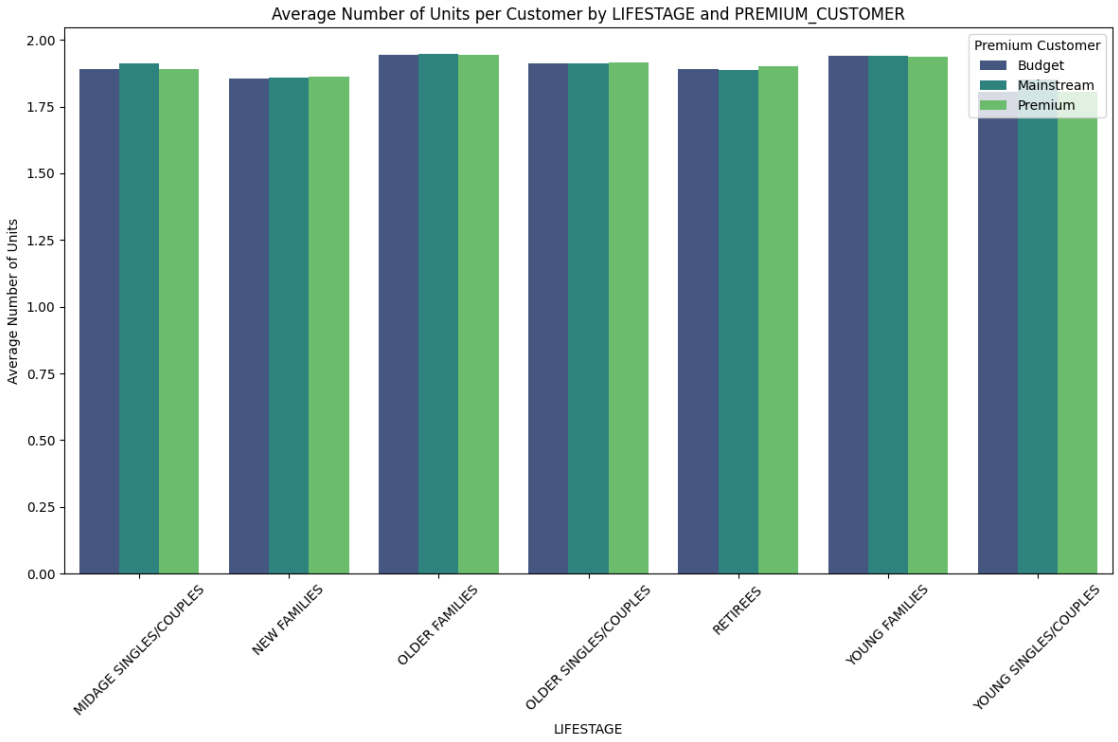
[298]:
```
# Create a bar plot
plt.figure(figsize=(10, 6))
sns.barplot(data=total_sales, x='LIFESTAGE', y='TOT_SALES',␣
 ↪hue='PREMIUM_CUSTOMER')

# Add titles and labels
plt.title('Total Sales by LIFESTAGE and PREMIUM_CUSTOMER')
plt.xlabel('Lifestage')
plt.ylabel('Total Sales')
plt.xticks(rotation=45)
plt.legend(title='Premium Customer')
plt.tight_layout()
plt.show()

# Budget Older Families have the biggest pruchase of chips, followed by␣
 ↪mainstream young singles/couples.
# Budget Older Families are more likely to buy chips as they are cheaper than␣
 ↪other food choices.
```



[300]:
```
# Group by LIFESTAGE and PREMIUM_CUSTOMER, then count the number of customers
customer_count = merged_df.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['TXN_ID'].
 ↪count().reset_index()
customer_count.rename(columns={'TXN_ID': 'CUSTOMER_COUNT'}, inplace=True)

# Create a bar plot
plt.figure(figsize=(10, 6))
sns.barplot(data=customer_count, x='LIFESTAGE', y='CUSTOMER_COUNT',␣
 ↪hue='PREMIUM_CUSTOMER')

# Add titles and labels
plt.title('Number of Customers by LIFESTAGE and PREMIUM_CUSTOMER')
plt.xlabel('Lifestage')
plt.ylabel('Number of Customers')
plt.xticks(rotation=45)
plt.legend(title='Premium Customer')
plt.tight_layout()
plt.show()

# We have more customers purchasing chips from Budget Older Families, followed␣
 ↪by Mainstream Young Singles/Couples.
# This supports the plot above.
```

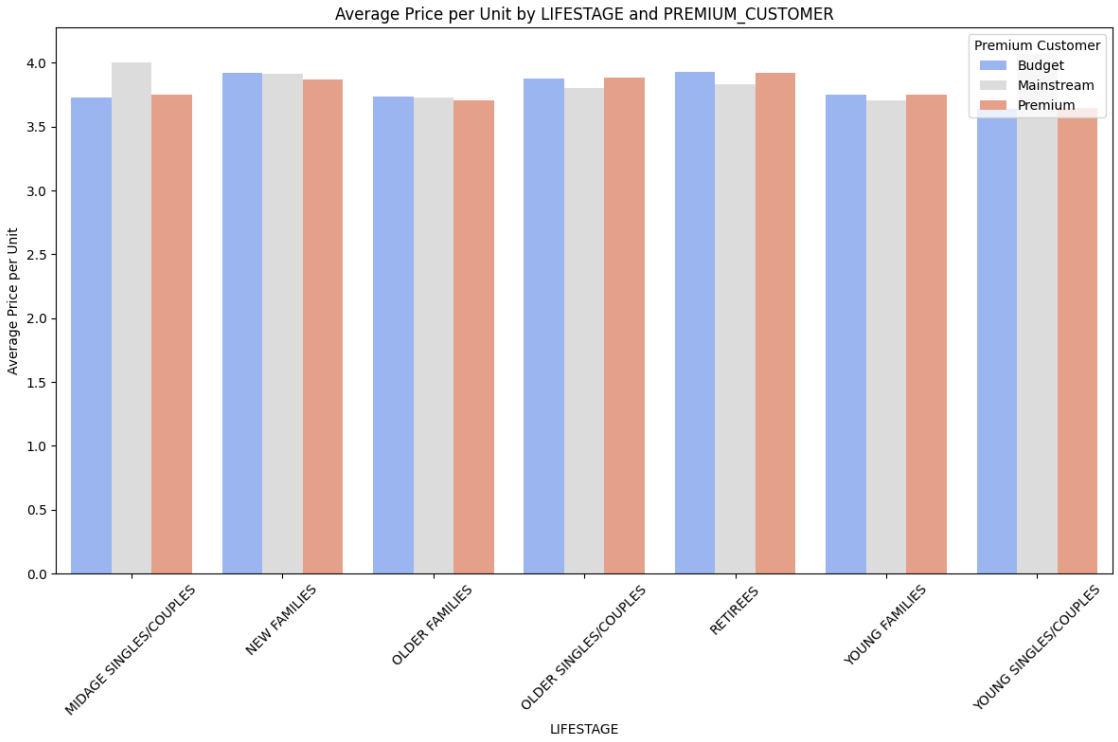


[301]:
```
# Calculate the average number of units per customer by LIFESTAGE and␣
 ↪PREMIUM_CUSTOMER
average_units = merged_df.groupby(['LIFESTAGE',␣
 ↪'PREMIUM_CUSTOMER'])['PROD_QTY'].mean().reset_index()

# Create a plot
plt.figure(figsize=(12, 8))
sns.barplot(data=average_units, x='LIFESTAGE', y='PROD_QTY',␣
 ↪hue='PREMIUM_CUSTOMER', palette='viridis')

# Customize the plot
plt.title('Average Number of Units per Customer by LIFESTAGE and␣
 ↪PREMIUM_CUSTOMER')
plt.xlabel('LIFESTAGE')
plt.ylabel('Average Number of Units')
plt.legend(title='Premium Customer')
plt.xticks(rotation=45)

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

# Older Families and Young Families from all customer types buy more chips than␣
 ↪the rest.
```



[303]:
```
# Calculate the average price per unit by LIFESTAGE and PREMIUM_CUSTOMER
# Assuming 'PRICE' is the column for price and 'UNITS' is the column for units
merged_df['PRICE_PER_UNIT'] = merged_df['TOT_SALES'] / merged_df['PROD_QTY']
average_price_per_unit = merged_df.groupby(['LIFESTAGE',␣
 ↪'PREMIUM_CUSTOMER'])['PRICE_PER_UNIT'].mean().reset_index()

# Create a plot
plt.figure(figsize=(12, 8))
sns.barplot(data=average_price_per_unit, x='LIFESTAGE', y='PRICE_PER_UNIT',␣
 ↪hue='PREMIUM_CUSTOMER', palette='coolwarm')

# Customize the plot
plt.title('Average Price per Unit by LIFESTAGE and PREMIUM_CUSTOMER')
plt.xlabel('LIFESTAGE')
plt.ylabel('Average Price per Unit')
plt.legend(title='Premium Customer')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Mainstream Midage and Young Singles/Couples are generally more willing to pay␣
 ↪a higher price per packet of
```

```
# chips compared to their Budget and Premium counterparts.This tendency might␣
 ↪be because Premium shoppers
# often prefer healthier snacks and purchase chips primarily for entertainment␣
 ↪rather than personal consumption.
# Additionally, this is reflected in the lower number of Premium Midage and␣
 ↪Young Singles/Couples buying chips
# compared to their Mainstream counterparts.
```



[304]:
```
# let's see if there is any statistical difference in average per unit for␣
 ↪three segments
# Performing t-test for differences

from scipy import stats

# Filter data for the first group: Mainstream Midage and Young Single and␣
 ↪Couples
mainstream = merged_df[merged_df['PREMIUM_CUSTOMER'] == 'Mainstream']
mainstream = mainstream[mainstream['LIFESTAGE'].isin(['MIDAGE SINGLES/COUPLES',␣
 ↪'YOUNG SINGLES/COUPLES'])]['PRICE_PER_UNIT']

# Filter data for the second group: Budget Midage and Young Single and Couples
budget = merged_df[merged_df['PREMIUM_CUSTOMER'] == 'Budget']
```

```
budget = budget[budget['LIFESTAGE'].isin(['MIDAGE SINGLES/COUPLES', 'YOUNG␣
 ↪SINGLES/COUPLES'])]['PRICE_PER_UNIT']

# Filter data for the third group: Premium Midage and Young Single and Couples
premium = merged_df[merged_df['PREMIUM_CUSTOMER'] == 'Premium']
premium = premium[premium['LIFESTAGE'].isin(['MIDAGE SINGLES/COUPLES', 'YOUNG␣
 ↪SINGLES/COUPLES'])]['PRICE_PER_UNIT']

# Perform independent t-test between the two groups
t_stat1, p_val1 = stats.ttest_ind(mainstream, budget, equal_var=False)
t_stat2, p_val2 = stats.ttest_ind(mainstream, premium, equal_var=False)

# Print results
print(f"Mainstream Midage and Young Singles and Couples vs Budget Midage and␣
 ↪Young Singles and Couples:")
print(f"T-statistic: {t_stat1}")
print(f"P-value: {p_val1}")

# Print results
print(f"Mainstream Midage and Young Singles and Couples vs Premium Midage and␣
 ↪Young Singles and Couples:")
print(f"T-statistic: {t_stat2}")
print(f"P-value: {p_val2}")
```

```
Mainstream Midage and Young Singles and Couples vs Budget Midage and Young
Singles and Couples:
T-statistic: 34.02371739507088
P-value: 3.964080604057843e-248
Mainstream Midage and Young Singles and Couples vs Premium Midage and Young
Singles and Couples:
T-statistic: 30.73527233963562
P-value: 9.433984178910887e-204
```

The t-test results in a p-value of 3.964080604057843e-248 and 9.433984178910887e-204 , the average unit price for mainstream, young and mid-age singles and couples ARE significantly higher than that of budget or premium, young and midage singles and couples.

[305]:
```
# Now that we have ruled out that there is a statistical difference,
# Let's look at their frequently bought brand of chips

from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# Filter data for Mainstream Young Singles/Couples
mainstream_young = merged_df[(merged_df['PREMIUM_CUSTOMER'] == 'Mainstream') &␣
 ↪(merged_df['LIFESTAGE'] == 'YOUNG SINGLES/COUPLES')]

# Filter data for Mainstream Midage Singles/Couples
```

```
mainstream_midage = merged_df[(merged_df['PREMIUM_CUSTOMER'] == 'Mainstream') &␣
 ↪(merged_df['LIFESTAGE'] == 'MIDAGE SINGLES/COUPLES')]

# For Mainstream Young Singles/Couples segment
young_transactions = mainstream_young.groupby('LYLTY_CARD_NBR')['BRAND'].
 ↪apply(list).tolist()

# For Mainstream Midage Singles/Couples segment
midage_transactions = mainstream_midage.groupby('LYLTY_CARD_NBR')['BRAND'].
 ↪apply(list).tolist()

# Combine all transactions to fit TransactionEncoder once
all_transactions = young_transactions + midage_transactions

# Convert transaction data into the one-hot encoded format
te = TransactionEncoder()
te_ary = te.fit(all_transactions).transform(all_transactions)

# Create DataFrame for both segments using the same columns
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

# Split the DataFrame into segments
df_young = df_encoded.iloc[:len(young_transactions)]
df_midage = df_encoded.iloc[len(young_transactions):]

# Apply Apriori algorithm to find frequent itemsets
frequent_itemsets_young = apriori(df_young, min_support=0.2, use_colnames=True)
frequent_itemsets_midage = apriori(df_midage, min_support=0.2,␣
 ↪use_colnames=True)

# Analyze and print the results
print("Mainstream Young Singles/Couples Brand Preferences:")
print(frequent_itemsets_young)

print("\nMainstream Midage Singles/Couples Brand Preferences:")
print(frequent_itemsets_midage)
```

```
Mainstream Young Singles/Couples Brand Preferences:
    support    itemsets
0  0.267928   (doritos)
1  0.378956    (kettle)
2  0.250742  (pringles)

Mainstream Midage Singles/Couples Brand Preferences:
    support    itemsets
0  0.320359   (doritos)
1  0.475150    (kettle)
```

```
2  0.299102  (pringles)
3  0.280539    (smiths)
```

With a minimum support of 0.2, for Mainstream Young Singles/Couples, the top brand preferences are Doritos, Kettle, and Pringles, with Kettle being the most favored. For Mainstream Midage Singles/Couples, the preferred brands include Doritos, Kettle, Pringles, and Smiths, with Kettle being the most popular and Doritos coming in second.

I recommend that Julia increase the stock of Kettle for the Mainstream Young Singles/Couples segment, and for the Mainstream Midage Singles/Couples segment, she should stock up on both Kettle and Doritos. Additionally, if there is funding available, Julia should consider running a targeted campaign focused on Kettle, as it is the most frequently purchased item by both segments. This strategy is likely to boost overall sales.

[306]:
```
# let's also investigate their frequently bought chip pack sizes

from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# Filter data for Mainstream Young Singles/Couples
mainstream_young = merged_df[(merged_df['PREMIUM_CUSTOMER'] == 'Mainstream') &␣
 ↪(merged_df['LIFESTAGE'] == 'YOUNG SINGLES/COUPLES')]

# Filter data for Mainstream Midage Singles/Couples
mainstream_midage = merged_df[(merged_df['PREMIUM_CUSTOMER'] == 'Mainstream') &␣
 ↪(merged_df['LIFESTAGE'] == 'MIDAGE SINGLES/COUPLES')]

# For Mainstream Young Singles/Couples segment
young_transactions = mainstream_young.groupby('LYLTY_CARD_NBR')['PACK_SIZE'].
 ↪apply(list).tolist()

# For Mainstream Midage Singles/Couples segment
midage_transactions = mainstream_midage.groupby('LYLTY_CARD_NBR')['PACK_SIZE'].
 ↪apply(list).tolist()

# Combine all transactions to fit TransactionEncoder once
all_transactions = young_transactions + midage_transactions

# Convert transaction data into the one-hot encoded format
te = TransactionEncoder()
te_ary = te.fit(all_transactions).transform(all_transactions)

# Create DataFrame for both segments using the same columns
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

# Split the DataFrame into segments
df_young = df_encoded.iloc[:len(young_transactions)]
df_midage = df_encoded.iloc[len(young_transactions):]
```

```
# Apply Apriori algorithm to find frequent itemsets
frequent_itemsets_young = apriori(df_young, min_support=0.2, use_colnames=True)
frequent_itemsets_midage = apriori(df_midage, min_support=0.2,␣
 ↪use_colnames=True)

# Analyze and print the results
print("Mainstream Young Singles/Couples Brand Preferences:")
print(frequent_itemsets_young)

print("\nMainstream Midage Singles/Couples Brand Preferences:")
print(frequent_itemsets_midage)
```

```
Mainstream Young Singles/Couples Brand Preferences:
    support itemsets
0  0.219708  (110.0)
1  0.250742  (134.0)
2  0.318867  (150.0)
3  0.448566  (175.0)

Mainstream Midage Singles/Couples Brand Preferences:
    support        itemsets
0  0.281437         (110.0)
1  0.299102         (134.0)
2  0.405389         (150.0)
3  0.226946         (170.0)
4  0.564072         (175.0)
5  0.243413  (150.0, 175.0)
```

With a minimum support of 0.2, for Mainstream Young Singles/Couples, the most popular pack sizes are 110g, 134g, 150g, and 175g, with 175g being the top choice. For Mainstream Midage Singles/Couples, the favored pack sizes include 110g, 134g, 150g, 170g, 175g, and both 150g and 175g combined, with 175g being the most popular.

I suggest that Julia increase the stock of 175g chips, as this size is frequently purchased by both segments. If there is available funding, a targeted campaign for 175g chips could further boost sales.

Based on these insights, Julia should also focus on stocking up 175g Kettle chips, as this size is frequently bought by the highest spenders. This strategy should help to further enhance overall sales.

[ ]: