Quantium Virtual Internship
Retail Strategy and Analytics
Task 1

Import Packages

In [1]:

```
import pandas as pd
import re
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from datetime import date
from scipy.stats import ttest_ind
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

Create Dataframes from Data Files

In [2]:

```
transaction_data = pd.read_csv('QVI_transaction_data.csv')
customer_data = pd.read_csv('QVI_purchase_behaviour.csv')
```

Exploratory Data Analysis - Transaction Data

Examine transaction data

There are no null values in the dataset. All columns are in their expected formats except the date column which is in integer format.

In [3]:

```
# Summary of the columns in the transaction dataset
transaction_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 264836 entries, 0 to 264835
Data columns (total 8 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   DATE            264836 non-null  int64  
 1   STORE_NBR       264836 non-null  int64  
 2   LYLTY_CARD_NBR  264836 non-null  int64  
 3   TXN_ID          264836 non-null  int64  
 4   PROD_NBR        264836 non-null  int64  
 5   PROD_NAME       264836 non-null  object 
 6   PROD_QTY        264836 non-null  int64  
 7   TOT_SALES       264836 non-null  float64
dtypes: float64(1), int64(6), object(1)
memory usage: 16.2+ MB
```

In [4]:

```
# The first 5 rows of the transaction dataset
transaction_data.head()
```

Out[4]:

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

Convert DATE column to a date format

In [5]:

```
transaction_data.DATE = transaction_data.DATE.apply(lambda x: date.fromordinal(date(1900, 1
transaction_data.head()
```

Out[5]:

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

Examine PROD_NAME

There are 114 unique entries for PROD_NAME with 'Kettle Mozzarella Basil & Pesto 175g' being the most frequent.

In [6]:

```
transaction_data.PROD_NAME.describe()
```

Out[6]:

```
count                                   264836
unique                                     114
top       Kettle Mozzarella   Basil & Pesto 175g
freq                                      3304
Name: PROD_NAME, dtype: object
```

In [7]:

```
transaction_data.PROD_NAME.value_counts()
```

Out[7]:

```
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: PROD_NAME, Length: 114, dtype: int64
```

Examine the words in PROD_NAME

In [8]:

```
# List individual words from entries in the PROD_NAME column - the 30 most frequent are dis
productWords = pd.DataFrame(transaction_data.PROD_NAME)
productWords.columns = ['words']
words_list = [words.lower().split() for words in productWords['words']]
words_list = [word for sublist in words_list for word in sublist]
words_list[:15]
```

Out[8]:

```
['natural',
 'chip',
 'compny',
 'seasalt175g',
 'ccs',
 'nacho',
 'cheese',
 '175g',
 'smiths',
 'crinkle',
 'cut',
 'chips',
 'chicken',
 '170g',
 'smiths']
```

In [9]:

```
# Remove digits and special characters from words and then remove all words with less than
cleaned_words_list = [re.sub(r'[^a-zA-z]',' ',words).lower().split() for words in words_lis
cleaned_words_list = [word for sublist in cleaned_words_list for word in sublist if len(wor
```

In [10]:

```
# Sort the distinct words by frequency of occurrence - the 30 most frequent are displayed b
word_count = [[x,cleaned_words_list.count(x)] for x in set(cleaned_words_list)]
sorted(word_count, key = lambda x: x[1], reverse=True)[:15]
```

Out[10]:

```
[['chips', 49770],
 ['kettle', 41288],
 ['smiths', 28860],
 ['salt', 27976],
 ['cheese', 27890],
 ['pringles', 25102],
 ['doritos', 24962],
 ['crinkle', 23960],
 ['corn', 22063],
 ['original', 21560],
 ['cut', 20754],
 ['chip', 18645],
 ['chicken', 18577],
 ['salsa', 18094],
 ['cream', 16926]]
```

Cleaning Data

In [11]:

```
# List all products containing the word salsa
transaction_data.PROD_NAME[transaction_data.PROD_NAME.str.contains('Salsa|salsa|SALSA')].un
```

Out[11]:

```
array(['Old El Paso Salsa   Dip Tomato Mild 300g',
       'Red Rock Deli SR    Salsa & Mzzrlla 150g',
       'Smiths Crinkle Cut  Tomato Salsa 150g',
       'Doritos Salsa       Medium 300g',
       'Old El Paso Salsa   Dip Chnky Tom Ht300g',
       'Woolworths Mild     Salsa 300g',
       'Old El Paso Salsa   Dip Tomato Med 300g',
       'Woolworths Medium   Salsa 300g', 'Doritos Salsa Mild  300g'],
      dtype=object)
```

In [12]:

```
# Remove products containing the word salsa if it is not a chip product
transaction_data = transaction_data[~transaction_data.PROD_NAME.str.contains('Old El Paso S
transaction_data = transaction_data[~transaction_data.PROD_NAME.str.contains('Doritos Salsa
transaction_data = transaction_data[~transaction_data.PROD_NAME.str.contains('Woolworths Mi
transaction_data = transaction_data[~transaction_data.PROD_NAME.str.contains('Woolworths Me
```

The mean, median and third quartile values for the columns PROD_QTY and TOT_SALES are much closer to the minimum values compared to the maximum values. This suggests that there could be at least one outlier in the dataset. The columns STORE_NBR, LYLTY_CARD_NBR, TXN_ID, and PROD_NBR are not relevant regarding outliers as they signify numbers that have no quantitative value. They could just as easily be represented by string values rather than integer values.

In [13]:

```
# Check for outliers
transaction_data.describe()
```

Out[13]:

	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_QTY	TOT_
count	249670.000000	2.496700e+05	2.496700e+05	249670.000000	249670.000000	249670.
mean	135.044278	1.355203e+05	1.351234e+05	56.294288	1.907762	7.
std	76.773591	8.065746e+04	7.813155e+04	33.528758	0.657126	3.
min	1.000000	1.000000e+03	1.000000e+00	1.000000	1.000000	1.
25%	70.000000	7.001625e+04	6.757425e+04	27.000000	2.000000	5.
50%	130.000000	1.303600e+05	1.351475e+05	53.000000	2.000000	7.
75%	203.000000	2.030798e+05	2.026338e+05	86.000000	2.000000	8.
max	272.000000	2.373711e+06	2.415841e+06	114.000000	200.000000	650.

There are two transactions with extreme values for the PROD_QTY column and hence the TOT_SALES column. These two transactions are for the same customer. The values in the PROD_QTY column are 200 which is far greater than the remaining values which range from 1 to 5. As a result the values in the TOT_SALES column are also much higher than the remaining range of values in that column.

In [14]:

```
# The number of extreme vales for PROD_QTY
transaction_data[transaction_data.PROD_QTY > 5]
```

Out[14]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY
69762	2018-08-19	226	226000	226201	4	Dorito Corn Chp Supreme 380g	200
69763	2019-05-20	226	226000	226210	4	Dorito Corn Chp Supreme 380g	200

In [15]:

```
# The number of extreme vales for TOT_SALES
transaction_data[transaction_data.TOT_SALES > 30]
```

Out[15]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY
69762	2018-08-19	226	226000	226201	4	Dorito Corn Chp Supreme 380g	200
69763	2019-05-20	226	226000	226210	4	Dorito Corn Chp Supreme 380g	200

There are no further transactions for this customer. This does not appear to be an ordinary retail customer, so it has been removed from the dataset.

In [16]:

```
# All purchasing transactions for the customer with extreme values for PROD_QTY and TOT_SAL
transaction_data[transaction_data.LYLTY_CARD_NBR == 226000]
```

Out[16]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY
69762	2018-08-19	226	226000	226201	4	Dorito Corn Chp Supreme 380g	200
69763	2019-05-20	226	226000	226210	4	Dorito Corn Chp Supreme 380g	200

In [17]:

```
# Remove all transactions with loyalty card number 226000
transaction_data = transaction_data[transaction_data.LYLTY_CARD_NBR != 226000]
```

All extreme values have been removed from the dataset which now has 249668 transactions. The range of values in the PROD_QTY and TOT_SALES columns appear to be more evenly distributed now.

In [18]:

```
# Ensure that values for the PROD_QTY and TOT_SALES columns are now more evenly distributed
transaction_data.describe()
```

Out[18]:

	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_QTY	TOT_
count	249668.000000	2.496680e+05	2.496680e+05	249668.000000	249668.000000	249668.
mean	135.043550	1.355196e+05	1.351227e+05	56.294707	1.906175	7.
std	76.773467	8.065737e+04	7.813144e+04	33.528566	0.342744	2.
min	1.000000	1.000000e+03	1.000000e+00	1.000000	1.000000	1.
25%	70.000000	7.001600e+04	6.757375e+04	27.000000	2.000000	5.
50%	130.000000	1.303595e+05	1.351465e+05	53.000000	2.000000	7.
75%	203.000000	2.030790e+05	2.026322e+05	86.000000	2.000000	8.
max	272.000000	2.373711e+06	2.415841e+06	114.000000	5.000000	29.

There are only 364 unique dates for the year when there should be 365. Therefore, there is one date missing in the dataset.

In [19]:

```
# The number of transactions by date
no_of_transactions = pd.DataFrame(transaction_data.DATE.value_counts())
no_of_transactions.columns = ['Transactions']
len(no_of_transactions)
```

Out[19]:

```
364
```

The missing date is christmas day when presumably the shops would have been closed.

In [20]:

```
# Display missing date from dataset
dates = pd.date_range('7-1-2018', '6-30-2019').date
missing_dates = [str(date) for date in dates if date not in no_of_transactions.index]
print(missing_dates)
```

```
['2018-12-25']
```

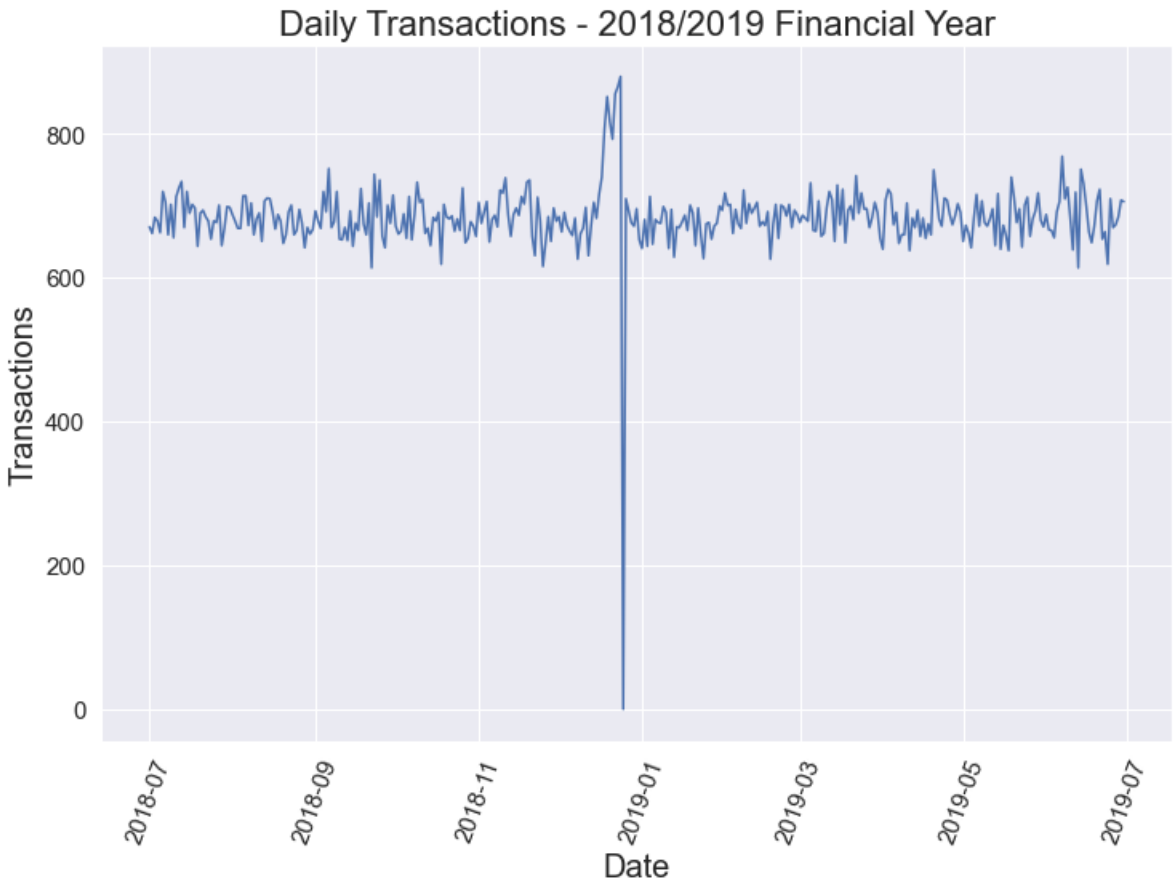
The frequency of transactions across the year are consistent other than the increase that occurs just before Christmas day and there being no transactions on Christmas day.

In [21]:

```
# Plot transactions of data over the financial year
no_of_transactions.loc[date(2018, 12, 25)] = [0]
no_of_transactions.sort_index(inplace=True)
no_of_transactions = no_of_transactions.rename_axis('Date').reset_index()
sns.set(rc={'figure.figsize':(12,8)})
sns.lineplot(data=no_of_transactions, x='Date', y='Transactions')
plt.xlabel('Date', fontsize = 20)
plt.xticks(fontsize=15, rotation=70)
plt.ylabel('Transactions', fontsize = 20)
plt.yticks(fontsize=15)
plt.title('Daily Transactions - 2018/2019 Financial Year', fontsize = 22)
```

Out[21]:

```
Text(0.5, 1.0, 'Daily Transactions - 2018/2019 Financial Year')
```

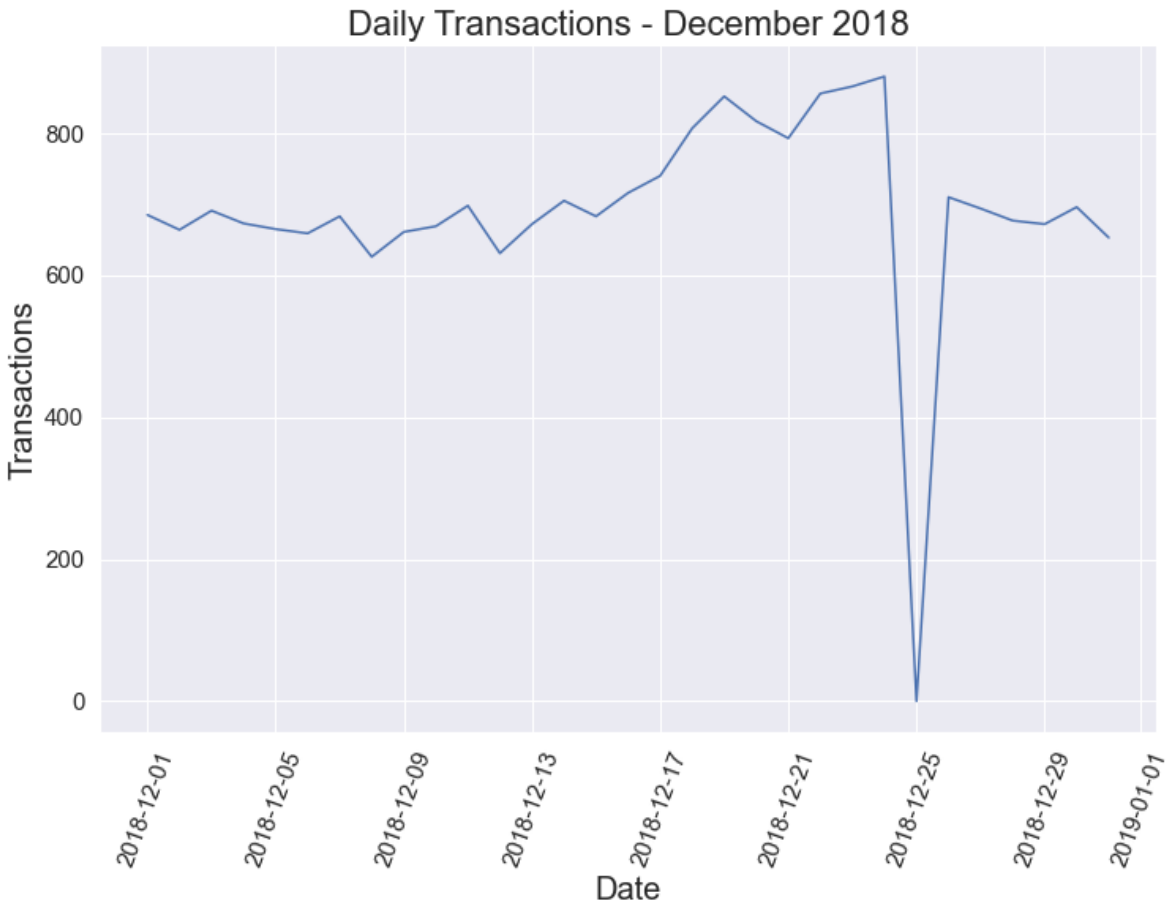


In [22]:

```
# Plot transactions for December 2018
dec_transactions = no_of_transactions[(no_of_transactions['Date'] >= date(2018, 12, 1)) &
                                      (no_of_transactions['Date'] <= date(2018, 12, 31))]
sns.set(rc={'figure.figsize':(12,8)})
sns.lineplot(data=dec_transactions, x='Date', y='Transactions')
plt.xlabel('Date', fontsize = 20)
plt.xticks(fontsize=15, rotation=70)
plt.ylabel('Transactions', fontsize = 20)
plt.yticks(fontsize=15)
plt.title('Daily Transactions - December 2018', fontsize = 22)
```

Out[22]:

```
Text(0.5, 1.0, 'Daily Transactions - December 2018')
```



Create Additional Features

In [23]:

```
# Create a column that displays the unit price for the chip packets bought in each transact
transaction_data['UNIT_PRICE'] = transaction_data.TOT_SALES / transaction_data.PROD_QTY
```

The pack sizes range between 70 and 380 grams with 175 grams being the most frequent.

In [24]:

```
# Create a pack size column from information in the PROD_NAME column and determine its rang
transaction_data['PACK_SIZE'] = transaction_data.PROD_NAME.str.extract('(\d+)').astype('int
pack_size = pd.DataFrame(transaction_data.PACK_SIZE.value_counts().rename_axis('Grams').res
pack_size['Grams'] = pack_size['Grams'].astype(int)
pack_size = pack_size.set_index('Grams').sort_index()
pack_size
```

Out[24]:

Grams	Frequency
70	1507
90	3008
110	22387
125	1454
134	25102
135	3257
150	43131
160	2970
165	15297
170	19983
175	66390
180	1468
190	2995
200	4473
210	6272
220	1564
250	3169
270	6285
330	12540
380	6416

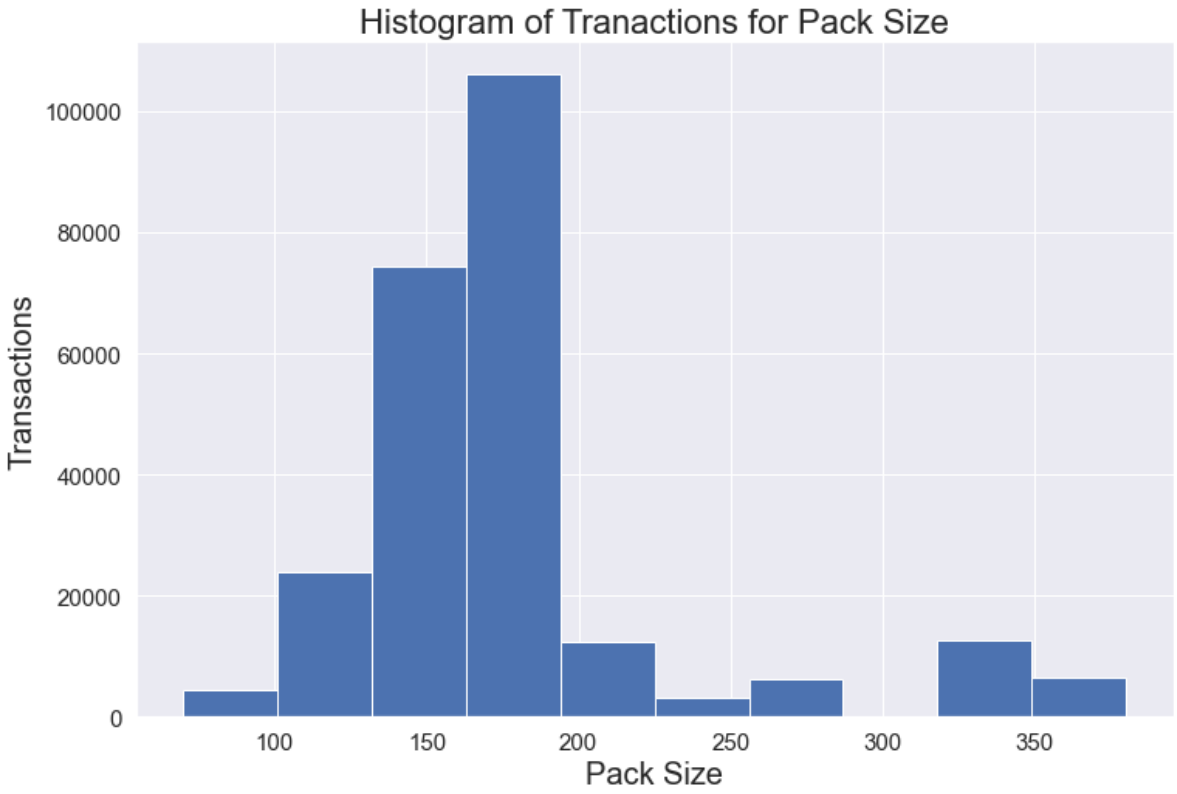
The histogram is skewed to the right with most values concentrated between the 100 and 200 gram pack sizes.

In [25]:

```
# Plot a histogram showing the number of transactions by pack size
plt.hist(transaction_data.PACK_SIZE)
plt.xlabel('Pack Size', fontsize = 20)
plt.xticks(fontsize=15)
plt.ylabel('Transactions', fontsize = 20)
plt.yticks(fontsize=15)
plt.title('Histogram of Tranactions for Pack Size', fontsize = 22)
```

Out[25]:

```
Text(0.5, 1.0, 'Histogram of Tranactions for Pack Size')
```



Some company names appear more than once with alternative names or abbreviations.

In [26]:

```
# Create a column which contains the brand of the product, by extracting the first word fro
transaction_data['BRAND'] = transaction_data.PROD_NAME.str.split().str.get(0)
transaction_data['BRAND'].value_counts()
```

Out[26]:

```
Kettle        41288
Smiths        28860
Pringles      25102
Doritos       22041
Thins         14075
RRD           11894
Infuzions     11057
WW            10320
Cobs           9693
Tostitos       9471
Twisties       9454
Tyrrells       6442
Grain          6272
Natural        6050
Red            5885
Cheezels       4603
CCs            4551
Dorito         3183
Infzns         3144
Smith          2963
Cheetos        2927
Snbts          1576
Burger         1564
Woolworths     1516
GrnWves        1468
Sunbites       1432
NCC            1419
French         1418
Name: BRAND, dtype: int64
```

NB: It has been assumed that the brand name WW refers to Weight Watchers and is not an alternate name for Woolworths.

In [27]:

```
# Combine alternate company names to single brand description and display summary
transaction_data['BRAND'] = transaction_data['BRAND'].str.replace('Dorito', 'Doritos')
transaction_data['BRAND'] = transaction_data['BRAND'].str.replace('Doritoss', 'Doritos')
transaction_data['BRAND'] = transaction_data['BRAND'].str.replace('Infzns', 'Infuzions')
transaction_data['BRAND'] = transaction_data['BRAND'].str.replace('GrnWves', 'Grain')
transaction_data['BRAND'] = transaction_data['BRAND'].str.replace('Natural', 'NCC')
transaction_data['BRAND'] = transaction_data['BRAND'].str.replace('Red', 'RRD')
transaction_data['BRAND'] = transaction_data['BRAND'].str.replace('Smith', 'Smiths')
transaction_data['BRAND'] = transaction_data['BRAND'].str.replace('Smithss', 'Smiths')
transaction_data['BRAND'] = transaction_data['BRAND'].str.replace('Snbts', 'Sunbites')

pd.DataFrame(transaction_data.BRAND.value_counts().rename_axis('Brands').to_frame('Frequenc
```

Out[27]:

Brands	Frequency
Burger	1564
CCs	4551
Cheetos	2927
Cheezels	4603
Cobs	9693
Doritos	25224
French	1418
Grain	7740
Infuzions	14201
Kettle	41288
NCC	7469
Pringles	25102
RRD	17779
Smiths	31823
Sunbites	3008
Thins	14075
Tostitos	9471
Twisties	9454
Tyrrells	6442
WW	10320
Woolworths	1516

Exploratory Data Analysis - Customer Data

The customer dataset has three attributes which are all qualitative.

In [28]:

```
# View first 5 rows of the customer dataset
customer_data.head()
```

Out[28]:

	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

There is no missing data in the customer dataset.

In [29]:

```
# Summary of the columns in the customer dataset
customer_data.info()
```

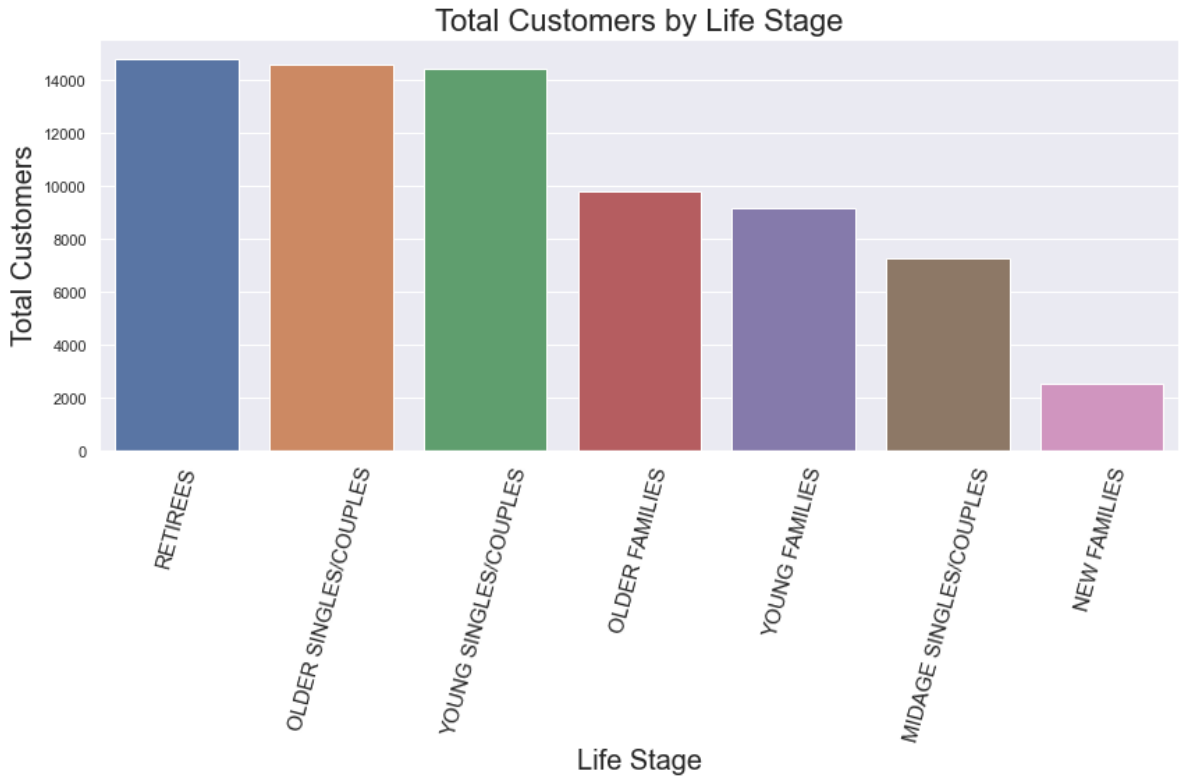
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72637 entries, 0 to 72636
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   LYLTY_CARD_NBR    72637 non-null  int64 
 1   LIFESTAGE         72637 non-null  object
 2   PREMIUM_CUSTOMER  72637 non-null  object
dtypes: int64(1), object(2)
memory usage: 1.7+ MB
```

Total Sales by Life Stage

Retirees, older singles/couples, and young singles/couples represent approximately 60 percent of all customers while all family types represent around 30 percent of all customers.

In [30]:

```
# Plot all customers by life stage
life_stage = pd.DataFrame(customer_data.LIFESTAGE.value_counts().rename_axis('Life Stage').
sns.set(rc={'figure.figsize':(12,8)})
sns.barplot(x='Life Stage', y='Total Customers', data=life_stage)
plt.xlabel('Life Stage', fontsize = 20)
plt.xticks(fontsize=15)
plt.ylabel('Total Customers', fontsize = 20)
plt.title('Total Customers by Life Stage', fontsize = 22)
plt.xticks(rotation=75)
plt.tight_layout()
```

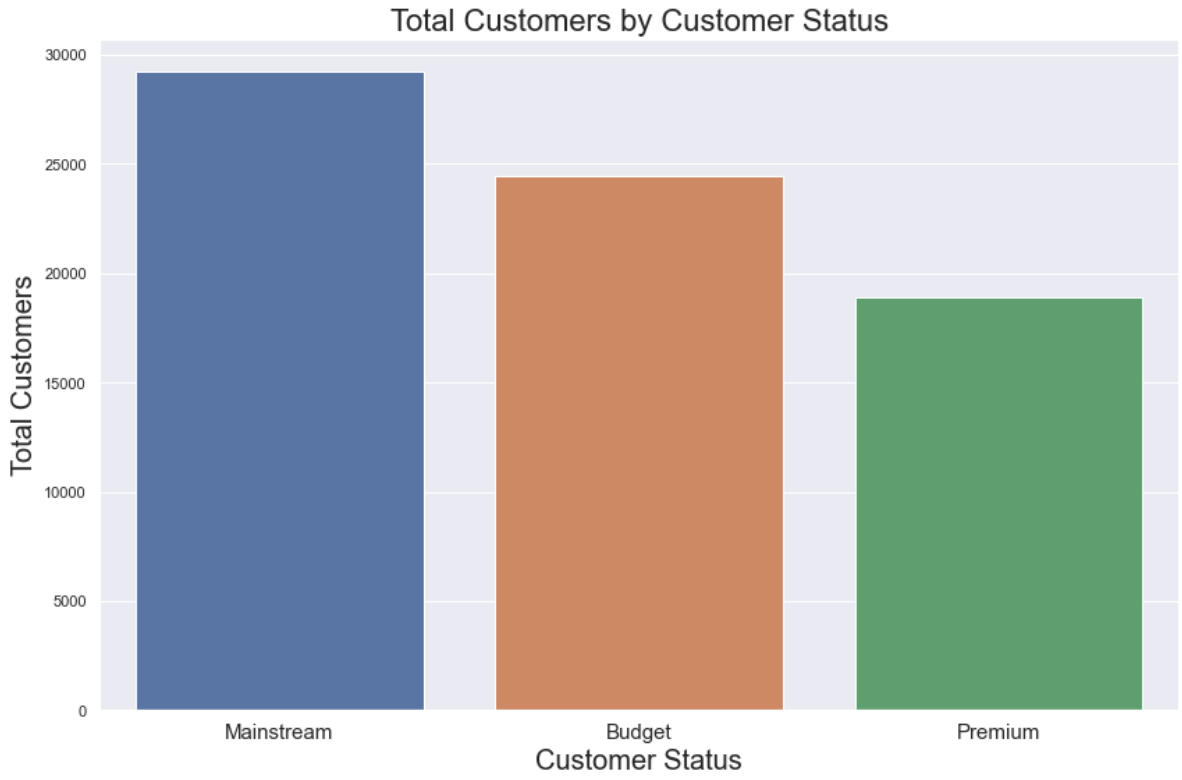


Total Customers by Customer Status

Mainstream and budget customers are more common than premium who represent only around one quarter of all customers.

In [31]:

```
# Plot all customers by values in the premium customer column
premium_cust = pd.DataFrame(customer_data.PREMIUM_CUSTOMER.value_counts().rename_axis('Cust
                            reset_index(name='Total Customers'))
sns.set(rc={'figure.figsize':(12,8)})
sns.barplot(x='Customer Status', y='Total Customers', data=premium_cust)
plt.xlabel('Customer Status', fontsize = 20)
plt.xticks(fontsize=15)
plt.ylabel('Total Customers', fontsize = 20)
plt.title('Total Customers by Customer Status', fontsize = 22)
plt.tight_layout()
```



Merge Transaction Data to Customer Data and Save

In [32]:

```
# Merge the transaction and customer datasets into a single dataset
data = pd.merge(transaction_data, customer_data, on=['LYLTY_CARD_NBR'])
```

The number of transactions in the merged dataset is the same as the transaction dataset so there are no duplicates. There are no null values in the merged dataset, so all transactions have a matching customer.

In [33]:

```
# View summary of the merged dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 249668 entries, 0 to 249667
Data columns (total 13 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   DATE              249668 non-null  object 
 1   STORE_NBR         249668 non-null  int64  
 2   LYLTY_CARD_NBR    249668 non-null  int64  
 3   TXN_ID            249668 non-null  int64  
 4   PROD_NBR          249668 non-null  int64  
 5   PROD_NAME         249668 non-null  object 
 6   PROD_QTY          249668 non-null  int64  
 7   TOT_SALES         249668 non-null  float64
 8   UNIT_PRICE        249668 non-null  float64
 9   PACK_SIZE         249668 non-null  int64  
 10  BRAND             249668 non-null  object 
 11  LIFESTAGE         249668 non-null  object 
 12  PREMIUM_CUSTOMER  249668 non-null  object 
dtypes: float64(2), int64(6), object(5)
memory usage: 26.7+ MB
```

In [34]:

```
# Calculate the number of customers where no transactions have been recorded
customer_data_length = len(customer_data)
customer_distinct = len(data.LYLTY_CARD_NBR.value_counts())
customer_no_transaction = customer_data_length - customer_distinct

print('There are {} unique customers in the customer dataset. However, there are only {} di
      'merged data  file. Therefore, there are {} customers in the customer dataset where n
      .format(customer_data_length, customer_distinct, customer_no_transaction))
```

```
There are 72637 unique customers in the customer dataset. However, there are
only 71517 distinct customers in the merged data  file. Therefore, there are
1120 customers in the customer dataset where no transaction has been recorde
d.
```

In [35]:

```
# Save the merged data file to a csv file for use in later tasks
data.to_csv('QVI_data.csv', index=False)
```

Analyse Merged Data File

Total Sales by Life Stage and Customer Status

The three highest total sales are:

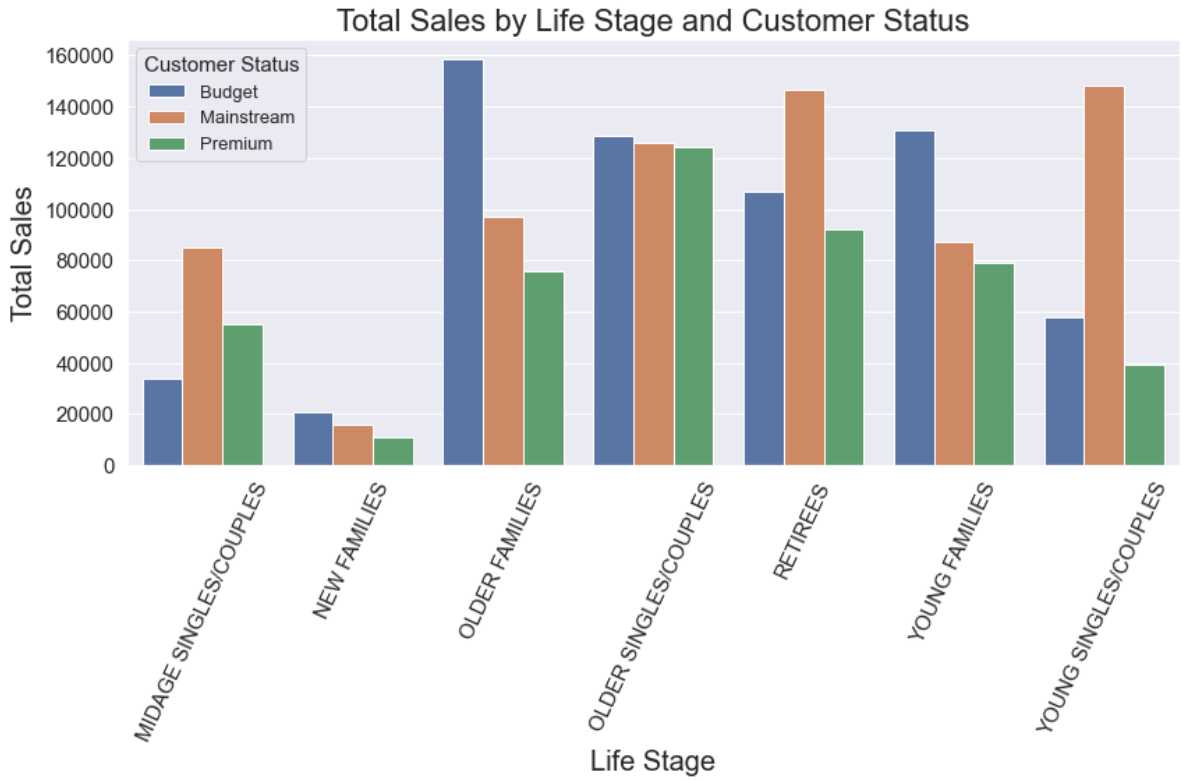
- Older Families - Budget
- Young Singles/Couples - Mainstream
- Retirees - Mainstream.

In [36]:

```
lp_tot_sales = (data.groupby(['LIFESTAGE','PREMIUM_CUSTOMER']).agg({'TOT_SALES': 'sum'})).r
sns.set(rc={'figure.figsize':(12,8)})
sns.barplot(x='LIFESTAGE', y='TOT_SALES', hue='PREMIUM_CUSTOMER', data=lp_tot_sales)
plt.xlabel('Life Stage', fontsize = 20)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.ylabel('Total Sales', fontsize = 20)
plt.title('Total Sales by Life Stage and Customer Status', fontsize = 22)
plt.xticks(rotation=65)
plt.tight_layout()
plt.legend(title='Customer Status', title_fontsize=15, loc=2, fontsize=13)
```

Out[36]:

```
<matplotlib.legend.Legend at 0x2760c084508>
```



Total Customers by Life Stage and Customer Status

The two groups containing the highest number of customers are:

- Young Singles/Couples - Mainstream
- Retirees - Mainstream

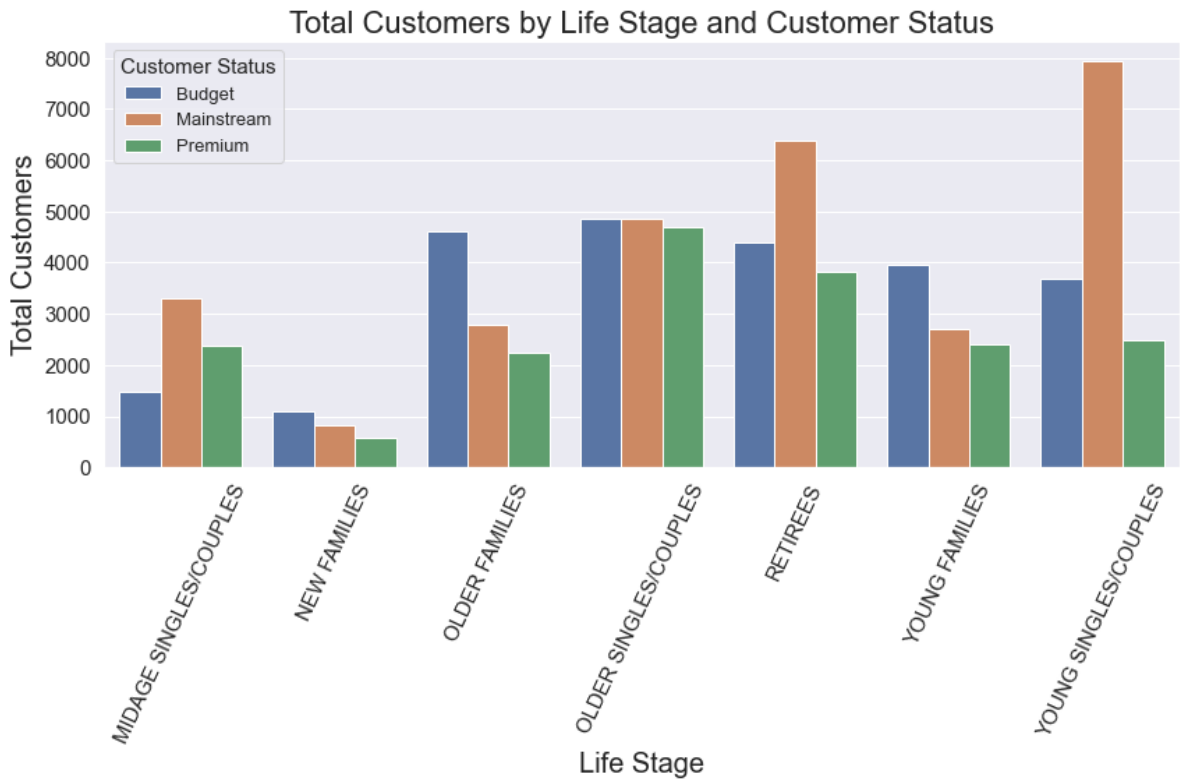
However, this summary does not account for high total sales for the Older Families - Budget group.

In [37]:

```
lp_tot_cust = (data.groupby(['LIFESTAGE','PREMIUM_CUSTOMER']).agg({'LYLTY_CARD_NBR': pd.Ser
sns.set(rc={'figure.figsize':(12,8)})
sns.barplot(x="LIFESTAGE", y='LYLTY_CARD_NBR', hue='PREMIUM_CUSTOMER', data=lp_tot_cust)
plt.xlabel('Life Stage', fontsize = 20)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.ylabel('Total Customers', fontsize = 20)
plt.title('Total Customers by Life Stage and Customer Status', fontsize = 22)
plt.xticks(rotation=65)
plt.tight_layout()
plt.legend(title='Customer Status', title_fontsize=15, loc=2, fontsize=13)
```

Out[37]:

```
<matplotlib.legend.Legend at 0x2760bb8d7c8>
```



Average Number of Units Per Customer by Life Stage and Customer Status

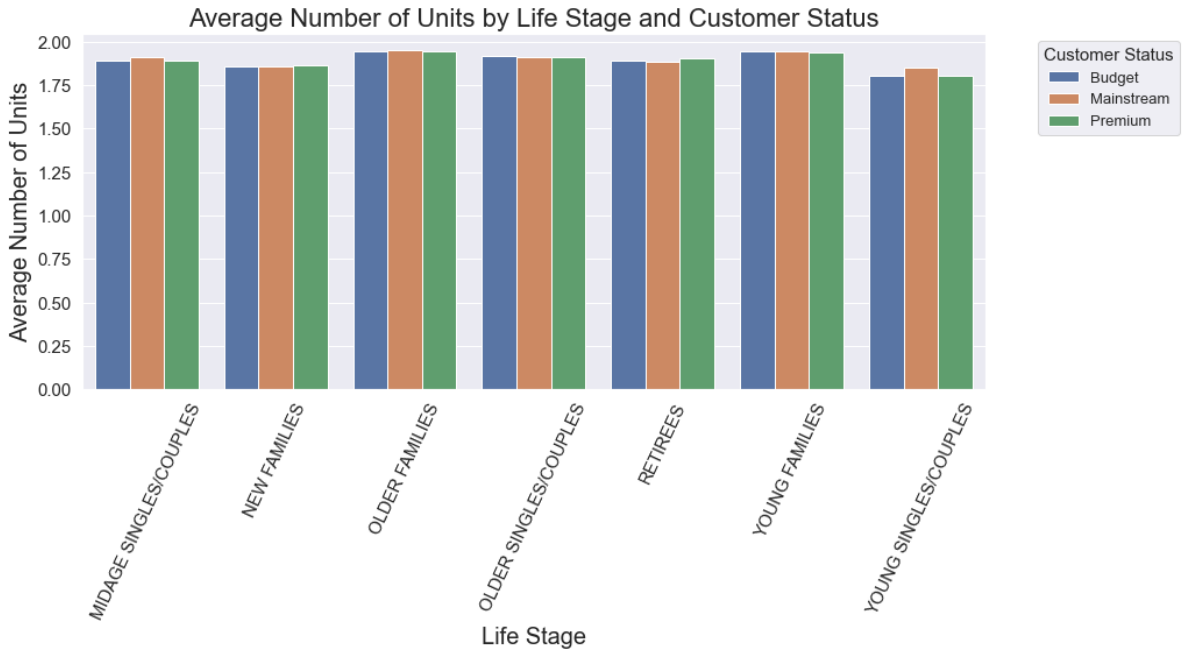
Older and younger families on average buy the highest qty of chip packets. However, the range of average values does not differ very much. All values are within a range of 1.75 to 2.

In [38]:

```
lp_avg_qty = (data.groupby(['LIFESTAGE','PREMIUM_CUSTOMER']).agg({'PROD_QTY': 'mean'})).res
sns.set(rc={'figure.figsize':(12,8)})
sns.barplot(x='LIFESTAGE', y='PROD_QTY', hue='PREMIUM_CUSTOMER', data=lp_avg_qty)
plt.xlabel('Life Stage', fontsize = 20)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.ylabel('Average Number of Units', fontsize = 20)
plt.title('Average Number of Units by Life Stage and Customer Status', fontsize = 22)
plt.xticks(rotation=65)
plt.tight_layout()
plt.legend(title='Customer Status', title_fontsize=15, bbox_to_anchor=(1.05, 1), loc=2, fon
```

Out[38]:

```
<matplotlib.legend.Legend at 0x2760c044d48>
```



Average Price Per Unit by Life Stage and Customer Status

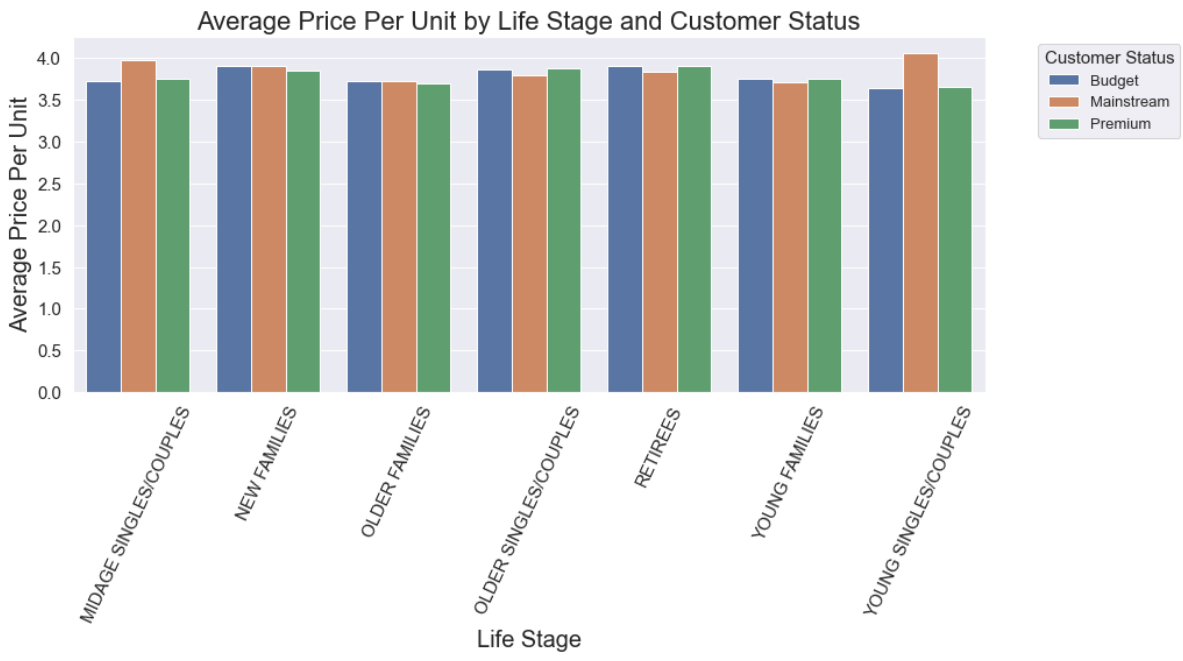
Across most life stage groups, the difference betwwen the customer status groups for the average price per unit appears to be insignificant. However, the mainstream groups for mid-age and young singles/couples is noticeably higher than the premium and budget groups. The mainstream young singles/couples group is also the only sub-group that has an average above four dollars while mainstream mid-age singles/couples is just under four dollars.

In [39]:

```
lp_avg_price = (data.groupby(['LIFESTAGE','PREMIUM_CUSTOMER']).agg({'UNIT_PRICE': 'mean'}))
sns.set(rc={'figure.figsize':(12,8)})
sns.barplot(x='LIFESTAGE', y='UNIT_PRICE', hue='PREMIUM_CUSTOMER', data=lp_avg_price)
plt.xlabel('Life Stage', fontsize = 20)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.ylabel('Average Price Per Unit', fontsize = 20)
plt.title('Average Price Per Unit by Life Stage and Customer Status', fontsize = 22)
plt.xticks(rotation=65)
plt.tight_layout()
plt.legend(title='Customer Status', title_fontsize=15, bbox_to_anchor=(1.05, 1), loc=2, fon
```

Out[39]:

```
<matplotlib.legend.Legend at 0x2760c067ac8>
```



Independent T-Test Between Mainstream and Remaining Customer Groups for Mid-Age and Young Singles/Couples

The p-value for the t-test is much less than 0.05 or even 0.01, confirming significance. Therefore, the average unit price for mainstream, young and mid-age singles and couples is significantly higher than that of budget or premium young and mid-age singles and couples.

In [40]:

```
midage_young_m = data[((data['LIFESTAGE'] == 'MIDAGE SINGLES/COUPLES') | (data['LIFESTAGE']
                       (data['PREMIUM_CUSTOMER'] == 'Mainstream')]

midage_young_bp = data[((data['LIFESTAGE'] == 'MIDAGE SINGLES/COUPLES') | (data['LIFESTAGE'
                       (data['PREMIUM_CUSTOMER'] != 'Mainstream')]

ttest_ind(midage_young_m['UNIT_PRICE'], midage_young_bp['UNIT_PRICE'])
```

Out[40]:

```
Ttest_indResult(statistic=38.301848987923314, pvalue=0.0)
```

Prepare Data for Affinity Analysis

In [41]:

```
# Split dataset into mainstream young single/couples and all remaining customers
young_sc_mainstream = data[(data['LIFESTAGE'] == 'YOUNG SINGLES/COUPLES') & (data['PREMIUM_
other = data[-((data['LIFESTAGE'] == 'YOUNG SINGLES/COUPLES') & (data['PREMIUM_CUSTOMER'] =
```

In [42]:

```
# Calculations required for affinity analyses
qty_y_sc_m = sum(young_sc_mainstream.PROD_QTY)
qty_other = sum(other.PROD_QTY)
```

Preferred Brand of Mainstream and Young Singles/Couples Versus Other Customers

Mainstream young singles/couples are approximately 23% more likely to buy the Tyrrells brand compared to the remaning types of customers in the dataset. Of the top five brands that mainstream young singles/couples are more likely to buy, Twisties and Tostitos, are the only brands that cost more than four dollars per packet. The other three brands which include Tyrrells, Doritos and Kettle sell packets both under and over four dollars each.

In [43]:

```
# Perform affinity analysis of brands for mainstream young singles/couples against other cu
qty_y_sc_m_brand = (young_sc_mainstream.groupby(['BRAND'])).agg({'PROD_QTY': 'sum'}).reset_
    columns={"PROD_QTY": "Target_Segment"})
qty_y_sc_m_brand['Target_Segment'] = qty_y_sc_m_brand['Target_Segment'].div(qty_y_sc_m)
qty_other_brand = (other.groupby(['BRAND'])).agg({'PROD_QTY': 'sum'}).reset_index().rename(
    columns={"PROD_QTY": "Other"})
qty_other_brand['Other'] = qty_other_brand['Other'].div(qty_other)
brand_proportions = pd.merge(qty_y_sc_m_brand, qty_other_brand, on=['BRAND'])
brand_proportions['Affinity_To_Brand'] = brand_proportions.Target_Segment / brand_proportio
brand_proportions.sort_values(by='Affinity_To_Brand', ascending=False).reset_index(drop=Tru
```

Out[43]:

	BRAND	Target_Segment	Other	Affinity_To_Brand
0	Tyrrells	0.031307	0.025387	1.233215
1	Twisties	0.045824	0.037426	1.224402
2	Doritos	0.121806	0.099872	1.219616
3	Kettle	0.196445	0.163584	1.200882
4	Tostitos	0.045057	0.037526	1.200698
5	Pringles	0.118491	0.099437	1.191617
6	Cobs	0.044290	0.038584	1.147889
7	Infuzions	0.064176	0.056386	1.138160
8	Thins	0.059903	0.056308	1.063839
9	Grain	0.032458	0.030817	1.053246
10	Cheezels	0.017831	0.018425	0.967771
11	Smiths	0.098852	0.129107	0.765660
12	French	0.003917	0.005690	0.688427
13	Cheetos	0.007971	0.011923	0.668508
14	RRD	0.048016	0.072583	0.661529
15	NCC	0.019447	0.030487	0.637889
16	CCs	0.011093	0.018671	0.594144
17	Sunbites	0.006300	0.012431	0.506802
18	WW	0.021091	0.042537	0.495816
19	Woolworths	0.002821	0.006302	0.447689
20	Burger	0.002903	0.006518	0.445446

In [44]:

```
# Brands that have products greater than $4 per unit
brand_unit_price = data[['BRAND', 'UNIT_PRICE']].drop_duplicates().reset_index(drop=True)
brand_unit_price = brand_unit_price[brand_unit_price.UNIT_PRICE > 4].sort_values(by = 'BRAN
brand_unit_price.BRAND.unique()
```

Out[44]:

```
array(['Cheezels', 'Doritos', 'Kettle', 'Smiths', 'Tostitos', 'Twisties',
       'Tyrrells'], dtype=object)
```

In [45]:

```
# Brands that have products less than $4 per unit
brand_unit_price = data[['BRAND', 'UNIT_PRICE']].drop_duplicates().reset_index(drop=True)
brand_unit_price = brand_unit_price[brand_unit_price.UNIT_PRICE < 4].sort_values(by = 'BRAN
brand_unit_price.BRAND.unique()
```

Out[45]:

```
array(['Burger', 'CCs', 'Cheetos', 'Cheezels', 'Cobs', 'Doritos',
       'French', 'Grain', 'Infuzions', 'Kettle', 'NCC', 'Pringles', 'RRD',
       'Smiths', 'Sunbites', 'Thins', 'Tyrrells', 'WW', 'Woolworths'],
      dtype=object)
```

In [46]:

```
# Brands that have products equal to $4 per unit
brand_unit_price = data[['BRAND', 'UNIT_PRICE']].drop_duplicates().reset_index(drop=True)
brand_unit_price = brand_unit_price[brand_unit_price.UNIT_PRICE == 4].sort_values(by = 'BRA
brand_unit_price.BRAND.unique()
```

Out[46]:

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

Preferred Pack Size of Mainstream and Young Singles/Couples Versus Other Customers

Mainstream young singles/couples are approximately 27% more likely to buy brands in a 270 gram pack size compared to the remaning types of customers in the dataset. Twisties is the only brand that is sold in a 270 gram packet size.

In [47]:

```
# Perform affinity analysis of pack sizes for mainstream young singles/couples against othe
qty_y_sc_m_pack = (young_sc_mainstream.groupby(['PACK_SIZE'])).agg({'PROD_QTY': 'sum'}).res
    columns={"PROD_QTY": "Target_Segment"})
qty_y_sc_m_pack['Target_Segment'] = qty_y_sc_m_pack['Target_Segment'].div(qty_y_sc_m)
qty_other_pack = (other.groupby(['PACK_SIZE'])).agg({'PROD_QTY': 'sum'}).reset_index().rena
    columns={"PROD_QTY": "Other"})
qty_other_pack['Other'] = qty_other_pack['Other'].div(qty_other)
pack_proportions = pd.merge(qty_y_sc_m_pack, qty_other_pack, on=['PACK_SIZE'])
pack_proportions['Affinity_To_Brand'] = pack_proportions.Target_Segment / pack_proportions.
pack_proportions.sort_values(by='Affinity_To_Brand', ascending=False).reset_index(drop=True
```

Out[47]:

	PACK_SIZE	Target_Segment	Other	Affinity_To_Brand
0	270	0.031581	0.024797	1.273574
1	380	0.031910	0.025280	1.262270
2	330	0.060807	0.049565	1.226810
3	134	0.118491	0.099437	1.191617
4	110	0.105453	0.088723	1.188571
5	210	0.028897	0.024822	1.164151
6	135	0.014654	0.012920	1.134219
7	250	0.014243	0.012629	1.127848
8	170	0.080145	0.080022	1.001527
9	175	0.253006	0.266794	0.948319
10	150	0.164151	0.173374	0.946803
11	165	0.055219	0.061527	0.897483
12	190	0.007423	0.012294	0.603777
13	180	0.003561	0.005995	0.594004
14	160	0.006355	0.012226	0.519773
15	90	0.006300	0.012431	0.506802
16	125	0.002986	0.005965	0.500520
17	200	0.008902	0.018434	0.482904
18	70	0.003013	0.006247	0.482295
19	220	0.002903	0.006518	0.445446

In [48]:

```
# Determine how many brands sell their product in 270 gram pack sizes
pack_size_270 = data[data['PACK_SIZE'] == 270].PROD_NAME.unique()
pack_size_270
```

Out[48]:

```
array(['Twisties Cheese     270g', 'Twisties Chicken270g'], dtype=object)
```

Conclusion

The sale of chips throughout the year is fairly consistent other than the peak that occurs just before Christmas. Although the pack sizes of all the brands range between 70 and 380 grams, the most frequent purchases lie mostly within 100 and 200 gram pack sizes.

When examining the variation of customers who made purchases over the financial year, a majority of these customers were either retirees, older singles/couples, and young singles/couples. The number of customers in each of these three groups is approximately the same. Therefore, all could be good candidates for further analysis rather than there being just one standout group. For customer status, mainstream is most common, although not a majority of customers, while premium is the least common.

When customers were grouped by both life stage and customer status, it was found that mainstream young singles/couples and mainstream retirees represented the most frequent purchasers. This is therefore fairly consistent with results observed for these attributes when examined separately. Two of the three highest total sales for the year also came from these groups. However, budget older families were responsible for the highest total sales despite its smaller representation across all customers.

The difference in the average purchase price per unit for the three various customer status groups was consistent across most of the life stage groups. However, the most noticeable exception was mainstream young singles/couples who were the only sub-group above four dollars. The difference between mainstream and the other customer status groups was shown to be significantly different for mainstream young singles/couples. Further analysis of mainstream young singles/couples showed that the top five brands that they were more likely to purchase over other customers were Tyrrells, Twisties, Doritos, Kettle and Tostitos. Of these five brands, two only sold packets greater than 4 dollars each, with the other three selling packets both higher and lower than four dollars. This may help explain why the average price per unit for mainstream young singles/couples is the highest and the only one with an average price per unit higher than four dollars.

Mainstream young singles/couples were also most likely to buy pack sizes of 270 grams compared to other customers. However, Twisties is the only brand that sells in this pack size so this could also just represent a preference for this brand.