Quantium Virtual Internship - Retail Strategy and Analytics - Task 1

DATA PROCESSING

Importing the nessesary Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Data Processing of the QVI_purchase_behaviour.csv

Loading the QVI_purchase_behaviour.csv data set and vewing the first five rows

```
purchase = pd.read_csv("QVI_purchase_behaviour.csv")
purchase.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

```
purchase.shape
```

```
(72637, 3)
```

```
purchase.describe().T
```

	count	mean	std	min	25%	50%	75%

```
#checking the Data type
purchase.dtypes
```

```
LYLTY_CARD_NBR        int64
LIFESTAGE            object
PREMIUM_CUSTOMER     object
dtype: object
```

```
#checking the unique values for the LYTY_CARD_NBR to check if its consistent with the numb
#using value_counts() to count unique values and their frequencies
value_counts = purchase["LYLTY_CARD_NBR"].value_counts()
#filtering values that have a countgreater than one
filtered_counts = value_counts[value_counts>1]

print ("Unique values with counts greater than one: ")
print(filtered_counts)

print("\nTotal counts of allunique values :")
print (value_counts.sum())

#values are 72637 same as number of rows in this data.
```

```
Unique values with counts greater than one:
Series([], Name: LYLTY_CARD_NBR, dtype: int64)

Total counts of allunique values :
72637
```

Checking Missing Values and rectifying it

```
#count of missing data
Missing_values =purchase.isna().sum()
print("{} ".format(Missing_values))
```

```
LYLTY_CARD_NBR      0
LIFESTAGE           0
PREMIUM_CUSTOMER    0
dtype: int64
```

Data Processing of the QVI_transaction_data.xlsx

Loading the QVI_transaction_data.xlsx data set and vewing the first five rows

```
transaction = pd.read_excel("QVI_transaction_data.xlsx")
transaction.head()
```

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TO
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	3	

```
transaction.shape
```

```
    (264836, 8)
```

```
#Converting the date  to date formart
transaction['DATE'] = pd.to_datetime(transaction['DATE'], origin='1899-12-30', unit='D')
```

```
transaction.describe(include='all').T
```

```
     C:\Users\josep\AppData\Local\Temp\ipykernel_11624\2251822836.py:1
```

```
#cheking the data types
transaction.dtypes
```

```
     DATE              datetime64[ns]
     STORE_NBR                  int64
     LYLTY_CARD_NBR             int64
     TXN_ID                     int64
     PROD_NBR                   int64
     PROD_NAME                 object
     PROD_QTY                   int64
     TOT_SALES                float64
     dtype: object
```

Examining the PROD_NAME COLUMN

```
Product_summary = transaction['PROD_NAME'].describe()
Product_summary
```

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

```
#Examine the words in PROD_NAME to filter out non-chip products
product_words =transaction['PROD_NAME'].str.split().explode()
product_words = product_words[~product_words.str.contains(r'[0-9&/]')]
```

```
#Remove salsa Products
transaction = transaction[~transaction['PROD_NAME'].str.contains('Salsa')]
```

Checking Missing Values and rectifying it

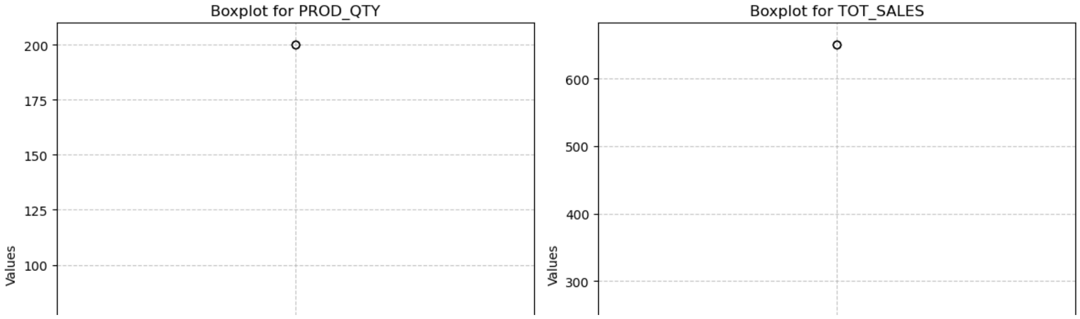
```
#count of missing data
Missing_values =transaction.isna().sum()
print("{} ".format(Missing_values))
```

```
     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
```

Checking For Outliers and rectifying it

Because of luck of good domain Knowledge I Will only inspect PROD_QTY AND TOT_SALES Columns for outliers by using the box plot method to visualize the outliers

```
plt.figure(figsize=(12,6))

#Boxplot for PROD_QTY
plt.subplot(1,2,1)
plt.boxplot(transaction['PROD_QTY'])
plt.title('Boxplot for PROD_QTY')
plt.ylabel('Values')
plt.grid(True, linestyle ='--', alpha = 0.7)

#Boxplot for TOT_SALES
plt.subplot(1,2,2)
plt.boxplot(transaction['TOT_SALES'])
plt.title('Boxplot for TOT_SALES')
plt.ylabel('Values')
plt.grid(True, linestyle ='--', alpha = 0.7)


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



Removing the Outliers using the IQR method

```
v1 = [200]
v2 = [650]
for columns, values in zip(["PROD_QTY","TOT_SALES"],[v1, v2]):
    transaction = transaction[~transaction[columns].isin(values)]

z = np.max(transaction["PROD_NAME"])
z
```

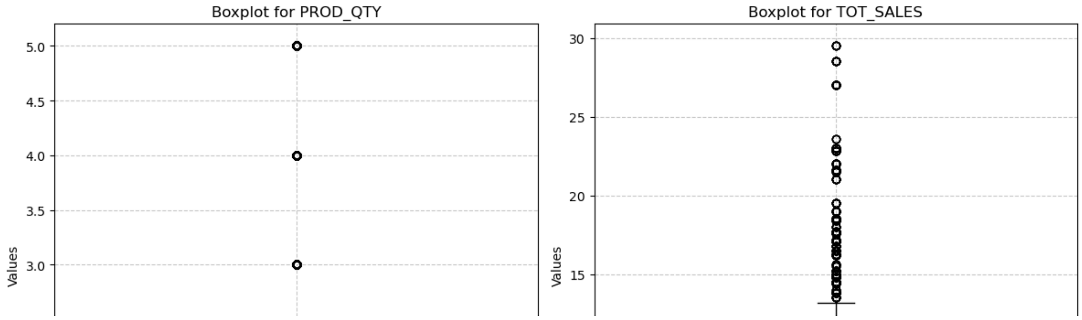
```
'Woolworths Cheese   Rings 190g'
```

Checking if the outliers have been removed

```
#count of missing data
Missing_values =transaction.isna().sum()
print("{} ".format(Missing_values))
```

```
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
```

```
plt.figure(figsize=(12,6))

#Boxplot for PROD_QTY
plt.subplot(1,2,1)
plt.boxplot(transaction['PROD_QTY'])
plt.title('Boxplot for PROD_QTY')
plt.ylabel('Values')
plt.grid(True, linestyle ='--', alpha = 0.7)

#Boxplot for TOT_SALES
plt.subplot(1,2,2)
plt.boxplot(transaction['TOT_SALES'])
plt.title('Boxplot for TOT_SALES')
plt.ylabel('Values')
plt.grid(True, linestyle ='--', alpha = 0.7)


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



Check the trasactions by date

```
transactions_by_day = transaction.groupby('DATE').size().reset_index(name='N')
```

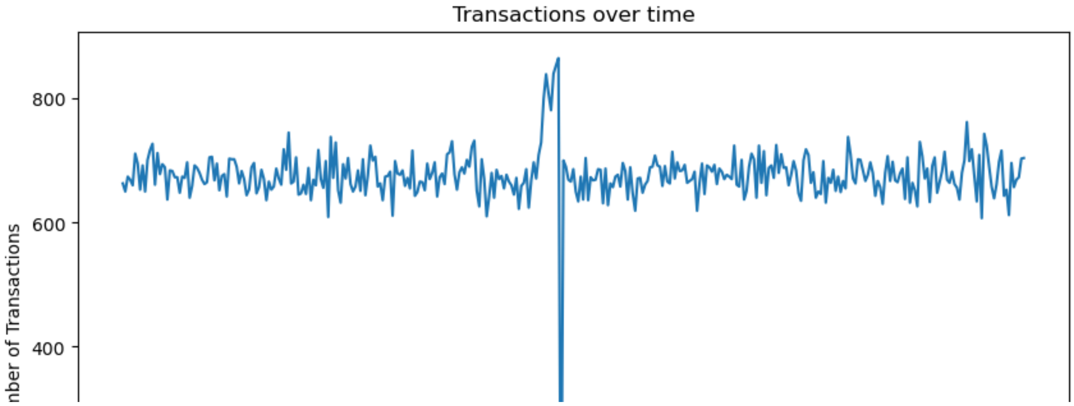
Create a sequence of dates and join it to fill missing dates

```
date_range = pd.date_range(start='2018-07-01', end = '2019-06-30')
all_dates = pd.DataFrame({'DATE':date_range})
```

```
transactions_by_day =  all_dates.merge(transactions_by_day, on ='DATE', how = 'left')
transactions_by_day['N'] = transactions_by_day['N'].fillna(0)
```

Transaction over time

```
#Plot Transactions over time
plt.figure(figsize=(10,6))
plt.plot(transactions_by_day['DATE'],transactions_by_day['N'])
plt.xlabel('Date')
plt.ylabel('Number of Transactions')
plt.title('Transactions over time')
plt.xticks(rotation = 90)
plt.show()
plt.clf()
```

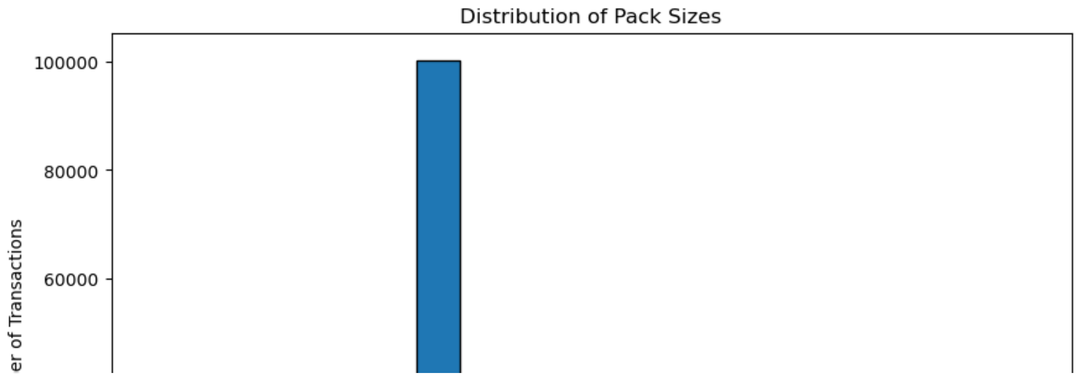


Create a pack size Column

```
transaction['PACK_SIZE'] = transaction['PROD_NAME'].str.extract('(\d+)g').astype(float)
transaction.head()
```

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME

```
#Plot a Histogram of Pack Size
plt.figure(figsize=(10, 6))
plt.hist(transaction['PACK_SIZE'], bins=20, edgecolor='k')
plt.xlabel('Pack Size (g)')
plt.ylabel('Number of Transactions')
plt.title('Distribution of Pack Sizes')
plt.show()
plt.clf()
```



Create a Brand Column

```
transaction['BRAND'] = transaction['PROD_NAME'].str.split().str[0]
```

```
# Clean Brand names
transaction['BRAND'] = transaction['BRAND'].replace('RRD','Red Rock Deli')
```

```
Missing_values =transaction.isna().sum()
print("{} ".format(Missing_values))
```

```
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         6064
BRAND                0
dtype: int64
```

if the missing values constitute of less than 10% we drop the rows if more than 10% we fill them with zero

```
percentage=  transaction['PACK_SIZE'].isna().sum() / transaction['BRAND'].count() *100
print("{}%".format(round(percentage,2)))
```

```
    2.46%
```

```
transaction = transaction.dropna()
```

```
transaction.isna().sum()
```

```
    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
    dtype: int64
```

MERGING the TWO data Purchase and Transaction data to data

```
data = pd.merge(purchase,transaction, on = "LYLTY_CARD_NBR", how = 'left')
data.head()
data.shape
```

```
      (242486, 12)
```

```
#count of missing data
Missing_values =data.isna().sum()
print("{}".format(Missing_values))
```

```
      LYLTY_CARD_NBR         0
      LIFESTAGE              0
      PREMIUM_CUSTOMER       0
      DATE                1810
      STORE_NBR           1810
      TXN_ID              1810
      PROD_NBR            1810
      PROD_NAME           1810
      PROD_QTY            1810
      TOT_SALES           1810
      PACK_SIZE           1810
      BRAND               1810
      dtype: int64
```

```
percentage=  data['PACK_SIZE'].isna().sum() / data['BRAND'].count() *100
print("{}%".format(round(percentage,2)))
```

```
      0.75%
```

```
#count of missing data
data = data.dropna()
```

```
Missing_values =data.isna().sum()
print("{}".format(Missing_values))
```

```
      LYLTY_CARD_NBR      0
      LIFESTAGE           0
      PREMIUM_CUSTOMER    0
      DATE                0
      STORE_NBR           0
      TXN_ID              0
      PROD_NBR            0
      PROD_NAME           0
      PROD_QTY            0
      TOT_SALES           0
      PACK_SIZE           0
      BRAND               0
      dtype: int64
```

```
data.shape
```

```
      (240676, 12)
```

```
data.describe(include='all').T
```

```
C:\Users\josep\AppData\Local\Temp\ipykernel_11624\3536075253.py:1
  data.describe(include='all').T
```

	count	unique	top	freq	first
LYLTY_CARD_NBR	240676.0	NaN	NaN	NaN	NaT
LIFESTAGE	240676	7	OLDER SINGLES/COUPLES	49547	NaT
PREMIUM_CUSTOMER	240676	3	Mainstream	92729	NaT
DATE	240676	364	2018-12-24 00:00:00	847	2018-07-01
STORE_NBR	240676.0	NaN	NaN	NaN	NaT
TXN_ID	240676.0	NaN	NaN	NaN	NaT
PROD_NBR	240676.0	NaN	NaN	NaN	NaT
PROD_NAME	240676	102	Kettle Mozzarella Basil & Pesto 175g	3304	NaT
PROD_QTY	240676.0	NaN	NaN	NaN	NaT
TOT_SALES	240676.0	NaN	NaN	NaN	NaT
PACK_SIZE	240676.0	NaN	NaN	NaN	NaT
BRAND	240676	28	Kettle	41288	NaT

DATA ANALYSIS

Metrics

1 - Total sales.

2 - some of the top product names with PROD_QTY = 5.0.

3 - Top and Bottom 10 products and the life stages.

4 - Top and Bottom 10 Products by the premium customers.

5 - Top and Bottom 10 stores by PROD_QTY by TOT_SALES.

6 - Total Sales by Customer segment (LIFESTAGE and PREMIUM_CUSTOMER) - here we want to find out which customers spend the most on chips.

7 - Number of Customers in each Segment - here we're intrested in knowing how many customers are in each of this customer groups.

8 - Average Number of Chips Bought per Customer by Segment - we are looking at how many chips, on average, each customerin this segmenst buys.

9 - Average chip price by Customer Segment - this metrics helps us understand the average price at which chips are sold to customers in different segments.

10 - Statistical Test for Price Differences - we perform a test to check if there's a significant difference in average chip prices between specific customer groups.

11 - Deep Dive into Mainstream,Young singles/Couples - we focus on a specific customer segment, "Mainstream, Young singles/couples," to gain deeper insights.

1 - Total sales

```
# Total sales
total_sales = np.sum(data['TOT_SALES'])
print ("The total sales is: {}".format(round(total_sales, 1)))
max_sales = max(data['TOT_SALES'])
print ("The highest sale is: {}".format(round(max_sales, 0)))
min_sales =min(data['TOT_SALES'])
print ("The lowest sale is: {}".format(round(min_sales, 0)))
```

```
The total sales is: 1767825.9
The highest sale is: 30.0
The lowest sale is: 2.0
```

2 - Products (PROD_NAME) that have a quantity of 5.0 (PROD_QTY)

```
filterd_values = data[data['PROD_QTY'] == 5.0]['PROD_NAME']
filterd_value =filterd_values.count()
print ('Count of all products with PROD_QTY 5.0: {}'.format(filterd_value))
print('list of five of the products')
filterd_values.head()
```

```
Count of all products with PROD_QTY 5.0: 405
list of five of the products
891       Smiths Chip Thinly  S/Cream&Onion 175g
1067            Thins Chips Salt &  Vinegar 175g
1103            Pringles Sthrn FriedChicken 134g
1270       Grain Waves         Sweet Chilli 210g
1637    Kettle Sensations   Camembert & Fig 150g
Name: PROD_NAME, dtype: object
```

3 - Top 10 and Bottom 10 products and the life stages

```
top_five = data.nlargest(10, 'TOT_SALES')[['PROD_NAME','LIFESTAGE','TOT_SALES']].rename(co
    'PROD_NAME': 'Product',
    'LIFESTAGE':'Life Stage',
    'TOT_SALES': 'Total Sales'
}).reset_index(drop=True)
top_five.index += 1
print ("The Top 10 products")
top_five
```

```
The Top 10 products
```

	Product	Life Stage	Total Sales
1	Smiths Crnkle Chip Orgnl Big Bag 380g	RETIREES	29.5
2	Smiths Crnkle Chip Orgnl Big Bag 380g	YOUNG FAMILIES	29.5
3	Smiths Crnkle Chip Orgnl Big Bag 380g	OLDER FAMILIES	29.5
4	Smiths Crnkle Chip Orgnl Big Bag 380g	MIDAGE SINGLES/COUPLES	29.5
5	Smiths Crnkle Chip Orgnl Big Bag 380g	RETIREES	29.5
6	Smiths Crnkle Chip Orgnl Big Bag 380g	OLDER FAMILIES	29.5
	Smiths Crnkle Chip Orgnl Big Bag	OLDER	

```
bottom_five = data.nsmallest(10, 'TOT_SALES')[['PROD_NAME','LIFESTAGE','TOT_SALES']].renam
    'PROD_NAME': 'Product',
    'LIFESTAGE':'Life Stage',
    'TOT_SALES': 'Total Sales'
}).reset_index(drop=True)

print ("The bottom 10 products")
bottom_five
```

```
The bottom 10 products
```

	Product	Life Stage	Total Sales
0	WW Crinkle Cut Chicken 175g	YOUNG FAMILIES	1.7
1	Sunbites Whlegrn Crisps Frch/Onin 90g	OLDER SINGLES/COUPLES	1.7
2	Sunbites Whlegrn Crisps Frch/Onin 90g	YOUNG SINGLES/COUPLES	1.7
3	Sunbites Whlegrn Crisps Frch/Onin 90g	NEW FAMILIES	1.7
4	Snbts Whlgrn Crisps Cheddr&Mstrd 90g	RETIREES	1.7
5	Snbts Whlgrn Crisps Cheddr&Mstrd 90g	NEW FAMILIES	1.7
6	WW Crinkle Cut Original 175g	NEW FAMILIES	1.7
7	Snbts Whlgrn Crisps Cheddr&Mstrd 90g	OLDER SINGLES/COUPLES	1.7
	Sunbites Whlegrn Crisps Frch/Onin	YOUNG	

4 - Top 10 and Bottom 10 PRODUCTS TAKEN BY THE PREMIUMS

```
#The Top 10 Products

unique_premium = data['PREMIUM_CUSTOMER'].unique()

count_dict ={}
for premium_value in unique_premium:
    filtered_df = data[data['PREMIUM_CUSTOMER'] == premium_value]
    count = filtered_df['PROD_NAME'].unique()
    count_dict[premium_value] = count



top_10_values = pd.DataFrame(count_dict[max(count_dict, key = lambda k : len(count_dict[k]
top_10_values.index += 1
top_10_values
print ('The Top 10 Products')
top_10_values
```

```
The Top 10 Products
```

	0
1	Natural Chip Compny SeaSalt175g
2	Doritos Cheese Supreme 330g
3	GrnWves Plus Btroot & Chilli Jam 180g
4	RRD Sweet Chilli & Sour Cream 165g
5	CCs Tasty Cheese 175g
6	Infuzions Mango Chutny Papadums 70g
7	Smiths Crinkle Chips Salt & Vinegar 330g
8	Cobs Popd Sea Salt Chips 110g
9	Natural ChipCo Sea Salt & Vinegr 175g
10	Burger Rings 220g

```
#the Bottom 10 Products
unique_premium = data['PREMIUM_CUSTOMER'].unique()

count_dict ={}
for premium_value in unique_premium:
    filtered_df = data[data['PREMIUM_CUSTOMER'] == premium_value]
    count = filtered_df['PROD_NAME'].unique()
    count_dict[premium_value] = count



botto_10_values = pd.DataFrame(count_dict[min(count_dict, key = lambda k : len(count_dict[
botto_10_values.index += 1
botto_10_values

print ('The Top 10 Products')
botto_10_values
```

```
The Top 10 Products
```

	0
1	Natural Chip Compny SeaSalt175g
2	Doritos Cheese Supreme 330g
3	GrnWves Plus Btroot & Chilli Jam 180g
4	RRD Sweet Chilli & Sour Cream 165g
5	CCs Tasty Cheese 175g

5 - Top 10 and Bottom 10 stores with PROD_QTY 5.0 by TOT_SALES

```
#Top 10 stores
#FILTERING BY PROD_QTY
filterd_df = data[data['PROD_QTY'] == 5.0]
#SORT BY TOT_SALES IN DESECNDING ORDER AND GET TOP 10 ROWS
top_10_sales = filterd_df.sort_values(by = 'TOT_SALES',ascending=False).head(10)
#rename columns
top_10_sales = top_10_sales[['STORE_NBR','PROD_QTY','TOT_SALES']].rename(columns={'STORE_N
top_10_sales.index +=1
top_10_sales
print('The Top 10 Stores')
top_10_sales
```

```
The Top 10 Stores
```

	Store number	Product Quality	Total Sales
1	194.0	5.0	29.5
2	190.0	5.0	29.5
3	24.0	5.0	29.5
4	94.0	5.0	29.5
5	44.0	5.0	29.5
6	49.0	5.0	29.5
7	118.0	5.0	29.5
8	154.0	5.0	28.5
9	96.0	5.0	28.5
10	181.0	5.0	28.5

```
#The Bottom 10
#FILTERING BY PROD_QTY
filterd_df = data[data['PROD_QTY'] == 1.0]
#SORT BY TOT_SALES IN DESECNDING ORDER AND GET TOP 10 ROWS
```

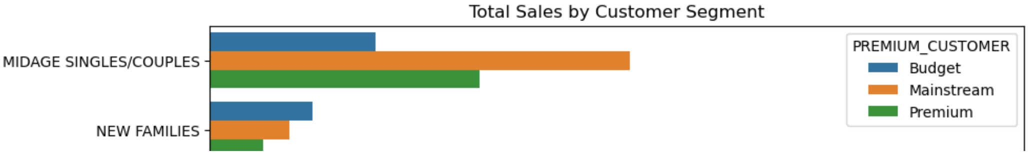
```
bottom_10_sales = filterd_df.sort_values(by = 'TOT_SALES',ascending=True).head(10)
#rename columns
bottom_10_sales = bottom_10_sales[['STORE_NBR','PROD_QTY','TOT_SALES']].rename(columns={'S
bottom_10_sales.index +=1
bottom_10_sales

print('The Bottom 10 Stores')
bottom_10_sales
```

```
The Bottom 10 Stores
```

	Store number	Product Quality	Total Sales
1	214.0	1.0	1.7
2	38.0	1.0	1.7
3	98.0	1.0	1.7
4	110.0	1.0	1.7
5	233.0	1.0	1.7
6	98.0	1.0	1.7
7	262.0	1.0	1.7
8	51.0	1.0	1.7
9	70.0	1.0	1.7
10	50.0	1.0	1.7

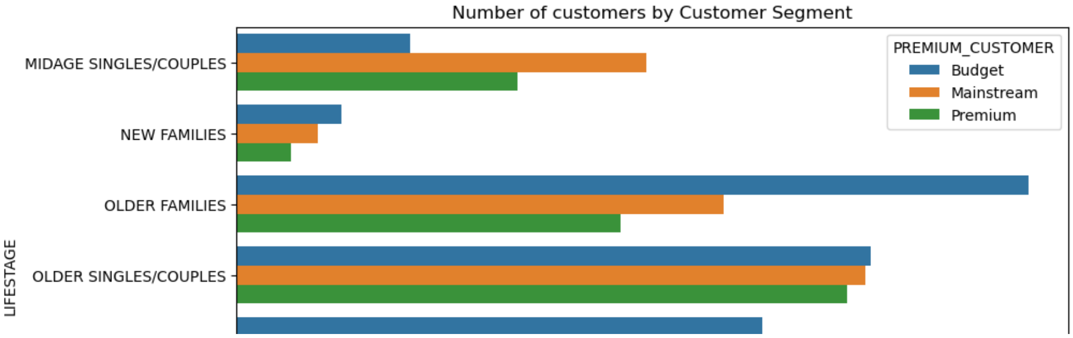
6 - Total Sales by Customer segment (LIFESTAGE and PREMIUM_CUSTOMER)

```
#Calculate the total sales by customers
sales_by_segment = data.groupby(['LIFESTAGE','PREMIUM_CUSTOMER'])['TOT_SALES'].sum().reset

#create a bar plot
plt.figure(figsize=(10,6))
sns.barplot(data=sales_by_segment, x = 'TOT_SALES', y = 'LIFESTAGE', hue = 'PREMIUM_CUSTOM
plt.title('Total Sales by Customer Segment')
plt.xlabel('Total Sales')
plt.ylabel('LIFESTAGE')
#adding data label
#for p in ax.patches:
  #  width = p.get_width()
   # plt.annotate(f'{width:.1f}', (width, p.get_y() + p.get_height() / 2), ha='left', va='

plt.show()
plt.clf()
```



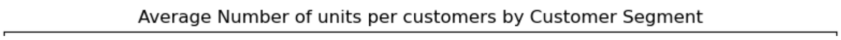
7 - Number of Customers in each Segment

```
#calculate the number of customer in each segment
customer_count_segment = data.groupby(['LIFESTAGE','PREMIUM_CUSTOMER'])['LYLTY_CARD_NBR'].

#create a bar plot
plt.figure(figsize=(10,6))
sns.barplot(data=customer_count_segment, x = 'LYLTY_CARD_NBR', y = 'LIFESTAGE', hue = 'PRE
plt.title('Number of customers by Customer Segment')
plt.xlabel('Number of Customers')
plt.ylabel('LIFESTAGE')
plt.show()
plt.clf()
```



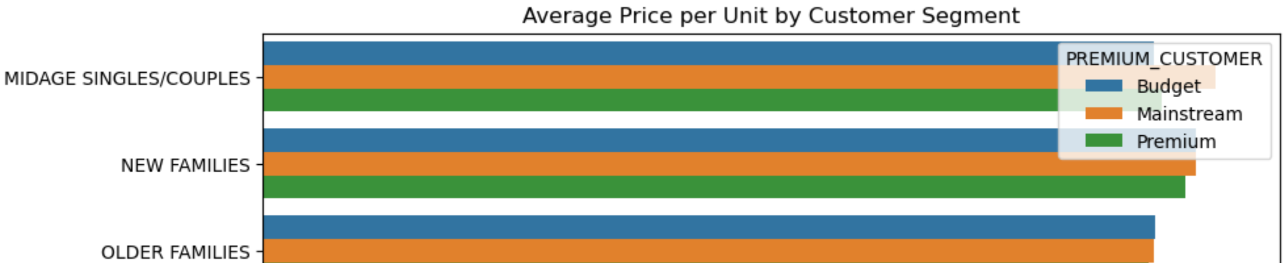
8 - Average Number of Chips Bought per Customer by Segment

```
#Calculate the average number of units(chips) bought per customer for each segment
average_units_segment = data.groupby(['LIFESTAGE','PREMIUM_CUSTOMER'])['PROD_QTY'].mean().

#Create the bar plot
plt.figure(figsize=(10,6))
sns.barplot(data=average_units_segment, x = 'PROD_QTY', y = 'LIFESTAGE', hue = 'PREMIUM_CU
plt.title('Average Number of units per customers by Customer Segment')
plt.xlabel('Average Number of units')
plt.ylabel('LIFESTAGE')

plt.show()
plt.clf()
```



9 - Average chip price by Customer Segment

```
data['PRICE_PER_UNIT'] = data['TOT_SALES'] / data['PROD_QTY']
average_price = data.groupby(['LIFESTAGE', 'PREMIUM_CUSTOMER'])['PRICE_PER_UNIT'].mean().r


#average price per unit customer
plt.figure(figsize=(10, 6))
sns.barplot(data=average_price, x='PRICE_PER_UNIT', y='LIFESTAGE', hue='PREMIUM_CUSTOMER')
plt.title('Average Price per Unit by Customer Segment')
plt.xlabel('Average Price per Unit')
plt.ylabel('LIFESTAGE')


plt.show()
```



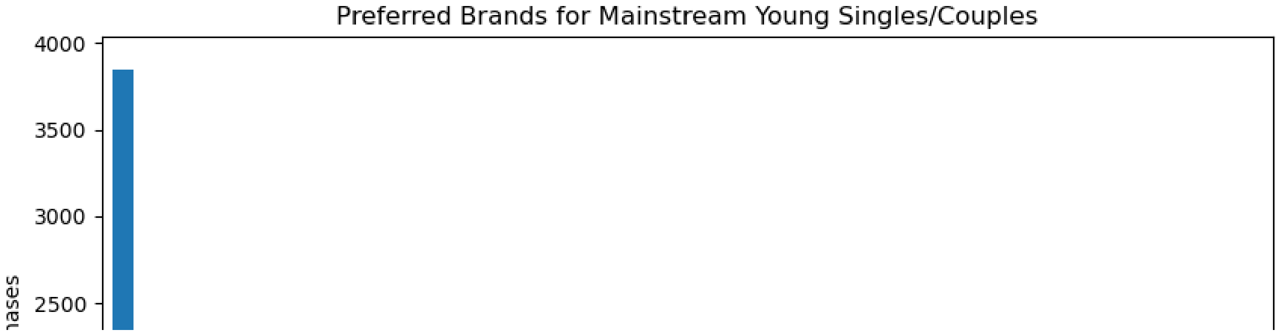
Performing an Indipendent T-test Between Mainstream vs Premimum and budget mid-age and young singles and couples

```
from scipy.stats import ttest_ind

#create mask for differnt customer segments
mainstream_mask = (data['PREMIUM_CUSTOMER'] == 'Mainstream')
budget_premium = (data['PREMIUM_CUSTOMER'] != 'Mainstream')
mid_age_young_mask = (data['LIFESTAGE'].isin(['MIDAGE SINGLES/COUPLES', 'YOUNG SINGLES/COU



#Calculate t-test for average price per unit
t_stat, p_value = ttest_ind(data[mainstream_mask & mid_age_young_mask]['PRICE_PER_UNIT'],
                            data[budget_premium & mid_age_young_mask]['PRICE_PER_UNIT'],
                            equal_var = False)


print ('this is the T statistics')
print(t_stat)
print("this is the P_value: " , p_value)
```

```
    this is the T statistics
    37.413441360485066
    this is the P_value:  2.005532452397739e-302
```

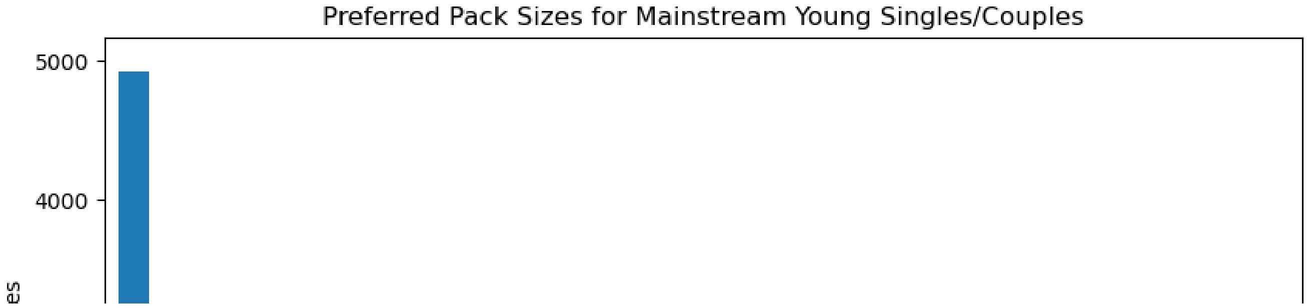
Deep Dive into the mainstream, Young singles/couples for brand and pack size analysis

```
mainstream_young = data[(data['PREMIUM_CUSTOMER'] == 'Mainstream') & (data['LIFESTAGE'] ==


# Analyze Preferred brands
brand_preferences = mainstream_young['BRAND'].value_counts()


#Ploting The prefeered brands
plt.figure(figsize=(10, 6))
brand_preferences.plot(kind='bar')
plt.title('Preferred Brands for Mainstream Young Singles/Couples')
plt.xlabel('Brand')
plt.ylabel('Number of Purchases')
plt.xticks(rotation=45)
plt.show()
```


Preferred Brands for Mainstream Young Singles/Couples

Analyze preferred Pack Sizes

```
pack_size = mainstream_young['PACK_SIZE'].value_counts()

#Plot Preferred Pack sizes
plt.figure(figsize=(10, 6))
pack_size.plot(kind='bar')
plt.title('Preferred Pack Sizes for Mainstream Young Singles/Couples')
plt.xlabel('Pack Size (g)')
plt.ylabel('Number of Purchases')
plt.xticks(rotation=0)
plt.show()
```


Preferred Pack Sizes for Mainstream Young Singles/Couples