Quantium Virtual Internship

Retail Strategy and Analytics - Task 1

(Python)

 # Importing Required Libraries.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set_theme(style="darkgrid")
plt.rcParams['figure.figsize'] = [12, 6]
```

 # Loading Data.

```
customers = pd.read_csv('/kaggle/input/quantium/QVI_purchase_behaviour.csv')
```

(purchase behviour data)

```
chips = pd.read_excel('/kaggle/input/quantium/QVI_transaction_data.xlsx')
```

(transactions data)

 # Exploring Data

```
customers.head()
customers.info()
chips.head()
chipss.info()
```

 # Date Format

```
chips['DATE'] = pd.to_datetime(chips['DATE'], origin='1899-12-30', unit='D')]
```

 # Prod Name

```
from nltk.tokenize import word_tokenize

from nltk.corpus import stopwords

import nltk

nltk.download('punkt')

nltk.download('stopwords')


chips['PROD_NAME'].value_counts().to_frame().reset_index()


# Tokenize the product names and create a list of unique words

product_names = ' '.join(chips['PROD_NAME'])

words = word_tokenize(product_names)

unique_words = pd.DataFrame(set(words), columns=['words'])


# Define keywords related to chips

chip_keywords = ['chip', 'chips']


# Check for words that are not in the chip_keywords list

non_chip_words =
unique_words[~unique_words['words'].str.lower().isin(chip_keywords +
list(stopwords.words('english')))]


# Display non-chip words

print(non_chip_words)


# Remove Salsa

chips = chips[~chips['PROD_NAME'].str.contains("Salsa", case=False, na=False)]
```

Statistical Summary

chips.describe()

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_QTY	TOT_SALES
count	264836	264836.00000	2.648360e+05	2.648360e+05	264836.000000	264836.000000	264836.000000
mean	2018-12-30 00:52:12.879215616	135.08011	1.355495e+05	1.351583e+05	56.583157	1.907309	7.304200
min	2018-07-01 00:00:00	1.00000	1.000000e+03	1.000000e+00	1.000000	1.000000	1.500000
25%	2018-09-30 00:00:00	70.00000	7.002100e+04	6.760150e+04	28.000000	2.000000	5.400000
50%	2018-12-30 00:00:00	130.00000	1.303575e+05	1.351375e+05	56.000000	2.000000	7.400000
75%	2019-03-31 00:00:00	203.00000	2.030942e+05	2.027012e+05	85.000000	2.000000	9.200000
max	2019-06-30 00:00:00	272.00000	2.373711e+06	2.415841e+06	114.000000	200.000000	650.000000
std	NaN	76.78418	8.057998e+04	7.813303e+04	32.826638	0.643654	3.083226

There are no nulls in the columns but product quantity appears to have an outlier which we should investigate further. Let's investigate further the case where 200 packets of chips are bought in one transaction.

chips[chips['PROD_QTY']== 200]

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES
69762	2018-08-19	226	226000	226201	4	Dorito Corn Chp Supreme 380g	200	650.0
69763	2019-05-20	226	226000	226210	4	Dorito Corn Chp Supreme 380g	200	650.0

chips[chips['LYLTY_CARD_NBR']== 226000]

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES
69762	2018-08-19	226	226000	226201	4	Dorito Corn Chp Supreme 380g	200	650.0
69763	2019-05-20	226	226000	226210	4	Dorito Corn Chp Supreme 380g	200	650.0

It looks like this customer has only had the two transactions over the year and is not an ordinary retail customer. The customer might be buying chips for commercial purposes instead. We'll remove this loyalty card number from further analysis

chips = chips[~(chips['LYLTY_CARD_NBR']== 226000)]

chips.describe()

	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	1.351365e+05	56.000000	2.000000	7.400000
75%	2019-03-31 00:00:00	203.000000	2.030940e+05	2.026998e+05	85.000000	2.000000	9.200000
max	2019-06-30 00:00:00	272.000000	2.373711e+06	2.415841e+06	114.000000	5.000000	29.500000
std	NaN	76.784063	8.057990e+04	7.813292e+04	32.826444	0.343436	2.527241

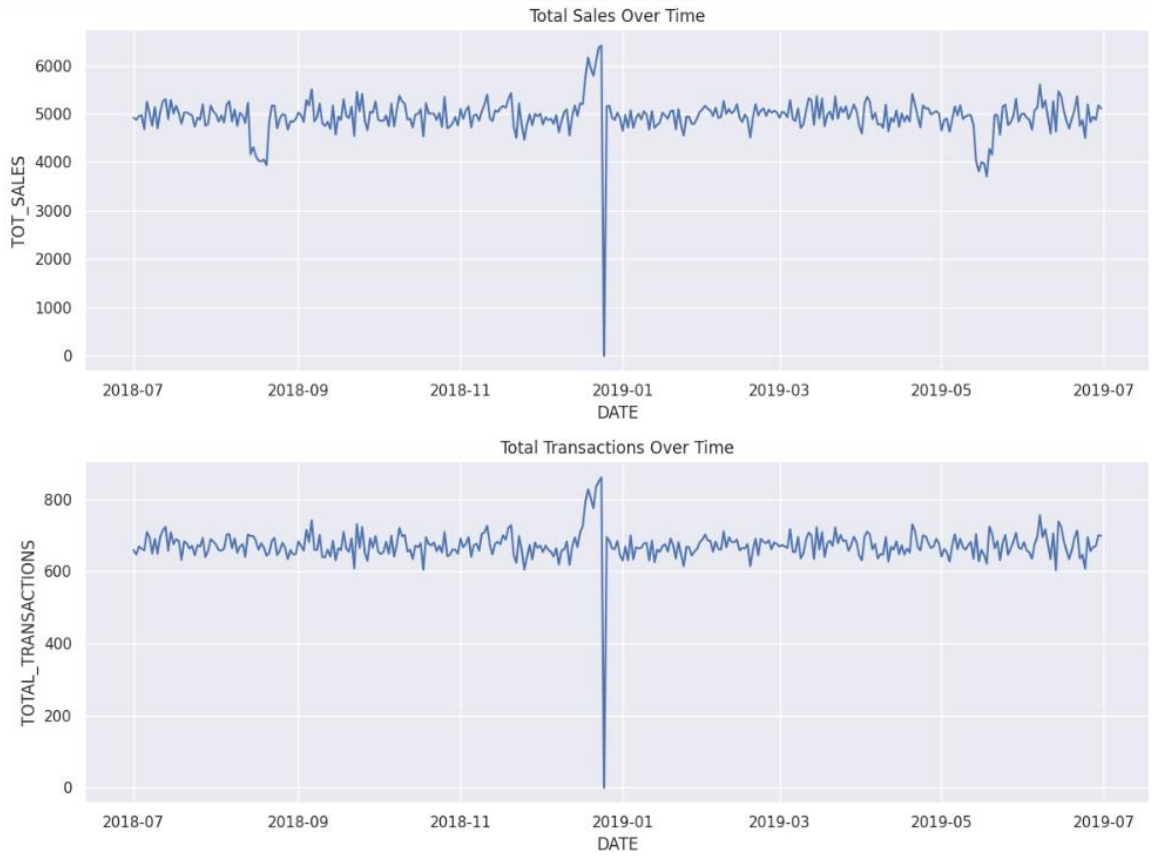
That's better. Now, let's look at the number of transaction lines over time to see if there are any obvious data issues such as missing data.

Filling Missing Day

```
# Create a date range

date_range = pd.date_range(start='2018-07-01', end='2019-06-30', freq='D')


# Create a DataFrame with the date range

all_dates = pd.DataFrame(date_range, columns=['DATE'])

chips = pd.merge(all_dates, chips, on='DATE', how='left')
```

Transactions over time

```
sales_over_time = chips.groupby('DATE')['TOT_SALES'].sum().reset_index()

transactions_over_time = chips.groupby('DATE')['TXN_ID'].nunique().reset_index()

transactions_over_time.rename(columns={'TXN_ID': 'TOTAL_TRANSACTIONS'},
inplace=True)

# ----------------

figure, axes = plt.subplots(2,1, figsize=(12,9))

sns.lineplot(data=sales_over_time, x='DATE', y='TOT_SALES',ax=axes[0])

sns.lineplot(data=transactions_over_time, x='DATE', y='TOTAL_TRANSACTIONS',ax=axes[1])

axes[0].set_title('Total Sales Over Time')

axes[1].set_title('Total Transactions Over Time')
```

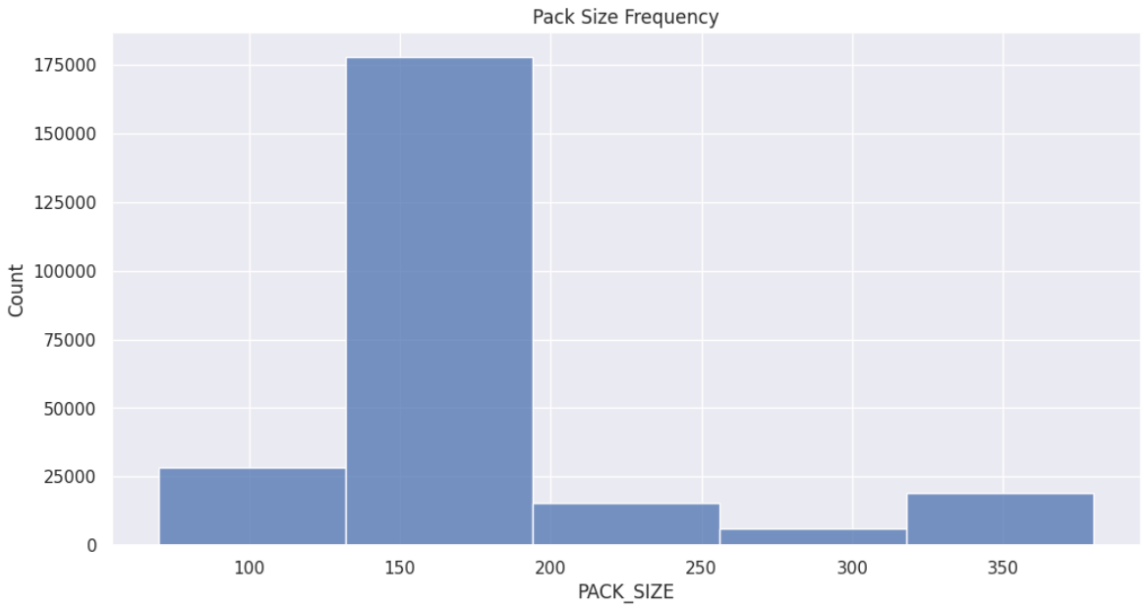


Pack Size

```
import re

# Function to parse the pack size from PROD_NAME

def parse_pack_size(prod_name):

  match = re.search(r'\d+', prod_name)

  if match:

    return int(match.group())

  return None


# Apply the function to create the PACK_SIZE column

chips['PROD_NAME'] = chips['PROD_NAME'].astype(str)

chips['PACK_SIZE'] = chips['PROD_NAME'].apply(parse_pack_size)

chips.head()
```

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	TRANSACTION_VALUE	PACK_SIZE
0	2018-07-01	47.0	47142.0	42540.0	14.0	Smiths Crnkle Chip Orgnl Big Bag 380g	2.0	11.8	23.6	380.0
1	2018-07-01	55.0	55073.0	48884.0	99.0	Pringles Sthrn FriedChicken 134g	2.0	7.4	14.8	134.0
2	2018-07-01	55.0	55073.0	48884.0	91.0	CCs Tasty Cheese 175g	2.0	4.2	8.4	175.0
3	2018-07-01	58.0	58351.0	54374.0	102.0	Kettle Mozzarella Basil & Pesto 175g	2.0	10.8	21.6	175.0
4	2018-07-01	68.0	68193.0	65598.0	44.0	Thins Chips Light& Tangy 175g	2.0	6.6	13.2	175.0

 # Pack Size Histplot



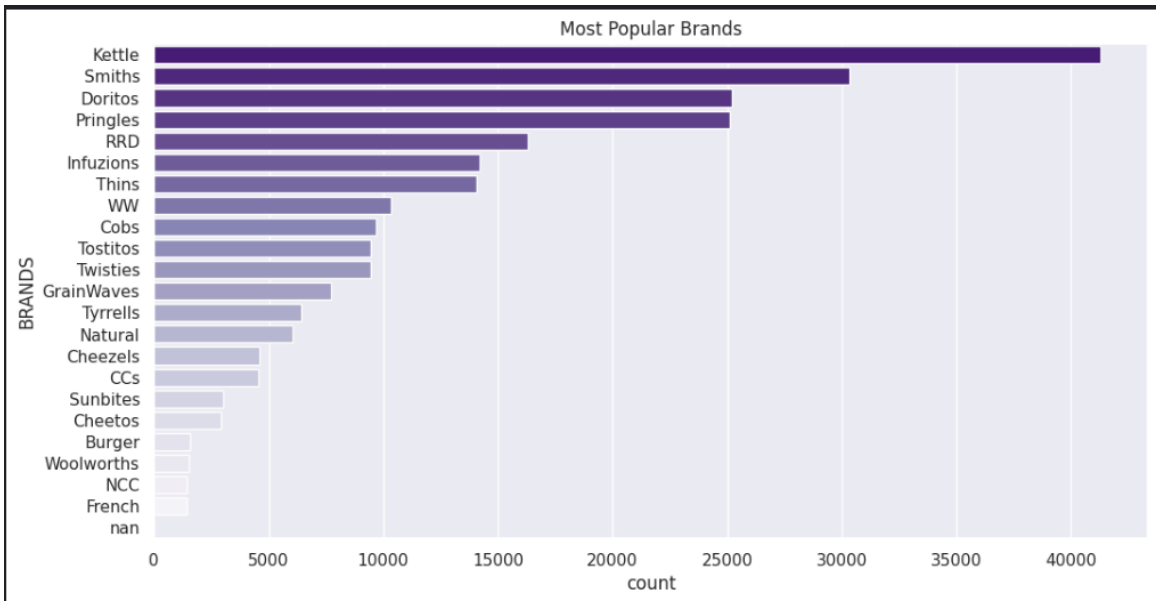
Brands

```
chips['BRANDS'] = chips['PROD_NAME'].apply(lambda x: x.split()[0])
```

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	TRANSACTION_VALUE	PACK_SIZE	BRANDS
0	2018-07-01	47.0	47142.0	42540.0	14.0	Smiths Crnkle Chip Orgnl Big Bag 380g	2.0	11.8	23.6	380.0	Smiths
1	2018-07-01	55.0	55073.0	48884.0	99.0	Pringles Sthrn FriedChicken 134g	2.0	7.4	14.8	134.0	Pringles
2	2018-07-01	55.0	55073.0	48884.0	91.0	CCs Tasty Cheese 175g	2.0	4.2	8.4	175.0	CCs
3	2018-07-01	58.0	58351.0	54374.0	102.0	Kettle Mozzarella Basil & Pesto 175g	2.0	10.8	21.6	175.0	Kettle
4	2018-07-01	68.0	68193.0	65598.0	44.0	Thins Chips Light& Tangy 175g	2.0	6.6	13.2	175.0	Thins

 # Combine similar brand names

```
chips['BRANDS'] = chips['BRANDS'].replace({'Red': 'RRD','Dorito':'Doritos',
                                           'Smith':'Smiths','Infzns':'Infuzions',
                                           'Snbts':'Sunbites',"Grain":"GrainWaves",
                                           'GrnWves':'GrainWaves'})


# Most Popular Brands
```



Customers

customers.info()

customers.rename(columns={'PREMIUM_CUSTOMER': 'MEMBERSHIP'}, inplace=True)

customers.head()

	LYLTY_CARD_NBR	LIFESTAGE	MEMBERSHIP
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

Merged Data

merged_data = pd.merge(chips.reset_index(), customers, on='LYLTY_CARD_NBR', how='left')

merged_data.head()

DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	TRANSACTION_VALUE	PACK_SIZE	BRANDS	LIFESTAGE	MEMBERSHIP
2018-07-01	47.0	47142.0	42540.0	14.0	Smiths Crnkle Chip Orgnl Big Bag 380g	2.0	11.8	23.6	380.0	Smiths	MIDAGE SINGLES/COUPLES	Budget
2018-07-01	55.0	55073.0	48884.0	99.0	Pringles Sthrn FriedChicken 134g	2.0	7.4	14.8	134.0	Pringles	MIDAGE SINGLES/COUPLES	Budget
2018-07-01	55.0	55073.0	48884.0	91.0	CCs Tasty Cheese 175g	2.0	4.2	8.4	175.0	CCs	MIDAGE SINGLES/COUPLES	Budget
2018-07-01	58.0	58351.0	54374.0	102.0	Kettle Mozzarella Basil & Pesto 175g	2.0	10.8	21.6	175.0	Kettle	MIDAGE SINGLES/COUPLES	Budget
2018-07-01	68.0	68193.0	65598.0	44.0	Thins Chips Light& Tangy 175g	2.0	6.6	13.2	175.0	Thins	MIDAGE SINGLES/COUPLES	Budget

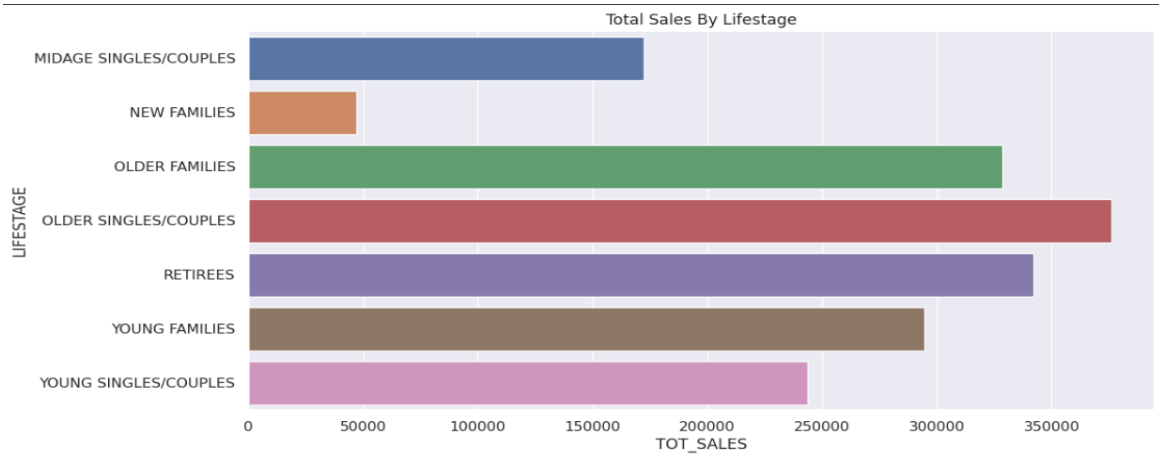
```
merged_data.isnull().value_counts()
```

Great, there are no nulls! So all our customers in the transaction data has been accounted for in the customer dataset.

Total Sales

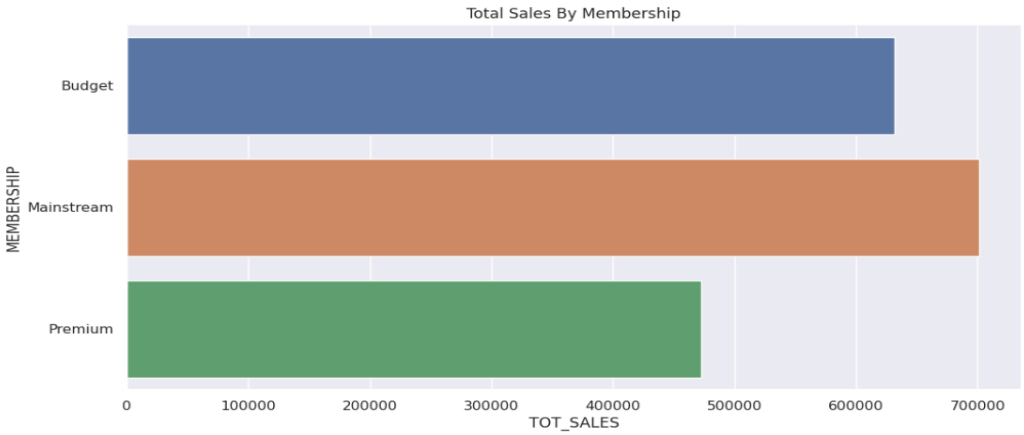
Lifestage

```
sales_by_lifestage = merged_data.groupby('LIFESTAGE')['TOT_SALES'].sum().reset_index()

sns.barplot(sales_by_lifestage,x='TOT_SALES',y='LIFESTAGE')

plt.title('Total Sales By Lifestage')
```



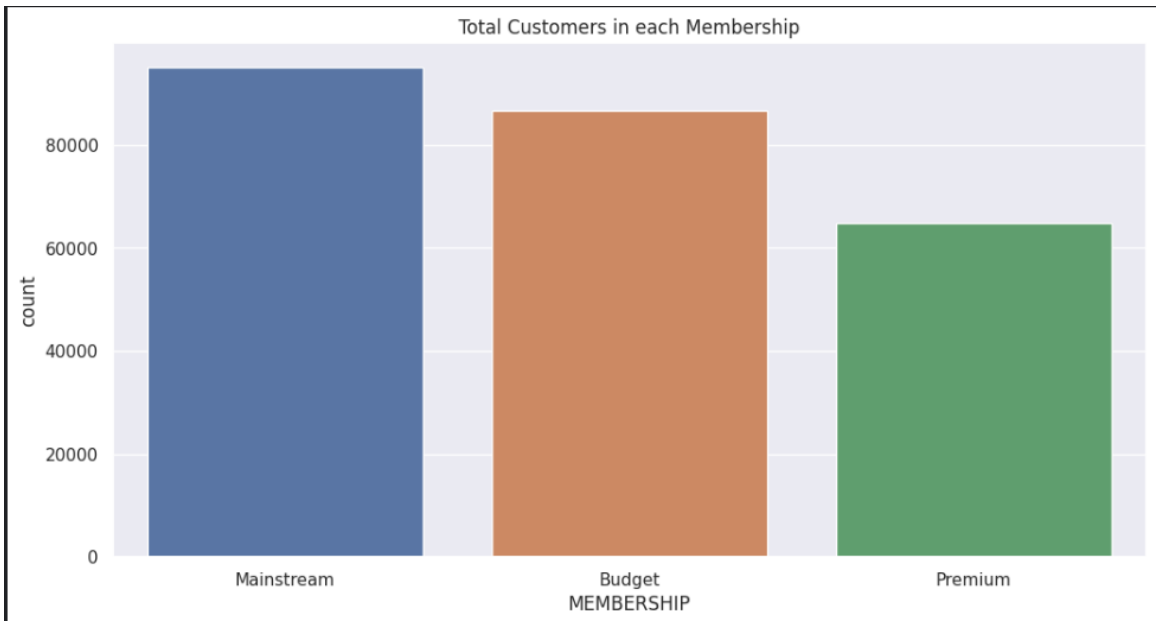
Membership

```
sales_by_premium_customer =
merged_data.groupby('MEMBERSHIP')['TOT_SALES'].sum().reset_index()

sns.barplot(sales_by_premium_customer,x='TOT_SALES',y='MEMBERSHIP')

plt.title('Total Sales By Membership')
```



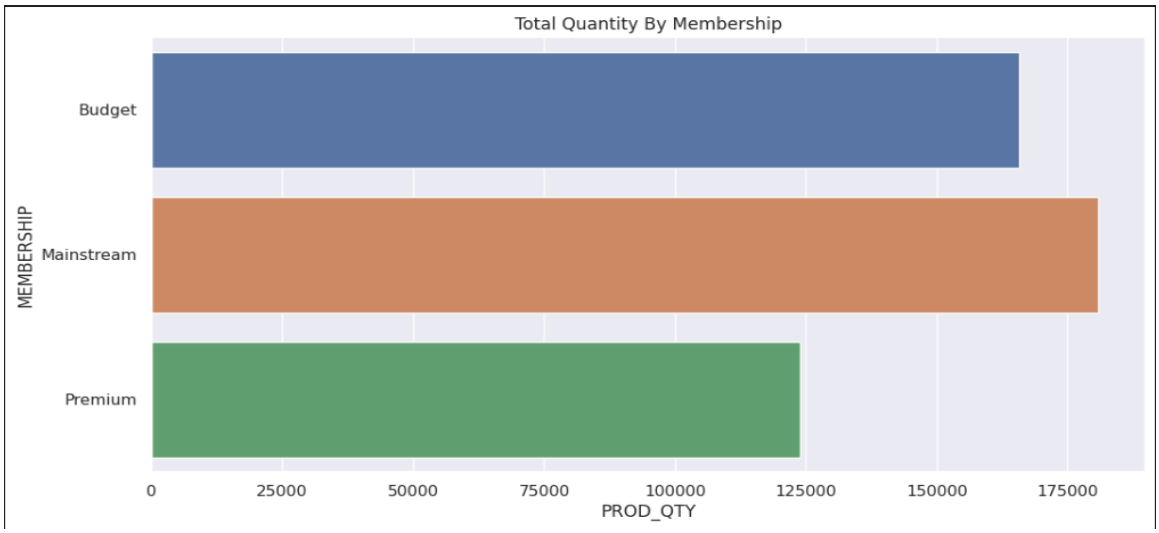
 # Total Customers in each Membership

```
total_customers_membership =
merged_data['MEMBERSHIP'].value_counts().to_frame().reset_index()

sns.barplot(data=total_customers_membership,x='MEMBERSHIP',y='count')
```



 # Total Qty by membership

```
qty_by_membership =
merged_data.groupby('MEMBERSHIP')['PROD_QTY'].sum().reset_index()

sns.barplot(qty_by_membership,x='PROD_QTY',y='MEMBERSHIP')

plt.title('Total Quantity By Membership')
```

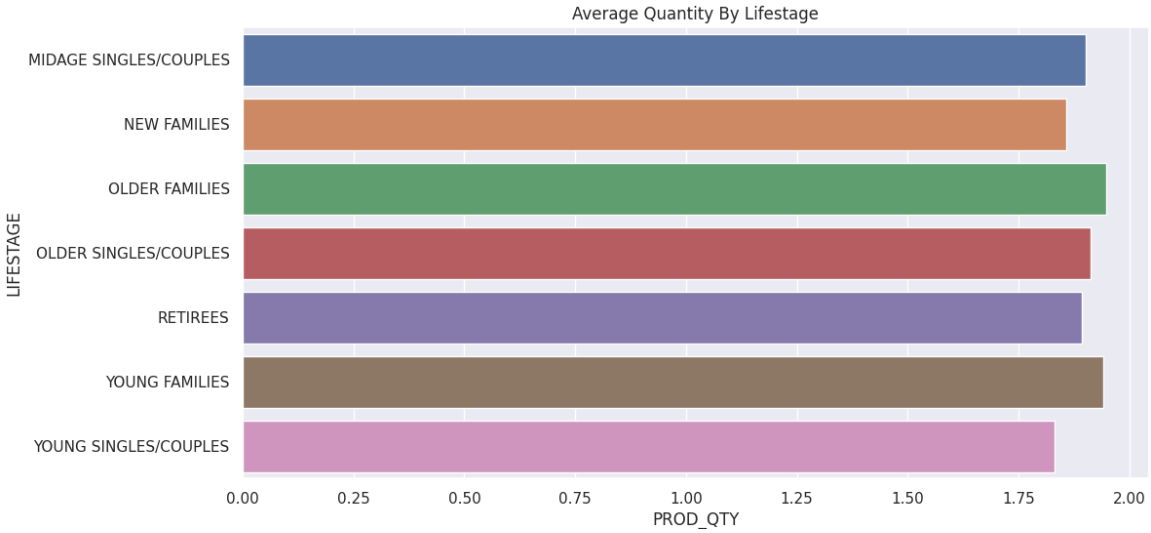


 # Average Price by Membership

```
ax = sns.scatterplot(data=avg_price_by_membership, x='MEMBERSHIP', y='TOT_SALES',
s=200,hue='MEMBERSHIP')


for i, row in avg_price_by_membership.iterrows():
  ax.annotate(f'{row["TOT_SALES"]:.2f}', (row['MEMBERSHIP'], row['TOT_SALES']),
      textcoords="offset points", xytext=(0,10), ha='center')


plt.title('Average Price by Membership')
plt.xlabel('Membership')
plt.ylabel('Average Price')
plt.show()
```

 # Average Qty by Lifestage

```
avg_qty_by_lifestage =
merged_data.groupby('LIFESTAGE')['PROD_QTY'].mean().reset_index()
sns.barplot(avg_qty_by_lifestage,x='PROD_QTY',y='LIFESTAGE')
plt.title('Average Quantity By Lifestage')
```

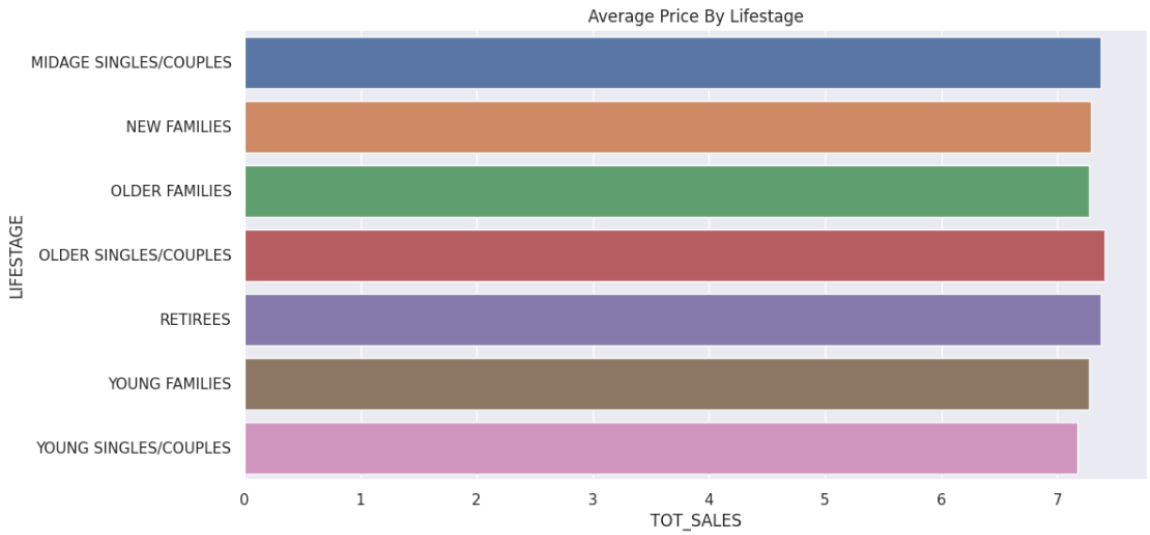


 # Average Qty by Membership

```
avg_qty_by_membership =
merged_data.groupby('MEMBERSHIP')['PROD_QTY'].mean().reset_index()


ax = sns.scatterplot(data=avg_qty_by_membership, x='MEMBERSHIP', y='PROD_QTY',
s=200,hue='MEMBERSHIP')


for i, row in avg_qty_by_membership.iterrows():
   ax.annotate(f'{row["PROD_QTY"]:.3f}', (row['MEMBERSHIP'], row['PROD_QTY']),
         textcoords="offset points", xytext=(0,10), ha='center')


plt.title('Average Qty by Membership')
plt.xlabel('Membership')
plt.ylabel('Average QTY')
```

 # Average Price By Lifestage

```
avg_sales_by_lifestage =
merged_data.groupby('LIFESTAGE')['TOT_SALES'].mean().reset_index()

sns.barplot(avg_sales_by_lifestage,x='TOT_SALES',y='LIFESTAGE')

plt.title('Average Price By Lifestage')
```



 # T-Test

```
from scipy import stats
```

 # independent t-test between mainstream vs premium and budget

```
mainstream = merged_data[merged_data['MEMBERSHIP'] == 'Mainstream']['TOT_SALES']

premium = merged_data[merged_data['MEMBERSHIP'] == 'Premium']['TOT_SALES']

budget = merged_data[merged_data['MEMBERSHIP'] == 'Budget']['TOT_SALES']


t_stat_mainstream_vs_premium, p_value_mainstream_vs_premium =
stats.ttest_ind(mainstream, premium)

t_stat_mainstream_vs_budget, p_value_mainstream_vs_budget =
stats.ttest_ind(mainstream, budget)
```

 # results

T-test Mainstream vs Premium: T-statistic = 7.27, P-value = 0.0000

T-test Mainstream vs Budget: T-statistic = 8.34, P-value = 0.0000

 # midage and young singles and couples

```
midage_singles_couples = merged_data[merged_data['LIFESTAGE'] == 'MIDAGE
SINGLES/COUPLES']['TOT_SALES']

young_singles_couples = merged_data[merged_data['LIFESTAGE'] == 'YOUNG
SINGLES/COUPLES']['TOT_SALES']

t_stat, p_value = stats.ttest_ind(midage_singles_couples, young_singles_couples)
```

 # results

T-test Midage Singles/Couples vs Young Singles/Couples: T-statistic = 9.16, P-value = 0.0000

p-value of 0.00 indicates a very strong statistical difference