Name	Gisanrin Oluwaseun
Email	seungisanrin@gmail.com
GitHub	github.com/seungisanrin

```
# Importing the libraries for use
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
## **Cleaning QVI_Sales_data.xlsx**
# Loading the transaction data Excel file into a dataframe


txn_data = pd.read_excel('QVI_transaction_data.xlsx')
# Looking at the shape of the data to be worked with
txn_data.shape
# Getting a glimpse of the datafram
txn_data.head()
# Checking the data types for all columns ensuring they are properly formatted
txn_data.dtypes
The `DATE` column is wrongly formatted, hence should be converted to a datetime object
# Making a copy of the dataframe for cleaning
clean_txn_data = txn_data.copy()
# Parsing the Excel date values to date string
clean_txn_data['DATE'] = pd.to_datetime(clean_txn_data['DATE'], origin='1899-12-30', unit='D')
clean_txn_data.info()
clean_txn_data.dtypes
# Setting all columns to lowercase letter variables
clean_txn_data.columns = clean_txn_data.columns.str.lower()
clean_txn_data.head()
# Checking for duplicates in the dataframe
clean_txn_data.duplicated().sum()
clean_txn_data['date'].sort_values().unique()
```

```
clean_txn_data['date']

for x in pd.date_range('2018-07-01','2019-06-30'):

  if x not in (clean_txn_data['date'].sort_values().unique()):

    print(x)
```

The dataset contains the chips' purchase transcations for one year, however the only date missing from the dataset is Christmas Day. I assume shops were closed for the celebration, further investigation should be carried out to ensure the validity of the hypothesis.

```
# Filtering out duplicate records from the dataframe

clean_txn_data = clean_txn_data[~clean_txn_data.duplicated()]

# Extracting the pack size for each product

clean_txn_data['pack_size'] = clean_txn_data['prod_name'].str[-4:]

clean_txn_data.head()

# Getting an overview of values in the pack_size column


clean_txn_data['pack_size'].unique()

# Cleaning the values in pack size column to solely lower case

clean_txn_data['pack_size'] = clean_txn_data['pack_size'].str.lower()


# Cleaning the pack size column

clean_txn_data.loc[clean_txn_data['pack_size'] == 'salt', 'pack_size'] = '135g'


# Converting pack size to integer values

clean_txn_data['pack_size'] = clean_txn_data['pack_size'].str[-4:-1].astype(int)

clean_txn_data['pack_size'].unique()

# Creating a brand_name column for further analysis

clean_txn_data['brand_name'] = clean_txn_data['prod_name'].str.split(' ').str[0]

clean_txn_data['brand_name'].unique()
```

There are some repititions in the form of abbreviations here, hence they need to be corrected before proceeding with the analysis

```
# Cleaning brand name repititions in the form of abbreviations
clean_txn_data.loc[clean_txn_data['brand_name'] == 'Infzns', 'brand_name'] = 'Infuzions'
clean_txn_data.loc[clean_txn_data['brand_name'] == 'Smith', 'brand_name'] = 'Smiths'
clean_txn_data.loc[clean_txn_data['brand_name'] == 'GrnWves', 'brand_name'] = 'Grain'
clean_txn_data.loc[clean_txn_data['brand_name'] == 'Dorito', 'brand_name'] = 'Doritos'
clean_txn_data.loc[clean_txn_data['brand_name'] == 'Snbts', 'brand_name'] = 'Sunbites'
clean_txn_data.loc[clean_txn_data['brand_name'] == 'WW', 'brand_name'] = 'Woolworths'
clean_txn_data.loc[clean_txn_data['brand_name'] == 'Red', 'brand_name'] = 'RRD'
clean_txn_data.loc[clean_txn_data['brand_name'] == 'Natural', 'brand_name'] = 'NCC'
clean_txn_data['brand_name'].unique()
clean_txn_data.head()
clean_txn_data.describe()
# Checking for outliers in the prod_qty column
plt.figure(figsize=(10,4))
plt.plot(clean_txn_data['prod_qty'])
plt.xlabel('Transaction ID')
plt.ylabel('Number of Chips')
plt.show()
# Removing the outlier from the data
clean_txn_data = clean_txn_data.loc[clean_txn_data['prod_qty'] != 200]


# Removing rows containing blanks
clean_txn_data = clean_txn_data.dropna(axis=0)
products = pd.DataFrame(clean_txn_data['prod_name'].unique())
# Checking for non-chips products
```

```
products.loc[~products[0].str.lower().str.contains(pat='chip',case=False)]

clean_txn_data =
clean_txn_data.loc[~clean_txn_data['prod_name'].str.lower().str.contains(pat='salsa',
case=False)]

clean_txn_data.shape

clean_txn_data.describe()

clean_txn_data['unit_price'] = clean_txn_data['tot_sales']/clean_txn_data['prod_qty']

clean_txn_data.head()

## **Cleaning QVI_purchase_behaviour.csv**

# Loading the dataset into a dataframe

purchase_behaviour = pd.read_csv('QVI_purchase_behaviour.csv')

cleaned_purchase = purchase_behaviour.copy()

cleaned_purchase.head()

# Setting all columns to lowercase

cleaned_purchase.columns = cleaned_purchase.columns.str.lower()

# Checking for duplicates in the Purchase Behaviour datafram

cleaned_purchase.duplicated().sum()

cleaned_purchase.head()

cleaned_purchase.shape

# Dropping blank rows if any

cleaned_purchase = cleaned_purchase.dropna(axis=0)

# Merging the transaction data and customer purchase information

df = pd.merge(clean_txn_data, cleaned_purchase, how='inner', on='lylty_card_nbr')

df.dtypes

# Visualisation and Analysis

plt.figure(figsize=(10,6))


sns.histplot(df,

        x='pack_size',
        binwidth=10)

plt.xticks(ticks=range(0,400,25))
plt.xlabel('Chips Pack Size (g)')
plt.ylabel('Number of Chips Transactions')
plt.tight_layout()
plt.show()
```

Majority of the chips purchased are of the sizes 175 g, followed by 150 g

```
# Aggregating purchase values by lifestage, and customer type (premium_customer)
sale_bvx = df.groupby(['lifestage','premium_customer'])['tot_sales'].aggregate(['sum','mean']).reset_index()
sale_bvx
plt.figure(figsize=(8,6))

sns.barplot(sale_bvx.groupby('lifestage')['sum'].sum().reset_index(),
      x='lifestage',
      y='sum',

order=sale_bvx.groupby('lifestage')['sum'].sum().reset_index().sort_values('sum',ascending=False).lifestage)

plt.xlabel('Lifestage')
plt.ylabel('Total Sales')
plt.xticks(rotation=90)
plt.title('Total Sales per Lifestage')
plt.tight_layout()
plt.show()
plt.figure(figsize=(8,6))
```

```
sns.barplot(sale_bvx.groupby('premium_customer')['sum'].sum().reset_index(),
        x='premium_customer',
        y='sum',

order=sale_bvx.groupby('premium_customer')['sum'].sum().reset_index().sort_values('sum',ascending=False).premium_customer)


plt.xlabel('Customer Type')
plt.ylabel('Total Sales')
plt.xticks(rotation=90)
plt.title('Total Sales per Customer Type')
plt.tight_layout()
plt.show()
plt.figure()


sns.barplot(cleaned_purchase['lifestage'].value_counts().reset_index(),
        x='lifestage',
        y='count')


plt.xticks(rotation=90)
plt.xlabel('Lifestage')
plt.ylabel('Number of Transcations')
plt.title('Number of Transactions per Lifestage')
plt.tight_layout()
plt.show()
plt.figure()


sns.barplot(cleaned_purchase['premium_customer'].value_counts().reset_index(),
    x='premium_customer',
    y='count')
plt.xlabel('Customer Type')
plt.ylabel('Number of Transcations')
plt.title('Number of Transactions per Customer Type')
plt.tight_layout()
plt.show()
plt.figure()

sns.barplot(df.groupby('lifestage')['prod_qty'].sum().reset_index().sort_values('prod_qty',ascending=False),
    x='lifestage',
    y='prod_qty')

plt.xticks(rotation=90)
plt.xlabel('Lifestage')
plt.ylabel('Number of Chips Purchased')
plt.title('Number of Chips Purchased per Lifestage')
plt.tight_layout()
plt.show()
plt.figure()

sns.barplot(df.groupby('premium_customer')['prod_qty'].sum().reset_index().sort_values('prod_qty',ascending=False),
    x='premium_customer',
    y='prod_qty')
plt.xlabel('Customer Type')
plt.ylabel('Number of Transcations')
plt.title('Number of Transactions per Customer Type')
plt.tight_layout()
plt.show()
plt.figure(figsize=(6,8))

sns.barplot(sale_bvx.groupby('lifestage')['mean'].mean().reset_index(),
    x='lifestage',
    y='mean',

order=sale_bvx.groupby('lifestage')['mean'].mean().reset_index().sort_values('mean',ascending=False).lifestage)

plt.xlabel('Lifestage')
plt.ylabel('Average Sale')
plt.title('Average Sale Revenue per Lifestage')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
plt.figure(figsize=(8,6))

sns.barplot(sale_bvx.groupby('premium_customer')['mean'].mean().reset_index(),
    x='premium_customer',
    y='mean',

order=sale_bvx.groupby('premium_customer')['mean'].mean().reset_index().sort_values('mean',ascending=False).premium_customer)

plt.xlabel('Customer Type')
plt.ylabel('Average Sale Revenue')
plt.title('Average Sale Revenue per Customer Type')
plt.tight_layout()

plt.show()

sale_bvx.pivot_table(values='sum',columns='premium_customer',index='lifestage').plot(kind='bar')

plt.xlabel('Lifestage')

plt.ylabel('Total Sale Revenue')

plt.title('Total Sale Revenue per Lifestage: Customer Type Analysis')

plt.legend(title = 'Customer Type', loc='center left', bbox_to_anchor=(1, 0.5), ncol=1)

sale_bvx.pivot_table(values='mean',columns='premium_customer',index='lifestage').plot(kind='bar')

plt.xlabel('Lifestage')

plt.ylabel('Average Sale Revenue')

plt.title('Average Sale Revenue per Lifestage: Customer Type Analysis')

plt.legend(title = 'Customer Type', loc='center left', bbox_to_anchor=(1, 0.5), ncol=1)

df.pivot_table(values='prod_qty',columns='premium_customer',index='lifestage')

prod_quant = df.groupby(['lifestage','premium_customer'])['prod_qty'].mean().reset_index().pivot_table(values='prod_qty',index='lifestage',columns='premium_customer')

prod_quant.plot(kind='bar')

plt.xlabel('Lifestage')

plt.ylabel('Average Number of Chips Purchased')

plt.title('Average Number of Chips Purchased per Lifestage: Customer Type Analysis')

plt.legend(title = 'Customer Type', loc='center left', bbox_to_anchor=(1, 0.5), ncol=1)

cat_count = purchase_behaviour.groupby(['LIFESTAGE','PREMIUM_CUSTOMER'])['LYLTY_CARD_NBR'].size().reset_index(name='COUNT').pivot_table(index='LIFESTAGE',columns='PREMIUM_CUSTOMER',values='COUNT')

cat_count

cat_count.plot(kind='bar')
```

```
plt.xlabel('Lifestage')

plt.ylabel('Number of Customers')

plt.title('Number of Customers per Lifestage: Customer Type Analysis')

plt.legend(title = 'Customer Type', loc='center left', bbox_to_anchor=(1, 0.5), ncol=1)

## t-test Analysis

main_mid_yg = df.loc[(df['premium_customer'] == 'Mainstream') & (df['lifestage'].isin(['MIDAGE SINGLES/COUPLES','YOUNG SINGLES/COUPLES']))]['unit_price']

nonmain_mid_yg = df.loc[~(df['premium_customer'] == 'Mainstream') & (df['lifestage'].isin(['MIDAGE SINGLES/COUPLES','YOUNG SINGLES/COUPLES']))]['unit_price']

from scipy.stats import ttest_ind

t_stat, p_val = ttest_ind(main_mid_yg, nonmain_mid_yg, alternative='greater')

from mlxtend.frequent_patterns import apriori, association_rules

# Making a copy fo the original dataframe for association analysis

assoc_df = df.copy()

assoc_df['group'] = assoc_df['lifestage'] + ' - ' + assoc_df['premium_customer']

group = pd.get_dummies(assoc_df['group'])


brand = pd.get_dummies(assoc_df['brand_name'])


group_brands = group.join(brand)

freq_groupsbands = apriori(group_brands, min_support=0.008, use_colnames=True)

rules = association_rules(freq_groupsbands, metric='lift', min_threshold=0.5)

rules.sort_values('confidence', ascending=False, inplace=True)

rules.head()

set_temp_association = assoc_df['group'].unique()

rules[rules['antecedents'].apply(lambda x: list(x)).apply(lambda x: x in set_temp_association)]
```

```
mask = (df['lifestage'] == 'YOUNG SINGLES/COUPLES') & (df['premium_customer'] == 'Mainstream')

young_main = df.loc[mask]

target_segment = young_main['brand_name'].value_counts(ascending=True).rename_axis('BRANDS').reset_index(name='TARGET')

target_segment['TARGET'] = target_segment['TARGET']/young_main.shape[0]

not_target_segment = df.loc[df['lifestage'] != "YOUNG SINGLES/COUPLES"]

not_target_segment = not_target_segment.loc[not_target_segment['premium_customer'] != "Mainstream"]

other = not_target_segment["brand_name"].value_counts().sort_values(ascending = True).rename_axis('BRANDS').reset_index(name='NON_TARGET')

other["NON_TARGET"] = other["NON_TARGET"] / not_target_segment.shape[0]

brand_proportions = target_segment.set_index('BRANDS').join(other.set_index('BRANDS'))

brand_proportions = brand_proportions.reset_index()

brand_proportions['AFFINITY'] = brand_proportions['TARGET']/brand_proportions['NON_TARGET']

brand_proportions.sort_values('AFFINITY', ascending = False)

group_gp = pd.get_dummies(assoc_df['group'])

brand_gp = pd.get_dummies(assoc_df['pack_size'])

group_brands_gp = group_gp.join(brand_gp)

group_brands_gp

freq_groupsbrands_gp = apriori(group_brands_gp, min_support=0.009, use_colnames=True)

rules_gp = association_rules(freq_groupsbrands_gp, metric="lift", min_threshold=0.5)

rules_gp.sort_values('confidence', ascending = False, inplace = True)

set_temp = assoc_df["group"].unique()

rules_gp[rules_gp["antecedents"].apply(lambda x: list(x)).apply(lambda x: x in set_temp)]
```