|  |  |
| --- | --- |
| **Name** | Gisanrin Oluwaseun |
| **Email** | [seungisanrin@gmail.com](mailto: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)]