Quantium_Data_Analytics

March 16, 2023

1 Regional Analysis of Chips Sales and Customer Behavior

Analyst: Albert Dellor

Table of Contents

2 Introduction

This project aims at analyzing the purchasing behavior of customers who buy chips, which includes identifying the frequency, quantity, and types of chips they purchase in an effort to inform and drive strategy for supermarket's chips division for the next half year.

2.1 Data description

3 Data Wrangling

Import all packages and set plots to be embedded inline

[388]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas.plotting import table
import seaborn as sb
import requests
from IPython.display import Image
%matplotlib inline
```

Loading the dataset: Get the URL of dataset. Create request, download transaction and purchase behaviour datasets

[10]:
```
urls = ['https://cdn.theforage.com/vinternships/companyassets/32A6DqtsbF7LbKdcq/
 ↪QVI_transaction_data.xlsx',
```

```
        'https://cdn.theforage.com/vinternships/companyassets/32A6DqtsbF7LbKdcq/
↪QVI_purchase_behaviour.csv']

for url in urls:
    data = requests.get(url)
    with open(url.split('/')[-1], mode = 'wb') as file:
        file.write(data.content)
```

Loading Datasets

[389]:
```
transactions = pd.read_excel('QVI_transaction_data.xlsx')
purchase_behavor = pd.read_csv('QVI_purchase_behaviour.csv')
```

3.1 Data Assessment

Visual assessment and programmatic assessments were conducted to detect quality and tidiness issues with both dataset

- Visual assessment of data in Microsoft excell
- Programmatic assessment in python

3.1.1 Assessment: QVI_purchase_behaviour.csv

[390]:
```
purchase_behavor.sample(5)
```

[390]:

	LYLTY_CARD_NBR	LIFESTAGE	PREMIUM_CUSTOMER
69683	261125	YOUNG SINGLES/COUPLES	Mainstream
18769	68279	YOUNG FAMILIES	Premium
21972	80057	OLDER SINGLES/COUPLES	Mainstream
19568	71240	YOUNG FAMILIES	Budget
57062	215319	MIDAGE SINGLES/COUPLES	Premium

Dimensions of data set

[391]:
```
print('Rows: {}\nColumns: {}'.format(purchase_behavor.shape[0]\
                                      ,purchase_behavor.shape[1]))
```

```
Rows: 72637
Columns: 3
```

Missing values and data types

[392]:
```
purchase_behavor.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
```

Number of unique values in PREMIUM_CUSTOMER and LIFESTAGE

[393]:
```
purchase_behavor.PREMIUM_CUSTOMER.nunique(),\
purchase_behavor.LIFESTAGE.nunique()
```

[393]:
```
(3, 7)
```

3.1.2 NB:

The data consists of **72637 rows** by **3 columns**. There are **no missing vaules**. However, **PREMIUM_CUSTOMER** and **LIFESTAGE** columns have few number of unique values, thus can be converted from string object to categorical date type as they hold no ordinal value.

3.1.3 Assessment: QVI_transaction_data.xlsx

[394]:
```
transactions.sample(5)
```

[394]:
```
         DATE  STORE_NBR  LYLTY_CARD_NBR  TXN_ID  PROD_NBR  \
215323  43334        232          232199  236314        92
22669   43366        184          184135  187107       102
187888  43606         40           40166   36876        17
174857  43310         59           59027   54713        63
213333  43464        191          191058  192108        64

                                       PROD_NAME  PROD_QTY  TOT_SALES
215323          WW Crinkle Cut      Chicken 175g         2        3.4
22669     Kettle Mozzarella    Basil & Pesto 175g         2       10.8
187888         Kettle Sensations   BBQ&Maple 150g         2        9.2
174857              Kettle 135g Swt Pot Sea Salt         2        8.4
213333  Red Rock Deli SR     Salsa & Mzzrlla 150g         2        5.4
```

Dimensions of data set

[395]:
```
print('Rows: {}\nColumns: {}'.format(transactions.shape[0],\
                                     transactions.shape[1]))
```

```
Rows: 264836
Columns: 8
```

Missing values and data types

[396]:
```
transactions.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
```

Descriptive statistics

[397]:
```
transactions.describe()[['PROD_QTY','TOT_SALES']]
```

[397]:

	PROD_QTY	TOT_SALES
count	264836.000000	264836.000000
mean	1.907309	7.304200
std	0.643654	3.083226
min	1.000000	1.500000
25%	2.000000	5.400000
50%	2.000000	7.400000
75%	2.000000	9.200000
max	200.000000	650.000000

check for duplicates. The only unique identifier in dataset is TXN_ID (transaction id). Thus is used to find duplicates

[398]:
```
transactions[transactions.duplicated()]
```

[398]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	\
124845	43374	107	107024	108462	45	

	PROD_NAME	PROD_QTY	TOT_SALES
124845	Smiths Thinly Cut Roast Chicken 175g	2	6.0

Further investigate the transaction ID

[399]:
```
transactions.query("TXN_ID ==␣
↪108462")[['TXN_ID','PROD_NAME','PROD_QTY','TOT_SALES']]
```

[399]:

	TXN_ID	PROD_NAME	PROD_QTY	TOT_SALES
124843	108462	Smiths Thinly Cut Roast Chicken 175g	2	6.0
124844	108462	Cheetos Chs & Bacon Balls 190g	2	6.6

```
124845  108462  Smiths Thinly Cut   Roast Chicken 175g         2         6.0
```

3.1.4 NB

There are **no missing data**. **PROD_QTY and TOT_SALES have 75% of entire data being under 2 and 9.2 respectively**, however the maximun values are 200 and 650 respectively, there might be outliers present. Further investigation reveal that there was double recording of a transaction involving **Smiths Thinly Cut Roast Chicken 175g**

3.1.5 Summary of assessment

Transaction data (unclean)

1. Tidiness issues

- product name contains product mass variable as well, thus must be separated into respective columns as each variable must form a column to comply with data tidiness standards.
- presence of duplicate transaction involving Smiths Thinly Cut Roast Chicken 175g. All entries must be unique. thus duplicates must be dropped.

2. Quality issues

- PROD_QTY and TOT_SALES may have outliers.
- product name has inconsistent spacing thus must be formated to comply with data quality standards.
- date format is in Excel serial number instead of pandas datetime object.
- convert `PREMIUM_CUSTOMER` and `LIFESTAGE` columns to categorical data type.

Purchase behavior data(clean)

3.2 Data Cleaning

In this sectionall the data issues outlined in assesment stage are cleaned

[400]:
```
# make copies of both data sets
purchase_behavor_copy = purchase_behavor.copy()
transactions_copy = transactions.copy()
```

Issue #1 & 4: separate product mass and product name into deparate columns, format spacing

[401]:
```
regex = r'(\d+)(g|G)'
transactions_copy['PROD_WEIGHT'] = pd.to_numeric(
                                    transactions_copy.PROD_NAME\
                                    .str.extract(regex)[0])
transactions_copy['PROD_NAME'] = transactions_copy.PROD_NAME\
                                    .str.replace(regex, '')\
                                    .str.replace(r'\s\s+', ' ')\
                                    .str.replace('&', ' & ')
```

```
/var/folders/jw/bf_46b5j2jdc8rkkcynkcqdr0000gn/T/ipykernel_3852/3124044391.py:5:
FutureWarning: The default value of regex will change from True to False in a
future version.
  transactions_copy['PROD_NAME'] = transactions_copy.PROD_NAME\
```

Issue #2: Remove duplicates in data set

[402]:
```
transactions.drop_duplicates(inplace = True)
```

Issue #5: Format date from Excel serial number to pandas datetime object

[403]:
```
transactions_copy.DATE = pd.to_datetime(transactions_copy.DATE, unit='d',␣
 ↪origin=pd.Timestamp('1900-01-01'))
```

Issue #3: PROD_QTY and TOT_SALES may have outliers

[404]:
```
transactions_copy.nlargest(6,'TOT_SALES')[['PROD_NAME','PROD_QTY','TOT_SALES']]
```

[404]:

	PROD_NAME	PROD_QTY	TOT_SALES
69762	Dorito Corn Chp Supreme	200	650.0
69763	Dorito Corn Chp Supreme	200	650.0
5179	Smiths Crnkle Chip Orgnl Big Bag	5	29.5
55558	Smiths Crnkle Chip Orgnl Big Bag	5	29.5
69496	Smiths Crnkle Chip Orgnl Big Bag	5	29.5
117850	Smiths Crnkle Chip Orgnl Big Bag	5	29.5

looking at the 6 largest TOT_SALES, its evident that 650 are outliers. To further comfirm this, it is general accepted that any data point that falls outside the range of Q1 - 1.5 x IQR to Q3 + 1.5 x IQR is considered a potential outlier

[405]:
```
stats = transactions_copy.describe()
```

[406]:
```
def outlier(column):
    Q1 = stats.loc['25%'][column]
    Q3 = stats.loc['75%'][column]
    IQR = Q3 - Q1
    low_range = Q1 - (1.5 * IQR)
    upper_range = Q3 + (1.5 * IQR)
    print("For {} an outlier is any value outside the range of {:.2f} and {:.
 ↪2f}".format(column,low_range,upper_range))
```

[407]:
```
outlier('PROD_QTY')
outlier('TOT_SALES')
```

```
For PROD_QTY an outlier is any value outside the range of 2.00 and 2.00
For TOT_SALES an outlier is any value outside the range of -0.30 and 14.90
```

It is evident that a customer with loyalty card number 226000, purchased large orders on two occasions in 2018 and 2019, thus creating the outliers for PROD_QTY and corresponding TOT_SALES

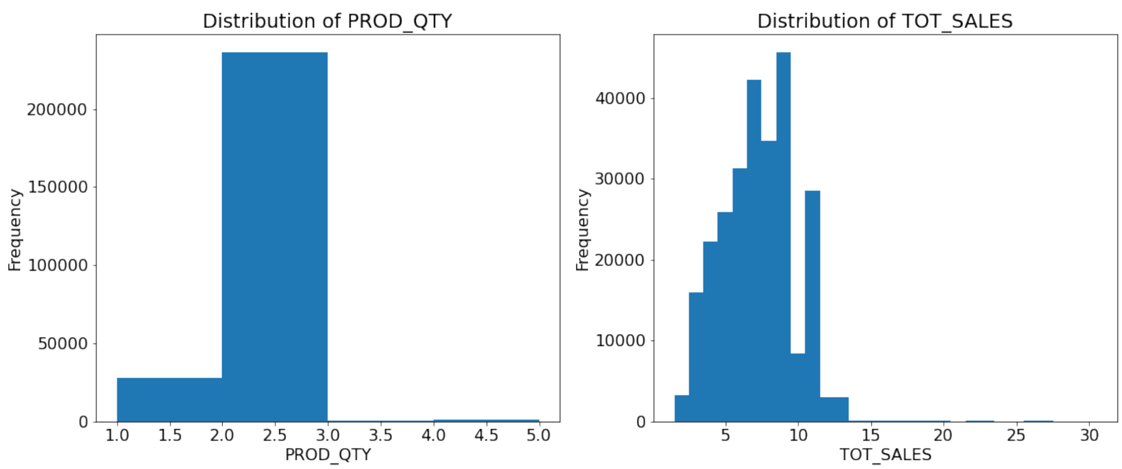
[408]:
```python
# Looking at the descriptive statistics of both data sets
pd.merge(pd.DataFrame(transactions_copy.PROD_QTY.describe()),\
         pd.DataFrame(transactions_copy.TOT_SALES.describe()),\
         on=pd.DataFrame(transactions_copy.TOT_SALES.describe()).index).
 ↪rename(columns={'key_0':'index'})
```

[408]:

	index	PROD_QTY	TOT_SALES
0	count	264836.000000	264836.000000
1	mean	1.907309	7.304200
2	std	0.643654	3.083226
3	min	1.000000	1.500000
4	25%	2.000000	5.400000
5	50%	2.000000	7.400000
6	75%	2.000000	9.200000
7	max	200.000000	650.000000

[409]:
```python
plt.figure(figsize=(18,7))
plt.rcParams['font.size'] = 16
plt.subplot(1,2,1)
bin = np.arange(1,5+1,1);
plt.hist(transactions_copy.PROD_QTY,bins=bin);
plt.xlabel('PROD_QTY');
plt.ylabel('Frequency');
plt.title('Distribution of PROD_QTY');

plt.subplot(1,2,2)
bin = np.arange(1.5, 30+1.5,1);
plt.hist(transactions_copy.TOT_SALES, bins=bin);
plt.xlabel('TOT_SALES');
plt.ylabel('Frequency');
plt.title('Distribution of TOT_SALES');
```



The above histograms clearly show that a majority of the data is located between 1 to 3 for PROD_QTY and 1 to 15 for TOT_SALES. Therefore is is safe to say that 200 and 650 for PROD_QTY and TOT_SALES are outliers respectively.

There it is droped from data set

[410]:
```
index = transactions_copy.query("TOT_SALES == 650").index.values
transactions_copy.drop(index,axis=0, inplace=True)
```

[411]:
```
transactions_copy.nlargest(6,'TOT_SALES')[['PROD_NAME','PROD_QTY','TOT_SALES']]
```

[411]:

	PROD_NAME	PROD_QTY	TOT_SALES
5179	Smiths Crnkle Chip Orgnl Big Bag	5	29.5
55558	Smiths Crnkle Chip Orgnl Big Bag	5	29.5
69496	Smiths Crnkle Chip Orgnl Big Bag	5	29.5
117850	Smiths Crnkle Chip Orgnl Big Bag	5	29.5
150683	Smiths Crnkle Chip Orgnl Big Bag	5	29.5
171815	Smiths Crnkle Chip Orgnl Big Bag	5	29.5

Issue #6: convert PREMIUM_CUSTOMER and LIFESTAGE columns to categorical data type

[412]:
```
purchase_behavor_copy.PREMIUM_CUSTOMER = purchase_behavor_copy.PREMIUM_CUSTOMER.
 ↪astype('category')
purchase_behavor_copy.LIFESTAGE = purchase_behavor_copy.LIFESTAGE.
 ↪astype('category')
```

[413]:
```
purchase_behavor_copy.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  category
 2   PREMIUM_CUSTOMER  72637 non-null  category
dtypes: category(2), int64(1)
memory usage: 709.9 KB
```

Add a unit cost column

[414]:
```
transactions_copy['UNIT_COST'] = np.divide(transactions_copy.
 ↪PROD_QTY,transactions_copy.TOT_SALES)
```

Finally merge two data sets together

[415]:
```
clean_data = pd.merge(transactions_copy, purchase_behavor_copy,␣
 ↪on='LYLTY_CARD_NBR')
```

Save clean dataset

[416]:
```
clean_data.to_csv('wrangled_data.csv')
```

3.3 Exploratory Data Analysis

[ ]:
```

```

[ ]:
```

```

3.4 Strategy & Conclusions

[ ]:
```

```