Data_preparation_and_customer_analytics

March 16, 2023

1 Regional Analysis of Chips Sales and Customer Behavior

1.1 Analyst: Albert Dellor

1.2 Table of Contents

1.3 1.0 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.

1.3.1 1.1 Data description

1.4 2.0 Data Wrangling

[1]:
```
# Import all packages and set plots to be embedded inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import requests
%matplotlib inline
```

[10]:
```
# Loading the dataset
# Get the URL of dataset
urls = ['https://cdn.theforage.com/vinternships/companyassets/32A6DqtsbF7LbKdcq/
 ↪QVI_transaction_data.xlsx',
        'https://cdn.theforage.com/vinternships/companyassets/32A6DqtsbF7LbKdcq/
 ↪QVI_purchase_behaviour.csv']

# Create request and get transaction and purchase behaviour datasets
for url in urls:
```

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

2.1 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 reveal
- Programmatic assessment in python

[2]:
```
transactions = pd.read_excel('QVI_transaction_data.xlsx')
purchase_behavor = pd.read_csv('QVI_purchase_behaviour.csv')
```

QVI__purchase__behaviour.csv

[3]:
```
purchase_behavor.sample(5)
```

[3]:

	LYLTY_CARD_NBR	LIFESTAGE	PREMIUM_CUSTOMER
37046	137084	YOUNG FAMILIES	Premium
9717	36080	OLDER SINGLES/COUPLES	Premium
28568	104228	OLDER SINGLES/COUPLES	Premium
54483	203381	RETIREES	Budget
28383	104043	OLDER SINGLES/COUPLES	Premium

[4]:
```
purchase_behavor.shape
```

[4]:
```
(72637, 3)
```

[5]:
```
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
```

[6]:
```
purchase_behavor.PREMIUM_CUSTOMER.nunique()
```

[6]:
```
3
```

[7]:
```
purchase_behavor.LIFESTAGE.nunique()
```

[7]:
```
7
```

PREMIUM_CUSTOMER and LIFESTAGE have few number of unique values, thus can be converted from string object to categorical date type as they hold no ordinal value

QVI_transaction_data.xlsx

[8]:
```
transactions.sample(5)
```

[8]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES
231175	43414	71	71219	70386	89	Kettle Sweet Chilli And Sour Cream 175g	2	10.8
232053	43450	95	95092	94538	30	Doritos Corn Chips Cheese Supreme 170g	2	8.8
31566	43389	196	196173	196505	40	Thins Chips Seasonedchicken 175g	2	6.6
156160	43450	70	70001	67487	78	Thins Chips Salt & Vinegar 175g	2	6.6
35055	43553	32	32173	29048	39	Smiths Crinkle Cut Tomato Salsa 150g	2	5.2

[9]:
```
# size of dataset
transactions.shape
```

[9]:
```
(264836, 8)
```

[10]:
```
# Glance over features and datatypes and missing values
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
```

There are no missing data

[11]:
```
# view descriptive statistics
transactions.describe()
```

[11]:
```
                DATE     STORE_NBR  LYLTY_CARD_NBR        TXN_ID  \
count  264836.000000  264836.00000    2.648360e+05  2.648360e+05
mean    43464.036260     135.08011    1.355495e+05  1.351583e+05
std       105.389282      76.78418    8.057998e+04  7.813303e+04
min     43282.000000       1.00000    1.000000e+03  1.000000e+00
25%     43373.000000      70.00000    7.002100e+04  6.760150e+04
50%     43464.000000     130.00000    1.303575e+05  1.351375e+05
75%     43555.000000     203.00000    2.030942e+05  2.027012e+05
max     43646.000000     272.00000    2.373711e+06  2.415841e+06

            PROD_NBR       PROD_QTY      TOT_SALES
count  264836.000000  264836.000000  264836.000000
mean       56.583157       1.907309       7.304200
std        32.826638       0.643654       3.083226
min         1.000000       1.000000       1.500000
25%        28.000000       2.000000       5.400000
50%        56.000000       2.000000       7.400000
75%        85.000000       2.000000       9.200000
max       114.000000     200.000000     650.000000
```

PROD_QTY and TOT_SALES have 75% of entire data being under 2 and9.2 respectively, however the maximun values are 200 and 650 respectively, there might be outliers present.

[12]:
```
# The only unique identifier in dataset is TXN_ID (transaction id)
# It is used to find duplicates
transactions[transactions.duplicated()]
```

[12]:
```
          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
```

[13]:
```
# investigate the transaction ID
transactions.query("TXN_ID == 108462")
```

[13]:
```
          DATE  STORE_NBR  LYLTY_CARD_NBR  TXN_ID  PROD_NBR  \
124843  43374        107          107024  108462        45
124844  43374        107          107024  108462        18
124845  43374        107          107024  108462        45

                                    PROD_NAME  PROD_QTY  TOT_SALES
```

```
124843  Smiths Thinly Cut   Roast Chicken 175g         2        6.0
124844           Cheetos Chs & Bacon Balls 190g         2        6.6
124845  Smiths Thinly Cut   Roast Chicken 175g         2        6.0
```

further investigation reveal that there was double recording of a transaction involving `Smiths Thinly Cut Roast Chicken 175g`

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)

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

[43]:
```
# 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

[44]:
```
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

[45]:
```
# drop transaction duplicates from dataset
transactions.drop_duplicates(inplace = True)
transactions.shape
```

[45]:
```
(264835, 8)
```

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

[46]:
```
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

[ ]:
```

```

[47]:
```
transactions_copy.nlargest(6,'TOT_SALES')
```

[47]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	\
69762	2018-08-21	226	226000	226201	4	
69763	2019-05-22	226	226000	226210	4	
5179	2018-08-17	94	94148	93390	14	
55558	2019-05-16	190	190113	190914	14	
69496	2018-08-17	49	49303	45789	14	
117850	2019-05-21	194	194308	194516	14	

	PROD_NAME	PROD_QTY	TOT_SALES	PROD_WEIGHT
69762	Dorito Corn Chp Supreme	200	650.0	380
69763	Dorito Corn Chp Supreme	200	650.0	380
5179	Smiths Crnkle Chip Orgnl Big Bag	5	29.5	380
55558	Smiths Crnkle Chip Orgnl Big Bag	5	29.5	380
69496	Smiths Crnkle Chip Orgnl Big Bag	5	29.5	380
117850	Smiths Crnkle Chip Orgnl Big Bag	5	29.5	380

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

[48]:
```
stats = transactions_copy.describe()
```

[49]:
```
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 {} and {}".
↪format(column,low_range,upper_range))
```

[50]:
```
outlier('PROD_QTY')
outlier('TOT_SALES')
```

```
For PROD_QTY an outlier is any value outside the range of 2.0 and 2.0
For TOT_SALES an outlier is any value outside the range of -0.29999999999999805
and 14.899999999999999
```

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

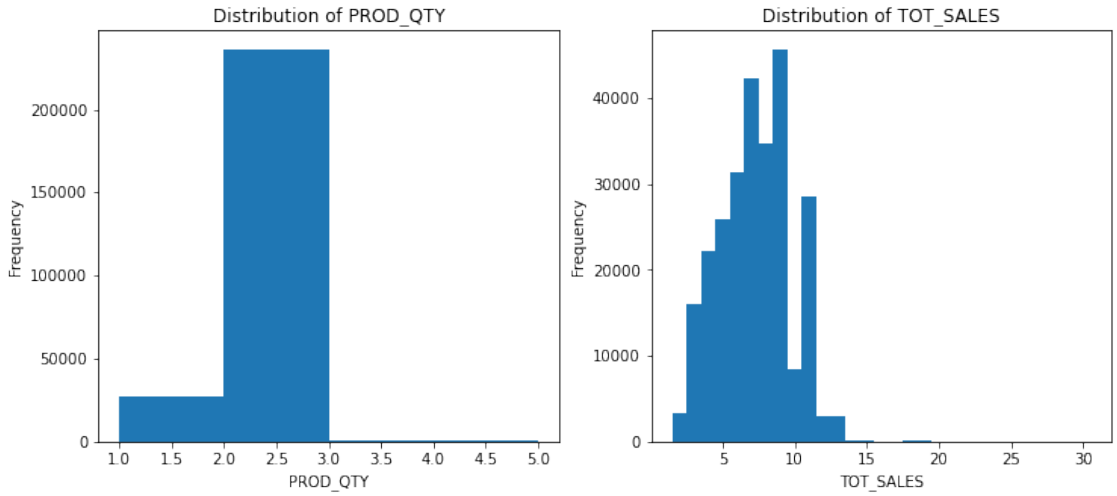
[90]:
```
# Looking at the descriptive statistics of both data sets
pd.merge(pd.DataFrame(transactions_copy.PROD_QTY.describe()),\
         pd.DataFrame(transactions_copy.TOT_SALES.describe()),\
         on=pd.DataFrame(transactions_copy.TOT_SALES.describe()).index).
 ↪rename(columns={'key_0':'index'})
```

[90]:

	index	PROD_QTY	TOT_SALES
0	count	264835.000000	264835.000000
1	mean	1.907308	7.304205
2	std	0.643655	3.083231
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

[93]:
```
plt.figure(figsize=(12,5))
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

[94]:
```
# drop the outlier rows
index = transactions_copy.query("TOT_SALES == 650").index.values
transactions_copy.drop(index,axis=0, inplace=True)
```

[95]:
```
transactions_copy.nlargest(6,'TOT_SALES')
```

[95]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PROD_WEIGHT
5179	2018-08-17	94	94148	93390	14	Smiths Crnkle Chip Orgnl Big Bag	5	29.5	380
55558	2019-05-16	190	190113	190914	14	Smiths Crnkle Chip Orgnl Big Bag	5	29.5	380
69496	2018-08-17	49	49303	45789	14	Smiths Crnkle Chip Orgnl Big Bag	5	29.5	380
117850	2019-05-21	194	194308	194516	14	Smiths Crnkle Chip Orgnl Big Bag	5	29.5	380
150683	2019-05-22	118	118021	120799	14	Smiths Crnkle Chip Orgnl Big Bag	5	29.5	380
171815	2018-08-19	24	24095	20797	14	Smiths Crnkle Chip Orgnl Big Bag	5	29.5	380

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

[99]:
```
purchase_behavor_copy.PREMIUM_CUSTOMER = purchase_behavor_copy.PREMIUM_CUSTOMER.
  ↪astype('category')
purchase_behavor_copy.LIFESTAGE = purchase_behavor_copy.LIFESTAGE.
  ↪astype('category')
```

[100]:
```
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

[101]:
```
transactions_copy['UNIT_COST'] = np.divide(transactions_copy.
  ↪PROD_QTY,transactions_copy.TOT_SALES)
```

[106]:
```
transactions_copy.sample(3)
```

[106]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PROD_WEIGHT	UNIT_COST
105754	2018-11-06	88	88060	86516	4	Dorito Corn Chp Supreme	2	13.0	380	0.153846
65527	2018-08-03	185	185453	188354	97	RRD Salt & Vinegar	2	6.0	165	0.333333
107184	2019-06-06	107	107058	108662	49	Infuzions SourCream & Herbs Veg Strws	2	7.6	110	0.263158

Finally merge two data sets together

[104]:
```
clean_data = pd.merge(transactions_copy, purchase_behavor_copy,␣
  ↪on='LYLTY_CARD_NBR')
```

Save clean dataset

[105]:
```
clean_data.to_csv('wrangled_data.csv')
```

1.5 Exploratory Data Analysis

[ ]:

[ ]:

1.6 Strategy & Conclusions

[ ]: