Product User Profile Generation - Part 2

(PUPs)

Test Purpose

Author: Ingrid Cadu

Last Update: ago 15, 2022

Note

In order to see the plots, please run the script on jupiter notebook or similar.

In [1]:
```
# Libraries
import pandas as pd # To dataframe
import numpy as np # To number manipulation
from urllib.request import Request, urlopen # To get url pages
from bs4 import BeautifulSoup # To extract info
import plotly.express as px # To plot chart
import plotly.graph_objects as go #To plot chart
```

```
/home/ingrid_silva/.local/lib/python3.9/site-packages/pandas/compat/_optional.py:161: UserWarning: Pandas requires version '1.3.1' or newer of 'bottleneck' (version '1.2.1' currently installed).
  warnings.warn(msg, UserWarning)
```

In [48]:
```
### FIRST ROW!
# Name, Basket Market
basket = pd.read_json("./pup.json")

#Price difference from the category and Competitors price
price = pd.read_csv("./20220914_PSWORK.csv")

#Loyalty, Service Rating and Rating
feeling = pd.read_csv("./20221409_Product_Demo_Info.csv")

# Cost per store
cost = pd.read_csv("./20220914_Store_Sale_Normal_price.csv")

# Retail most purchased by users - link ;)
retail = "https://brandirectory.com/rankings/retail/table"
```

In [56]:
```
# Used as filter ;)
```

```
# Possible Options:
['Gillette','Keebler','Logitech','Olay','Oral-
B','Pampers','Tide']

string = "Gillette"
```

First page: Product

General info

- Name
  - Price difference from the category
- Trends
  - Basket
  - Rating
  - Service Rating
  - Loyalty
- competitors price

In [57]:

```
def filter_google(dataframe,column,fil):
    """
    Function to filter dataframe accordingly a string previously
set.
    Input:
    - dataframe: data used to filter
    - columns: column used to look through
    - fil: string to look for
    Output:
    - data: related row containing the string
    """
    data = pd.DataFrame()
    rows=-1
    for x in dataframe[column]:
        rows = rows+1
        if (x == fil).any():
            category = dataframe['categories_textform'][rows]
            father = set(x)
            child = pd.DataFrame(father)[:6]
            data = pd.concat([data,child])
        else:
            pass
    return data,category
```

In [58]:

```
# Price diff and competitors per category
#Grouping it into one table
competite = price.groupby(['categories_textform'])
['short_name'].unique().to_frame().reset_index()

# Applying funtion and getting dataframe and category
enemy,cats = filter_google(competite,"short_name",string)

# Transforming into dataframe
enemy = enemy.rename(columns={0:'short_name'})
enemy.loc[-1] = [string]  # adding a row
enemy.index = enemy.index + 1  # shifting index

if len(enemy) > 0:
    cobi = pd.merge(enemy, price, left_on='short_name',
right_on='short_name', how='inner')
    cobi = cobi.dropna(subset=['price'])
    general = cobi.groupby(['short_name'])
['price'].min().to_frame().reset_index()
    if len(general) > 1:
        general['percentage'] = general['price'].apply(lambda x:
round(x/general['price'].mean()*general['price'].std()),1)
    else:
        general['percentage'] = 0
```

In [59]:

```
#Ploting
inc_plot = go.Figure(go.Bar(
            x=general['price'],
            y=general['short_name'], name='ACTUAL<br>COMPETITOR',
            orientation='h'))

inc_plot.update_layout(
    yaxis=dict(
        showline=True,
        showgrid=True,
        gridcolor="#eee",
        showticklabels=True,
        linecolor='#eee',
        linewidth=2,
        ticks='outside',
        tickfont=dict(family='Arial',size=12, color='rgb(82, 82,
```

```
82)')),
    autosize=False,
    margin=dict(l=5,r=5,b=5,t=50,pad=4),
    showlegend=False,
    width=600,
    height=300,
    paper_bgcolor="white",
    plot_bgcolor='white')
inc_plot.update_traces(marker_color='#85b3e0', hovertemplate="%
{y}<br>%{x:.2f}%",
                       selector=dict(type='bar'))
inc_plot.update_xaxes(showticklabels=False, showgrid=True)
inc_plot.show()
```

In [60]:
```
# Getting the percentage of product in relation to category
best_pp = general[general['short_name'].str.contains(string)]
percent = best_pp['percentage'].mean()
percent
```

Out[60]:
```
30.5
```

In [61]:
```
# Getting the product category
cats
```

Out[61]:
```
'PERSONAL CARE'
```

In [62]:
```
# Related ;)
# Creating list of related items
basket_mark = basket.loc[:,['brand','related_items']].copy()
```

```
# Dropping the empty lists
clean_basket =
basket_mark[basket_mark['related_items'].map(bool)]

# filtering here
bak = clean_basket[clean_basket['brand'].str.contains(string)]

# Checking if the value is less than zero
if len(bak)<=0:
    pass
else:
    baks = bak['related_items'].apply(pd.Series).T
    baks.columns = ['short_name']
    cteg = pd.merge(baks,price, left_on='short_name',
right_on='short_name', how='inner')


cteg[:3]
```

Out[62]:

	short_name	Unnamed: 0	categories_textform	price	sales_count
0	Tide	130	GROCERIES	8.49	519.0
1	Tide	344	GROCERIES	8.49	519.0
2	Tide	884	GROCERIES	8.49	519.0

In [87]:

```
# Related Items

basket_mark[basket_mark['brand'].str.contains(string)]
```

Out[87]:

	brand	related_items
231	Gillette	[Suave, Colgate, Tide, Dove]

In [88]:

```
# Why they did not appeared above? -> the name brand did not
match the strings!
price[price['short_name'].str.contains("Colgate ")]
```

Out[88]:

	Unnamed: 0	categories_textform	short_name	price	sales_count
484	484	PERSONAL CARE	Colgate Toothpaste	3.0	36.0
2045	2045	GROCERIES	Colgate Toothpaste	3.0	57.0
2265	2265	PERSONAL CARE	Colgate Toothpaste	3.0	36.0
3061	3061	PERSONAL CARE	Colgate Toothpaste	3.0	36.0
3071	3071	GROCERIES	Colgate Toothpaste	3.0	57.0
3721	3721	GROCERIES	Colgate Toothpaste	3.0	57.0
3831	3831	PERSONAL CARE	Colgate Toothpaste	3.0	36.0
4747	4747	GROCERIES	Colgate Toothpaste	3.0	57.0

In [63]:

```
# Basket output
if len(cteg['short_name'].values) == 0:
    ctk_nm_0 = 'No Data Avalaible'
    ctk_nm_1 = 'No Data Avalaible'
    ctk_nm_2 = 'No Data Avalaible'
    ctk_ct_0 = 'No Data Avalaible'
    ctk_ct_1 = 'No Data Avalaible'
    ctk_ct_2 = 'No Data Avalaible'
else:
    # Basket Names
    ctk_nm_0 = cteg['short_name'].values[0]
    ctk_nm_1 = cteg['short_name'].values[1]
    ctk_nm_2 = cteg['short_name'].values[2]

    # Basket categories
    ctk_ct_0 = cteg['categories_textform'].values[0]
    ctk_ct_1 = cteg['categories_textform'].values[1]
    ctk_ct_2 = cteg['categories_textform'].values[2]
```

In [65]:

```
# First related
ctk_nm_0
```

Out[65]:

```
'Tide'
```

In [66]:

```
# Service Rating ;)
# Removing NA values from brands
feeling = feeling.dropna(axis=0, subset=['brand'])

# Grouping brands by name
group_brand = feeling.groupby(['brand'])
[['1.0','2.0','3.0','4.0','5.0']].mean().reset_index()
```

```
# Getting the highest star
group_brand['higest_score'] =
group_brand[['1.0','2.0','3.0','4.0','5.0']].apply(lambda x:
x.idxmax(axis=0), axis=1)

# Getting the rate and its value (%)
group_brand['valuable'] = group_brand.apply(lambda row:
row[row.higest_score], axis=1)
group_brand['valuable'] = round(group_brand['valuable'], 1)

# Applying filter
rating = group_brand[group_brand['brand'].str.contains(string)]

# Rating ;)
rates = feeling[feeling['brand'].str.contains(string)]
rt = str(rates['ratings'].values)
print(rt.strip("[]"))
```

```
3.9
```

In [67]:

```
# Loyalty ;)
loyalty = feeling.dropna(axis=0, subset=['brand'])

#
group_loy = feeling.groupby(['brand'])
[['loyal','disloyal','neutral']].mean().reset_index()

#
group_loy['higest_score'] =
group_loy[['loyal','disloyal','neutral']].apply(lambda x:
x.idxmax(axis=0), axis=1)

# Getting the rate and its value (%)
group_loy['valuable'] = group_loy.apply(lambda row:
row[row.higest_score], axis=1)
group_loy['valuable'] = round(group_loy['valuable'], 1)

# Applying filter
group_loy[group_loy['brand'].str.contains(string)]
```

Out[67]:

	brand	loyal	disloyal	neutral	higest_score	valuable
640	Gillette	33.415	33.32	33.266	loyal	33.4

Stores

- Retail most purchased by users (text)
- Cost per store (graph - lollipop)
- Sales per category (graph - de quadradinho XD)

In [68]:

```
# Scrapping the retail store most purchased from web...
req = Request(retail, headers={'User-Agent': 'Mozilla/5.0'})
webpage = urlopen(req).read()

#Creating a loop to scrap on webpage <!ALWAYS CHECK THE PAGE
BEFORE UPDATE!>
table = list()
soup = BeautifulSoup(webpage,'html.parser')
tags = soup.find_all('td', class_='')
for tag in tags:
    x = tag.find('span')
    word = x.text.strip()
    if word.isdigit():
        pass
    else:
        table.append(word)

# Cleaning data
usa = table[:12]

# Names
name_usa = {'name_retail':[usa[0],usa[4],usa[8]], 'rank':
[usa[2],usa[6],usa[10]]}

# Creating a table
rank_retails = pd.DataFrame.from_records(name_usa, columns=
['name_retail','rank'])
rank_retails
```

Out[68]:

	name_retail	rank
0	Amazon	AAA+
1	Walmart	AAA-
2	Home Depot	AAA-

In [69]:

```
# Cost per Store <! what if more than one retail located in
different place?>
# apply min price and show it!
#Filtering accordingly the product
select_store = cost[cost['name_product'].str.contains(string)]

#Sellecting what we want ;)
sell = select_store.loc[:,['name_store',
                           'state',
                           'zip_code',
                           'price',
                           'inventory_price',
                          'sales_count']].copy()

# Setting name uppercase to match with the retail style
sell['name_store'] = sell['name_store'].str.title()

maps = pd.DataFrame() # Keep name
holy = pd.DataFrame() # keep no name
# Getting the store which match the retail ranking
for x in rank_retails['name_retail']:
    if (sell['name_store'].str.contains(x)).any():
        daisy = sell[sell['name_store'].str.contains(x)]
        maps = pd.concat([maps,daisy], axis=0)
    else:
        gol = {'name_store': [x],'state':[pd.NA],
               'zip_code':[pd.NA],'price':[pd.NA],
               'inventory_price':[pd.NA]}
        doly = pd.DataFrame(gol)
        holy = pd.concat([holy,doly])


# Concating to show the data
maps = pd.concat([maps,holy], axis=0)
lowest = sell[sell['price']==sell['price'].min()]
low_map = pd.concat([lowest,maps], axis=0)
```

```
low_map = low_map.sort_values(by="price", ascending=True)
low_map
#other_store = {'name_store':[],}
```

Out[69]:

	name_store	state	zip_code	price	inventory_price	sales_count
231	Westside Station	AR	72035	5.49	5.99	5.0
0	Amazon	<NA>	<NA>	NaN	NaN	NaN
0	Walmart	<NA>	<NA>	NaN	NaN	NaN
0	Home Depot	<NA>	<NA>	NaN	NaN	NaN

In [70]:

```
# Selecting stores
select_store = cost[cost['name_product'].str.contains(string)]

# All stores here!
pur_store = select_store.loc[:,
['name_store','state','zip_code','price',

'inventory_price','sales_count']].copy()

pur_store = pur_store[pur_store['price'] ==
pur_store['price'].min()]
pur_store
```

Out[70]:

	name_store	state	zip_code	price	inventory_price	sales_count
231	westside station	AR	72035	5.49	5.99	5.0

In [71]:

```
# Picking out the store
name_store = pur_store['name_store'].values[0]
state_store = pur_store['state'].values[0]
zip_store = pur_store['zip_code'].values[0]
price_store = pur_store['inventory_price'].values[0]
sales_store = pur_store['sales_count'].values[0]

# Getting info about the store selling the
low_map['name_store'].values[0]
```

Out[71]:

```
'Westside Station'
```

Second page: Users

Product Demographic

- Feeling for age
- income demo
- race demo

In [72]:
```
### SECOND ROW !!!

#feeling per age
age = pd.read_csv("./20220914_Product_By_Age.csv")
income = pd.read_csv("./20220914_Product_Income_Demo.csv")
```

In [73]:
```
# Getting the data from product
#
https://en.wikipedia.org/wiki/Erikson's_stages_of_psychosocial_deve

aging = age[age['brand'].str.contains(string)]

# Picking out the ages available
aging['age'] = aging['age'].replace({'Under 5 years': 'Early
childhood',
                                     '5 to 9 years': 'Middle
Childhood',
                                     '10 to 14 years':
'Adolescence',
                                     '15 to 19 years':
'Adolescence',
                                     '20 to 24 years': 'Early
adulthood',
                                     '25 to 34 years': 'Early
adulthood',
                                     '35 to 44 years': 'Early
adulthood',
                                     '45 to 54 years': 'Middle
Adulthood',
                                     '55 to 59 years': 'Middle
Adulthood',
                                     '60 to 64 years': 'Middle
Adulthood',
                                     '65 to 74 years': 'Late
Adulthood',
                                     '75 to 84 years': 'Late
Adulthood',
                                     '85 years and over':
```

```
'Late Adulthood'})

# Grouping by stage to generate chart
group_stage_age = aging.groupby(['age'])
[['sentiment_pos','sentiment_neg','sentiment_neu']].mean()
group_stage_age = round(group_stage_age, 2)

# Selecting the most important cols
group_stage_age['higest_score'] = group_stage_age.loc[:,
['sentiment_pos',

'sentiment_neg',

'sentiment_neu']].apply(lambda x: x.idxmax(axis=0), axis=1)

# Getting the rate and its value (%)
group_stage_age['valuable'] = group_stage_age.apply(lambda row:
row[row.higest_score], axis=1)
group_stage_age['valuable'] = round(group_stage_age['valuable'],
1)
group_stage_age = group_stage_age.loc[:,
['higest_score','valuable']].copy()

# Ploting ages ;)
age_plot = go.Figure()
age_plot.add_trace(go.Scatter(x=group_stage_age.index,
y=group_stage_age['valuable'],
                              mode='lines',
line=dict(color='#dbdbdb', width=2),
                              connectgaps=True))
age_plot.add_trace(go.Scatter(
    x=group_stage_age.index,
    y=group_stage_age['valuable'],
    name='DEMOGRAPHIC<br>SENTIMENT',#put the varible inputed
here!
    mode='markers', marker=dict(color='#e65e19', size=50)))

# Age feeling
age_plot.update_layout(
    xaxis=dict(
        showline=True,
```

```
        showgrid=True,
        gridcolor="#eee",
        showticklabels=True,
        linecolor='#eee',
        linewidth=2,
        ticks='outside',
        tickfont=dict(family='Arial',size=12,color='rgb(82, 82,
82)')),
    autosize=False,
    margin=dict(l=5,r=5,b=5,t=5,pad=4),
    showlegend=False,
    width=700,
    height=300,
    paper_bgcolor="white",
    plot_bgcolor='white')
age_plot.update_yaxes(showticklabels=False,
showgrid=True,fixedrange=True) # hide all the xticks
age_plot.update_traces(hovertemplate="%{x}<br>%{y:.2f}%",
selector=dict(type='scatter'))
age_plot.show(config=dict(displayModeBar=False))
```

```
/tmp/ipykernel_4059/2926589280.py:6: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [74]:

```
### Plotting the User Income!
```

```
income_list =
income[income['brand'].str.contains(string)].reset_index()
income_list = round(income_list,2)

# Ordering the cols
income_list = income_list.loc[:,
['14,999','24,999','34,999','49,999',

'74,999','99,999','149,999','199,999','200,000']].copy()
#Plotting!!!
income_list = pd.to_numeric(np.array(income_list).ravel())
labels = ['14,999','24,999','34,999','49,999',
          '74,999','99,999','149,999','199,999',
          '200,000']

#Ploting
inc_plot = go.Figure(go.Bar(
            x=income_list,
            y=labels, name='DEMOGRAPHIC<br>INCOME',
            orientation='h'))

inc_plot.update_layout(
    yaxis=dict(
        showline=True,
        showgrid=True,
        gridcolor="#eee",
        showticklabels=True,
        linecolor='#eee',
        linewidth=2,
        ticks='outside',
        tickfont=dict(family='Arial',size=12, color='rgb(82, 82,
82)')),
    autosize=False,
    margin=dict(l=5,r=5,b=5,t=50,pad=4),
    showlegend=False,
    width=600,
    height=300,
    paper_bgcolor="white",
    plot_bgcolor='white')
inc_plot.update_traces(marker_color='#85b3e0', hovertemplate="%
```

```
{y}<br>%{x:.2f}%",
                        selector=dict(type='bar'))
inc_plot.update_xaxes(showticklabels=False, showgrid=True)
inc_plot.show()
```

In [75]:

```
### Plotting the User Race !
races = feeling[feeling['brand'].str.contains(string)]
num = str(races['race_demo'].values).strip("'['']'")
vac = pd.to_numeric(num)

#creating the variables
label = ["White","Others"]
value = [vac, 100-vac]
colors =['#304050','#85b3e0']

race_plot = go.Figure(data=[go.Pie(labels=label,
                            values=value,

hole=.85,name='DEMOGRAPHIC<br>AGE',textinfo='none')])
race_plot.update_layout(width=300,
                  height=300,
                  showlegend=True,
                  margin=dict(l=0,r=0,b=30,t=5,pad=4),
                  paper_bgcolor='#fff',
                  plot_bgcolor='#fff',
                  legend=dict(font_size=15,
                            yanchor="top",
                             y=0,
                             xanchor="left",
```

```
                                x=0))
race_plot.update_traces(marker=dict(colors=colors),
hoverinfo='label+percent+name', selector=dict(type='pie'))
race_plot.show()
```

In [76]:

```
# Getting unique price brand from all data
xi = price['short_name'].unique()

# Getting the brand within basket and rate data
basket_brand = basket_mark[basket_mark['brand'].isin(xi)]
rating_brand = group_brand[group_brand['brand'].isin(xi)]
age_brand = age[age['brand'].isin(xi)]

# Getting the brand in both to filter all!
rat_list = rating_brand['brand'].unique()
bas_list = basket_mark[basket_mark['brand'].isin(rat_list)] # has
15

# If both price and brand are equal than i just need to apply one
of them over age ;)
fil_bas = bas_list['brand'].unique()
age_list = age[age['brand'].isin(fil_bas)] # has 10!
options = age_list['brand'].unique()
print(options)
```

```
['Gerber' 'Gillette' 'Keebler' 'Logitech' 'Olay' 'Oral-B' 'Pampers' 'Ragu'
 'Staedtler' 'Tide']
```