# Product User Profile Generation - Part 1

# (PUPs)

## Cleaning Purpose

Author: Ingrid Cadu

Last Update: ago 15, 2022

Note. The empty values are filled by the mean.
If we apply the inferencial statistics logical, if we infer a sample over the population the new mean will be the mean of total population.
In this case, the mean of empty values (sample infered by the non-values) will be the mean of the infered sample. :D!

Data Source:

- pup.json
- brand_affinity2.json
- teststore_product*sale**.json

# Libraries and Data

In [2]:
```
# Libraries
import pandas as pd
import numpy as np
from glob import glob
from textblob import TextBlob
```

```
/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 [3]:
```
# Databases - Price and Product
products = pd.read_json("./price/pup.json")
brands = pd.read_json("./price/brand_affinity2.json")
```

In [4]:
```
#Stores data
df = sorted(glob("./test/teststore_product_sale_*.json"))
stores = pd.concat((pd.read_json(file) for file in df),
ignore_index=True)
```

# Table 1. Product Affinity

## Product Affinity

> Product Affinity - Affinity of a given product based on categories (rating ;D)

## Affinity

> Affinity - Rating of 0-20, with 0 being 0% and 20 being 100%

## Products

In [5]:
```
# Splitting Brands col...
brang = brands['brands'].apply(pd.Series)

# Taking the cols we need
br = brands.loc[:,['affinity','category']].copy()
# Creating a new dataframe
BRD = pd.concat([brang, br], axis=1)
BRD.columns = ['brand_one','brand_two','affinity','category']
BRD[:2]
```

Out[5]:

| | brand_one | brand_two | affinity | category |
|---|---|---|---|---|
| **0** | Air Wick | Aquafina | 8.79 | GROCERIES |
| **1** | Air Wick | Aquafresh | 9.00 | GROCERIES |

## Affinity

In [6]:
```
# Setting a rank
BRD['rating'] = pd.qcut(BRD['affinity'],5,labels=[0,5,10,15,20])
BRD['rating'].value_counts()
```

Out[6]:
```
15    1174
0     1170
5     1144
10    1129
20    1118
Name: rating, dtype: int64
```

## Table 1. Product Affinity

In [7]:
```
# Save in csv
BRD.to_csv("./20220914_Product_Affinity.csv")
```

```
BRD[:5]
```

Out[7]:

| | brand_one | brand_two | affinity | category | rating |
|---|---|---|---|---|---|
| **0** | Air Wick | Aquafina | 8.79 | GROCERIES | 5 |
| **1** | Air Wick | Aquafresh | 9.00 | GROCERIES | 10 |
| **2** | Air Wick | Arm & Hammer | 9.74 | GROCERIES | 20 |
| **3** | Air Wick | Arm & Hammer | 9.39 | HOUSEHOLD ESSENTIALS | 15 |
| **4** | Air Wick | Aveeno | 9.26 | GROCERIES | 15 |

# Table 2. Product Section

## Pt. 1 - General Info

### Name

> Name - name of the product

### Year

> Year Release - Year the product is released (optional)

### Time of day most bought

> Time of day most bought - Time of day that it the product is usually bought (optional)

### Related Items Bought

> Related Items Bought - Product names of related items that are commonly bought alongside with the product

## Year

In [8]:

```
# removing object XP
products['year_released'] =
np.where(products['year_released']=="Google",0,
products['year_released'])

# Casting datatype
products['year_released'] =
products['year_released'].apply(lambda x: pd.to_numeric(x))
```

## Time of day

Note:

- Data has 5K unique brands

**Issue:**

- Inconsistent: Data is missing too much values and can't be supposed randomly.

In [9]:
```
# Cleaning the data to get just dicts
time = products['time_of_day'].dropna()

bags = {'brand':[], 'time_of_day':[], 'pct_time_of_day':[]}
for key, value in time.items():
    if len(value)>1:
        for black in value:
            bags['brand'].append(black.get('brand'))
            bags['time_of_day'].append(black.get('time_of_day'))

bags['pct_time_of_day'].append(black.get('pct_time_of_day'))
    else:
        pass


# Transforming into a dataframe
time_of_day = pd.DataFrame(bags)

# Grouping them!
time_group = time_of_day.groupby(['brand'])
['time_of_day'].max().to_frame().reset_index()
time_group.time_of_day.value_counts()
```

Out[9]:
```
Noon    454
Name: time_of_day, dtype: int64
```

## Related Items Bought

In [11]:
```
# link with stores!
prod_section = products.loc[:,
['brand','year_released','related_items']].copy()

# Products empty rows
prod_section =
prod_section[prod_section['related_items'].map(bool)]
```

```
# Merging time and save as csv
merged_time = pd.merge(prod_section, time_group,
                       left_on='brand',
                       right_on='brand',
                       how='outer')
```

In [12]:
```
prod_section[:5]
```

Out[12]:

| | brand | year_released | related_items |
|---|---|---|---|
| **0** | Clairol | 1956.0 | [Suave, Mr. Clean, Colgate , Cover Girl] |
| **1** | Persil | 1907.0 | [Oxi Clean, Palmolive, Scott, Downy] |
| **2** | Purell | 1988.0 | [Crayola, Elmer's, Bic, Softsoap] |
| **3** | Duracell | 1965.0 | [Ziploc, Dixie, Tide, Downy] |
| **4** | Old Spice | 1937.0 | [Suave, Colgate, Dove, Dial] |

In [14]:
```
# View table
merged_time[:5]
```

Out[14]:

| | brand | year_released | related_items | time_of_day |
|---|---|---|---|---|
| **3132** | 100 plus | NaN | [] | NaN |
| **1208** | 1960 | NaN | [Christopher C. King, Zondervan, VARIOUS ARTIS... | NaN |
| **1207** | 1962 | NaN | [Lakeside Games, Milton Bradley, The Beatles, ... | NaN |
| **4691** | 2XU | NaN | [Oiselle, Zoot, Mizuno, CW-X, Copper Fit, De S... | NaN |
| **3769** | 3 Musketeers | NaN | [M&M's, Kit Kat, Starbucks, Mars, Best Foods, ... | NaN |

In [15]:
```
# How much non-null rows this table has?
#print(f'{round(len(merged_time)/len(products)*100, 1)}% of data from original.')
```

# Table 2. Product Section

## Pt. 2 - Demographic Info

### Product Loyalty

> Product Loyalty - Measure of consumer loyalty for a given product

### Sentiment by Demographic

> Sentiment by Demographic: Measure of age sentiment

## Product Ratings

> Product Ratings: Rating of 0-5, with 0 being 0% and 5 being 100%

## Customer Service Ratings

> Customer Service Ratings: Measure of customer's star

## General Demographic Sentiment

> General Demographic Sentiment: General sentiment measured by age

## Income Demogs

## Race Demogs

# Product Loyalty

In [16]:
```
# Setting data to use
voucher = products.loc[:,['brand','general_loyalty']].dropna()

# Setting a dict for loyalty data
loyal = {'loyal':[], 'disloyal':[], 'neutral':[]}
for key, value in voucher['general_loyalty'].items():
    if len(value)>1:
        loyal['loyal'].append(value.get('loyal')*100)
        loyal['disloyal'].append(value.get('disloyal')*100)
        loyal['neutral'].append(value.get('neutral')*100)
    else:
        pass

# Transforming into dataframe
general_loyalty = pd.DataFrame(loyal).set_index(voucher['brand'])
general_loyalty = round(general_loyalty,3)
general_loyalty[:3]
```

Out[16]:

| brand | loyal | disloyal | neutral |
|---|---|---|---|
| **Clairol** | 33.480 | 33.229 | 33.291 |
| **Persil** | 33.385 | 33.419 | 33.196 |
| **Purell** | 33.276 | 33.313 | 33.412 |

In [17]:
```
# How much non-null rows this table has?
print(f'{round(len(general_loyalty)/len(products)*100, 1)}% of
```

```
data from original.')
```

```
8.3% of data from original.
```

## Sentiment by Demographic

In [18]:

```
# Ages feeling
sent = {'id':[],'age':[],'sentiment_pos':[],'sentiment_neg':
[],'sentiment_neu':[]}
row = 0

for aloha, dubai in products['age_sentiment'].items():
    if np.array(dubai).dtype==float:
        sent['id'].append(row)
        sent['age'].append(pd.NA)
        sent['sentiment_pos'].append(pd.NA)
        sent['sentiment_neg'].append(pd.NA)
        sent['sentiment_neu'].append(pd.NA)
    else:
        for dool in dubai:
            sent['id'].append(row)
            sent['age'].append(dool.get('age'))

sent['sentiment_pos'].append(dool.get('sentiment_pos'))

sent['sentiment_neg'].append(dool.get('sentiment_neg'))

sent['sentiment_neu'].append(dool.get('sentiment_neu'))
    row = row+1


# Creating dataframe
aging = pd.DataFrame(sent)
aging = aging.dropna()

# Without merging by now
new_prod = products.dropna(axis=0, subset=['age_sentiment'])
AGES = aging.merge(new_prod['brand'], left_on='id',
right_on=new_prod.index)
AGES.drop('id', inplace=True, axis=1)
```

In [19]:

```
# Changing the datatype
```

```
AGES['sentiment_pos'] = pd.to_numeric(AGES['sentiment_pos'])*100
AGES['sentiment_neg'] = pd.to_numeric(AGES['sentiment_neg'])*100
AGES['sentiment_neu'] = pd.to_numeric(AGES['sentiment_neu'])*100

# Rouding
AGES = round(AGES, 4)
```

In [20]:

```
# Collecting the feeling
AGES = AGES.sort_values(by='brand', ascending=True)
[['brand','age','sentiment_pos',

'sentiment_neg','sentiment_neu']].copy()
AGES[:5]
```

Out[20]:

| | brand | age | sentiment_pos | sentiment_neg | sentiment_neu |
|---|---|---|---|---|---|
| **3254** | 7Up | 35 to 44 years | 32.6842 | 33.6272 | 33.6886 |
| **3261** | 7Up | 85 years and over | 33.5000 | 32.5921 | 33.9079 |
| **3262** | 7Up | Under 5 years | 33.1974 | 33.0439 | 33.7588 |
| **3260** | 7Up | 75 to 84 years | 33.5351 | 33.1140 | 33.3509 |
| **3259** | 7Up | 65 to 74 years | 33.0702 | 33.4167 | 33.5132 |

In [21]:

```
# Save as csv
AGES.to_csv("./20220914_Product_By_Age.csv")
```

In [22]:

```
# How much non-null rows this table has?
print(f'{round((len(AGES)/13)/len(products)*100, 1)}% of data
from original.')
```

```
8.3% of data from original.
```

## Product Ratings

In [23]:

```
# Checking the rating from clients
bags = list()
for x in products['ratings']:
    if np.array(x).dtype==float:
        bags.append(x)
    else:
        bags.append(0)

# Creatinf a dict
prd_rate = {'brand':products['brand'], 'ratings':bags}
```

```
# Creating a data from it
PRD = pd.DataFrame.from_records(prd_rate, columns=
['brand','ratings'])
#PRD['ratings'] = PRD['ratings'].replace(0,
products['ratings'].mean())
PRD['ratings'] = round(PRD['ratings'], 1)
PRD = PRD.sort_values(by='brand', ascending=True)
PRD[:5]
```

Out[23]:

| | brand | ratings |
|---|---|---|
| **3804** | 100 plus | 4.4 |
| **1437** | 1960 | 4.1 |
| **1436** | 1962 | 4.1 |
| **1345** | 25 Hours | 3.6 |
| **3666** | 28 Black | 5.0 |

In [24]:

```
# How much non-null rows this table has?
print(f'{round(len(PRD)/len(products)*100, 1)}% of data from
original.')
```

```
100.0% of data from original.
```

## Customer Service Ratings

In [25]:

```
# Cleaning the data to get just dicts
cust_rate = products['customer_service_ratings'].dropna()

rates = {'brand':[], 'star':[], 'rating':[]}
for key, value in cust_rate.items():
    if len(value)>1:
        for black in value:
            rates['brand'].append(black.get('brand'))
            rates['star'].append(black.get('star'))
            rates['rating'].append(black.get('rating')*100)
    else:
        pass

# Transformiong dict to dataframe
customer_service = pd.DataFrame(rates)

# Grouping them!
```

```
customer_service_ratings =
customer_service.pivot_table(index="brand",
                             columns="star",

values="rating").sort_values(by='brand')
# View
customer_service_ratings[:5]
```

Out[25]:

| brand | star 1.0 | 2.0 | 3.0 | 4.0 | 5.0 |
|---|---|---|---|---|---|
| **7Up** | 26.0 | 6.0 | 0.0 | 13.0 | 56.0 |
| **9Lives** | 7.0 | 8.0 | 8.0 | 14.0 | 62.0 |
| **ACT Oral Care** | 21.0 | 0.0 | 0.0 | 21.0 | 59.0 |
| **Absolut** | 2.0 | 2.0 | 4.0 | 4.0 | 89.0 |
| **Adams (Peanut Butter)** | 4.0 | 3.0 | 9.0 | 13.0 | 71.0 |

In [26]:

```
# How much non-null rows this table has?
print(f'{round(len(customer_service_ratings)/len(products)*100,
1)}% of data from original.')
```

```
8.2% of data from original.
```

## General Demographic Sentiment

In [27]:

```
# Setting general sentiment
gema = products.dropna(axis=0, subset=['general_sentiment'])

general_sentiment =
pd.DataFrame.from_records(gema['general_sentiment'],
                                    columns=['sentiment_pos',
                                             'sentiment_neg',

'sentiment_neu']).set_index(gema['brand'])

# Viewing
general_sentiment = general_sentiment*100
general_sentiment[:3]
```

Out[27]:

| brand | sentiment_pos | sentiment_neg | sentiment_neu |
|---|---|---|---|
| **Clairol** | 33.263833 | 33.426788 | 33.309379 |
| **Persil** | 33.441970 | 33.351215 | 33.206815 |
| **Purell** | 33.379555 | 33.275641 | 33.344804 |

In [28]:

```
# How much non-null rows this table has?
print(f'{round(len(general_sentiment)/len(products)*100, 1)}% of data from original.')
```

```
8.3% of data from original.
```

## Income Demographic

In [29]:

```
# Splitting the columns
maca = products[['brand','income_demogs']].dropna()

# LOoping through teh data
rates = {'brand':[],'income':[], 'wt':[]}
for keys, values in maca['income_demogs'].items():
    if len(values)>1:
        for black in values:
            gui = black.get('income')
            bui = str(gui).split(" to ")
            if len(bui)<=1:
                if str(bui) == "['$200,000 or more']":
                    rates['brand'].append(maca['brand'][keys])
                    rates['income'].append("200,000")
                    rates['wt'].append(black.get('wt')*1000000)
                else:
                    rates['brand'].append(maca['brand'][keys])
                    rates['income'].append("100,000")
                    rates['wt'].append(black.get('wt')*1000000)
            else:
                rates['brand'].append(maca['brand'][keys])
                rates['income'].append(bui[1].strip("$"))
                rates['wt'].append(black.get('wt')*1000000)

    else:
        pass

# Creating the dataframe
```

```
income = pd.DataFrame(rates)
income[:3]
```

Out[29]:

| | brand | income | wt |
|---|---|---|---|
| **0** | Clairol | 14,999 | 16.438901 |
| **1** | Clairol | 149,999 | 45.498425 |
| **2** | Clairol | 24,999 | 34.227204 |

In [30]:

```
# Creating data from income demog
income_demog = income.pivot_table(index=['brand'],
                                  columns=['income'],
                                  values=['wt']).reset_index()

income_demog.columns = income_demog.columns.droplevel([0])
income_demog = income_demog.rename(columns={'':'brand'})

# Getting just the profitable target
# Highest
income_demog['highest_income'] = income_demog[['100,000',
'14,999', '149,999', '199,999',
       '200,000', '24,999', '34,999', '49,999', '74,999',
       '99,999']].apply(lambda x: x.idxmax(axis=0), axis=1)
#Lowest
income_demog['lowest_income'] = income_demog[['100,000',
'14,999', '149,999', '199,999',
       '200,000', '24,999', '34,999', '49,999', '74,999',
       '99,999']].apply(lambda x: x.idxmin(axis=0), axis=1)


# To merg with other datas
inc = income_demog.loc[:,
['brand','highest_income','lowest_income']].copy()
inc[:5]
```

Out[30]:

| income | brand | highest_income | lowest_income |
|---|---|---|---|
| **0** | 7Up | 74,999 | 14,999 |
| **1** | 9Lives | 74,999 | 14,999 |
| **2** | A&W | 74,999 | 14,999 |
| **3** | ACT Oral Care | 74,999 | 14,999 |
| **4** | Absolut | 74,999 | 14,999 |

In [31]:

```
# Saving the data as csv
```

```
income_demog.to_csv("./20220914_Product_Income_Demo.csv")
```

## Race demogs

In [32]:

```
# Splitting the columns
cama = products[['brand','race_demogs']].dropna()

# LOoping through teh data
race = {'brand':[],'wt':[]}
for keys, values in cama['race_demogs'].items():
    if len(values)>1:
        for black in values:
            race['brand'].append(cama['brand'][keys])
            race['wt'].append(black.get('wt')*100)
    else:
        print(values)


# Setting the dataframe here
racing = pd.DataFrame(race)

# Checking the dataframe
racem = racing.groupby(['brand'])
['wt'].max().reset_index().rename(columns={'wt':'race_demo'})
racem[:5]
```

Out[32]:

| | brand | race_demo |
|---|---|---|
| 0 | 7Up | 83.210072 |
| 1 | 9Lives | 83.210072 |
| 2 | A&W | 83.210072 |
| 3 | ACT Oral Care | 83.210072 |
| 4 | Absolut | 83.210072 |

## Table 2 - Product Section

In [33]:

```
# Merging all together
SPCG_1 = pd.merge(customer_service_ratings, general_sentiment,
                  left_on='brand',
                  right_on ='brand',
                  how='outer')
SPCG_2 = pd.merge(SPCG_1, PRD,
```

```
                  left_on='brand',
                  right_on='brand',
                  how='outer')
SPCG_3 = pd.merge(SPCG_2, general_loyalty,
                  left_on='brand',
                  right_on='brand',
                  how='outer')
SPCG_4 = pd.merge(SPCG_3, inc,
                  left_on='brand',
                  right_on='brand',
                  how='outer')
SPCG_5 = pd.merge(SPCG_4, racem,
                  left_on='brand',
                  right_on='brand',
                  how='outer')
```

In [34]:

```
# Col by col filling the gaps by the mean
SPCG_5[1.0] = round(SPCG_5[1.0].fillna(SPCG_5[1.0].mean()),3)
SPCG_5[2.0] = round(SPCG_5[2.0].fillna(SPCG_5[2.0].mean()),3)
SPCG_5[3.0] = round(SPCG_5[3.0].fillna(SPCG_5[3.0].mean()),3)
SPCG_5[4.0] = round(SPCG_5[4.0].fillna(SPCG_5[4.0].mean()),3)
SPCG_5[5.0] = round(SPCG_5[5.0].fillna(SPCG_5[5.0].mean()),3)
SPCG_5['sentiment_pos'] =
round(SPCG_5['sentiment_pos'].fillna(SPCG_5['sentiment_pos'].mean()

SPCG_5['sentiment_neg'] =
round(SPCG_5['sentiment_neg'].fillna(SPCG_5['sentiment_neg'].mean()

SPCG_5['sentiment_neu'] =
round(SPCG_5['sentiment_neu'].fillna(SPCG_5['sentiment_neu'].mean()

SPCG_5['loyal'] =
round(SPCG_5['loyal'].fillna(SPCG_5['loyal'].mean()),3)
SPCG_5['disloyal'] =
round(SPCG_5['disloyal'].fillna(SPCG_5['disloyal'].mean()),3)
SPCG_5['neutral'] =
round(SPCG_5['neutral'].fillna(SPCG_5['neutral'].mean()),3)
SPCG_5['ratings'] = SPCG_5['ratings'].apply(lambda x:
round(SPCG_5['ratings'].mean(),1) if x == 0.0 else x)
SPCG_5['highest_income'] =
SPCG_5['highest_income'].fillna(SPCG_5['highest_income'].mode())
```

```
SPCG_5['lowest_income'] =
SPCG_5['lowest_income'].fillna(SPCG_5['lowest_income'].mode())
SPCG_5['race_demo'] =
round(SPCG_5['race_demo'].fillna(SPCG_5['race_demo'].mean()),3)

# Viewing
SPCG_5[:5]
```

Out[34]:

| | brand | 1.0 | 2.0 | 3.0 | 4.0 | 5.0 | sentiment_pos | sentiment_neg | sentiment_neu | ratings | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7Up | 26.0 | 6.0 | 0.0 | 13.0 | 56.0 | 33.341 | 33.249 | 33.410 | 3.9 | 3 |
| **1** | 9Lives | 7.0 | 8.0 | 8.0 | 14.0 | 62.0 | 33.345 | 33.237 | 33.418 | 3.9 | 3 |
| **2** | ACT Oral Care | 21.0 | 0.0 | 0.0 | 21.0 | 59.0 | 33.254 | 33.269 | 33.477 | 3.9 | 3 |
| **3** | Absolut | 2.0 | 2.0 | 4.0 | 4.0 | 89.0 | 33.353 | 33.370 | 33.277 | 3.9 | 3 |
| **4** | Adams (Peanut Butter) | 4.0 | 3.0 | 9.0 | 13.0 | 71.0 | 33.292 | 33.215 | 33.493 | 3.9 | 3 |

In [35]:
```
# Saving it as csv
SPCG_5.to_csv("./20221409_Product_Demo_Info.csv")
```

In [36]:
```
# How much non-null rows this table has?
print(f'{round(len(SPCG_3)/len(products)*100, 1)}% of data from
original.')
```

```
100.0% of data from original.
```

# Table 3. Stores

## Sales Per Location

> Sales Per Location - Number of sales listed in online resources per zip code (optional)

## Competitors

> Competitors - derived based on category or product type

## Price Comparison vs Competitors

> Price Comparison vs Competitors - Price comparison based on competing products

## Price

> Price - percentage price comparison (optional)

## Cost per Store

> Cost per store - Total cost per store by category

## Normal Price

> Normal Price - Normal price in the store (inventory price)

## Sale Price

> Sale Price - Sale Price of the store

## Sales per Location

In [37]:
```
# Setting a dict for stores data
storing = {'name':[], 'address':[], 'summary':[]}
for key, value in stores['store_info2'].items():
    if len(value)>1:
        storing['name'].append(value.get('name'))
        storing['address'].append(value.get('address'))
        storing['summary'].append(value.get('summary'))
    else:
        pass

# Transforming to dts
stores_info = pd.DataFrame(storing)

# Setting brand names in lowcase for better understanding
stores_info['name'] = stores_info['name'].str.lower()
```

In [66]:
```
# stores address and summary
str_add = pd.DataFrame.from_records(stores_info['address'])
str_sum = pd.DataFrame.from_records(stores_info['summary'])

# Lowercasing some columns
str_add['street'] = str_add['street'].str.lower()
str_add['city'] = str_add['city'].str.lower()
str_add['zip_code'] =
```

```
str_add['zip_code'].str.strip(",").str.strip(" ")

# Filling empty rows
str_add['zip_code'] =
pd.to_numeric(str_add['zip_code']).fillna(0.0).astype(int)
str_sum['priced_sales'] = str_sum['priced_sales'].fillna(0.0)

# All together!
store_sales = pd.concat([str_add,str_sum], axis=1)
store_sales.insert(0,'name_product',products['brand'])
store_sales.insert(0,'name_store', stores_info['name'])
store_sales.insert(0,'sales_count',PS_work['sales_count'])

#Correcting the col 'unpriced products'
store_sales['unpriced_sales'] = store_sales['sales'] -
store_sales['priced_sales']
```

In [67]:

```
# Sales per zip-code - just view ;)
zip_code_sales = store_sales.loc[:,
['name_product','name_store','zip_code','sales']].copy()
zip_code_sales = zip_code_sales.sort_values(by='name_store',
ascending=True)
zip_code_sales[:2]
```

Out[67]:

| | name_product | name_store | zip_code | sales |
|---|---|---|---|---|
| **2018** | model car | aldi development partners | 0 | 0.0 |
| **142** | Pace (Dips & Sauces) | aldi inc | 35206 | 0.0 |

In [68]:

```
# View table saved
store_sales[:2]
```

Out[68]:

| | sales_count | name_store | name_product | street | city | state | country | latitude | longi |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 111.0 | walgreens | Clairol | 1721 e parks hwy | wasilla | AK | USA | 61.577462 | -149.40 |
| **1** | NaN | walmart supercenter | Persil | 9248 parkway east | juneau | AK | USA | 58.3592 | -134.5 |

In [69]:

```
# Save as csv
store_sales.to_csv("./20220914_Store_Sales_Location.csv")
```

# Competitors

In [63]:

```
# Expanding the product_info col
# Setting a dict for products data
products_stores =
pd.DataFrame.from_records(stores['product_info'],
                                                columns=
['_id','name','description',

'picture','popular_products',

'categories','flag_hla',

'hla','standard_categories',

'categories_textform','sales_count',

'confidence'])

# Gettin' the columns to most important
PS = products_stores.loc[:,
['name','categories_textform','sales_count']].copy()

# Cleaning name col - changing the type
germen = PS['name'].apply(lambda x: x.split(",
")).apply(pd.Series)
PS['short_name'] = [TextBlob(x).noun_phrases[0] for x in
germen[0]]
PS.insert(3,'price', stores['price'])
PS['short_name'] = PS.short_name.str.title()

# Gonna work just with category, short_name and price
PS_work = PS.loc[:,
['categories_textform','short_name','price','sales_count']].copy()


jt = PS['categories_textform'].apply(lambda x: str(x).split(",
")).apply(pd.Series)
PS_work['categories_textform'] = jt[0].str.strip("['']")
```

In [65]:

```
# Save as csv
PS_work.to_csv("./20220914_PSWORK.csv")
```

In [44]:
```
# Getting Competitors by Category
Competitors = PS_work.groupby(['categories_textform'])
['short_name'].unique().to_frame().reset_index()
Competitors[15:20]
```

Out[44]:

| | categories_textform | short_name |
|---|---|---|
| **15** | COMPUTER & TECH ACCESSORIES | [Sandisk Ultra Dual] |
| **16** | CONDIMENTS | [Classic Balsamic Glaze, Original Habanero Eat... |
| **17** | COOKIES | [Pringles Potato Crisps Chips, Olay Ultimate E... |
| **18** | DAIRY | [Cheez-It Cheese Crackers, Olay, Pringles Pota... |
| **19** | DAIRY & EGGS | [Fit Strawberry Banana Nonfat Yogurt, Planet, ... |

In [45]:
```
# Save as csv
Competitors.to_csv("./20220914_Products_Store_Competition.csv")
```

# Price Comparison vs Competitors

In [46]:
```
# Getting Competitors by Category
price_compet =
PS_work.groupby(['categories_textform']).agg({'short_name':'unique'


'price': ['min','max']}).reset_index()
price_compet[15:20]
```

Out[46]:

| | categories_textform | short_name | price | |
|---|---|---|---|---|
| | | unique | min | max |
| **15** | COMPUTER & TECH ACCESSORIES | [Sandisk Ultra Dual] | 29.98 | 29.98 |
| **16** | CONDIMENTS | [Classic Balsamic Glaze, Original Habanero Eat... | 5.69 | 11.49 |
| **17** | COOKIES | [Pringles Potato Crisps Chips, Olay Ultimate E... | 1.29 | 275.98 |
| **18** | DAIRY | [Cheez-It Cheese Crackers, Olay, Pringles Pota... | 1.74 | 71.98 |
| **19** | DAIRY & EGGS | [Fit Strawberry Banana Nonfat Yogurt, Planet, ... | 1.50 | 5.49 |

In [47]:
```
# save as csv
price_compet.to_csv("./20220914_Store_Price_Comparison.csv")
```

# Price

In [48]:
```
# Dropping one level
price_compet.columns = price_compet.columns.droplevel(0)
price_compet = price_compet.rename(columns={'':'category'})
# Getting Competitors by Category
price_compet['%_diff_price'] = round((price_compet['max']-
price_compet['min'])/price_compet['max']*100,1)

# View
price_compet[:2]
```

Out[48]:

| | category | unique | min | max | %_diff_price |
|---|---|---|---|---|---|
| **0** | | [Cheez-It Cheese Crackers, Eggo Thick, Kellogg... | 2.44 | 33.99 | 92.8 |
| **1** | BABY ITEMS | [Olay, Olay Lotion Moisturizer, Fixodent Dentu... | 2.59 | 40.79 | 93.7 |

In [49]:
```
# save as csv
price_compet.to_csv("./20220914_Store_Price_Difference.csv")
```

## Cost per Store, Normal Price and Sale Price

In [72]:
```
# Concating price and store sales dataframe
store_raw = stores.loc[:,['start_date','end_date',

'is_actual_price','price','inventory_price']].copy()



sale_store = pd.concat([store_sales,store_raw],axis=1)
```

In [51]:
```
# Creating a new col called - total profit_sales per store
sale_store['profit_sales'] =
sale_store['priced_sales']*sale_store['price']

# Grouping store per sum of price ;)
store_profit = sale_store.groupby(['name_store'])
['profit_sales'].sum().to_frame().reset_index()
store_profit[store_profit['profit_sales']>0][:3]
```

Out[51]:

| | name_store | profit_sales |
|---|---|---|
| **43** | cook county whole foods co-op | 113589.27 |
| **150** | gnld whole food supplemen | 72467.46 |
| **155** | healthy living with whole food | 92848.41 |

In [74]:
```

```

```
# Saving sale_store and store_profit
sale_store.to_csv("./20220914_Store_Sale_Normal_price.csv")
store_profit.to_csv("./20220914_Store_Sales_profit_sales.csv")
```

End of Script ;)