Amazon Apparel Recommendations

[4.2] Data and Code:

https://drive.google.com/open?id=0BwNkduBnePt2VWhCYXhMV3p4dTg

[4.3] Overview of the data

In [1]:

```
#import all the necessary packages.

from PIL import Image
import requests
from io import BytesIO
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import warnings
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import math
import time
import re
import os
import seaborn as sns
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import pairwise_distances
from matplotlib import gridspec
from scipy.sparse import hstack
import plotly
import plotly.figure_factory as ff
from plotly.graph_objs import Scatter, Layout

import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dropout, Flatten, Dense
from keras import applications
from sklearn.metrics import pairwise_distances
import matplotlib.pyplot as plt
import requests
from PIL import Image
import pandas as pd
import pickle

plotly.offline.init_notebook_mode(connected=True)
warnings.filterwarnings("ignore")
```

```
Using TensorFlow backend.
```

In [2]:

```
# we have give a json file which consists of all information about
# the products
# loading the data using pandas' read_json file.
data = pd.read_json('tops_fashion.json')
```

In [3]:

```
print ('Number of data points : ', data.shape[0], \
```

```
        'Number of features/variables:', data.shape[1])
```

```
Number of data points :  183138 Number of features/variables: 19
```

Terminology:

What is a dataset?
Rows and columns
Data-point
Feature/variable

In [4]:

```
# each product/item has 19 features in the raw dataset.
data.columns # prints column-names or feature-names.
```

Out[4]:

```
Index(['asin', 'author', 'availability', 'availability_type', 'brand', 'color',
       'editorial_reivew', 'editorial_review', 'formatted_price',
       'large_image_url', 'manufacturer', 'medium_image_url', 'model',
       'product_type_name', 'publisher', 'reviews', 'sku', 'small_image_url',
       'title'],
      dtype='object')
```

Of these 19 features, we will be using only 6 features in this workshop.

1. asin (Amazon standard identification number)
2. brand (brand to which the product belongs to)
3. color (Color information of apparel, it can contain many colors as a value ex: red an
d black stripes)
4. product_type_name (type of the apperal, ex: SHIRT/TSHIRT)
5. medium_image_url (url of the image)
6. title (title of the product.)
7. formatted_price (price of the product)

In [5]:

```
data = data[['asin', 'brand', 'color', 'medium_image_url', 'product_type_name', 'title', 'formatted
_price']]
```

In [6]:

```
print ('Number of data points : ', data.shape[0], \
       'Number of features:', data.shape[1])
data.head() # prints the top rows in the table.
```

```
Number of data points :  183138 Number of features: 7
```

Out[6]:

	asin	brand	color	medium_image_url	product_type_name	title	formatted_price
0	B016I2TS4W	FNC7C	None	https://images-na.ssl-images-amazon.com/images...	SHIRT	Minions Como Superheroes Ironman Long Sleeve R...	None
1	B01N49AI08	FIG Clothing	None	https://images-na.ssl-images-amazon.com/images...	SHIRT	FIG Clothing Womens Izo Tunic	None
2	B01JDPCOHO	FIG Clothing	None	https://images-na.ssl-images-amazon.com/images...	SHIRT	FIG Clothing Womens Won Top	None
3	B01N19U5H5	Focal18	None	https://images-na.ssl-images-amazon.com/images...	SHIRT	Focal18 Sailor Collar Bubble Sleeve Blouse Shi...	None
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT	Featherlite Ladies' Long Sleeve Stain Resistan...	$26.26

[5.1] Missing data for various features.

Basic stats for the feature: product_type_name

In [7]:

```
# We have total 72 unique type of product_type_names
print(data['product_type_name'].describe())

# 91.62% (167794/183138) of the products are shirts,
```

```
count     183138
unique        72
top         SHIRT
freq      167794
Name: product_type_name, dtype: object
```

In [8]:

```
# names of different product types
print(data['product_type_name'].unique())
```

```
['SHIRT' 'SWEATER' 'APPAREL' 'OUTDOOR_RECREATION_PRODUCT'
 'BOOKS_1973_AND_LATER' 'PANTS' 'HAT' 'SPORTING_GOODS' 'DRESS' 'UNDERWEAR'
 'SKIRT' 'OUTERWEAR' 'BRA' 'ACCESSORY' 'ART_SUPPLIES' 'SLEEPWEAR'
 'ORCA_SHIRT' 'HANDBAG' 'PET_SUPPLIES' 'SHOES' 'KITCHEN' 'ADULT_COSTUME'
 'HOME_BED_AND_BATH' 'MISC_OTHER' 'BLAZER' 'HEALTH_PERSONAL_CARE'
 'TOYS_AND_GAMES' 'SWIMWEAR' 'CONSUMER_ELECTRONICS' 'SHORTS' 'HOME'
 'AUTO_PART' 'OFFICE_PRODUCTS' 'ETHNIC_WEAR' 'BEAUTY'
 'INSTRUMENT_PARTS_AND_ACCESSORIES' 'POWERSPORTS_PROTECTIVE_GEAR' 'SHIRTS'
 'ABIS_APPAREL' 'AUTO_ACCESSORY' 'NONAPPARELMISC' 'TOOLS' 'BABY_PRODUCT'
 'SOCKSHOSIERY' 'POWERSPORTS_RIDING_SHIRT' 'EYEWEAR' 'SUIT'
 'OUTDOOR_LIVING' 'POWERSPORTS_RIDING_JACKET' 'HARDWARE' 'SAFETY_SUPPLY'
 'ABIS_DVD' 'VIDEO_DVD' 'GOLF_CLUB' 'MUSIC_POPULAR_VINYL'
 'HOME_FURNITURE_AND_DECOR' 'TABLET_COMPUTER' 'GUILD_ACCESSORIES'
 'ABIS_SPORTS' 'ART_AND_CRAFT_SUPPLY' 'BAG' 'MECHANICAL_COMPONENTS'
 'SOUND_AND_RECORDING_EQUIPMENT' 'COMPUTER_COMPONENT' 'JEWELRY'
 'BUILDING_MATERIAL' 'LUGGAGE' 'BABY_COSTUME' 'POWERSPORTS_VEHICLE_PART'
 'PROFESSIONAL_HEALTHCARE' 'SEEDS_AND_PLANTS' 'WIRELESS_ACCESSORY']
```

In [9]:

```
# find the 10 most frequent product_type_names.
product_type_count = Counter(list(data['product_type_name']))
product_type_count.most_common(10)
```

Out[9]:

```
[('SHIRT', 167794),
 ('APPAREL', 3549),
 ('BOOKS_1973_AND_LATER', 3336),
 ('DRESS', 1584),
 ('SPORTING_GOODS', 1281),
 ('SWEATER', 837),
 ('OUTERWEAR', 796),
 ('OUTDOOR_RECREATION_PRODUCT', 729),
 ('ACCESSORY', 636),
 ('UNDERWEAR', 425)]
```

Basic stats for the feature: brand

In [10]:

```
# there are 10577 unique brands
print(data['brand'].describe())

# 183138 - 182987 = 151 missing values.
```

```
count     182987
unique     10577
top         Zago
freq          223
Name: brand, dtype: object
```

In [11]:

```
brand_count = Counter(list(data['brand']))
brand_count.most_common(10)
```

Out[11]:

```
[('Zago', 223),
 ('XQS', 222),
 ('Yayun', 215),
 ('YUNY', 198),
 ('XiaoTianXin-women clothes', 193),
 ('Generic', 192),
 ('Boohoo', 190),
 ('Alion', 188),
 ('Abetteric', 187),
 ('TheMogan', 187)]
```

Basic stats for the feature: color

In [12]:

```
print(data['color'].describe())


# we have 7380 unique colors
# 7.2% of products are black in color
# 64956 of 183138 products have brand information. That's approx 35.4%.
```

```
count     64956
unique     7380
top       Black
freq      13207
Name: color, dtype: object
```

In [13]:

```
color_count = Counter(list(data['color']))
color_count.most_common(10)
```

Out[13]:

```
[(None, 118182),
 ('Black', 13207),
 ('White', 8616),
 ('Blue', 3570),
 ('Red', 2289),
 ('Pink', 1842),
 ('Grey', 1499),
 ('*', 1388),
 ('Green', 1258),
 ('Multi', 1203)]
```

Basic stats for the feature: formatted_price

In [14]:

```

print(data['formatted_price'].describe())

# Only 28,395 (15.5% of whole data) products with price information
```

```
count      28395
unique      3135
top       $19.99
freq          945
Name: formatted_price, dtype: object
```

In [15]:

```
price_count = Counter(list(data['formatted_price']))
price_count.most_common(10)
```

Out[15]:

```
[(None, 154743),
 ('$19.99', 945),
 ('$9.99', 749),
 ('$9.50', 601),
 ('$14.99', 472),
 ('$7.50', 463),
 ('$24.99', 414),
 ('$29.99', 370),
 ('$8.99', 343),
 ('$9.01', 336)]
```

Basic stats for the feature: title

In [17]:

```
print(data['title'].describe())

# All of the products have a title.
# Titles are fairly descriptive of what the product is.
# We use titles extensively in this workshop
# as they are short and informative.
```

```
count                                                183138
unique                                               175985
top       Nakoda Cotton Self Print Straight Kurti For Women
freq                                                     77
Name: title, dtype: object
```

In [18]:

```
data.to_pickle('pickels/180k_apparel_data')
```

We save data files at every major step in our processing in "pickle" files. If you are stuck anywhere (or) if some code takes too long to run on your laptop, you may use the pickle files we give you to speed things up.

In [19]:

```
# consider products which have price information
# data['formatted_price'].isnull() => gives the information
#about the dataframe row's which have null values price == None|Null
data = data.loc[~data['formatted_price'].isnull()]
print('Number of data points After eliminating price=NULL :', data.shape[0])
```

```
Number of data points After eliminating price=NULL : 28395
```

In [20]:

```
# consider products which have color information
# data['color'].isnull() => gives the information about the dataframe row's which have null values
price == None|Null
data =data.loc[~data['color'].isnull()]
print('Number of data points After eliminating color=NULL :', data.shape[0])
```

```
Number of data points After eliminating color=NULL : 28385
```

We brought down the number of data points from 183K to 28K.

We are processing only 28K points so that most of the workshop participants can run this code on thier laptops in a reasonable amount of time.

For those of you who have powerful computers and some time to spare, you are recommended to use all of the 183K images.

In [21]:

```
data.to_pickle('pickels/28k_apparel_data')
```

In [0]:

```
# You can download all these 28k images using this code below.
# You do NOT need to run this code and hence it is commented.


'''
from PIL import Image
import requests
from io import BytesIO

for index, row in images.iterrows():
        url = row['large_image_url']
        response = requests.get(url)
        img = Image.open(BytesIO(response.content))
        img.save('images/28k_images/'+row['asin']+'.jpeg')


'''
```

Out[0]:

```
"\nfrom PIL import Image\nimport requests\nfrom io import BytesIO\n\nfor index, row in
images.iterrows():\n        url = row['large_image_url']\n        response = requests.get(url)\n
img = Image.open(BytesIO(response.content))\n
img.save('workshop/images/28k_images/'+row['asin']+'.jpeg')\n\n\n"
```

[5.2] Remove near duplicate items

[5.2.1] Understand about duplicates.

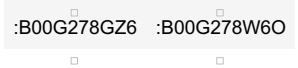
In [22]:

```
# read data from pickle file from previous stage
data = pd.read_pickle('pickels/28k_apparel_data')

# find number of products that have duplicate titles.
print(sum(data.duplicated('title')))
# we have 2325 products which have same title but different color
```

```
2325
```

These shirts are exactly same except in size (S, M,L,XL)

These shirts exactly same except in color



:B00G278Z2A :B00G2786X8

In our data there are many duplicate products like the above examples, we need to de-dupe them for better results.

[5.2.2] Remove duplicates : Part 1

In [23]:

```
# read data from pickle file from previous stage
data = pd.read_pickle('pickels/28k_apparel_data')
```

In [24]:

```
data.head()
```

Out[24]:

	asin	brand	color	medium_image_url	product_type_name	title	formatted_price
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT	Featherlite Ladies' Long Sleeve Stain Resistan...	$26.26
6	B012YX2ZPI	HX-Kingdom Fashion T-shirts	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	Women's Unique 100% Cotton T - Special Olympic...	$9.99
11	B001LOUGE4	Fitness Etc.	Black	https://images-na.ssl-images-amazon.com/images...	SHIRT	Ladies Cotton Tank 2x1 Ribbed Tank Top	$11.99
15	B003BSRPB0	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	FeatherLite Ladies' Moisture Free Mesh Sport S...	$20.54
21	B014ICEDNA	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT	Supernatural Chibis Sam Dean And Castiel Short...	$7.50

In [25]:

```
# Remove All products with very few words in title
data_sorted = data[data['title'].apply(lambda x: len(x.split())>4)]
print("After removal of products with short description:", data_sorted.shape[0])
```

```
After removal of products with short description: 27949
```

In [26]:

```
# Sort the whole data based on title (alphabetical order of title)
data_sorted.sort_values('title',inplace=True, ascending=False)
data_sorted.head()
```

Out[26]:

	asin	brand	color	medium_image_url	product_type_name	title	formatted_price
61973	B06Y1KZ2WB	Éclair	Black/Pink	https://images-na.ssl-images-amazon.com/images...	SHIRT	Éclair Women's Printed Thin Strap Blouse Black...	$24.99
133820	B010RV33VE	xiaoming	Pink	https://images-na.ssl-images-amazon.com/images...	SHIRT	xiaoming Womens Sleeveless Loose Long T-shirts...	$18.19
81461	B01DDSDLNS	xiaoming	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	xiaoming Women's White Long Sleeve Single Brea...	$21.58
75995	B00X5LYO9Y	xiaoming	Red Anchors	https://images-na.ssl-images-amazon.com/images...	SHIRT	xiaoming Stripes Tank Patch/Bear Sleeve Anchor...	$15.91
151570	B00WPJG35K	xiaoming	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	xiaoming Sleeve Sheer Loose Tassel Kimono Woma...	$14.32

Some examples of dupliacte titles that differ only in the last few words.

```
Titles 1:
16. woman's place is in the house and the senate shirts for Womens XXL White
17. woman's place is in the house and the senate shirts for Womens M Grey

Title 2:
25. tokidoki The Queen of Diamonds Women's Shirt X-Large
26. tokidoki The Queen of Diamonds Women's Shirt Small
27. tokidoki The Queen of Diamonds Women's Shirt Large

Title 3:
61. psychedelic colorful Howling Galaxy Wolf T-shirt/Colorful Rainbow Animal Print Head
Shirt for woman Neon Wolf t-shirt
62. psychedelic colorful Howling Galaxy Wolf T-shirt/Colorful Rainbow Animal Print Head
Shirt for woman Neon Wolf t-shirt
63. psychedelic colorful Howling Galaxy Wolf T-shirt/Colorful Rainbow Animal Print Head
Shirt for woman Neon Wolf t-shirt
64. psychedelic colorful Howling Galaxy Wolf T-shirt/Colorful Rainbow Animal Print Head
Shirt for woman Neon Wolf t-shirt
```

In [29]:

```
indices = []
for i,row in data_sorted.iterrows():
    indices.append(i)
```

In [30]:

```
import itertools
stage1_dedupe_asins = []
i = 0
j = 0
num_data_points = data_sorted.shape[0]
while i < num_data_points and j < num_data_points:

    previous_i = i

    # store the list of words of ith string in a, ex: a = ['tokidoki', 'The', 'Queen', 'of', 'Diam
onds', 'Women's', 'Shirt', 'X-Large']
    a = data['title'].loc[indices[i]].split()

    # search for the similar products sequentially
    j = i+1
    while j < num_data_points:

        # store the list of words of jth string in b, ex: b = ['tokidoki', 'The', 'Queen', 'of', '
Diamonds', 'Women's', 'Shirt', 'Small']
        b = data['title'].loc[indices[j]].split()

        # store the maximum length of two strings
        length = max(len(a), len(b))

        # count is used to store the number of words that are matched in both strings
        count  = 0

        # itertools.zip_longest(a,b): will map the corresponding words in both strings, it will
appened None in case of unequal strings
        # example: a =['a', 'b', 'c', 'd']
        # b = ['a', 'b', 'd']
        # itertools.zip_longest(a,b): will give [('a','a'), ('b','b'), ('c','d'), ('d', None)]
        for k in itertools.zip_longest(a,b):
            if (k[0] == k[1]):
                count += 1

        # if the number of words in which both strings differ are > 2 , we are considering it as t
hose two apperals are different
        # if the number of words in which both strings differ are < 2 , we are considering it as t
```

```
nose two apperals are same, hence we are ignoring them
        if (length - count) > 2: # number of words in which both sensences differ
            # if both strings are differ by more than 2 words we include the 1st string index
            stage1_dedupe_asins.append(data_sorted['asin'].loc[indices[i]])


            # start searching for similar apperals corresponds 2nd string
            i = j
            break
        else:
            j += 1
    if previous_i == i:
        break
```

In [31]:

```
data = data.loc[data['asin'].isin(stage1_dedupe_asins)]
```

We removed the dupliactes which differ only at the end.

In [32]:

```
print('Number of data points : ', data.shape[0])
```

```
Number of data points :  17592
```

In [33]:

```
data.to_pickle('pickels/17k_apperal_data')
```

[5.2.3] Remove duplicates : Part 2

In the previous cell, we sorted whole data in alphabetical order of titles.Then, we removed titles which are adjacent and very similar title

But there are some products whose titles are not adjacent but very similar.

Examples:

Titles-1
86261. UltraClub Women's Classic Wrinkle-Free Long Sleeve Oxford Shirt, Pink, XX-Large
115042. UltraClub Ladies Classic Wrinkle-Free Long-Sleeve Oxford Light Blue XXL

TItles-2
75004. EVALY Women's Cool University Of UTAH 3/4 Sleeve Raglan Tee
109225. EVALY Women's Unique University Of UTAH 3/4 Sleeve Raglan Tees
120832. EVALY Women's New University Of UTAH 3/4-Sleeve Raglan Tshirt

In [34]:

```
data = pd.read_pickle('pickels/17k_apperal_data')
```

In [35]:

```
# This code snippet takes significant amount of time.
# O(n^2) time.
# Takes about an hour to run on a decent computer.

indices = []
for i,row in data.iterrows():
    indices.append(i)
```

```
stage2_dedupe_asins = []
while len(indices)!=0:
    i = indices.pop()
    stage2_dedupe_asins.append(data['asin'].loc[i])
    # consider the first apperal's title
    a = data['title'].loc[i].split()
    # store the list of words of ith string in a, ex: a = ['tokidoki', 'The', 'Queen', 'of', 'Diam
onds', 'Women's', 'Shirt', 'X-Large']
    for j in indices:

        b = data['title'].loc[j].split()
        # store the list of words of jth string in b, ex: b = ['tokidoki', 'The', 'Queen', 'of', '
Diamonds', 'Women's', 'Shirt', 'X-Large']

        length = max(len(a),len(b))

        # count is used to store the number of words that are matched in both strings
        count  = 0

        # itertools.zip_longest(a,b): will map the corresponding words in both strings, it will
appened None in case of unequal strings
        # example: a =['a', 'b', 'c', 'd']
        # b = ['a', 'b', 'd']
        # itertools.zip_longest(a,b): will give [('a','a'), ('b','b'), ('c','d'), ('d', None)]
        for k in itertools.zip_longest(a,b):
            if (k[0]==k[1]):
                count += 1

        # if the number of words in which both strings differ are < 3 , we are considering it as t
hose two apperals are same, hence we are ignoring them
        if (length - count) < 3:
            indices.remove(j)
```

In [36]:

```
# from whole previous products we will consider only
# the products that are found in previous cell
data = data.loc[data['asin'].isin(stage2_dedupe_asins)]
```

In [37]:

```
print('Number of data points after stage two of dedupe: ',data.shape[0])
# from 17k apperals we reduced to 16k apperals
```

```
Number of data points after stage two of dedupe:  16434
```

In [46]:

```
data.to_pickle('pickels/16k_apperal_data')
# Storing these products in a pickle file
# candidates who wants to download these files instead
# of 180K they can download and use them from the Google Drive folder.
```

6. Text pre-processing

In [38]:

```
data = pd.read_pickle('pickels/16k_apperal_data')

# NLTK download stop words. [RUN ONLY ONCE]
# goto Terminal (Linux/Mac) or Command-Prompt (Window)
# In the temrinal, type these commands
# $python3
# $import nltk
# $nltk.download()
```

In [39]:

```
# we use the list of stop words that are downloaded from nltk lib.
stop_words = set(stopwords.words('english'))
```

```
stop_words = set(stopwords.words('english'))
print ('list of stop words:', stop_words)

def nlp_preprocessing(total_text, index, column):
    if type(total_text) is not int:
        string = ""
        for words in total_text.split():
            # remove the special chars in review like '"#$@!%^&*()_+-~?>< etc.
            word = ("".join(e for e in words if e.isalnum()))
            # Conver all letters to lower-case
            word = word.lower()
            # stop-word removal
            if not word in stop_words:
                string += word + " "
        data[column][index] = string
```

```
list of stop words: {'had', 'my', 'before', 'these', 'does', 'there', 'myself', "don't", "didn't",
'this', 'having', 'too', 'me', "hasn't", 'its', "you'd", 'again', 'if', 'shouldn', 're', 'doing',
"hadn't", 'will', 'in', 'because', 'aren', 'no', 'don', "mightn't", "couldn't", 'than', 'here', 'w
hile', 'didn', 'why', "she's", 'is', "wouldn't", 'mightn', 'hadn', 'do', "doesn't", 'll', 'can', '
once', 'be', 'more', 'their', 'yourself', 'both', 'isn', 'into', 'nor', "shan't", 'few',
"should've", 'himself', 'i', 'an', 'o', "that'll", 'and', 've', 'ma', 'until', 'she', 'theirs', 'y
ours', 'yourselves', 'was', 'who', 'has', 'it', "weren't", 'but', 'just', "isn't", 'him', 'about',
'down', 'wasn', 'herself', "you're", 'whom', 'on', 'that', 'have', 'at', 'being', 'for', 'he',
'your', 'were', 'against', 'so', 'are', 'ourselves', 'won', 'should', 'the', 'when', 'shan',
'through', 'own', 'from', 'ain', 'couldn', 'y', 'itself', 'those', 'or', "it's", 'did', 'am', 'und
er', 'a', 'his', "aren't", 'which', 'by', 'over', 'other', 'what', 'mustn', 'our', 'm', 'during',
'them', "needn't", 'd', 't', 'out', 'of', 'weren', 'some', 'to', "you'll", 'you', 'between',
'now', 'with', 'hers', 'themselves', 'very', 'wouldn', 'each', 'we', 'hasn', 'only', 'after',
'below', 'where', 'not', 'haven', "haven't", 'then', 's', 'all', 'same', 'they', 'how', 'ours', 'u
p', 'off', "wasn't", 'any', 'her', 'most', "mustn't", 'further', 'as', 'above', "won't", 'such', '
needn', 'been', 'doesn', "shouldn't", "you've"}
```

In [45]:

```
start_time = time.clock()
# we take each title and we text-preprocess it.
for index, row in data.iterrows():
    nlp_preprocessing(row['title'], index, 'title')
# we print the time it took to preprocess whole titles
print(time.clock() - start_time, "seconds")
```

```
8.752180300001783 seconds
```

In [40]:

```
data.head()
```

Out[40]:

	asin	brand	color	medium_image_url	product_type_name	title	formatted_price
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT	Featherlite Ladies' Long Sleeve Stain Resistan...	$26.26
6	B012YX2ZPI	HX-Kingdom Fashion T-shirts	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	Women's Unique 100% Cotton T - Special Olympic...	$9.99
15	B003BSRPB0	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	FeatherLite Ladies' Moisture Free Mesh Sport S...	$20.54
27	B014ICEJ1Q	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT	Supernatural Chibis Sam Dean And Castiel O Nec...	$7.39
46	B01NACPBG2	Fifth Degree	Black	https://images-na.ssl-images-amazon.com/images...	SHIRT	Fifth Degree Womens Gold Foil Graphic Tees Jun...	$6.95

In [41]:

```
data.to_pickle('pickels/16k_apperal_data_preprocessed')
```

Stemming

In [38]:

```
from nltk.stem.porter import *
stemmer = PorterStemmer()
print(stemmer.stem('arguing'))
print(stemmer.stem('fishing'))


# We tried using stemming on our titles and it didnot work very well.
```

```
argu
fish
```

[8] Text based product similarity

In [46]:

```
data = pd.read_pickle('pickels/16k_apperal_data_preprocessed')
data.head()
```

Out[46]:

	asin	brand	color	medium_image_url	product_type_name	title	formatted_price
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT	featherlite ladies long sleeve stain resistant...	$26.26
6	B012YX2ZPI	HX-Kingdom Fashion T-shirts	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	womens unique 100 cotton special olympics wor...	$9.99
15	B003BSRPB0	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	featherlite ladies moisture free mesh sport sh...	$20.54
27	B014ICEJ1Q	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT	supernatural chibis sam dean castiel neck tshi...	$7.39
46	B01NACPBG2	Fifth Degree	Black	https://images-na.ssl-images-amazon.com/images...	SHIRT	fifth degree womens gold foil graphic tees jun...	$6.95

In [47]:

```
# Utility Functions which we will use through the rest of the workshop.


#Display an image
def display_img(url,ax,fig):
    # we get the url of the apparel and download it
    response = requests.get(url)
    img = Image.open(BytesIO(response.content))
    # we will display it in notebook
    plt.imshow(img)

#plotting code to understand the algorithm's decision.
def plot_heatmap(keys, values, labels, url, text):
        # keys: list of words of recommended title
        # values: len(values) ==  len(keys), values(i) represents the occurence of the word
keys(i)
        # labels: len(labels) == len(keys), the values of labels depends on the model we are using
                # if model == 'bag of words': labels(i) = values(i)
                # if model == 'tfidf weighted bag of words':labels(i) = tfidf(keys(i))
                # if model == 'idf weighted bag of words':labels(i) = idf(keys(i))
        # url : apparel's url

        # we will devide the whole figure into two parts
        gs = gridspec.GridSpec(2, 2, width_ratios=[4,1], height_ratios=[4,1])
        fig = plt.figure(figsize=(25,3))
```

```
        fig = plt.figure(figsize=(25,5))

        # 1st, ploting heat map that represents the count of commonly ocurred words in title2
        ax = plt.subplot(gs[0])
        # it displays a cell in white color if the word is intersection(lis of words of title1 and
list of words of title2), in black if not
        ax = sns.heatmap(np.array([values]), annot=np.array([labels]))
        ax.set_xticklabels(keys) # set that axis labels as the words of title
        ax.set_title(text) # apparel title

        # 2nd, plotting image of the the apparel
        ax = plt.subplot(gs[1])
        # we don't want any grid lines for image and no labels on x-axis and y-axis
        ax.grid(False)
        ax.set_xticks([])
        ax.set_yticks([])

        # we call dispaly_img based with paramete url
        display_img(url, ax, fig)

        # displays combine figure ( heat map and image together)
        plt.show()

def plot_heatmap_image(doc_id, vec1, vec2, url, text, model):

    # doc_id : index of the title1
    # vec1 : input apparels's vector, it is of a dict type {word:count}
    # vec2 : recommended apparels's vector, it is of a dict type {word:count}
    # url : apparels image url
    # text: title of recomonded apparel (used to keep title of image)
    # model, it can be any of the models,
        # 1. bag_of_words
        # 2. tfidf
        # 3. idf

    # we find the common words in both titles, because these only words contribute to the distance
between two title vec's
    intersection = set(vec1.keys()) & set(vec2.keys())

    # we set the values of non intersecting words to zero, this is just to show the difference in
heatmap
    for i in vec2:
        if i not in intersection:
            vec2[i]=0

    # for labeling heatmap, keys contains list of all words in title2
    keys = list(vec2.keys())
    #  if ith word in intersection(lis of words of title1 and list of words of title2):
values(i)=count of that word in title2 else values(i)=0
    values = [vec2[x] for x in vec2.keys()]

    # labels: len(labels) == len(keys), the values of labels depends on the model we are using
        # if model == 'bag of words': labels(i) = values(i)
        # if model == 'tfidf weighted bag of words':labels(i) = tfidf(keys(i))
        # if model == 'idf weighted bag of words':labels(i) = idf(keys(i))

    if model == 'bag_of_words':
        labels = values
    elif model == 'tfidf':
        labels = []
        for x in vec2.keys():
            # tfidf_title_vectorizer.vocabulary_ it contains all the words in the corpus
            # tfidf_title_features[doc_id, index_of_word_in_corpus] will give the tfidf value of w
rd in given document (doc_id)
            if x in  tfidf_title_vectorizer.vocabulary_:
                labels.append(tfidf_title_features[doc_id, tfidf_title_vectorizer.vocabulary_[x]])
            else:
                labels.append(0)
    elif model == 'idf':
        labels = []
        for x in vec2.keys():
            # idf_title_vectorizer.vocabulary_ it contains all the words in the corpus
            # idf_title_features[doc_id, index_of_word_in_corpus] will give the idf value of word
in given document (doc_id)
            if x in  idf_title_vectorizer.vocabulary_:
                labels.append(idf_title_features[doc_id, idf_title_vectorizer.vocabulary_[x]])
            else:
                labels.append(0)
```

```
                labels.append(0)

    plot_heatmap(keys, values, labels, url, text)


# this function gets a list of wrods along with the frequency of each
# word given "text"
def text_to_vector(text):
    word = re.compile(r'\w+')
    words = word.findall(text)
    # words stores list of all words in given string, you can try 'words = text.split()' this will
also gives same result
    return Counter(words) # Counter counts the occurence of each word in list, it returns dict
type object {word1:count}



def get_result(doc_id, content_a, content_b, url, model):
    text1 = content_a
    text2 = content_b

    # vector1 = dict{word11:#count, word12:#count, etc.}
    vector1 = text_to_vector(text1)

    # vector1 = dict{word21:#count, word22:#count, etc.}
    vector2 = text_to_vector(text2)

    plot_heatmap_image(doc_id, vector1, vector2, url, text2, model)
```

[8.2] Bag of Words (BoW) on product titles.

In [48]:

```
from sklearn.feature_extraction.text import CountVectorizer
title_vectorizer = CountVectorizer()
title_features   = title_vectorizer.fit_transform(data['title'])
title_features.get_shape() # get number of rows and columns in feature matrix.
# title_features.shape = #data_points * #words_in_corpus
# CountVectorizer().fit_transform(corpus) returns
# the a sparase matrix of dimensions #data_points * #words_in_corpus

# What is a sparse vector?

# title_features[doc_id, index_of_word_in_corpus] = number of times the word occured in that doc
```

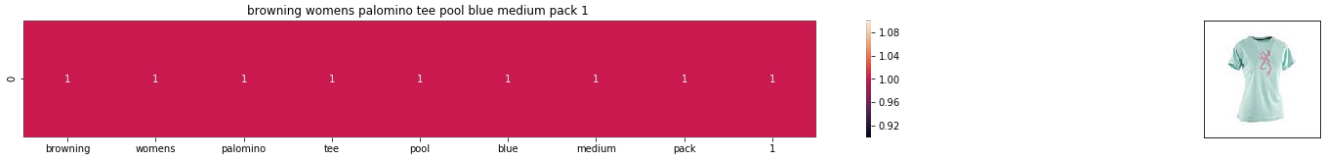
Out[48]:

```
(16042, 12609)
```

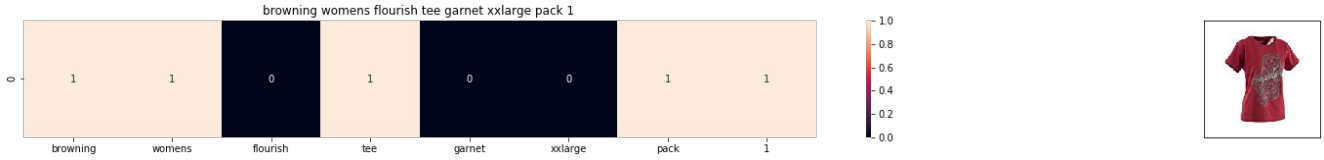
In [49]:

```
def bag_of_words_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

    # pairwise_dist will store the distance from given input apparel to all remaining apparels
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y) = <X, Y> / (||X
||*||Y||)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(title_features,title_features[doc_id])

    # np.argsort will return indices of the smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the smallest distances
    pdists  = np.sort(pairwise_dist.flatten())[0:num_results]

    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])

    for i in range(0,len(indices)):
        # we will pass 1. doc_id, 2. title1, 3. title2, url, model
        get_result(indices[i],data['title'].loc[df_indices[0]], data['title'].loc[df_indices[i]], d
ata['medium_image_url'].loc[df_indices[i]], 'bag_of_words')
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print ('Brand:', data['brand'].loc[df_indices[i]])
```

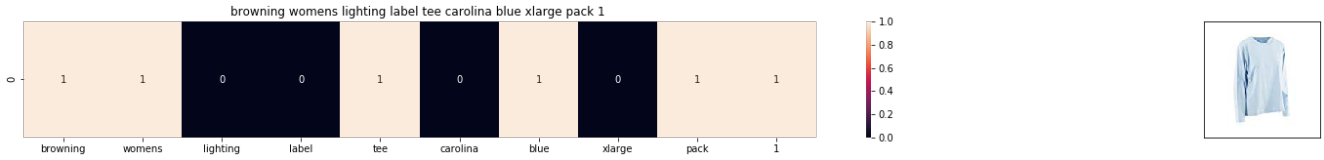
```
        print ('Title:', data['title'].loc[df_indices[i]])
        print ('Euclidean similarity with the query image :', pdists[i])
        print('='*60)

#call the bag-of-words model for a product to get similar products.
bag_of_words_model(12000, 20) # change the index if you want to.
# In the output heat map each value represents the count value
# of the label word, the color represents the intersection
# with inputs title.

#try 12566
#try 931
```



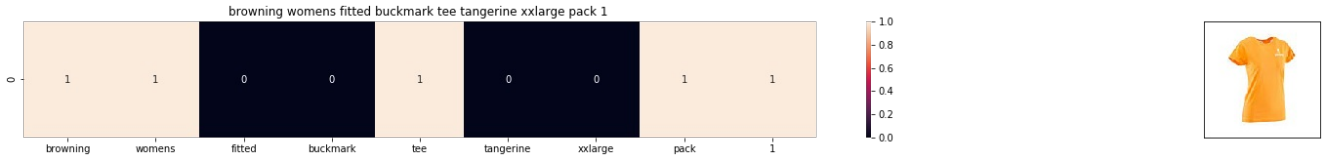
ASIN : B074KQ3SY1
Brand: Browning
Title: browning womens palomino tee pool blue medium pack 1
Euclidean similarity with the query image : 0.0
==



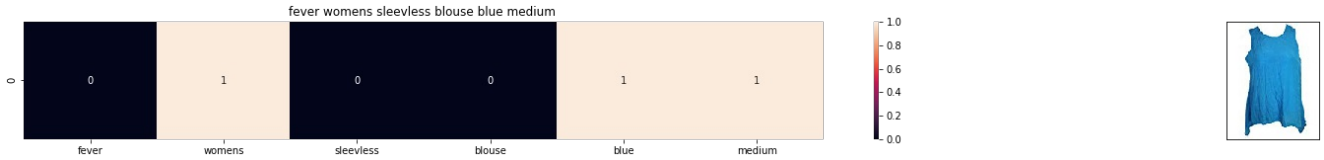
ASIN : B01IL3NF40
Brand: Browning
Title: browning womens flourish tee garnet xxlarge pack 1
Euclidean similarity with the query image : 2.6457513110645907
==



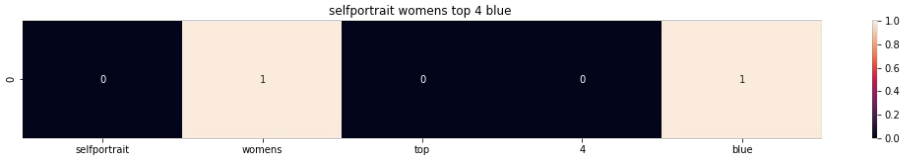
ASIN : B074KQJ8KC
Brand: Browning
Title: browning womens lighting label tee carolina blue xlarge pack 1
Euclidean similarity with the query image : 2.6457513110645907
==



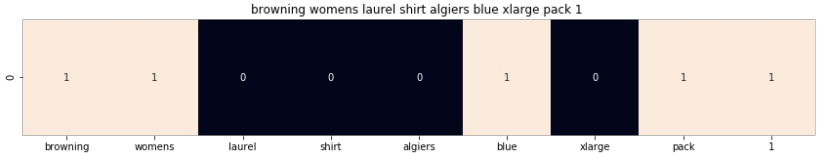
ASIN : B074KQNC89
Brand: Browning
Title: browning womens fitted buckmark tee tangerine xxlarge pack 1
Euclidean similarity with the query image : 2.8284271247461903
==



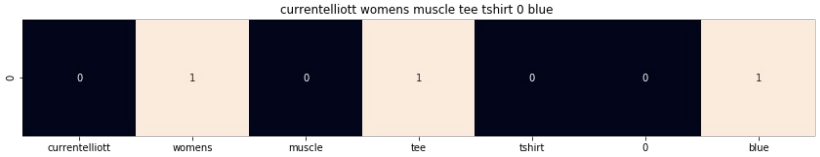
ASIN : B01LZ0VNQ4
Brand: Fever
Title: fever womens sleevless blouse blue medium
Euclidean similarity with the query image : 2.8284271247461903
==========



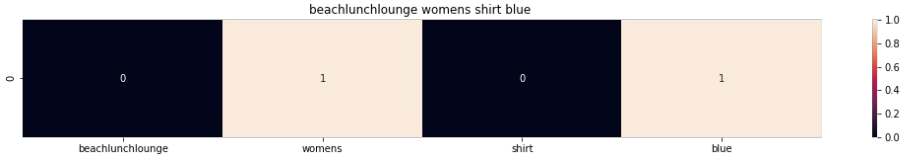
ASIN : B0753LB49H
Brand: Self-Portrait
Title: selfportrait womens top 4 blue
Euclidean similarity with the query image : 2.8284271247461903
==========



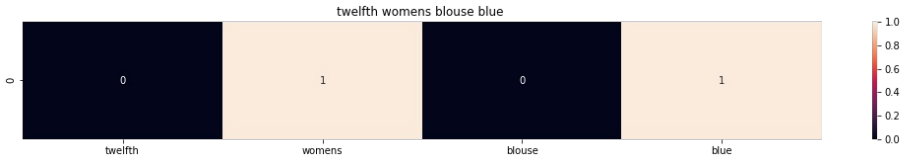
ASIN : B01KAYFT7O
Brand: SPG Outdoors
Title: browning womens laurel shirt algiers blue xlarge pack 1
Euclidean similarity with the query image : 2.8284271247461903
==========



ASIN : B01GEVBQU6
Brand: Current/Elliott
Title: currentelliott womens muscle tee tshirt 0 blue
Euclidean similarity with the query image : 2.8284271247461903
==========

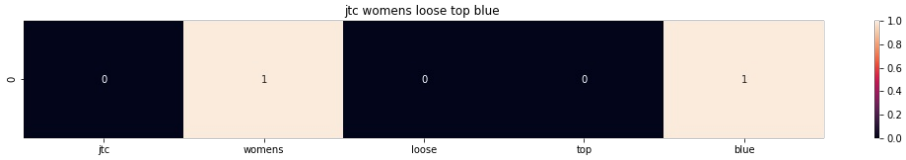


ASIN : B074SZ54KJ
Brand: Beach Lunch Lounge
Title: beachlunchlounge womens shirt blue
Euclidean similarity with the query image : 2.8284271247461903
==========

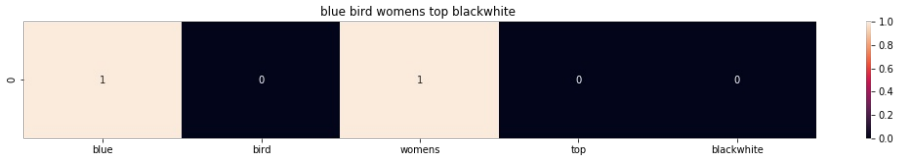


ASIN : B074F5BP5F
Brand: On Twelfth
Title: twelfth womens blouse blue
Euclidean similarity with the query image : 2.8284271247461903
==========

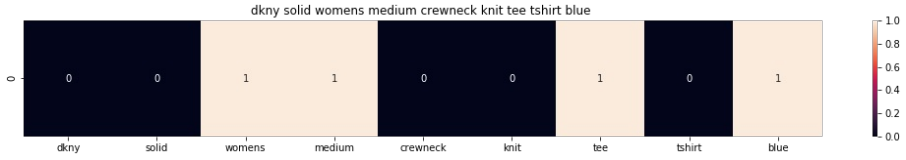
ASIN : B07583CQFT
Brand: Very J
Title: j womens shoulder blue small
Euclidean similarity with the query image : 2.8284271247461903
==



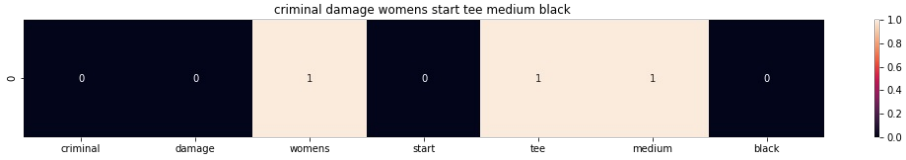
ASIN : B00JYNFMZC
Brand: Jtc
Title: jtc womens loose top blue
Euclidean similarity with the query image : 3.0
==



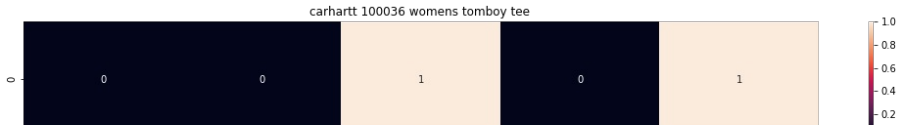
ASIN : B0711D8WMM
Brand: NRS
Title: blue bird womens top blackwhite
Euclidean similarity with the query image : 3.0
==



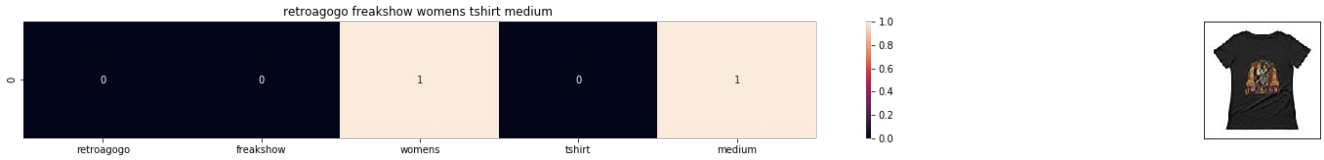
ASIN : B071JRTC33
Brand: DKNY
Title: dkny solid womens medium crewneck knit tee tshirt blue
Euclidean similarity with the query image : 3.0
==



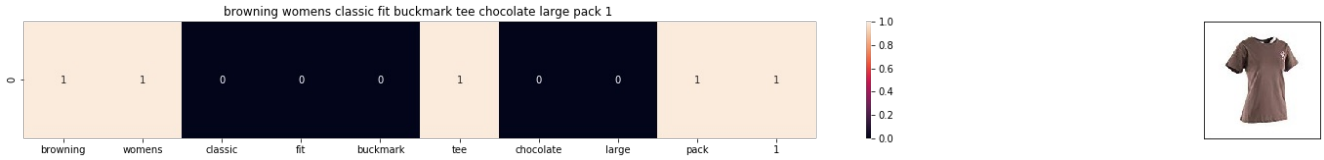
ASIN : B00HHIQTX0
Brand: Criminal Damage
Title: criminal damage womens start tee medium black
Euclidean similarity with the query image : 3.0
==



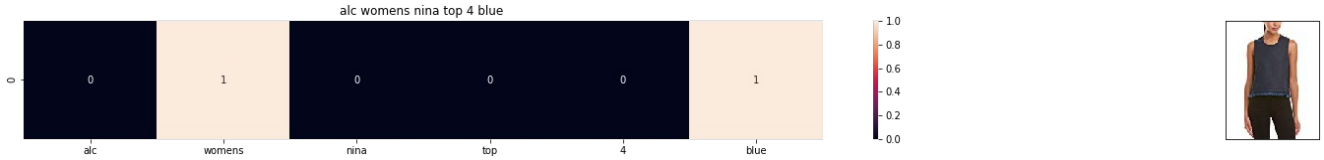

```
ASIN : B0088YO61O
Brand: Carhartt
Title: carhartt 100036 womens tomboy tee
Euclidean similarity with the query image : 3.0
============================================================
```



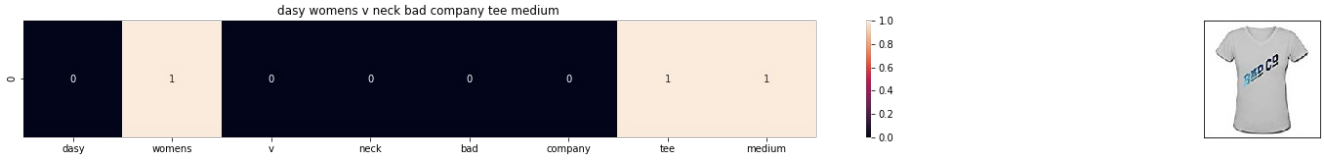

```
ASIN : B06Y5Z4FQ2
Brand: Retro-a-go-go!
Title: retroagogo freakshow womens tshirt medium
Euclidean similarity with the query image : 3.0
============================================================
```




```
ASIN : B074KPYSRC
Brand: Browning
Title: browning womens classic fit buckmark tee chocolate large pack 1
Euclidean similarity with the query image : 3.0
============================================================
```




```
ASIN : B01MPX3OQJ
Brand: A.L.C.
Title: alc womens nina top 4 blue
Euclidean similarity with the query image : 3.0
============================================================
```




```
ASIN : B01628Q0GG
Brand: Dasy
Title: dasy womens v neck bad company tee medium
Euclidean similarity with the query image : 3.0
============================================================
```

[8.5] TF-IDF based product similarity

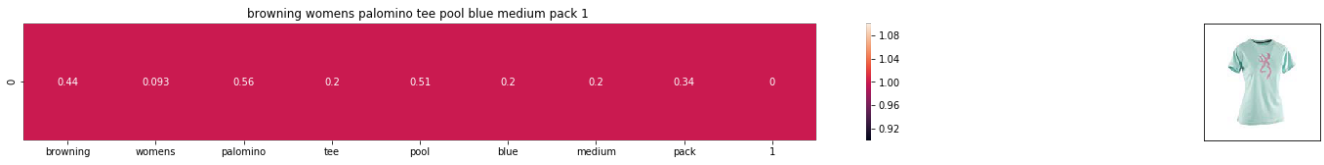
In [50]:

```
tfidf_title_vectorizer = TfidfVectorizer(min_df = 0)
tfidf_title_features = tfidf_title_vectorizer.fit_transform(data['title'])
# tfidf_title_features.shape = #data_points * #words_in_corpus
# CountVectorizer().fit_transform(courpus) returns the a sparase matrix of dimensions #data_points
* #words_in_corpus
```

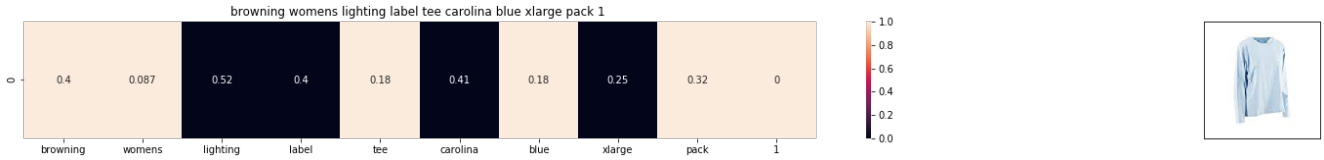
```
# tfidf_title_features[doc_id, index_of_word_in_corpus] = tfidf values of the word in given doc
```

In [51]:

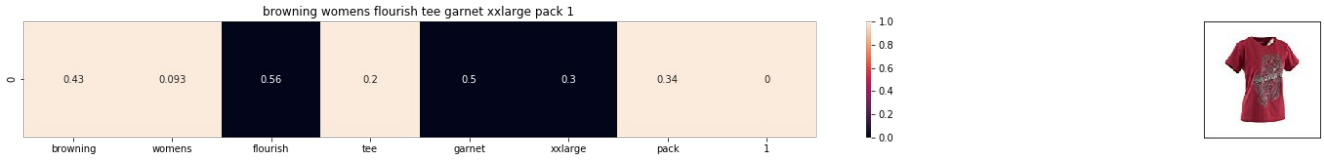
```
def tfidf_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

    # pairwise_dist will store the distance from given input apparel to all remaining apparels
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y) = <X, Y> / (||X
||*||Y||)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(tfidf_title_features,tfidf_title_features[doc_id])

    # np.argsort will return indices of 9 smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the 9 smallest distances
    pdists  = np.sort(pairwise_dist.flatten())[0:num_results]

    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])

    for i in range(0,len(indices)):
        # we will pass 1. doc_id, 2. title1, 3. title2, url, model
        get_result(indices[i], data['title'].loc[df_indices[0]], data['title'].loc[df_indices[i]],
data['medium_image_url'].loc[df_indices[i]], 'tfidf')
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('BRAND :',data['brand'].loc[df_indices[i]])
        print ('Eucliden distance from the given image :', pdists[i])
        print('='*125)
tfidf_model(12000, 20)
# in the output heat map each value represents the tfidf values of the label word, the color repre
sents the intersection with inputs title
```



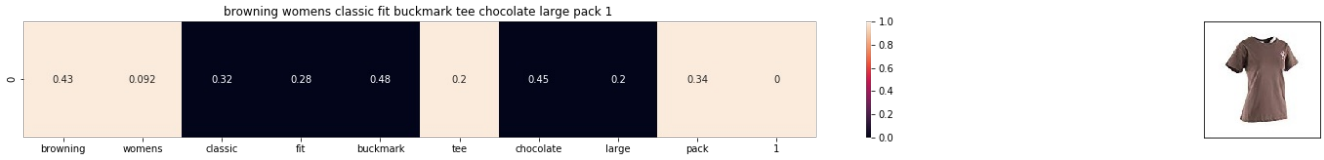
ASIN : B074KQ3SY1
BRAND : Browning
Eucliden distance from the given image : 0.0
==========



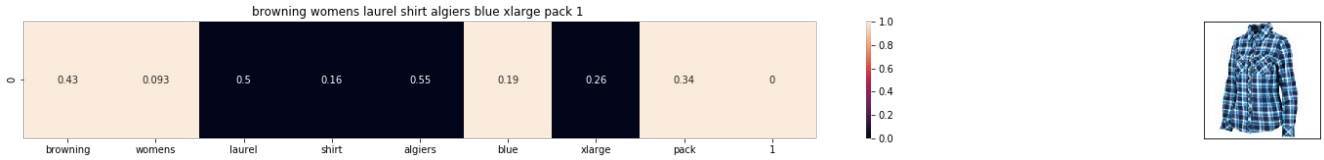
ASIN : B074KQJ8KC
BRAND : Browning
Eucliden distance from the given image : 1.127354176779969
==========



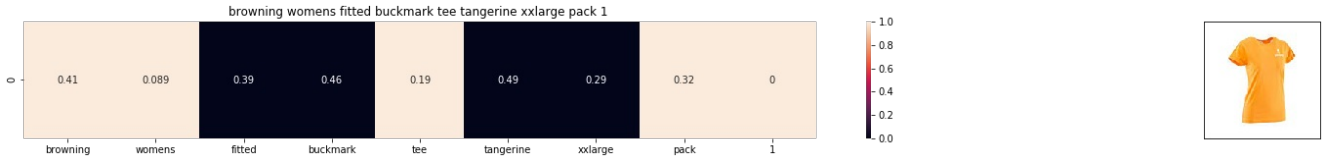
ASIN : B01IL3NF40
BRAND : Browning
Eucliden distance from the given image : 1.1369882005941534
==========



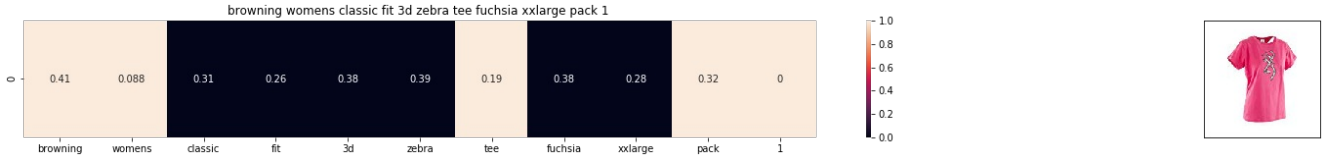
ASIN : B074KPYSRC
BRAND : Browning
Eucliden distance from the given image : 1.1397068537220458
==========



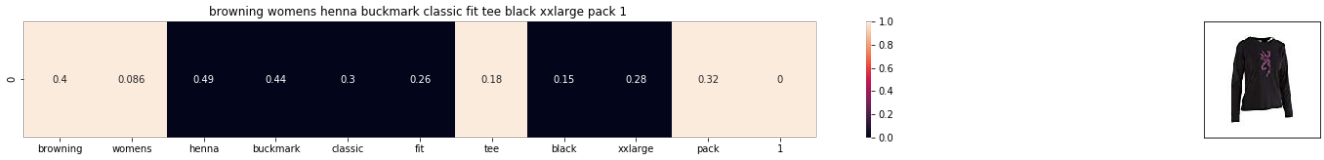
ASIN : B01KAYFT7O
BRAND : SPG Outdoors
Eucliden distance from the given image : 1.1403212681065322
==========



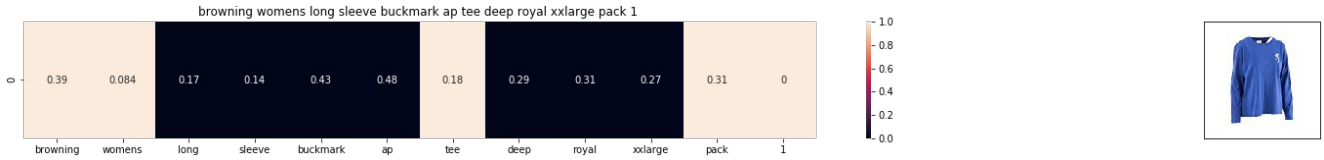
ASIN : B074KQNC89
BRAND : Browning
Eucliden distance from the given image : 1.15239859368034
==========



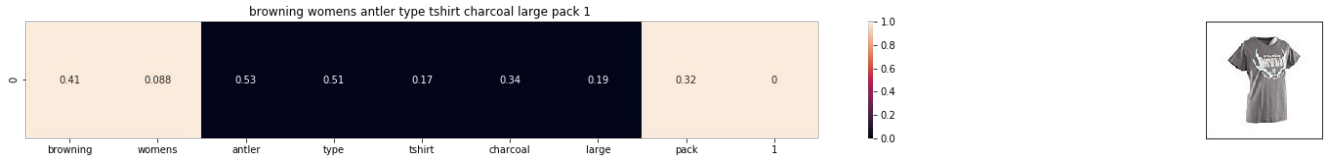
ASIN : B074KQ9P3W
BRAND : Browning
Eucliden distance from the given image : 1.1541889269613568
==========



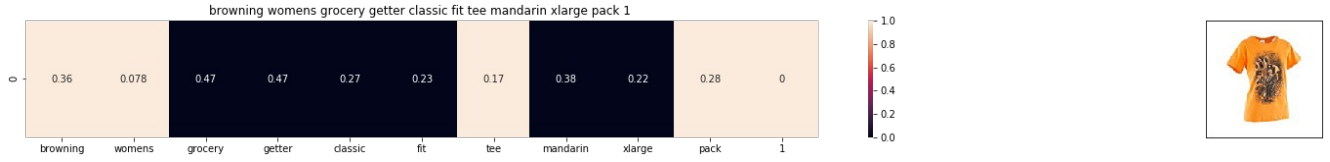
ASIN : B00NQFC32Y
BRAND : Browning
Eucliden distance from the given image : 1.1599276944891332
==========



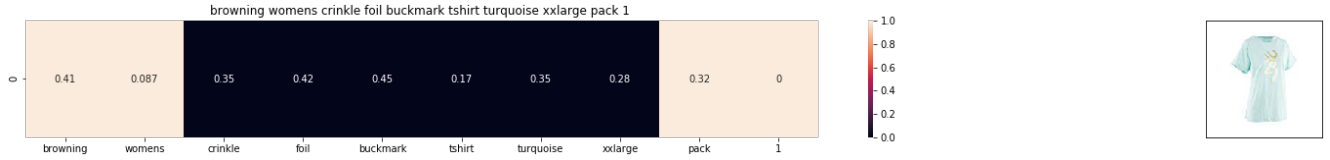
ASIN : B00QMT5KPS
BRAND : Browning
Eucliden distance from the given image : 1.168492346439867
==========



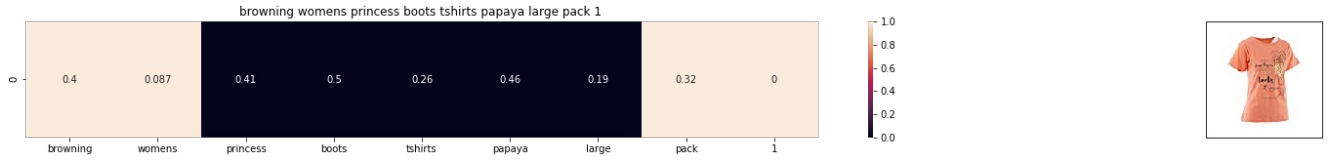
ASIN : B074KPDZH8
BRAND : Browning
Eucliden distance from the given image : 1.185439268372708
==========



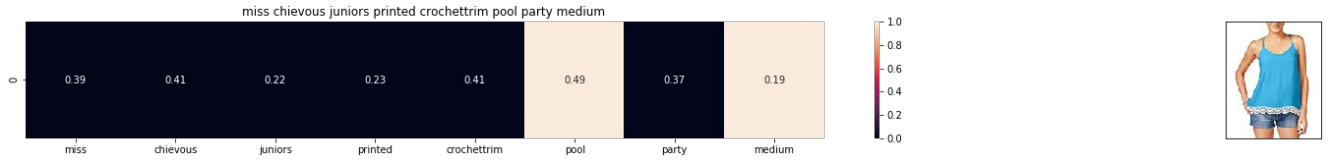
ASIN : B00QMSXHY0
BRAND : Browning
Eucliden distance from the given image : 1.1871058874699503
==========



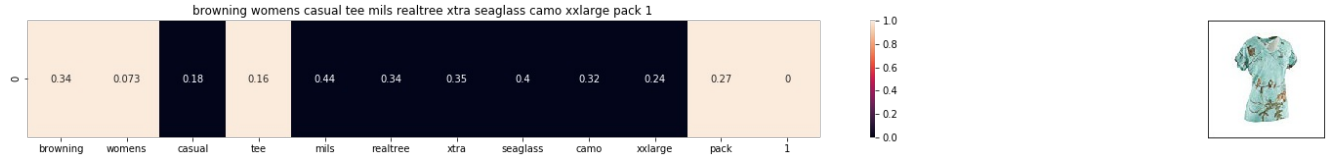
ASIN : B074KQGXQD
BRAND : Browning
Eucliden distance from the given image : 1.1892270034914678
==========



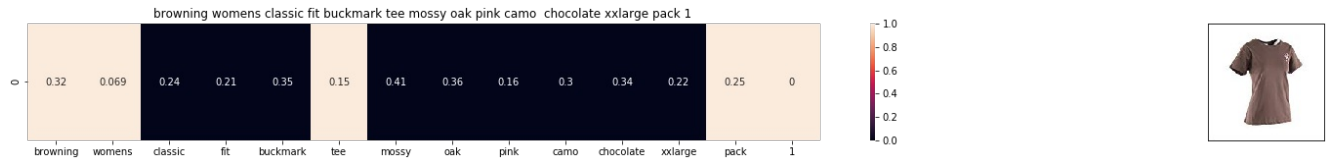
ASIN : B074KPJ8YP
BRAND : Browning
Eucliden distance from the given image : 1.190266896947773
==========



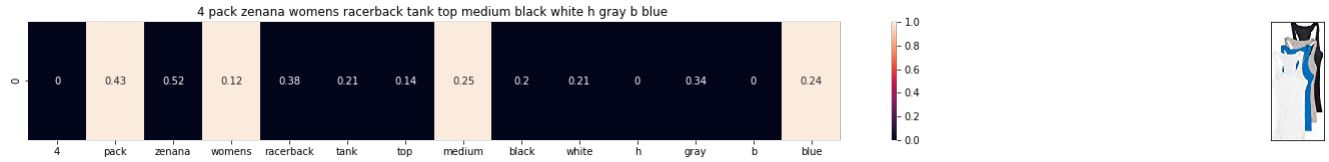
ASIN : B07287C489
BRAND : Miss Chievous
Eucliden distance from the given image : 1.192689370349002
==========



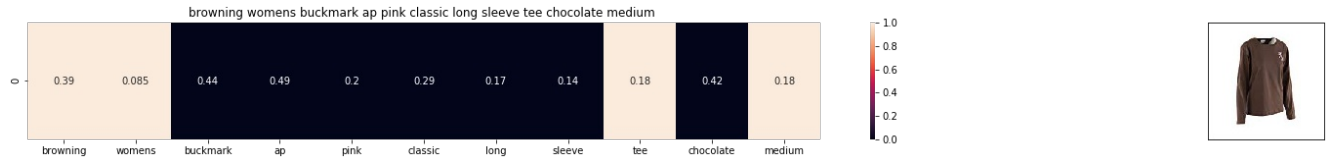
ASIN : B074KQCY8N
BRAND : Browning
Eucliden distance from the given image : 1.20113095419572
==========



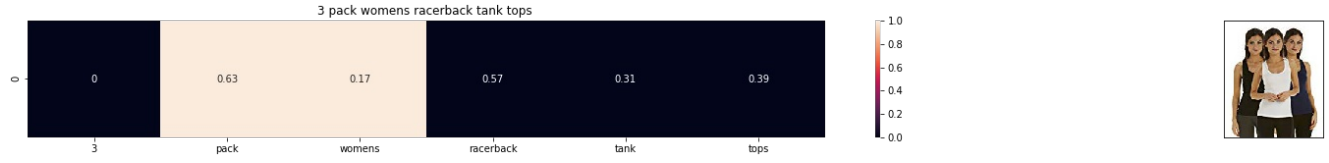
ASIN : B074KPXCPC
BRAND : Browning
Eucliden distance from the given image : 1.2153310659676564
==========



ASIN : B01ESA6WKE
BRAND : Zenana Outfitters
Eucliden distance from the given image : 1.2220806252309797
==========



ASIN : B00NQFH7MA
BRAND : Browning
Eucliden distance from the given image : 1.2238096341041258
==========



ASIN : B01MUAOXSN
BRAND : Free to Live
Eucliden distance from the given image : 1.2396359259238885
==========


```
ASIN : B01MS4162P
BRAND : Horseware Ireland
Eucliden distance from the given image : 1.2434676636122615
=============================================================================================================
======================
```

[8.5] IDF based product similarity

In [52]:

```
idf_title_vectorizer = CountVectorizer()
idf_title_features = idf_title_vectorizer.fit_transform(data['title'])

# idf_title_features.shape = #data_points * #words_in_corpus
# CountVectorizer().fit_transform(courpus) returns the a sparase matrix of dimensions #data_points
* #words_in_corpus
# idf_title_features[doc_id, index_of_word_in_corpus] = number of times the word occured in that d
oc
```

In [53]:

```
def n_containing(word):
    # return the number of documents which had the given word
    return sum(1 for blob in data['title'] if word in blob.split())

def idf(word):
    # idf = log(#number of docs / #number of docs which had the given word)
    return math.log(data.shape[0] / (n_containing(word)))
```

In [54]:

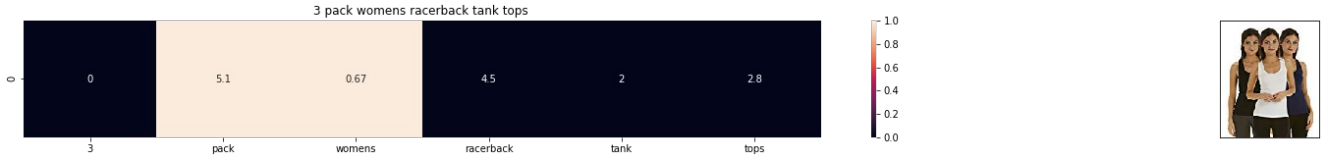
```
# we need to convert the values into float
idf_title_features  = idf_title_features.astype(np.float)

for i in idf_title_vectorizer.vocabulary_.keys():
    # for every word in whole corpus we will find its idf value
    idf_val = idf(i)

    # to calculate idf_title_features we need to replace the count values with the idf values of t
he word
    # idf_title_features[:, idf_title_vectorizer.vocabulary_[i]].nonzero()[0] will return all docu
ments in which the word i present
    for j in idf_title_features[:, idf_title_vectorizer.vocabulary_[i]].nonzero()[0]:

        # we replace the count values of word i in document j with  idf_value of word i
        # idf_title_features[doc_id, index_of_word_in_courpus] = idf value of word
        idf_title_features[j,idf_title_vectorizer.vocabulary_[i]] = idf_val
```

In [55]:

```
def idf_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

    # pairwise_dist will store the distance from given input apparel to all remaining apparels
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y) = <X, Y> / (||X
||*||Y||)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(idf_title_features,idf_title_features[doc_id])

    # np.argsort will return indices of 9 smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the 9 smallest distances
    pdists  = np.sort(pairwise_dist.flatten())[0:num_results]

    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])

    for i in range(0,len(indices)):
```

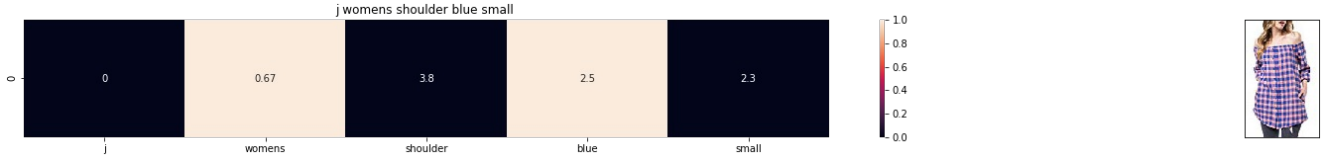
```
    for i in range(0,len(indices)):
        get_result(indices[i],data['title'].loc[df_indices[0]], data['title'].loc[df_indices[i]], d
ata['medium_image_url'].loc[df_indices[i]], 'idf')
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('Brand :',data['brand'].loc[df_indices[i]])
        print ('euclidean distance from the given image :', pdists[i])
        print('='*125)



idf_model(12000,20)
# in the output heat map each value represents the idf values of the label word, the color represe
nts the intersection with inputs title
```

ASIN : B074KQ3SY1
Brand : Browning
euclidean distance from the given image : 0.0
==========



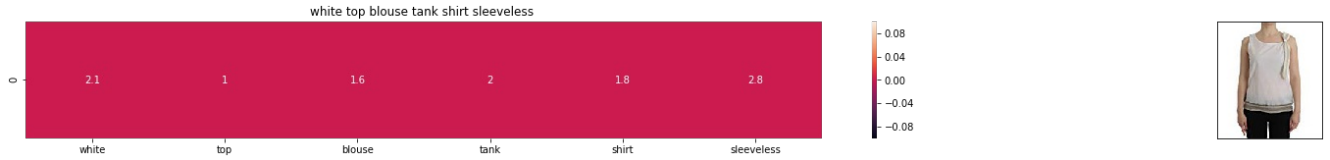
ASIN : B01MUAOXSN
Brand : Free to Live
euclidean distance from the given image : 16.146626132996104
==========

ASIN : B00JPOZ9GM
Brand : Sofra
euclidean distance from the given image : 16.253121081411198
==========



ASIN : B07583CQFT
Brand : Very J
euclidean distance from the given image : 16.388052867689414
==========

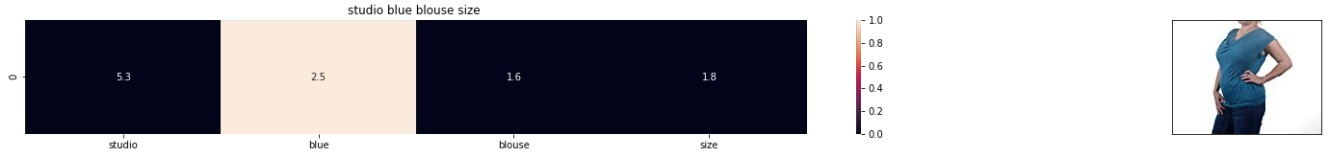
ASIN : B00KF2N5PU
Brand : Vietsbay
euclidean distance from the given image : 16.512204860612744
==========



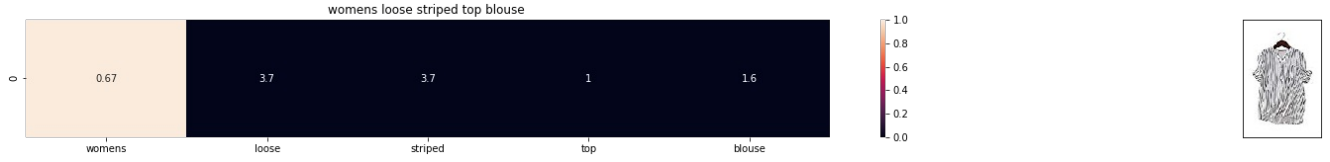
ASIN : B074G5G5RK
Brand : ERMANNO SCERVINO
euclidean distance from the given image : 16.687740268391035
==========

ASIN : B074T9KG9Q
Brand : Rain
euclidean distance from the given image : 16.740372508536527
==========

ASIN : B017I2YWUQ
Brand : Z SUPPLY
euclidean distance from the given image : 16.787771739481727
==========

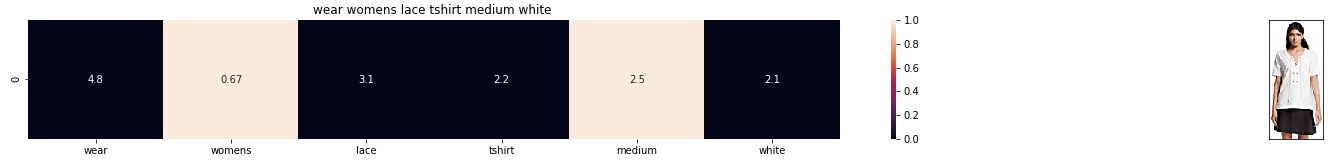


ASIN : B016P80OKQ
Brand : Studio M
euclidean distance from the given image : 16.807798945363338
==========

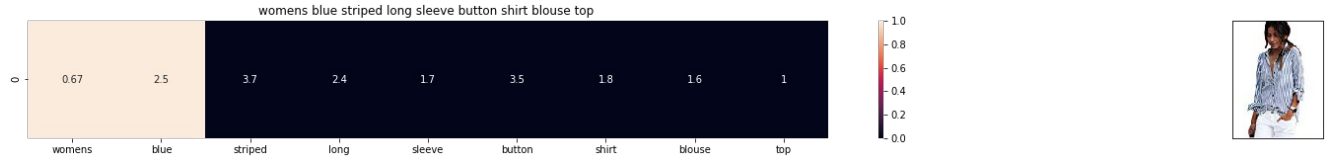


ASIN : B00ZZMYBRG
Brand : HP-LEISURE
euclidean distance from the given image : 16.912808152182386
==========

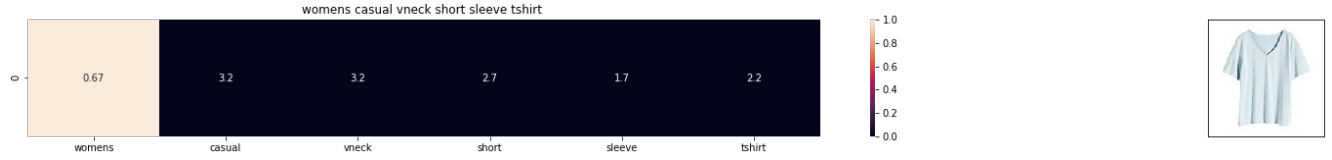
ASIN : B073JWSM1V
Brand : Fuming
euclidean distance from the given image : 17.020566054273353
==========



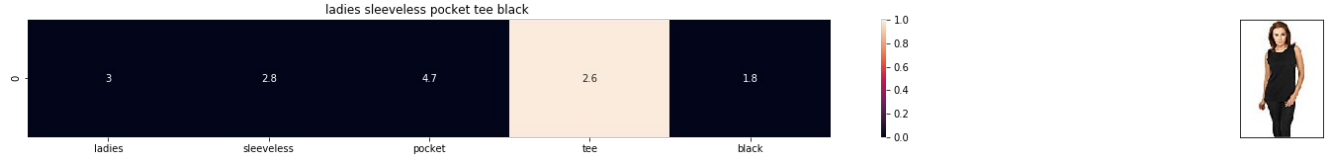
ASIN : B01MZZF0BB
Brand : Who What Wear
euclidean distance from the given image : 17.026179048093606
==========



ASIN : B01KM2OLIW
Brand : Voguegirl
euclidean distance from the given image : 17.028926287950878
==========



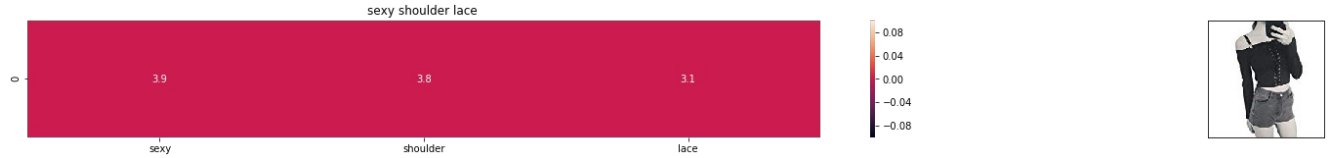
ASIN : B074V45DCX
Brand : Rain
euclidean distance from the given image : 17.048413760545156
==========



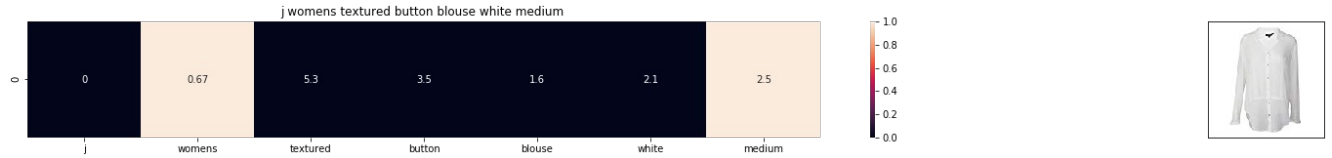
ASIN : B01GG0SZ0Y
Brand : Urban Classics
euclidean distance from the given image : 17.075706632527957
==========

ASIN : B00ZZPR4Y0
Brand : HP-LEISURE
euclidean distance from the given image : 17.13287164584408
==========

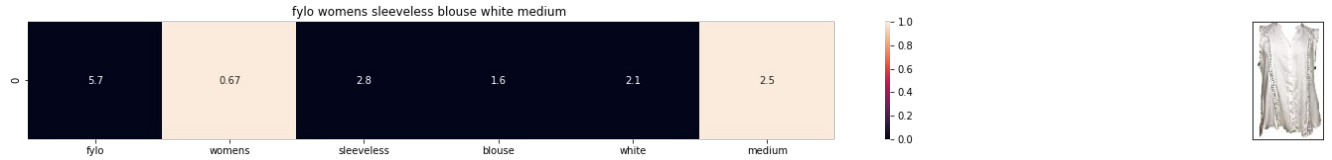
ASIN : B073GJGVBN
Brand : Ivan Levi
euclidean distance from the given image : 17.140178663387896
==========



ASIN : B01JVWUB3S
Brand : bylexie
euclidean distance from the given image : 17.154076390230408
==========



ASIN : B073G73C2Q
Brand : Very J
euclidean distance from the given image : 17.183364048332603
==========



ASIN : B07213CJ5T
Brand : Fylo
euclidean distance from the given image : 17.196360744937436
==========

[9] Text Semantics based product similarity

In [0]:

```
# credits: https://www.kaggle.com/c/word2vec-nlp-tutorial#part-2-word-vectors
# Custom Word2Vec using your own text data.
# Do NOT RUN this code.
# It is meant as a reference to build your own Word2Vec when you have
# lots of data
```

```
# lots of data.

'''
# Set values for various parameters
num_features = 300    # Word vector dimensionality
min_word_count = 1    # Minimum word count
num_workers = 4       # Number of threads to run in parallel
context = 10          # Context window size

downsampling = 1e-3   # Downsample setting for frequent words

# Initialize and train the model (this will take some time)
from gensim.models import word2vec
print ("Training model...")
model = word2vec.Word2Vec(sen_corpus, workers=num_workers, \
            size=num_features, min_count = min_word_count, \
            window = context)

'''
```

In [56]:

```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.

'''
model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
'''

#if you do NOT have RAM >= 12GB, use the code below.
with open('word2vec_model', 'rb') as handle:
    model = pickle.load(handle)
```

In [57]:

```
# Utility functions

def get_word_vec(sentence, doc_id, m_name):
    # sentence : title of the apparel
    # doc_id: document id in our corpus
    # m_name: model information it will take two values
        # if  m_name == 'avg', we will append the model[i], w2v representation of word i
        # if m_name == 'weighted', we will multiply each w2v[word] with the idf(word)
    vec = []
    for i in sentence.split():
        if i in vocab:
            if m_name == 'weighted' and i in  idf_title_vectorizer.vocabulary_:
                vec.append(idf_title_features[doc_id, idf_title_vectorizer.vocabulary_[i]] * model[
i])
            elif m_name == 'avg':
                vec.append(model[i])
        else:
            # if the word in our courpus is not there in the google word2vec corpus, we are just
ignoring it
            vec.append(np.zeros(shape=(300,)))
    # we will return a numpy array of shape (#number of words in title * 300 ) 300 =
len(w2v_model[word])
    # each row represents the word2vec representation of each word (weighted/avg) in given
sentance
    return  np.array(vec)

def get_distance(vec1, vec2):
    # vec1 = np.array(#number_of_words_title1 * 300), each row is a vector of length 300
corresponds to each word in give title
    # vec2 = np.array(#number_of_words_title2 * 300), each row is a vector of length 300
```

```
corresponds to each word in give title

    final_dist = []
    # for each vector in vec1 we caluclate the distance(euclidean) to all vectors in vec2
    for i in vec1:
        dist = []
        for j in vec2:
            # np.linalg.norm(i-j) will result the euclidean distance between vectors i, j
            dist.append(np.linalg.norm(i-j))
        final_dist.append(np.array(dist))
    # final_dist = np.array(#number of words in title1 * #number of words in title2)
    # final_dist[i,j] = euclidean distance between vectors i, j
    return np.array(final_dist)


def heat_map_w2v(sentence1, sentence2, url, doc_id1, doc_id2, model):
    # sentance1 : title1, input apparel
    # sentance2 : title2, recommended apparel
    # url: apparel image url
    # doc_id1: document id of input apparel
    # doc_id2: document id of recommended apparel
    # model: it can have two values, 1. avg 2. weighted

    #s1_vec = np.array(#number_of_words_title1 * 300), each row is a vector(weighted/avg) of lengt
h 300 corresponds to each word in give title
    s1_vec = get_word_vec(sentence1, doc_id1, model)
    #s2_vec = np.array(#number_of_words_title1 * 300), each row is a vector(weighted/avg) of lengt
h 300 corresponds to each word in give title
    s2_vec = get_word_vec(sentence2, doc_id2, model)

    # s1_s2_dist = np.array(#number of words in title1 * #number of words in title2)
    # s1_s2_dist[i,j] = euclidean distance between words i, j
    s1_s2_dist = get_distance(s1_vec, s2_vec)



    # devide whole figure into 2 parts 1st part displays heatmap 2nd part displays image of appare
l
    gs = gridspec.GridSpec(2, 2, width_ratios=[4,1],height_ratios=[2,1])
    fig = plt.figure(figsize=(15,15))

    ax = plt.subplot(gs[0])
    # ploting the heap map based on the pairwise distances
    ax = sns.heatmap(np.round(s1_s2_dist,4), annot=True)
    # set the x axis labels as recommended apparels title
    ax.set_xticklabels(sentence2.split())
    # set the y axis labels as input apparels title
    ax.set_yticklabels(sentence1.split())
    # set title as recommended apparels title
    ax.set_title(sentence2)

    ax = plt.subplot(gs[1])
    # we remove all grids and axis labels for image
    ax.grid(False)
    ax.set_xticks([])
    ax.set_yticks([])
    display_img(url, ax, fig)

    plt.show()
```

In [58]:

```
# vocab = stores all the words that are there in google w2v model
# vocab = model.wv.vocab.keys() # if you are using Google word2Vec

vocab = model.keys()
# this function will add the vectors of each word and returns the avg vector of given sentance
def build_avg_vec(sentence, num_features, doc_id, m_name):
    # sentace: its title of the apparel
    # num_features: the lenght of word2vec vector, its values = 300
    # m_name: model information it will take two values
        # if  m_name == 'avg', we will append the model[i], w2v representation of word i
        # if m_name == 'weighted', we will multiply each w2v[word] with the idf(word)

    featureVec = np.zeros((num_features,), dtype="float32")
    # we will intialize a vector of size 300 with all zeros
```

```
    # we add each word2vec(word1) to this featureVec
    nwords = 0

    for word in sentence.split():
        nwords += 1
        if word in vocab:
            if m_name == 'weighted' and word in  idf_title_vectorizer.vocabulary_:
                featureVec = np.add(featureVec, idf_title_features[doc_id, idf_title_vectorizer.voc
abulary_[word]] * model[word])
            elif m_name == 'avg':
                featureVec = np.add(featureVec, model[word])
    if(nwords>0):
        featureVec = np.divide(featureVec, nwords)
    # returns the avg vector of given sentance, its of shape (1, 300)
    return featureVec
```

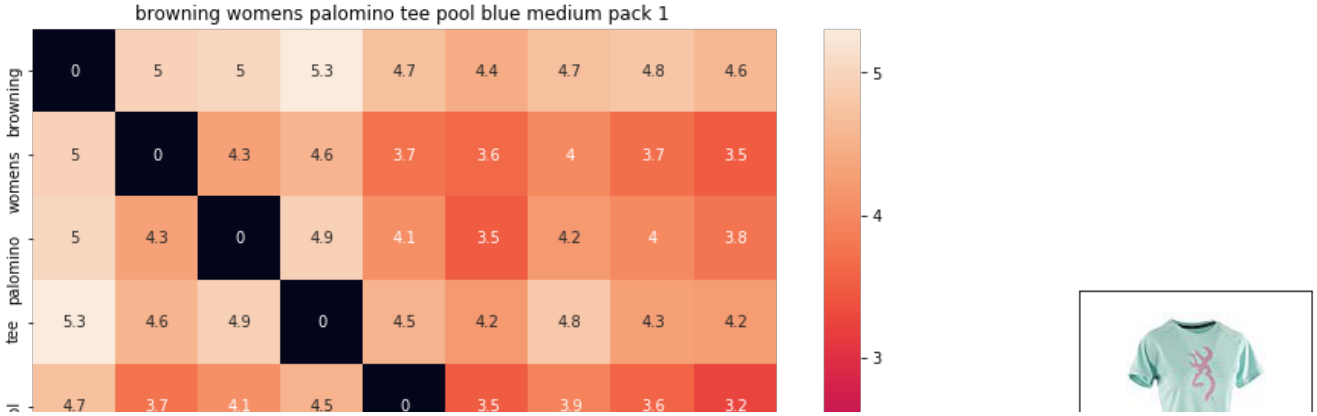
[9.2] Average Word2Vec product similarity.

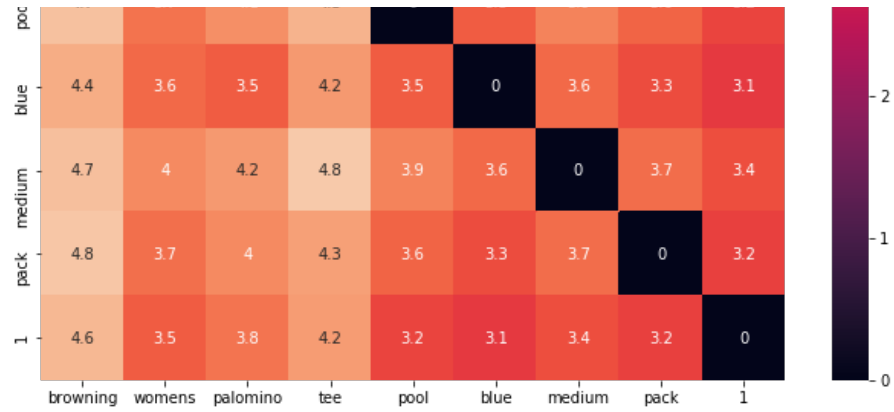
In [59]:

```
doc_id = 0
w2v_title = []
# for every title we build a avg vector representation
for i in data['title']:
    w2v_title.append(build_avg_vec(i, 300, doc_id,'avg'))
    doc_id += 1

# w2v_title = np.array(# number of doc in courpus * 300), each row corresponds to a doc
w2v_title = np.array(w2v_title)
```

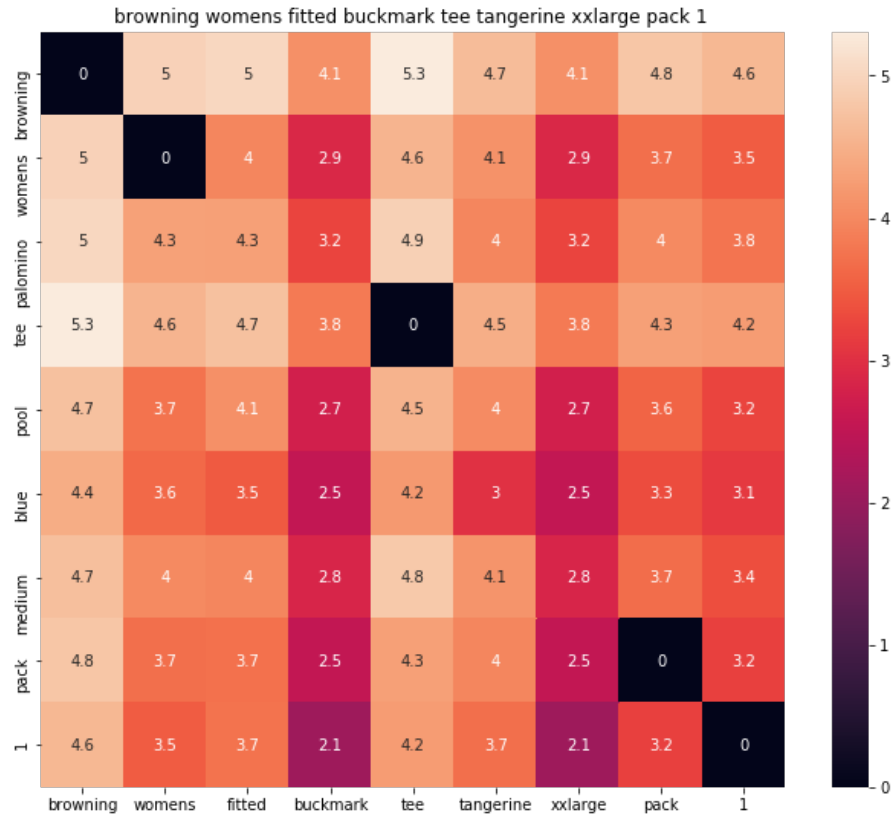
In [60]:

```
def avg_w2v_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

    # dist(x, y) = sqrt(dot(x, x) - 2 * dot(x, y) + dot(y, y))
    pairwise_dist = pairwise_distances(w2v_title, w2v_title[doc_id].reshape(1,-1))

    # np.argsort will return indices of 9 smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the 9 smallest distances
    pdists  = np.sort(pairwise_dist.flatten())[0:num_results]

    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])

    for i in range(0, len(indices)):
        heat_map_w2v(data['title'].loc[df_indices[0]],data['title'].loc[df_indices[i]], data['mediu
m_image_url'].loc[df_indices[i]], indices[0], indices[i], 'avg')
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('BRAND :',data['brand'].loc[df_indices[i]])
        print ('euclidean distance from given input image :', pdists[i])
        print('='*125)


avg_w2v_model(12000, 20)
# in the give heat map, each cell contains the euclidean distance between words i, j
```

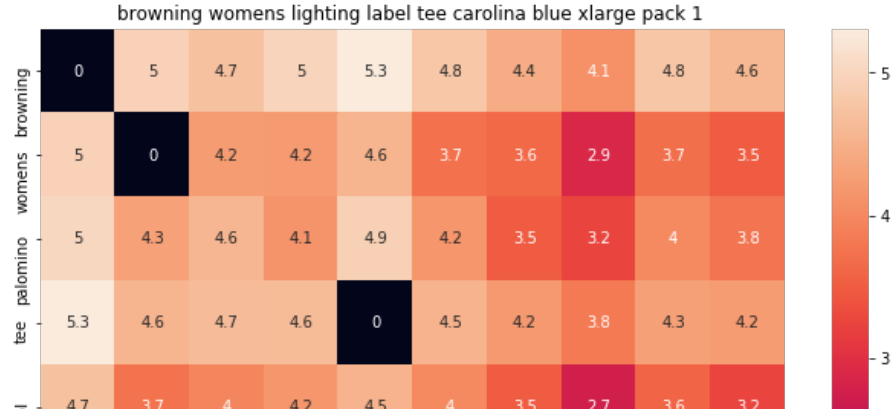


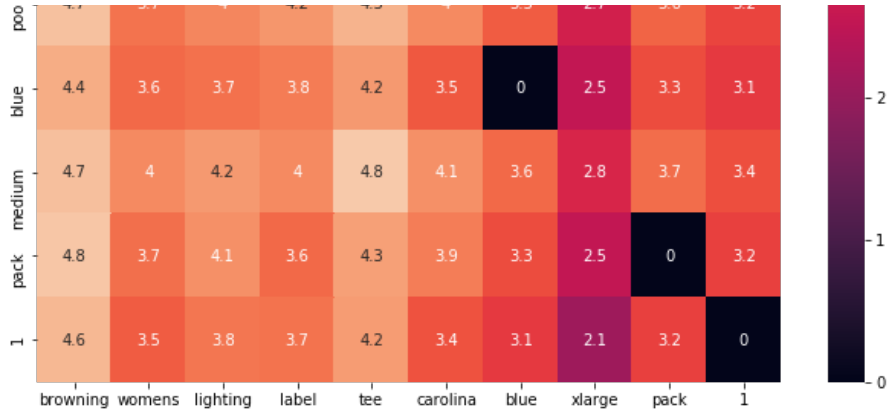


ASIN : B074KQ3SY1
BRAND : Browning
euclidean distance from given input image : 0.0
==========

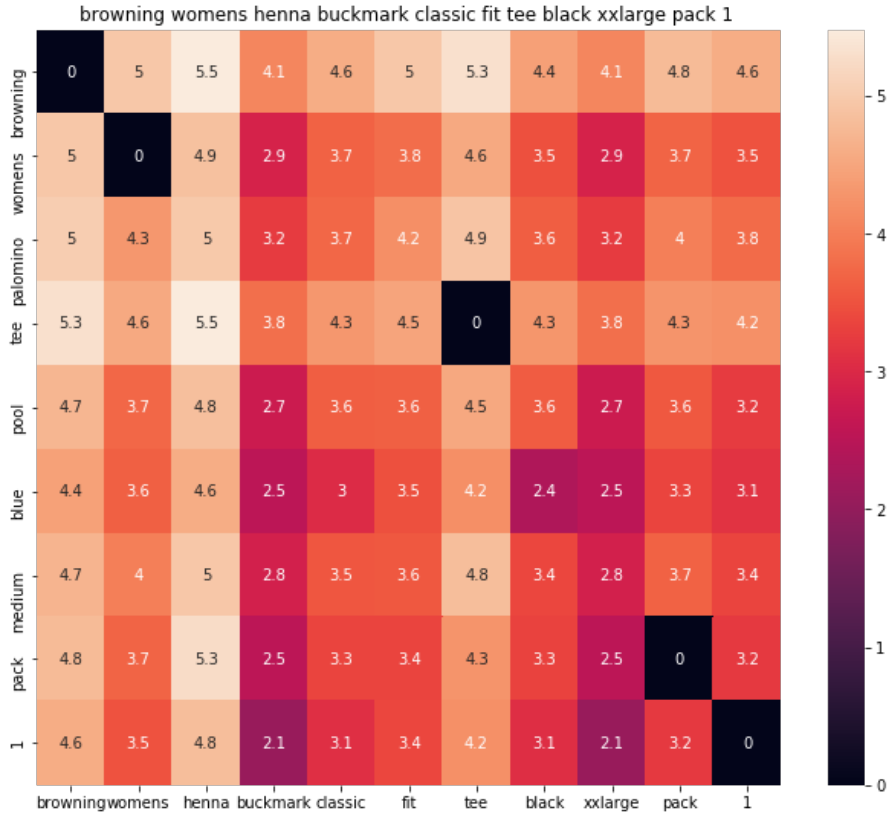


ASIN : B074KQNC89
BRAND : Browning
euclidean distance from given input image : 0.739195
==========

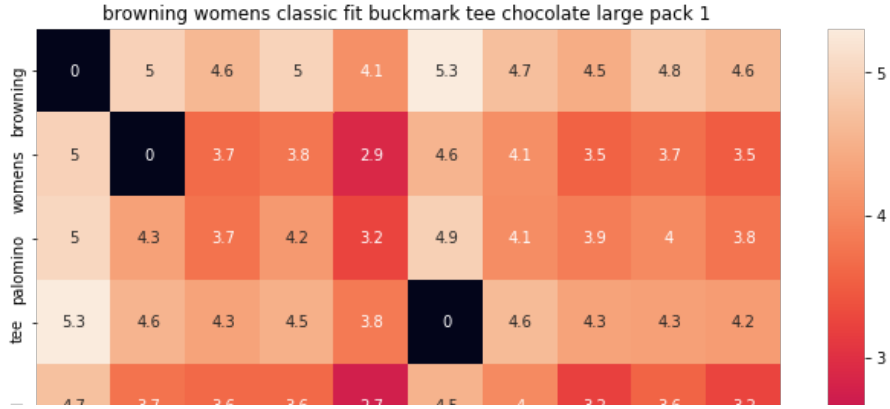


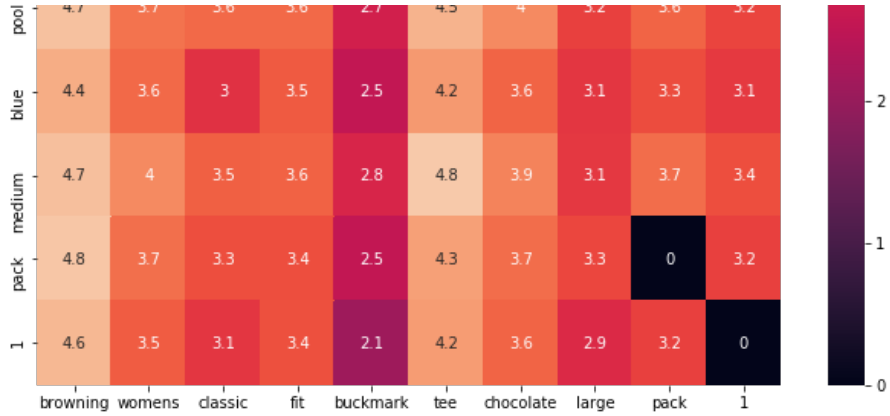


ASIN : B074KQJ8KC
BRAND : Browning
euclidean distance from given input image : 0.73952204
==========

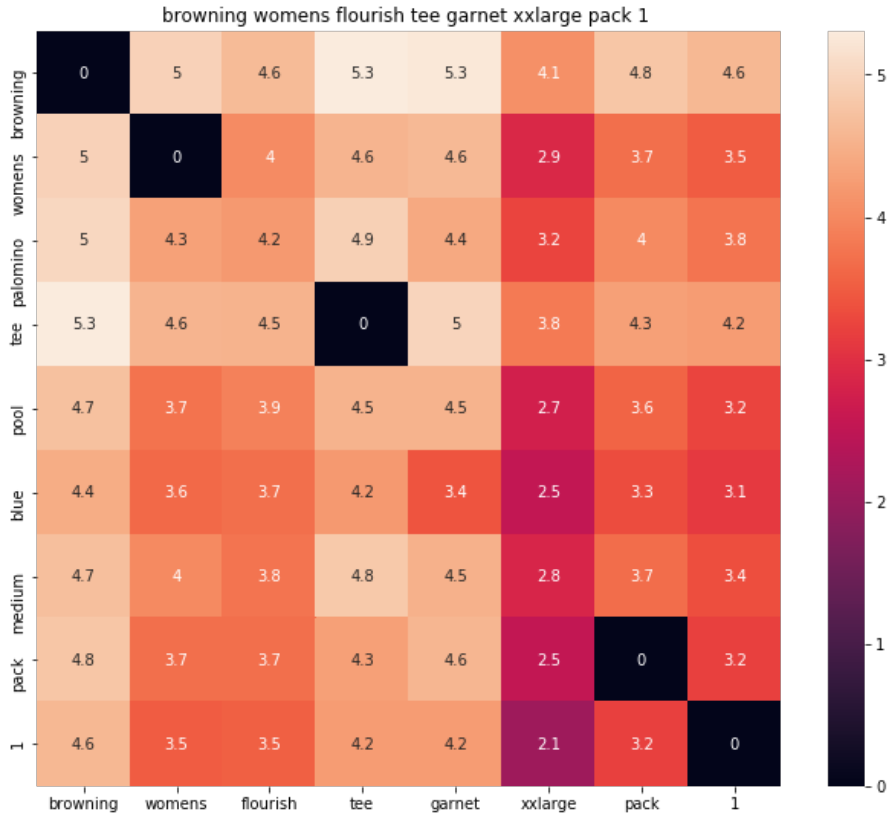


ASIN : B00NQFC32Y
BRAND : Browning
euclidean distance from given input image : 0.7458579
==========

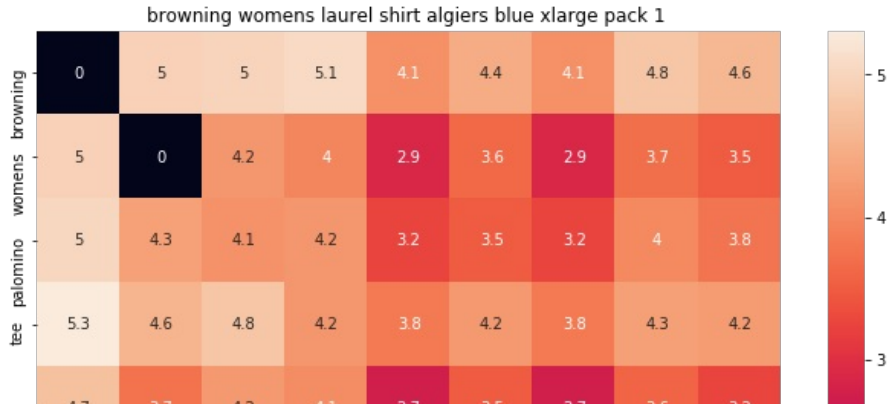


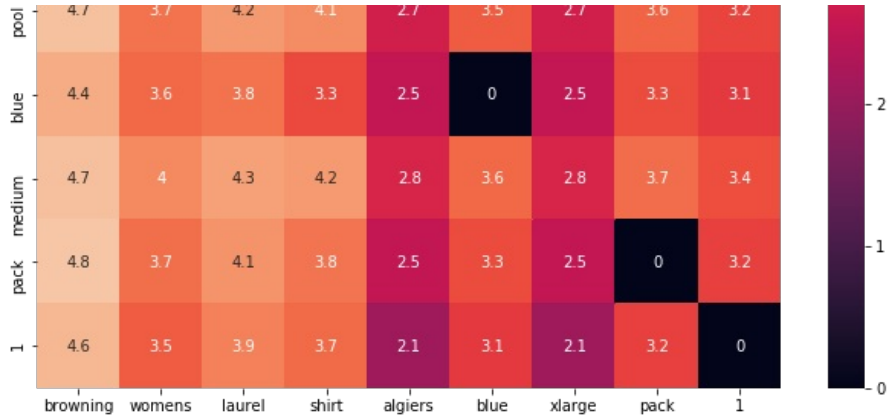


ASIN : B074KPYSRC
BRAND : Browning
euclidean distance from given input image : 0.7602023
==========

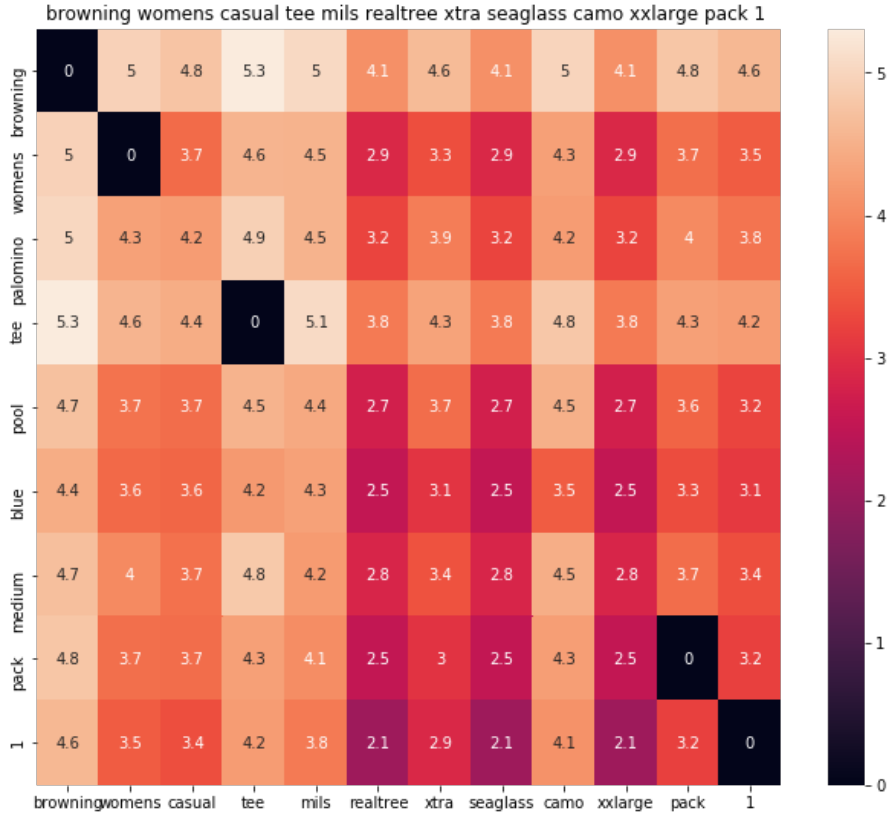


ASIN : B01IL3NF40
BRAND : Browning
euclidean distance from given input image : 0.76251435
==========

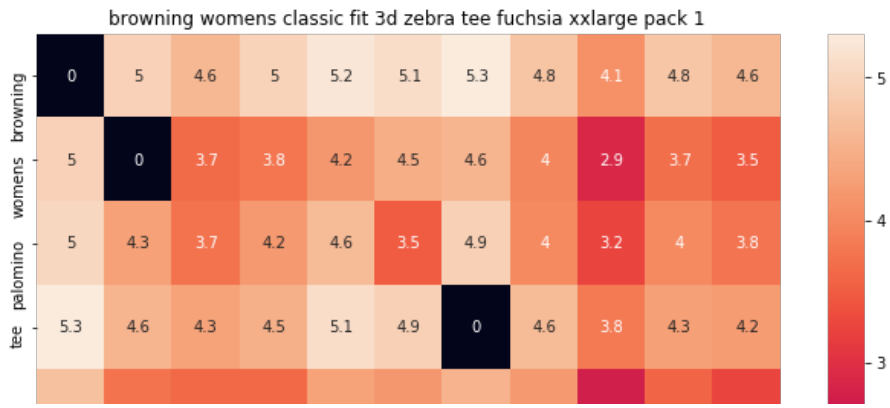


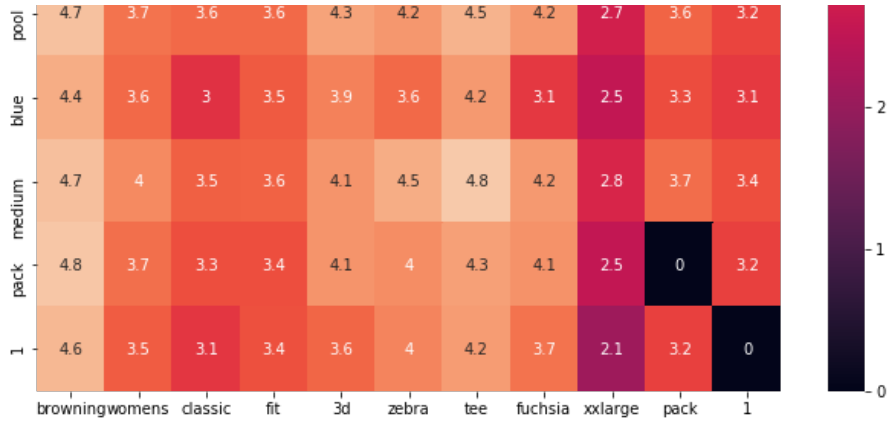


ASIN : B01KAYFT7O
BRAND : SPG Outdoors
euclidean distance from given input image : 0.7803534
==========

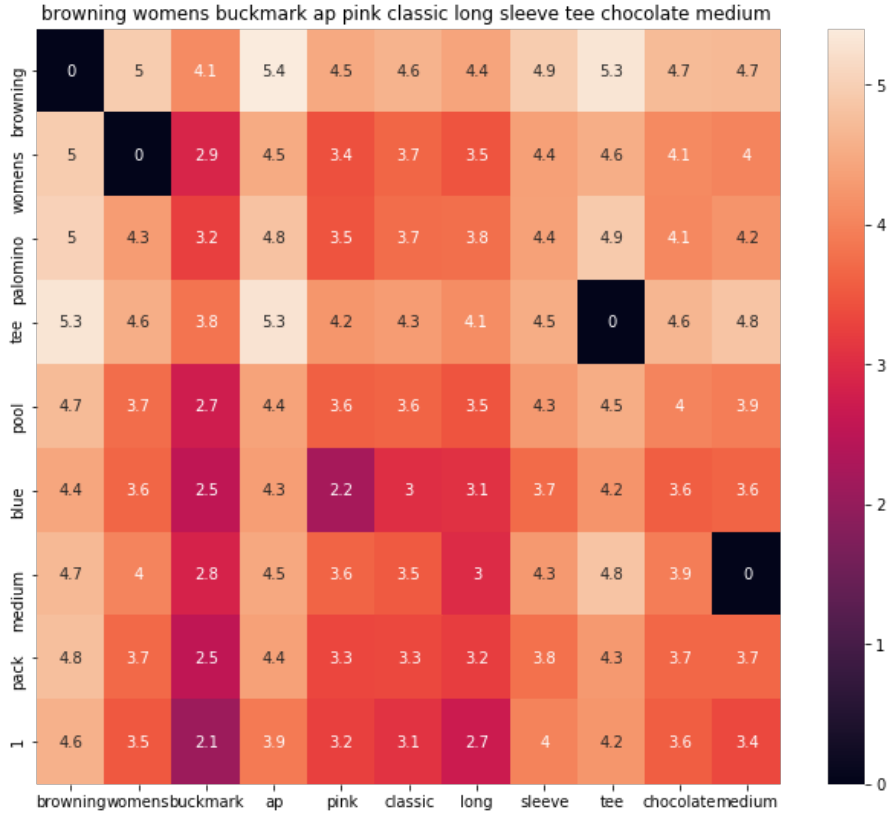


ASIN : B074KQCY8N
BRAND : Browning
euclidean distance from given input image : 0.78496605
==========

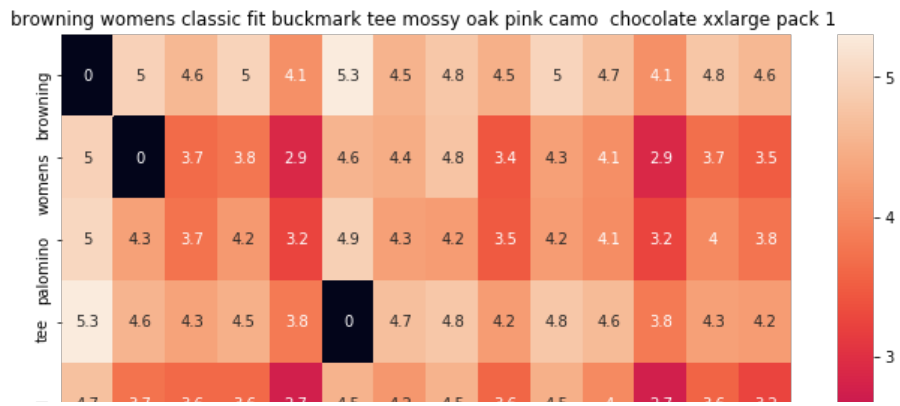


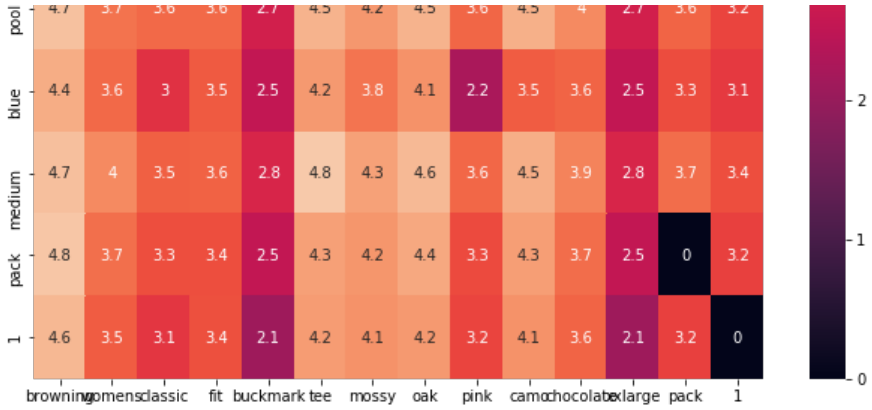


ASIN : B074KQ9P3W
BRAND : Browning
euclidean distance from given input image : 0.796898
==========

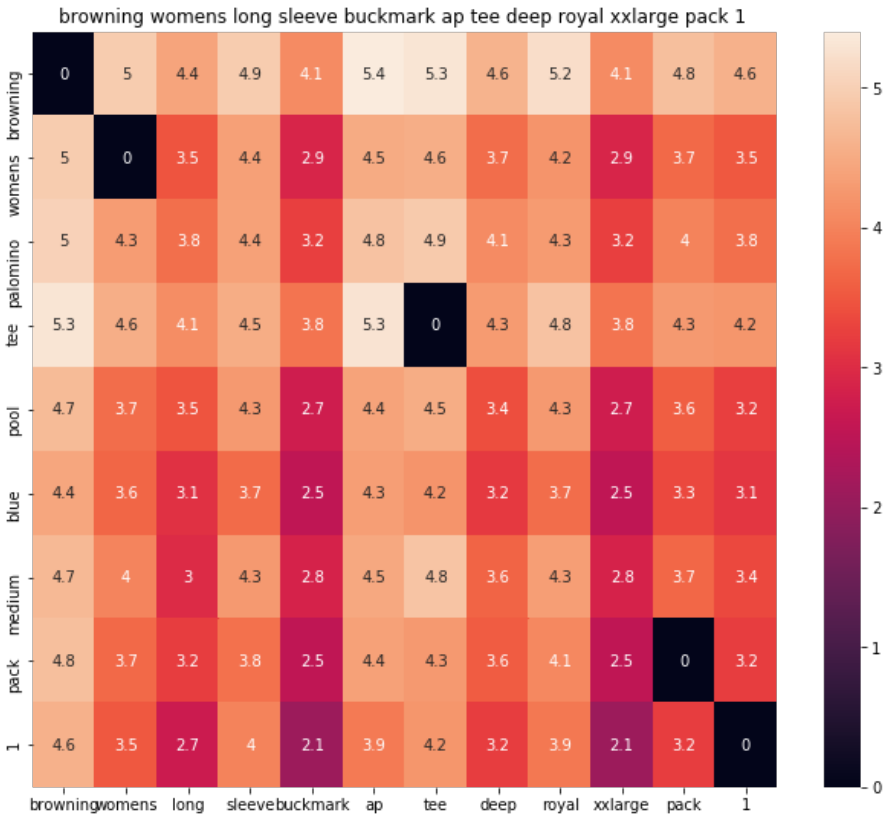


ASIN : B00NQFH7MA
BRAND : Browning
euclidean distance from given input image : 0.8161413
==========

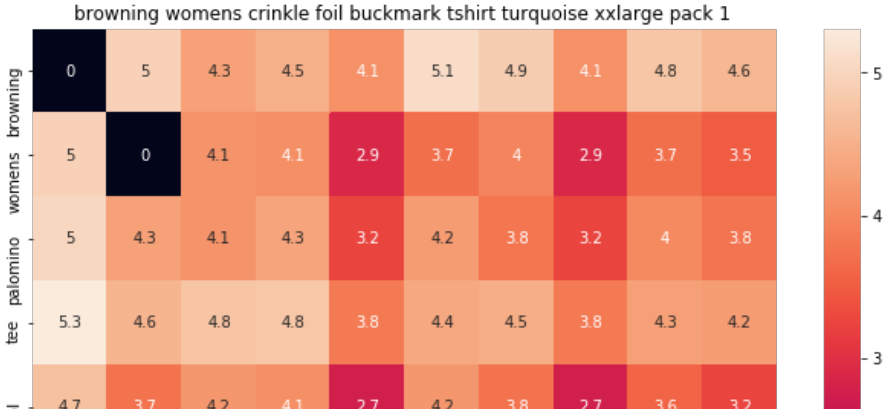


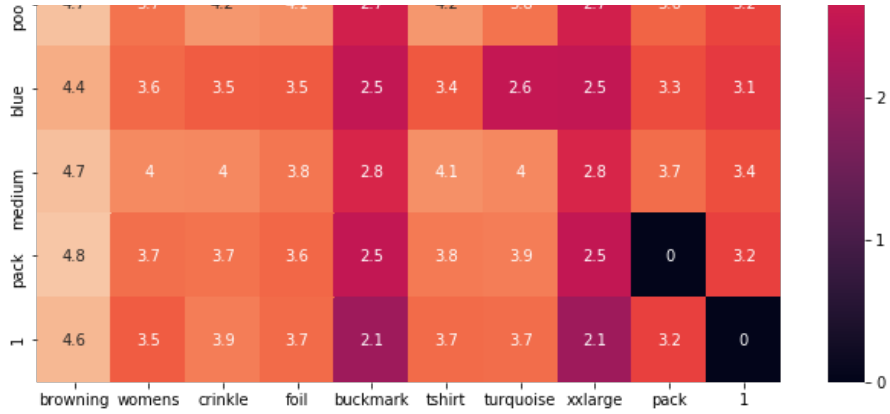


ASIN : B074KPXCPC
BRAND : Browning
euclidean distance from given input image : 0.8277478
==========

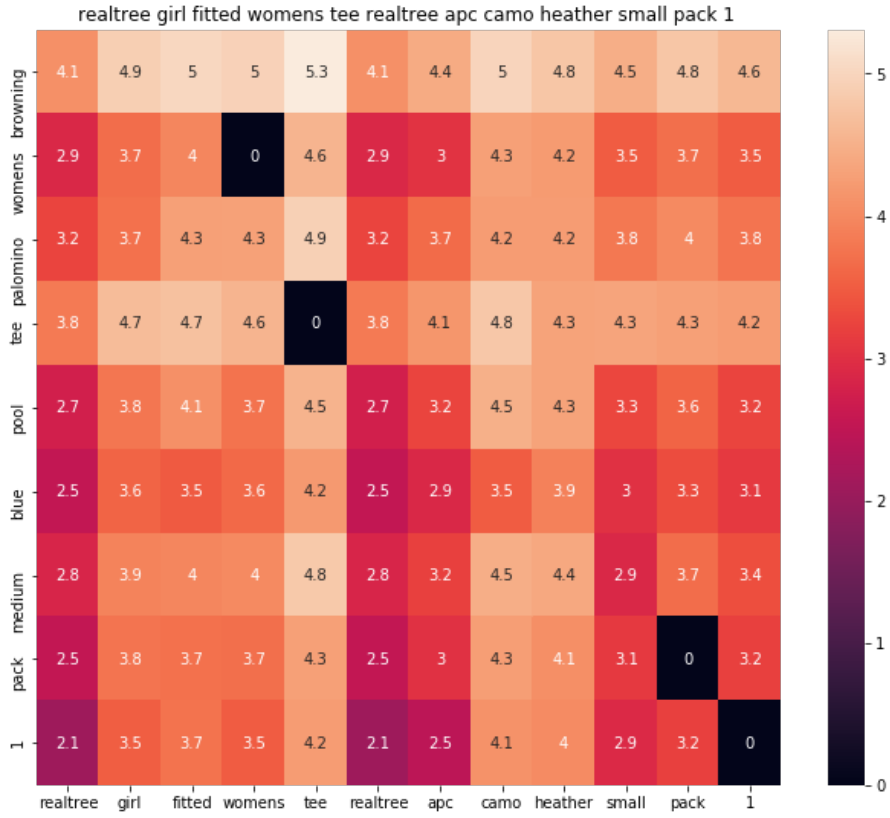


ASIN : B00QMT5KPS
BRAND : Browning
euclidean distance from given input image : 0.8409315
==========

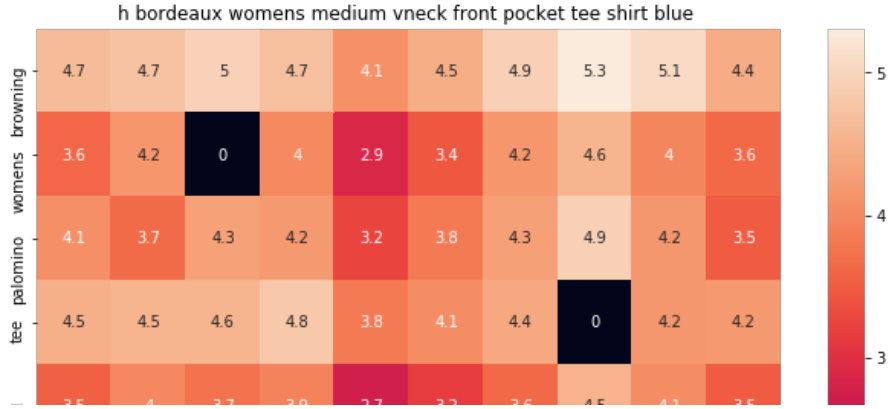


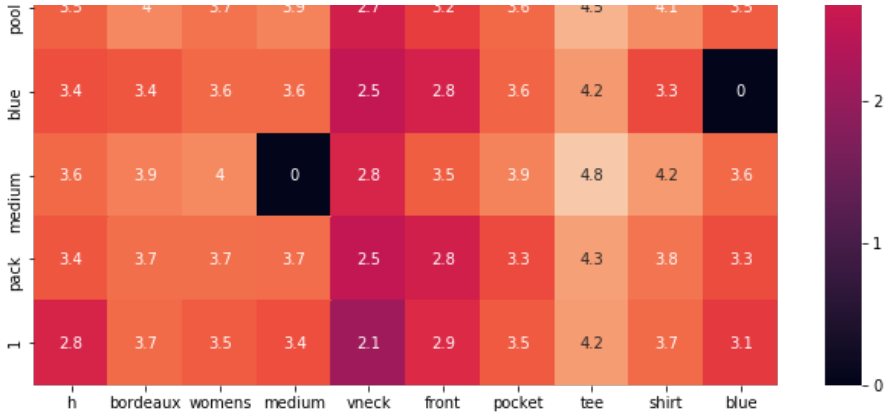


ASIN : B074KQGXQD
BRAND : Browning
euclidean distance from given input image : 0.85859907
==========

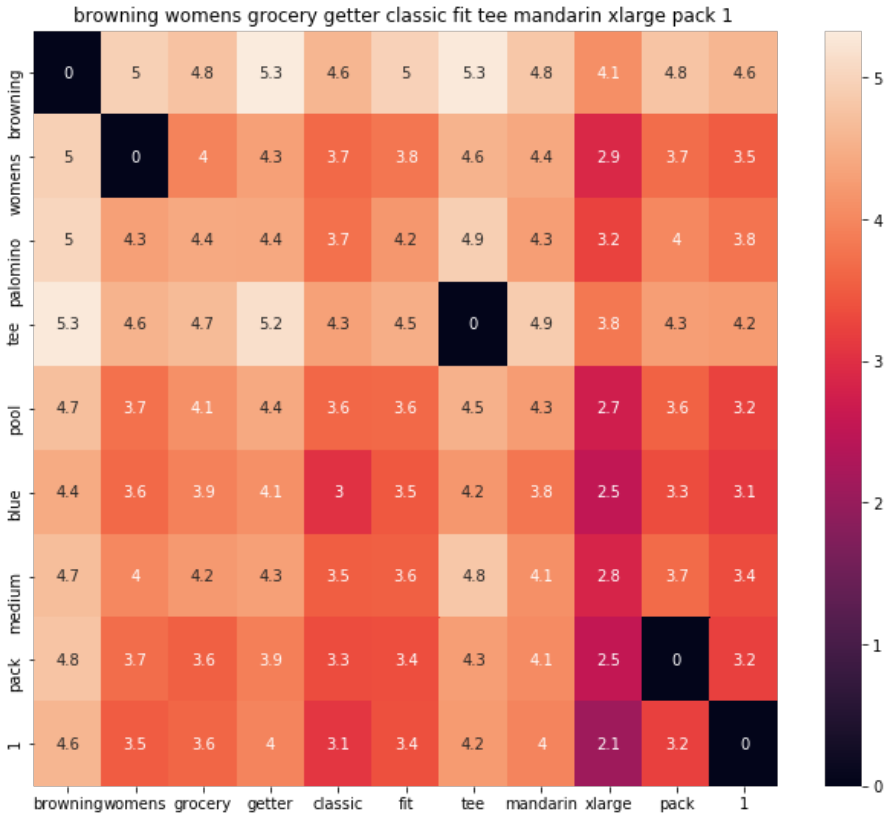


ASIN : B074KPGGMW
BRAND : Realtree Girl
euclidean distance from given input image : 0.88400334
==========

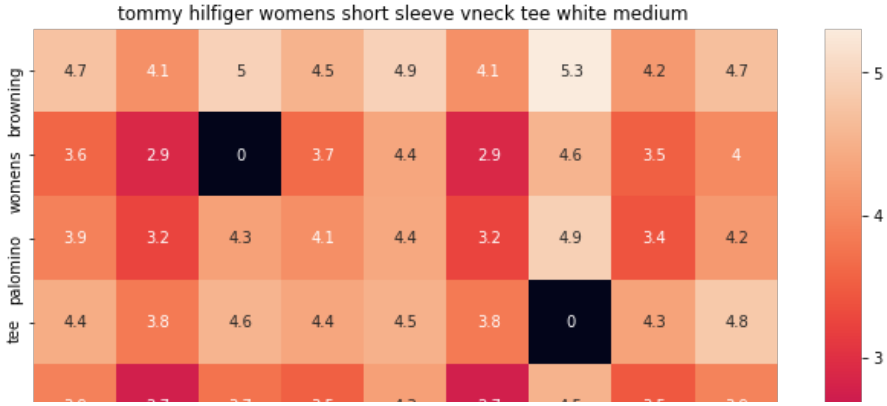


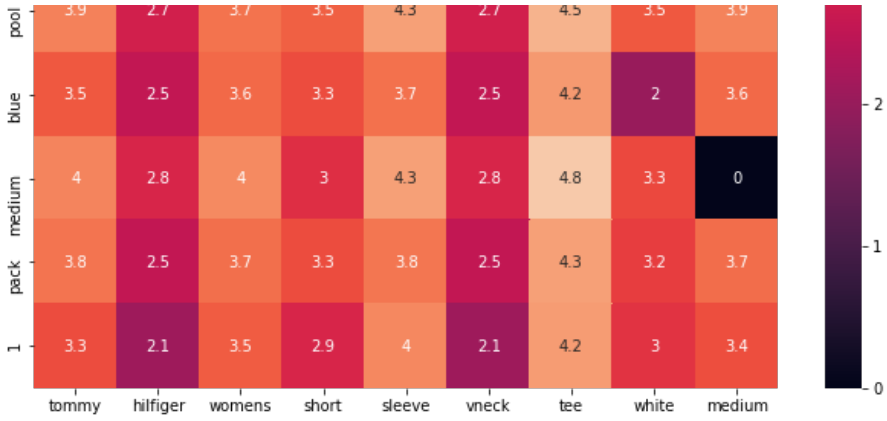


ASIN : B073378HMT
BRAND : H By Bordeaux
euclidean distance from given input image : 0.88861364
==========

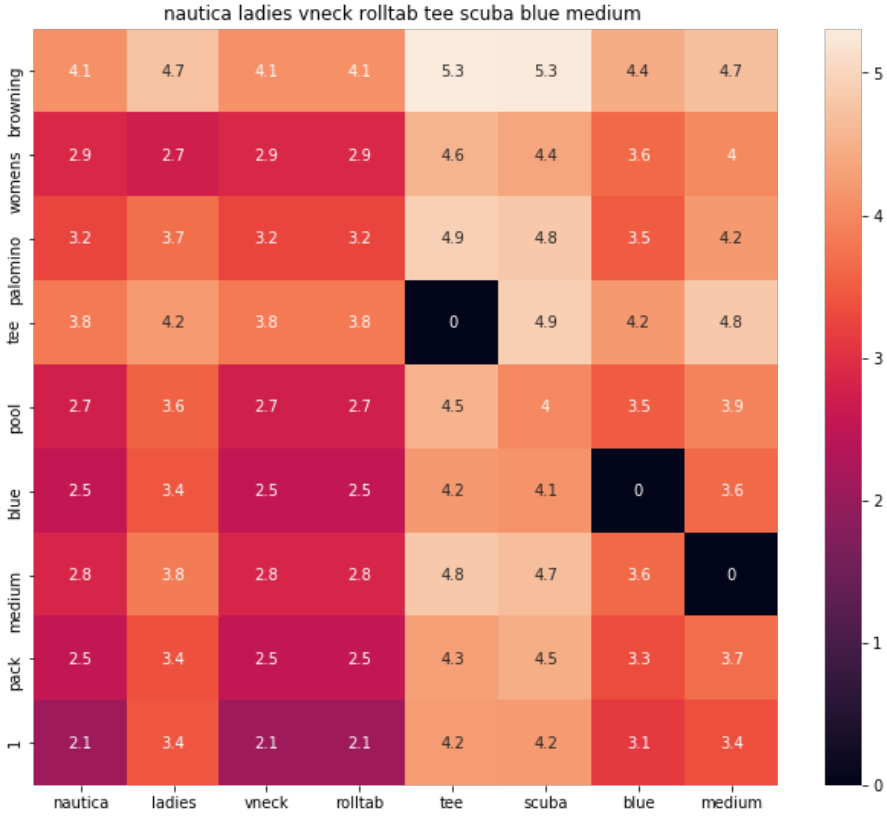


ASIN : B00QMSXHY0
BRAND : Browning
euclidean distance from given input image : 0.89360654
==========

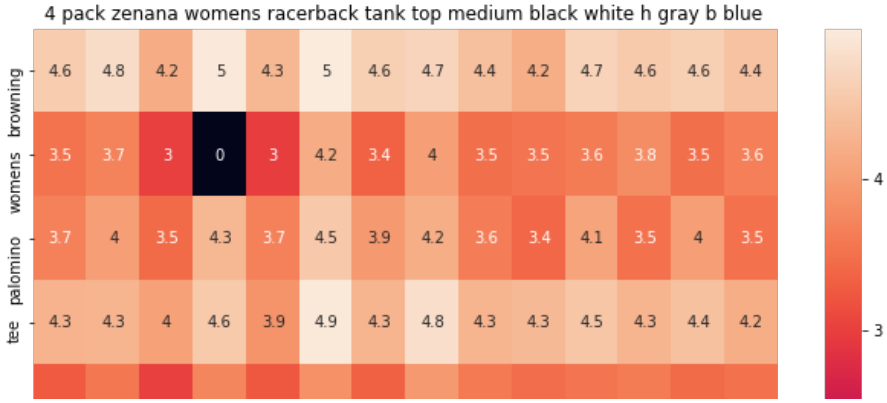


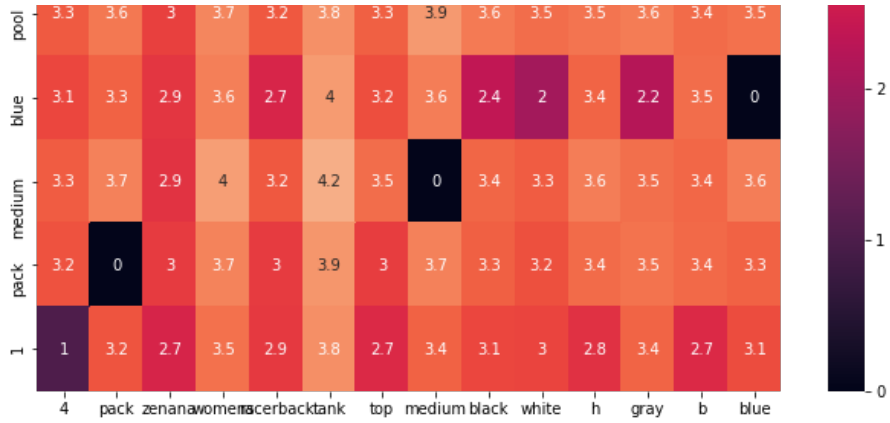


ASIN : B06XG9WD8B
BRAND : Tommy Hilfiger
euclidean distance from given input image : 0.89403474
==========

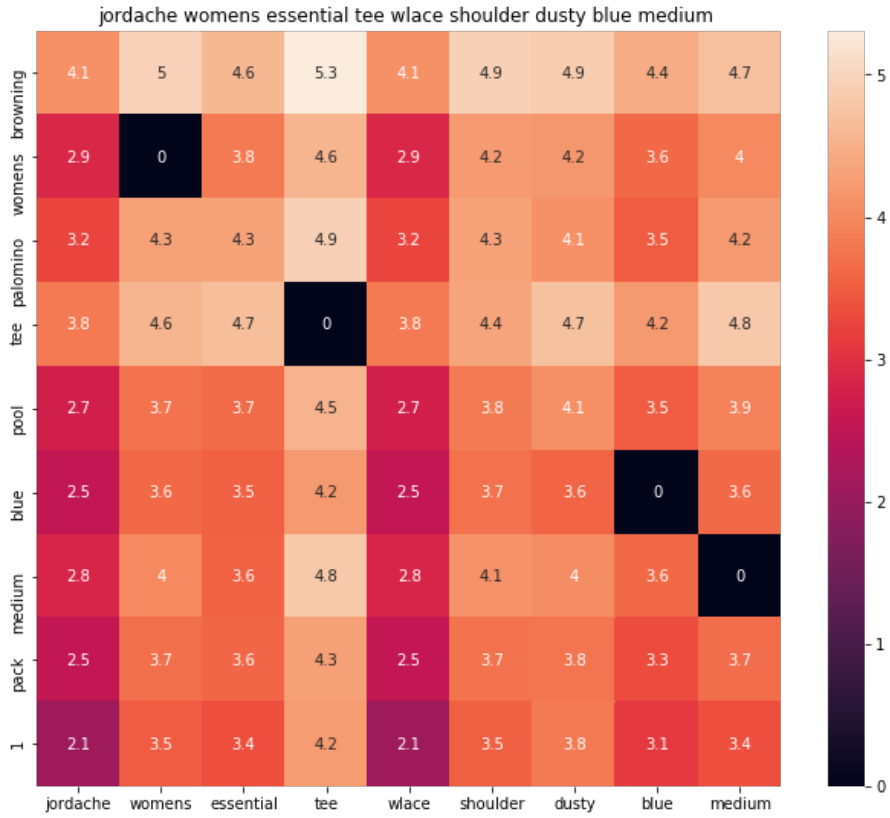


ASIN : B01M0L3O6T
BRAND : Nautica
euclidean distance from given input image : 0.9012432
==========





ASIN : B01ESA6WKE
BRAND : Zenana Outfitters
euclidean distance from given input image : 0.90381825
==========



ASIN : B01KFMCVXG
BRAND : Jordache
euclidean distance from given input image : 0.90402347
==========

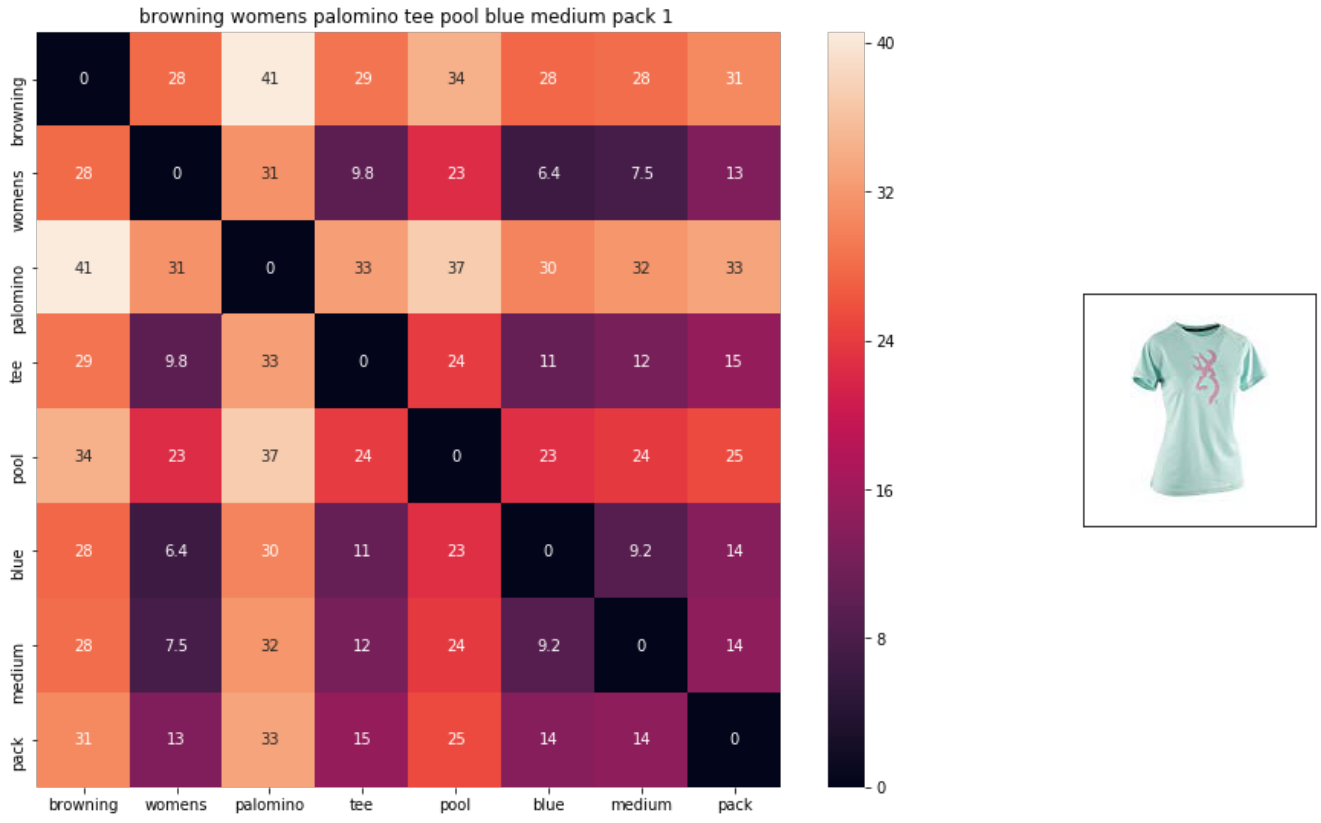
[9.4] IDF weighted Word2Vec for product similarity

In [61]:

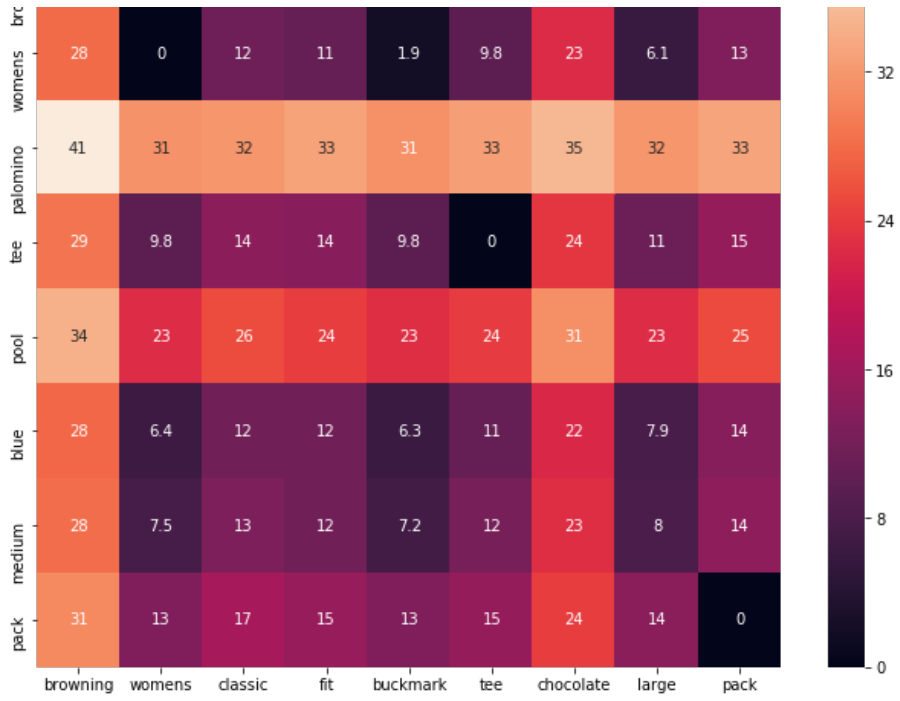
```
doc_id = 0
w2v_title_weight = []
# for every title we build a weighted vector representation
for i in data['title']:
    w2v_title_weight.append(build_avg_vec(i, 300, doc_id,'weighted'))
    doc_id += 1
# w2v_title = np.array(# number of doc in courpus * 300), each row corresponds to a doc
w2v_title_weight = np.array(w2v_title_weight)
```

In [62]:

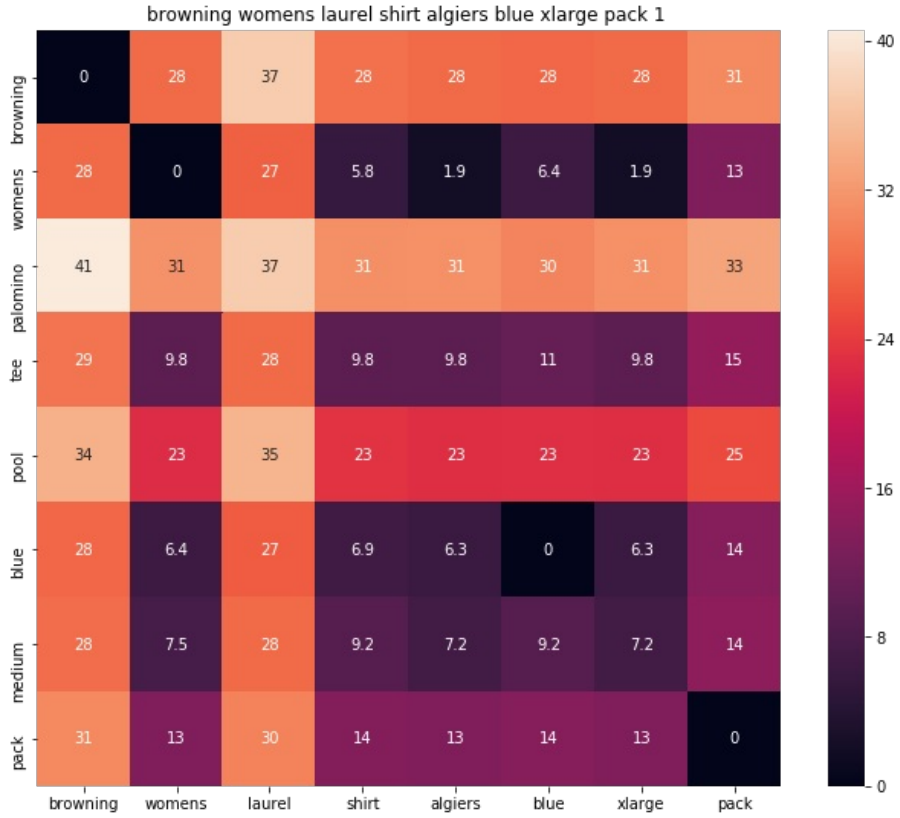
```
def weighted_w2v_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

    # pairwise_dist will store the distance from given input apparel to all remaining apparels
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y) = <X, Y> / (||X
||*||Y||)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(w2v_title_weight, w2v_title_weight[doc_id].reshape(1,-1))

    # np.argsort will return indices of 9 smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the 9 smallest distances
    pdists  = np.sort(pairwise_dist.flatten())[0:num_results]

    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])

    for i in range(0, len(indices)):
        heat_map_w2v(data['title'].loc[df_indices[0]],data['title'].loc[df_indices[i]], data['mediu
m_image_url'].loc[df_indices[i]], indices[0], indices[i], 'weighted')
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('Brand :',data['brand'].loc[df_indices[i]])
        print('euclidean distance from input :', pdists[i])
        print('='*125)

weighted_w2v_model(12000, 20)
#931
#12566
# in the give heat map, each cell contains the euclidean distance between words i, j
```



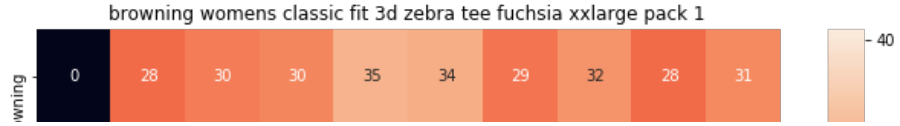

```
ASIN : B074KQ3SY1
Brand : Browning
euclidean distance from input : 0.0
=============================================================================================================
======================
```

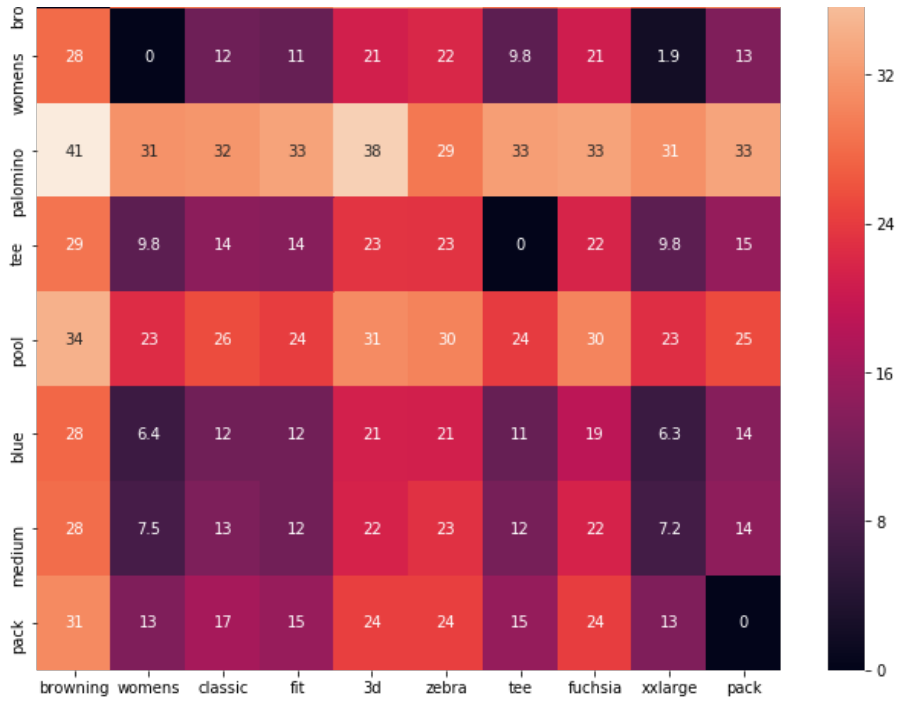


ASIN : B074KPYSRC
Brand : Browning
euclidean distance from input : 5.0235777

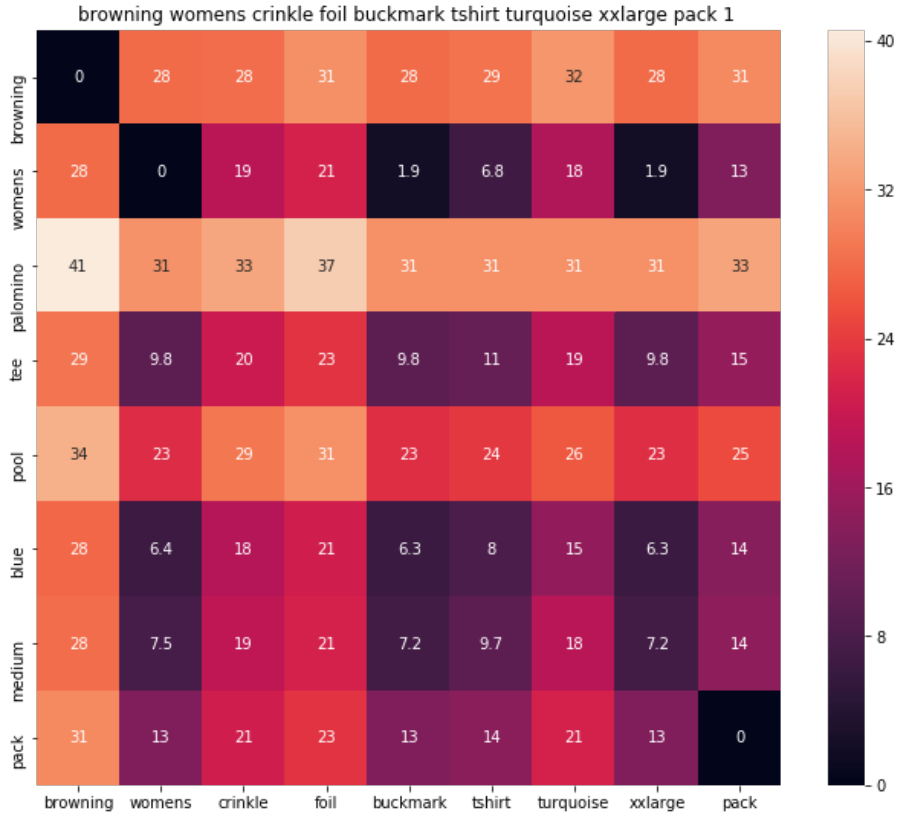


ASIN : B01KAYFT7O
Brand : SPG Outdoors
euclidean distance from input : 5.1092153

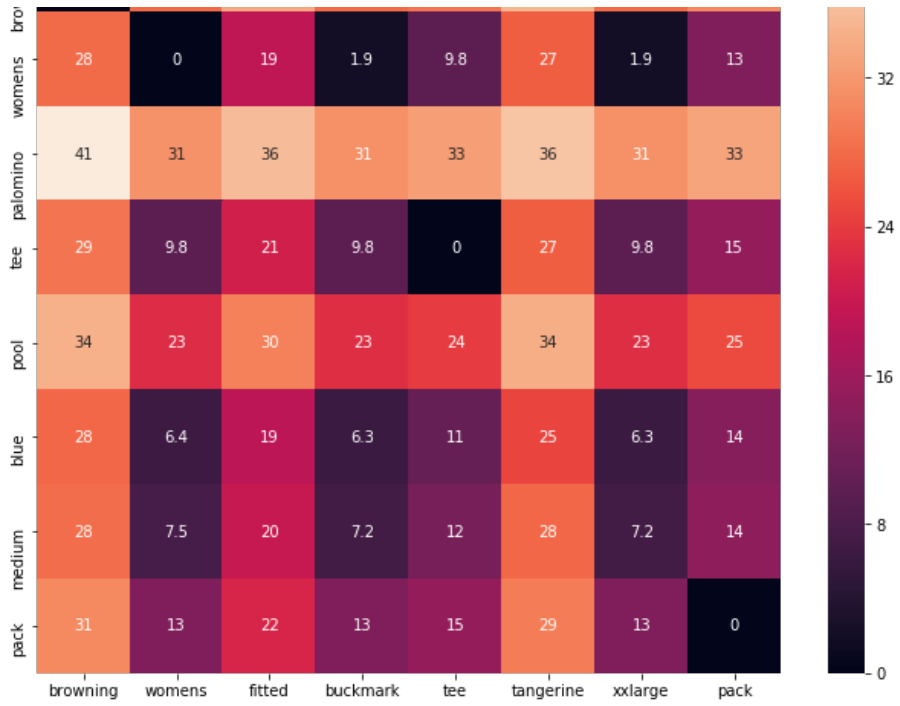




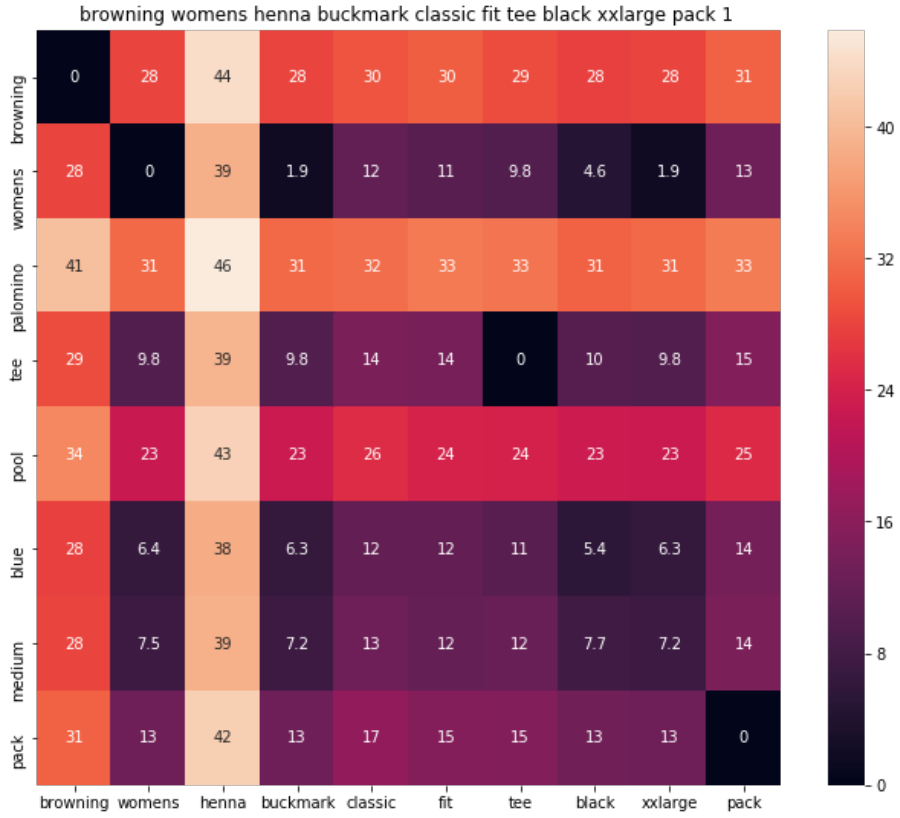
ASIN : B074KQ9P3W
Brand : Browning
euclidean distance from input : 5.176423



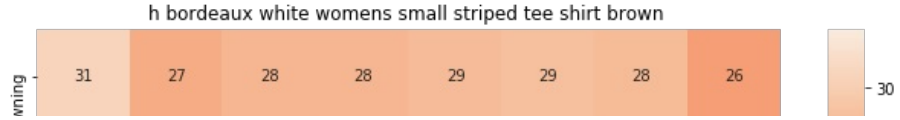
ASIN : B074KQGXQD
Brand : Browning
euclidean distance from input : 5.2815127

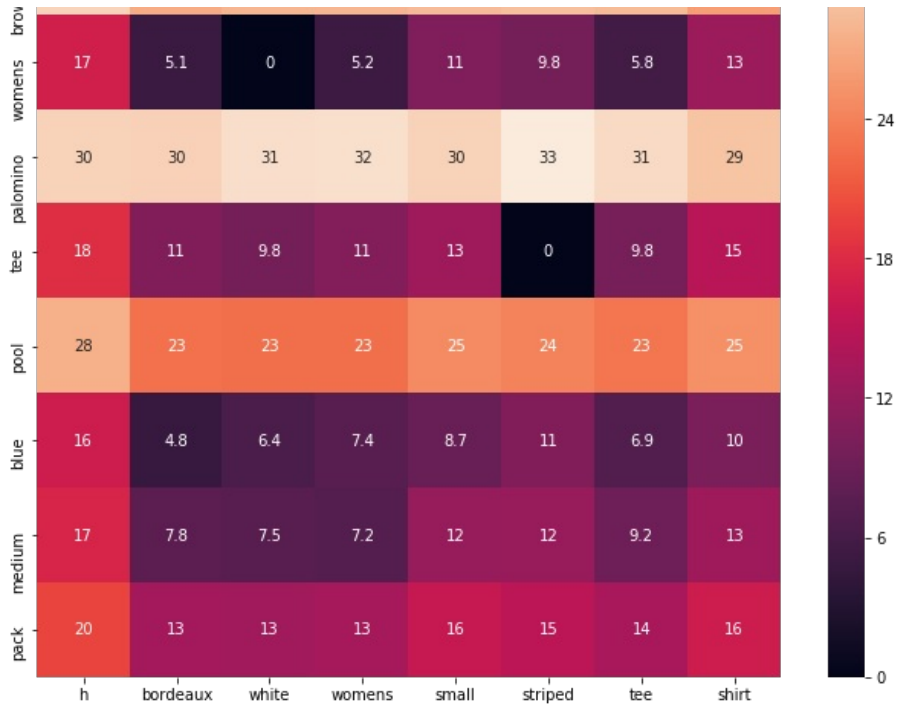


ASIN : B074KQNC89
Brand : Browning
euclidean distance from input : 5.3695297
==========

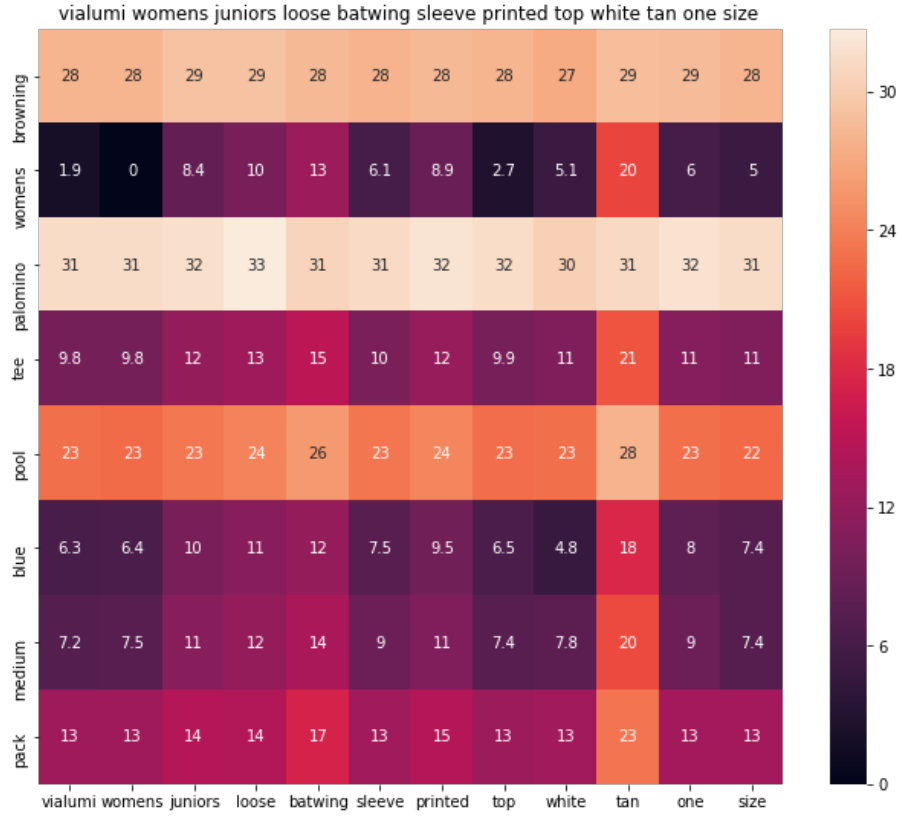


ASIN : B00NQFC32Y
Brand : Browning
euclidean distance from input : 5.3812003
==========

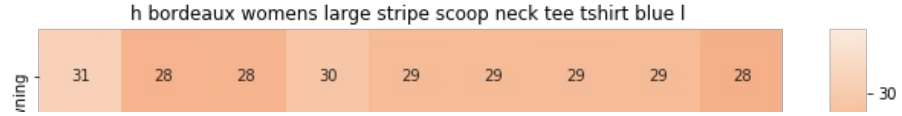




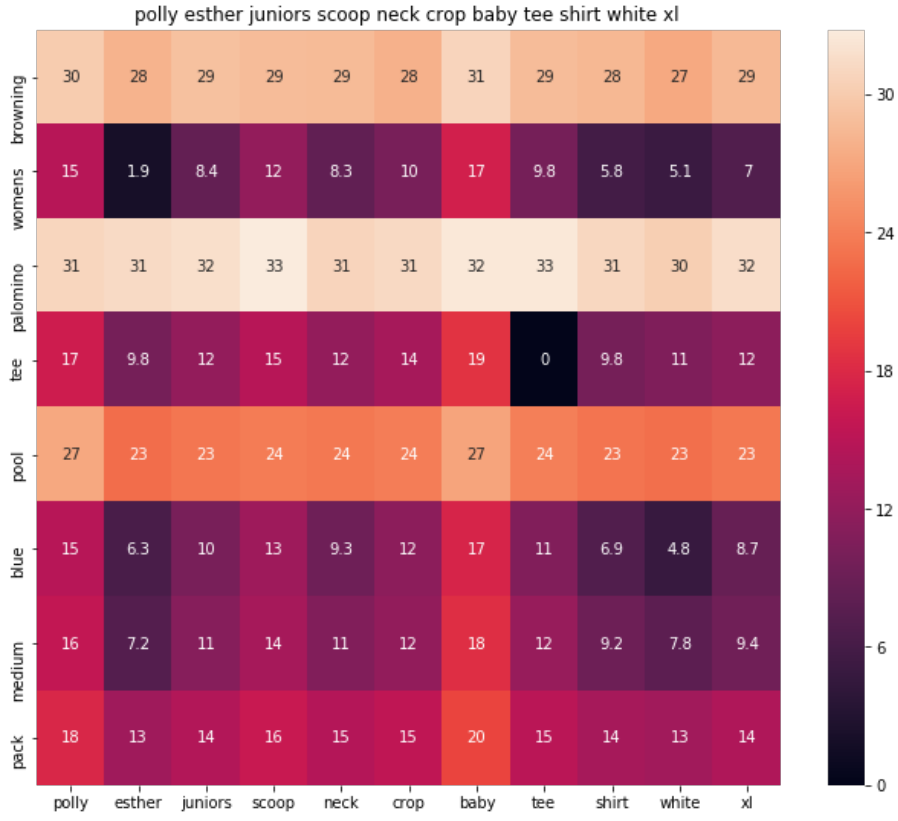
ASIN : B072BVB47Z
Brand : H By Bordeaux
euclidean distance from input : 5.4222302



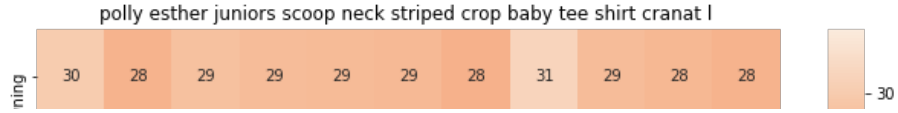
ASIN : B01BN4F3VW
Brand : Vialumi
euclidean distance from input : 5.474339

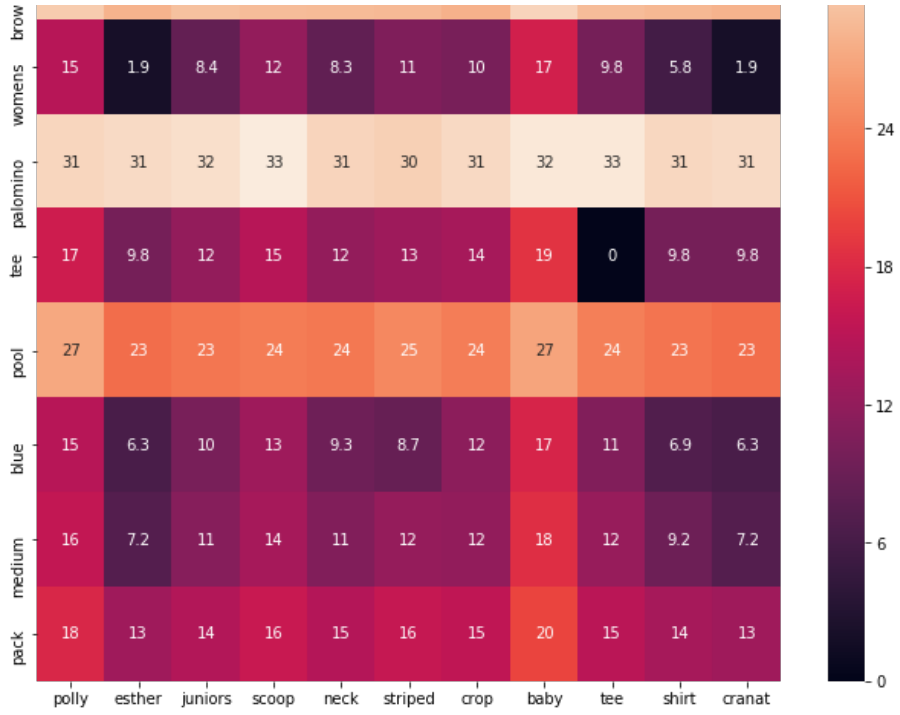


ASIN : B072LTMQN8
Brand : H By Bordeaux
euclidean distance from input : 5.4839187

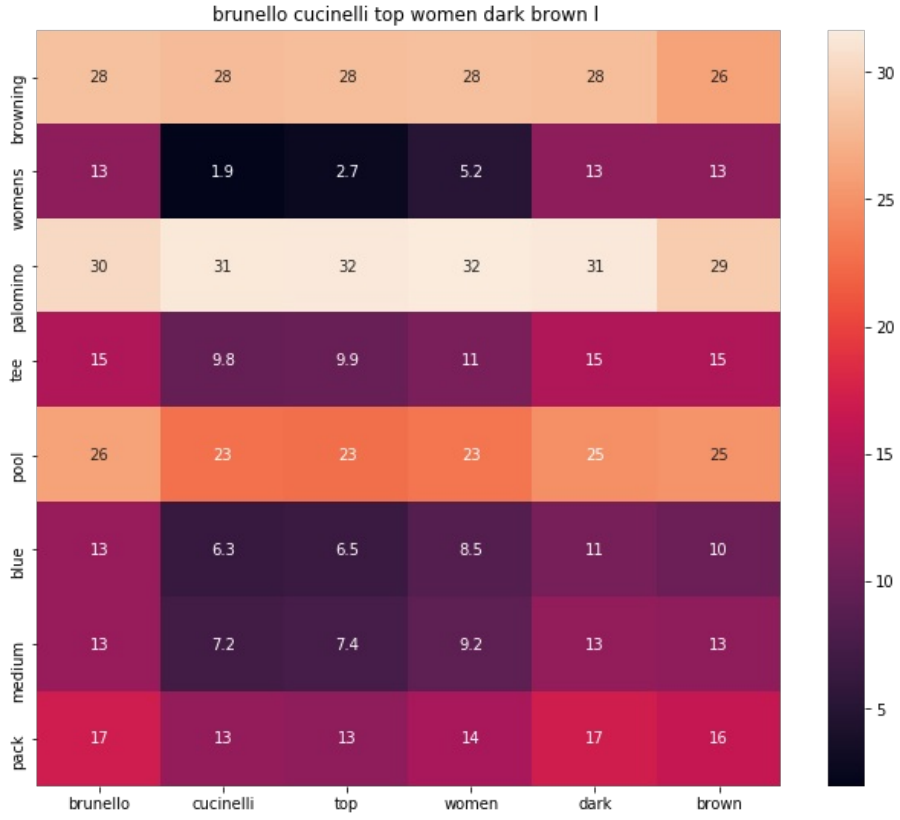


ASIN : B074ZJPQ3Z
Brand : Polly Esther
euclidean distance from input : 5.494056

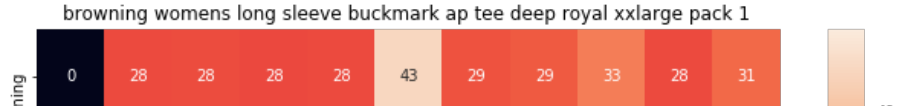




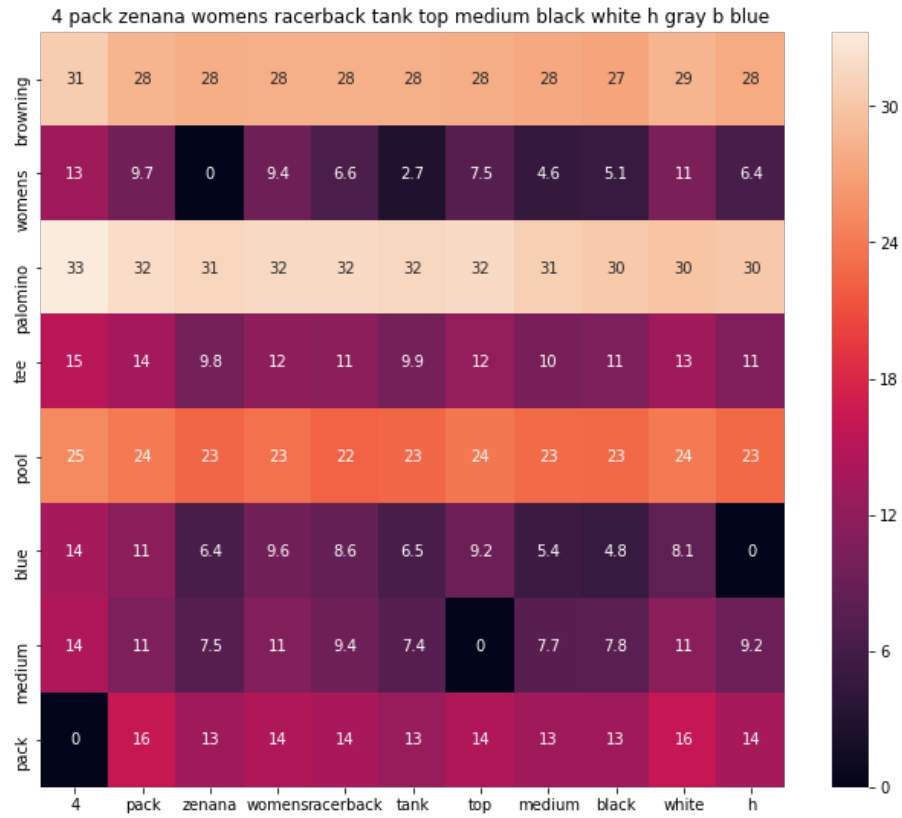
ASIN : B074ZJXZY3
Brand : Polly Esther
euclidean distance from input : 5.514052



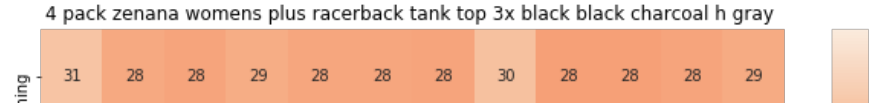
ASIN : B06X8Z1431
Brand : Brunello Cucinelli
euclidean distance from input : 5.5188174

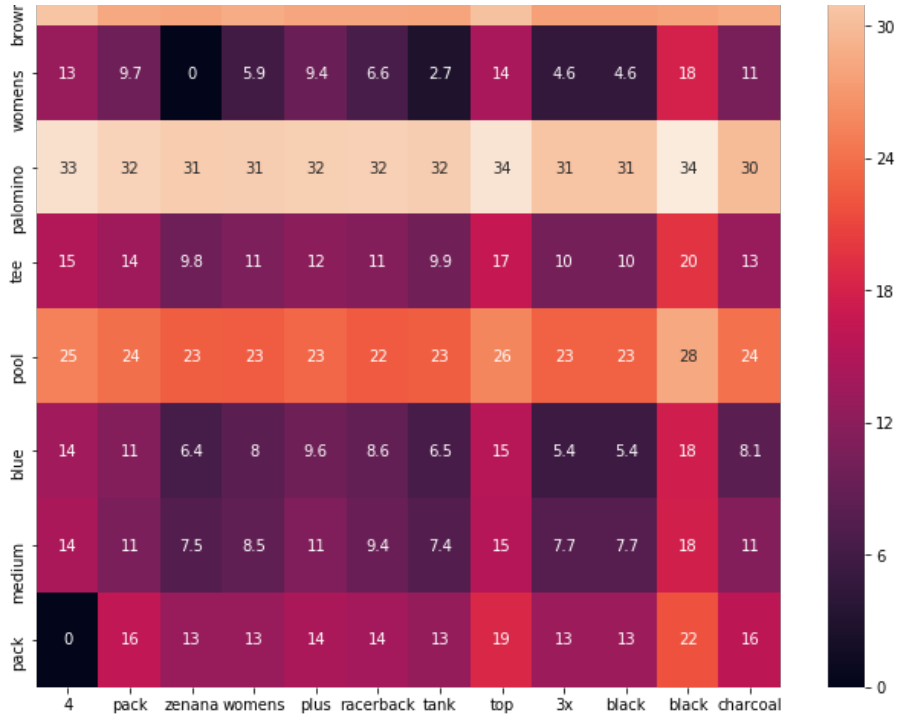


ASIN : B00QMT5KPS
Brand : Browning
euclidean distance from input : 5.5298977

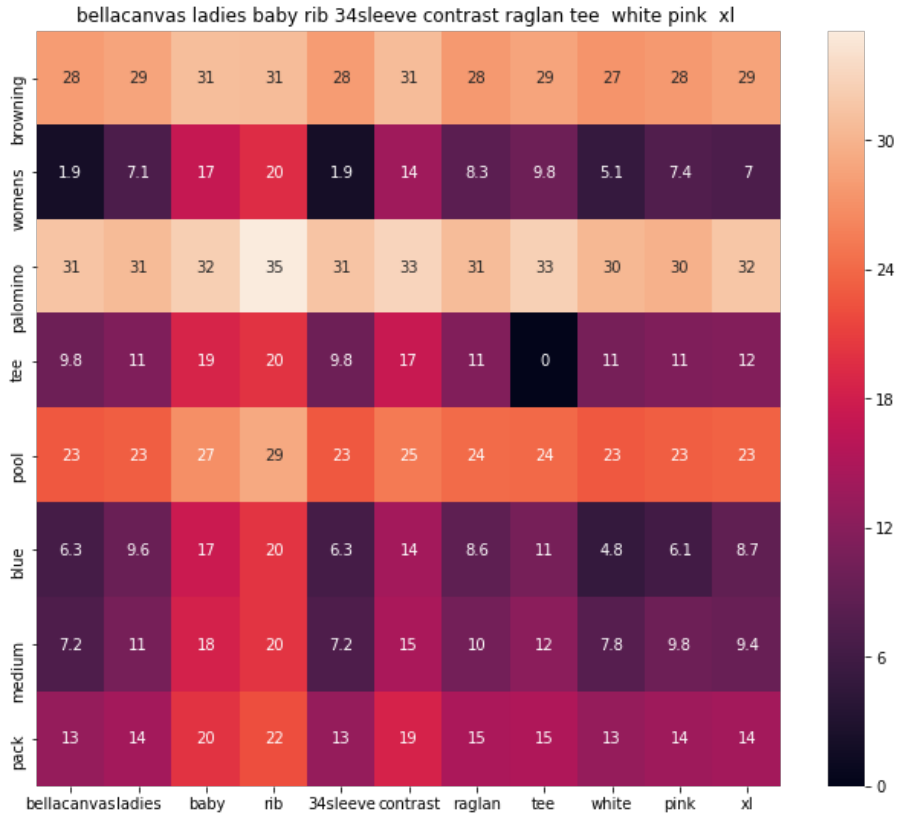


ASIN : B01ESA6WKE
Brand : Zenana Outfitters
euclidean distance from input : 5.529905

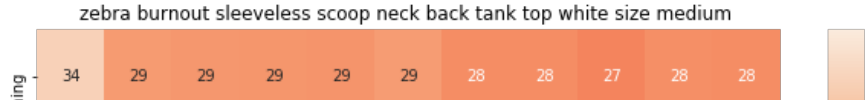


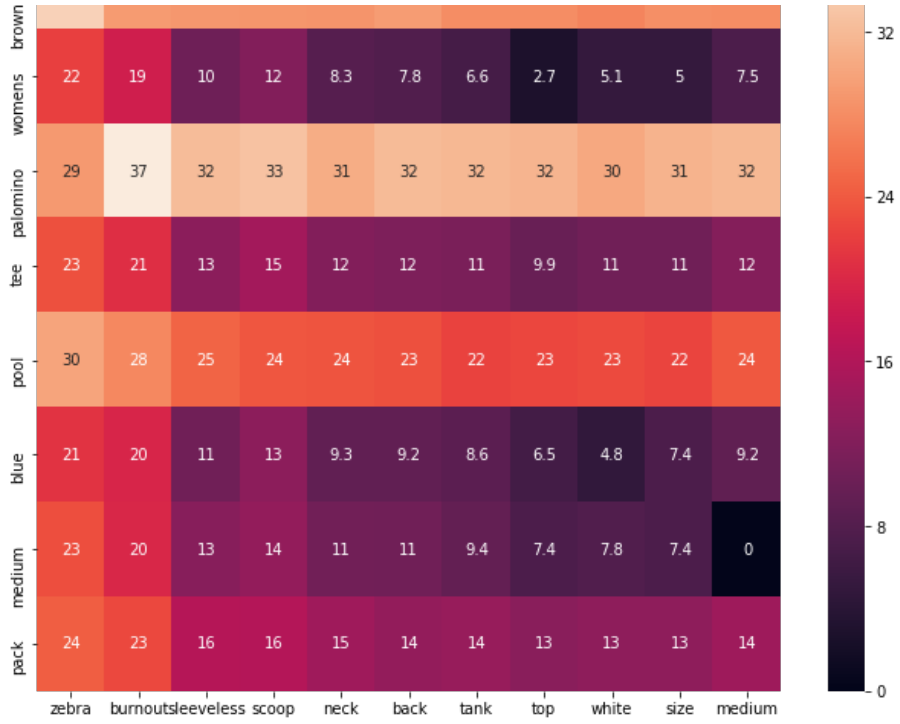


ASIN : B01ESA57Q4
Brand : Zenana Outfitters
euclidean distance from input : 5.548828

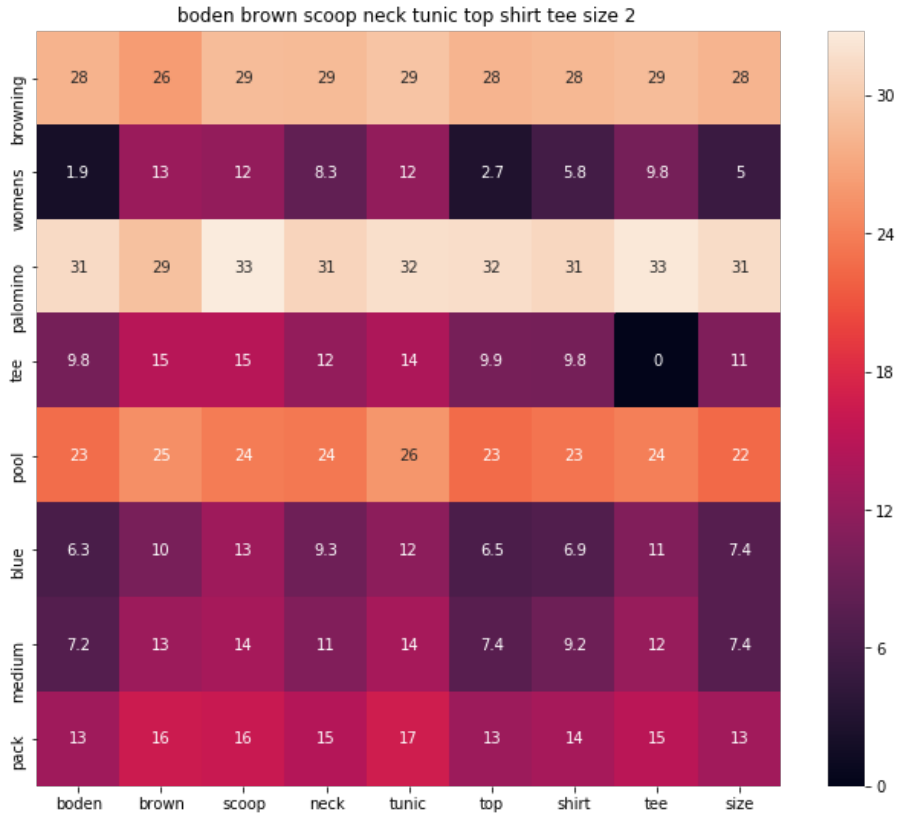


ASIN : B00E6Z1NMY
Brand : Leadoff
euclidean distance from input : 5.5523267

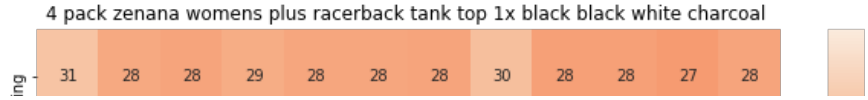




ASIN : B06XGYTZ49
Brand : Concert
euclidean distance from input : 5.5526686



ASIN : B0718ZFSY7
Brand : BODEN
euclidean distance from input : 5.5542603



```
ASIN : B01ESA5G0G
Brand : Zenana Outfitters
euclidean distance from input : 5.5769234
================================================================================================================
======================
```

[9.6] Weighted similarity using brand and color.

In [63]:

```
# some of the brand values are empty.
# Need to replace Null with string "NULL"
data['brand'].fillna(value="Not given", inplace=True )

# replace spaces with hypen
brands = [x.replace(" ", "-") for x in data['brand'].values]
types = [x.replace(" ", "-") for x in data['product_type_name'].values]
colors = [x.replace(" ", "-") for x in data['color'].values]

brand_vectorizer = CountVectorizer()
brand_features = brand_vectorizer.fit_transform(brands)

type_vectorizer = CountVectorizer()
type_features = type_vectorizer.fit_transform(types)

color_vectorizer = CountVectorizer()
color_features = color_vectorizer.fit_transform(colors)

extra_features = hstack((brand_features, type_features, color_features)).tocsr()
```

In [64]:

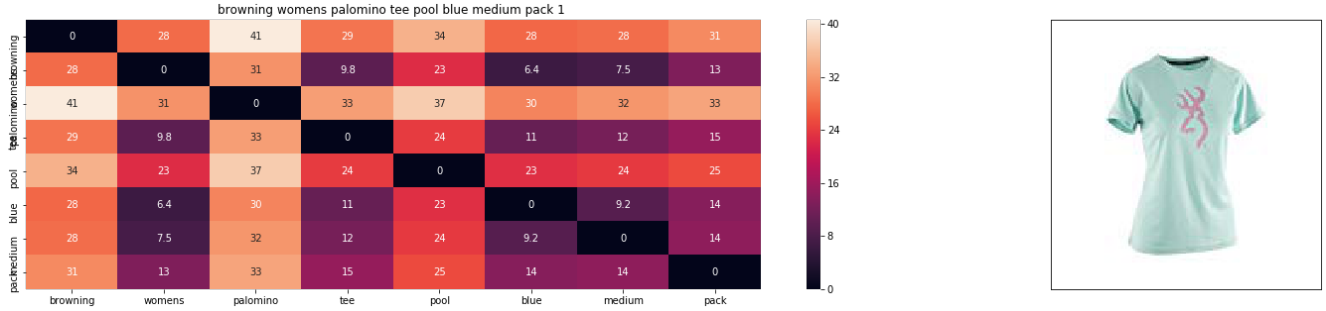
```
def heat_map_w2v_brand(sentance1, sentance2, url, doc_id1, doc_id2, df_id1, df_id2, model):

    # sentance1 : title1, input apparel
    # sentance2 : title2, recommended apparel
    # url: apparel image url
    # doc_id1: document id of input apparel
    # doc_id2: document id of recommended apparel
    # df_id1: index of document1 in the data frame
    # df_id2: index of document2 in the data frame
    # model: it can have two values, 1. avg 2. weighted

    #s1_vec = np.array(#number_of_words_title1 * 300), each row is a vector(weighted/avg) of lengt
h 300 corresponds to each word in give title
```

```
    s1_vec = get_word_vec(sentance1, doc_id1, model)
    #s2_vec = np.array(#number_of_words_title2 * 300), each row is a vector(weighted/avg) of lengt
h 300 corresponds to each word in give title
    s2_vec = get_word_vec(sentance2, doc_id2, model)

    # s1_s2_dist = np.array(#number of words in title1 * #number of words in title2)
    # s1_s2_dist[i,j] = euclidean distance between words i, j
    s1_s2_dist = get_distance(s1_vec, s2_vec)

    data_matrix = [['Asin','Brand', 'Color', 'Product type'],
                   [data['asin'].loc[df_id1],brands[doc_id1], colors[doc_id1], types[doc_id1]], # input
apparel's features
                   [data['asin'].loc[df_id2],brands[doc_id2], colors[doc_id2], types[doc_id2]]] #
recommonded apparel's features

    colorscale = [[0, '#1d004d'],[.5, '#f2e5ff'],[1, '#f2e5d1']] # to color the headings of each cc
lumn

    # we create a table with the data_matrix
    table = ff.create_table(data_matrix, index=True, colorscale=colorscale)
    # plot it with plotly
    plotly.offline.iplot(table, filename='simple_table')

    # devide whole figure space into 25 * 1:10 grids
    gs = gridspec.GridSpec(25, 15)
    fig = plt.figure(figsize=(25,5))

    # in first 25*10 grids we plot heatmap
    ax1 = plt.subplot(gs[:, :-5])
    # ploting the heap map based on the pairwise distances
    ax1 = sns.heatmap(np.round(s1_s2_dist,6), annot=True)
    # set the x axis labels as recommended apparels title
    ax1.set_xticklabels(sentance2.split())
    # set the y axis labels as input apparels title
    ax1.set_yticklabels(sentance1.split())
    # set title as recommended apparels title
    ax1.set_title(sentance2)

    # in last 25 * 10:15 grids we display image
    ax2 = plt.subplot(gs[:, 10:16])
    # we dont display grid lins and axis labels to images
    ax2.grid(False)
    ax2.set_xticks([])
    ax2.set_yticks([])

    # pass the url it display it
    display_img(url, ax2, fig)

    plt.show()
```

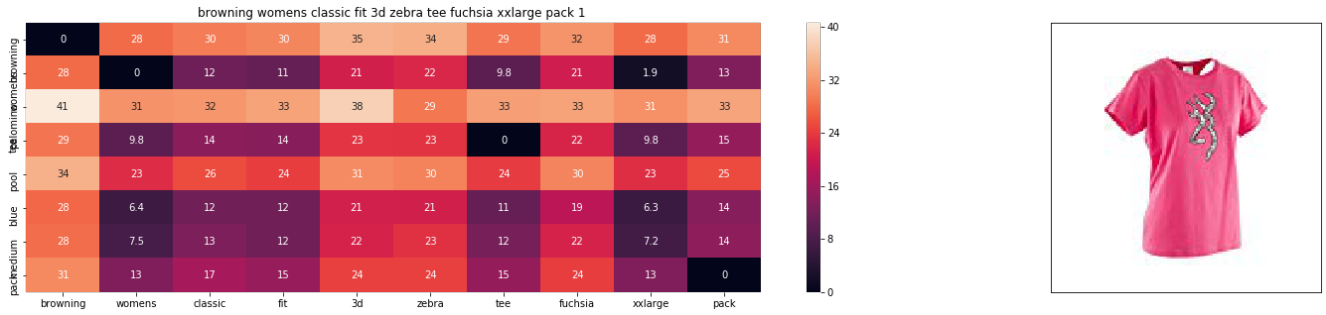
In [65]:

```
def idf_w2v_brand(doc_id, w1, w2, num_results):
    # doc_id: apparel's id in given corpus
    # w1: weight for  w2v features
    # w2: weight for brand and color features

    # pairwise_dist will store the distance from given input apparel to all remaining apparels
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y) = <X, Y> / (||X
||*||Y||)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    idf_w2v_dist  = pairwise_distances(w2v_title_weight, w2v_title_weight[doc_id].reshape(1,-1))
    ex_feat_dist = pairwise_distances(extra_features, extra_features[doc_id])
    pairwise_dist   = (w1 * idf_w2v_dist +  w2 * ex_feat_dist)/float(w1 + w2)

    # np.argsort will return indices of 9 smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the 9 smallest distances
    pdists  = np.sort(pairwise_dist.flatten())[0:num_results]

    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])


    for i in range(0, len(indices)):
        heat_map_w2v_brand(data['title'].loc[df_indices[0]],data['title'].loc[df_indices[i]], data[
```

```
'medium_image_url'].loc[df_indices[i]], indices[0], indices[i],df_indices[0], df_indices[i], 'weigh
ted')
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('Brand :',data['brand'].loc[df_indices[i]])
        print('euclidean distance from input :', pdists[i])
        print('='*125)

idf_w2v_brand(12000, 5, 5, 20)
# in the give heat map, each cell contains the euclidean distance between words i, j
```

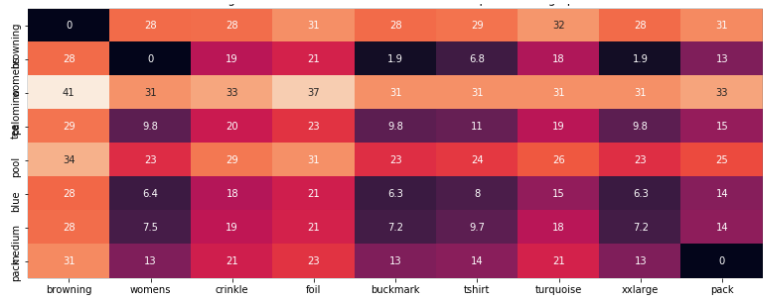



```
ASIN : B074KQ3SY1
Brand : Browning
euclidean distance from input : 0.0
=========================================================================================================
======================
```

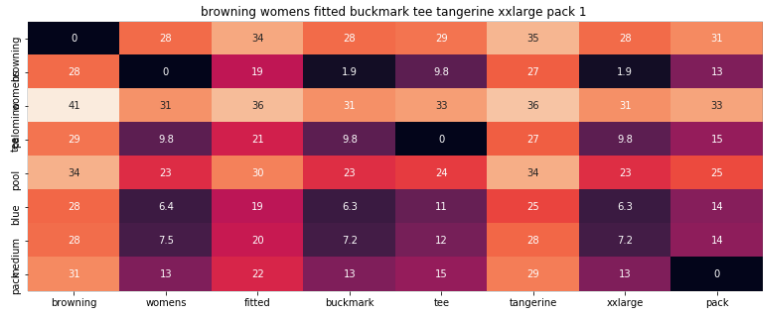



```
ASIN : B074KQ9P3W
Brand : Browning
euclidean distance from input : 3.4542368448244773
=========================================================================================================
======================
```

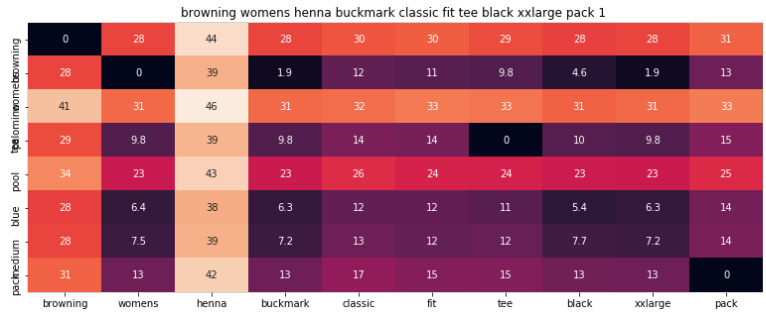
browning womens crinkle foil buckmark tshirt turquoise xxlarge pack 1



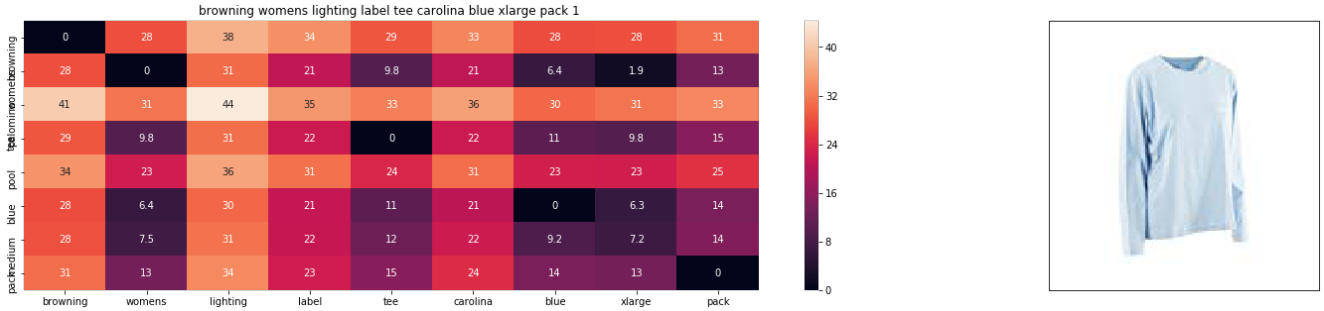
ASIN : B074KQGXQD
Brand : Browning
euclidean distance from input : 3.506781820105239



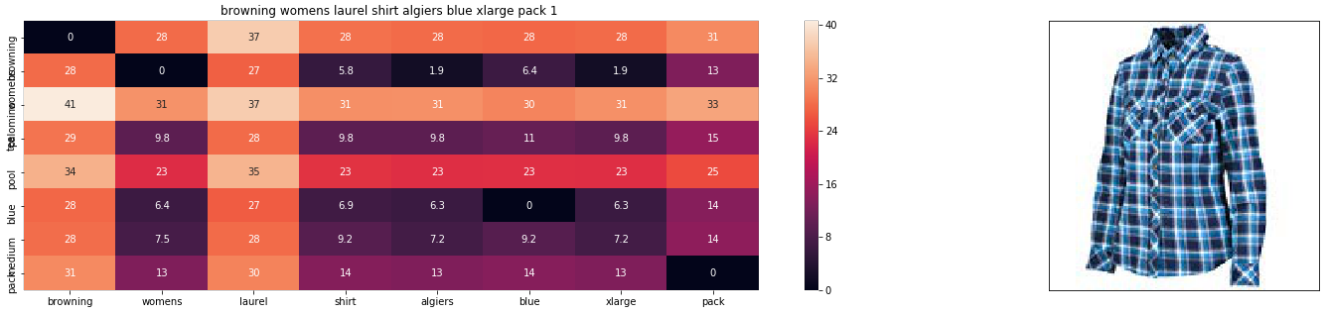
ASIN : B074KQNC89
Brand : Browning
euclidean distance from input : 3.5507902658449852



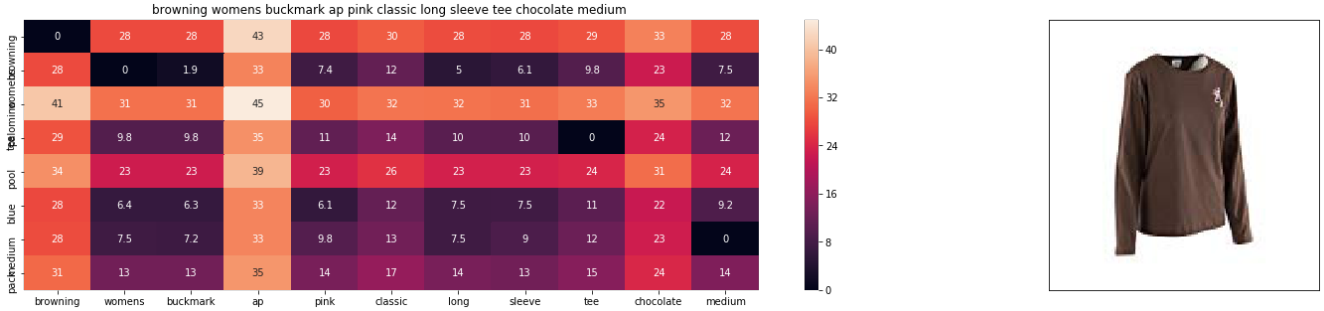
ASIN : B00NQFC32Y
Brand : Browning
euclidean distance from input : 3.5566256082522116



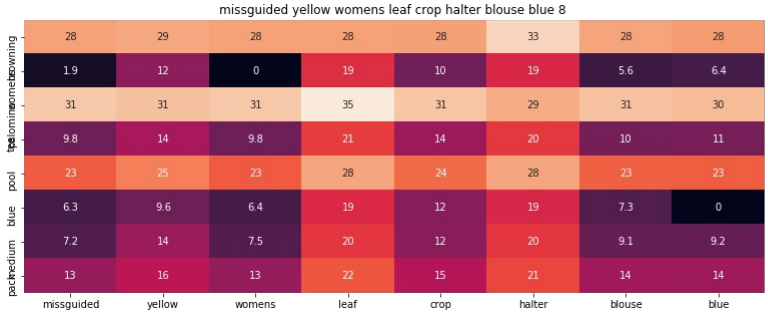
ASIN : B074KQJ8KC
Brand : Browning
euclidean distance from input : 3.612644386472192



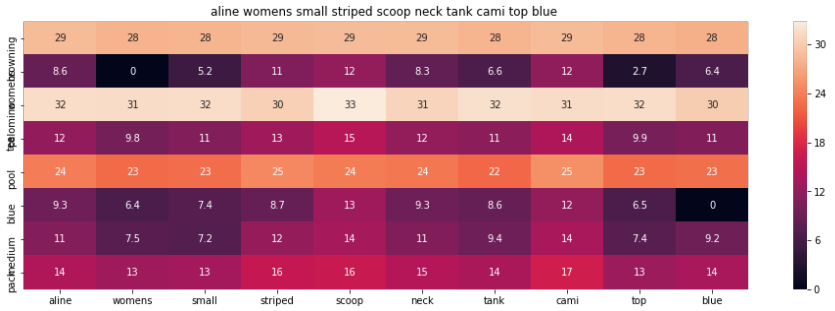
ASIN : B01KAYFT7O
Brand : SPG Outdoors
euclidean distance from input : 3.67264157084218



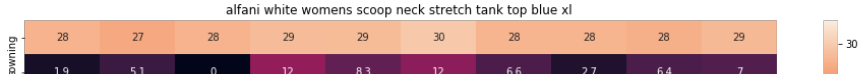
ASIN : B00NQFH7MA
Brand : Browning
euclidean distance from input : 3.677447561072036
==========

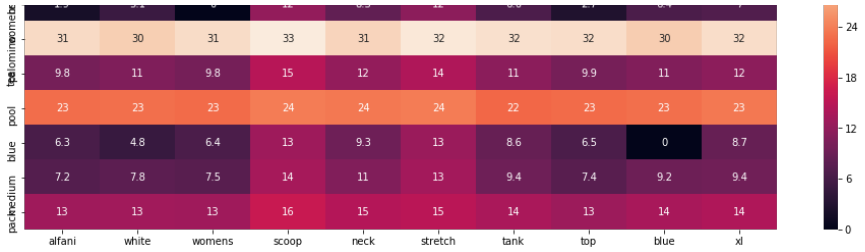


ASIN : B07147JSY5
Brand : Missguided
euclidean distance from input : 3.701518682287856
==========

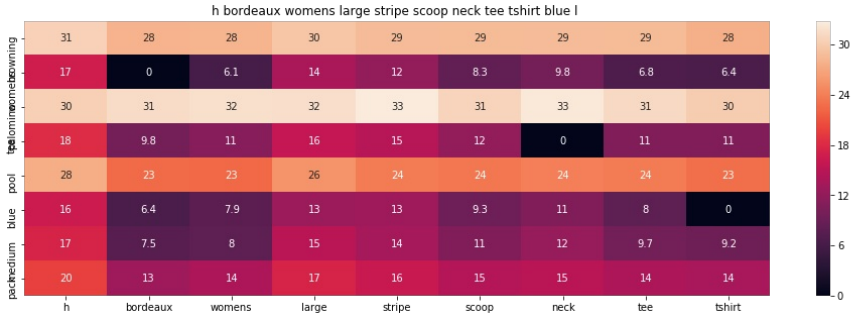


ASIN : B073V44QB8
Brand : a_line
euclidean distance from input : 3.7103458917605123
==========

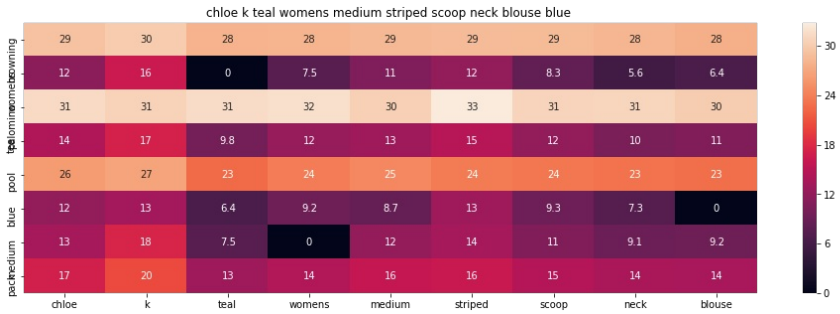




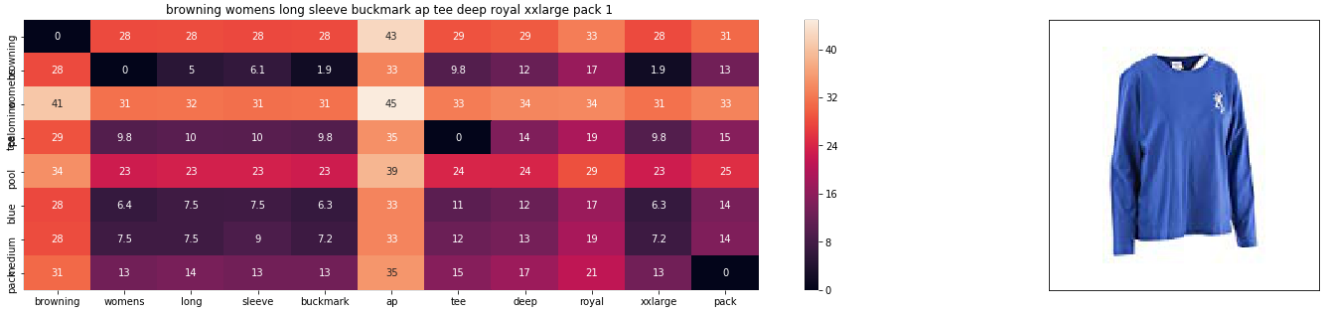
ASIN : B071K8S7WS
Brand : Alfani
euclidean distance from input : 3.716952375219985
==========



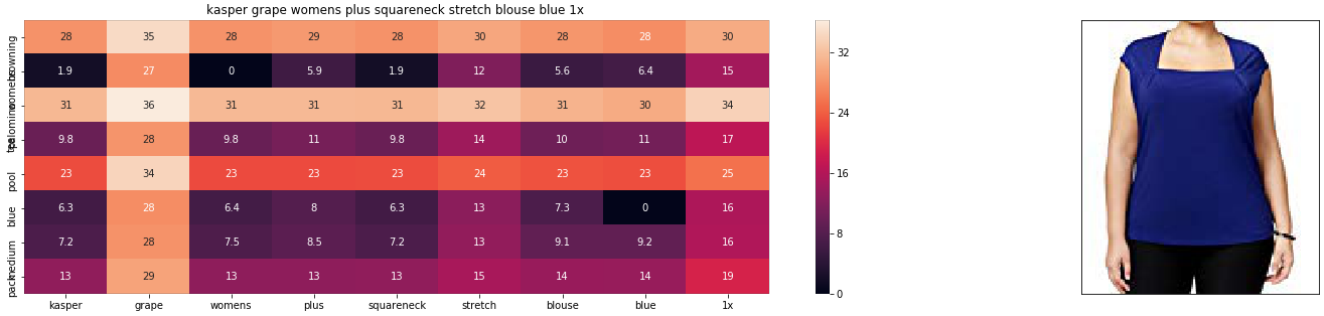
ASIN : B072LTMQN8
Brand : H By Bordeaux
euclidean distance from input : 3.7419593811035154
==========



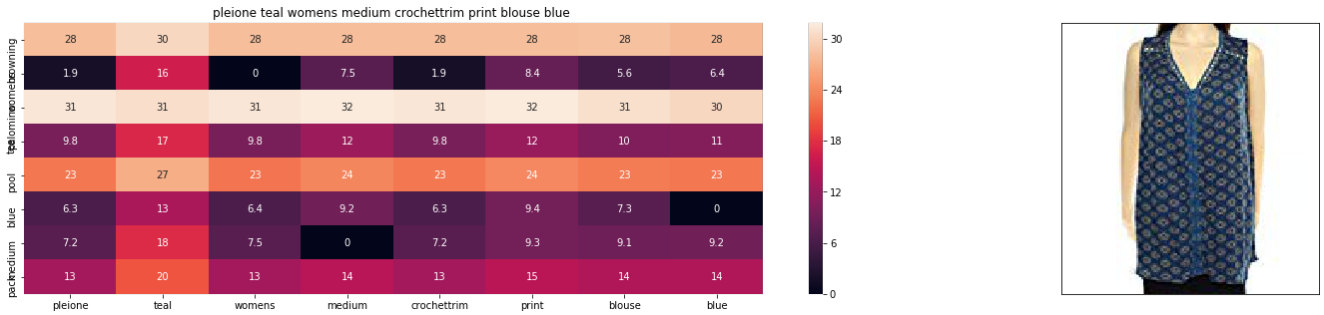
ASIN : B0725RLC8T
Brand : Chloe K.
euclidean distance from input : 3.7473917520510396
==========



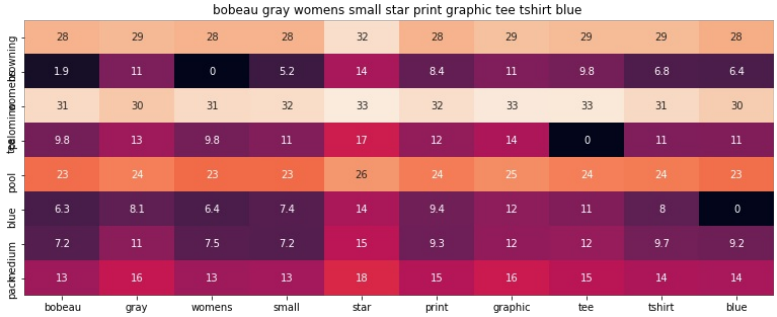
ASIN : B00QMT5KPS
Brand : Browning
euclidean distance from input : 3.764948844909668



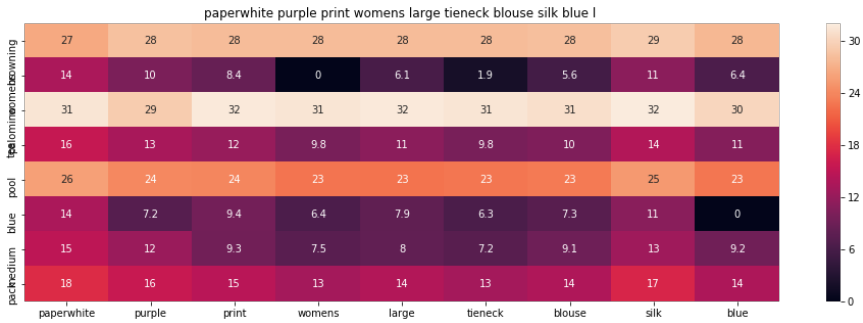
ASIN : B06ZYHTRSD
Brand : Kasper
euclidean distance from input : 3.7746510065066063



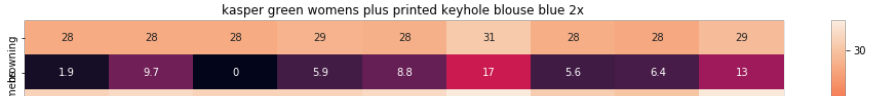
ASIN : B0753KBF1F
Brand : Pleione
euclidean distance from input : 3.7765596902834617
==========

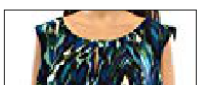


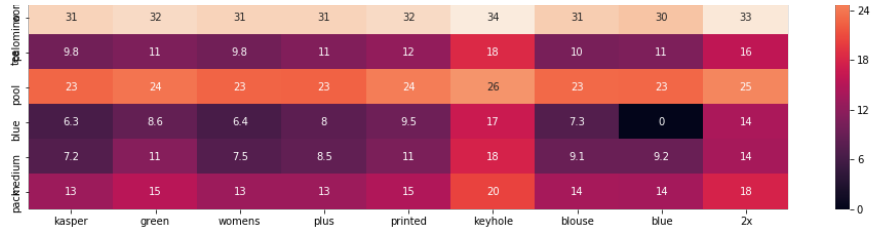
ASIN : B0757WTXX6
Brand : Bobeau
euclidean distance from input : 3.7902340448366845
==========



ASIN : B071J264ST
Brand : Paperwhite
euclidean distance from input : 3.799036649511977
==========

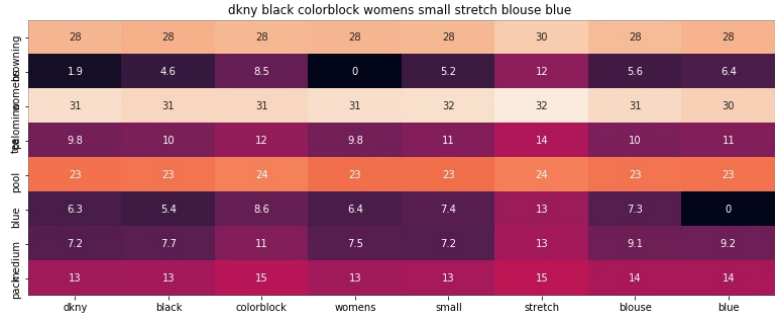






ASIN : B071D6H6S1
Brand : Kasper
euclidean distance from input : 3.807042936133071

==========

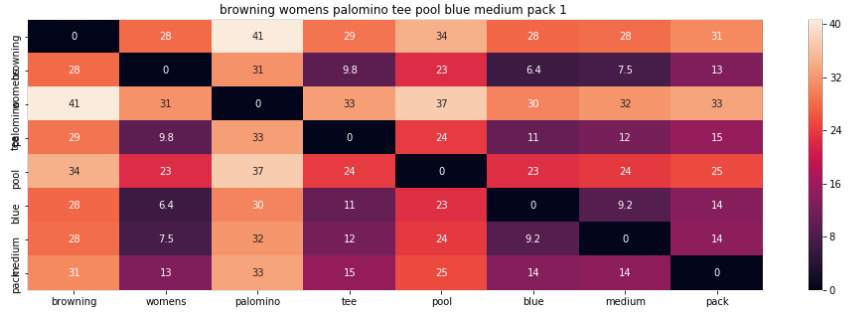


ASIN : B01J298XVW
Brand : DKNY
euclidean distance from input : 3.8127832925783833

==========

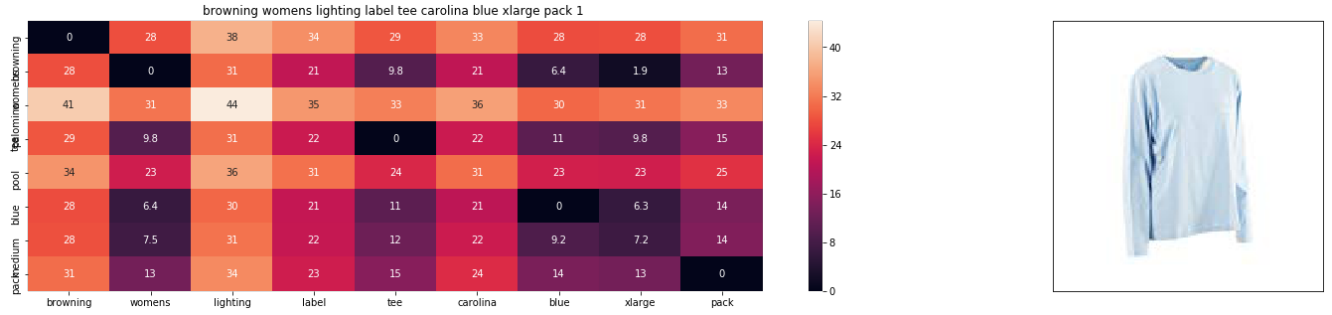
In [66]:

```
# brand and color weight =50
# title vector weight = 5

idf_w2v_brand(12000, 5, 50, 20)
```

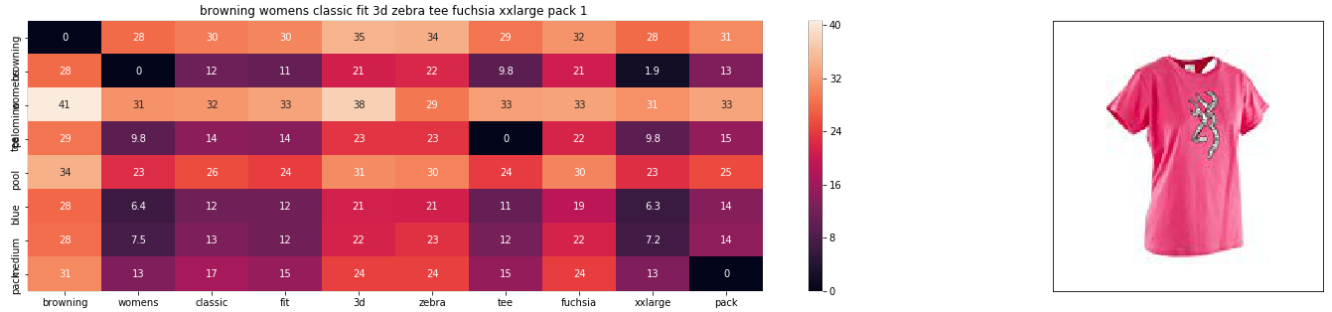


ASIN : B074KQ3SY1
Brand : Browning

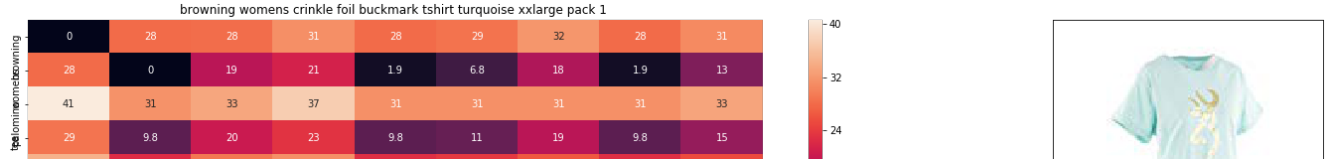
Brand : Browning
euclidean distance from input : 0.0
==========

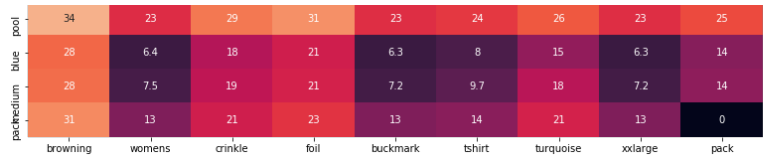


ASIN : B074KQJ8KC
Brand : Browning
euclidean distance from input : 1.81392825766384
==========

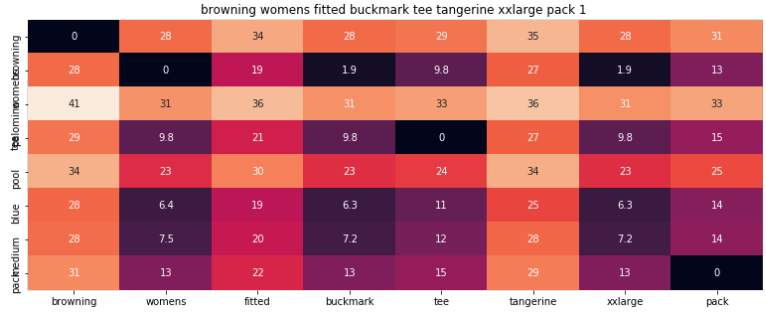


ASIN : B074KQ9P3W
Brand : Browning
euclidean distance from input : 2.04517554161535
==========

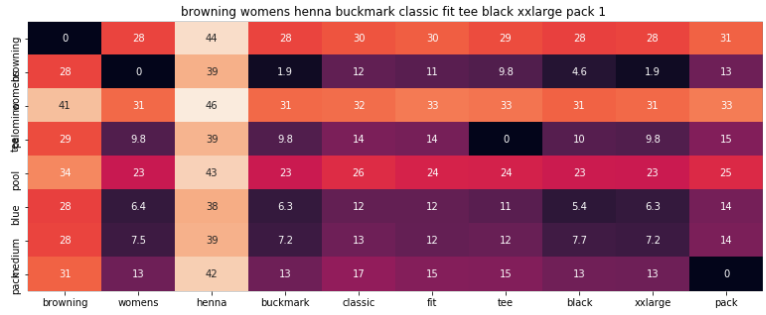




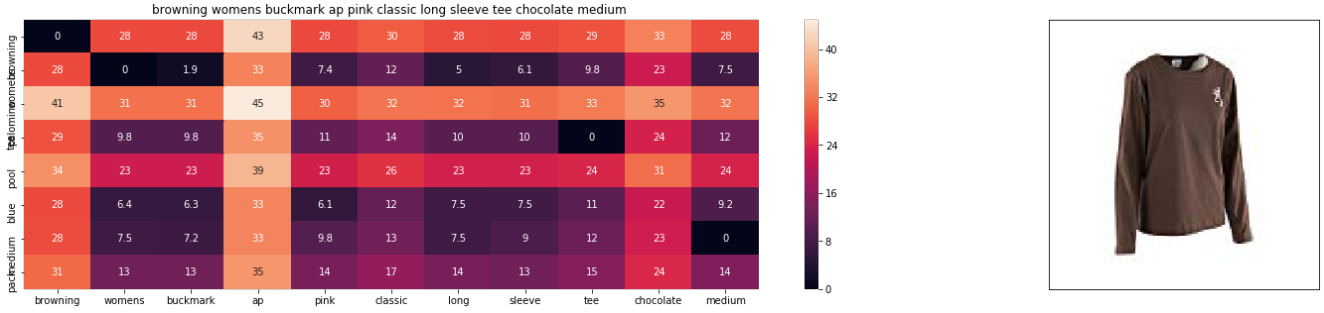
ASIN : B074KQGXQD
Brand : Browning
euclidean distance from input : 2.0547291734845796
==========



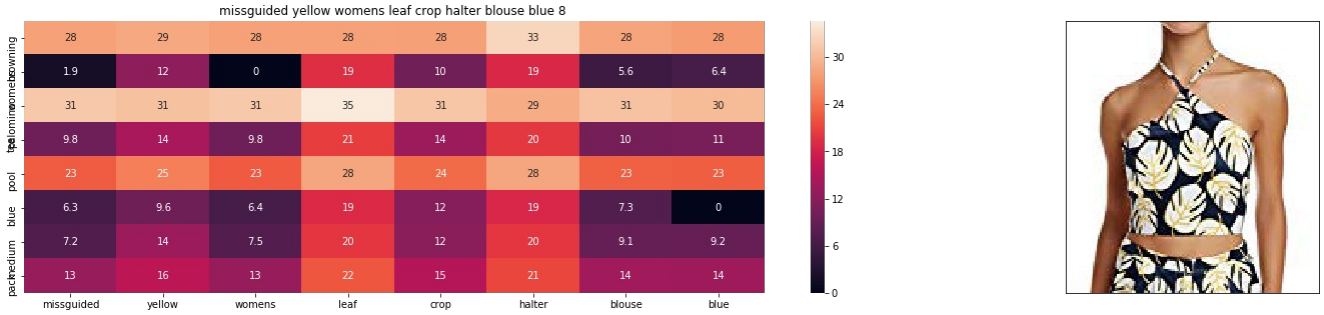
ASIN : B074KQNC89
Brand : Browning
euclidean distance from input : 2.062730709073624
==========



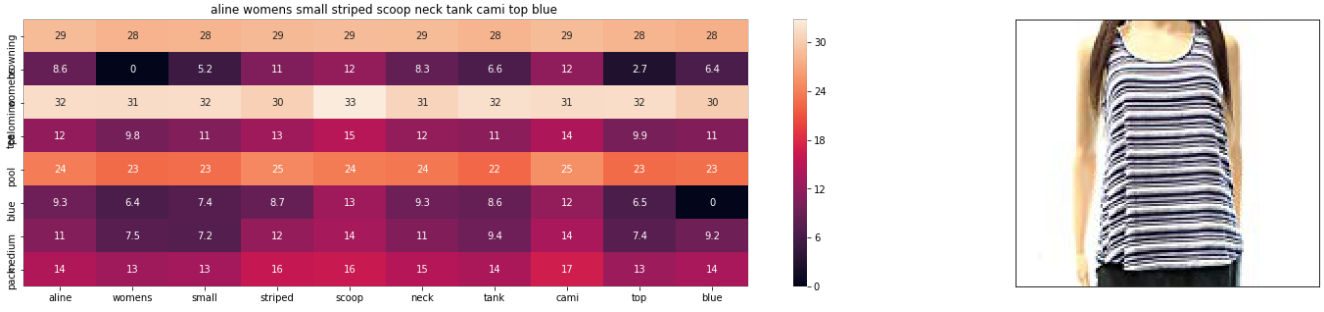
ASIN : B00NQFC32Y
Brand : Browning
euclidean distance from input : 2.0637916804203926
==========



ASIN : B00NQFH7MA
Brand : Browning
euclidean distance from input : 2.0857593082058155

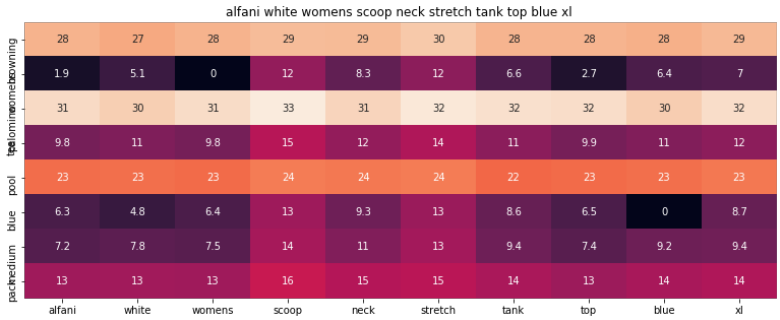


ASIN : B07147JSY5
Brand : Missguided
euclidean distance from input : 2.090135875699601



ASIN : B073V44QB8
Brand : a_line
euclidean distance from input : 2.091740822876447

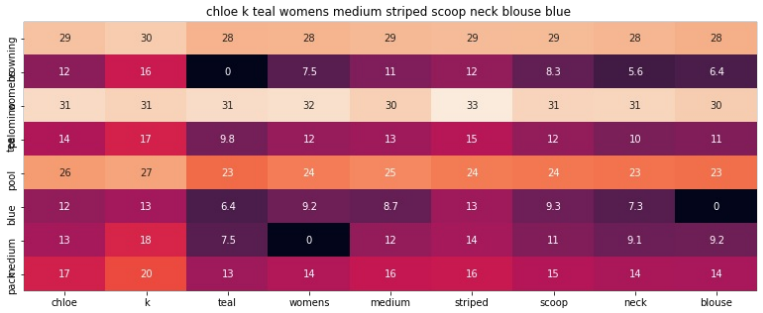
==========



ASIN : B071K8S7WS
Brand : Alfani
euclidean distance from input : 2.0929420016872604

==========

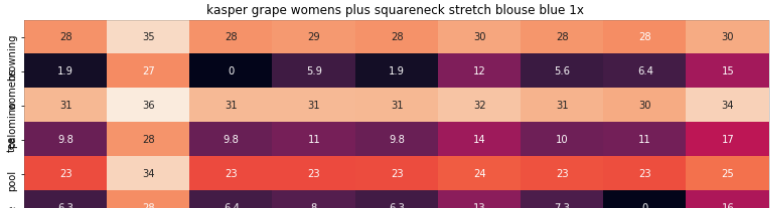
==========

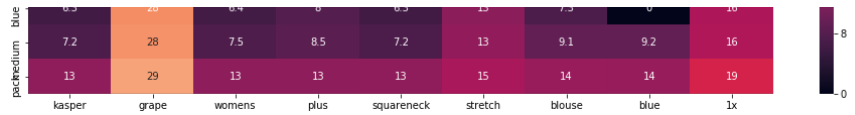


ASIN : B0725RLC8T
Brand : Chloe K.
euclidean distance from input : 2.0984764338383615

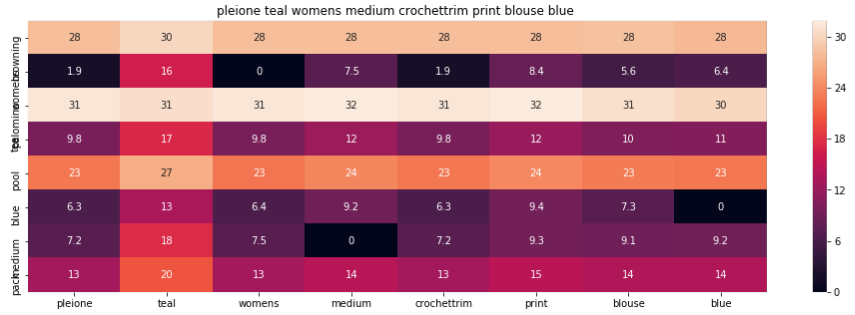
==========

==========

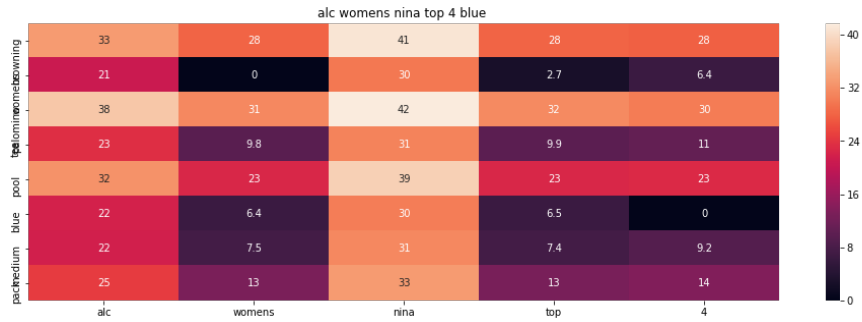




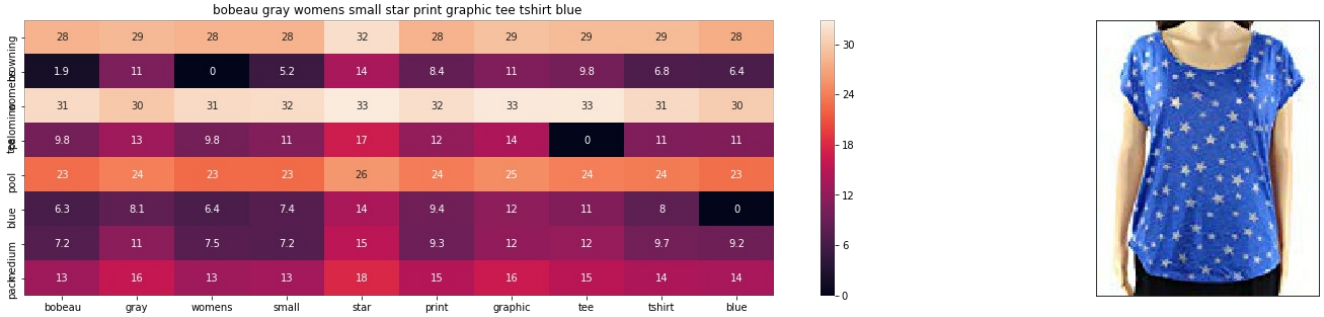
ASIN : B06ZYHTRSD
Brand : Kasper
euclidean distance from input : 2.103432661921192
==========
==========



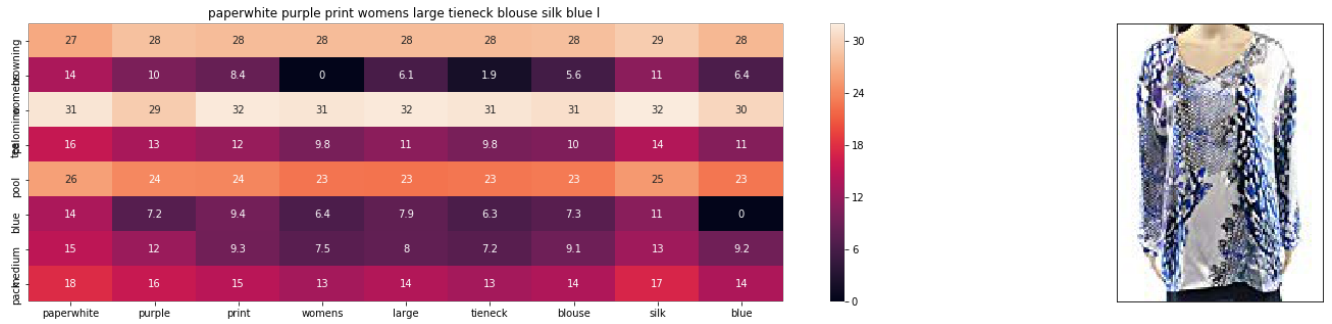
ASIN : B0753KBF1F
Brand : Pleione
euclidean distance from input : 2.1037796953351653
==========
==========



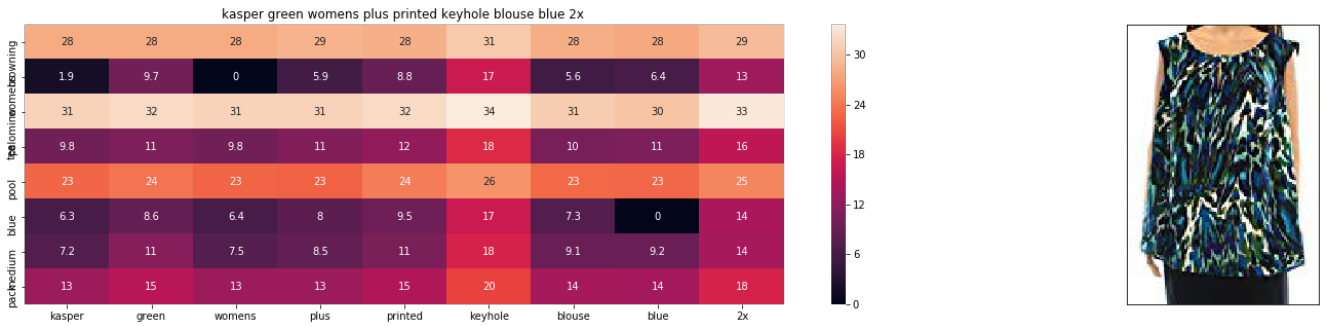
ASIN : B01MPX3OQJ
Brand : A.L.C.
euclidean distance from input : 2.105340160351606
==========
==========



ASIN : B0757WTXX6
Brand : Bobeau
euclidean distance from input : 2.1062659416175693
==========

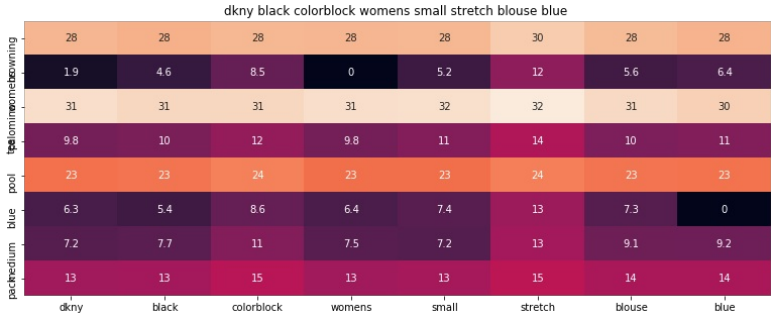


ASIN : B071J264ST
Brand : Paperwhite
euclidean distance from input : 2.1078664151948954
==========



ASIN : B071D6H6S1
Brand : Kasper
euclidean distance from input : 2.1093221036714582
==========

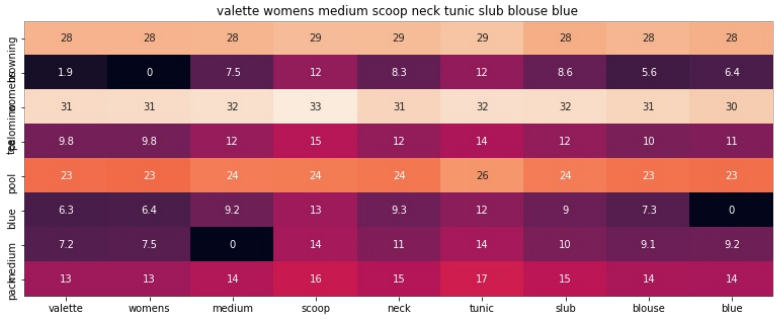
==========



ASIN : B01J298XVW
Brand : DKNY
euclidean distance from input : 2.110365804843333

==========

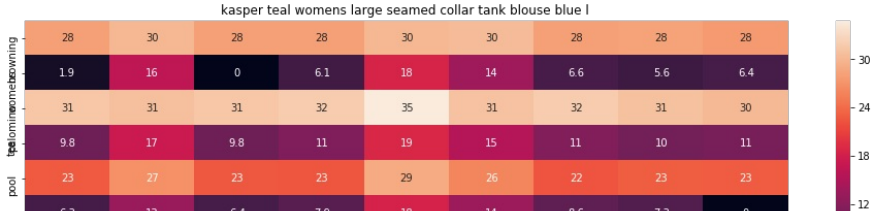
==========

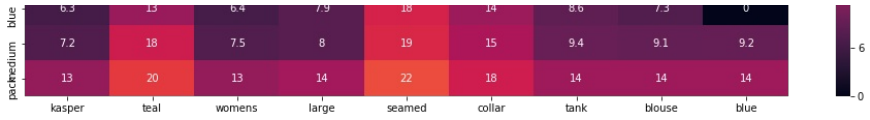


ASIN : B072VHHYSD
Brand : Valette
euclidean distance from input : 2.1104581551962274

==========

==========





```
ASIN : B0722DJVQP
Brand : Kasper
euclidean distance from input : 2.1108748241748314
================================================================================================================
======================
```

[10.2] Keras and Tensorflow to extract features

In [68]:

```
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dropout, Flatten, Dense
from keras import applications
from sklearn.metrics import pairwise_distances
import matplotlib.pyplot as plt
import requests
from PIL import Image
import pandas as pd
import pickle
```

```
Using TensorFlow backend.
```

In [0]:

```
# https://gist.github.com/fchollet/f35fbc80e066a49d65f1688a7e99f069
# Code reference: https://blog.keras.io/building-powerful-image-classification-models-using-very-l
ittle-data.html



# This code takes 40 minutes to run on a modern GPU (graphics card)
# like Nvidia  1050.
# GPU (NVidia 1050): 0.175 seconds per image

# This codse takes 160 minutes to run on a high end i7 CPU
# CPU (i7): 0.615 seconds per image.

#Do NOT run this code unless you want to wait a few hours for it to generate output

# each image is converted into 25088 length dense-vector


'''
# dimensions of our images.
img_width, img_height = 224, 224

top_model_weights_path = 'bottleneck_fc_model.h5'
train_data_dir = 'images2/'
nb_train_samples = 16042
epochs = 50
batch_size = 1


def save_bottlebeck_features():

    #Function to compute VGG-16 CNN for image feature extraction.

    asins = []
    datagen = ImageDataGenerator(rescale=1. / 255)

    # build the VGG16 network
    model = applications.VGG16(include_top=False, weights='imagenet')
    generator = datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_width, img_height),
```

```
        batch_size=batch_size,
        class_mode=None,
        shuffle=False)

    for i in generator.filenames:
        asins.append(i[2:-5])

    bottleneck_features_train = model.predict_generator(generator, nb_train_samples // batch_size)
    bottleneck_features_train = bottleneck_features_train.reshape((16042,25088))

    np.save(open('16k_data_cnn_features.npy', 'wb'), bottleneck_features_train)
    np.save(open('16k_data_cnn_feature_asins.npy', 'wb'), np.array(asins))


save_bottlebeck_features()

'''
```

[10.3] Visual features based product similarity.

In [67]:

```
#load the features and corresponding ASINS info.
bottleneck_features_train = np.load('16k_data_cnn_features.npy')
asins = np.load('16k_data_cnn_feature_asins.npy')
asins = list(asins)

# load the original 16K dataset
data = pd.read_pickle('pickels/16k_apperal_data_preprocessed')
df_asins = list(data['asin'])


from IPython.display import display, Image, SVG, Math, YouTubeVideo


#get similar products using CNN features (VGG-16)
def get_similar_products_cnn(doc_id, num_results):
    doc_id = asins.index(df_asins[doc_id])
    pairwise_dist = pairwise_distances(bottleneck_features_train, bottleneck_features_train[doc_id]
.reshape(1,-1))

    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    pdists  = np.sort(pairwise_dist.flatten())[0:num_results]

    for i in range(len(indices)):
        rows = data[['medium_image_url','title']].loc[data['asin']==asins[indices[i]]]
        for indx, row in rows.iterrows():
            display(Image(url=row['medium_image_url'], embed=True))
            print('Product Title: ', row['title'])
            print('Euclidean Distance from input image:', pdists[i])
            print('Amazon Url: www.amzon.com/dp/'+ asins[indices[i]])

get_similar_products_cnn(12000, 20)
```

```
Product Title:  browning womens palomino tee pool blue medium pack 1
Euclidean Distance from input image: 3.015783e-06
Amazon Url: www.amzon.com/dp/B074KQ3SY1
```

Product Title: browning womens crinkle foil buckmark tshirt turquoise xxlarge pack 1
Euclidean Distance from input image: 30.399572
Amazon Url: www.amzon.com/dp/B074KQGXQD

Product Title: browning womens classic fit buckmark tee mossy oak pink camo chocolate xxlarge pa
ck 1
Euclidean Distance from input image: 31.081032
Amazon Url: www.amzon.com/dp/B074KPXCPC

Product Title: browning womens classic fit buckmark tee chocolate large pack 1
Euclidean Distance from input image: 31.081032
Amazon Url: www.amzon.com/dp/B074KPYSRC

Product Title: womens flirty cap sleeve top white xsmall
Euclidean Distance from input image: 32.52637
Amazon Url: www.amzon.com/dp/B00U7PRVEG

Product Title: browning womens antler type tshirt charcoal large pack 1
Euclidean Distance from input image: 33.660416
Amazon Url: www.amzon.com/dp/B074KPDZH8

Product Title: relatree girl snow hoot tee silver large pack 1
Euclidean Distance from input image: 33.944954
Amazon Url: www.amzon.com/dp/B074KPCX5H

Product Title: daisy floral dot pastel green tee tops womens
Euclidean Distance from input image: 33.97971
Amazon Url: www.amzon.com/dp/B01IZKX6IO

Product Title: browning womens princess boots tshirts papaya large pack 1
Euclidean Distance from input image: 34.056347
Amazon Url: www.amzon.com/dp/B074KPJ8YP

Product Title: danskin womens vneck loose performance tee xsmall pink ombre
Euclidean Distance from input image: 34.065056
Amazon Url: www.amzon.com/dp/B01F7PHXY8

Product Title: browning womens fitted buckmark tee tangerine xxlarge pack 1
Euclidean Distance from input image: 34.281143
Amazon Url: www.amzon.com/dp/B074KQNC89

Product Title: realtree girl sweet tea short sleeve tee texas orange large pack 1
Euclidean Distance from input image: 34.28302
Amazon Url: www.amzon.com/dp/B074KPF553

Product Title: mlv womens beaded v back luna tank top sz dove grey 270745dh
Euclidean Distance from input image: 34.408257
Amazon Url: www.amzon.com/dp/B071XDBQWZ

Product Title: cutter buck 34 sleeve highland park osu reflect womens shirt
Euclidean Distance from input image: 34.694305
Amazon Url: www.amzon.com/dp/B06VTH8C81

Product Title: womens pastel pink tops tees leaf design print size
Euclidean Distance from input image: 34.82372
Amazon Url: www.amzon.com/dp/B014YFEOIK

Product Title: browning womens classic fit 3d zebra tee fuchsia xxlarge pack 1
Euclidean Distance from input image: 34.842373
Amazon Url: www.amzon.com/dp/B074KQ9P3W

```
Product Title:  szhem womens modal loose short sleeve tops v neck pocket tshirt blouse
Euclidean Distance from input image: 34.907215
Amazon Url: www.amzon.com/dp/B01BEV6JNK
```

```
Product Title:  polo ralph lauren womens short sleeve oxford button shirt new rose xlarge
Euclidean Distance from input image: 34.99459
Amazon Url: www.amzon.com/dp/B06WD6MBMT
```

```
Product Title:  realreee girl mila tshirt realtree max1 camo brown xsmall pack 1
Euclidean Distance from input image: 35.10164
Amazon Url: www.amzon.com/dp/B01IL4Q9LK
```

```
Product Title:  nancy lopez luster short sleeve  lemon drop
Euclidean Distance from input image: 35.10559
Amazon Url: www.amzon.com/dp/B01MCWSKXM
```

Weighted Euclidian-Distance base Similarity usingText, Brand, Color, Image.

In [98]:

```
bottleneck_features_train = np.load('16k_data_cnn_features.npy')
asins = np.load('16k_data_cnn_feature_asins.npy')
data = pd.read_pickle('pickels/16k_apperal_data_preprocessed')
df_asins = list(data['asin'])
asins = list(asins)
```

In [99]:

```
import pickle
with open('word2vec_model', 'rb') as handle:
```

```
    model = pickle.load(handle)
```

In [103]:

```
doc_id = 0
w2v_title_weight = []
# for every title we build a weighted vector representation
for i in data['title']:
    w2v_title_weight.append(build_avg_vec(i, 300, doc_id,'weighted'))
    doc_id += 1
# w2v_title = np.array(# number of doc in courpus * 300), each row corresponds to a doc
w2v_title_weight = np.array(w2v_title_weight)
#print ("hi")
```

In [113]:

```
from PIL import Image
def display_img(url,ax,fig):
    # we get the url of the apparel and download it
    response = requests.get(url)
    img = Image.open(BytesIO(response.content))
    # we will display it in notebook
    plt.imshow(img)
```

In [105]:

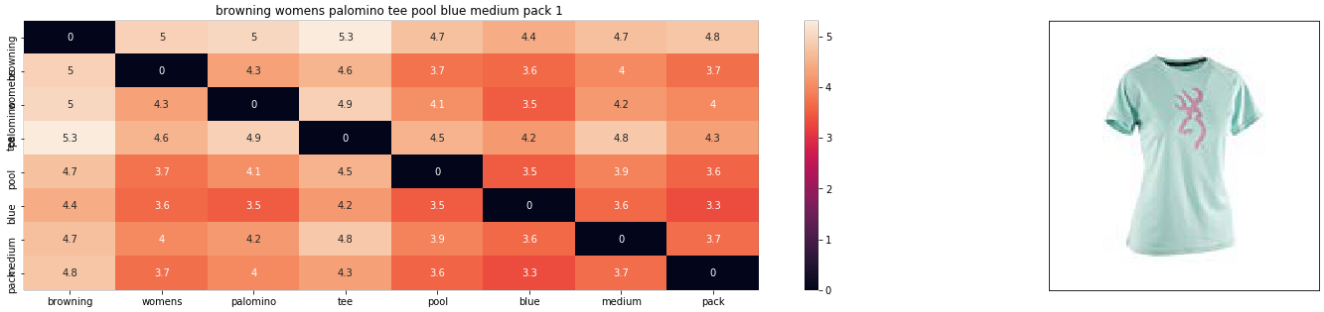
```
def get_word_vec(sentence, doc_id, m_name):
    # sentence : title of the apparel
    # doc_id: document id in our corpus
    # m_name: model information it will take two values
        # if  m_name == 'avg', we will append the model[i], w2v representation of word i
        # if m_name == 'weighted', we will multiply each w2v[word] with the idf(word)
    vec = []
    for i in sentence.split():
        if i in vocab:
            if m_name == 'weighted' and i in  idf_title_vectorizer.vocabulary_:
                vec.append(idf_title_features[doc_id, idf_title_vectorizer.vocabulary_[i]] * model[
i])
            elif m_name == 'avg':
                vec.append(model[i])
        else:
            # if the word in our courpus is not there in the google word2vec corpus, we are just
ignoring it
            vec.append(np.zeros(shape=(300,)))
    # we will return a numpy array of shape (#number of words in title * 300 ) 300 =
len(w2v_model[word])
    # each row represents the word2vec representation of each word (weighted/avg) in given
sentance
    return  np.array(vec)

def get_distance(vec1, vec2):
    # vec1 = np.array(#number_of_words_title1 * 300), each row is a vector of length 300
corresponds to each word in give title
    # vec2 = np.array(#number_of_words_title2 * 300), each row is a vector of length 300
corresponds to each word in give title

    final_dist = []
    # for each vector in vec1 we caluclate the distance(euclidean) to all vectors in vec2
    for i in vec1:
        dist = []
        for j in vec2:
            # np.linalg.norm(i-j) will result the euclidean distance between vectors i, j
            dist.append(np.linalg.norm(i-j))
        final_dist.append(np.array(dist))
    # final_dist = np.array(#number of words in title1 * #number of words in title2)
    # final_dist[i,j] = euclidean distance between vectors i, j
    return np.array(final_dist)
def heat_map_w2v_brand(sentance1, sentance2, url, doc_id1, doc_id2, df_id1, df_id2, model):

    # sentance1 : title1, input apparel
    # sentance2 : title2, recommended apparel
    # url: apparel image url
    # doc_id1: document id of input apparel
    # doc_id2: document id of recommended apparel
```

```
    # df_id1: index of document1 in the data frame
    # df_id2: index of document2 in the data frame
    # model: it can have two values, 1. avg 2. weighted

    #s1_vec = np.array(#number_of_words_title1 * 300), each row is a vector(weighted/avg) of lengt
h 300 corresponds to each word in give title
    s1_vec = get_word_vec(sentance1, doc_id1, model)
    #s2_vec = np.array(#number_of_words_title2 * 300), each row is a vector(weighted/avg) of lengt
h 300 corresponds to each word in give title
    s2_vec = get_word_vec(sentance2, doc_id2, model)

    # s1_s2_dist = np.array(#number of words in title1 * #number of words in title2)
    # s1_s2_dist[i,j] = euclidean distance between words i, j
    s1_s2_dist = get_distance(s1_vec, s2_vec)

    data_matrix = [['Asin','Brand', 'Color', 'Product type'],
                [data['asin'].loc[df_id1],brands[doc_id1], colors[doc_id1], types[doc_id1]], # input
apparel's features
                [data['asin'].loc[df_id2],brands[doc_id2], colors[doc_id2], types[doc_id2]] #
recommonded apparel's features

    colorscale = [[0, '#1d004d'],[.5, '#f2e5ff'],[1, '#f2e5d1']] # to color the headings of each cc
lumn

    # we create a table with the data_matrix
    table = ff.create_table(data_matrix, index=True, colorscale=colorscale)
    # plot it with plotly
    plotly.offline.iplot(table, filename='simple_table')

    # devide whole figure space into 25 * 1:10 grids
    gs = gridspec.GridSpec(25, 15)
    fig = plt.figure(figsize=(25,5))

    # in first 25*10 grids we plot heatmap
    ax1 = plt.subplot(gs[:, :-5])
    # ploting the heap map based on the pairwise distances
    ax1 = sns.heatmap(np.round(s1_s2_dist,6), annot=True)
    # set the x axis labels as recommended apparels title
    ax1.set_xticklabels(sentance2.split())
    # set the y axis labels as input apparels title
    ax1.set_yticklabels(sentance1.split())
    # set title as recommended apparels title
    ax1.set_title(sentance2)

    # in last 25 * 10:15 grids we display image
    ax2 = plt.subplot(gs[:, 10:16])
    # we dont display grid lins and axis labels to images
    ax2.grid(False)
    ax2.set_xticks([])
    ax2.set_yticks([])

    # pass the url it display it
    display_img(url, ax2, fig)

    plt.show()
```

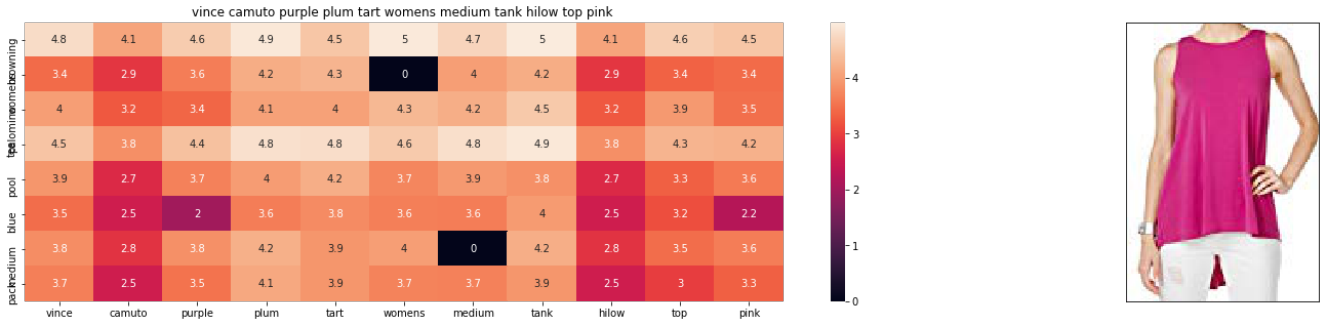
In [114]:

```
def idf_w2v_brand_image(doc_id, w1, w2, w3, num_results):
    # doc_id: apparel's id in given corpus
    # w1: weight for  w2v features
    # w2: weight for brand and color features

    # pairwise_dist will store the distance from given input apparel to all remaining apparels
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y) = <X, Y> / (||X
||*||Y||)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    idf_w2v_dist  = pairwise_distances(w2v_title_weight, w2v_title_weight[doc_id].reshape(1,-1))
    ex_feat_dist = pairwise_distances(extra_features, extra_features[doc_id])
    imgext_feat_dist = pairwise_distances(bottleneck_features_train,
bottleneck_features_train[doc_id].reshape(1,-1))
    pairwise_dist   = (w1 * idf_w2v_dist +  w2 * ex_feat_dist + w3 * imgext_feat_dist)/float(w1 + w
2 + w3)

    # np.argsort will return indices of 9 smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
```

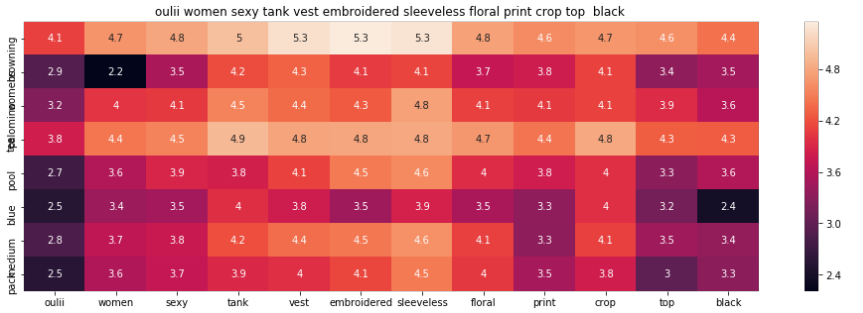
```
    #pdists will store the 9 smallest distances
    pdists  = np.sort(pairwise_dist.flatten())[0:num_results]

    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])


    for i in range(0, len(indices)):
        heat_map_w2v_brand(data['title'].loc[df_indices[0]],data['title'].loc[df_indices[i]], data['medium_image_url'].loc[df_indices[i]], indices[0], indices[i],df_indices[0], df_indices[i], 'weighted')
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('Brand :',data['brand'].loc[df_indices[i]])
        print('euclidean distance from input :', pdists[i])
        print('='*125)

idf_w2v_brand_image(12000, 5, 5, 5, 20)
# in the give heat map, each cell contains the euclidean distance between words i, j
```



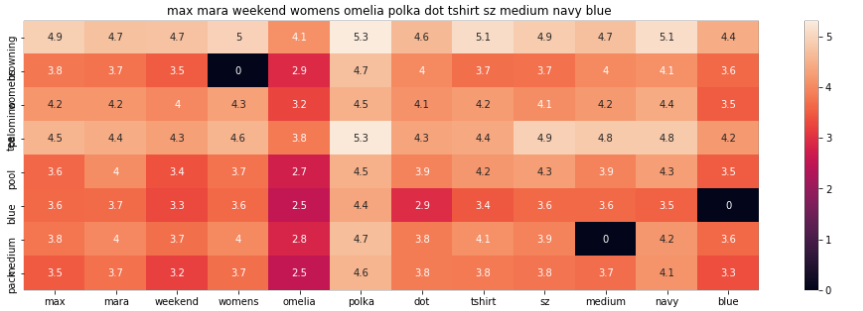
ASIN : B074KQ3SY1
Brand : Browning
euclidean distance from input : 2.156044744576017e-06
==========



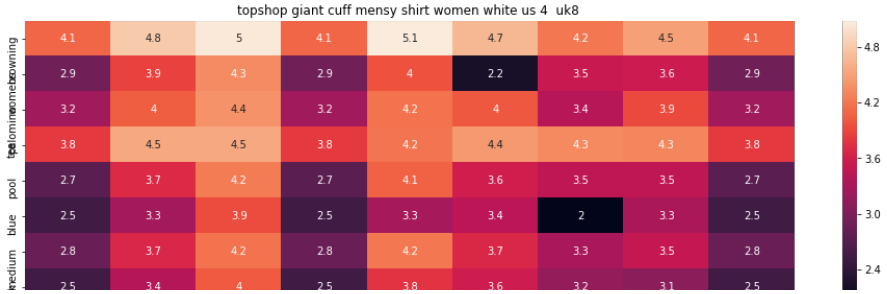
ASIN : B06XTR6T7J
Brand : Vince Camuto
euclidean distance from input : 15.922381323664544
==========



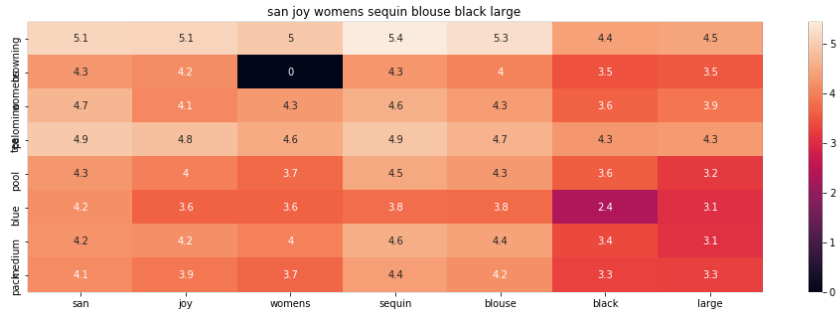
ASIN : B074KZ12G1
Brand : OULII
euclidean distance from input : 16.85175215898349
==========



ASIN : B0749RP46C
Brand : MaxMara
euclidean distance from input : 16.937298075358072
==========

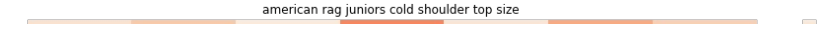


ASIN : B07538JKWD
Brand : Top Shop
euclidean distance from input : 16.952299136011955
==========

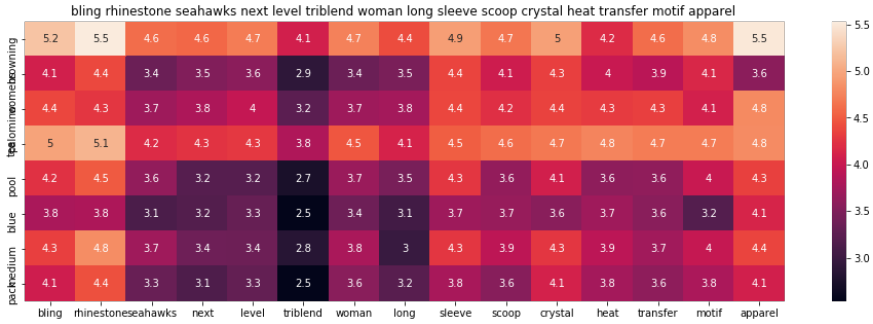


ASIN : B073WKFKLZ
Brand : Sanjoy
euclidean distance from input : 17.00877244172885
==========

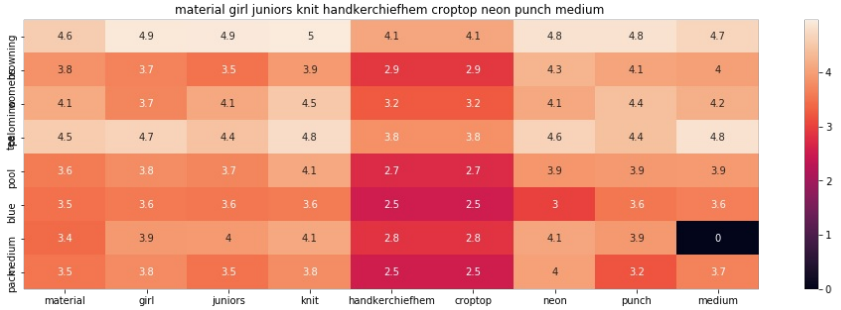
ASIN : B010KALHDI
Brand : MKP Crop Top
euclidean distance from input : 17.013113411824182
==========



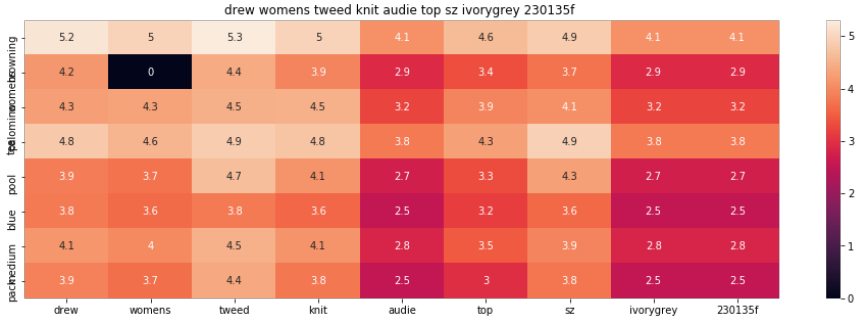
ASIN : B06XY6HK4Z
Brand : American Rag
euclidean distance from input : 17.103888829790073
==========



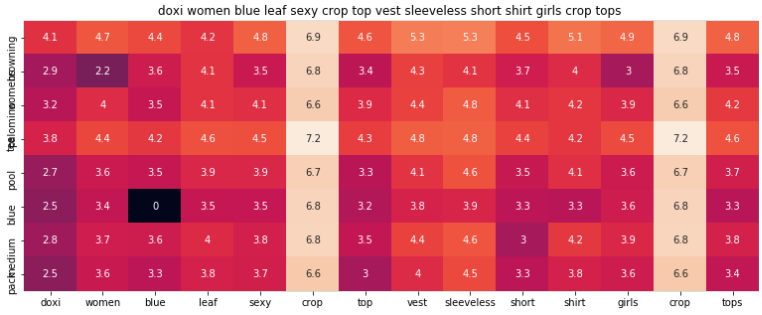
ASIN : B019HKGIR2
Brand : Crystal Trends
euclidean distance from input : 17.123469911107577
==========



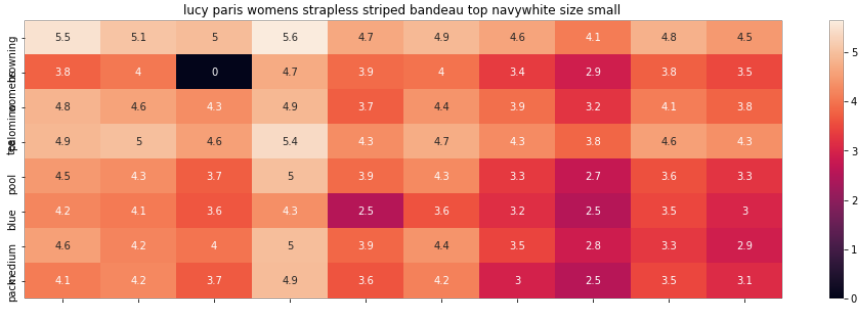
ASIN : B01NC3AFJ9
Brand : Material Girl
euclidean distance from input : 17.21490895803797
==========



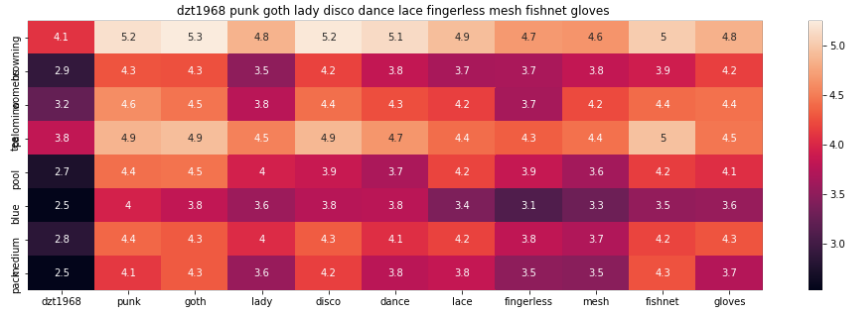
ASIN : B01ETLYQ6E
Brand : Drew Shoe
euclidean distance from input : 17.23541187500981
==========



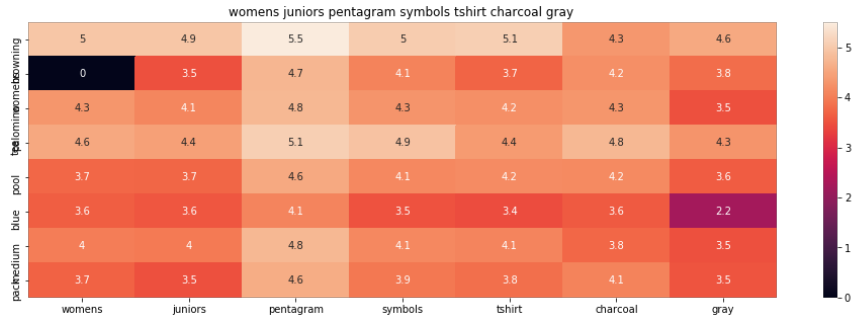
ASIN : B01LF90Q0I
Brand : Doxi Supermall
euclidean distance from input : 17.286419836680093
==========



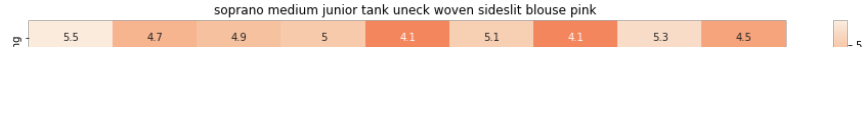
ASIN : B01LFTRTXU
Brand : Lucy Paris
euclidean distance from input : 17.291343125264124
==========

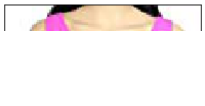


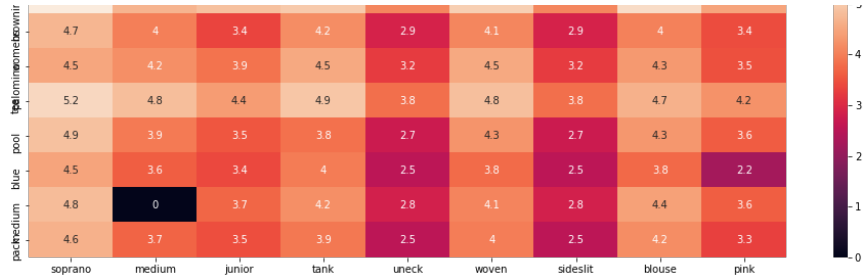
ASIN : B0154OHQZS
Brand : DZT1968
euclidean distance from input : 17.308274401211683
==========



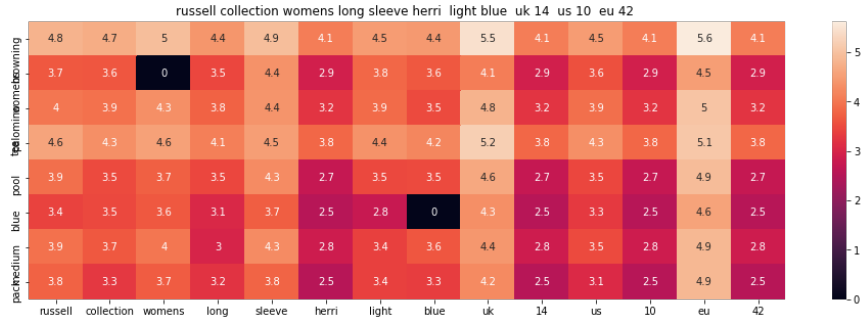
ASIN : B0748ZQQ49
Brand : Buddys USA
euclidean distance from input : 17.3268311585588
==========



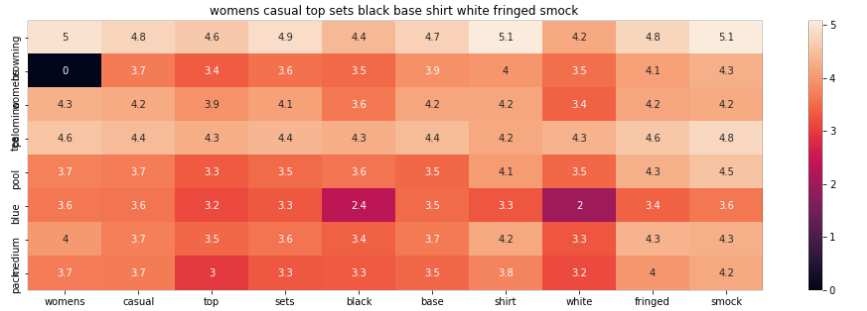




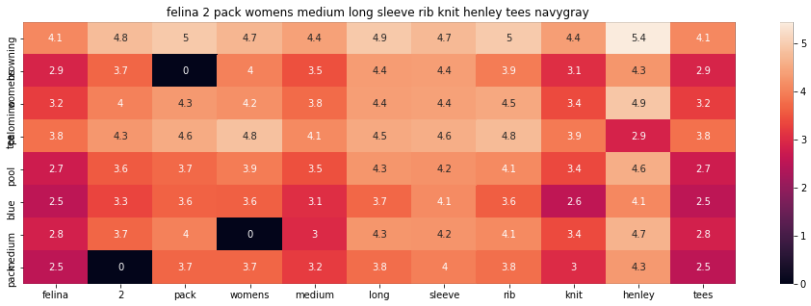
ASIN : B072BXFTKQ
Brand : Soprano
euclidean distance from input : 17.34365476467604
==========



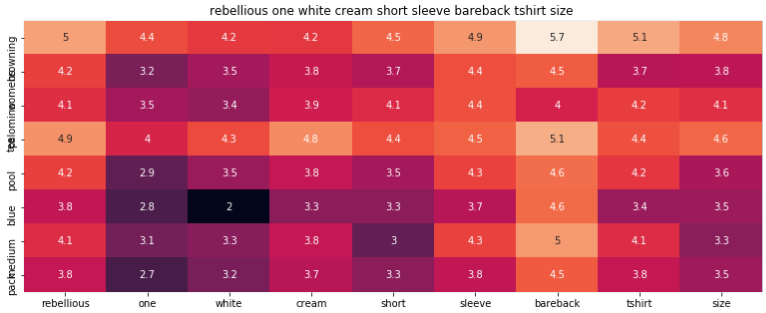
ASIN : B00K072ZIS
Brand : Russell Collection
euclidean distance from input : 17.41198893088494
==========



ASIN : B011TOPBRC
Brand : HP-LEISURE
euclidean distance from input : 17.41948137307417
==========



ASIN : B06Y1S7ZLM
Brand : Felina
euclidean distance from input : 17.447000489720857

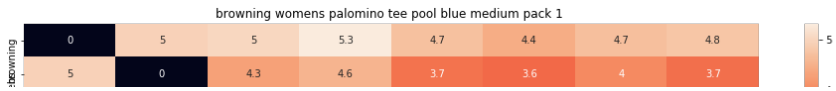


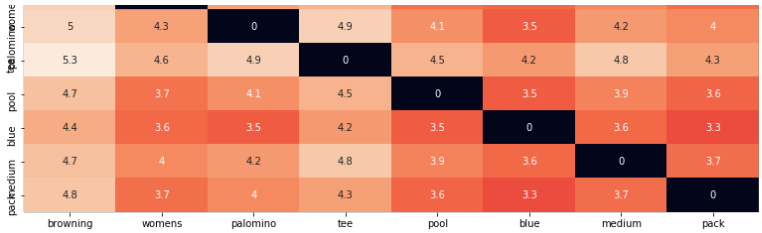
ASIN : B01H5JFDVE
Brand : Rebellious One
euclidean distance from input : 17.451064572820222

Considered More Weightage to Image

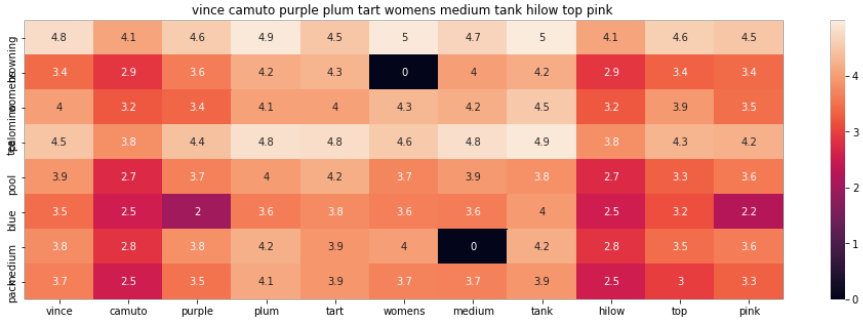
In [116]:

```
# giving more priority to the image
idf_w2v_brand_image(12000,5,5,80, 20)
```

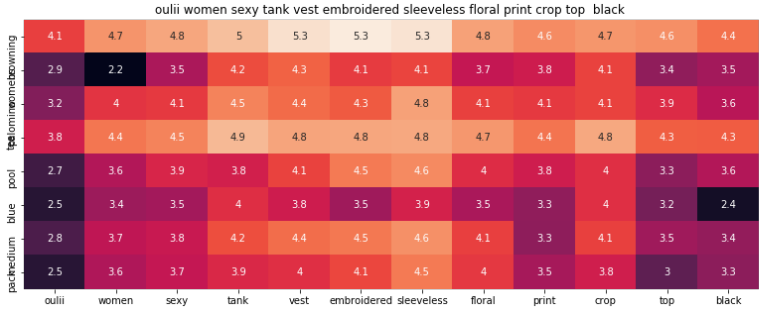




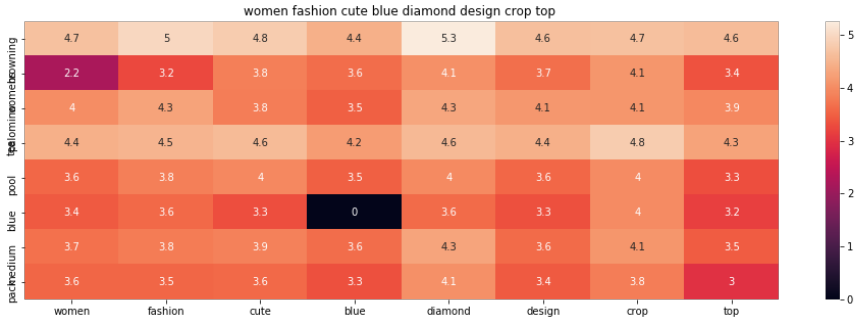
ASIN : B074KQ3SY1
Brand : Browning
euclidean distance from input : 5.749452652202712e-06
==========



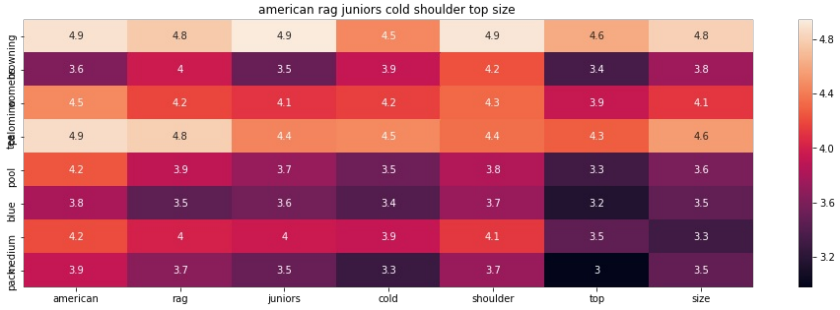
ASIN : B06XTR6T7J
Brand : Vince Camuto
euclidean distance from input : 39.56846665046101
==========



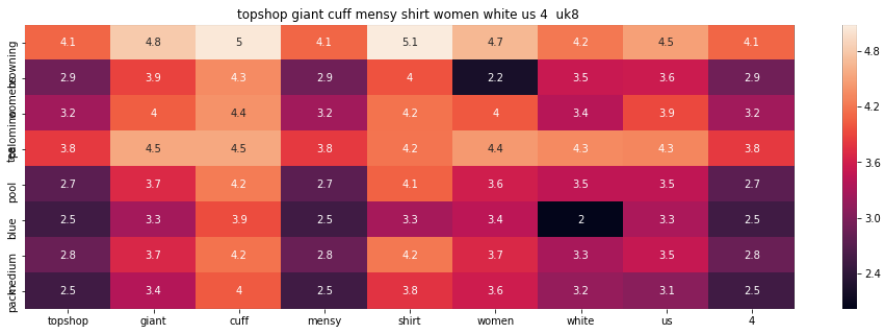
ASIN : B074KZ12G1
Brand : OULII
euclidean distance from input : 42.07947140882147
==========



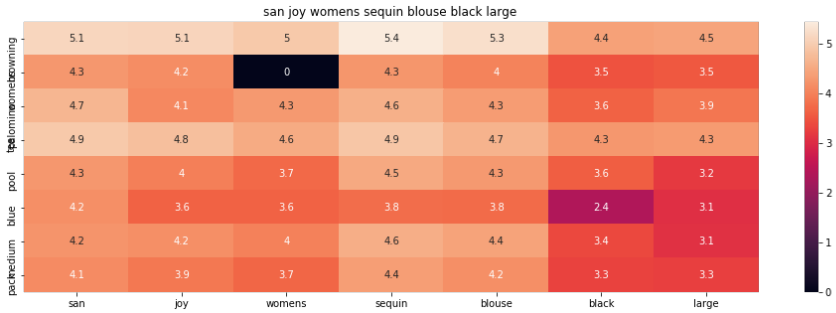
ASIN : B010KALHDI
Brand : MKP Crop Top
euclidean distance from input : 42.137136842237624
==========



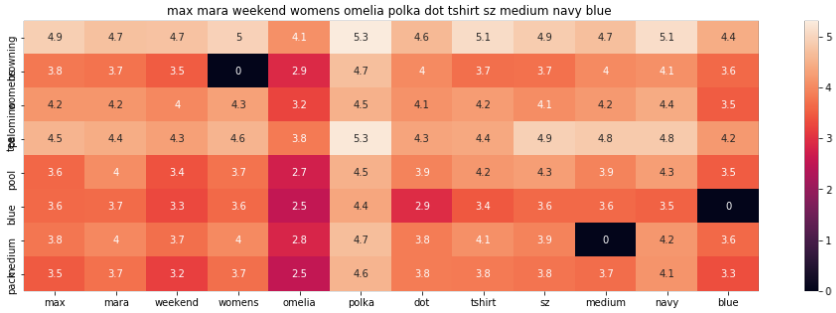
ASIN : B06XY6HK4Z
Brand : American Rag
euclidean distance from input : 42.212256007764495
==========



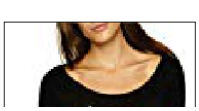
ASIN : B07538JKWD
Brand : Top Shop
euclidean distance from input : 42.25872008305277
==========

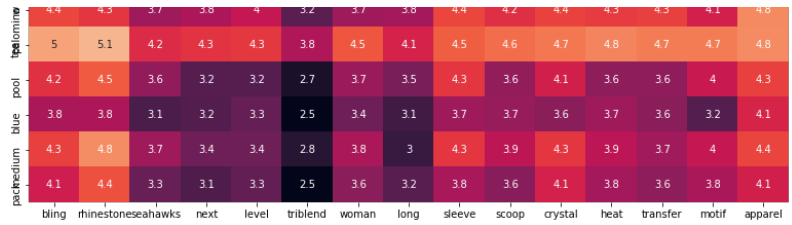


ASIN : B073WKFKLZ
Brand : Sanjoy
euclidean distance from input : 42.32619453618658
==========

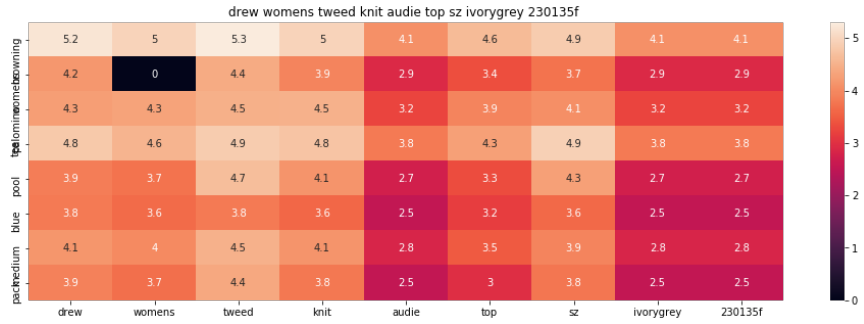


ASIN : B0749RP46C
Brand : MaxMara
euclidean distance from input : 42.62360068427192
==========

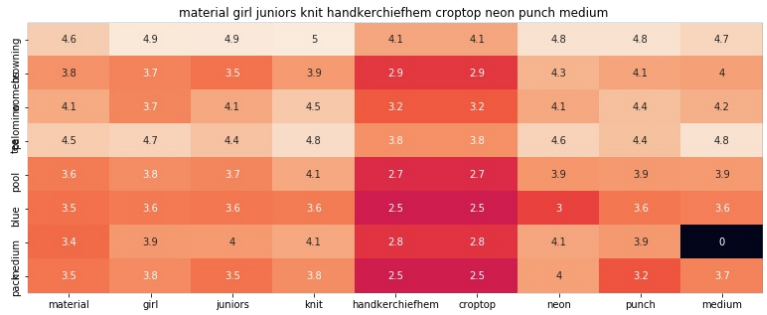




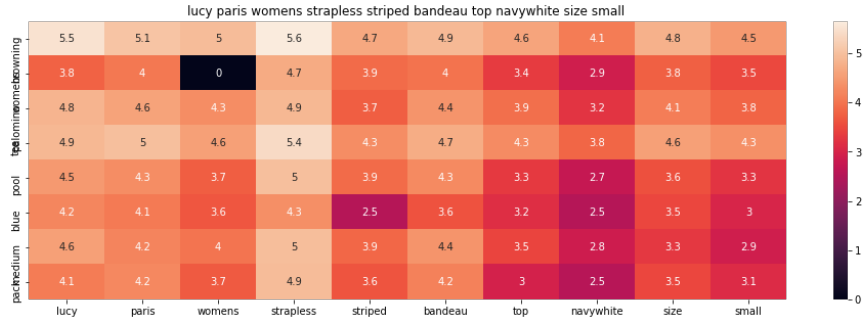
ASIN : B019HKGIR2
Brand : Crystal Trends
euclidean distance from input : 42.692916168452825
==========



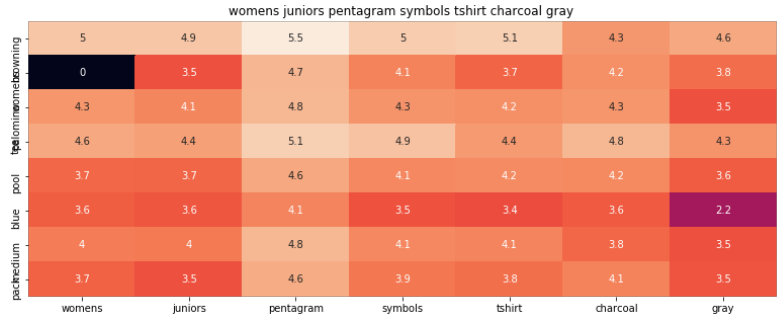
ASIN : B01ETLYQ6E
Brand : Drew Shoe
euclidean distance from input : 42.750752646486006
==========



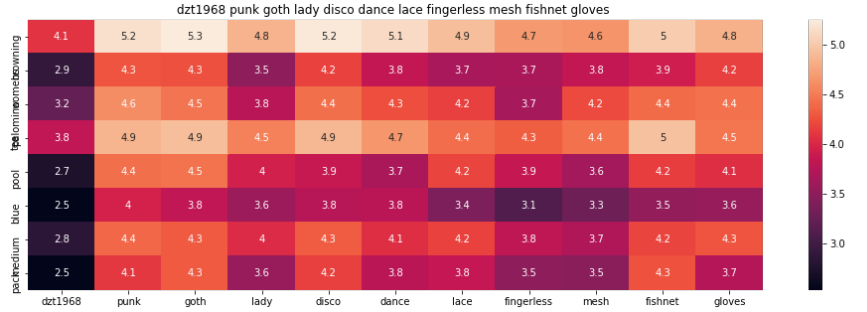
ASIN : B01NC3AFJ9
Brand : Material Girl
euclidean distance from input : 42.772934655441226
==========



ASIN : B01LFTRTXU
Brand : Lucy Paris
euclidean distance from input : 42.80319331628751
==========

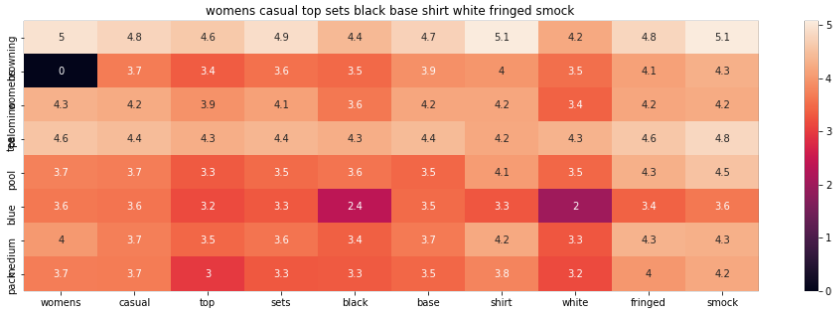


ASIN : B0748ZQQ49
Brand : Buddys USA
euclidean distance from input : 42.87460379742256
==========

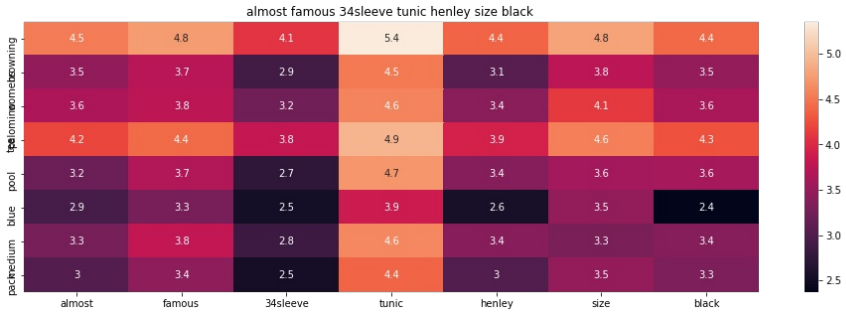


ASIN : B015AQHQZC

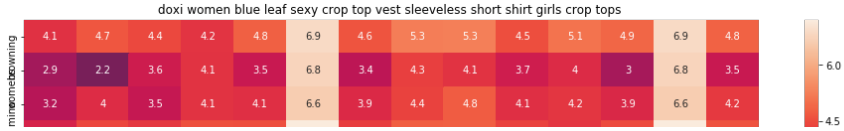
ASIN : B01540HQZS
Brand : DZT1968
euclidean distance from input : 42.93959826445976
==========

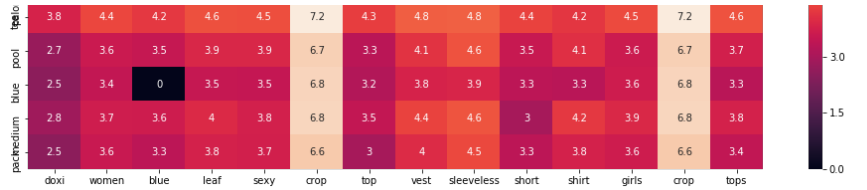


ASIN : B011TOPBRC
Brand : HP-LEISURE
euclidean distance from input : 43.104047473311844
==========

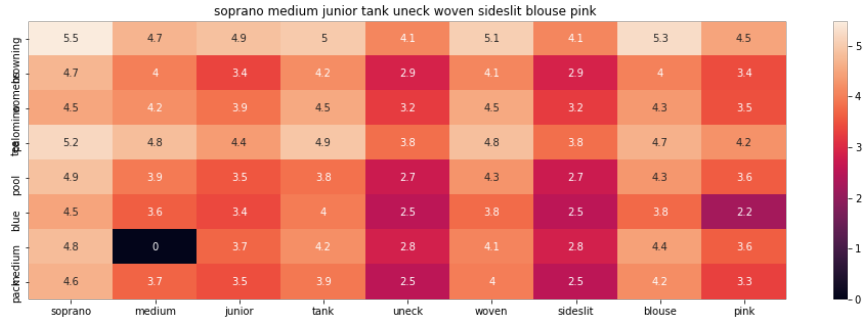


ASIN : B01I2IRKLI
Brand : Almost Famous
euclidean distance from input : 43.12303227848477
==========

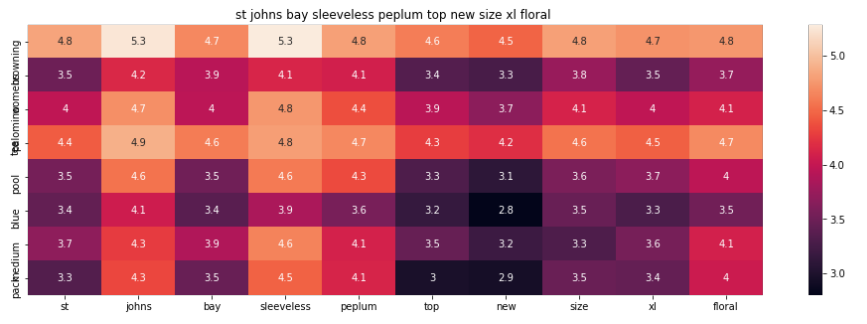




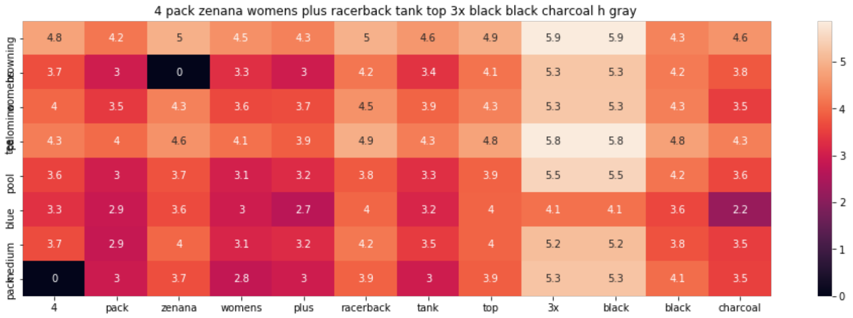
ASIN : B01LF90Q0I
Brand : Doxi Supermall
euclidean distance from input : 43.278515651490956
==========



ASIN : B072BXFTKQ
Brand : Soprano
euclidean distance from input : 43.41280303613741
==========



ASIN : B01N0IDEI6
Brand : St. John's Bay
euclidean distance from input : 43.4617990843343
==========

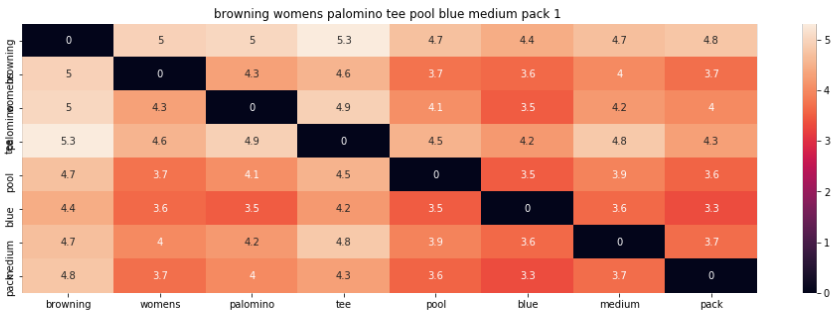


```
ASIN : B01ESA57Q4
Brand : Zenana Outfitters
euclidean distance from input : 43.46652193896885
=============================================================================================================
========================
```

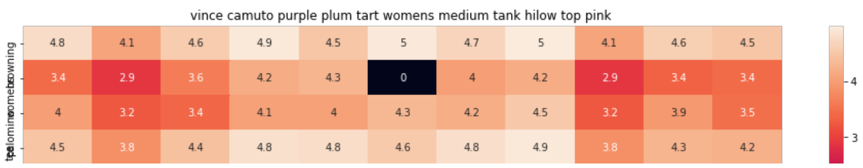
Check With More Weightage Model

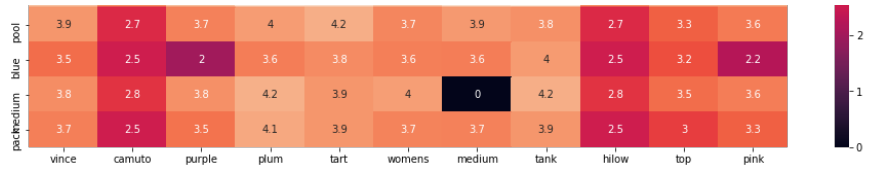
In [120]:

```
# giving more priority to the image
idf_w2v_brand_image(12000,5, 5, 50, 20)
```

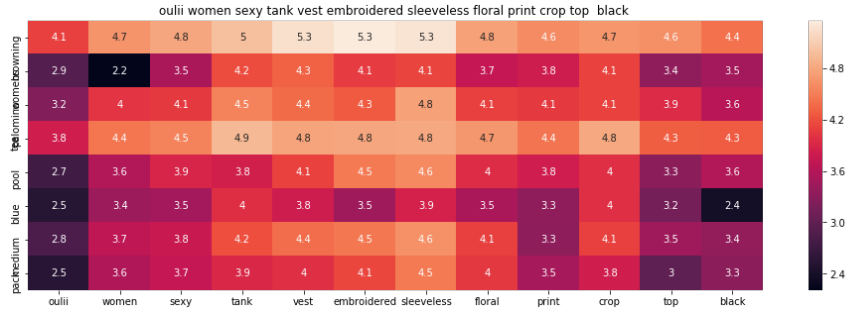


```
ASIN : B074KQ3SY1
Brand : Browning
euclidean distance from input : 5.390111861440043e-06
=============================================================================================================
========================
```

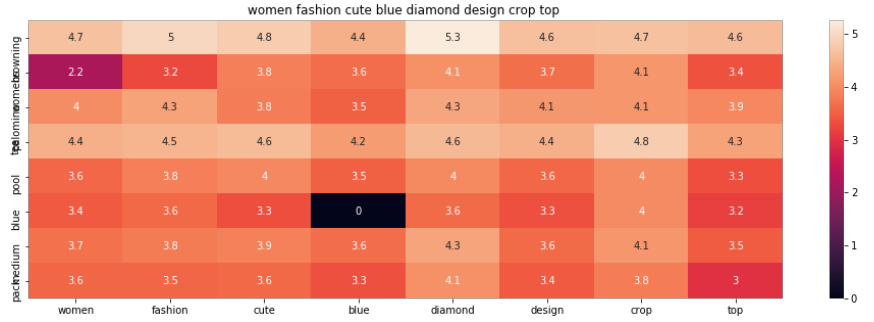




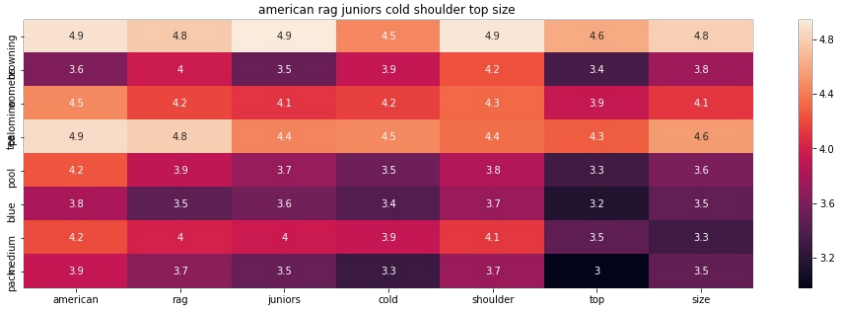
ASIN : B06XTR6T7J
Brand : Vince Camuto
euclidean distance from input : 37.20385964366027
==========



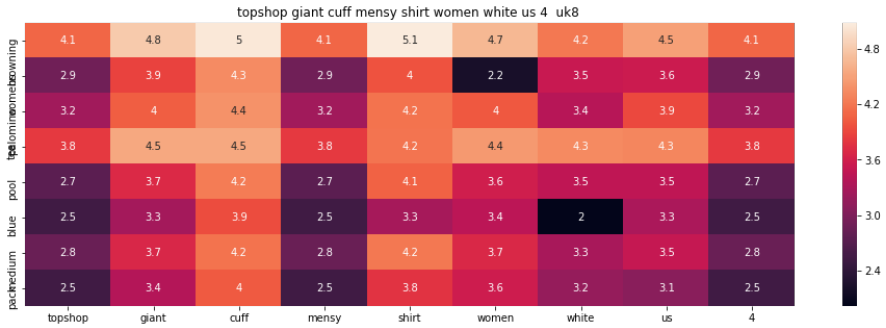
ASIN : B074KZ12G1
Brand : OULII
euclidean distance from input : 39.55669897521137
==========



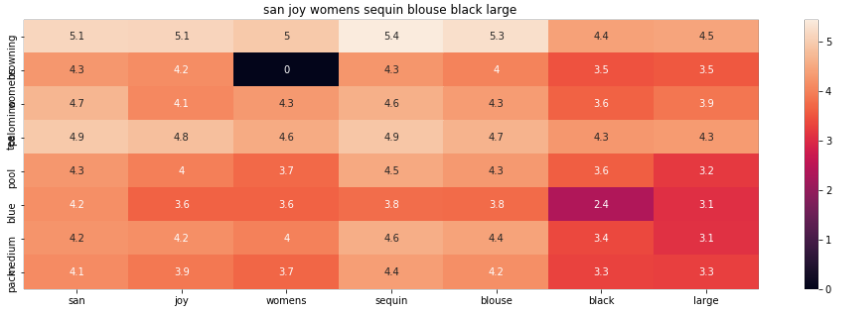
ASIN : B010KALHDI
Brand : MKP Crop Top
euclidean distance from input : 39.62473602507519
==========



ASIN : B06XY6HK4Z
Brand : American Rag
euclidean distance from input : 39.70142030721966
==========
==========



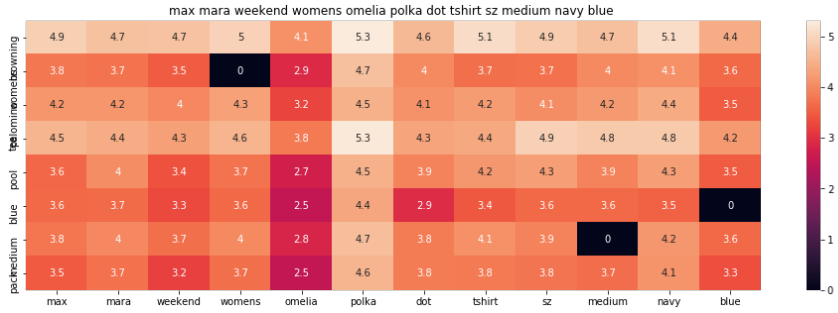
ASIN : B07538JKWD
Brand : Top Shop
euclidean distance from input : 39.72807849697499
==========
==========



ASIN : B073WKFKLZ
Brand : Sanjoy
euclidean distance from input : 39.7044513094882

euclidean distance from input : 39.79445130948827

==========

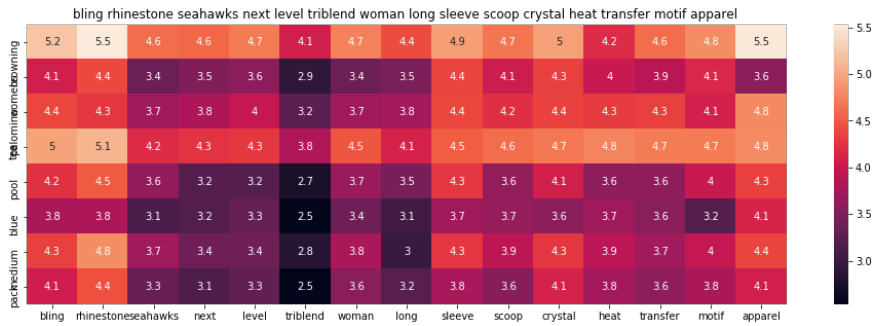


ASIN : B0749RP46C

Brand : MaxMara

euclidean distance from input : 40.05496889750163

==========

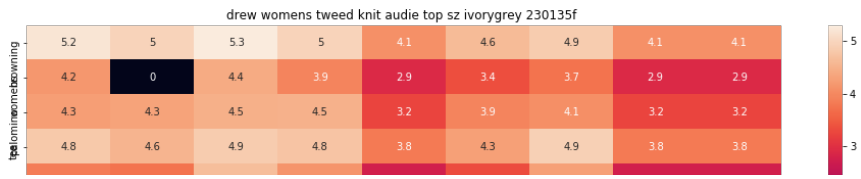


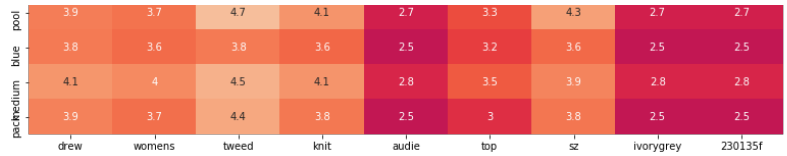
ASIN : B019HKGIR2

Brand : Crystal Trends

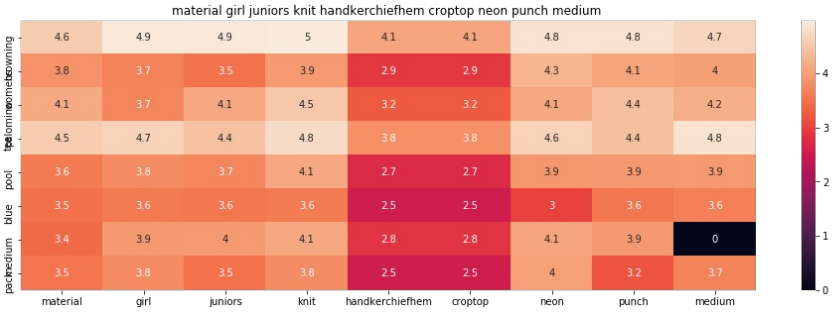
euclidean distance from input : 40.1359725599709

==========

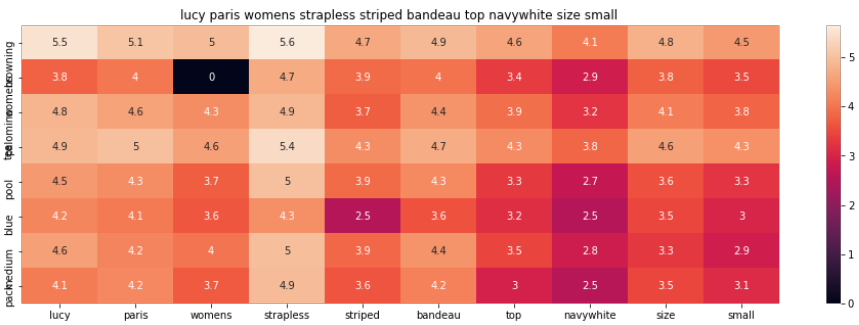




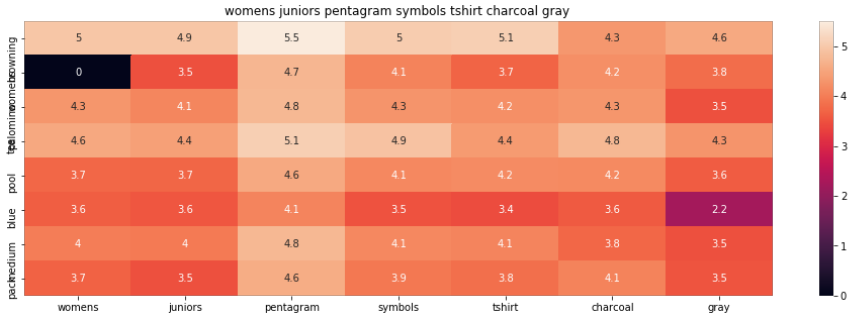
ASIN : B01ETLYQ6E
Brand : Drew Shoe
euclidean distance from input : 40.199216534833184
==========



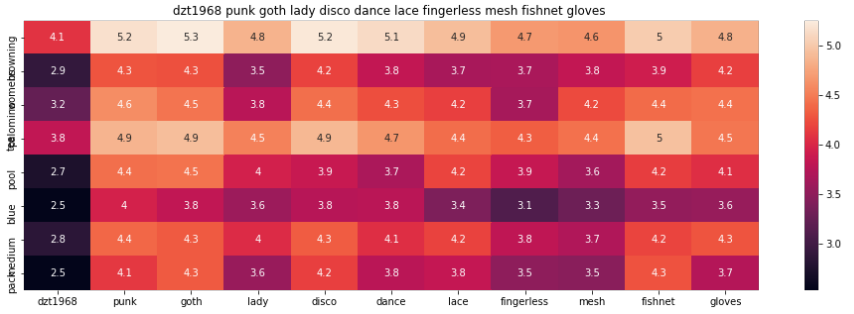
ASIN : B01NC3AFJ9
Brand : Material Girl
euclidean distance from input : 40.217133102953504
==========



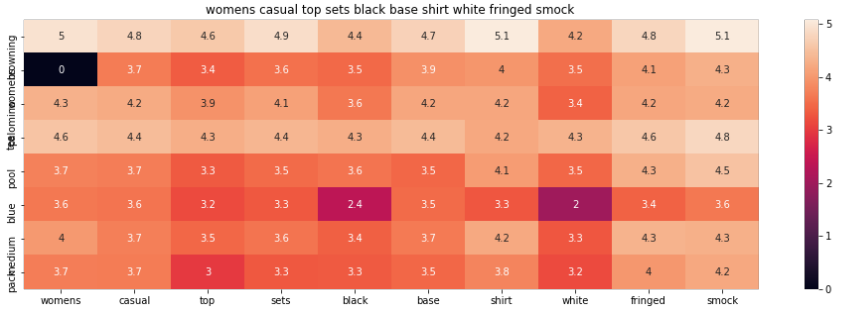
ASIN : B01LFTRTXU
Brand : Lucy Paris
euclidean distance from input : 40.25200677130626
==========



ASIN : B0748ZQQ49
Brand : Buddys USA
euclidean distance from input : 40.31982399040467
==========

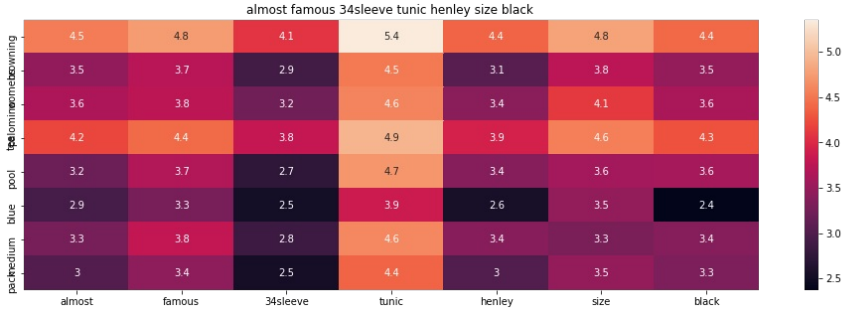


ASIN : B0154OHQZS
Brand : DZT1968
euclidean distance from input : 40.37646689538756
==========



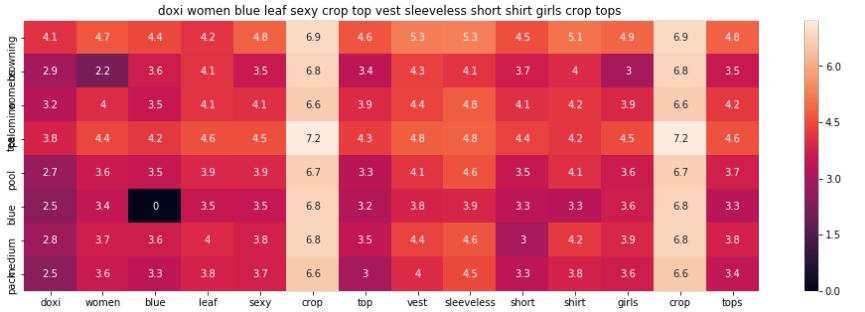
ASIN : B011TOPBRC
Brand : HP-LEISURE
euclidean distance from input : 40.53558781153026

==========



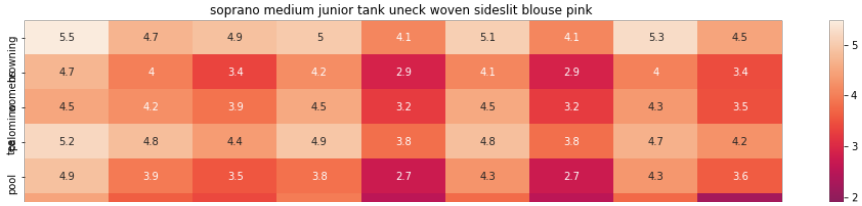
ASIN : B01I2IRKLI
Brand : Almost Famous
euclidean distance from input : 40.56040290991465

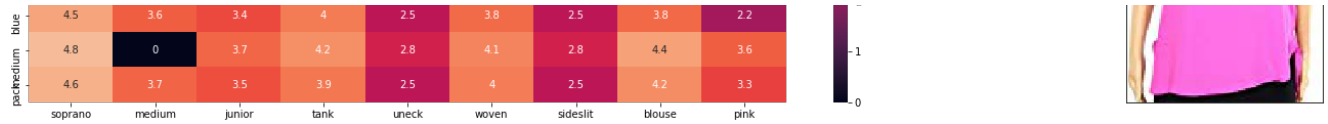
==========



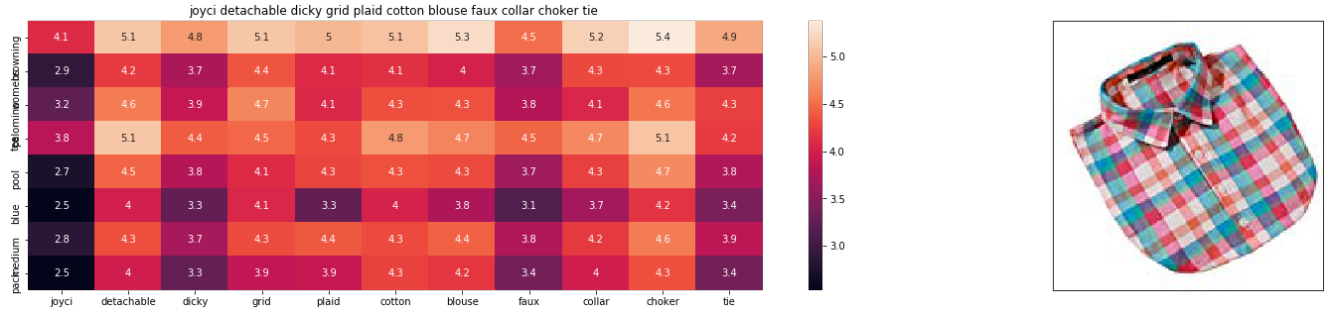
ASIN : B01LF90Q0I
Brand : Doxi Supermall
euclidean distance from input : 40.67930912176768

==========

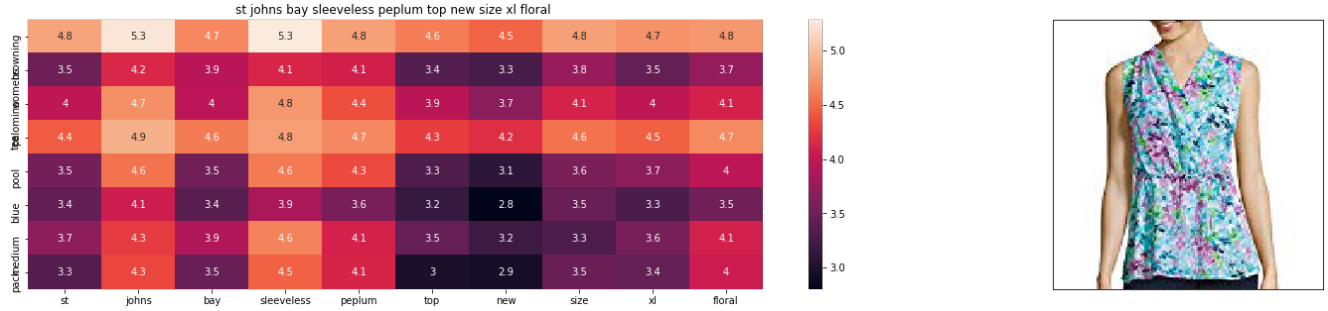




ASIN : B072BXFTKQ
Brand : Soprano
euclidean distance from input : 40.80588668311237
==========



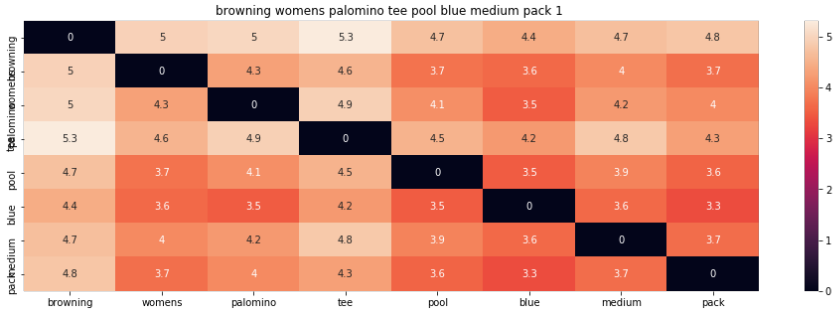
ASIN : B017U6WXG0
Brand : Joyci
euclidean distance from input : 40.88375905009338
==========



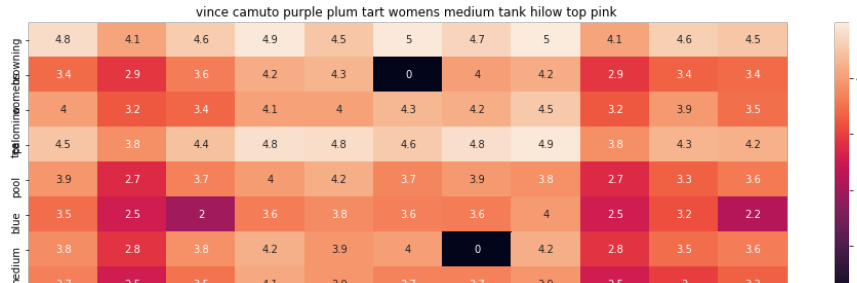
ASIN : B01N0IDEI6
Brand : St. John's Bay
euclidean distance from input : 40.88386887389729
==========

In [121]:

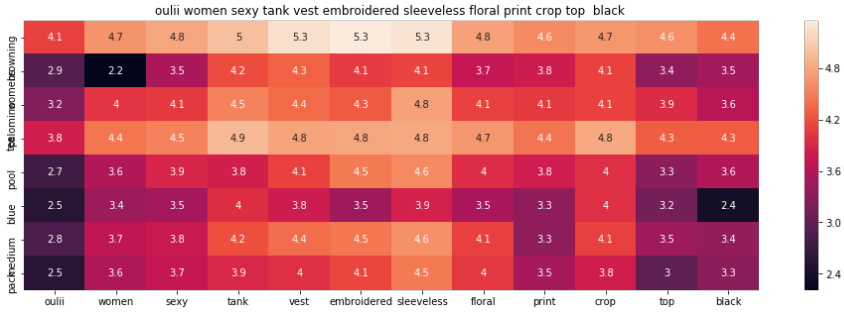
```
# giving more priority to the image
idf_w2v_brand_image(12000,5, 5, 20, 20)
```



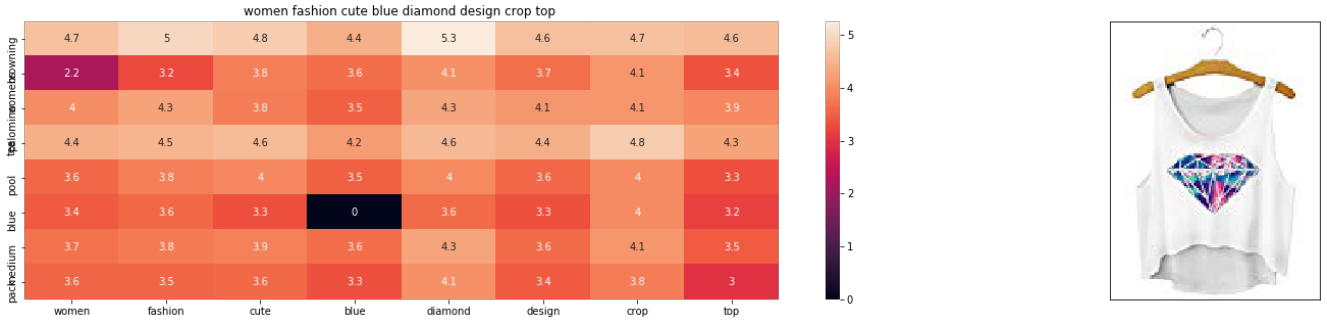
ASIN : B074KQ3SY1
Brand : Browning
euclidean distance from input : 4.312089489152034e-06
==========



ASIN : B06XTR6T7J
Brand : Vince Camuto
euclidean distance from input : 30.11003251974243
==========

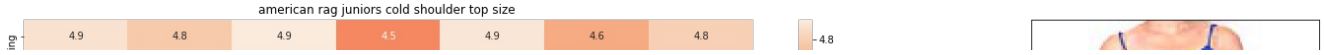


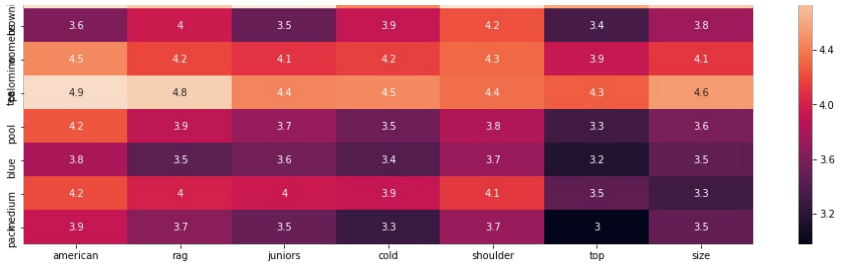
ASIN : B074KZ12G1
Brand : OULII
euclidean distance from input : 31.988383708886275



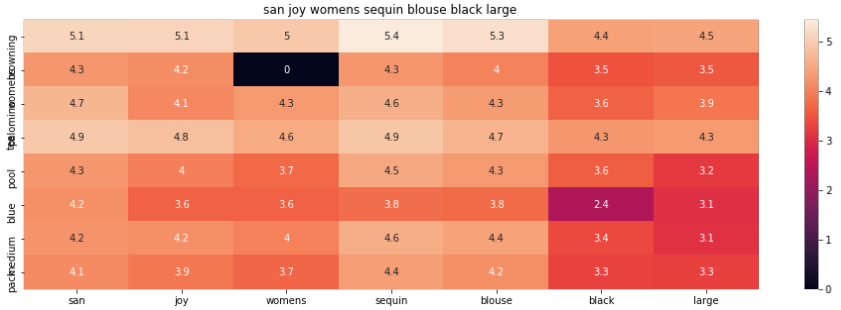
ASIN : B010KALHDI
Brand : MKP Crop Top
euclidean distance from input : 32.08752747007225

ASIN : B07538JKWD
Brand : Top Shop
euclidean distance from input : 32.13615170423645

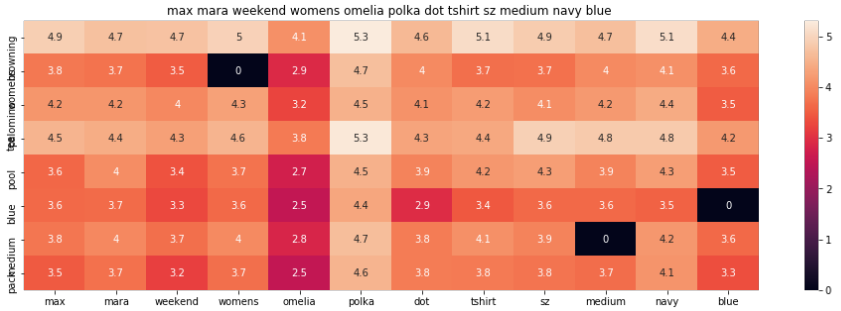




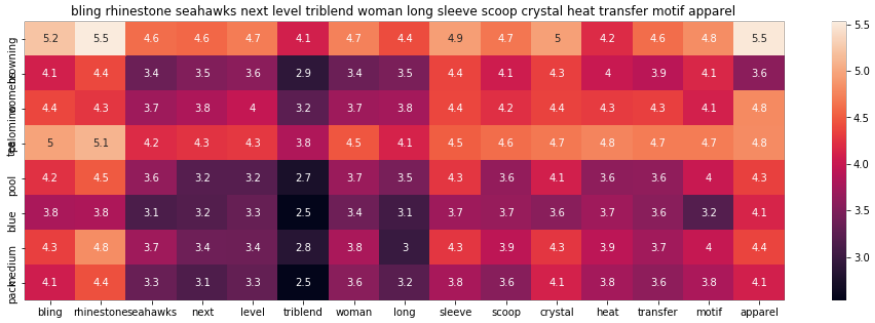
ASIN : B06XY6HK4Z
Brand : American Rag
euclidean distance from input : 32.168909136574726
==========
==========



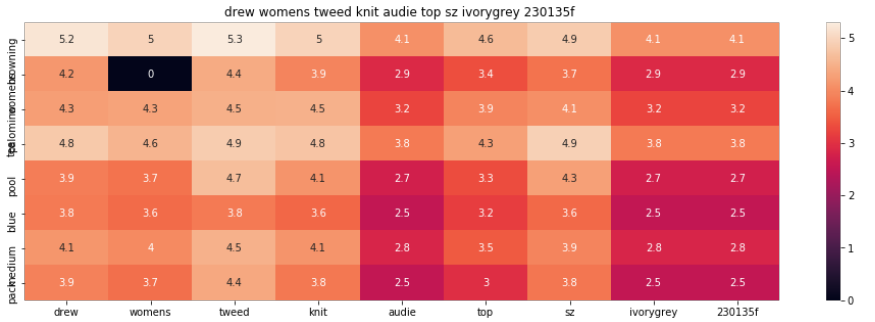
ASIN : B073WKFKLZ
Brand : Sanjoy
euclidean distance from input : 32.19922569840349
==========
==========



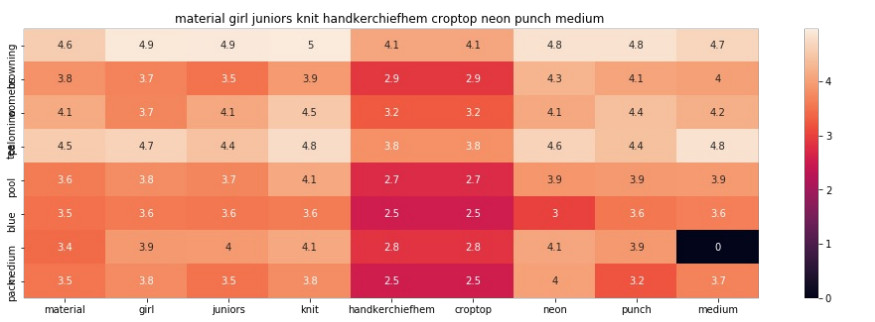
ASIN : B0749RP46C
Brand : MaxMara
euclidean distance from input : 32.34907964070638
==========
==========



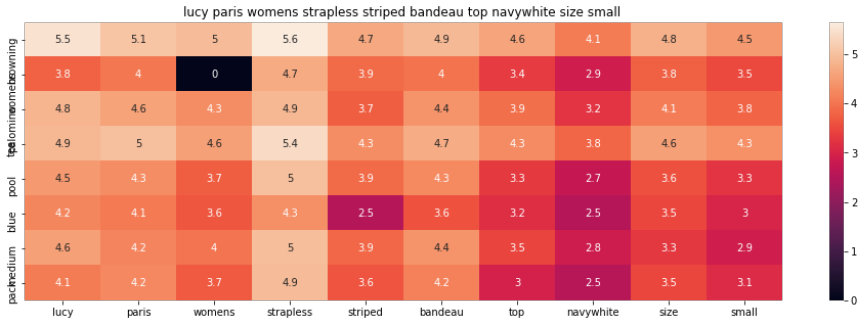
ASIN : B019HKGIR2
Brand : Crystal Trends
euclidean distance from input : 32.465137665514725
==========



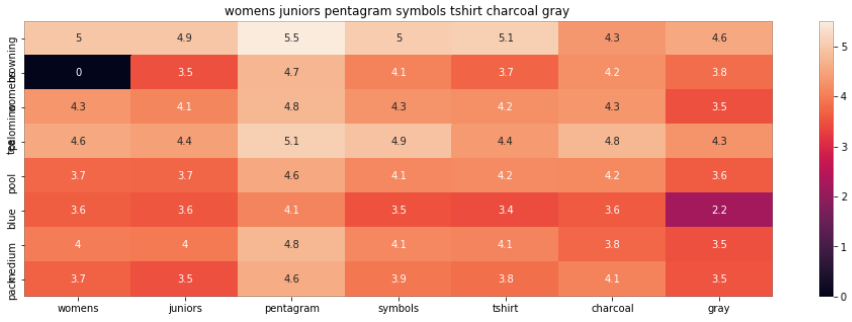
ASIN : B01ETLYQ6E
Brand : Drew Shoe
euclidean distance from input : 32.54461633789553
==========



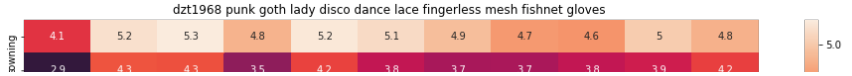
ASIN : B01NC3AFJ9
Brand : Material Girl
euclidean distance from input : 32.54972437647992
==========
==========

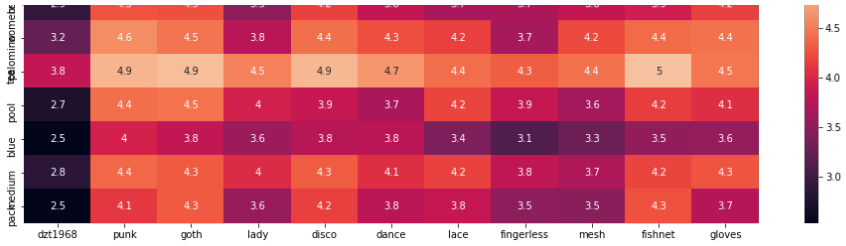


ASIN : B01LFTRTXU
Brand : Lucy Paris
euclidean distance from input : 32.59845323987816
==========
==========

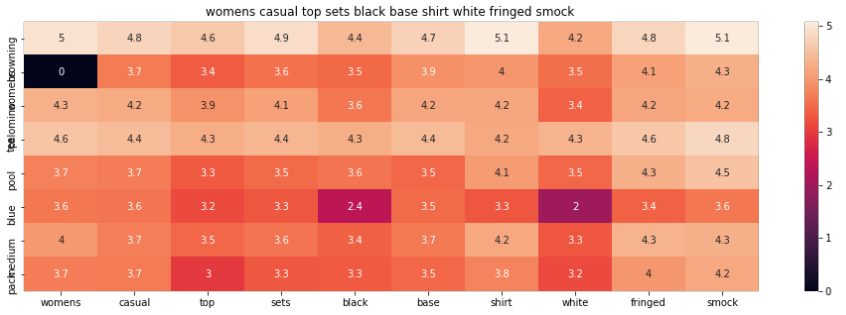


ASIN : B0748ZQQ49
Brand : Buddys USA
euclidean distance from input : 32.65549474187706
==========
==========

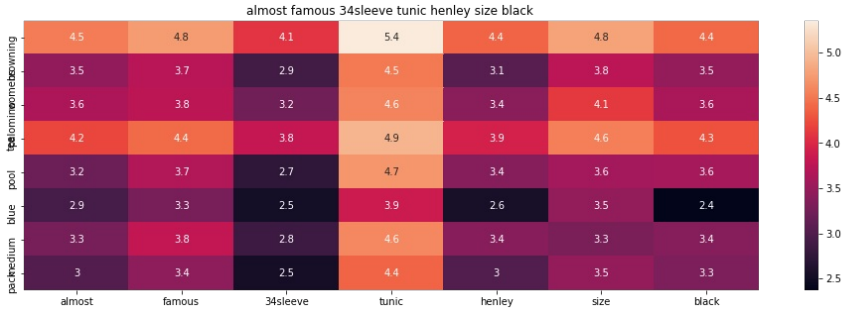




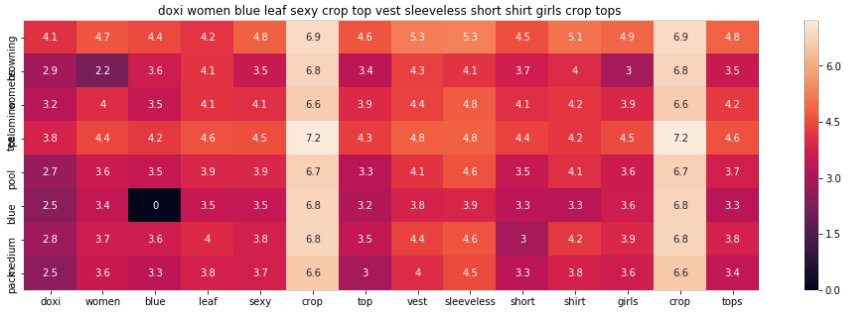
ASIN : B0154OHQZS
Brand : DZT1968
euclidean distance from input : 32.68706871916053
==========



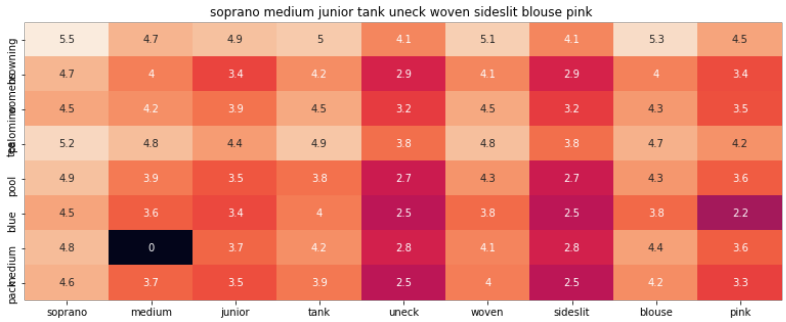
ASIN : B011TOPBRC
Brand : HP-LEISURE
euclidean distance from input : 32.83022103321677
==========



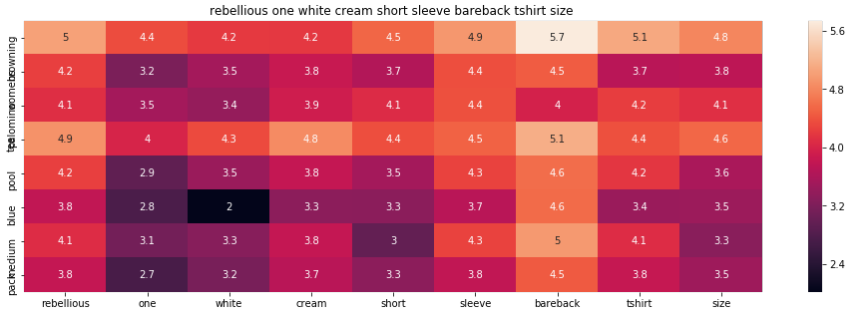
ASIN : B01I2IRKLI
Brand : Almost Famous
euclidean distance from input : 32.872527011235555
==========



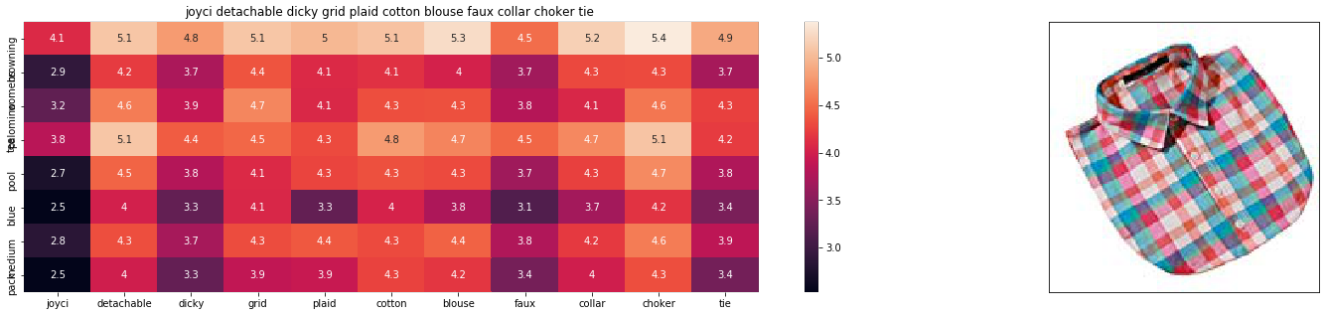
ASIN : B01LF90Q0I
Brand : Doxi Supermall
euclidean distance from input : 32.881677325566606
==========



ASIN : B072BXFTKQ
Brand : Soprano
euclidean distance from input : 32.98514372755286
==========



```
ASIN : B01H5JFDVE
Brand : Rebellious One
euclidean distance from input : 33.09350103641011
================================================================================================================
======================
```




```
ASIN : B017U6WXG0
Brand : Joyci
euclidean distance from input : 33.094288119718016
================================================================================================================
======================
```

Conclusions:

- Experimented with data set of 16k datapoints out of 183k for final model , after processing through two stage duplication removal.
- Considered Bow, TFIDF, W2v, idf based similarity methods for title id 12000.
- Compare to BOW , Tfidf performed better as it is term frequency based model with weightage.
- Using Visual similarity of Brand and Color got good recommendations with respective brand and color.
- Finally experimented weighted simialrity with text description, image , color, brand.
- Experimented with giving more weightage to image by passing multiple values to defined function.
- when tried with different weights of image observed not much change compare to previous weights.
- This can be changed or altered if we try on whole 183k data set as no of images will be high.
- Giving much weightage to brand also recommended simliarity is high.