

# Amazon Apparel Recommendations

## [4.2] Data and Code:

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

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

```
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(['sku', 'asin', 'product_type_name', 'formatted_price', 'author',
       'color', 'brand', 'publisher', 'availability', 'reviews',
       'large_image_url', 'availability_type', 'small_image_url',
       'editorial_review', 'title', 'model', 'medium_image_url',
       'manufacturer', 'editorial_reivew'],
      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 and black stripes )
4. product\_type\_name (type of the apparel, 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

	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 [16]:

```
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 [17]:

```
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 [18]:

```
# 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 [19]:

```
# consider products which have color information
# data['color'].isnull() => gives the information about the dataframe row's which have null values price == Non
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 their 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 [20]:

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

In [21]: # 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[21]: "  
from 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('images/28k\_images/' + row['asin'] + '.jpeg')\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



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 Olympi...	\$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()
```

	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 duplicate titles that differ only in the last few words.

Title 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 [27]: indices = []
for i, row in data_sorted.iterrows():
    indices.append(i)
```

```
In [28]: 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', 'Diamonds', 'Women'
    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', 'W
        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 append None in
        # 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 those two appear
        # if the number of words in which both strings differ are < 2 , we are considering it as those two appear
        if (length - count) > 2: # number of words in which both sentences 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]])

        # if the comparison between is between num_data_points, num_data_points-1 strings and they differ
        if j == num_data_points-1: stage1_dedupe_asins.append(data_sorted['asin'].loc[indices[j]])

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

```
In [29]: data = data.loc[data['asin'].isin(stage1_dedupe_asins)]
```

We removed the duplicates which differ only at the end.

```
In [30]: print('Number of data points : ', data.shape[0])
Number of data points : 17593
```

```
In [31]: data.to_pickle('pickels/17k_appral_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 [32]: data = pd.read_pickle('pickels/17k_apperal_data')
```

```
In [150]: '''# 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 apparel's title  
    a = data['title'].loc[i].split()  
    # store the list of words of ith string in a, ex: a = ['tokidoki', 'The', 'Queen', 'of', 'Diamonds', 'Women'  
    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', 'W  
        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  
        # 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 those two appare:  
            if (length - count) < 3:  
                indices.remove(j)'''
```

```
Out[150]: "# This code snippet takes significant amount of time.\n# O(n^2) time.\n# Takes about an hour to run on a dec  
ent computer.\n\nindices = []\nfor i, row in data.iterrows():\n    indices.append(i)\nstage2_dedupe_asins = []\nwhile len(indices)!=0:\n    i = indices.pop()\nstage2_dedupe_asins.append(data['asin'].loc[i])\n    # consider the first apparel's title\n    a = data['title'].loc[i].split()\n        # store the list of words of it  
h string in a, ex: a = ['tokidoki', 'The', 'Queen', 'of', 'Diamonds', 'Women's', 'Shirt', 'X-Large']\n    for j in indices:\n        \nb = data['title'].loc[j].split()\n            # store the list of words of jth s  
tring in b, ex: b = ['tokidoki', 'The', 'Queen', 'of', 'Diamonds', 'Women's', 'Shirt', 'X-Large']\n        length = max(len(a),len(b))\n            \n            # count is used to store the number of words that are matched  
in both strings\n            count = 0\n            \n            # itertools.zip_longest(a,b): will map the corresponding words  
in both strings, it will appened None in case of unequal strings\n            # example: a =['a', 'b', 'c', 'd']\n            \n            # b = ['a', 'b', 'd']\n            # itertools.zip_longest(a,b): will give [('a','a'), ('b','b'), ('c','d'), ('d', None)]\n            for k in itertools.zip_longest(a,b):\n                \n                if (k[0]==k[1]):\n                    count += 1\n                \n                # if the number of words in which both strings differ are < 3 , we are considering it a  
s those two apperals are same, hence we are ignoring them\n                if (length - count) < 3:\n                    indices.remove(j)"
```

```
In [34]: # 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 [35]: 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: 358

```
In [36]: 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 [2]: 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 [3]: # we use the list of stop words that are downloaded from nltk lib.
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: {'yourself', 'against', "you're", 'from', 'during', 'through', "she's", 'each', 'just', 'is', 'again', 'my', 'm', 'd', 'myself', 'our', 'himself', 'a', "hadn't", "that'll", 'down', 'had', 'those', 'me', 'into', 'are', 's', 'before', "shan't", 'so', 'can', 'won', 'most', 'of', 'should', "didn't", "aren't", 'having', 'have', 'ours', "couldn't", 'than', "wouldn't", 'yours', 'about', 'isn', 'don', 'with', 'what', 'a m', 'off', 'or', 'being', 'between', 'she', 'aren', 'her', "don't", 'while', 't', 'for', 'below', 'mustn', 'o ver', 'him', 'you', 'be', 'do', 'which', 'theirs', 'your', 'by', 'where', 'why', "you'd", "should've", 'and', 'then', 'other', 'on', 'how', 'when', 'if', 'to', 'after', 'ourselves', 'up', 'same', 'in', 'themselves', 'it self', 'i', 'he', 'out', 'few', 'been', 'will', 'y', 're', "haven't", "mightn't", "won't", 'weren', 'until', 'shan', 'any', 'very', 'wouldn', 'hers', 'did', 'doing', 'needn', 'that', 'all', 'weren', 'further', 've', 'only', 'doesn', 'there', 'couldn', 'haven', 'we', "mustn", "shouldn", 'its', 'no', 'own', 'shouldn', 'yourselves', 'nor', 'above', 'his', 'this', "wasn", 'under', "you'll", 'them', 'but', 'more', 'they', 'her e', 'too', 'once', 'hadn', 'the', 'because', "isn", 'ma', 'herself', 'at', 'both', 'not', 'didn', 'as', 'we re', 'wasn', 'some', 'whom', "it's", 'such', 'ain', 'does', 'now', 'o', 'it', 'mightn', 'has', 'doesn', 'yo u've', 'hasn', 'these', 'an', 'll', "hasn't", 'their', 'was', "needn", 'who'}

```
In [4]: 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")
```

0.1632009999999993 seconds

```
In [5]: data.head()
```

	asin	brand	color	medium_image_url	product_type_name	title	formatted_price
178059	B073MQDFVF	Worthington	Multi	https://images-na.ssl-images-amazon.com/images...	ACCESSORY	worthington long sleeve crew neck woven blouse...	\$26.99
178088	B0714MN43W	Two by Vince Camuto	Pink	https://images-na.ssl-images-amazon.com/images...	SHIRT	two vince camuto womens small hilow tank top p...	\$22.99
178095	B06XW47599	Focal20	Blue	https://images-na.ssl-images-amazon.com/images...	SHIRT	focal20 womens cartoon cat cardigan long sleev...	\$18.99
178110	B0759NH3ZR	Tommy Hilfiger	Blue	https://images-na.ssl-images-amazon.com/images...	SHIRT	tommy hilfiger women large embroidered button ...	\$27.15
178127	B06XSG711M	Greylin	Plum	https://images-na.ssl-images-amazon.com/images...	SHIRT	greylin womens chiffon peplum silk top l purple	\$34.99

```
In [6]: data.to_pickle('pickels/16k_appral_data_preprocessed')
```

## Stemming

```
In [7]: 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 [13]: data = pd.read_pickle('pickels/16k_apperal_data_preprocessed')
data.head()
```

Out[13]:

	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 [87]: # Utility Functions which we will use through the rest of the workshop.
import PIL.Image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%pylab inline

#Display an image
def display_img(url,ax,fig):
    # we get the url of the apparel and download it
    response = requests.get(url)
    img = PIL.Image.open(BytesIO(response.content)).convert("RGB")
    # we will display it in notebook
    plt.imshow(img)
    #plotting code to understand the algorithm's decision.

#Display an image
def display_1(url):
    # we get the url of the apparel and download it
    response = requests.get(url)
    img = PIL.Image.open(BytesIO(response.content)).convert("RGB")
    # 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 occurrence 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))

    # 1st, plotting heat map that represents the count of commonly occurred words in title2
    ax = plt.subplot(gs[0])
    # it displays a cell in white color if the word is intersection(list of words of title1 and list of word
    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 apparel's vector, it is of a dict type {word:count}
    # vec2 : recommended apparel's vector, it is of a dict type {word:count}
    # url : apparel's 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
    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(list of words of title1 and list of words of title2): values(i)=count of that
    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 word in given
            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 docu
            if x in idf_title_vectorizer.vocabulary_:
                labels.append(idf_title_features[doc_id, idf_title_vectorizer.vocabulary_[x]])
            else:
                labels.append(0)

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

# this function gets a list of words 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 s
    return Counter(words) # Counter counts the occurrence of each word in list, it returns dict type object {wor

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)

```

Populating the interactive namespace from numpy and matplotlib

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

```
In [20]: 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 sparse 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 occurred in that doc
```

Out[20]: (16042, 12609)

```
In [21]: 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  $K(X, Y) = \langle X, Y \rangle / (\|X\| * \|Y\|)$ 
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(title_features,title_features[doc_id])

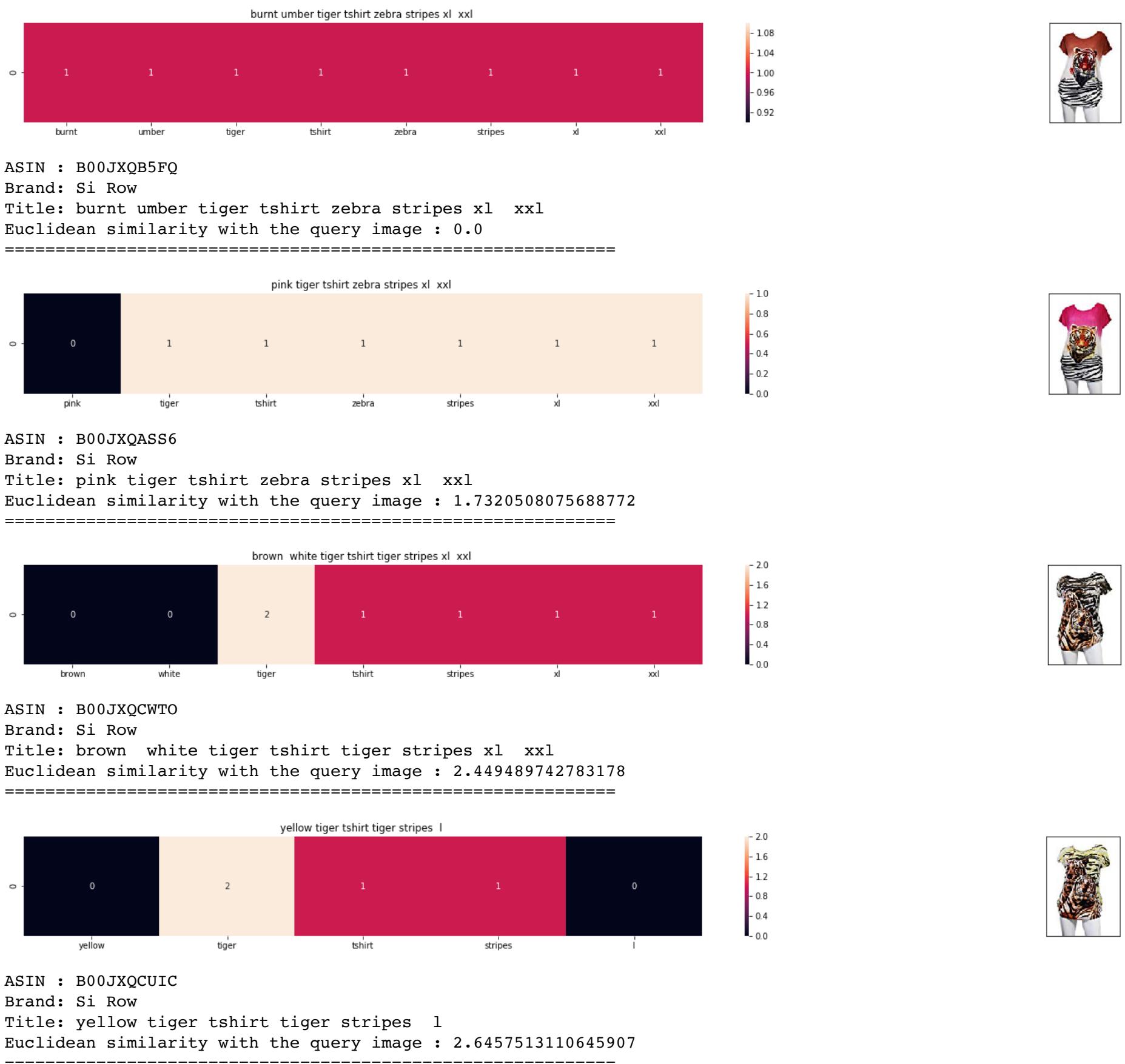
    # np.argsort will return indices of the smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the smallest distances
    pdists = np.sort(pairwise_dist.flatten())[0:num_results]

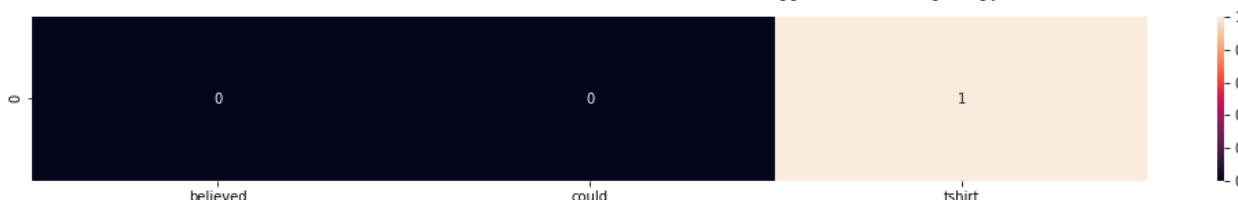
    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])

    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]], data['medium_'
        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(12566, 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 : B07568NZX4

Brand: Rustic Grace

Title: believed could tshirt

Euclidean similarity with the query image : 3.0

=====



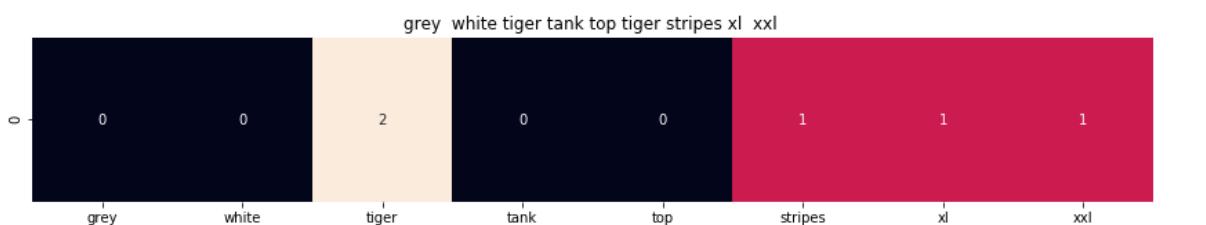
ASIN : B01NB0NKRO

Brand: Ideology

Title: ideology graphic tshirt xl white

Euclidean similarity with the query image : 3.0

=====



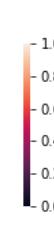
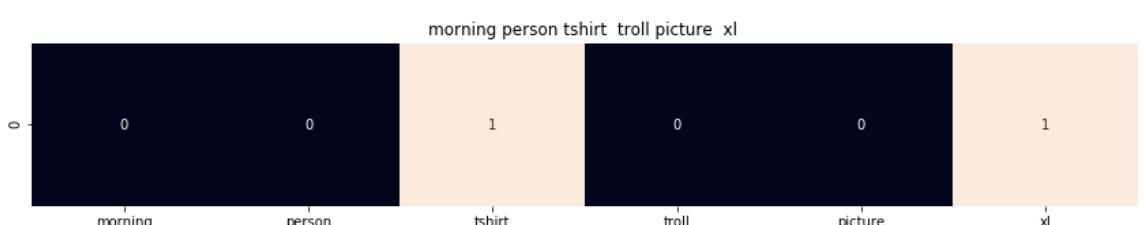
ASIN : B00JXQAFZ2

Brand: Si Row

Title: grey white tiger tank top tiger stripes xl xxl

Euclidean similarity with the query image : 3.0

=====



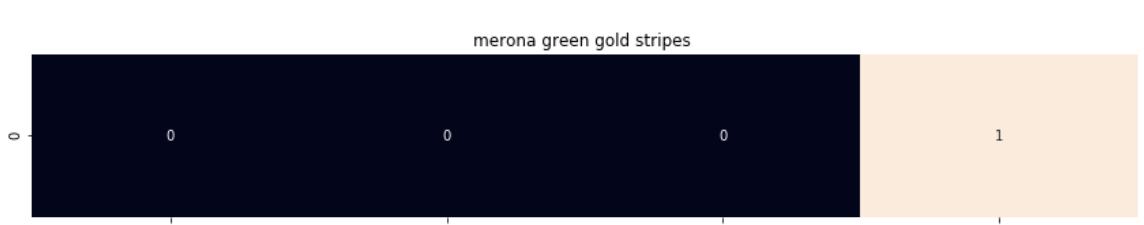
ASIN : B01CLS8LMW

Brand: Awake

Title: morning person tshirt troll picture xl

Euclidean similarity with the query image : 3.1622776601683795

=====



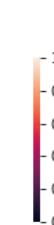
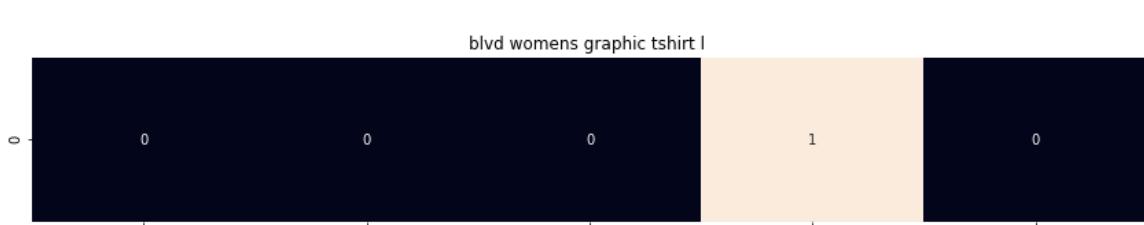
ASIN : B01KVZUB6G

Brand: Merona

Title: merona green gold stripes

Euclidean similarity with the query image : 3.1622776601683795

=====



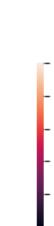
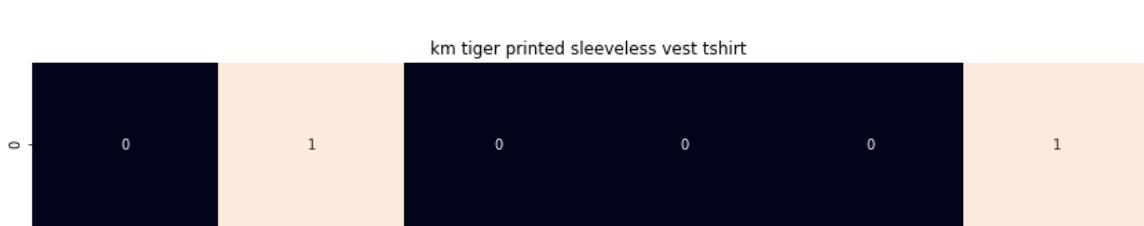
ASIN : B0733R2CJK

Brand: BLVD

Title: blvd womens graphic tshirt l

Euclidean similarity with the query image : 3.1622776601683795

=====

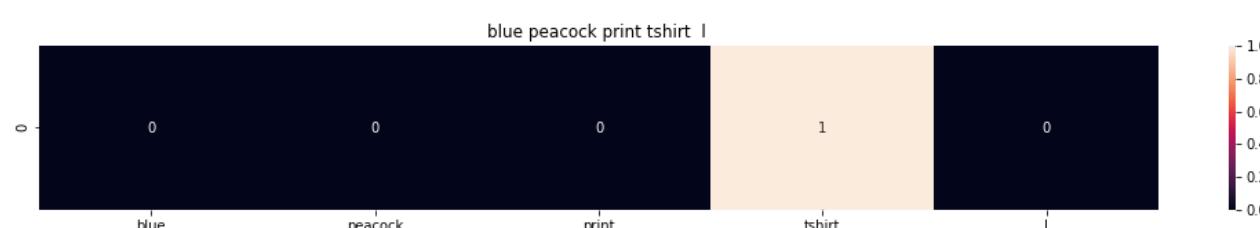


ASIN : B012VQLT6Y

Brand: KM T-shirt

Title: km tiger printed sleeveless vest tshirt

Euclidean similarity with the query image : 3.1622776601683795



ASIN : B00JXQC8L6

Brand: Si Row

Title: blue peacock print tshirt l

Euclidean similarity with the query image : 3.1622776601683795



ASIN : B06XC3CZF6

Brand: Fjallraven

Title: fjallraven womens ovik tshirt plum xxl

Euclidean similarity with the query image : 3.1622776601683795



ASIN : B005IT8OBA

Brand: Hetalia

Title: hetalia us girl tshirt

Euclidean similarity with the query image : 3.1622776601683795

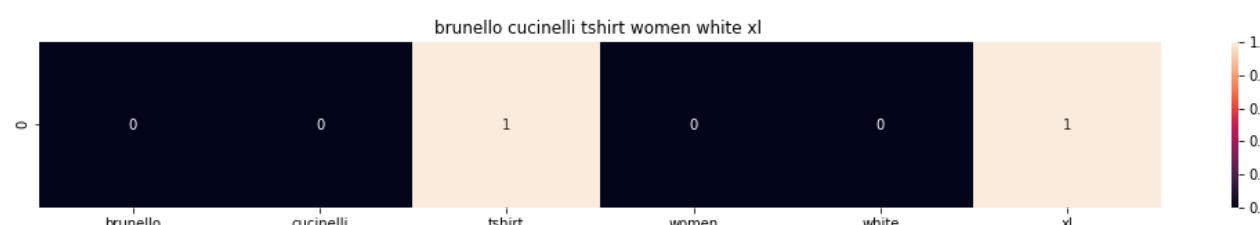


ASIN : B0088PN0LA

Brand: Red House

Title: canvas 3001 30s tshirt kelly xl

Euclidean similarity with the query image : 3.1622776601683795



ASIN : B06X99V6WC

Brand: Brunello Cucinelli

Title: brunello cucinelli tshirt women white xl

Euclidean similarity with the query image : 3.1622776601683795



ASIN : B06Y1JPW1Q

Brand: Xhilaration

Title: xhilaration womens lace tshirt salmon xxl

Euclidean similarity with the query image : 3.1622776601683795

ASIN : B06X6GX6WG  
 Brand: Animal  
 Title: animal oceania tshirt yellow  
 Euclidean similarity with the query image : 3.1622776601683795



ASIN : B017X8PW9U  
 Brand: Diesel  
 Title: diesel tserraf tshirt black  
 Euclidean similarity with the query image : 3.1622776601683795



ASIN : B00IAA4JIQ  
 Brand: I Love Lucy  
 Title: juniors love lucyaaaaahhhh tshirt size xl  
 Euclidean similarity with the query image : 3.1622776601683795

In [ ]:

## [8.5] TF-IDF based product similarity

```
In [22]: 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_
# tfidf_title_features[doc_id, index_of_word_in_corpus] = tfidf values of the word in given doc

In [ ]: 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']
        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(12566, 20)
    # in the output heat map each value represents the tfidf values of the label word, the color represents the int
```

## [8.5] IDF based product similarity

```
In [23]: 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_
# idf_title_features[doc_id, index_of_word_in_corpus] = number of times the word occured in that doc
```

```
In [24]: def nContaining(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] / (nContaining(word)))
```

```
In [25]: # 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 the word
    # idf_title_features[:, idf_title_vectorizer.vocabulary_[i]].nonzero()[0] will return all documents in which
    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_corpus] = idf value of word
        idf_title_features[j,idf_title_vectorizer.vocabulary_[i]] = idf_val
```

```
In [0]: 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  $K(X, Y) = \langle X, Y \rangle / (\|X\| * \|Y\|)$ 
    # 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)):
        get_result(indices[i],data['title'].loc[df_indices[0]], data['title'].loc[df_indices[i]], data['medium_'])
        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(12566,20)
# in the output heat map each value represents the idf values of the label word, the color represents the inter
```

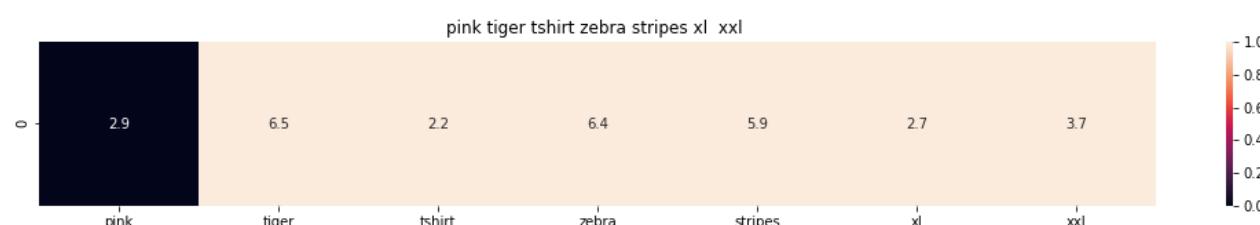


ASIN : B00JXQB5FQ  
Brand : Si Row  
euclidean distance from the given image : 0.0

---



---



ASIN : B00JXQASS6  
Brand : Si Row  
euclidean distance from the given image : 12.2050713112

---



---

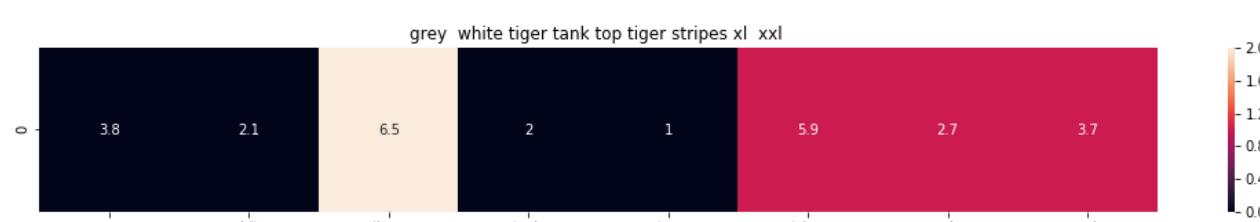


ASIN : B00JXQCWTO  
Brand : Si Row  
euclidean distance from the given image : 14.4683626856

---



---

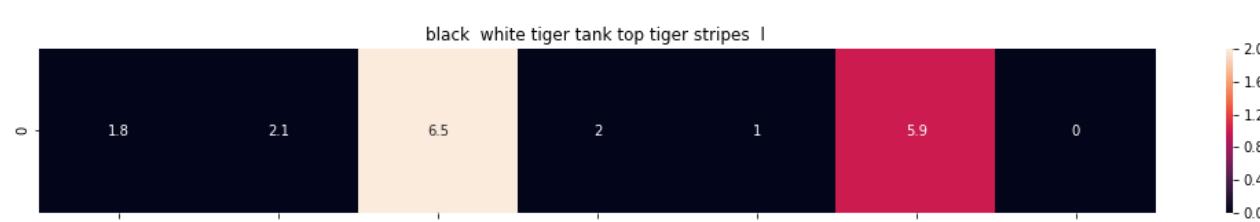


ASIN : B00JXQAFZ2  
Brand : Si Row  
euclidean distance from the given image : 14.4868329248

---



---

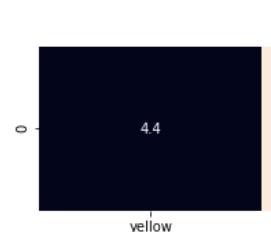


ASIN : B00JXQAO94

Brand : Si Row

euclidean distance from the given image : 14.8333929667

=====

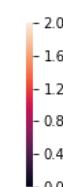
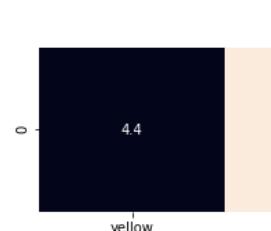


ASIN : B00JXQCUIC

Brand : Si Row

euclidean distance from the given image : 14.8987445167

=====

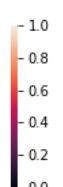


ASIN : B00JXQAUWA

Brand : Si Row

euclidean distance from the given image : 15.2244582873

=====

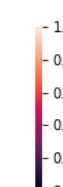
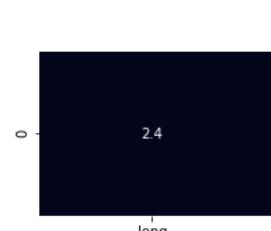


ASIN : B074T8ZYGX

Brand : MKP Crop Top

euclidean distance from the given image : 17.0808129556

=====

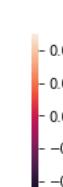
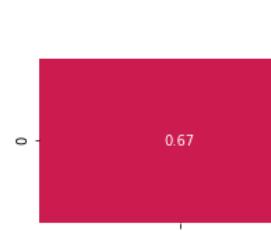


ASIN : B00KF2N5PU

Brand : Vietsbay

euclidean distance from the given image : 17.0901681256

=====

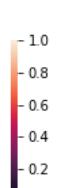
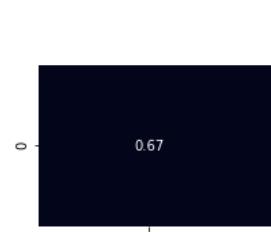


ASIN : B00JPOZ9GM

Brand : Sofra

euclidean distance from the given image : 17.1532153376

=====

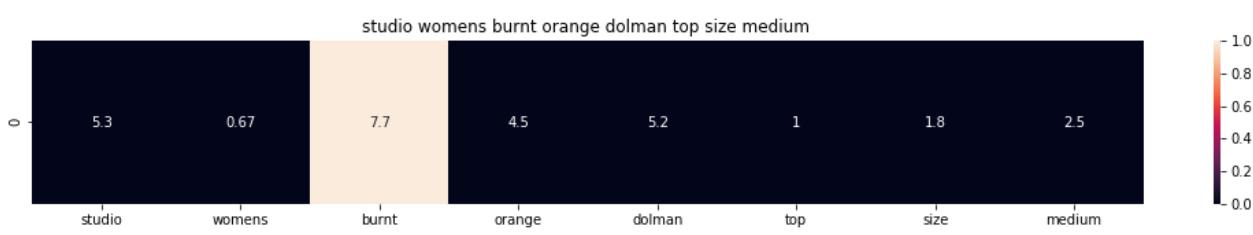
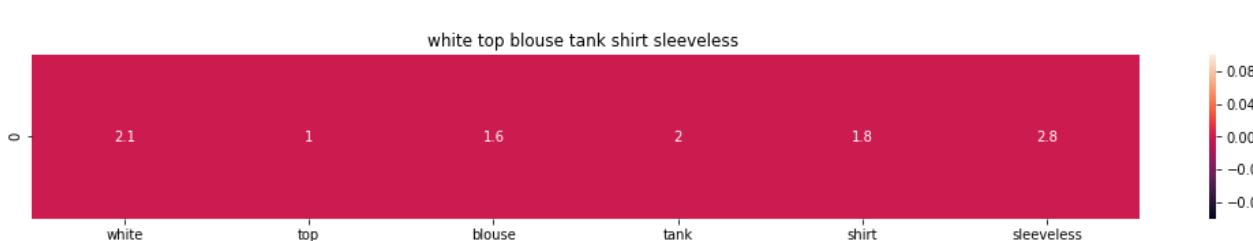
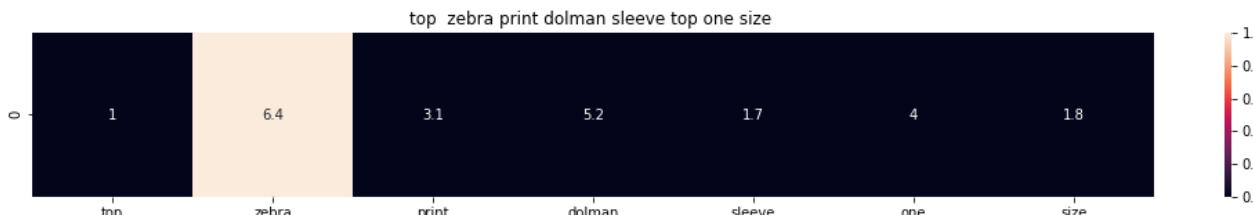


ASIN : B074T9KG9Q

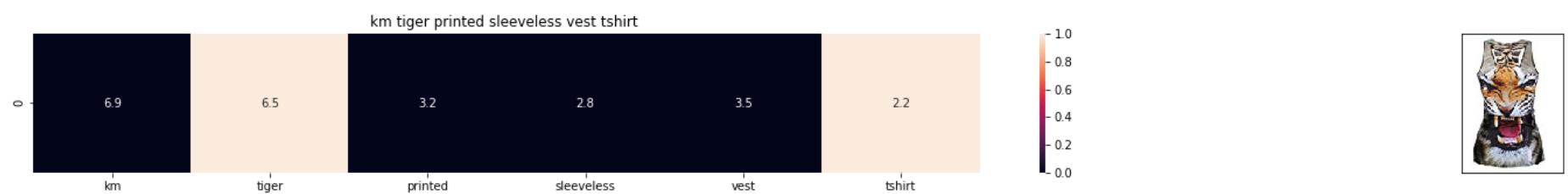
Brand : Rain

euclidean distance from the given image : 17.3367152387

=====



```
euclidean distance from the given image : 17.7230738913
=====
```



```
ASIN : B012VQLT6Y
```

```
Brand : KM T-shirt
```

```
euclidean distance from the given image : 17.7625885612
=====
```



```
ASIN : B00ZZMYBRG
```

```
Brand : HP-LEISURE
```

```
euclidean distance from the given image : 17.7795368647
=====
```

## [9] Text Semantics based product similarity

In [26]:

```
# 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.

...
# 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)

...
```

Out[26]:

```
'\n# Set values for various parameters\nnum_features = 300      # Word vector dimensionality
\nmin_word_count = 1        # Minimum word count
\nnum_workers = 4           # Number of threads to run in parallel\ncontext = 10               # Context window size
\ndownsampling = 1e-3        # Downsample setting for frequent words\n\n# Initialize and train the model (this will take some time)\nfrom gensim.models import word2vec\nprint ("Training model...")\nmodel = word2vec.Word2Vec(sen_corpus, workers=num_workers,
                           size=num_features, min_count = min_word_count,
                           window = context)\n  \n'
```

```
In [27]: 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 which contains a dict ,
# and it contains all our corpus 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/0B7XkCwpI5KDYNlNUTT1SS21pQmM/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 [28]: # 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 corpus 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 sentence
    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 wo
    # vec2 = np.array(#number_of_words_title2 * 300), each row is a vector of length 300 corresponds to each wo

    final_dist = []
    # for each vector in vec1 we calculate 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):
    # sentence1 : title1, input apparel
    # sentence2 : 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 length 300 correspon
    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 length 300 correspon
    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 apparel
    gs = gridspec.GridSpec(2, 2, width_ratios=[4,1], height_ratios=[2,1])
    fig = plt.figure(figsize=(15,15))

    ax = plt.subplot(gs[0])
    # plotting 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 [32]: # 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 sentence
def build_avg_vec(sentence, num_features, doc_id, m_name):
    # sentence: 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(wordi) to this fatureVec
    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.vocabulary_[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 [33]: 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 [0]: def avg_w2v_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

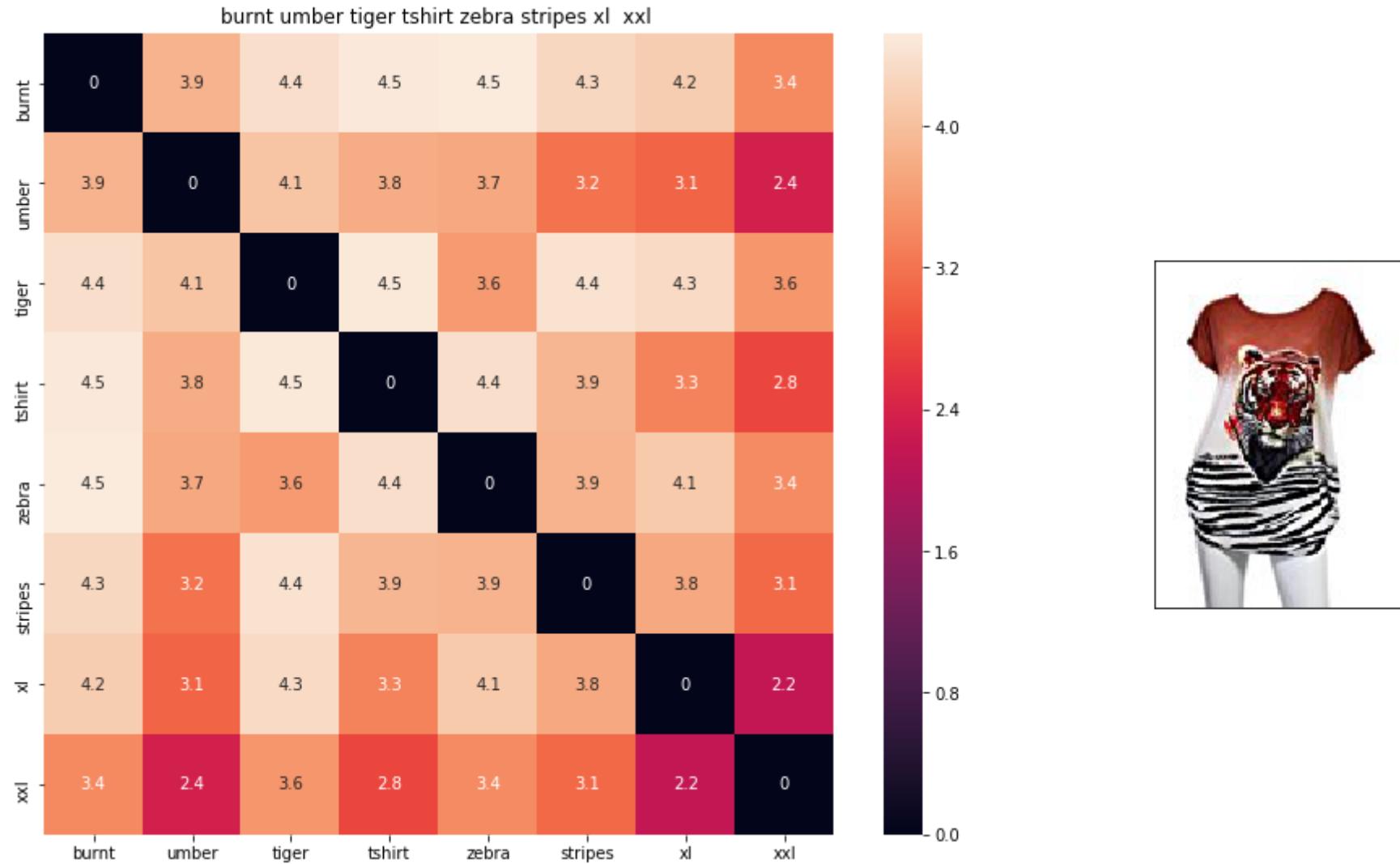
    # dist(x, y) = sqrt(dot(x, x) - 2 * dot(x, y) + dot(y, y))
    pairwise_dist = pairwise_distances(w2v_title, w2v_title[doc_id].reshape(1,-1))

    # 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]

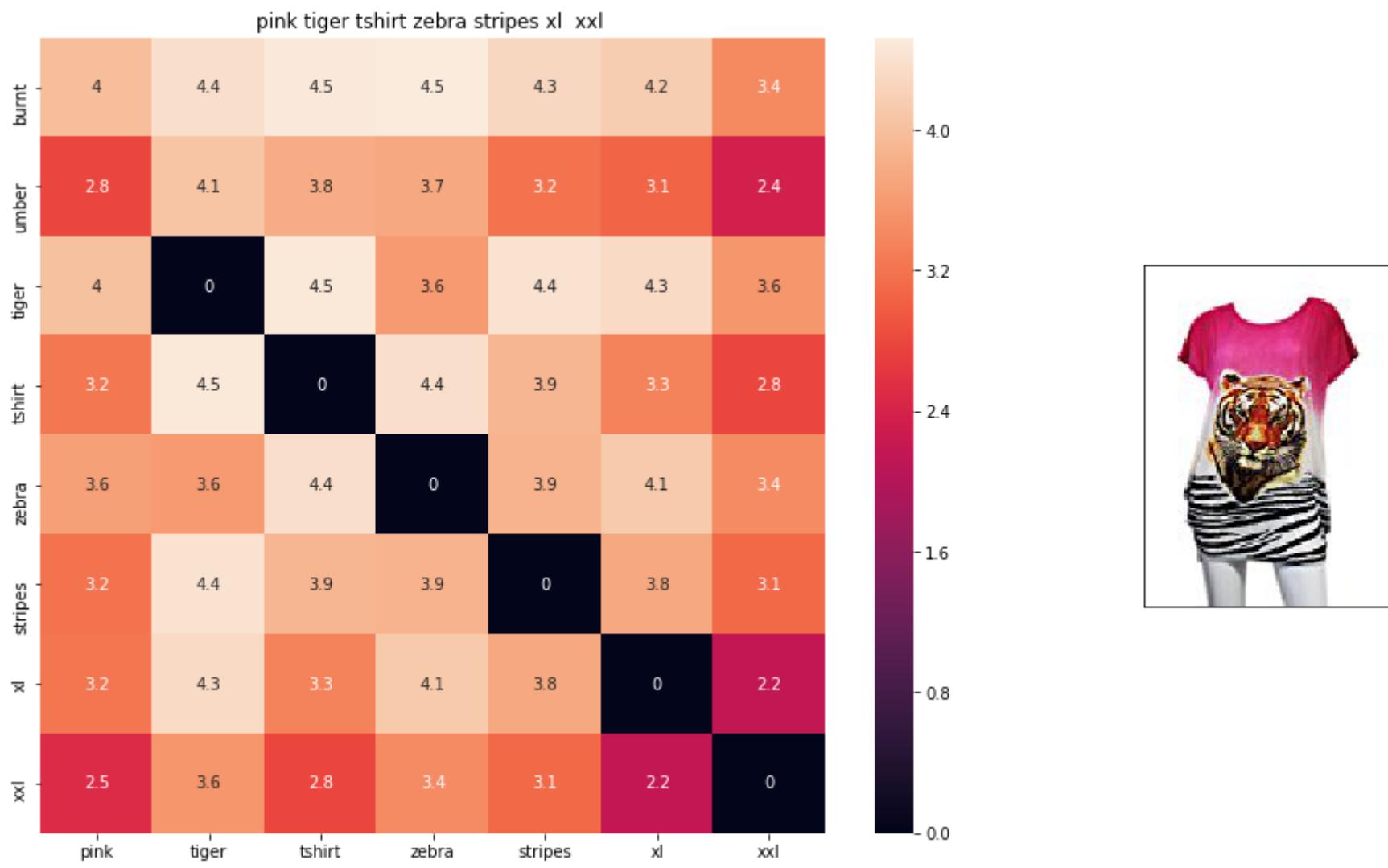
    #data frame indices of the 9 smallest distance's
    df_indices = list(data.index[indices])

    for i in range(0, len(indices)):
        heat_map_w2v(data['title'].loc[df_indices[0]], data['title'].loc[df_indices[i]], data['medium_image_url'])
        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(12566, 20)
# in the give heat map, each cell contains the euclidean distance between words i, j
```



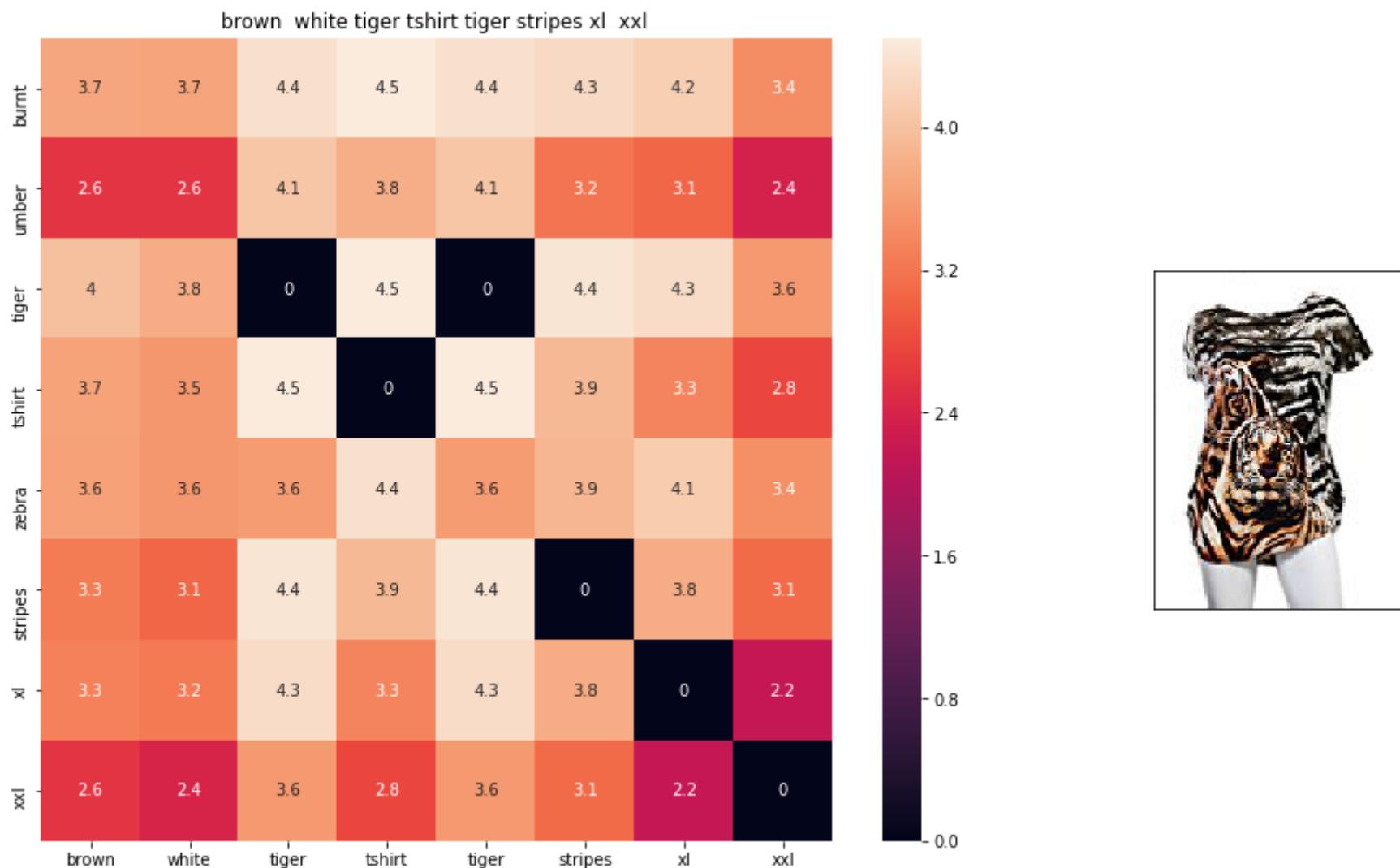
ASIN : B00JXQB5FQ  
 BRAND : Si Row  
 euclidean distance from given input image : 0.000690534  
 =====  
 =====



ASIN : B00JXQASS6

BRAND : Si Row

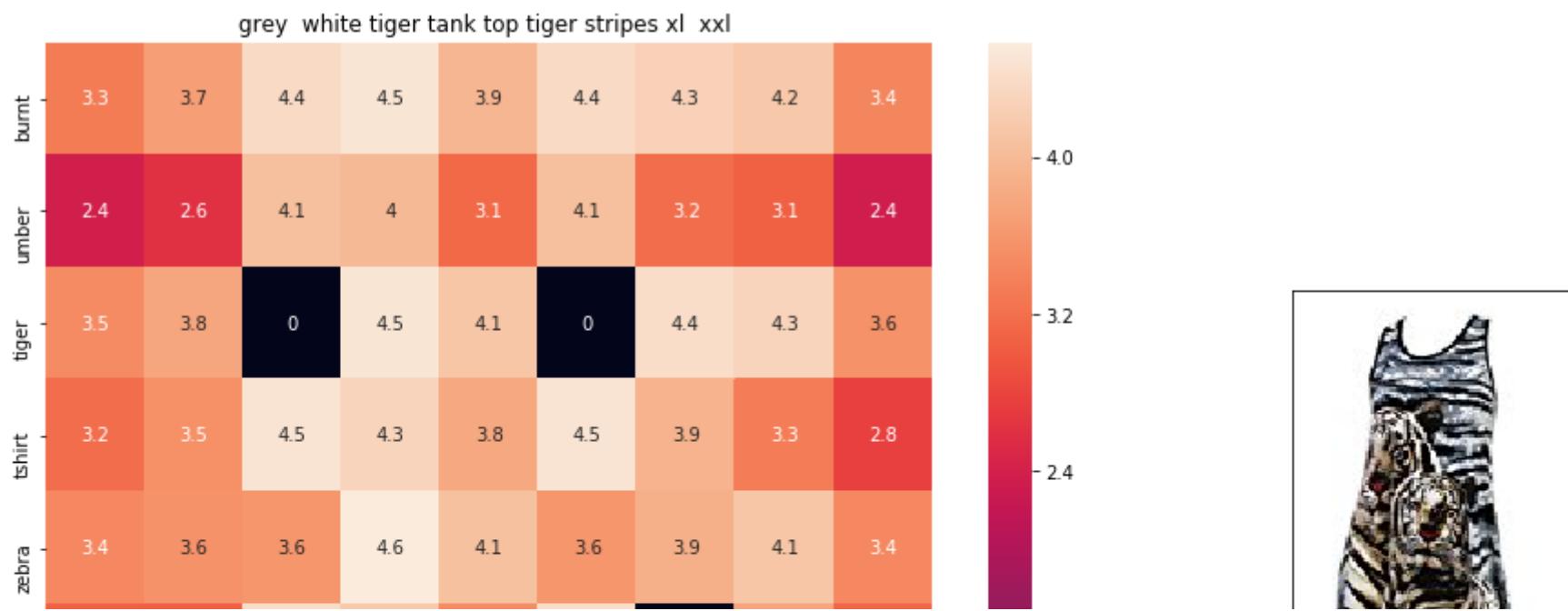
euclidean distance from given input image : 0.589193



ASIN : B00JXQCWTO

BRAND : Si Row

euclidean distance from given input image : 0.700344



ASIN : B00JXQAFZ2

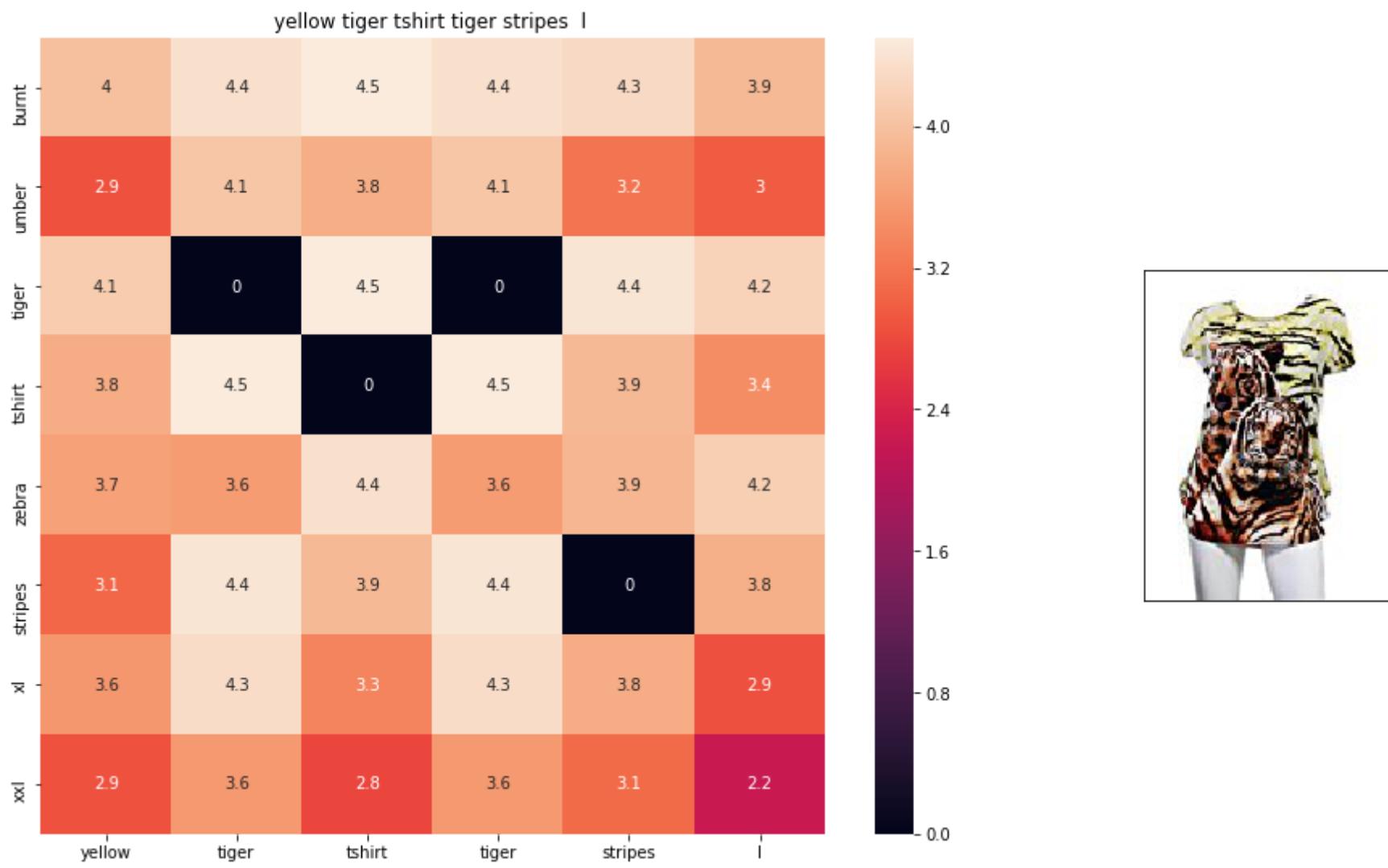
BRAND : Si Row

euclidean distance from given input image : 0.89284

---



---



ASIN : B00JXQCUIC

BRAND : Si Row

euclidean distance from given input image : 0.956013

---



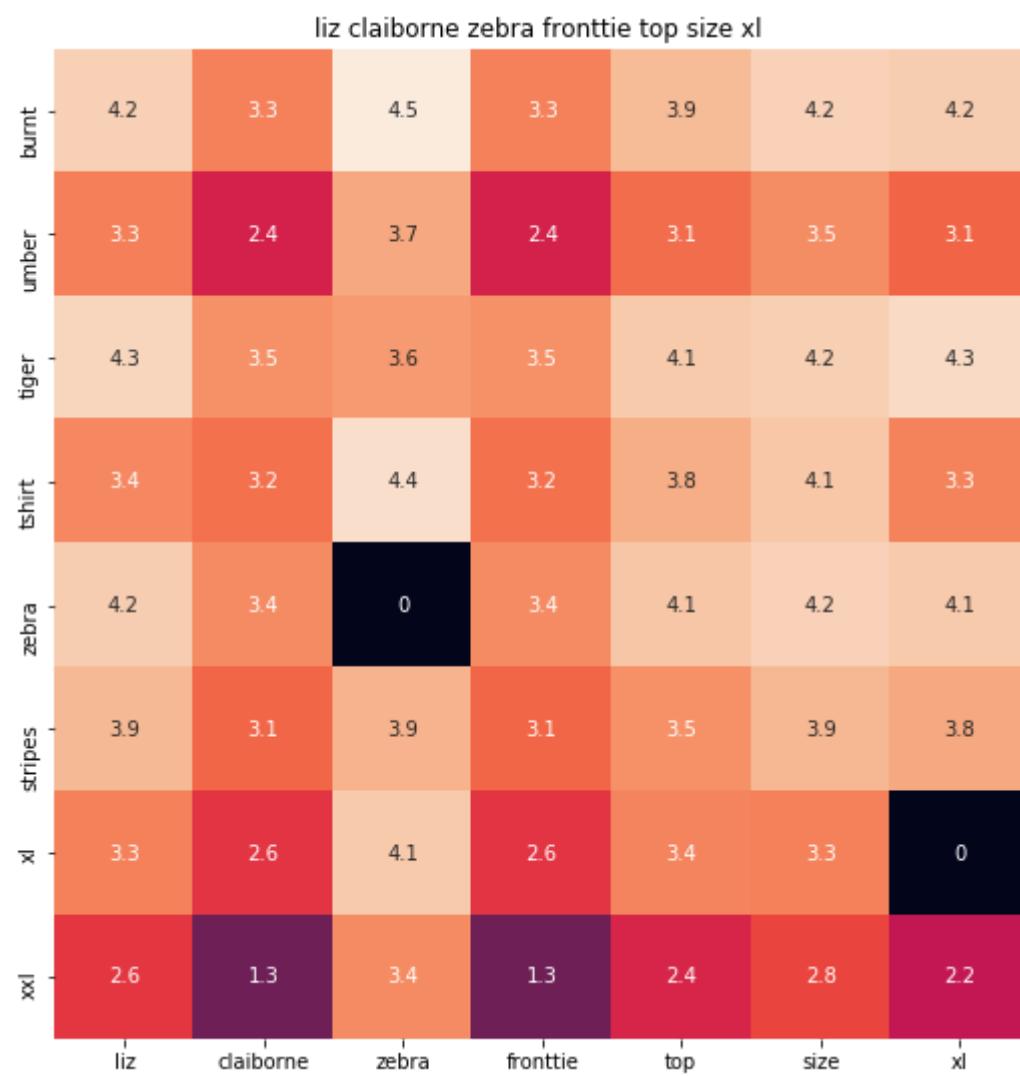
---



ASIN : B073R5Q8HD

BRAND : Colosseum

euclidean distance from given input image : 1.02297



ASIN : B06XBY5QXL

BRAND : Liz Claiborne

euclidean distance from given input image : 1.06693

hotgirl4 women owl ovo raglan triblend sleeve color black xl

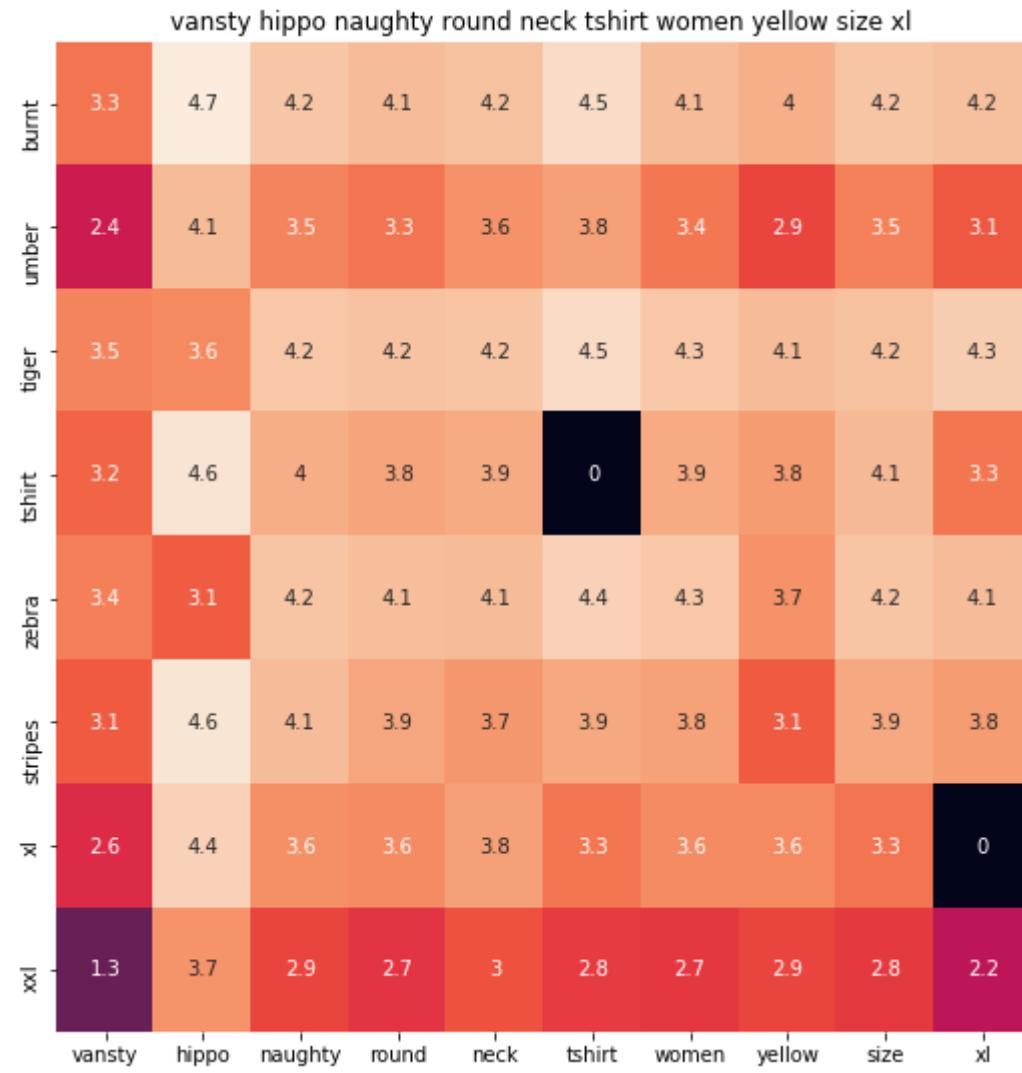
ASIN : B01L8L73M2

BRAND : Hotgirl4 Raglan Design

euclidean distance from given input image : 1.07314

=====

=====



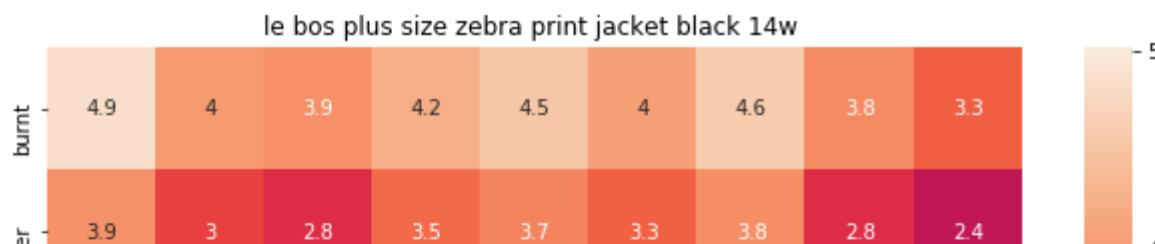
ASIN : B01EJS5H06

BRAND : Vansty

euclidean distance from given input image : 1.07572

=====

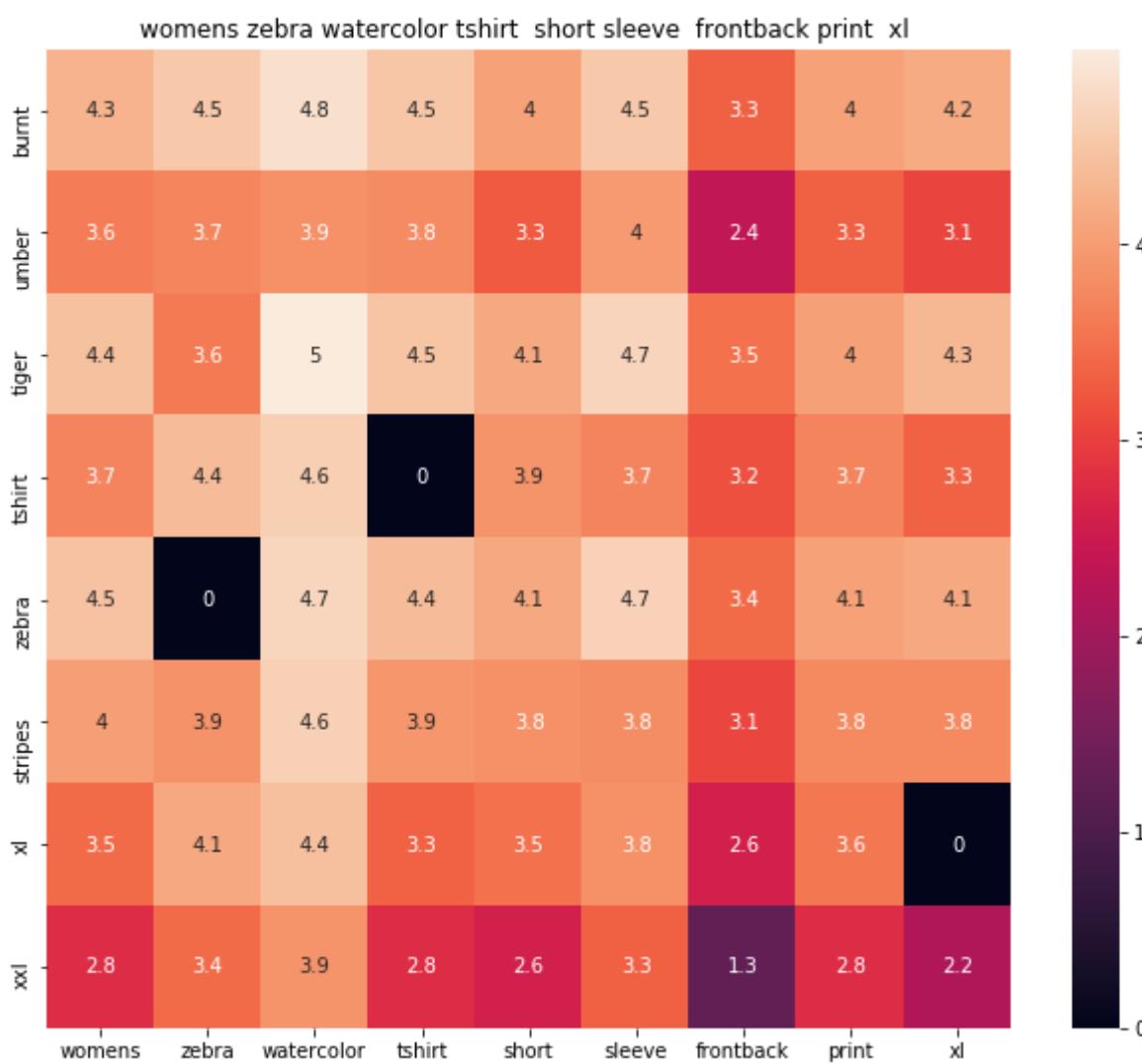
=====



ASIN : B01BO1XRK8

BRAND : Le Bos

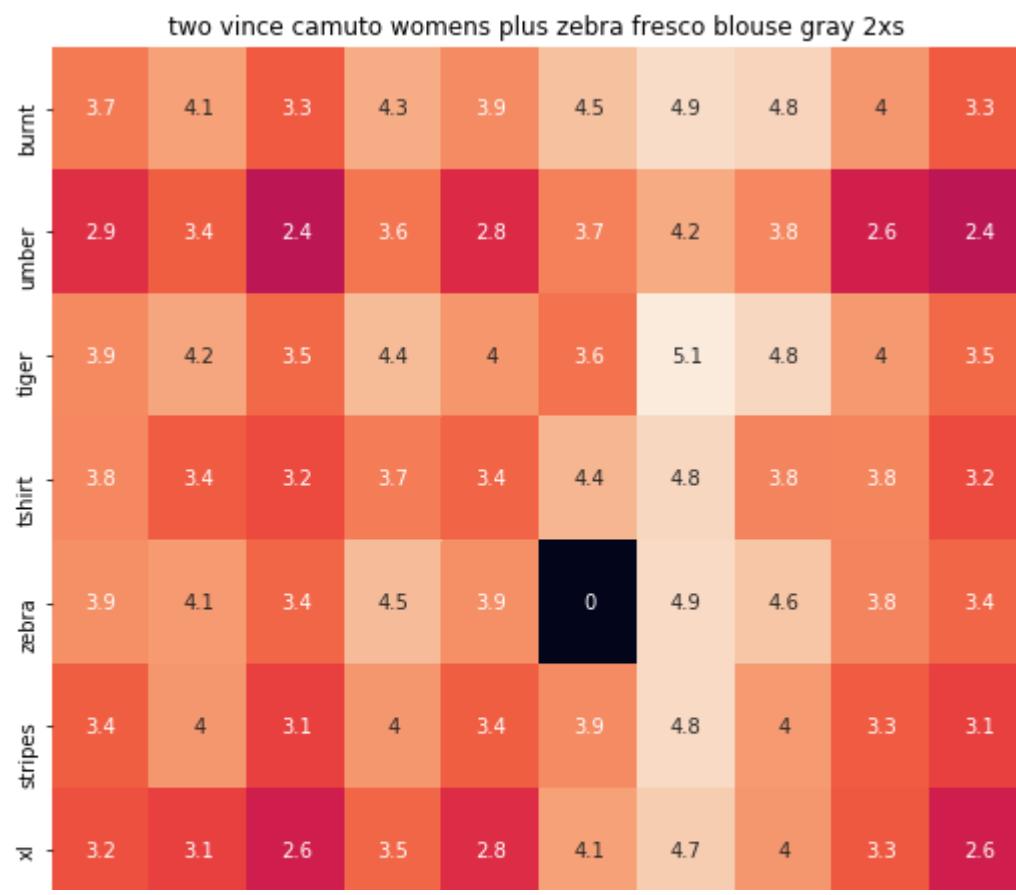
euclidean distance from given input image : 1.084



ASIN : B072R2JXKW

BRAND : WHAT ON EARTH

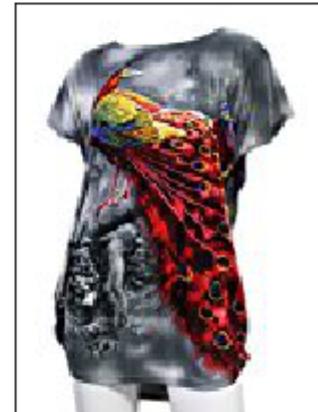
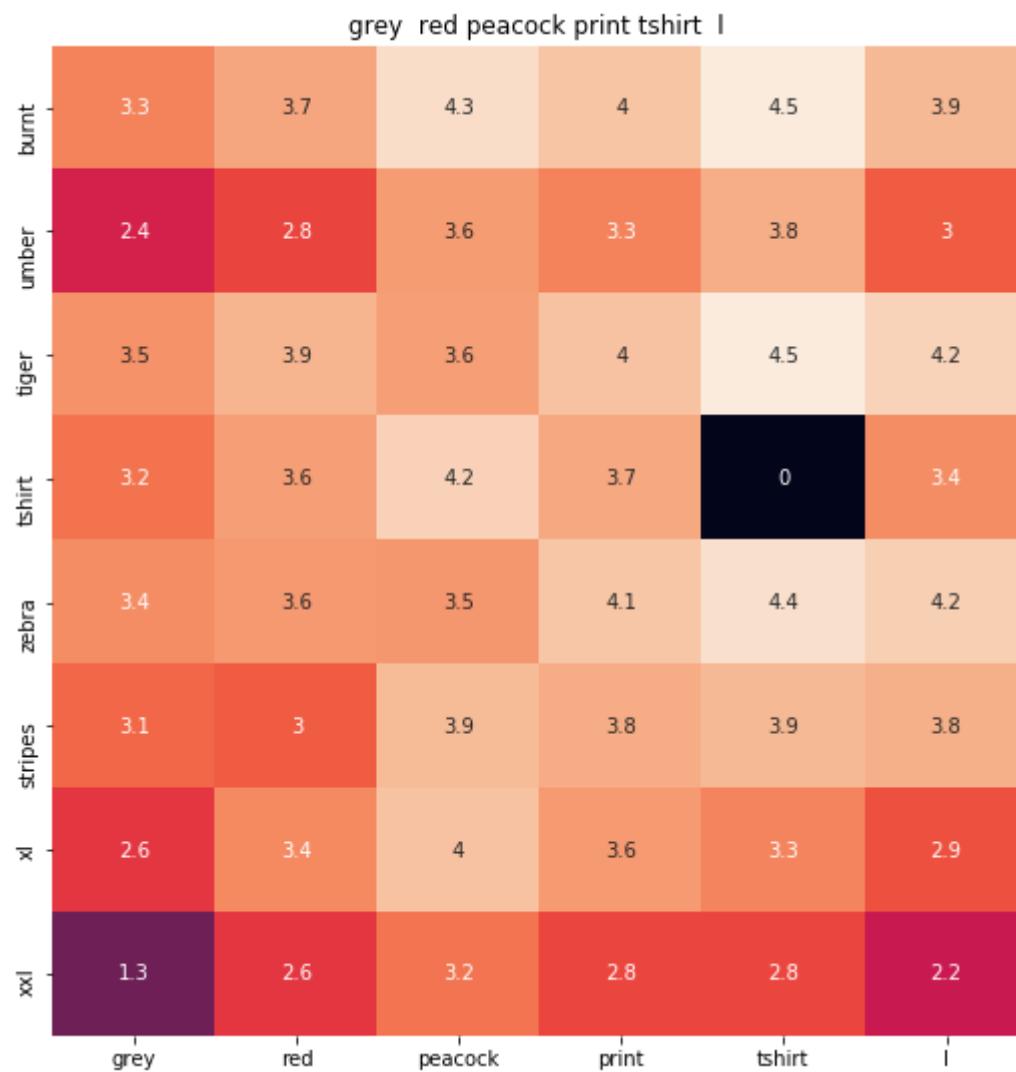
euclidean distance from given input image : 1.08422



ASIN : B074MJRGW6

BRAND : Two by Vince Camuto

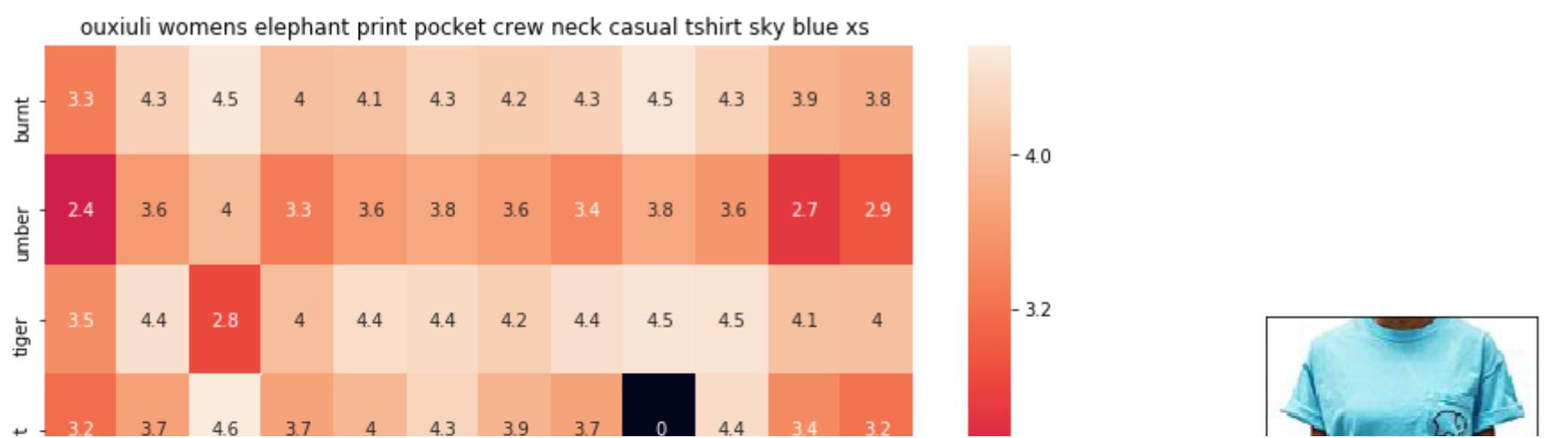
euclidean distance from given input image : 1.0895



ASIN : B00JXQCFRS

BRAND : Si Row

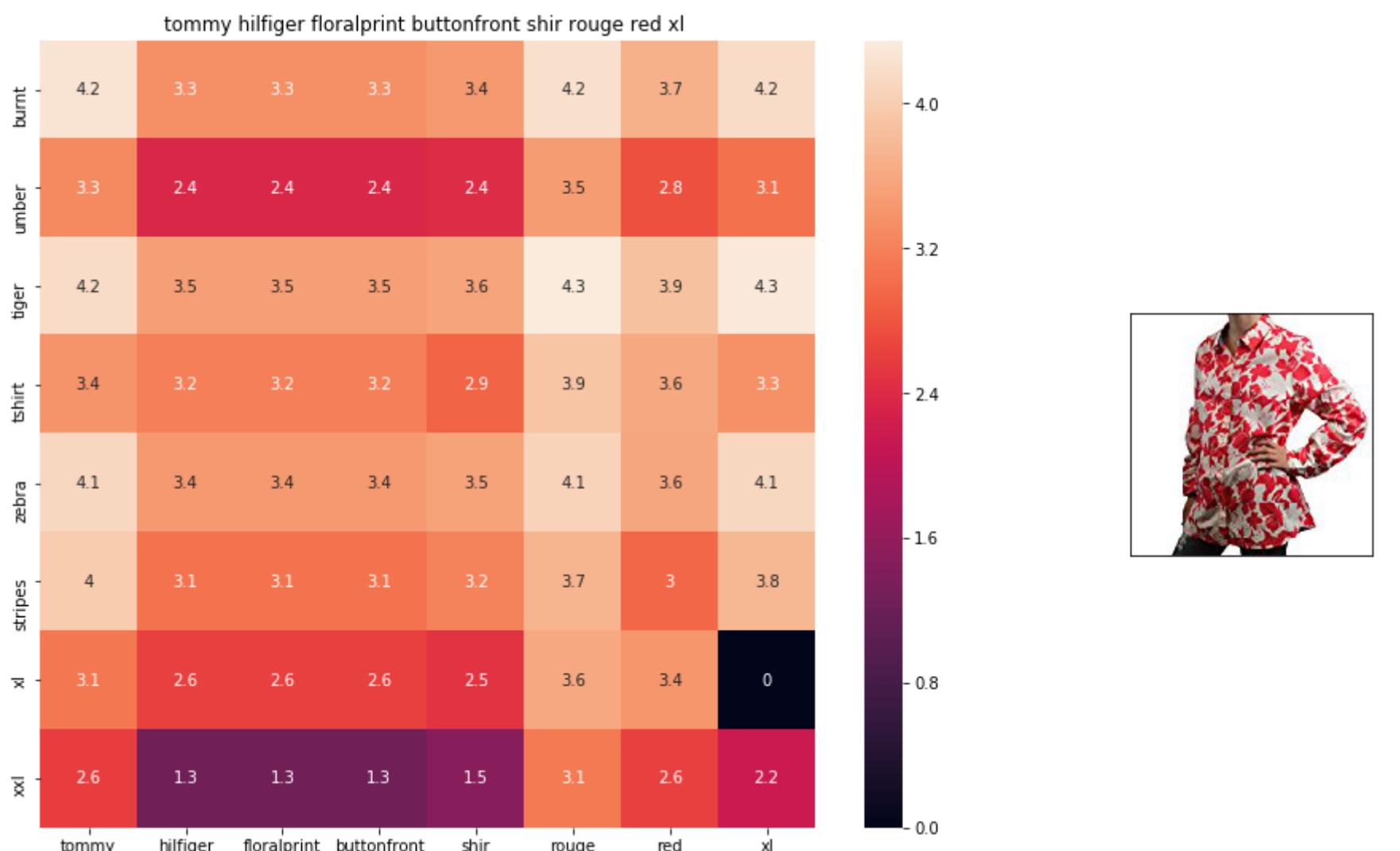
euclidean distance from given input image : 1.09006



ASIN : B01I53HU6K

BRAND : ouxiuli

euclidean distance from given input image : 1.09201



ASIN : B0711NGTQM

BRAND : THILFIGER RTW

euclidean distance from given input image : 1.09234



ASIN : B01EFSLO8Y

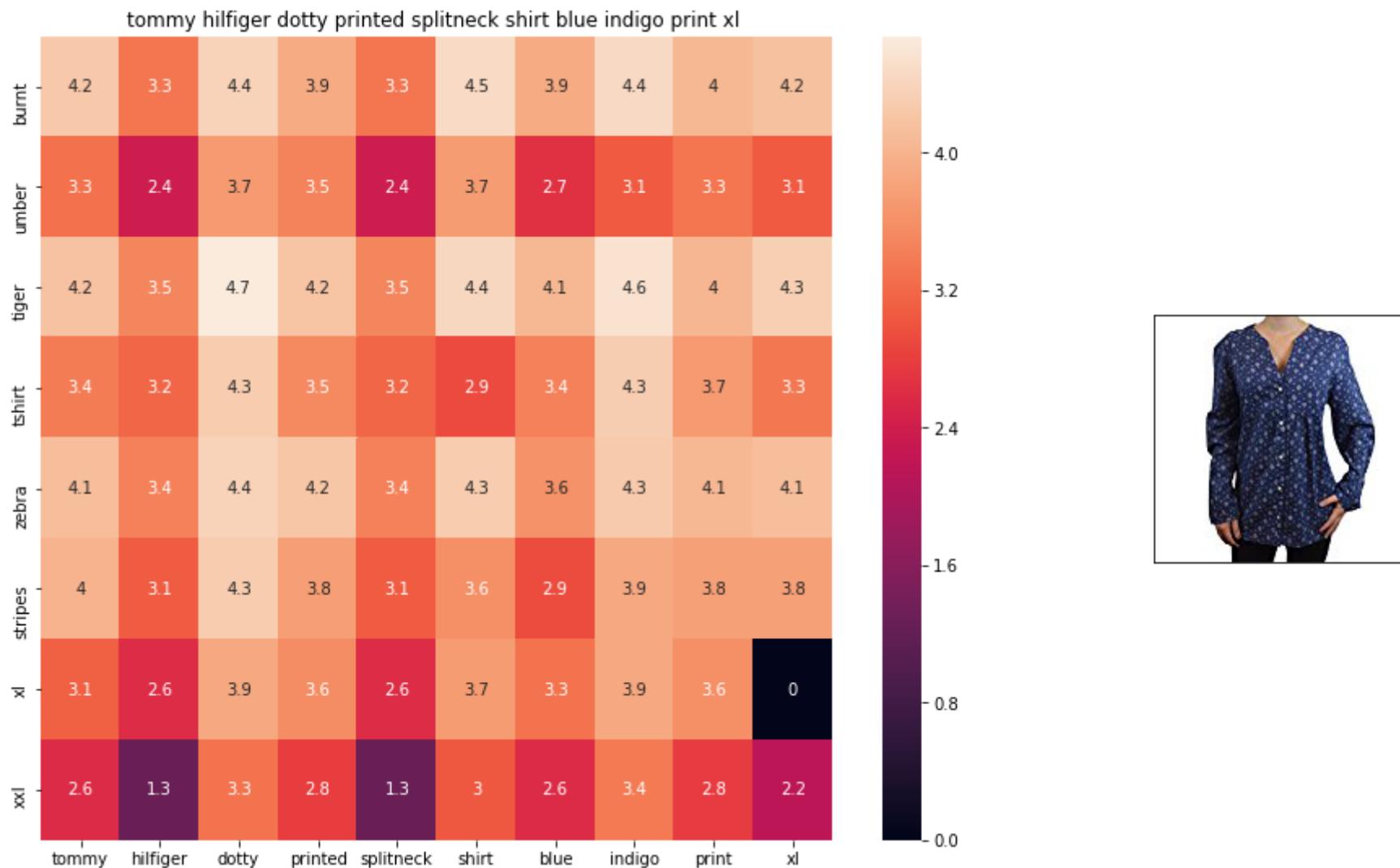
BRAND : Vansty

euclidean distance from given input image : 1.0934

---



---



ASIN : B0716TVWQ4

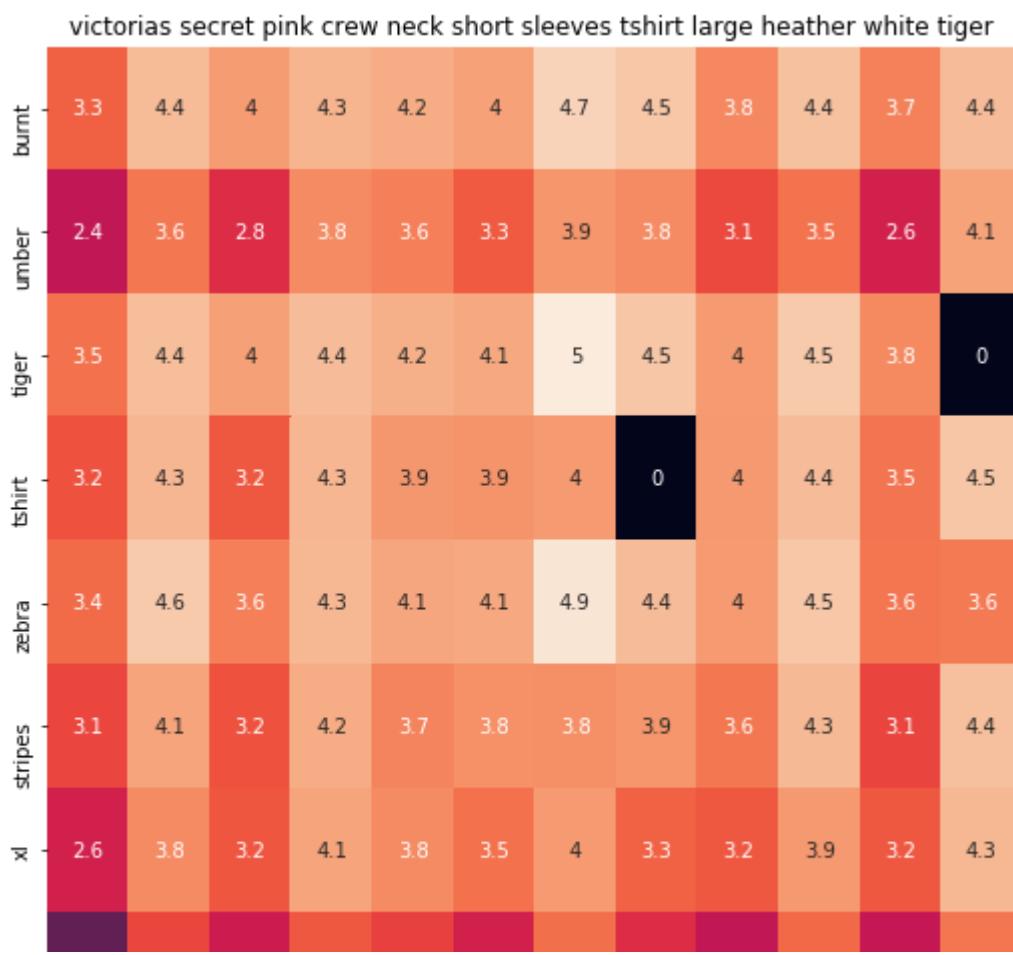
BRAND : THILFIGER RTW

euclidean distance from given input image : 1.0942

---



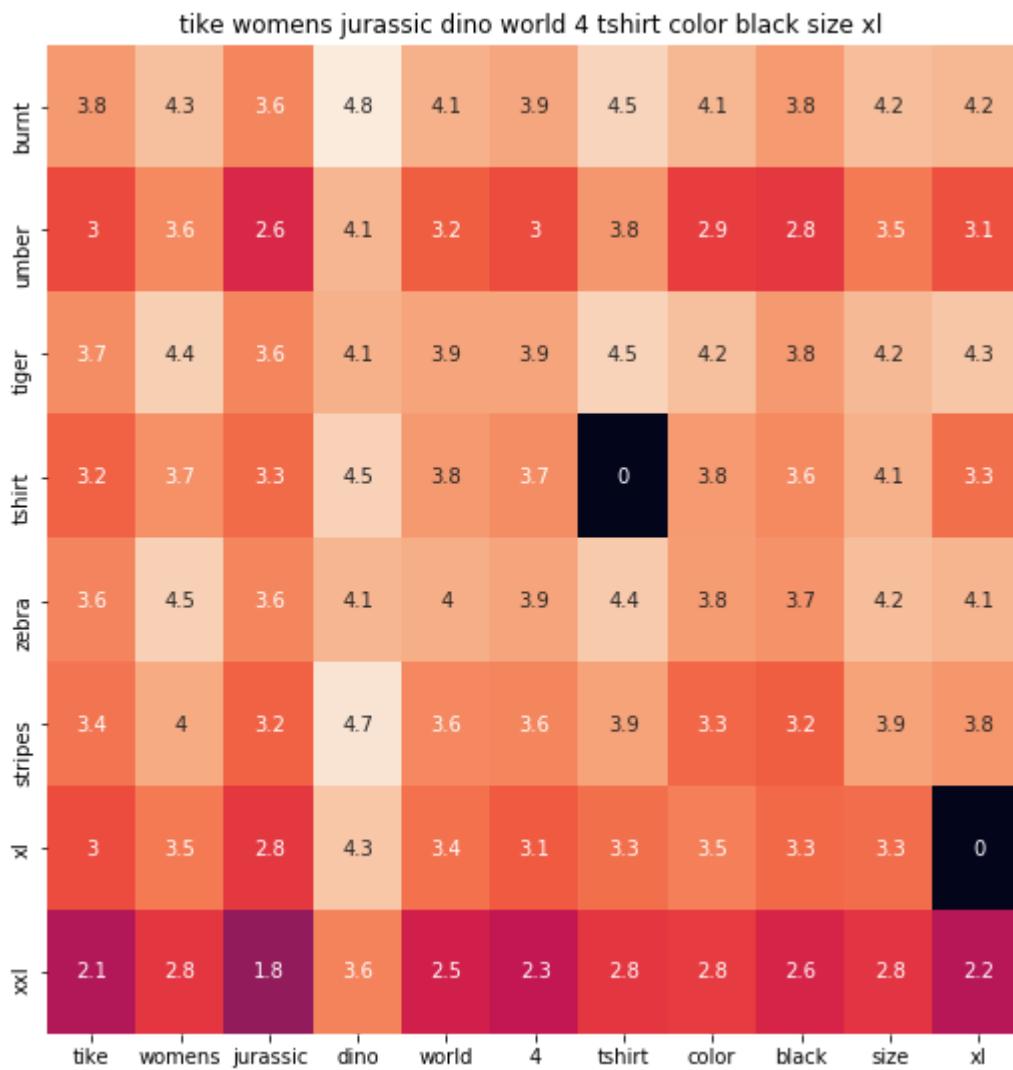
---



ASIN : B0716MVPGV

BRAND : V.Secret

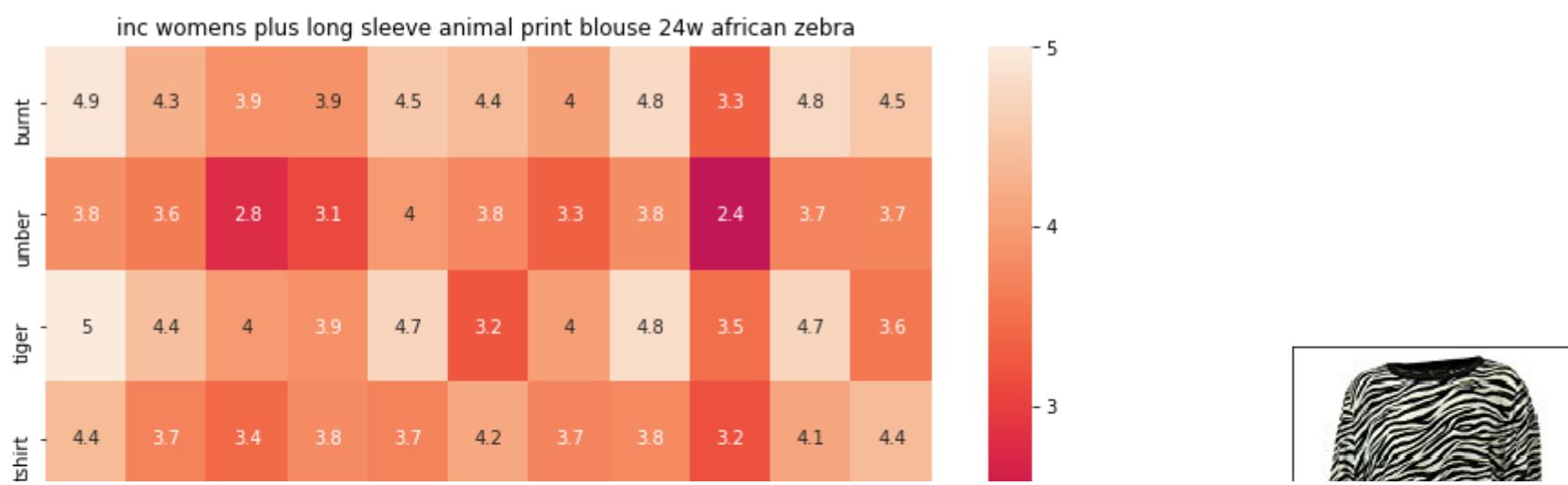
euclidean distance from given input image : 1.09483



ASIN : B0160PN4OI

BRAND : TIKE Fashions

euclidean distance from given input image : 1.09513



ASIN : B018WDJCUA

BRAND : INC - International Concepts Woman

euclidean distance from given input image : 1.09669



## [9.4] IDF weighted Word2Vec for product similarity

```
In [37]: 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 [15]: def weighted_w2v_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

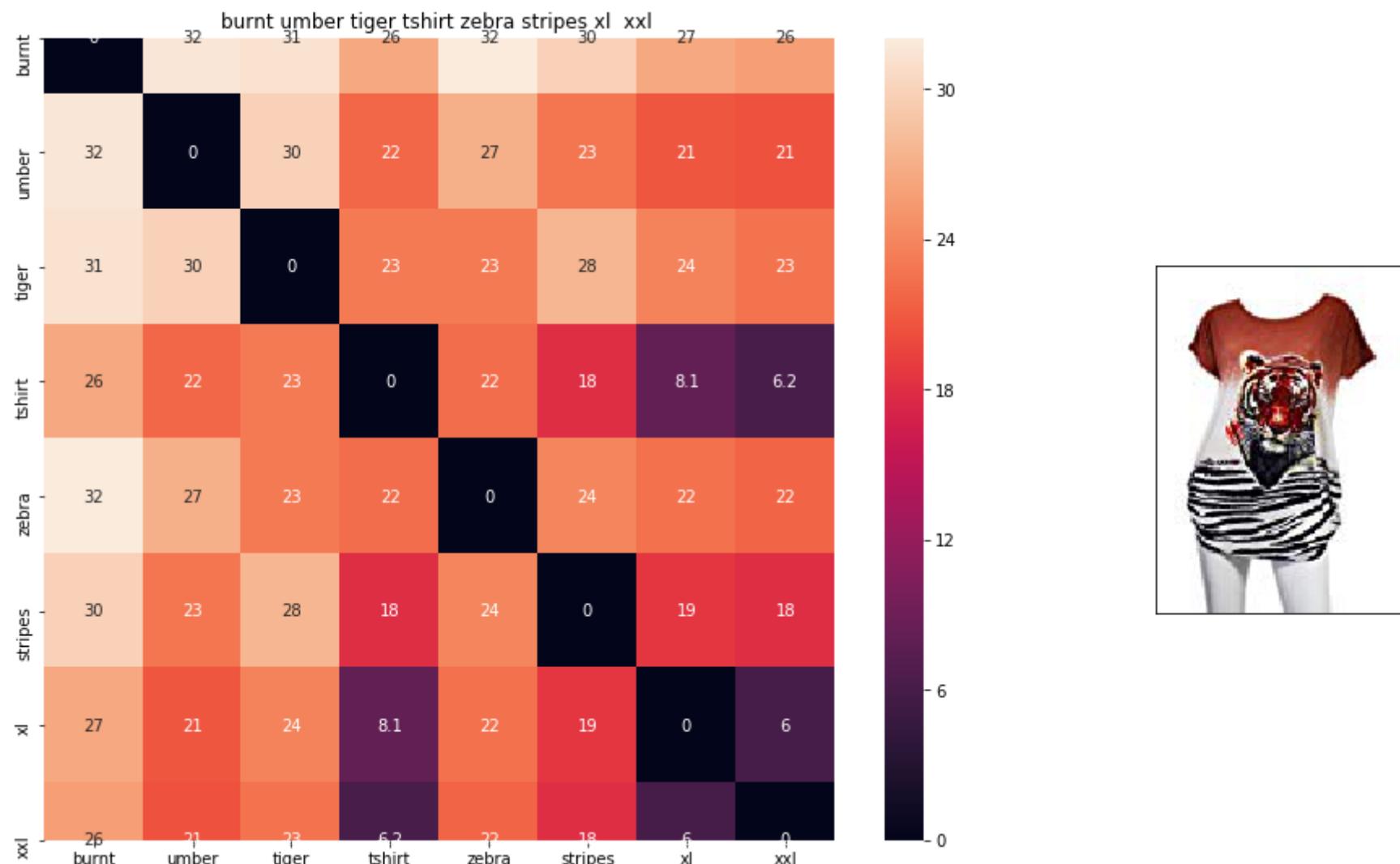
    # pairwise_dist will store the distance from given input apparel to all remaining apparels
    # the metric we used here is cosine, the coside distance is mesured as  $K(X, Y) = \langle X, Y \rangle / (\|X\| * \|Y\|)$ 
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist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

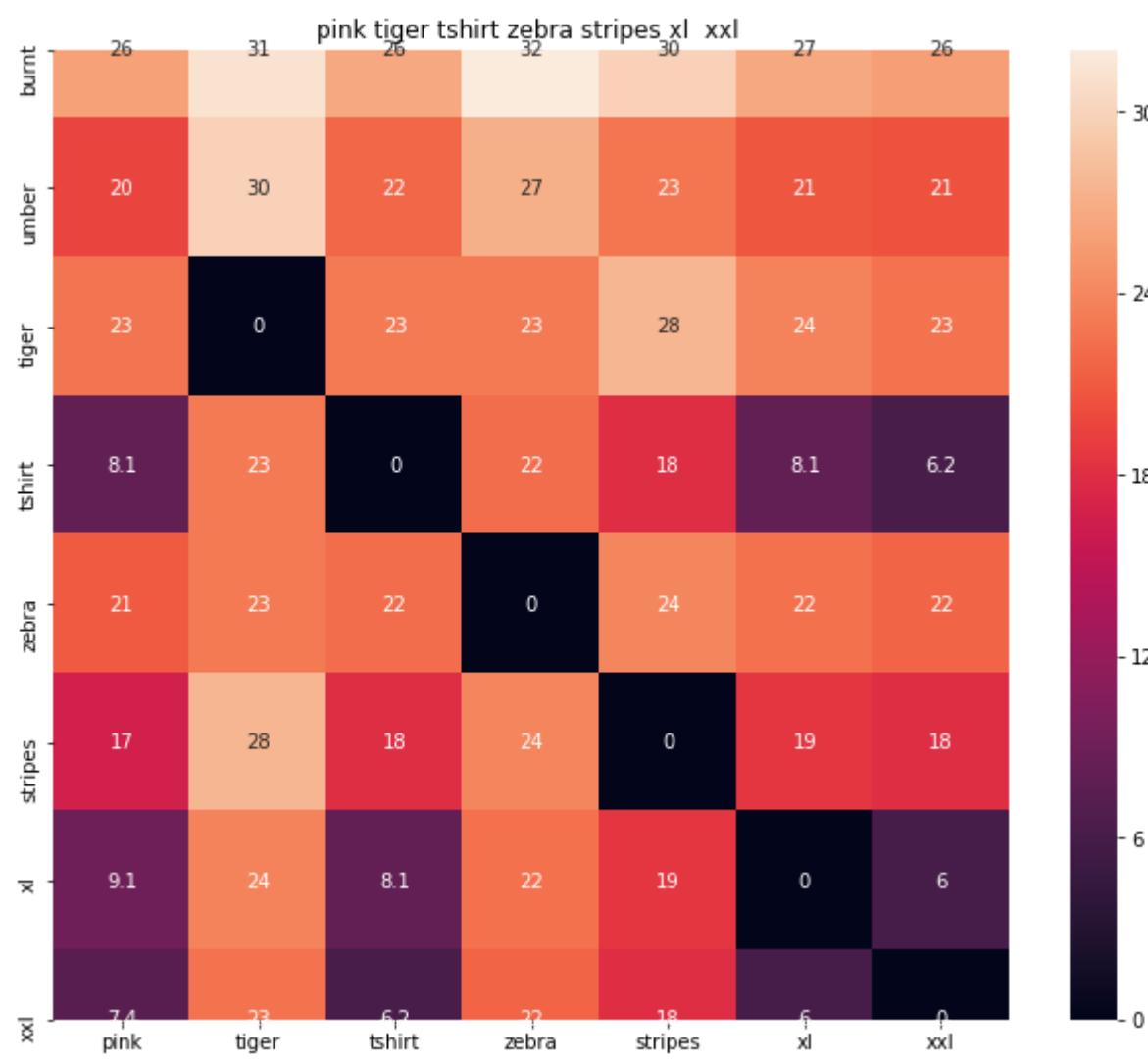
    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])

    for i in range(0, len(indices)):
        heat_map_w2v(data['title'].loc[df_indices[0]], data['title'].loc[df_indices[i]], data['medium_image_url'])
        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(12566, 20)
#931
#12566
# in the give heat map, each cell contains the euclidean distance between words i, j
```



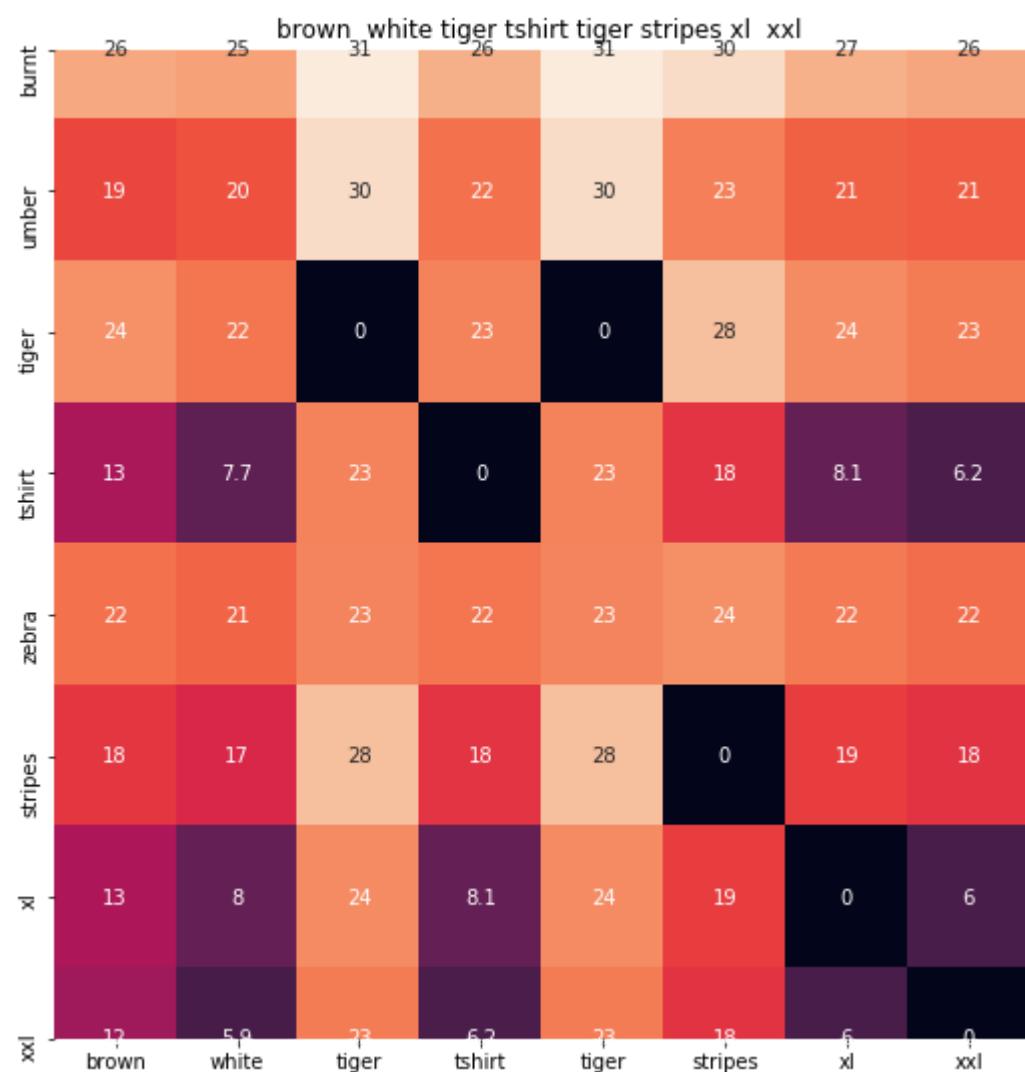
ASIN : B00JXQB5FQ  
 Brand : Si Row  
 euclidean distance from input : 0.0  
 =====  
 =====



ASIN : B00JXQASS6

Brand : Si Row

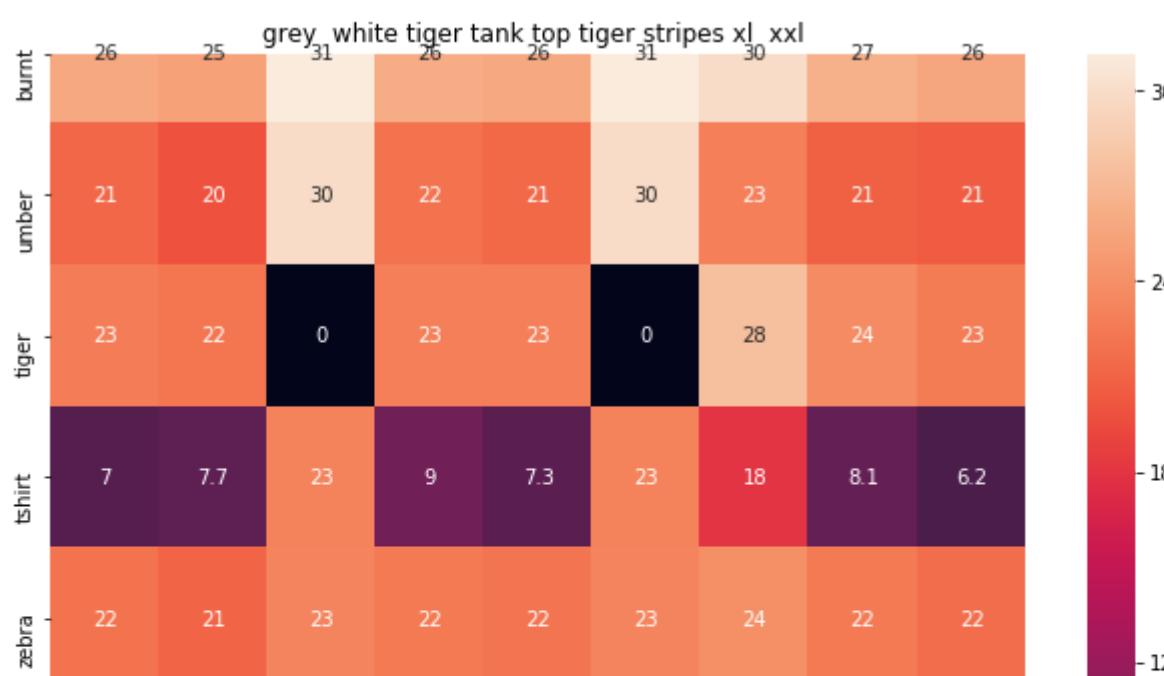
euclidean distance from input : 4.0638866



ASIN : B00JXQCWTO

Brand : Si Row

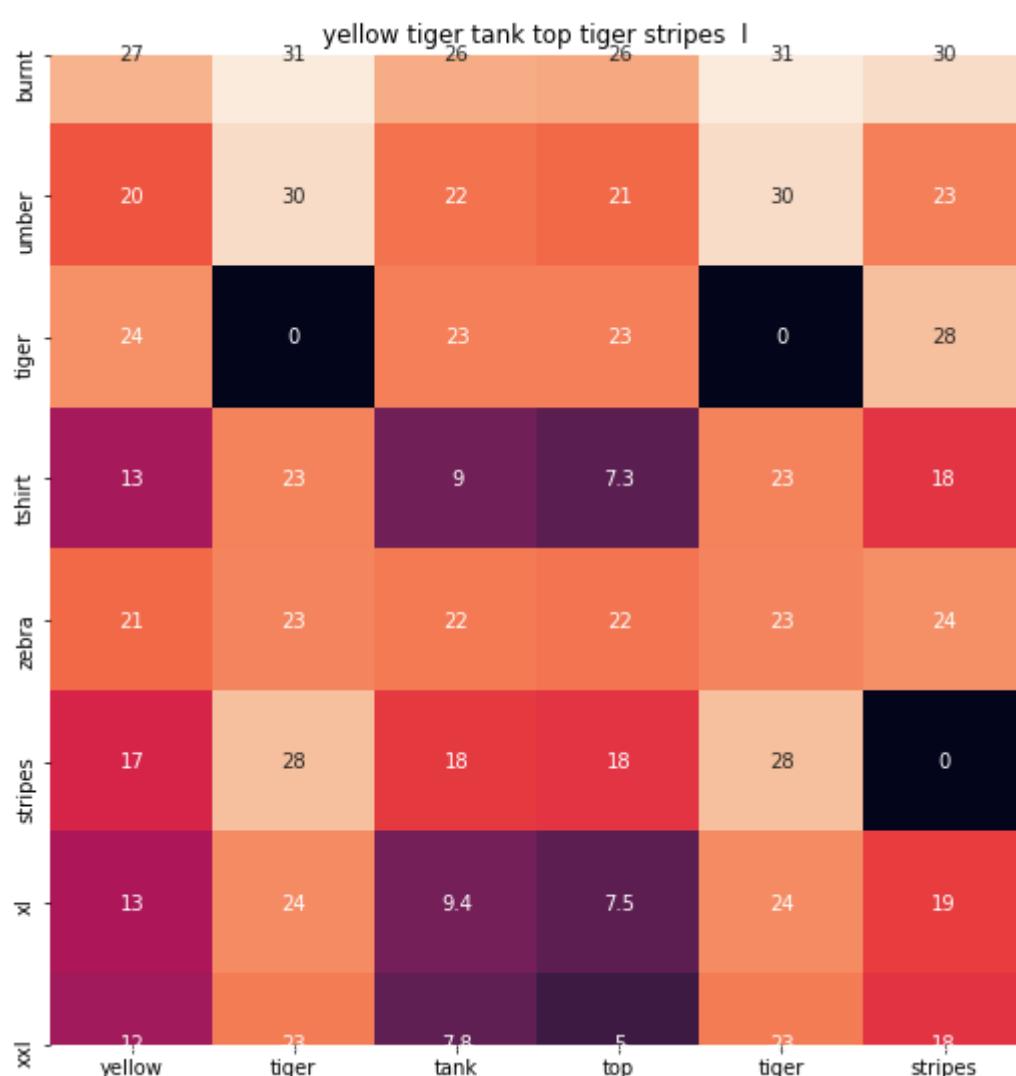
euclidean distance from input : 4.7709413



ASIN : B00JXQAFZ2

Brand : Si Row

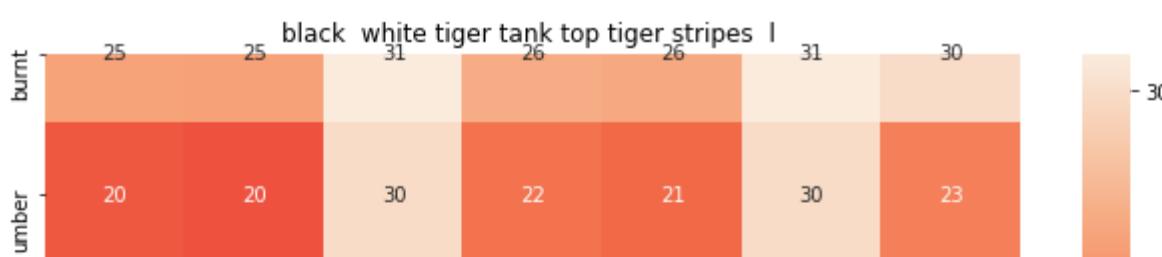
euclidean distance from input : 5.3601604



ASIN : B00JXQAUWA

Brand : Si Row

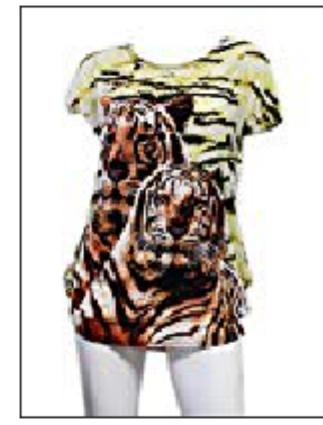
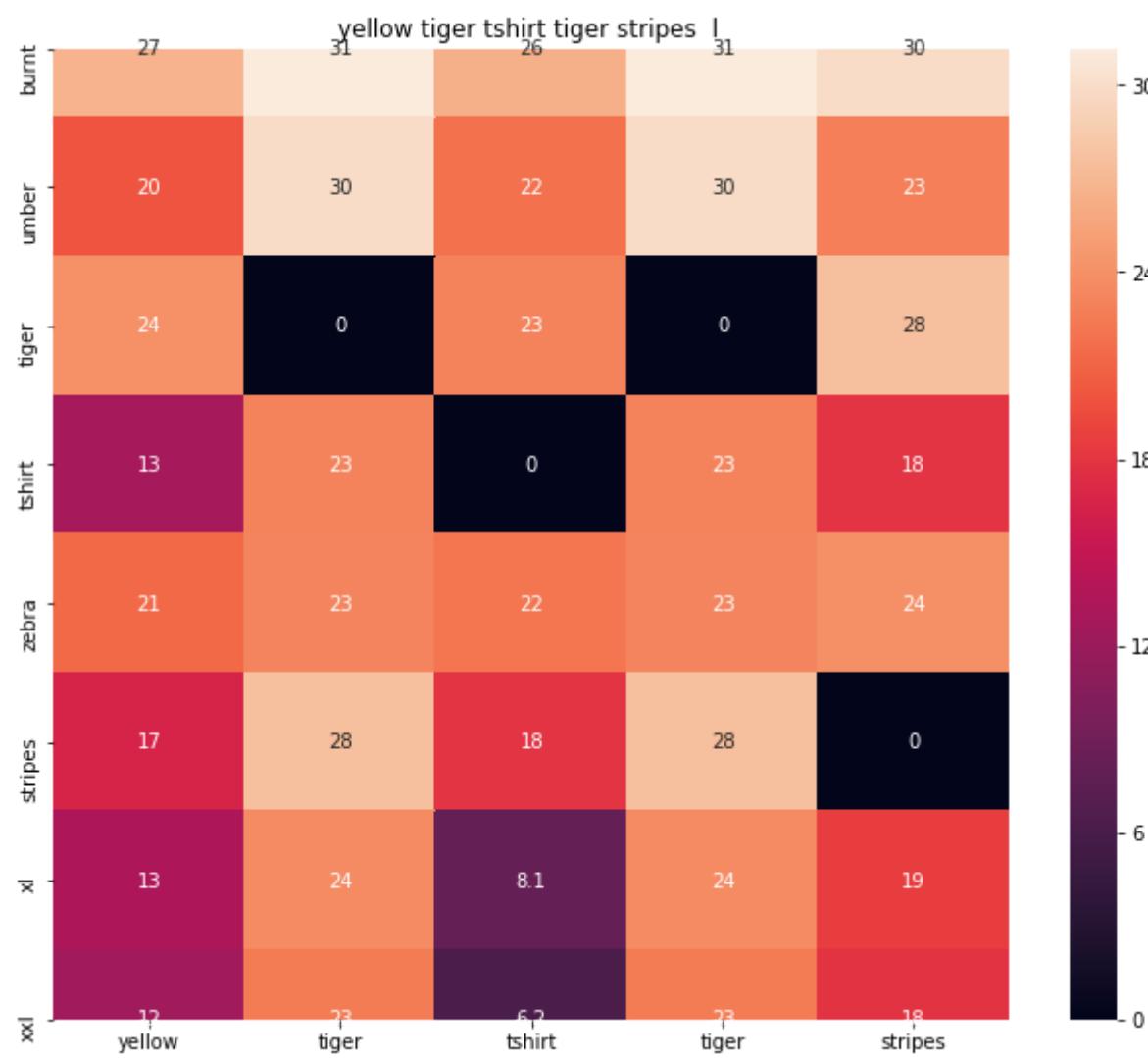
euclidean distance from input : 5.6895227



ASIN : B00JXQAO94

Brand : Si Row

euclidean distance from input : 5.693021



ASIN : B00JXQCUIC

Brand : Si Row

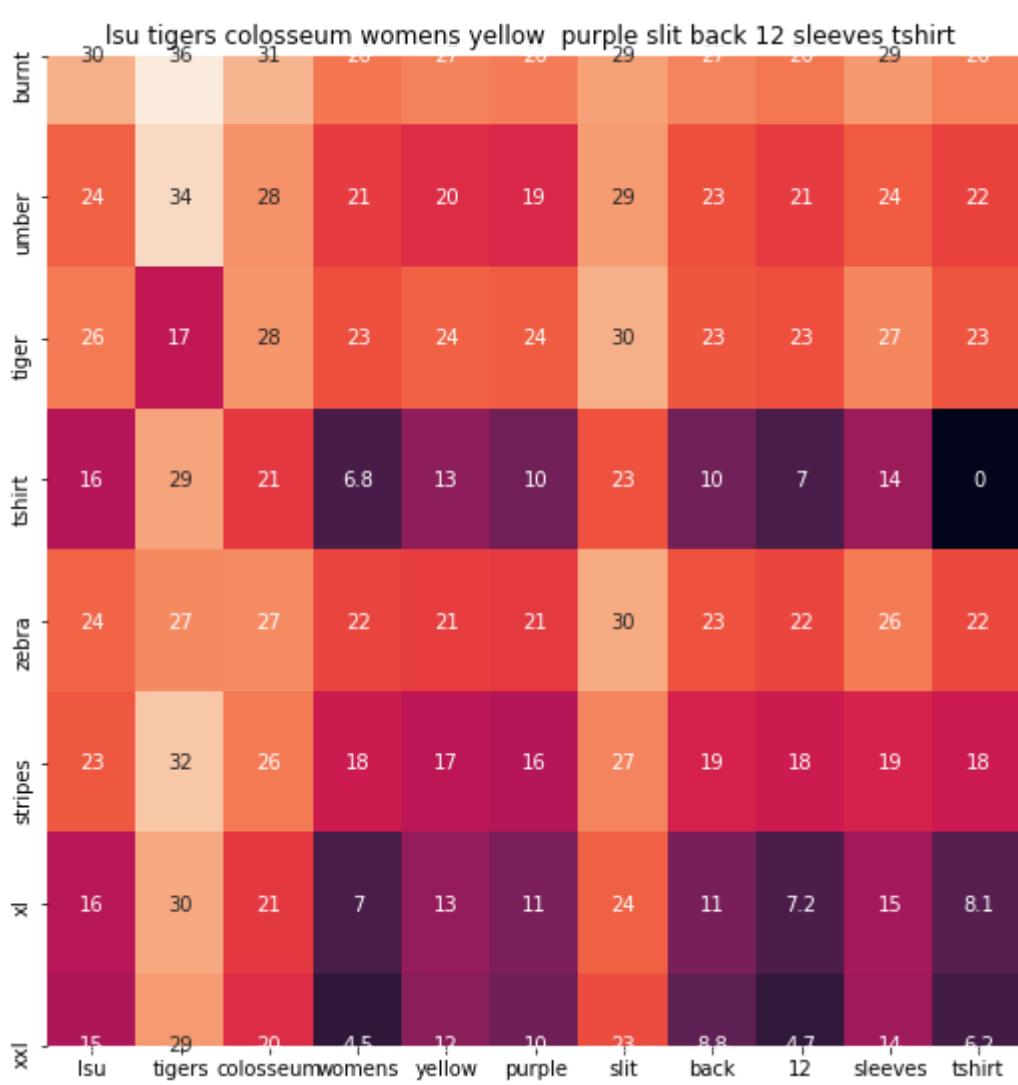
euclidean distance from input : 5.893442



ASIN : B015H41F6G

Brand : KINGDE

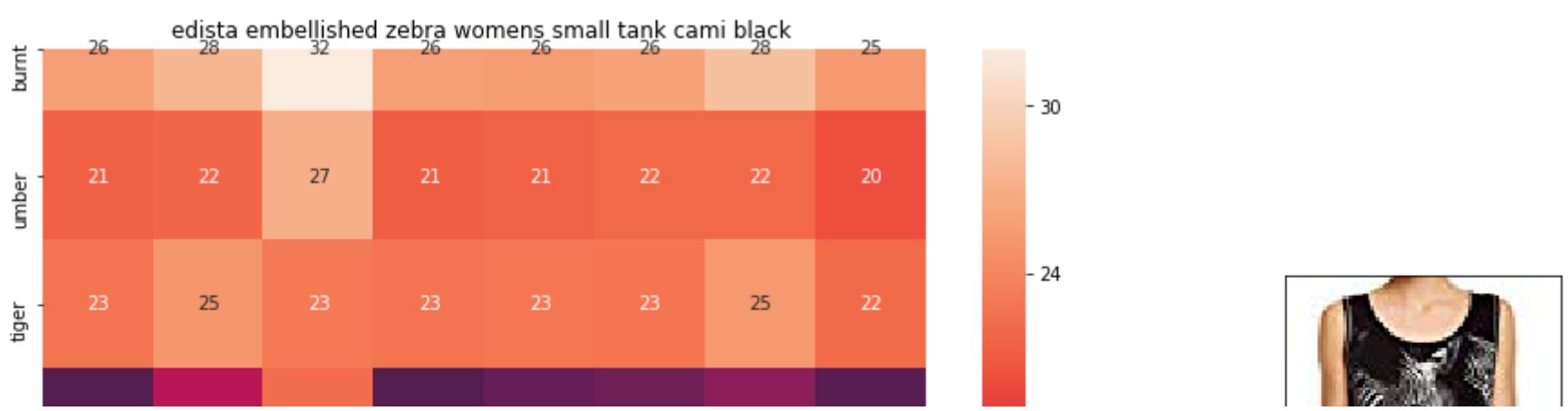
euclidean distance from input : 6.1329894



ASIN : B073R5Q8HD

Brand : Colosseum

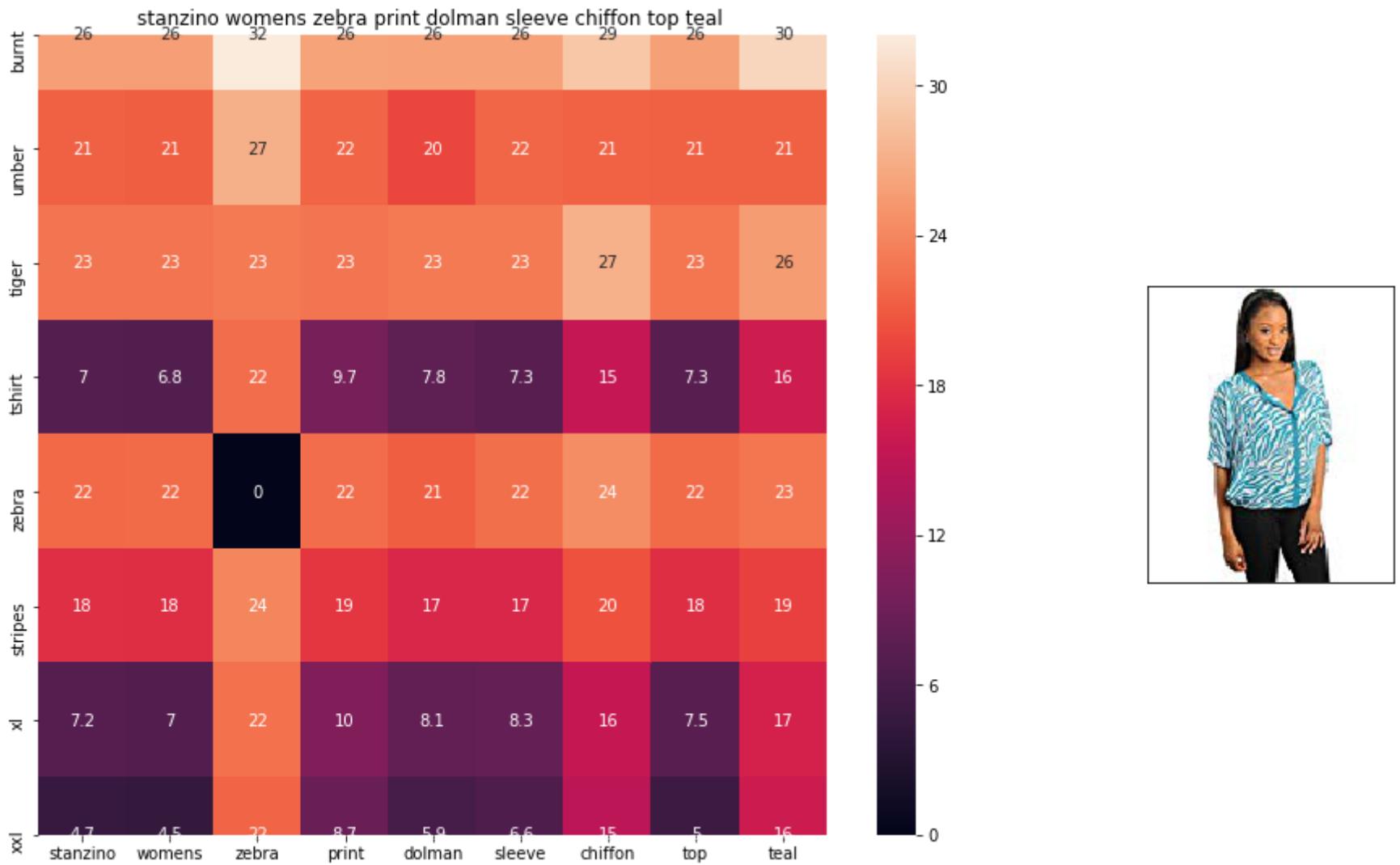
euclidean distance from input : 6.2567053



ASIN : B074P8MD22

Brand : Edista

euclidean distance from input : 6.3922033



ASIN : B00C0I3U3E

Brand : Stanzino

euclidean distance from input : 6.4149003

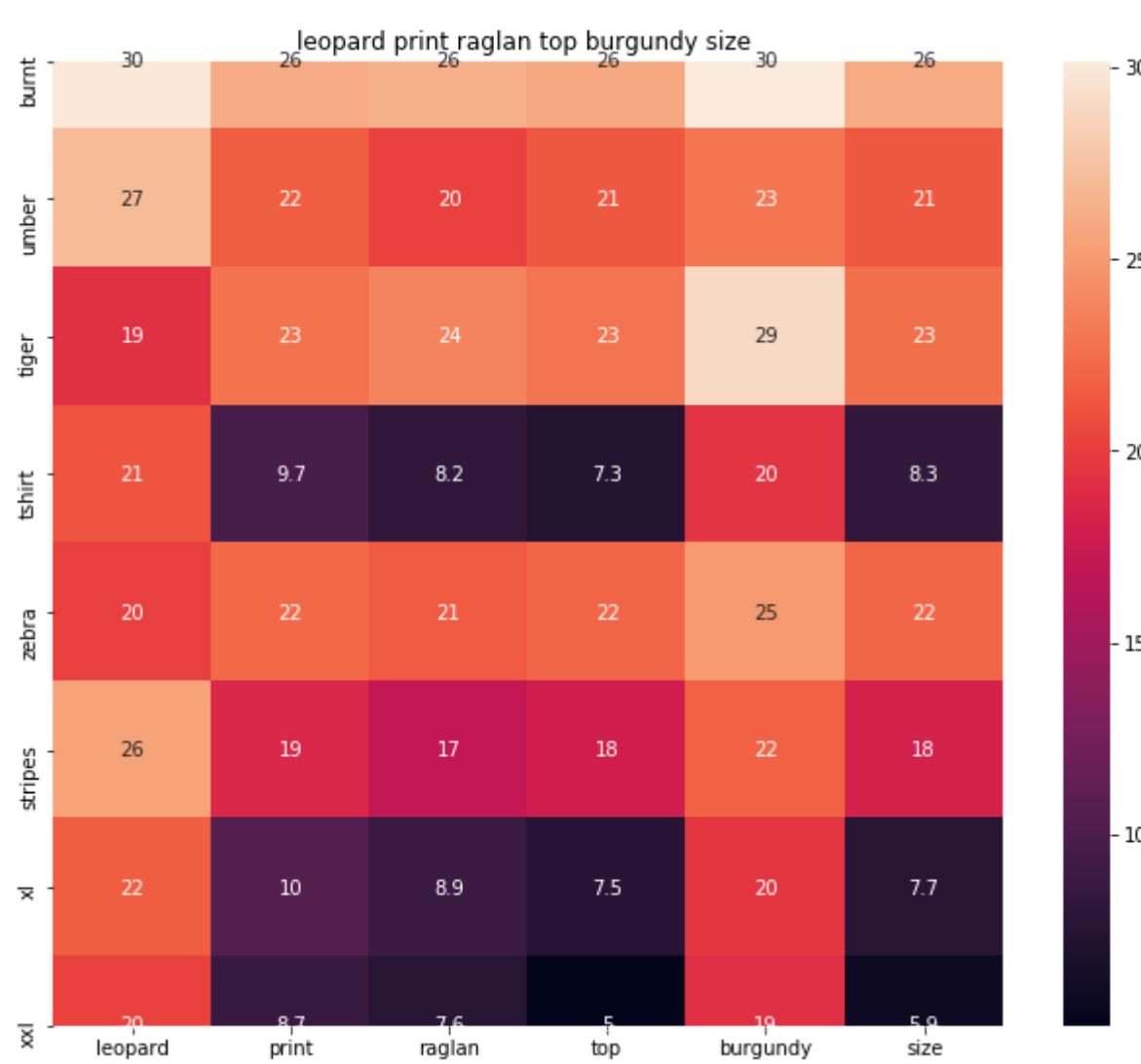
lsu tigers colosseum womens charcoal gray striped back sleeveless tank top

ASIN : B073R4ZM7Y

Brand : Colosseum

euclidean distance from input : 6.450959

=====

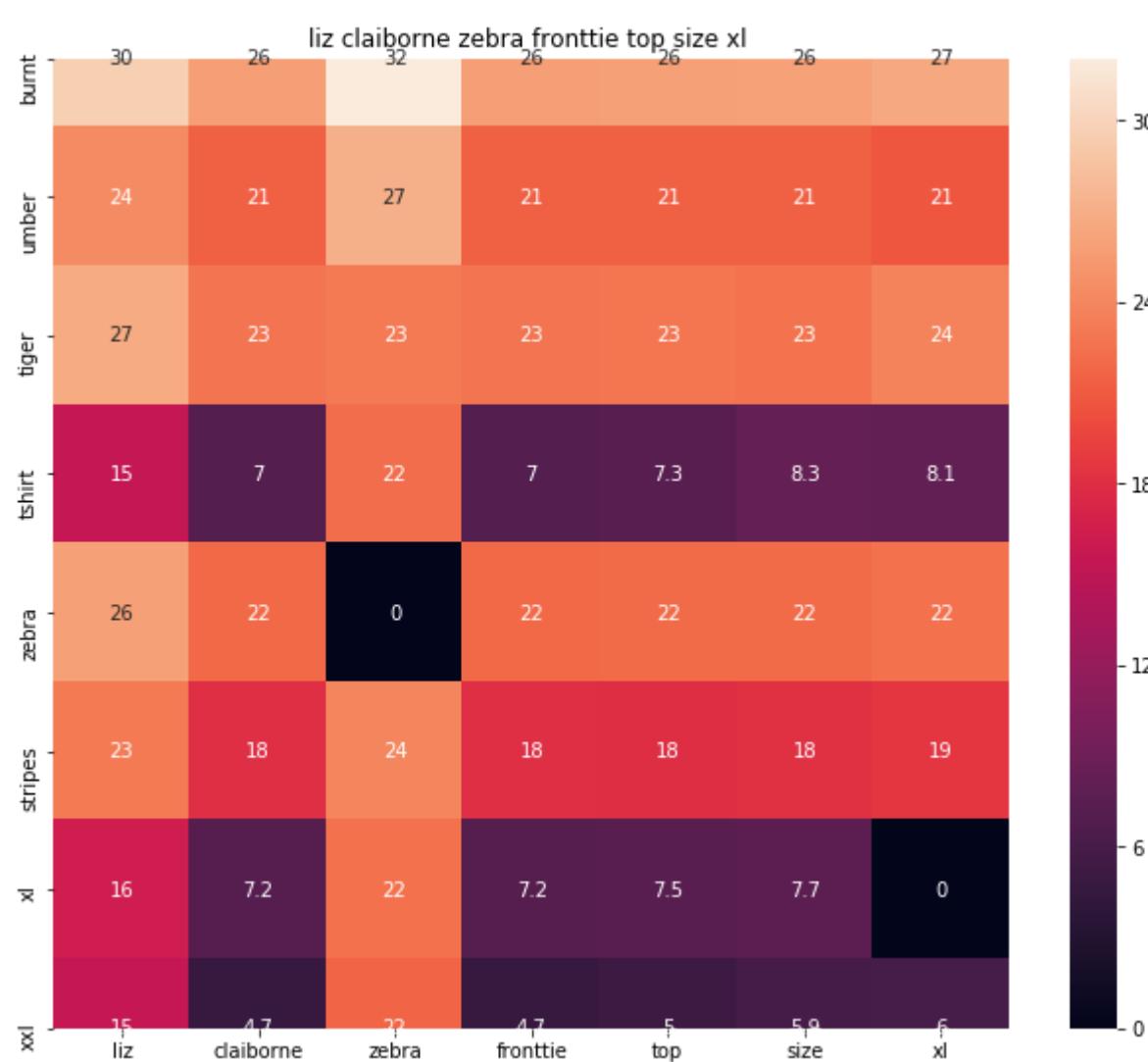


ASIN : B01C60RLDQ

Brand : 1 Mad Fit

euclidean distance from input : 6.463409

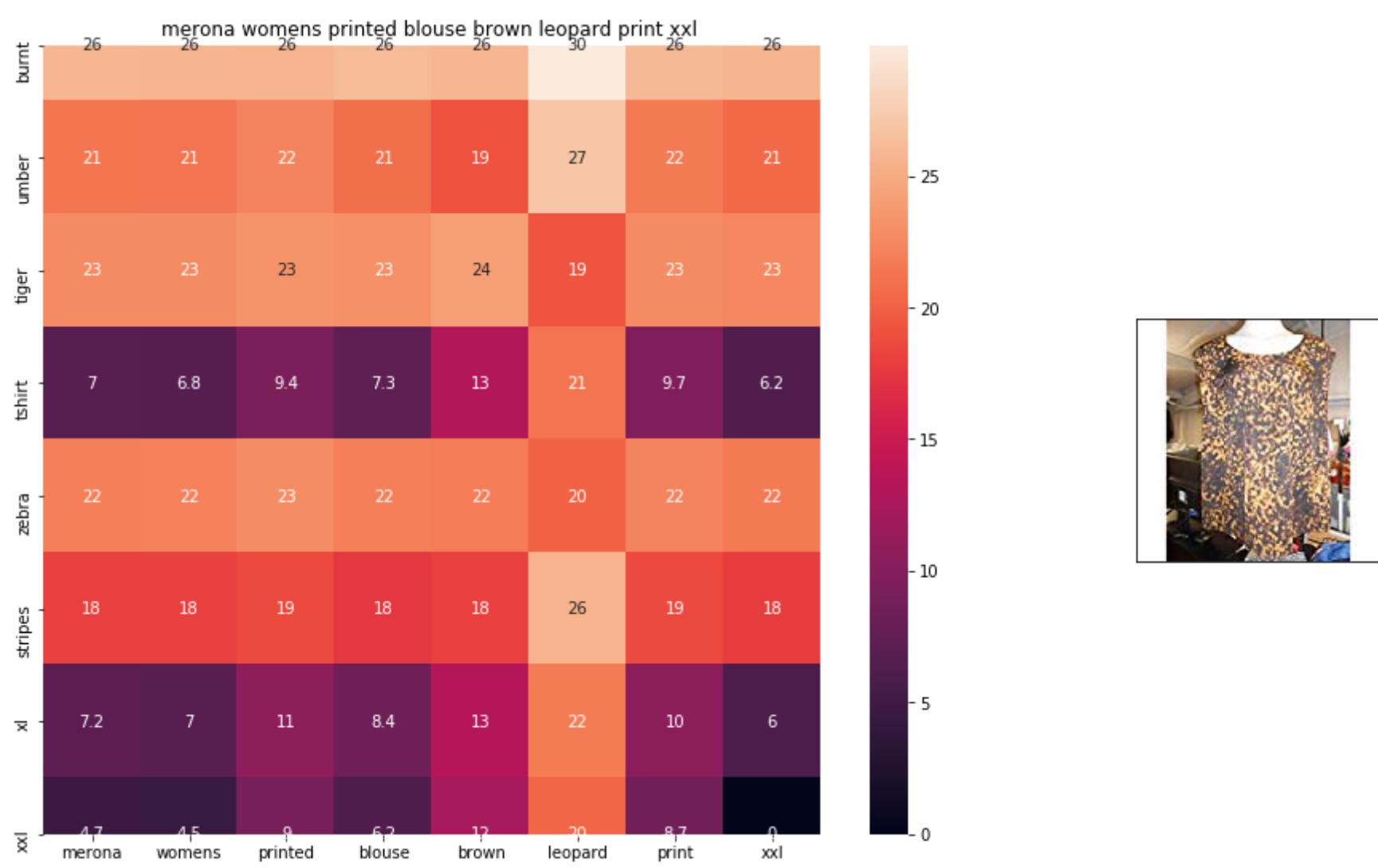
=====



ASIN : B06XBY5QXL

Brand : Liz Claiborne

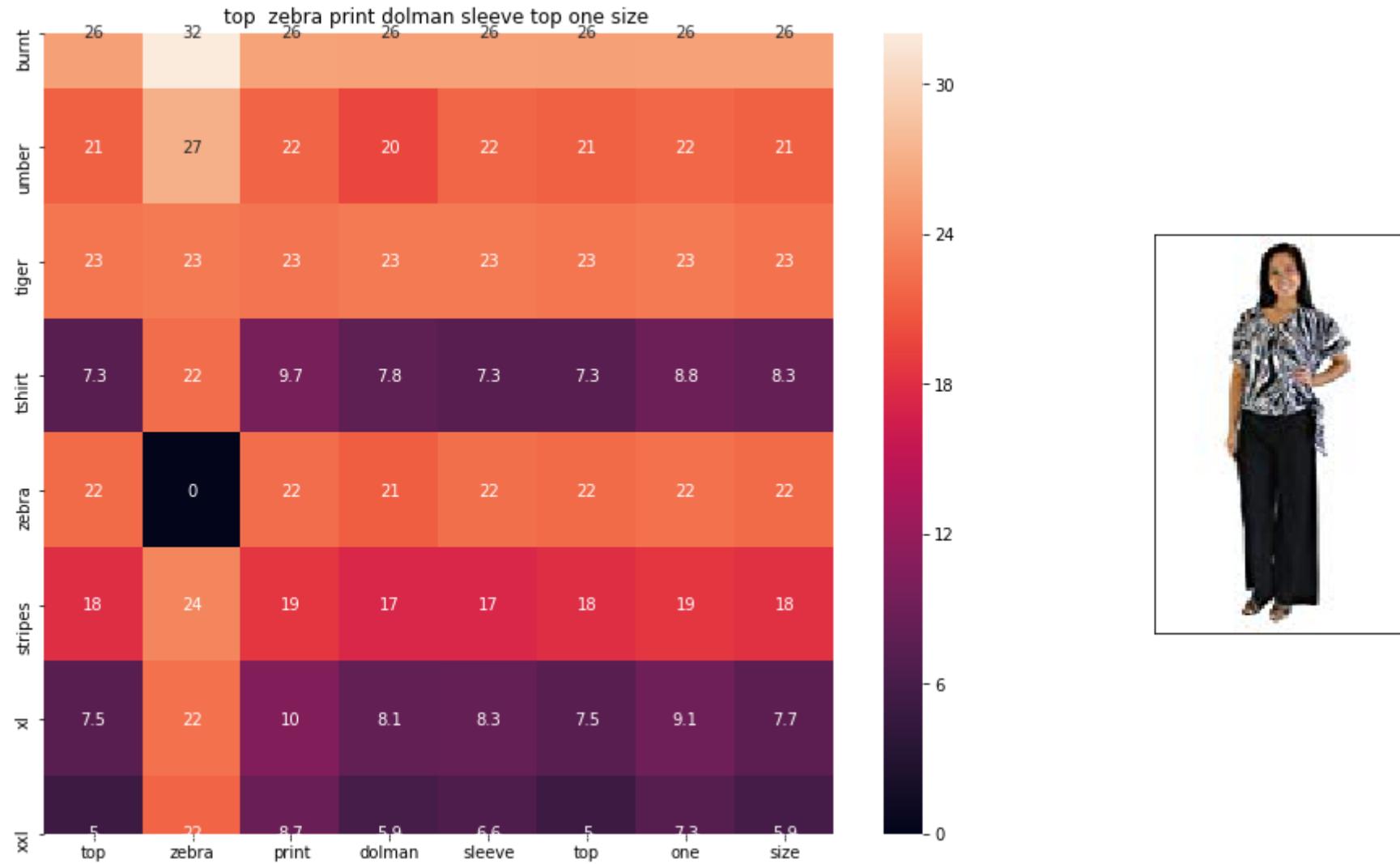
euclidean distance from input : 6.5392227



ASIN : B071YF3WDD

Brand : Merona

euclidean distance from input : 6.5755024



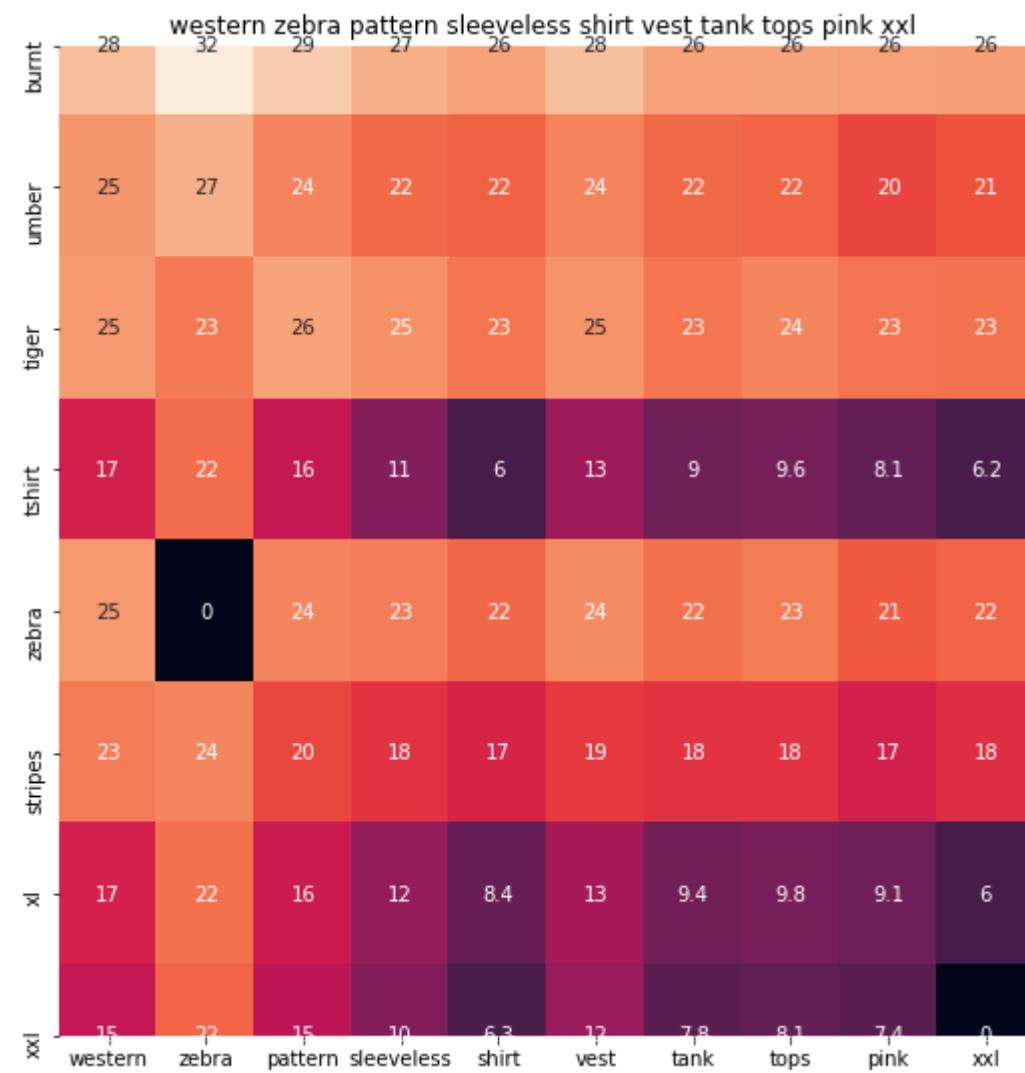
ASIN : B00H8A6ZLI

Brand : Vivian's Fashions

euclidean distance from input : 6.6382146

=====

=====



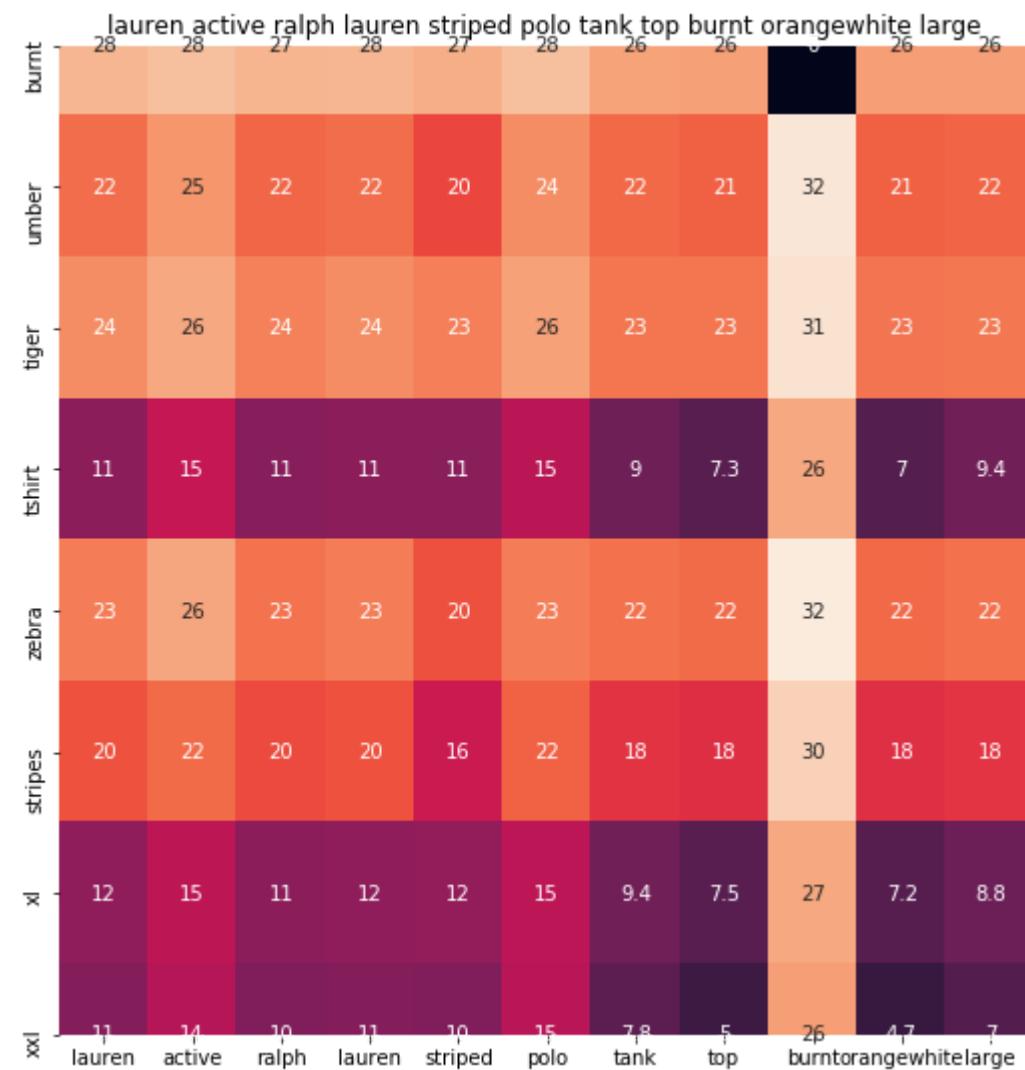
ASIN : B00Z6HEXWI

Brand : Black Temptation

euclidean distance from input : 6.6607366

=====

=====



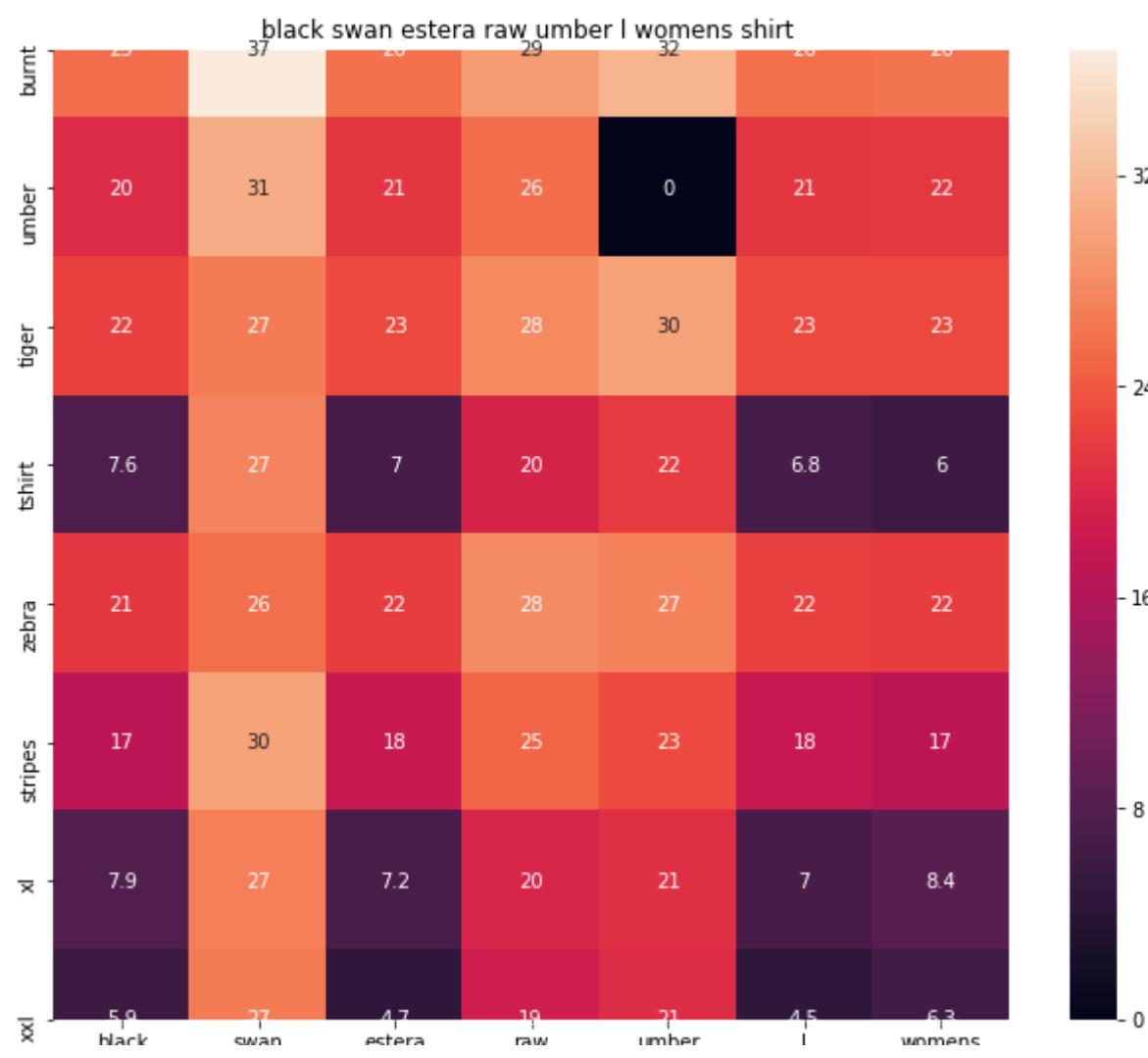
ASIN : B00ILGH5OY

Brand : Ralph Lauren Active

euclidean distance from input : 6.6839046

=====

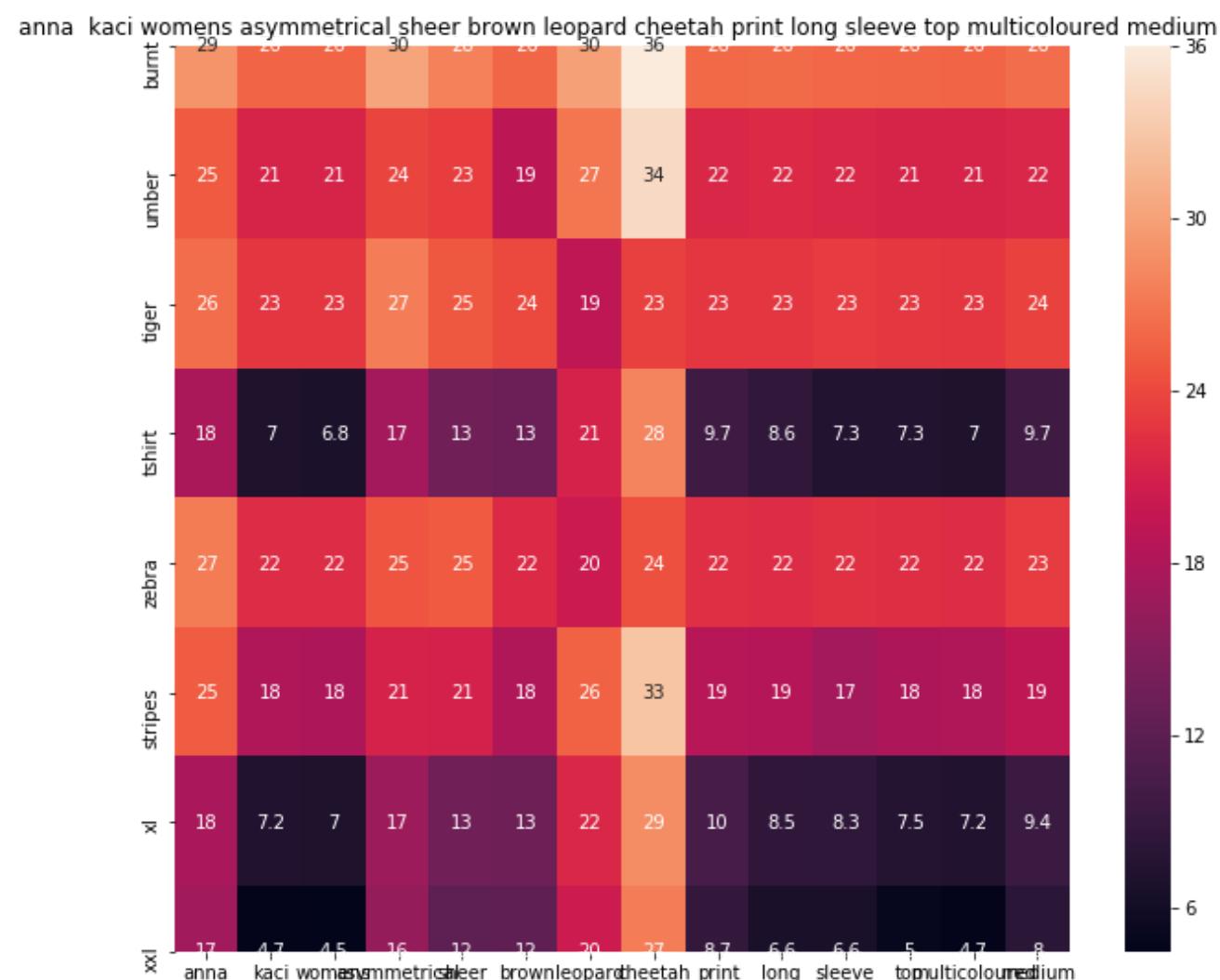
=====



ASIN : B06Y1VN8WQ

Brand : Black Swan

euclidean distance from input : 6.705763



ASIN : B00KSNTY7Y

Brand : Anna-Kaci

euclidean distance from input : 6.7061243

## [9.6] Weighted similarity using brand and color.

```
In [39]: # 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 hyphen
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((type_features, color_features)).tocsr()
```

```
In [40]: 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 length 300 corresp
    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 length 300 corresp
    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 f
                  [data[ 'asin' ].loc[df_id2], brands[doc_id2], colors[doc_id2], types[doc_id2]]] # recommended appar

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

    # 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])
    # plotting 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 lines 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 [37]: 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

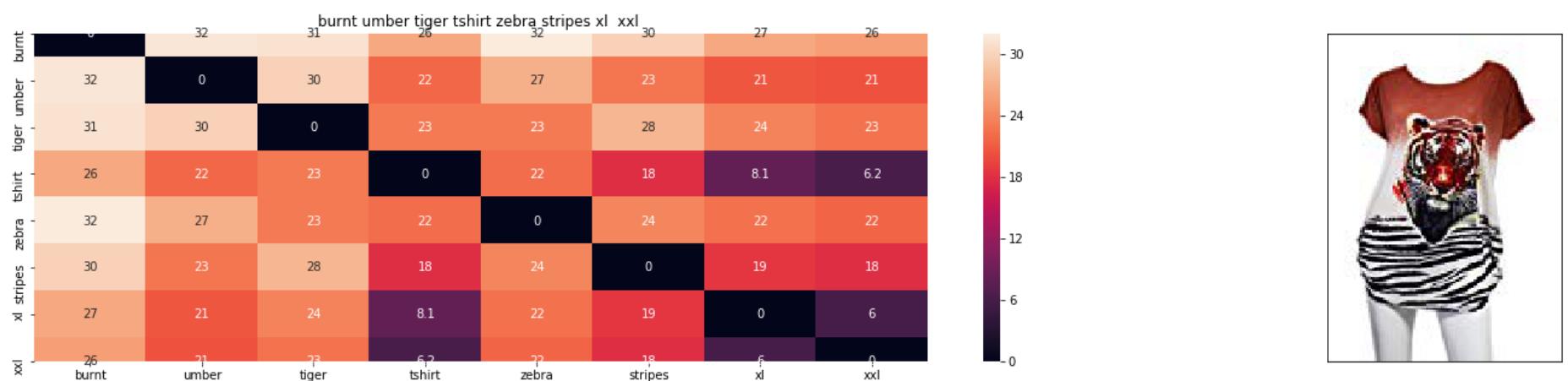
    # pairwise_dist will store the distance from given input apparel to all remaining apparels
    # the metric we used here is cosine, the coside distance is mesured as  $K(X, Y) = \langle X, Y \rangle / (\|X\| * \|Y\|)$ 
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    idf_w2v_dist = 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]

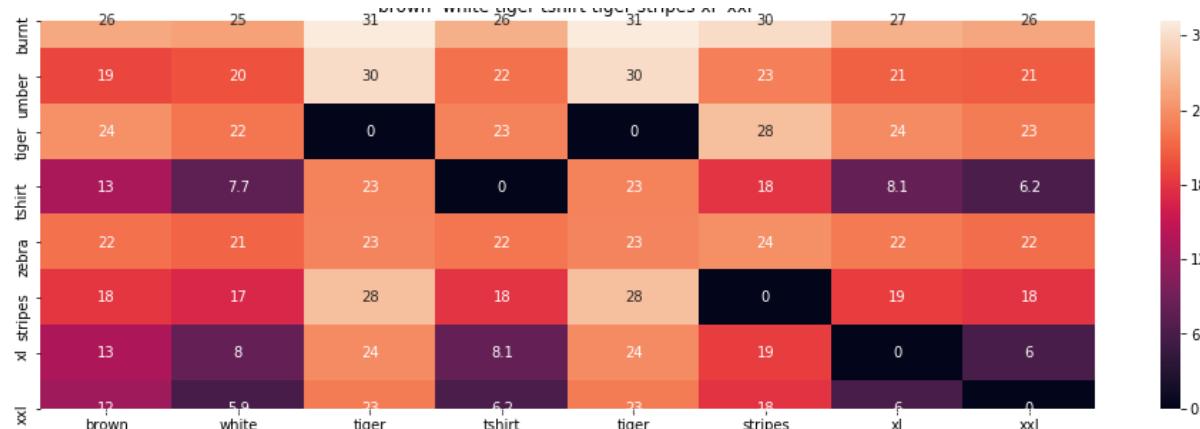
    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])

    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_imag
            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(12566, 5, 5, 20)
# in the give heat map, each cell contains the euclidean distance between words i, j
```



ASIN : B00JXQB5FQ  
 Brand : Si Row  
 euclidean distance from input : 0.0  
 =====  
 =====

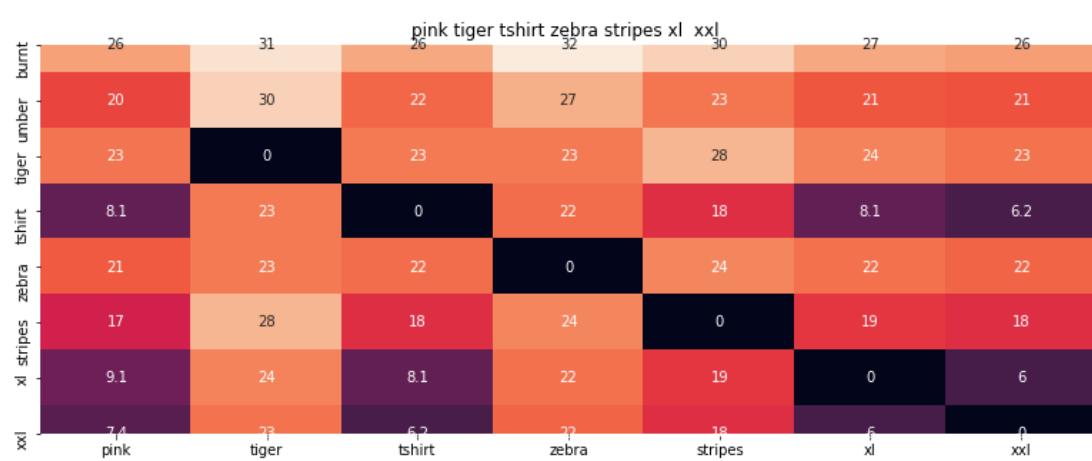


ASIN : B00JXQCWTO

Brand : Si Row

euclidean distance from input : 2.3854705810546877

=====

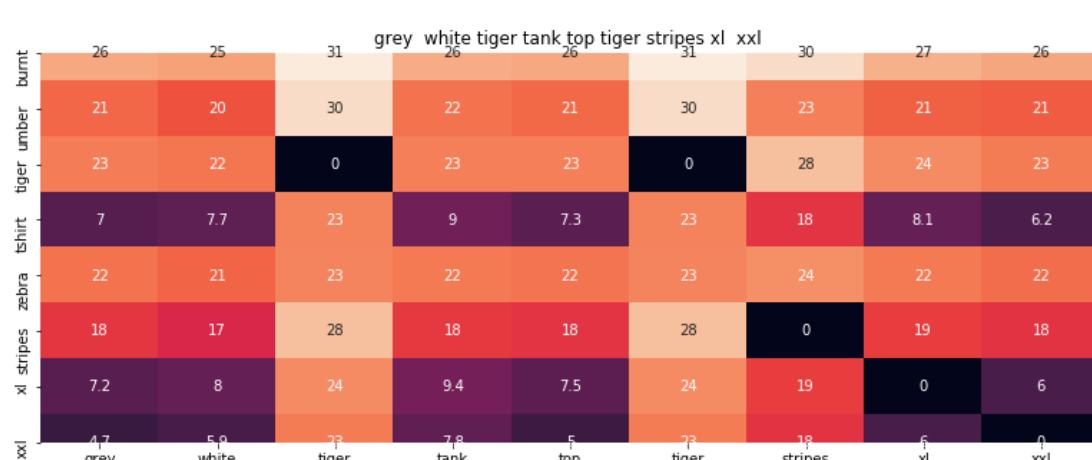


ASIN : B00JXQASS6

Brand : Si Row

euclidean distance from input : 2.739050102414575

=====



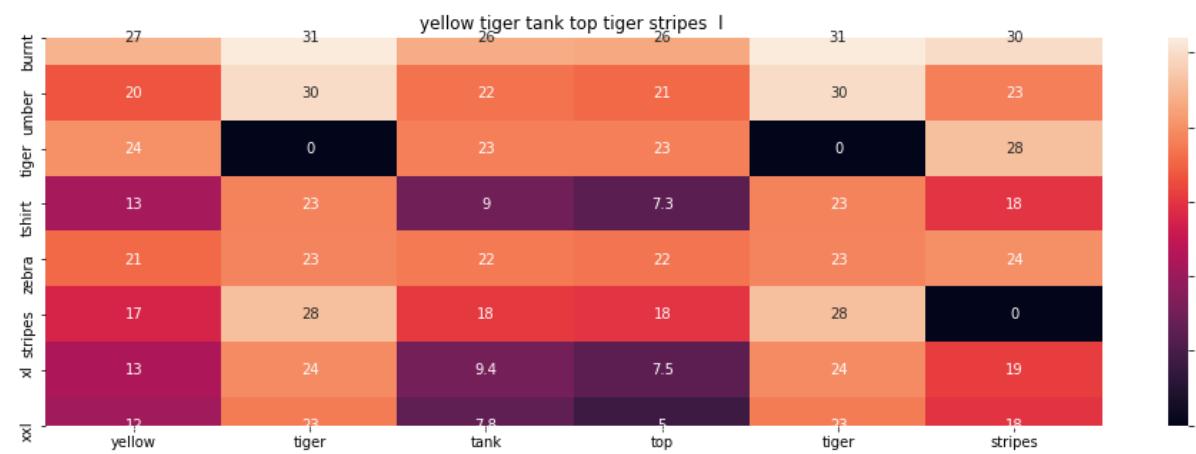
ASIN : B00JXQAFZ2

Brand : Si Row

euclidean distance from input : 3.387187004270044

=====

=====



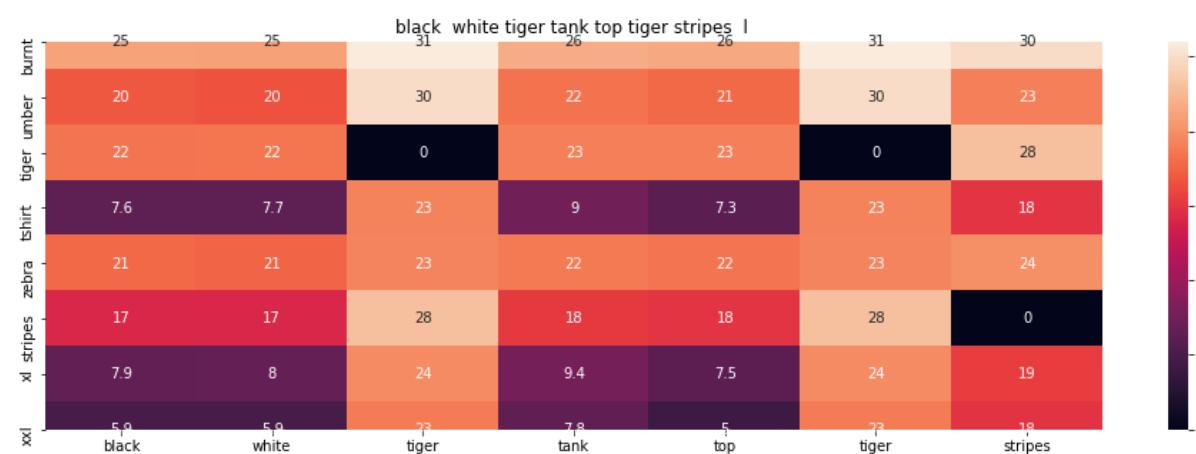
ASIN : B00JXQAUWA

Brand : Si Row

euclidean distance from input : 3.5518680574316646

=====

=====



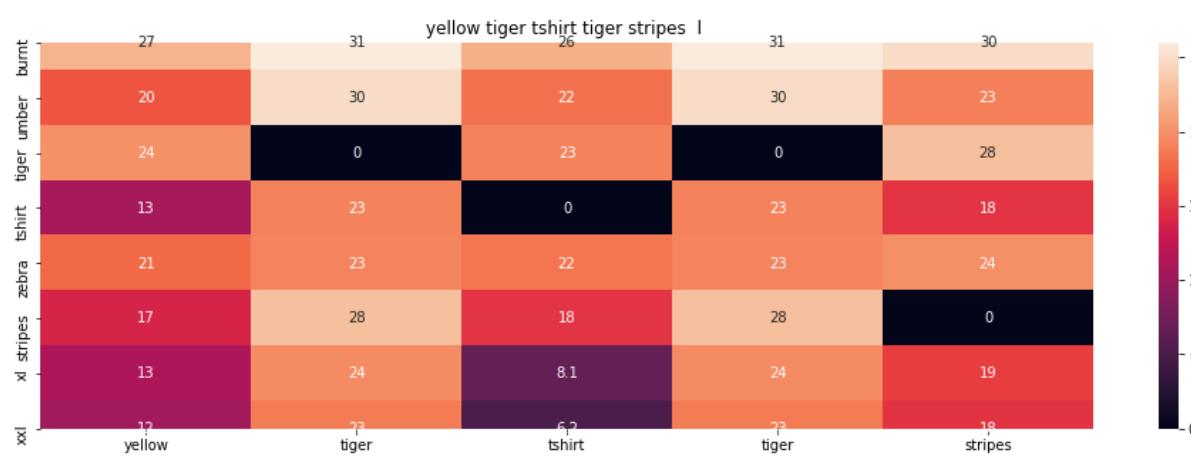
ASIN : B00JXQAO94

Brand : Si Row

euclidean distance from input : 3.5536170961279536

=====

=====



ASIN : B00JXQCUIC

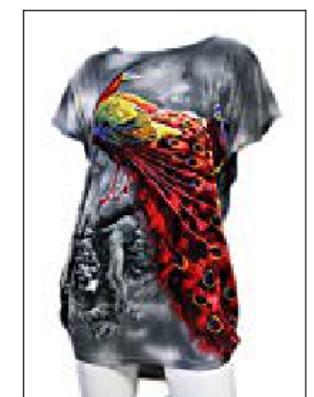
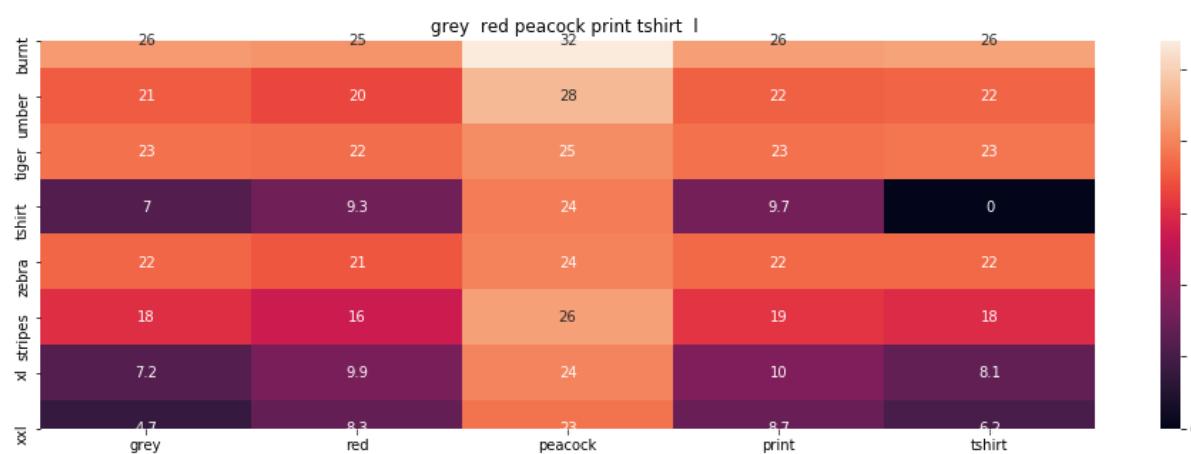
Brand : Si Row

euclidean distance from input : 3.6538278581518795

---



---



ASIN : B00JXQCFRS

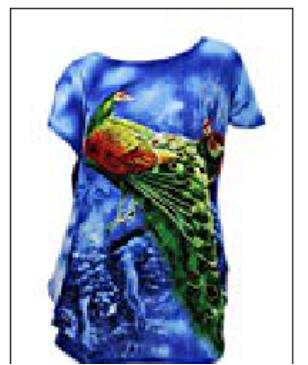
Brand : Si Row

euclidean distance from input : 4.128811264218774

---



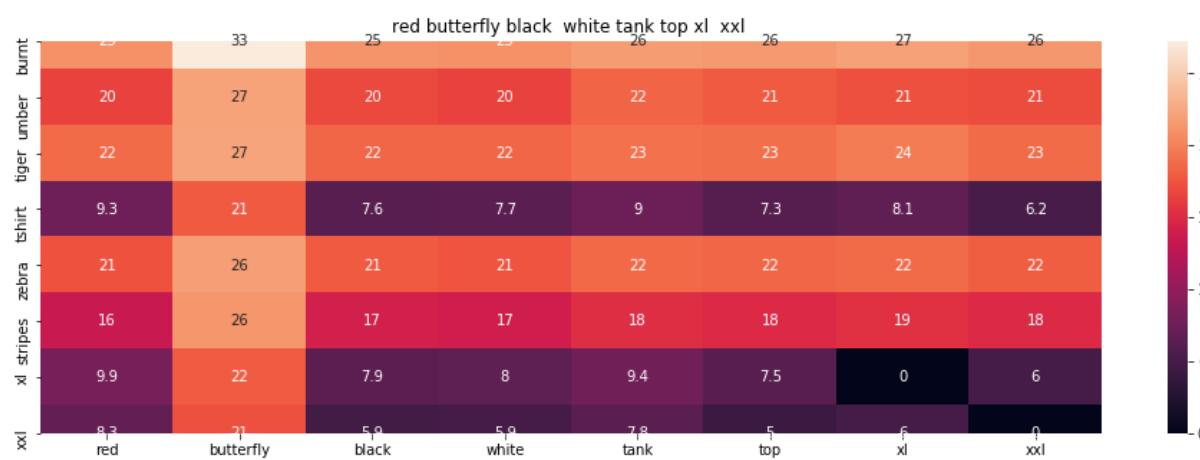
---



ASIN : B00JXQC8L6

Brand : Si Row

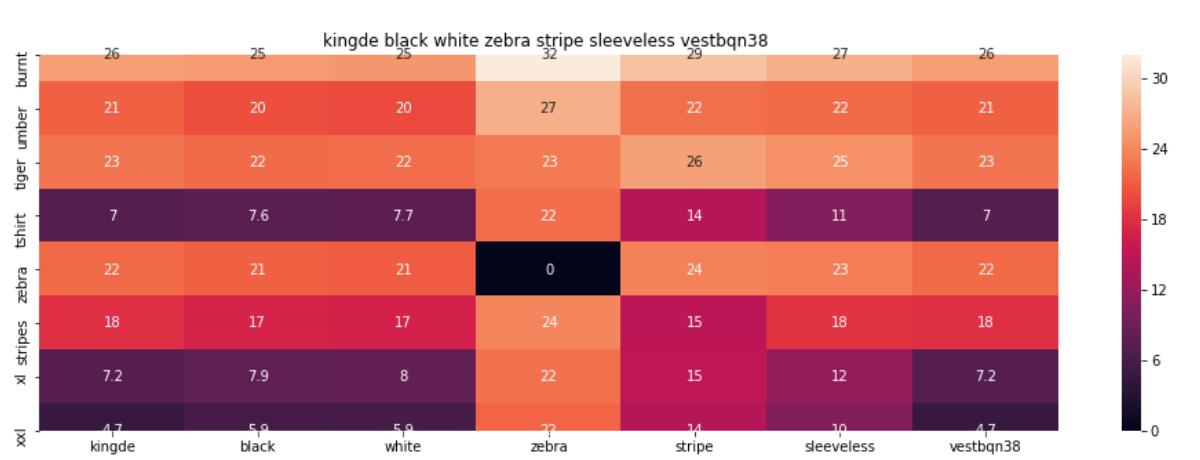
euclidean distance from input : 4.203900146665063



ASIN : B00JV63CW2

Brand : Si Row

euclidean distance from input : 4.286586380185571



ASIN : B015H41F6G

Brand : KINGDE

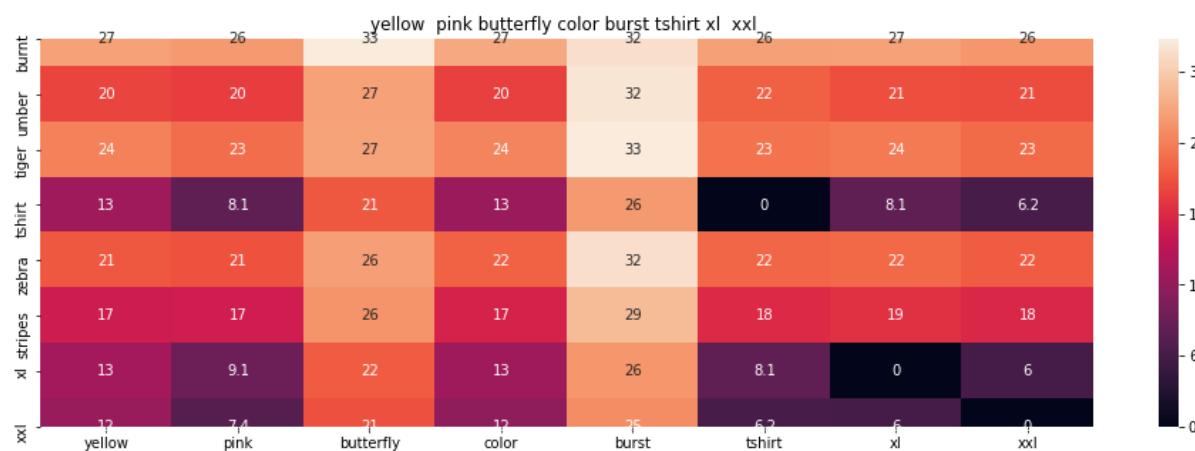
euclidean distance from input : 4.389370406508858

euclidean distance from input : 4.397909927548852

---



---



ASIN : B00JXQBBMI

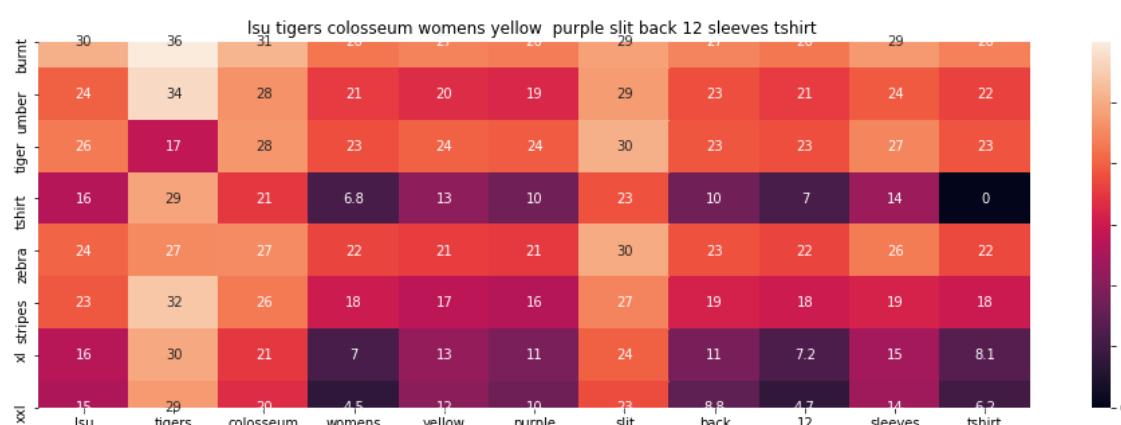
Brand : Si Row

euclidean distance from input : 4.397909927548852

---



---



ASIN : B073R5Q8HD

Brand : Colosseum

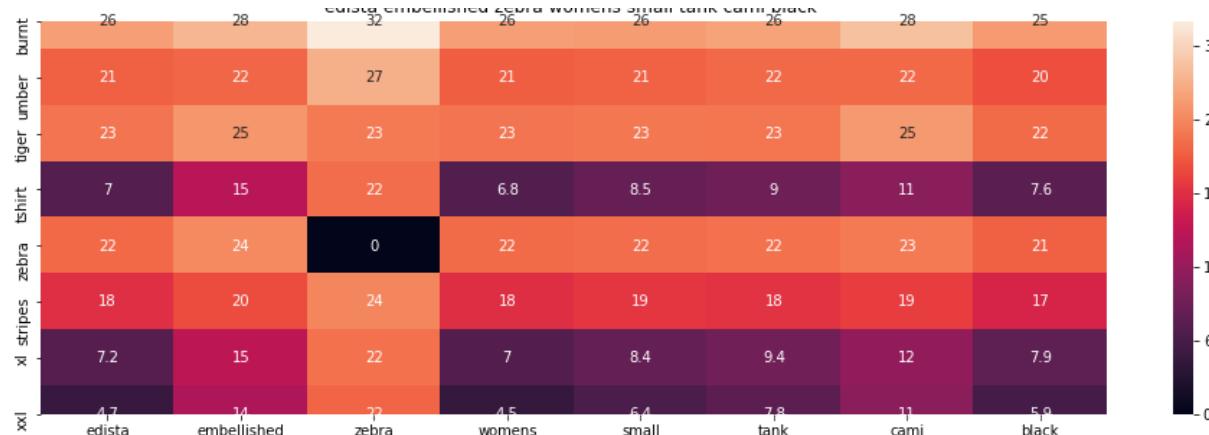
euclidean distance from input : 4.451228392959053

---



---

01\_AppliedAIWorkshop - Jupyter Notebook

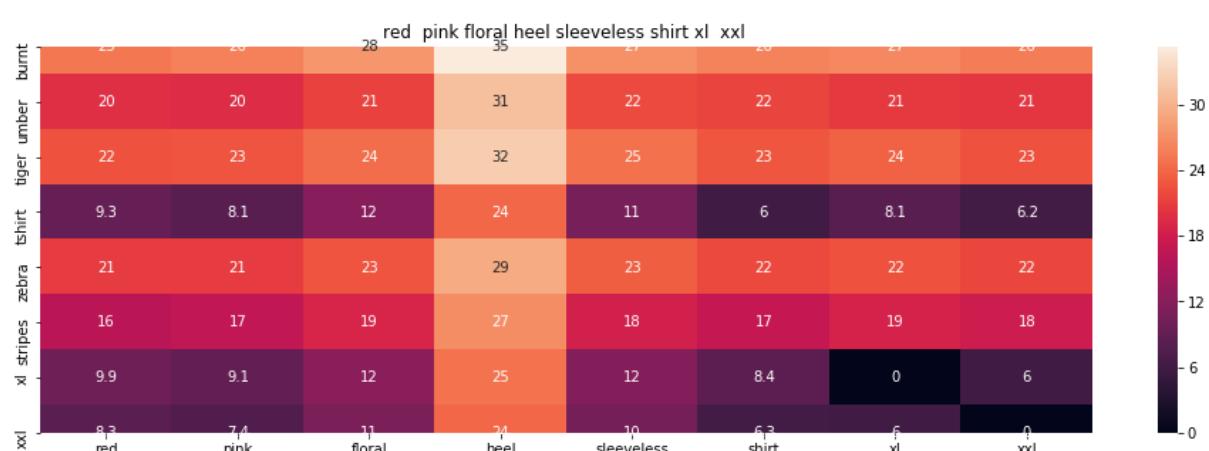


ASIN : B074P8MD22

Brand : Edista

euclidean distance from input : 4.518977416396553

=====  
=====

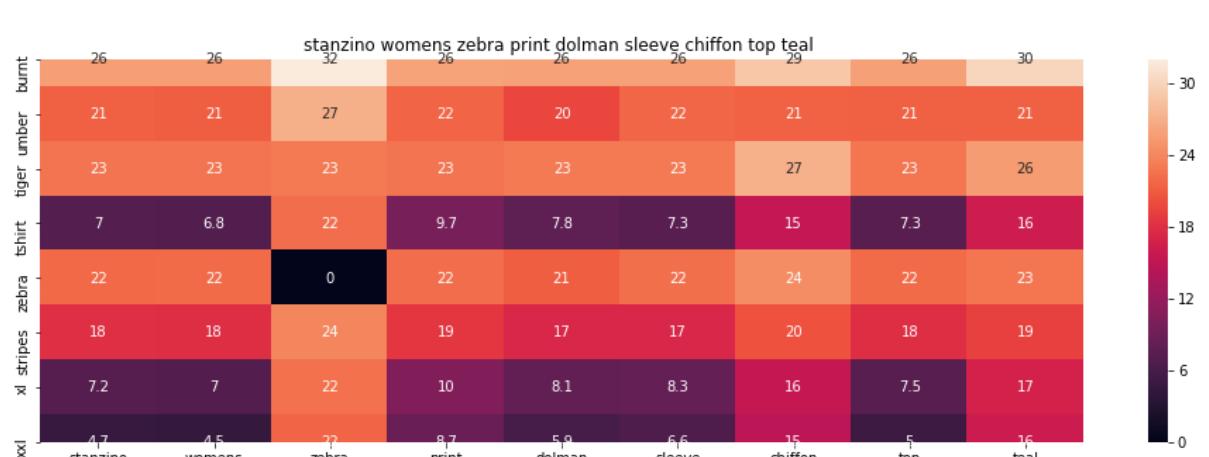


ASTN : B00.TV6300E

ASIN : B00JV63F  
Brand : Si Bow

Brand : Si Row  
euclidean distance from input : 4.529374695004907

Euclidean distance



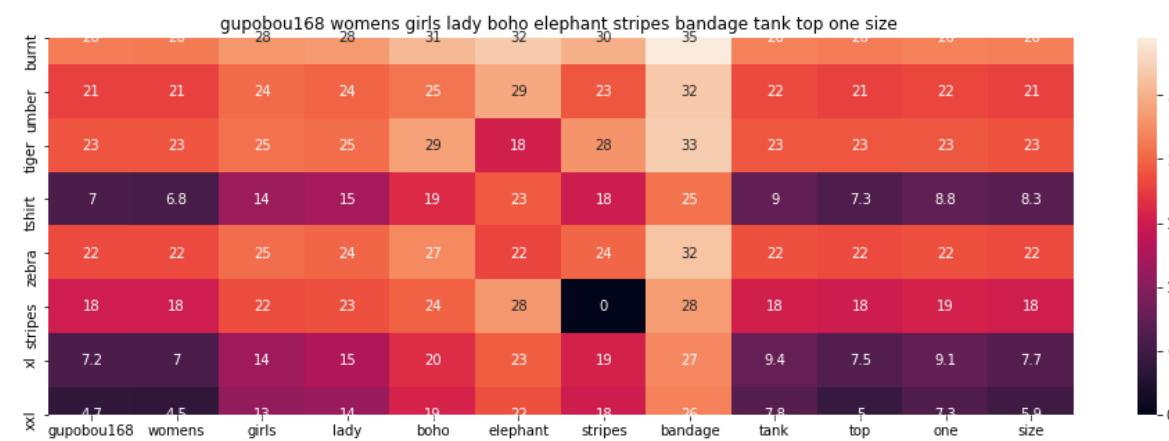
ASIN : B00C0I3U3E

Brand : Stanzino

euclidean distance from input : 4.530325759292061

=====

=====



ASIN : B01ER18406

Brand : GuPoBoU168

euclidean distance from input : 4.546816642558488

=====

=====



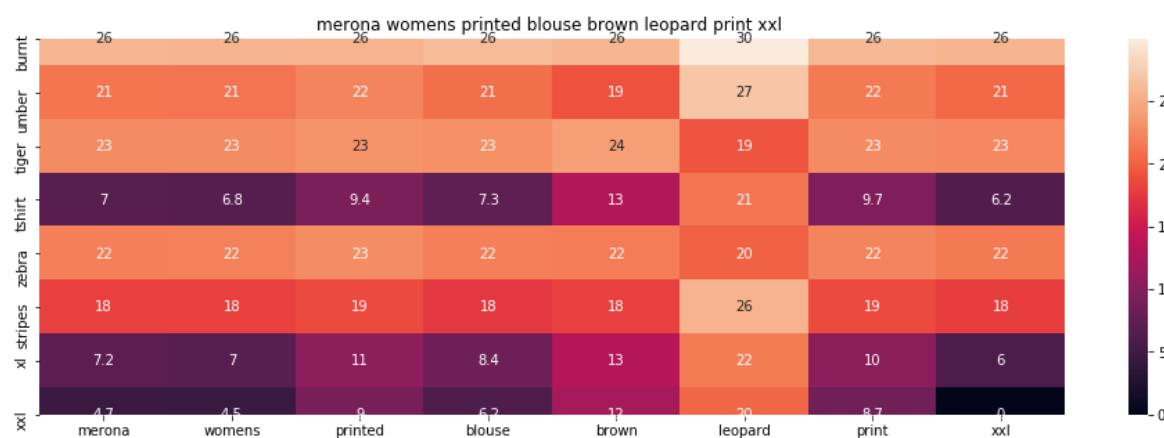
ASIN : B073R4ZM7Y

Brand : Colosseum

euclidean distance from input : 4.548355162978584

=====

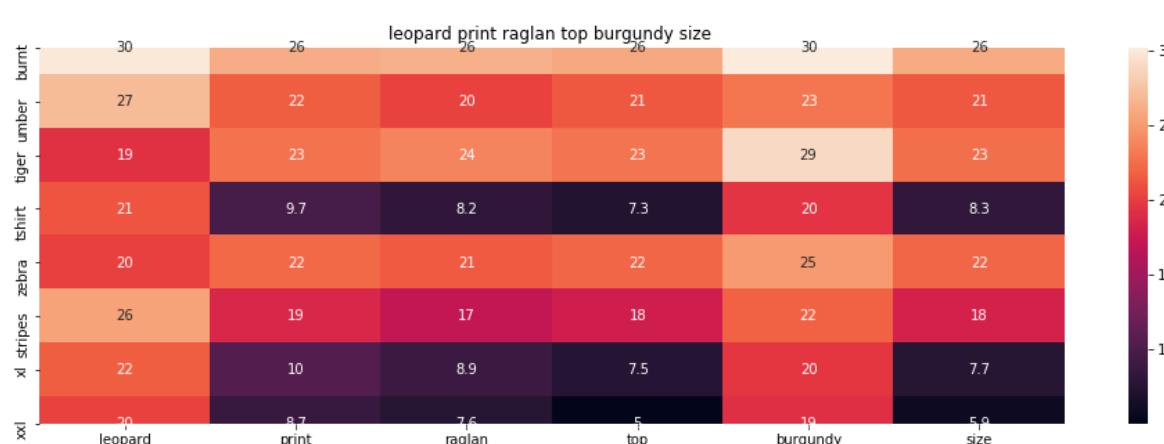
=====



ASIN : B071YF3WDD

Brand : Merona

euclidean distance from input : 4.610626662612374



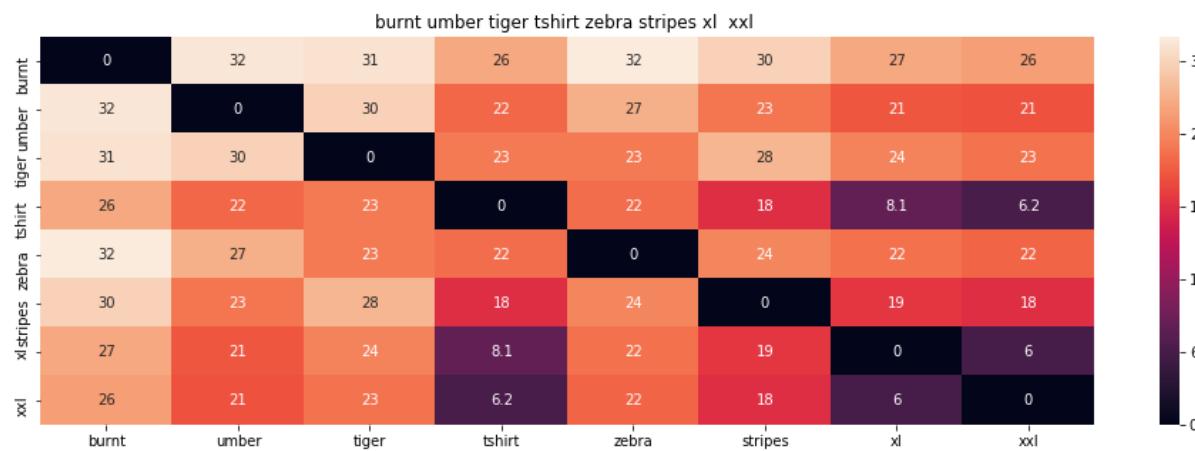
ASIN : B01C60RLDQ

Brand : 1 Mad Fit

euclidean distance from input : 4.645917892817431

```
In [0]: # brand and color weight =50
# title vector weight = 5

idf_w2v_brand(12566, 5, 50, 20)
```



ASIN : B00JXQB5FQ

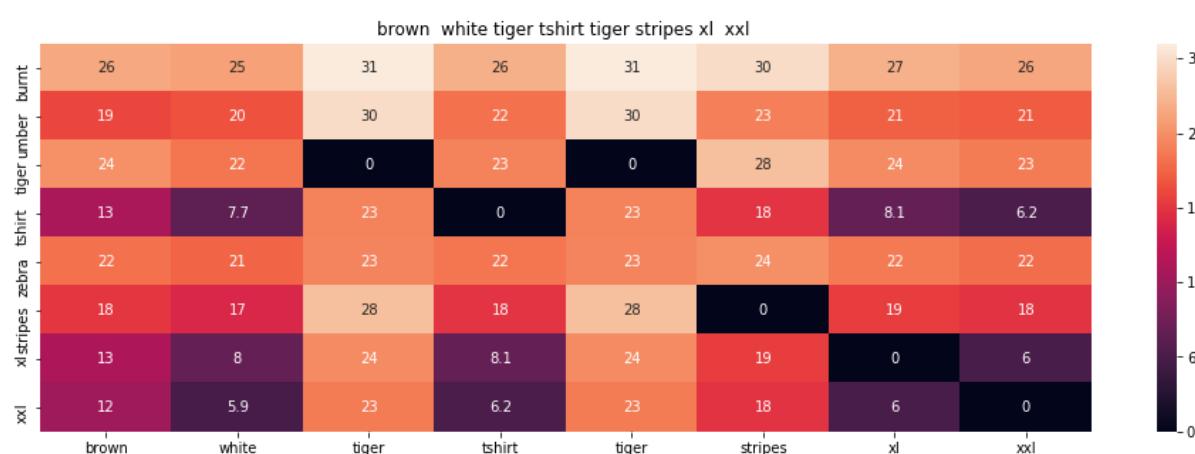
Brand : Si Row

euclidean distance from input : 0.000355113636364

---



---



ASIN : B00JXQCWTO

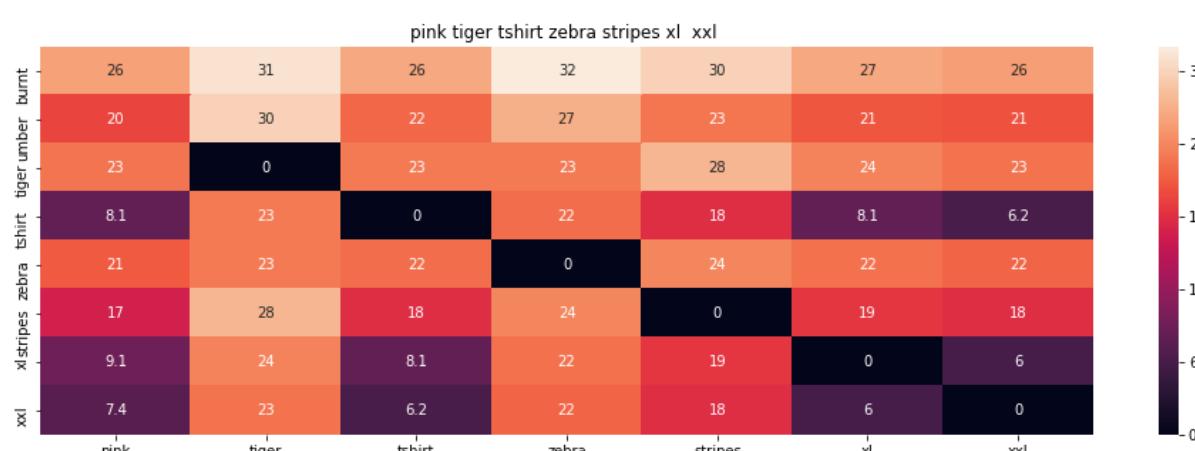
Brand : Si Row

euclidean distance from input : 0.433722027865

---



---



ASIN : B00JXQASS6

Brand : Si Row

euclidean distance from input : 1.65509310669

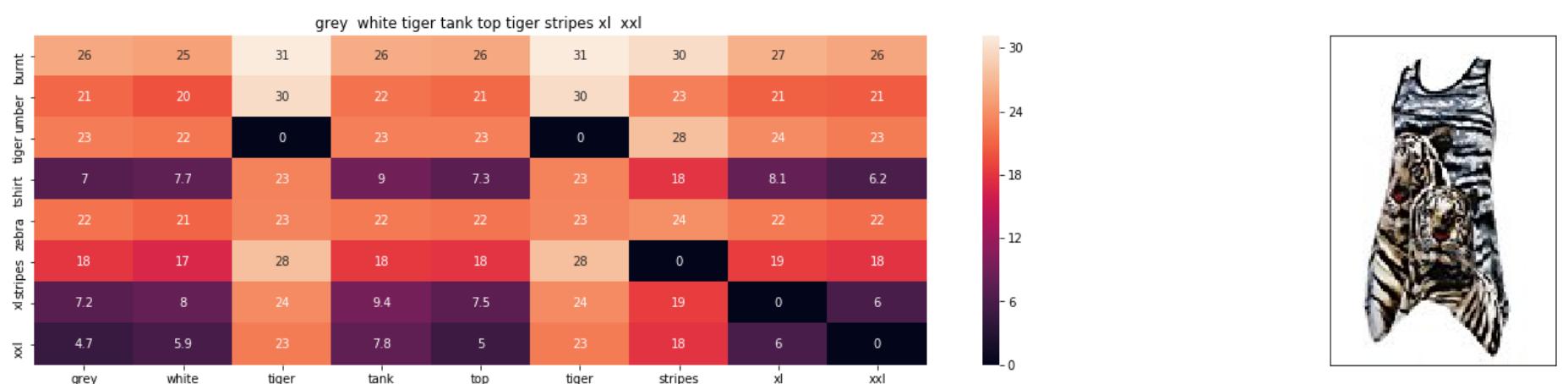
---



---



---



ASIN : B00JXQAFZ2

Brand : Si Row

euclidean distance from input : 1.77293604103

---



---



ASIN : B00JXQAUWA

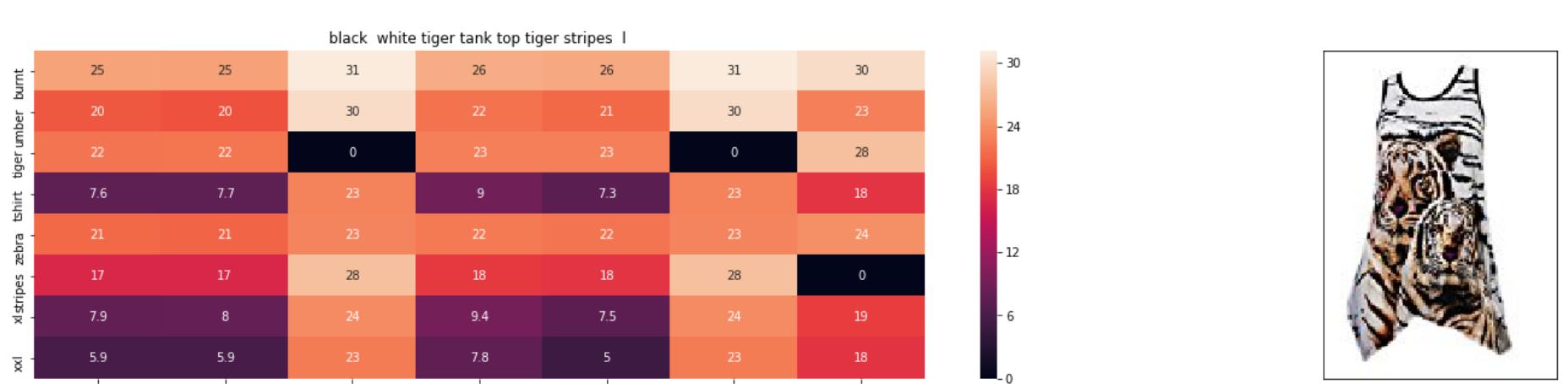
Brand : Si Row

euclidean distance from input : 1.80287808538

---



---



ASIN : B00JXQAO94

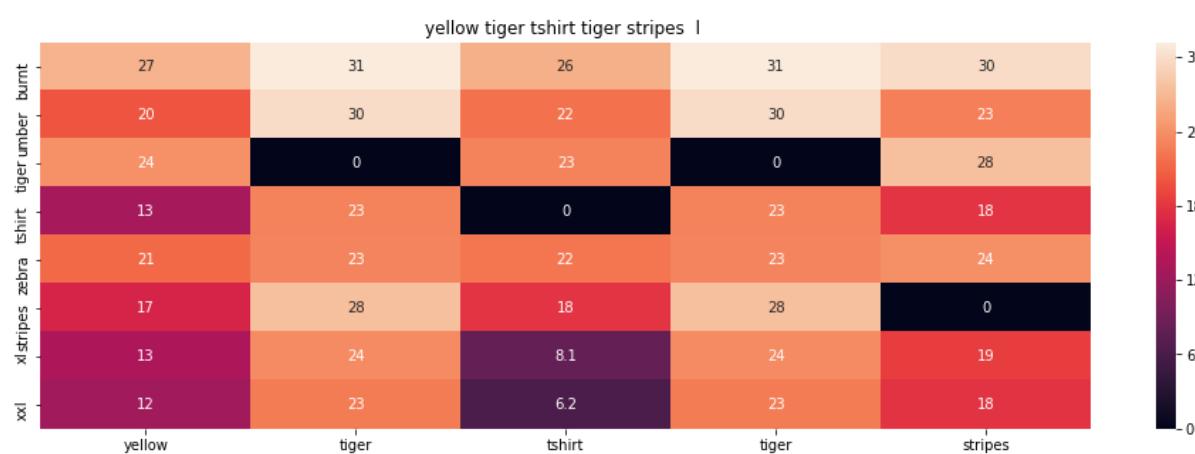
Brand : Si Row

euclidean distance from input : 1.80319609241

---



---



ASIN : B00JXQCUIC

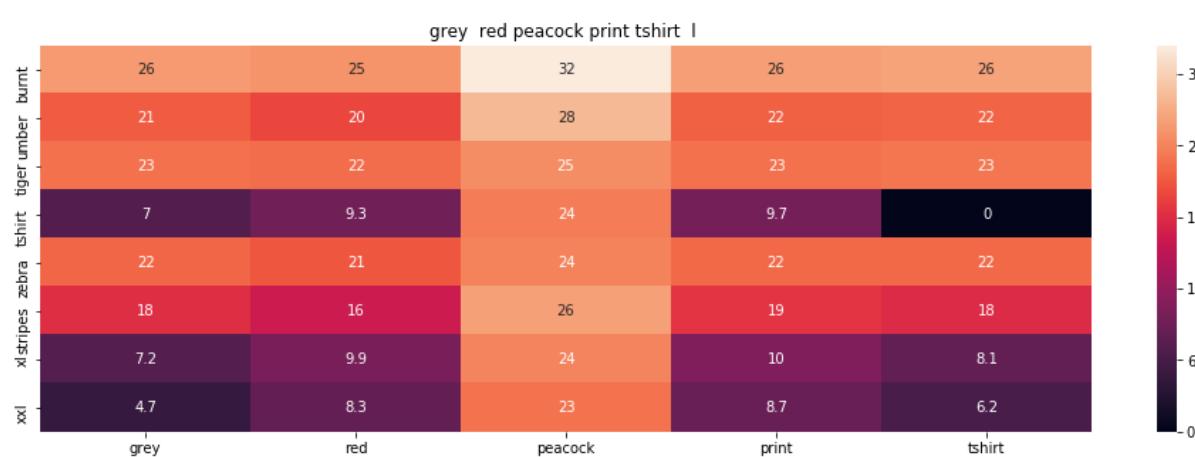
Brand : Si Row

euclidean distance from input : 1.82141619628

---



---



ASIN : B00JXQCFRS

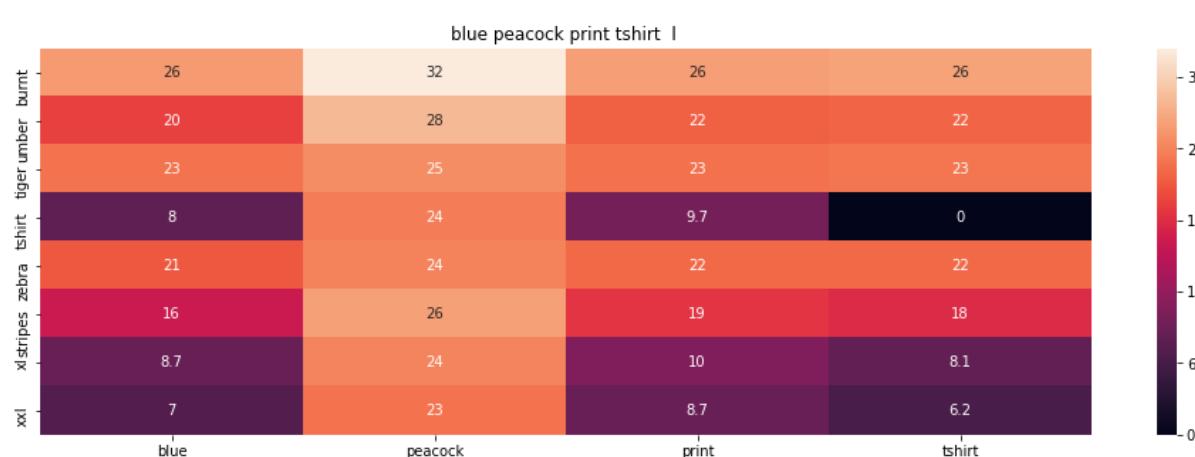
Brand : Si Row

euclidean distance from input : 1.90777685025

---



---



ASIN : B00JXQC8L6

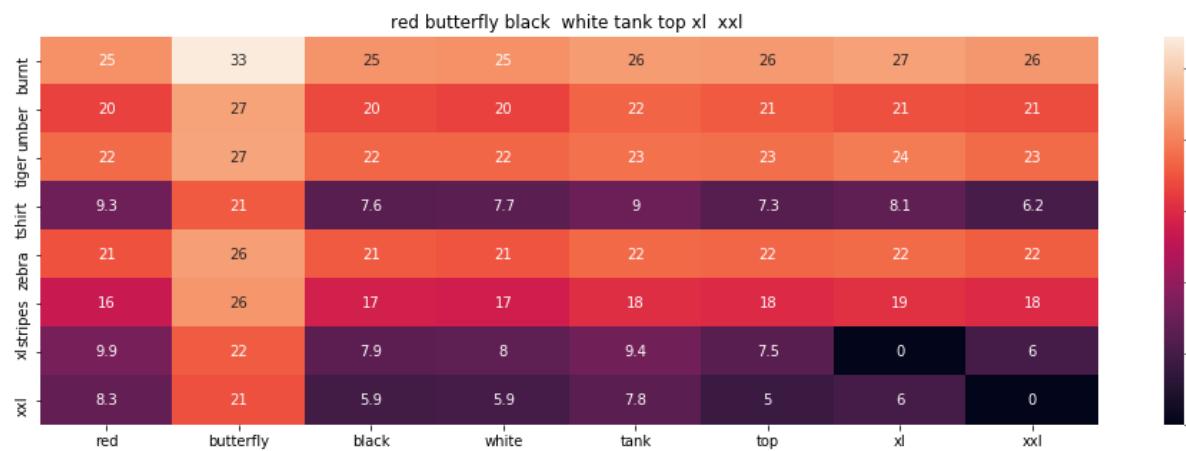
Brand : Si Row

euclidean distance from input : 1.92142937433

---



---



ASIN : B00JV63CW2

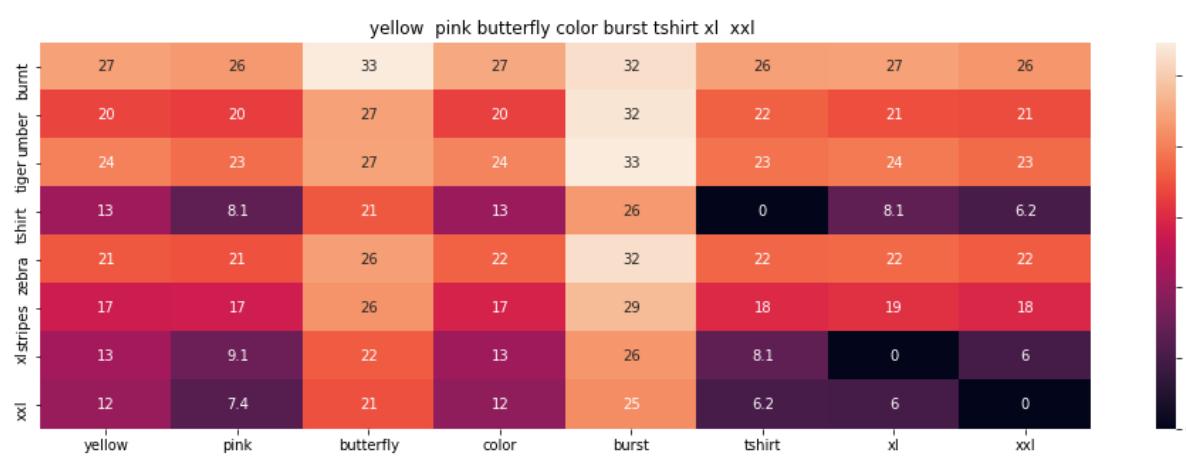
Brand : Si Row

euclidean distance from input : 1.93646323497

---



---



ASIN : B00JXQBBMI

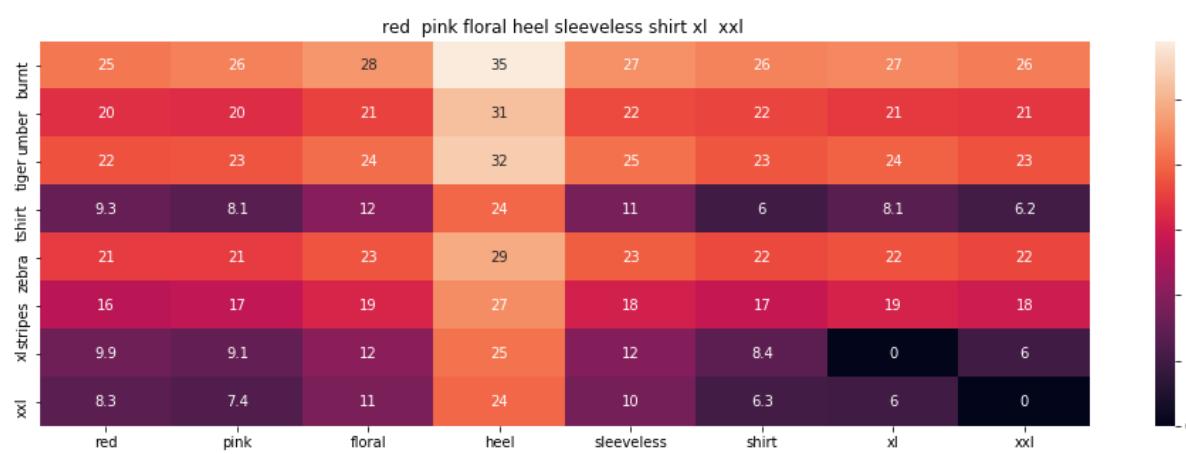
Brand : Si Row

euclidean distance from input : 1.95670381059

---



---



ASIN : B00JV63QOE

Brand : Si Row

euclidean distance from input : 1.9806066343

---



---



ASIN : B00JV63VC8

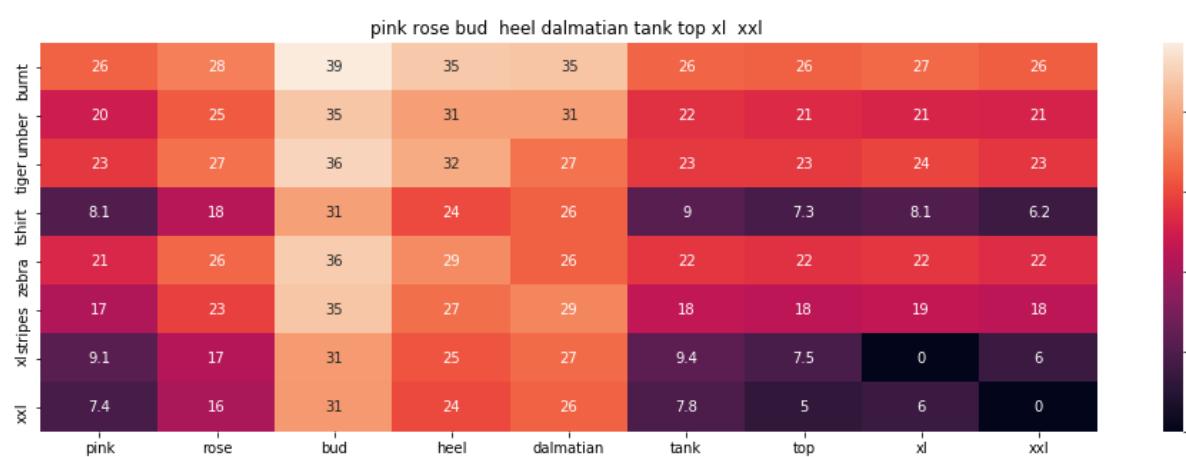
Brand : Si Row

euclidean distance from input : 2.01218559992

---



---



ASIN : B00JXQAX2C

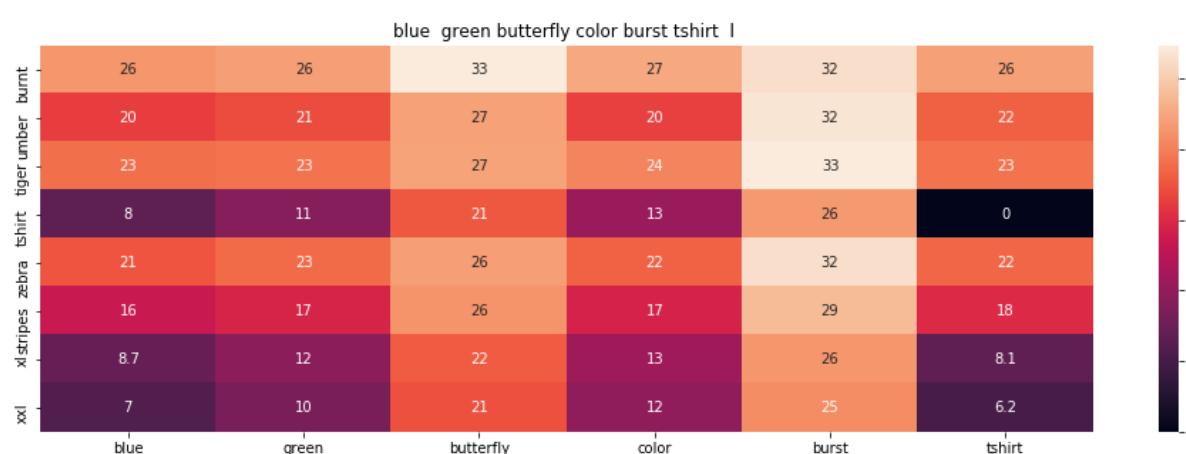
Brand : Si Row

euclidean distance from input : 2.01335178755

---



---



ASIN : B00JXQC0C8

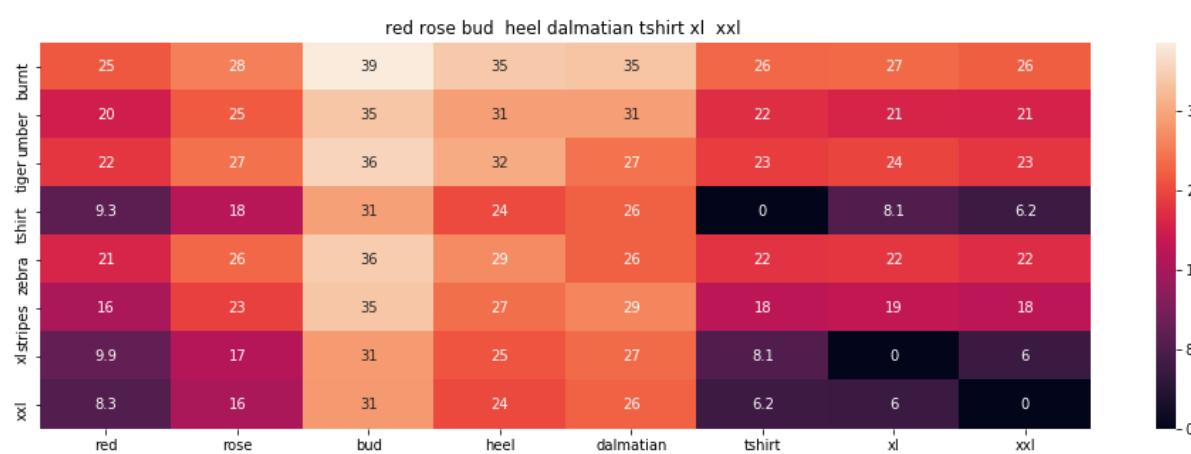
Brand : Si Row

euclidean distance from input : 2.01388334827

---



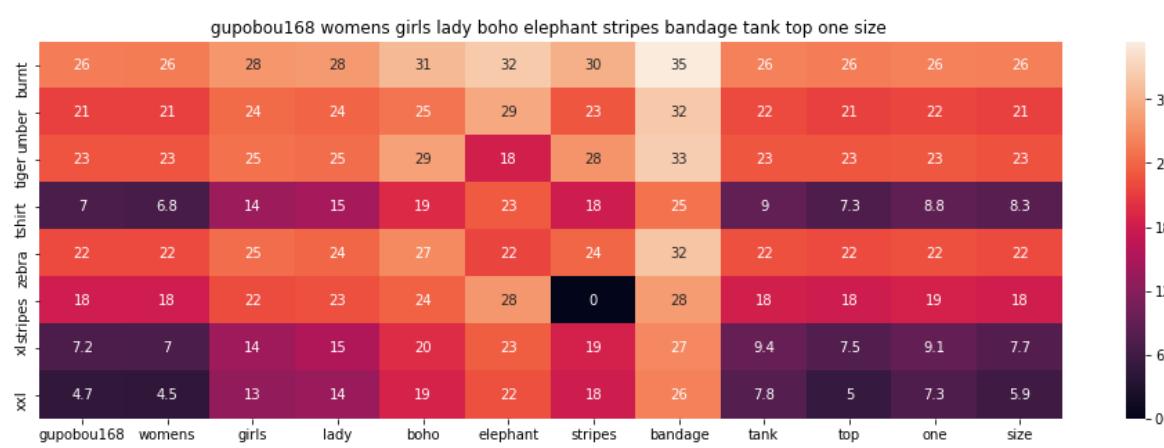
---



ASIN : B00JXQABB0

Brand : Si Row

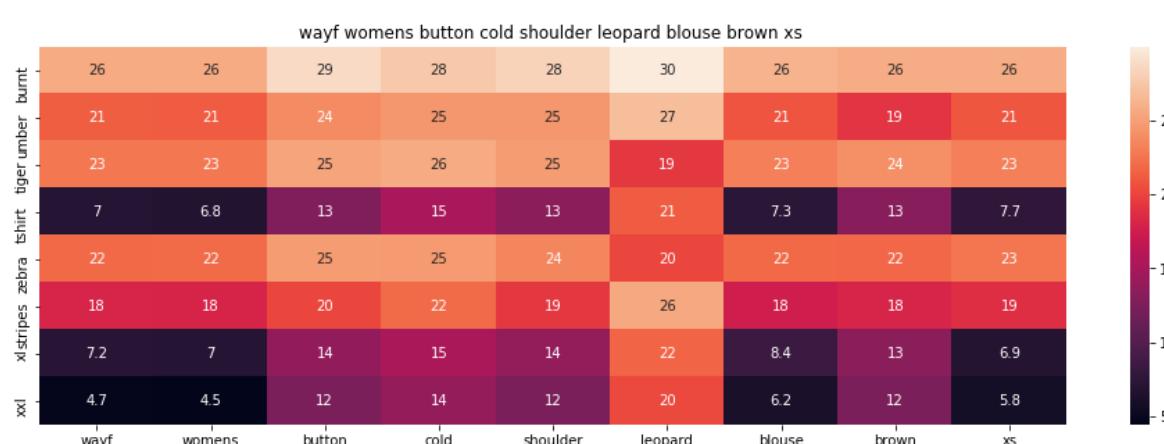
euclidean distance from input : 2.0367257555



ASIN : B01ER18406

Brand : GuPoBoU168

euclidean distance from input : 2.65620416778



ASIN : B01LZ7BQ4H

Brand : WAYF

euclidean distance from input : 2.6849067823

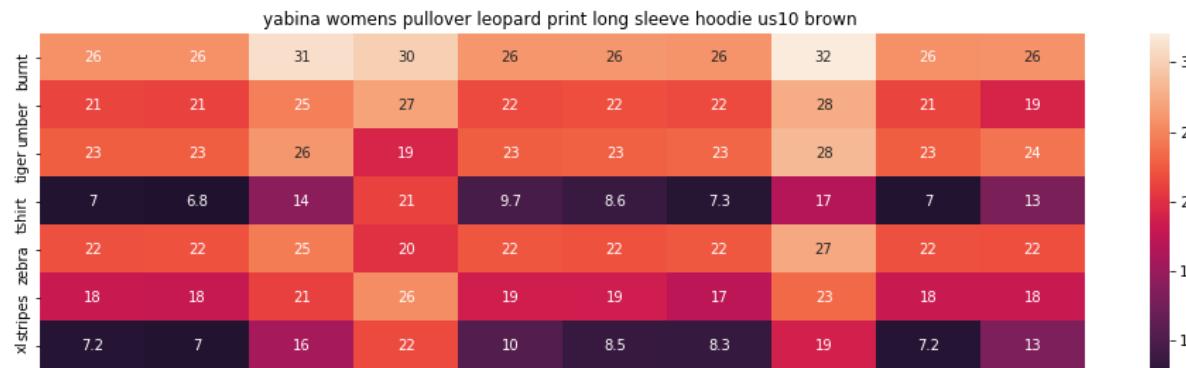
---



---



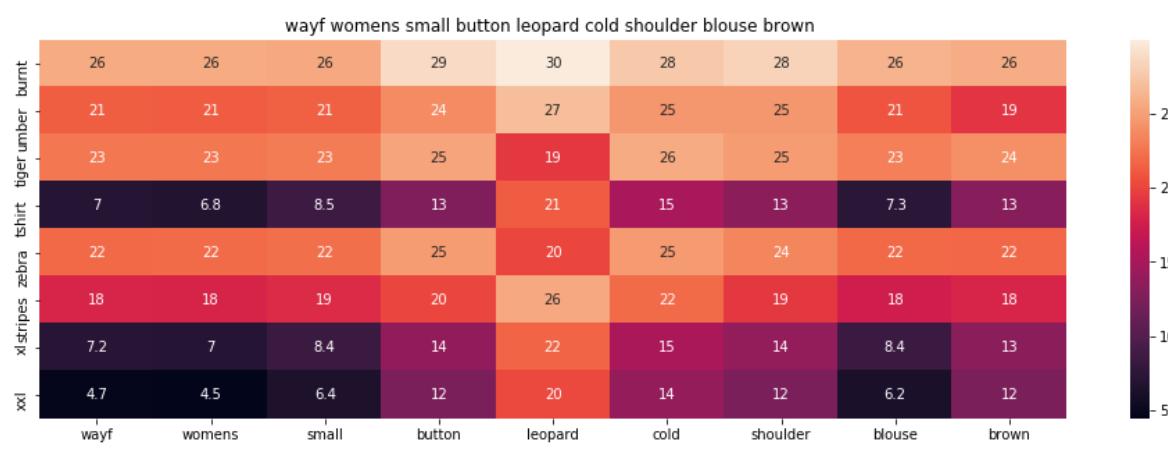
---



ASIN : B01KJUM6JI

Brand : YABINA

euclidean distance from input : 2.68583819266



ASIN : B01M06V4X1

Brand : WAYF

euclidean distance from input : 2.69476194865

=====

## [10.2] Keras and Tensorflow to extract features

```
In [ ]: 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
```

```
In [ ]: # https://gist.github.com/fchollet/f35fbc80e066a49d65f1688a7e99f069
# Code reference: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-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 code 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 [108]: #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_appral_data_preprocessed')
df_asins = list(data['asin'])
```

```
In [127]: #creating new index to reorganize bottleneck feature in same order as tata is:
new_index_cnn = []
for value in range(16042):
    index = asins.index(df_asins[value])
    new_index_cnn.append(index)
```

```
In [128]: bottleneck_features_train_1 = []
for values in new_index_cnn:
    bottleneck_features_train_1.append(bottleneck_features_train[values])
```

In [137]:

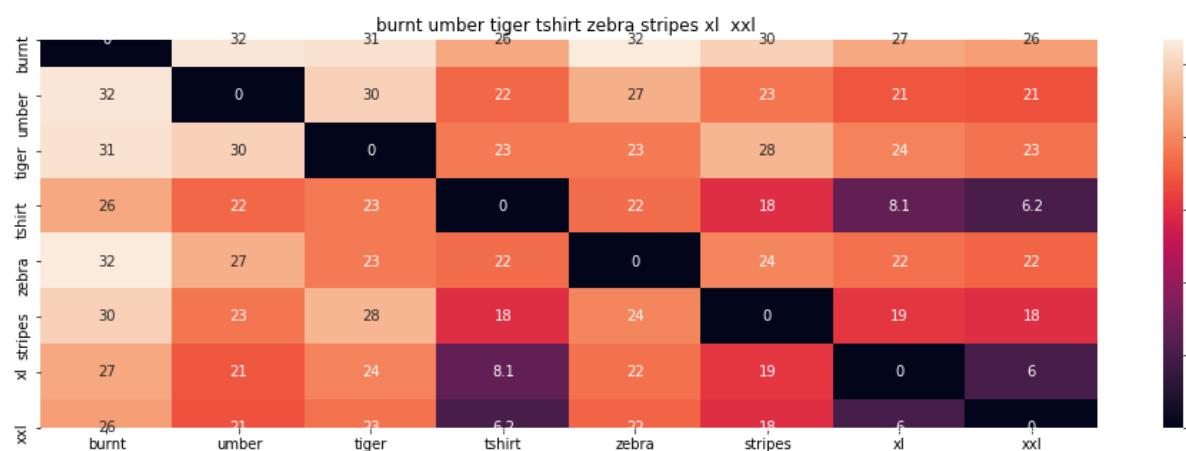
```
'''  
from IPython.display import display, Image, SVG, Math, YouTubeVideo  
import matplotlib.pyplot as plt  
  
#get similar products using CNN features (VGG-16)  
def get_similar_products_cnn(doc_id, num_results):  
    doc_id = asins.index(df_asins[doc_id])  
    pairwise_dist = pairwise_distances(bottleneck_features_train, 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_1(url=row['medium_image_url'])  
            plt.show()  
            print('Product Title: ', row['title'])  
            print('Euclidean Distance from input image:', pdists[i])  
            print('Amazon Url: www.amazon.com/dp/' + asins[indices[i]])  
  
get_similar_products_cnn(12566, 20)'''
```

```
Out[137]: "\nfrom IPython.display import display, Image, SVG, Math, YouTubeVideo\nimport matplotlib.pyplot as plt\n\n# get similar products using CNN features (VGG-16)\ndef get_similar_products_cnn(doc_id, num_results):\n    doc_id = asins.index(df_asins[doc_id])\n    pairwise_dist = pairwise_distances(bottleneck_features_train, bottleneck_features_train[doc_id].reshape(1,-1))\n    indices = np.argsort(pairwise_dist.flatten())[0:num_results]\n    pdists = np.sort(pairwise_dist.flatten())[0:num_results]\n    for i in range(len(indices)):\n        rows = data[['medium_image_url','title']].loc[data['asin']==asins[indices[i]]]\n        for indx, row in rows.iterrows():\n            display_1(url=row['medium_image_url'])\n            plt.show()\n            print('Product Title: ', row['title'])\n            print('Euclidean Distance from input image:', pdists[i])\n            print('Amazon Url: www.amazon.com/dp/' + asins[indices[i]])\nget_similar_products_cnn(12566, 20)"
```

In [135]:

```
import PIL.Image  
  
def get_similar_products_cnn(doc_id, 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  
  
    pairwise_dist = pairwise_distances(bottleneck_features_train_1, bottleneck_features_train_1[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_brand(data['title'].loc[df_indices[0]], data['title'].loc[df_indices[i]], data['medium_image_url'].loc[df_indices[0]], data['medium_image_url'].loc[df_indices[i]])  
        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)  
  
    # in the give heat map, each cell contains the euclidean distance between words i, j  
    #B01M0XXFKK
```

In [136]: `get_similar_products_cnn(12566, 20)`



ASIN : B00JXQB5FQ

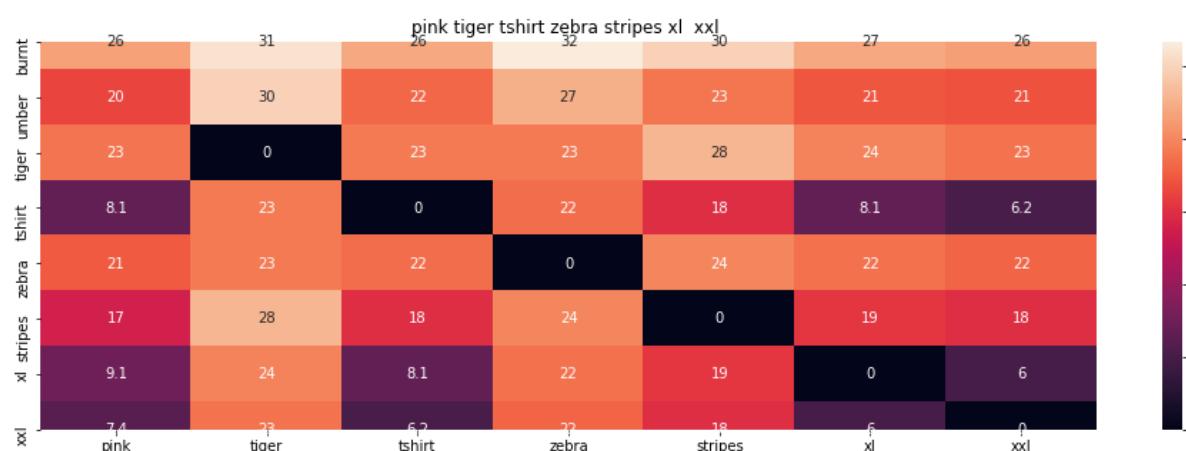
Brand : Si Row

euclidean distance from input : 6.32596e-06

---



---



ASIN : B00JXQASS6

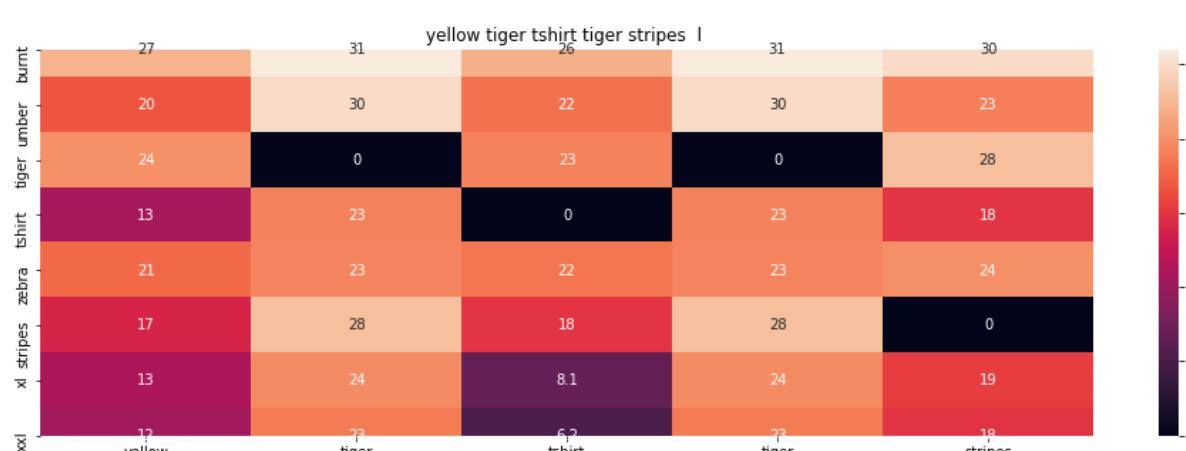
Brand : Si Row

euclidean distance from input : 30.05017

---



---



ASIN : B00JXQCUIC

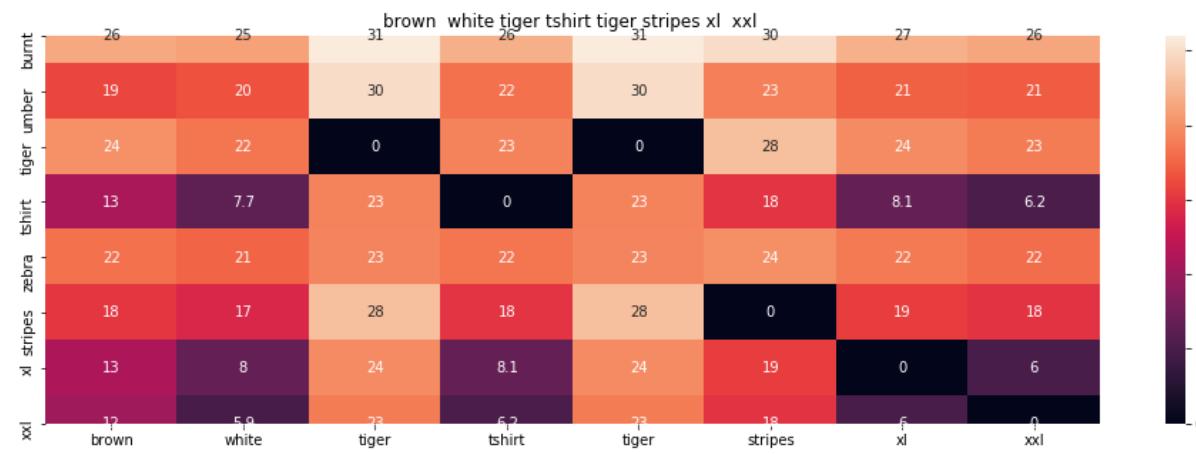
Brand : Si Row

euclidean distance from input : 41.261116

---



---



ASIN : B00JXQCWTO

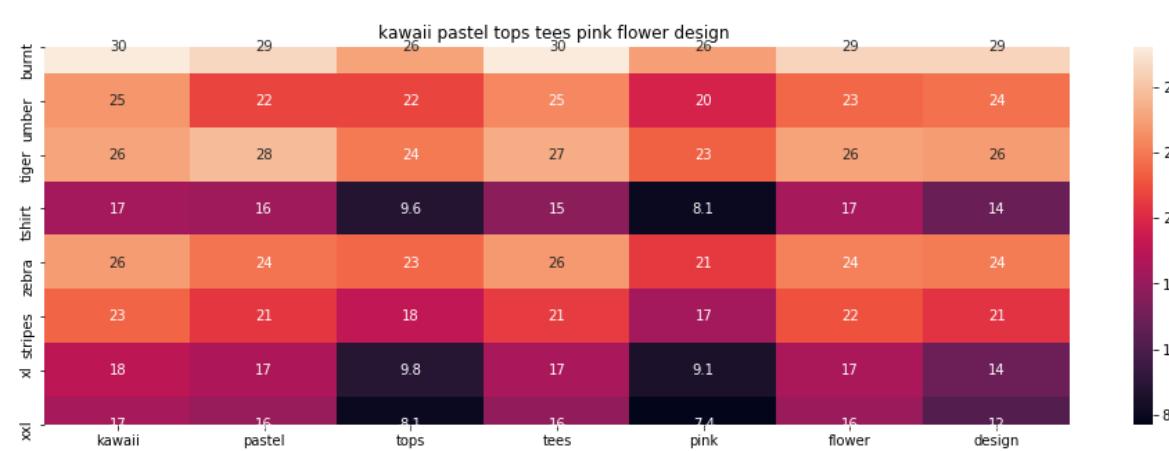
Brand : Si Row

euclidean distance from input : 44.000156

---



---



ASIN : B071FCWD97

Brand : Namnoi Clothing Store

euclidean distance from input : 47.38248

---



---

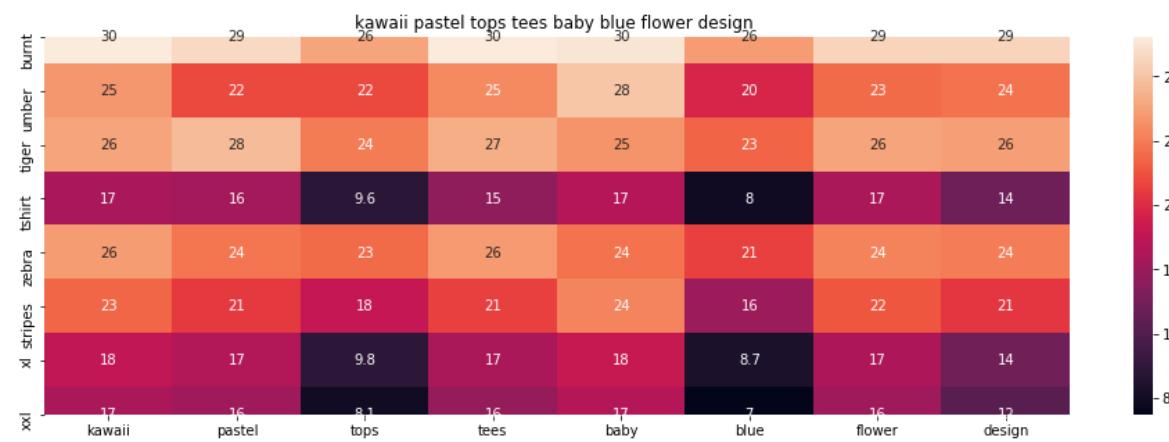


ASIN : B01JUNHBRM

Brand : Namnoi Clothing Store

euclidean distance from input : 47.71842

=====

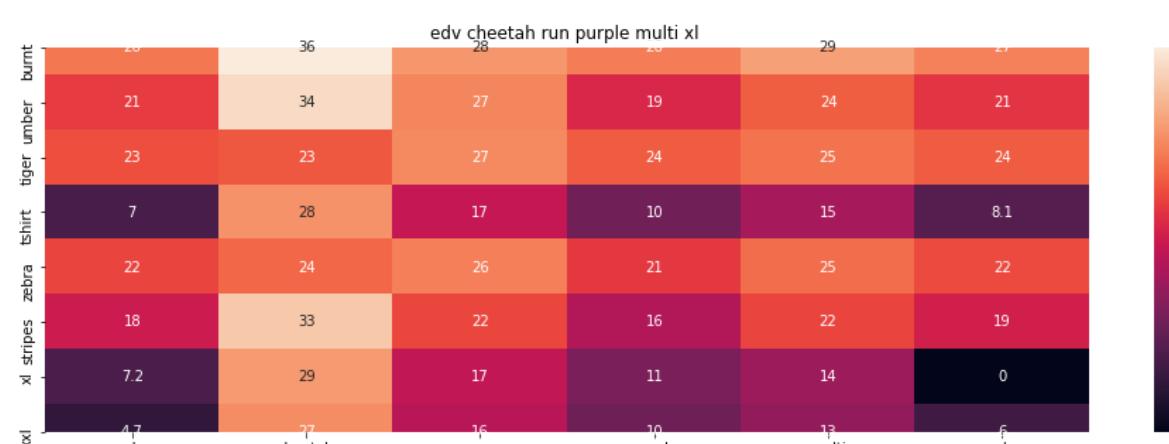


ASIN : B071SBCY9W

Brand : Namnoi Clothing Store

euclidean distance from input : 47.90206

=====



ASIN : B01CUPYBM0

Brand : Styleco

euclidean distance from input : 48.046482

=====

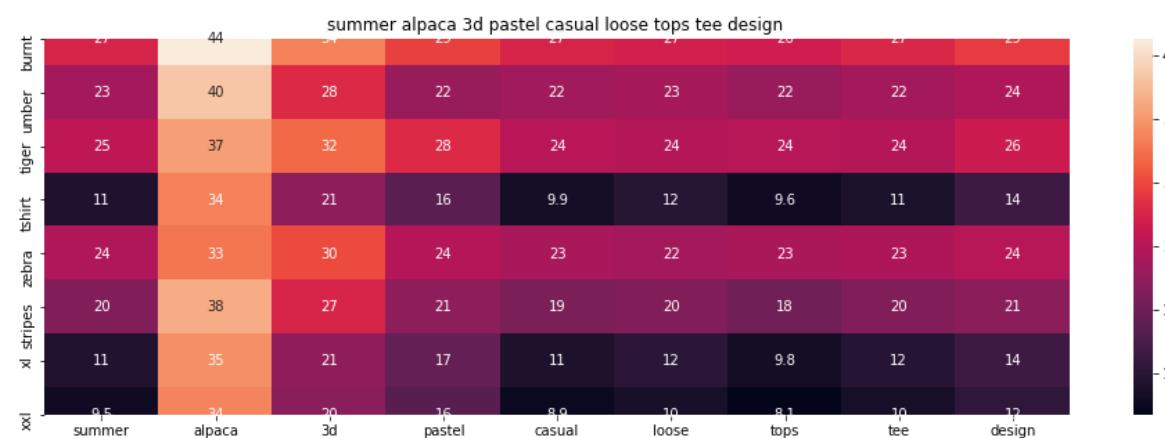


ASIN : B01F7PHXY8

Brand : Danskin Now

euclidean distance from input : 48.101837

=====

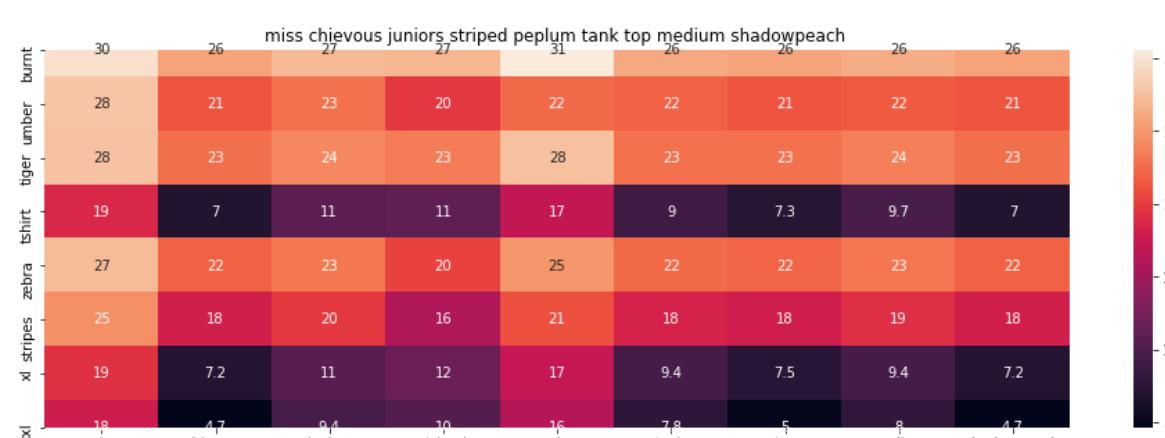


ASIN : B01I80A93G

Brand : Namnoi Clothing Store

euclidean distance from input : 48.118866

=====

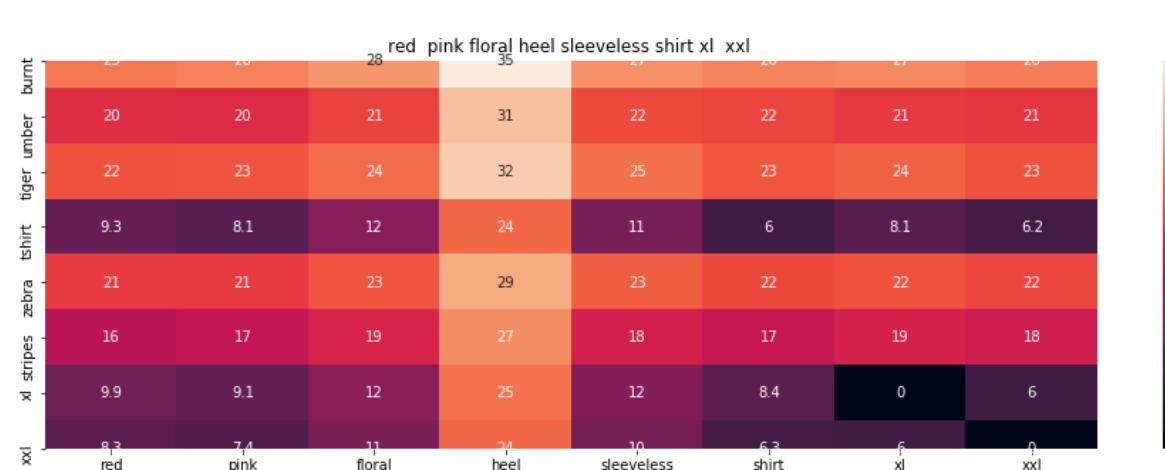


ASIN : B0177DM70S

Brand : MISS CHIEVOUS/AROUND THE WORLD APP

euclidean distance from input : 48.13122

=====



ASIN : B00JV63QOE

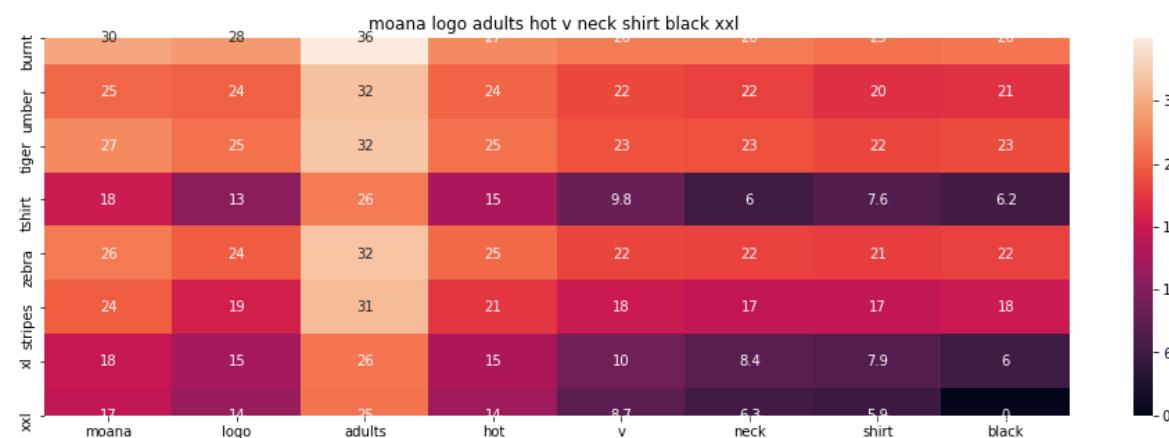
Brand : Si Row

euclidean distance from input : 48.16945

---



---



ASIN : B01LX6H43D

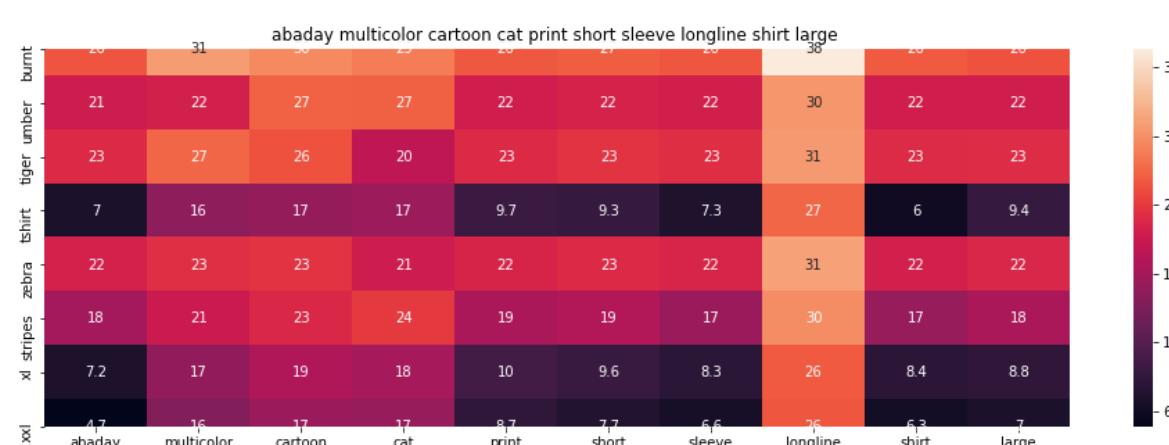
Brand : BOBOB

euclidean distance from input : 48.256786

---



---



ASIN : B01CR57YY0

Brand : ABADAY

euclidean distance from input : 48.265686

---



---



ASIN : B071WYLBZS

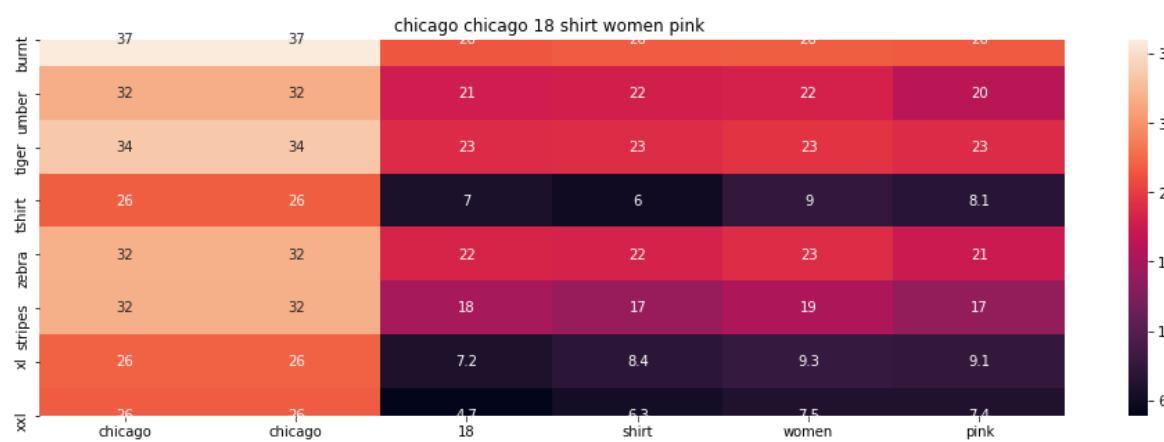
Brand : Namnoi Clothing Store

euclidean distance from input : 48.362602

---



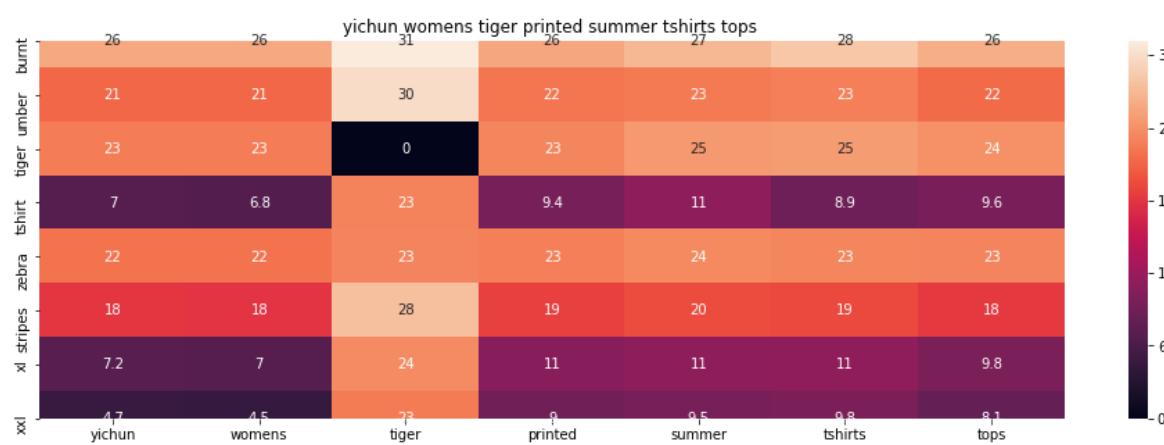
---



ASIN : B01GXAZTRY

Brand : Tony Arden

euclidean distance from input : 48.383606



ASIN : B010NN9RXO

Brand : YICHUN

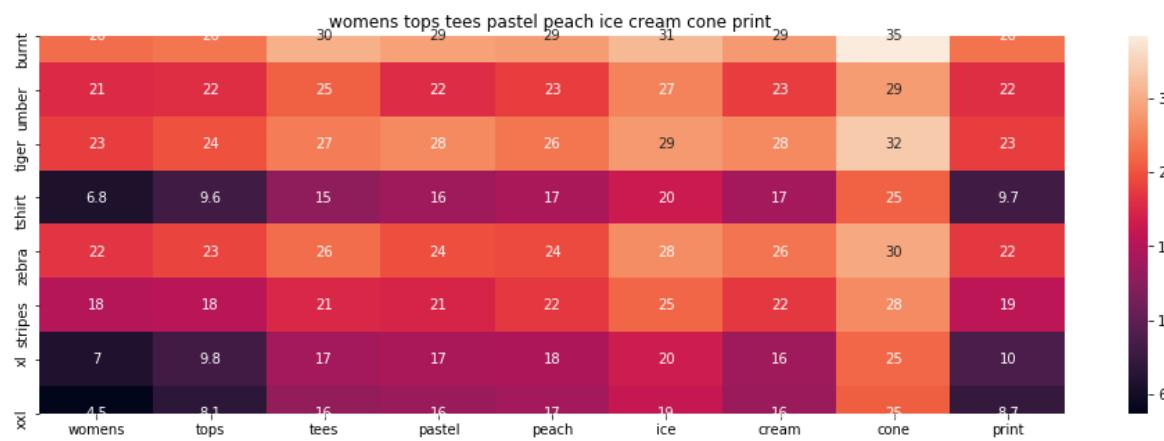
euclidean distance from input : 48.449356



ASIN : B01MPX6IDX

Brand : Nancy Lopez

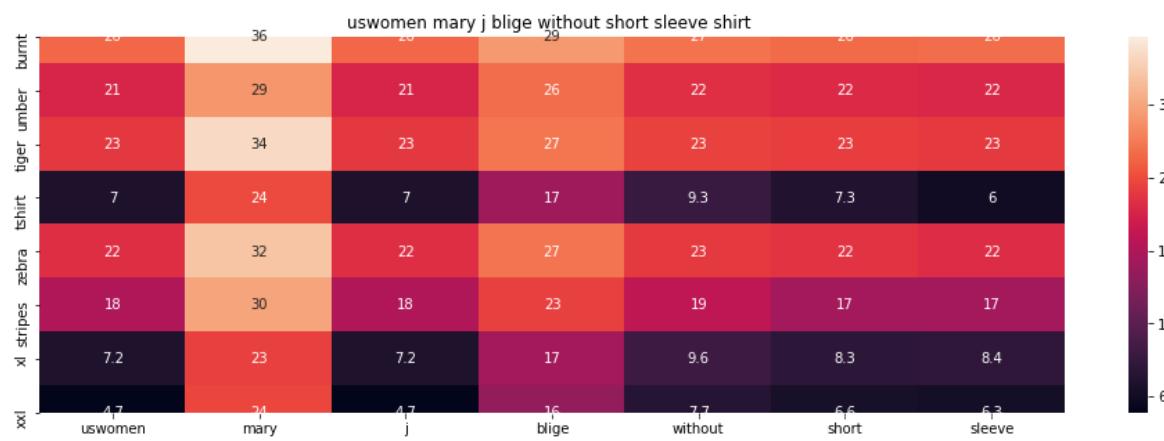
euclidean distance from input : 48.47889



ASIN : B0734GRKZL

Brand : Namnoi Clothing Store

euclidean distance from input : 48.557957



ASIN : B01LYNWX9M

Brand : South Dakota A.

euclidean distance from input : 48.614372

In [143]:

```
import PIL.Image

def idf_w2v_bottelnack_brand(doc_id, w1, w2, w3, w4, 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  $K(X, Y) = \langle X, Y \rangle / (\|X\| * \|Y\|)$ 
    # 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])

    brand_dist = pairwise_distances(brand_features,brand_features[doc_id])

    bottelnack_dist = pairwise_distances(bottleneck_features_train_1, bottleneck_features_train_1[doc_id].reshape(1,-1))

    pairwise_dist = (w1 * idf_w2v_dist + w2 * ex_feat_dist + w3 * brand_dist + w4 * bottelnack_dist)/float(w)

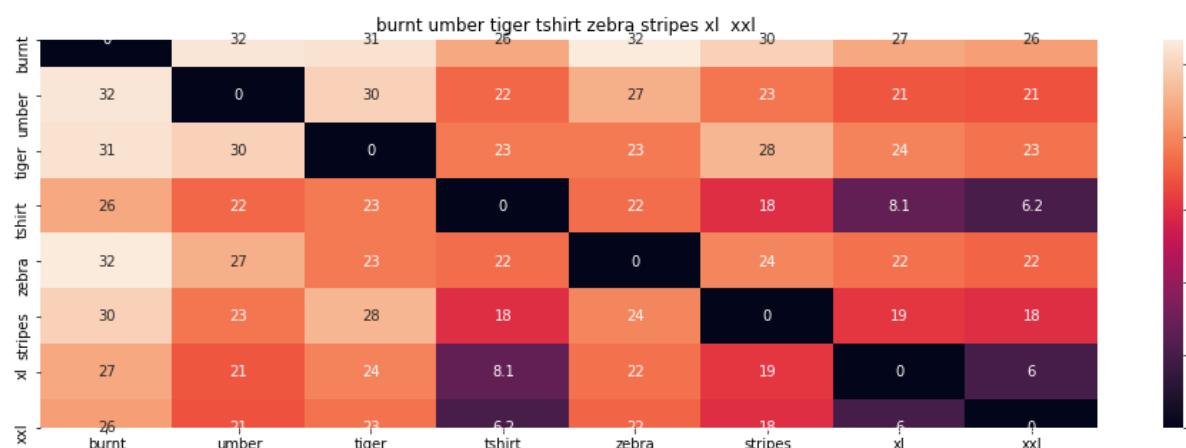
    # np.argsort will return indices of 9 smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the 9 smallest distances
    pdists = np.sort(pairwise_dist.flatten())[0:num_results]

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

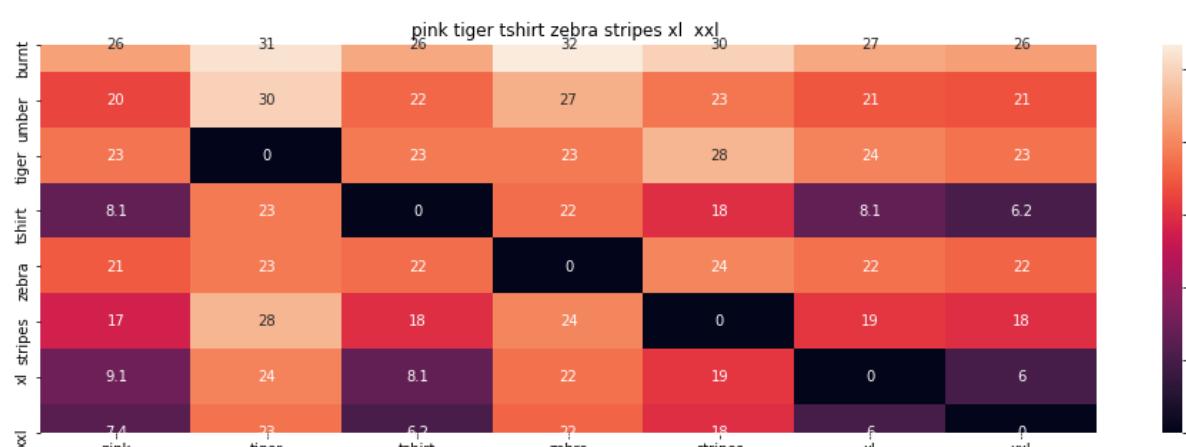
    for i in range(0, len(indices)):
        heat_map_w2v_brand(data['title'].loc[df_indices[0]],data['title'].loc[df_indices[i]], data['medium_image'].loc[df_indices[i]])
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('Brand :',data['brand'].loc[df_indices[i]])
        print('euclidean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

    # in the give heat map, each cell contains the euclidean distance between words i, j
    #B01M0XXFKK
```

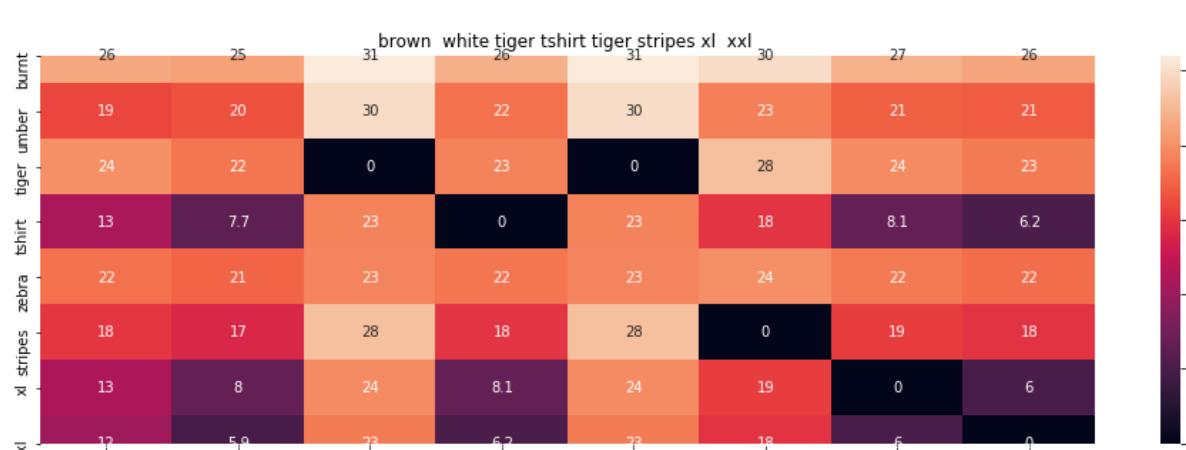
```
In [139]: idf_w2v_bottelnack_brand(12566,10,1,2,1, 23)
```



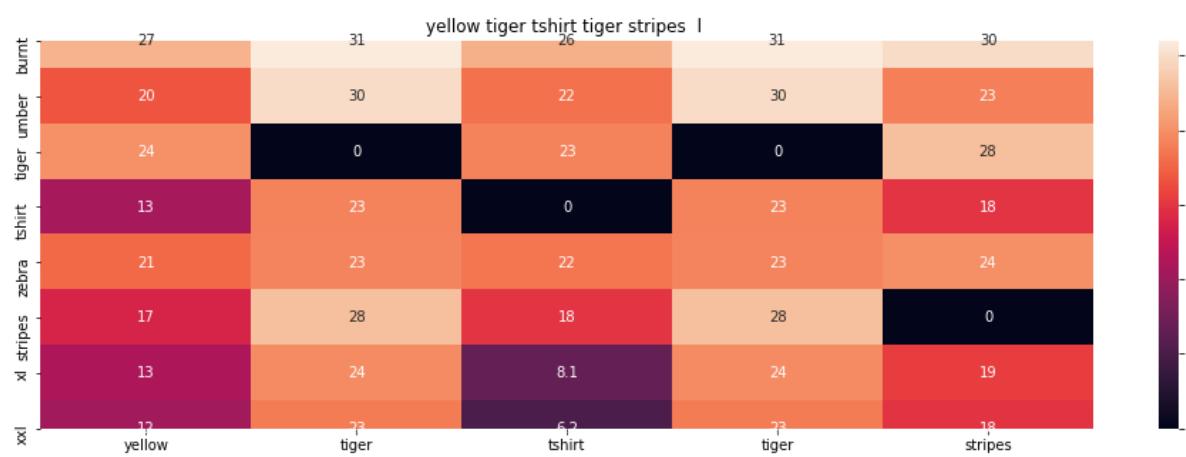
ASIN : B00JXQB5FQ  
 Brand : Si Row  
 euclidean distance from input : 4.518542807220781e-07  
 =====  
 =====



ASIN : B00JXQASS6  
 Brand : Si Row  
 euclidean distance from input : 5.150232206097939  
 =====  
 =====



ASIN : B00JXQCWTO  
 Brand : Si Row  
 euclidean distance from input : 6.550683430262974  
 =====  
 =====



ASIN : B00JXQCUIC

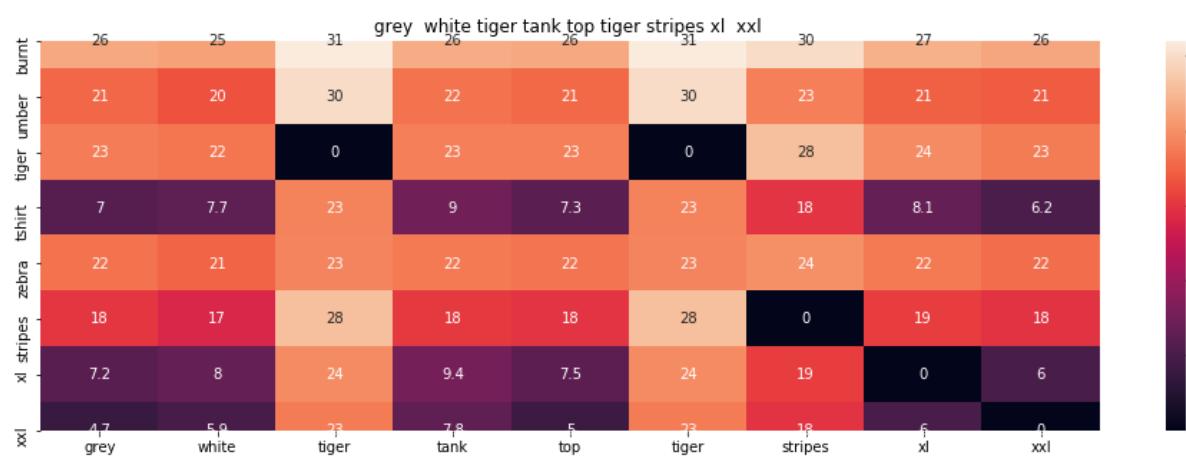
Brand : Si Row

euclidean distance from input : 7.257839366393697

---



---



ASIN : B00JXQAFZ2

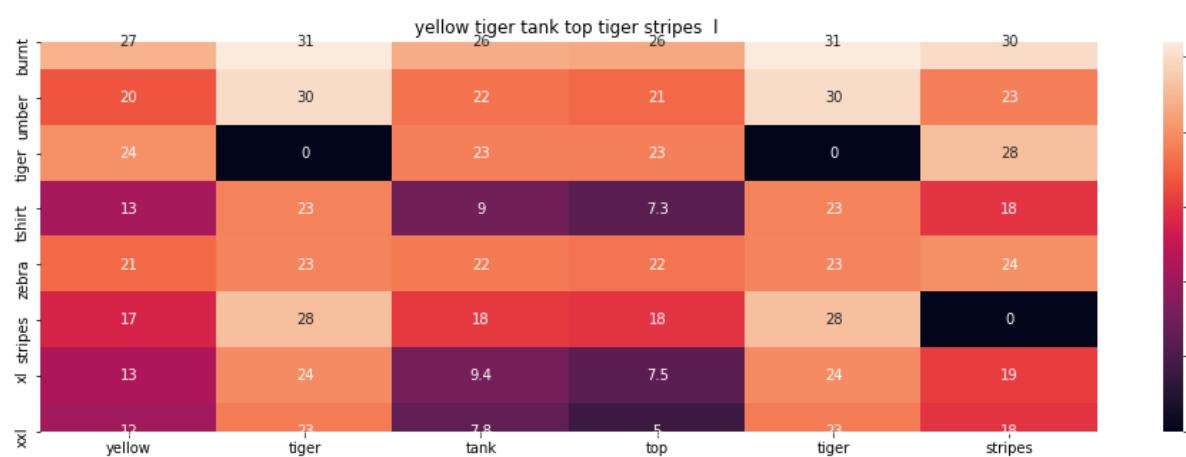
Brand : Si Row

euclidean distance from input : 7.578161948093614

---



---



ASIN : B00JXQAUWA

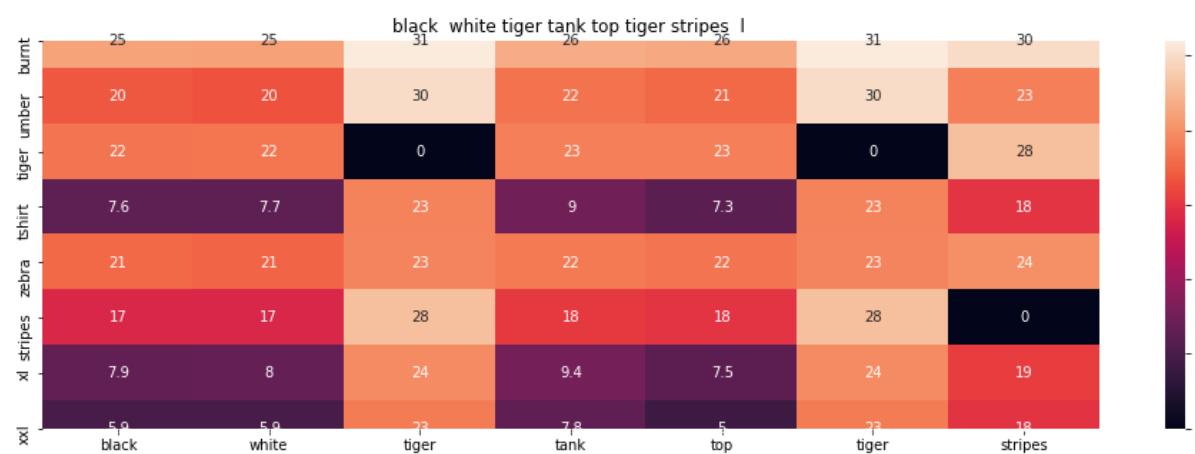
Brand : Si Row

euclidean distance from input : 7.74997482302386

---



---



ASIN : B00JXQAO94

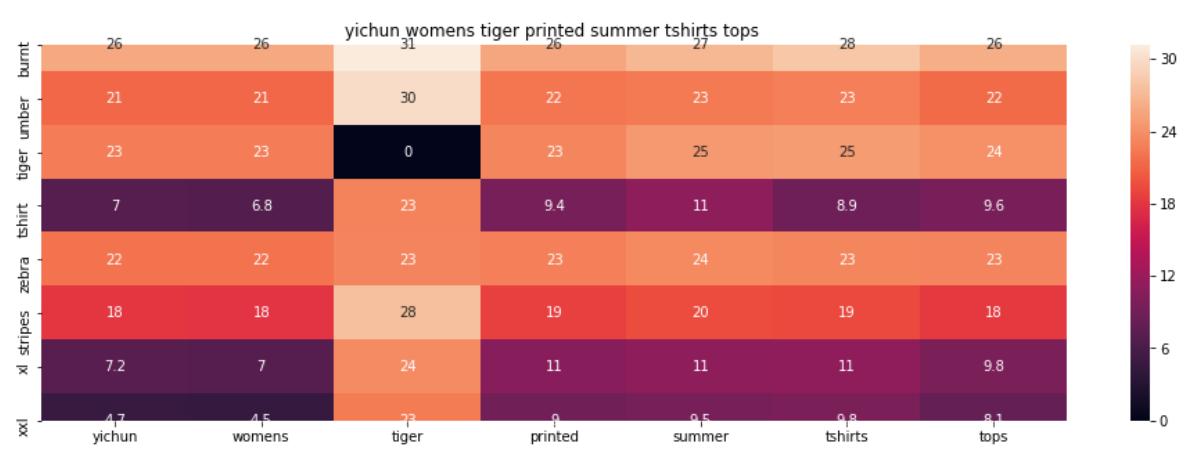
Brand : Si Row

euclidean distance from input : 7.760693032426761

---



---



ASIN : B010NN9RXO

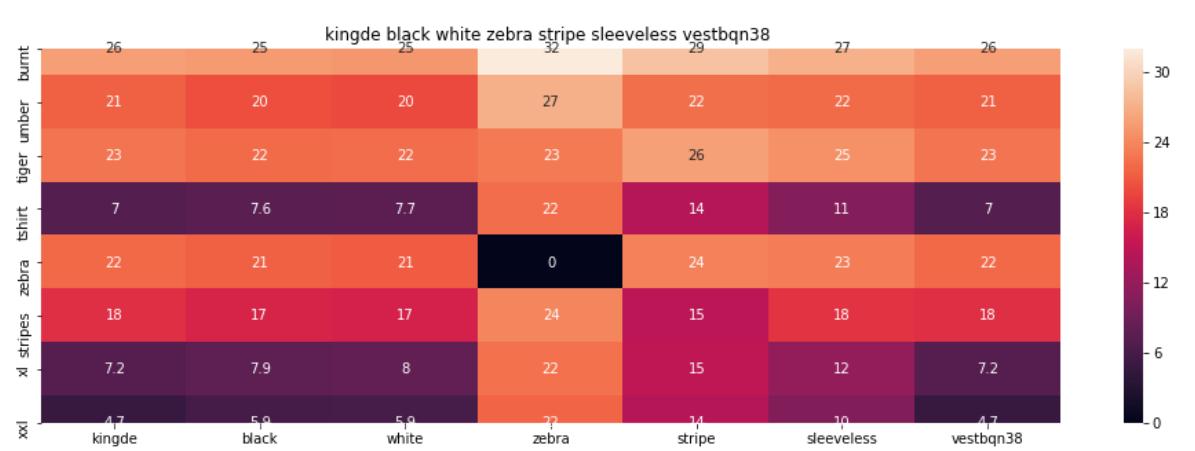
Brand : YICHUN

euclidean distance from input : 8.754642337744214

---



---



ASIN : B015H41F6G

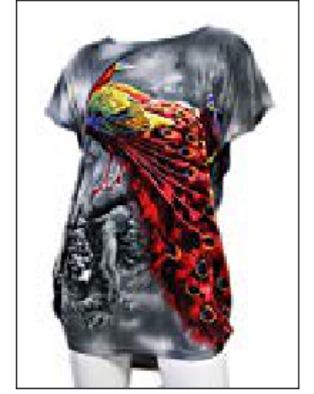
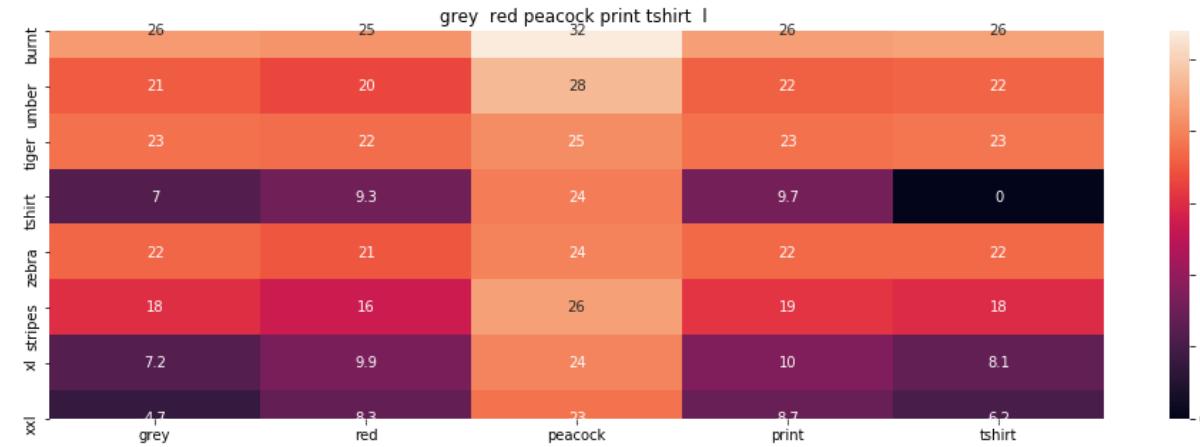
Brand : KINGDE

euclidean distance from input : 8.758145864704314

---



---



ASIN : B00JXQCFRS

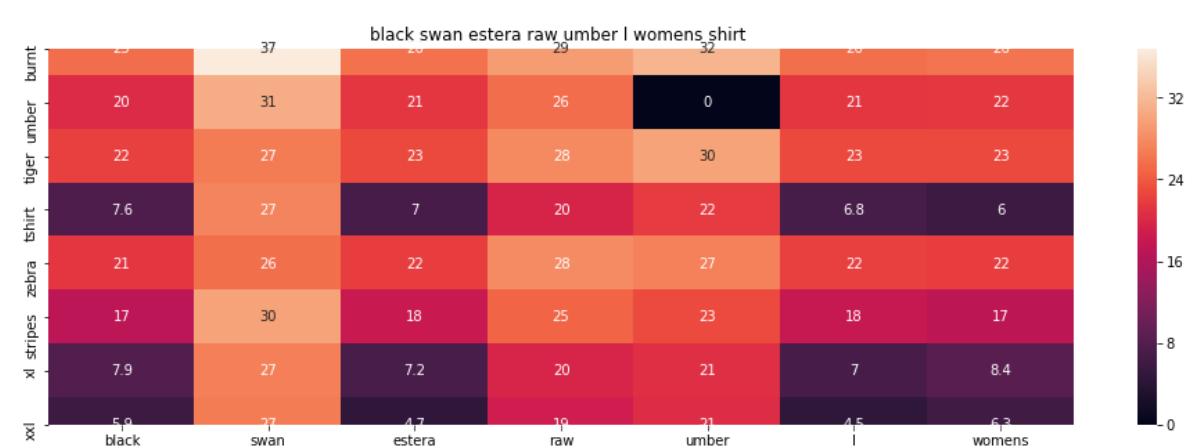
Brand : Si Row

euclidean distance from input : 8.77311123441825

---



---



ASIN : B06Y1VN8WQ

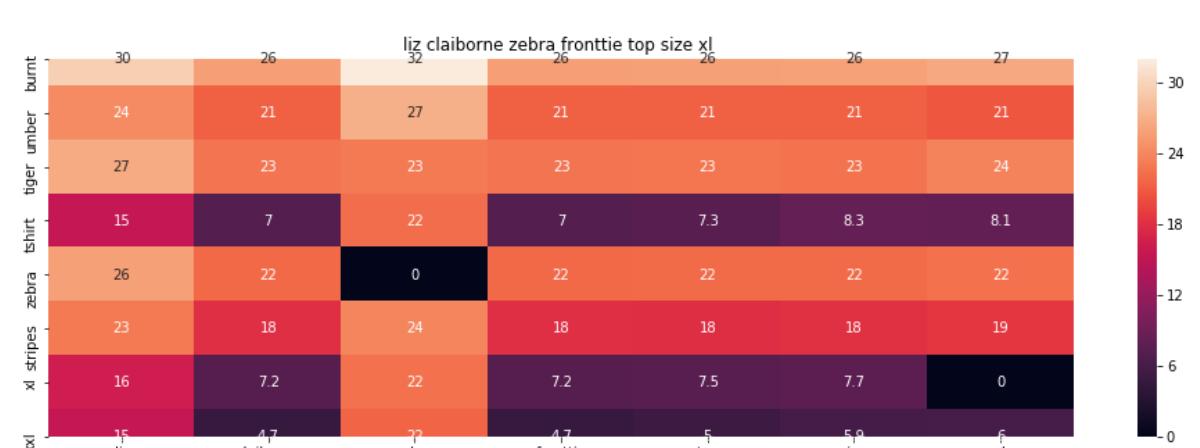
Brand : Black Swan

euclidean distance from input : 8.834639632059835

---



---

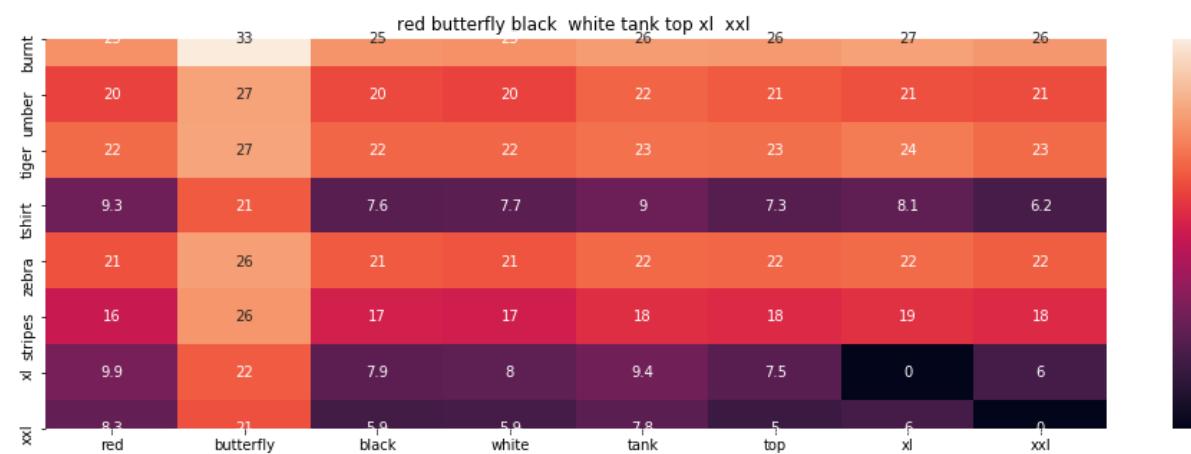


ASIN : B06XBY5QXL

Brand : Liz Claiborne

euclidean distance from input : 8.8444113156815

=====



ASIN : B00JV63CW2

Brand : Si Row

euclidean distance from input : 8.869242967903881

=====

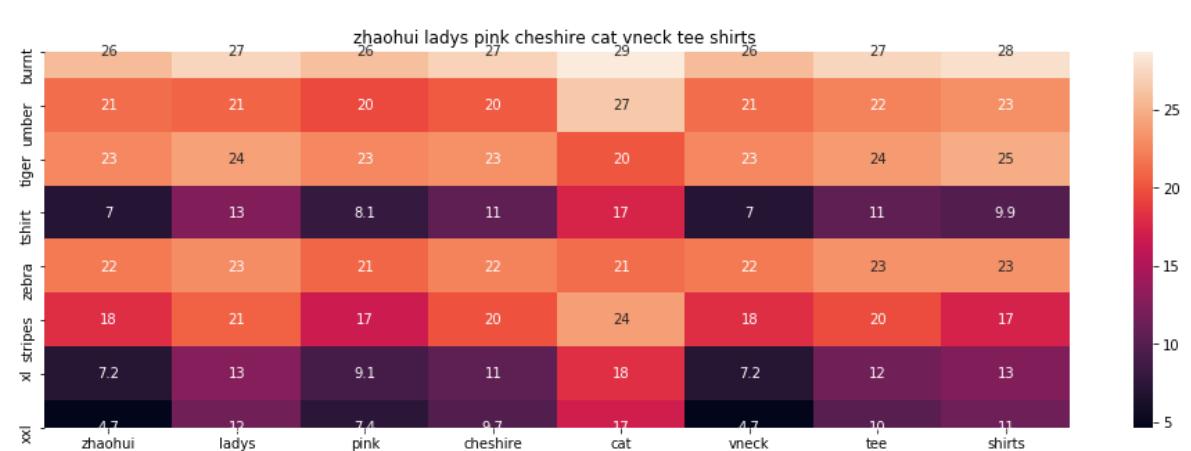


ASIN : B06Y2GTYPM

Brand : No Boundaries

euclidean distance from input : 8.87136208776575

=====

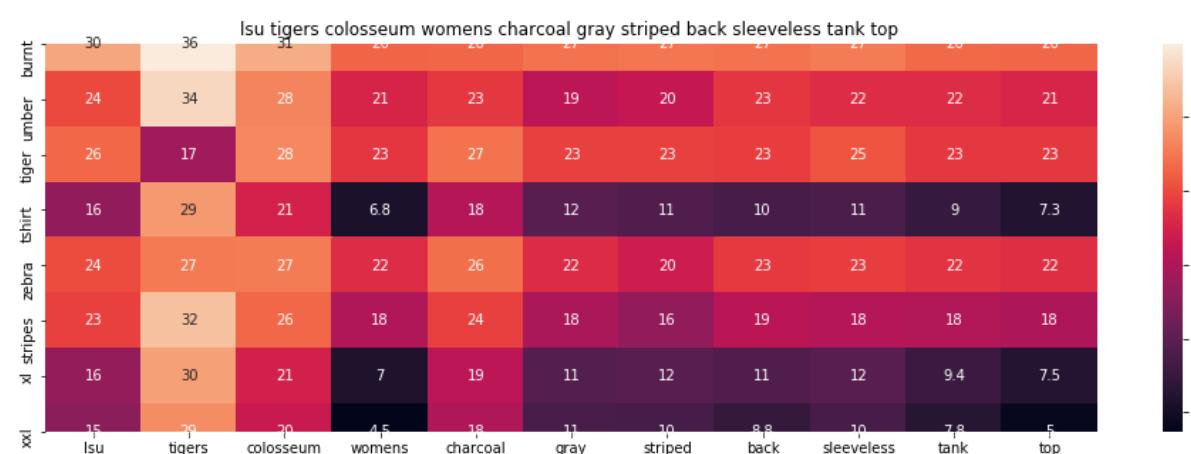


ASIN : B018DDFT4M

Brand : ZhaoHui

euclidean distance from input : 8.935352449089779

=====

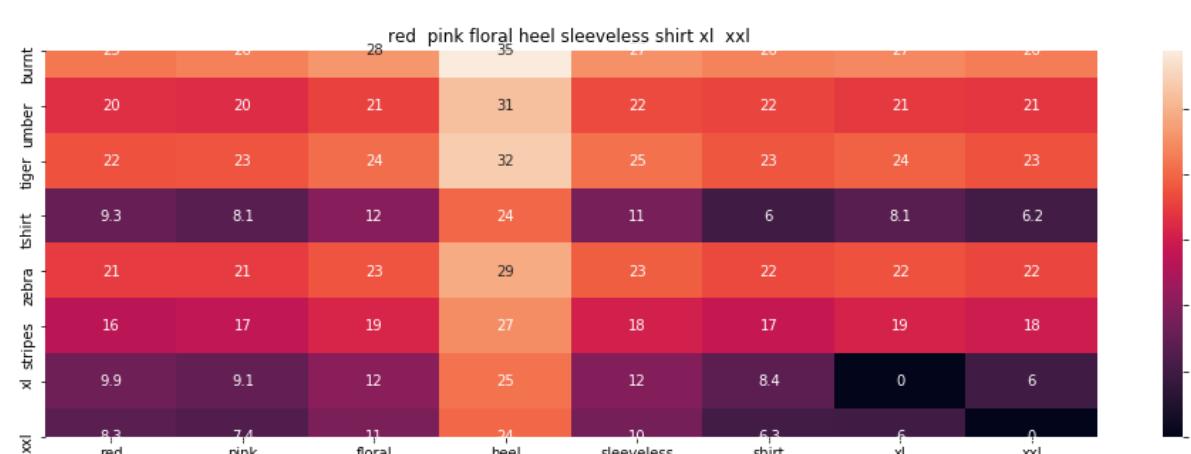


ASIN : B073R4ZM7Y

Brand : Colosseum

euclidean distance from input : 8.989712430081687

=====

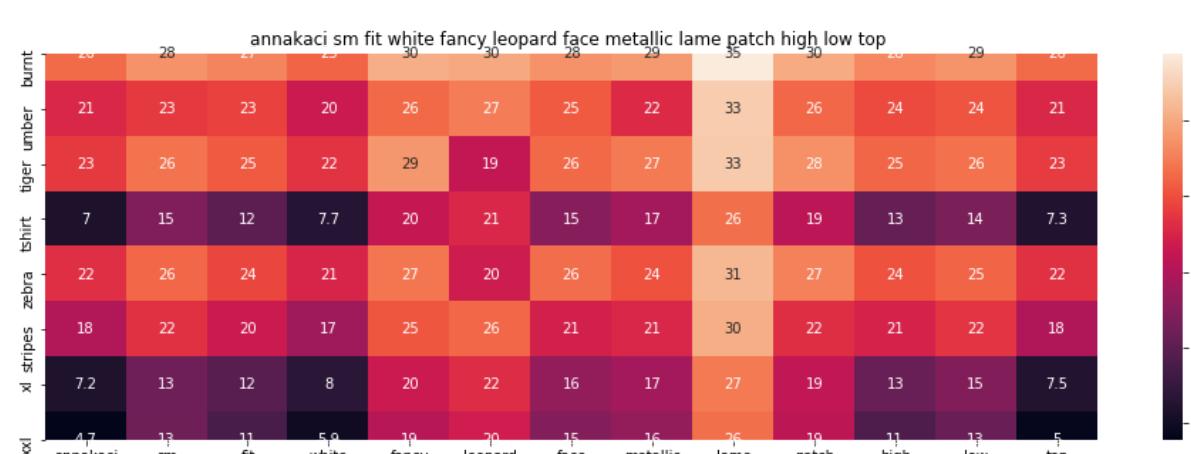


ASIN : B00JV63QOE

Brand : Si Row

euclidean distance from input : 9.002072906519953

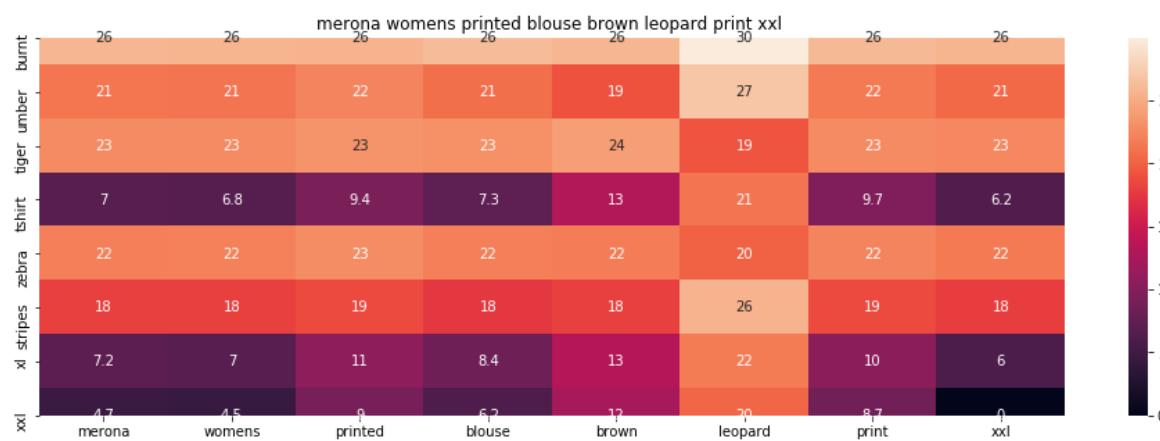
=====



ASIN : B00F4IJRES

Brand : Anna-Kaci

euclidean distance from input : 9.053692408970424



ASIN : B071YF3WDD

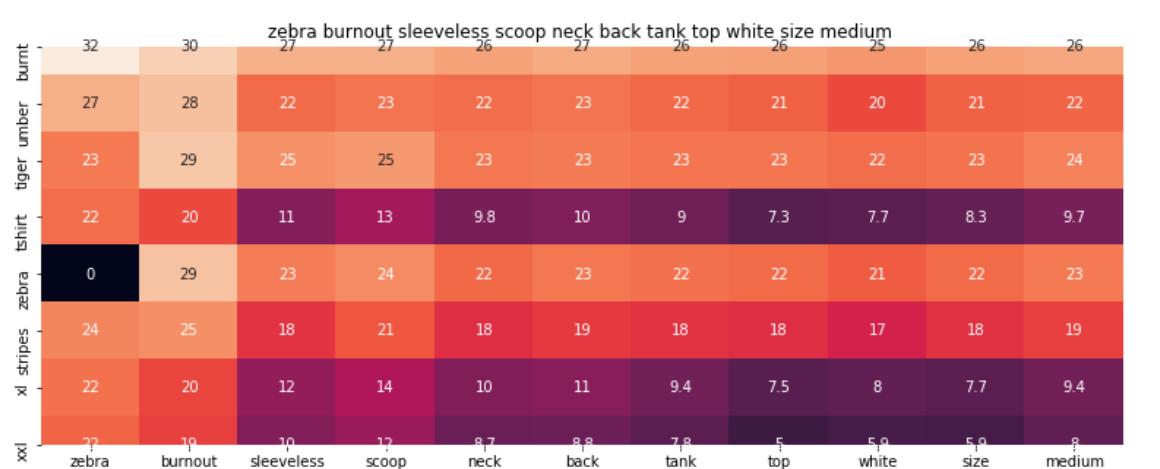
Brand : Merona

euclidean distance from input : 9.12290558237312

---



---



ASIN : B06XGYTZ49

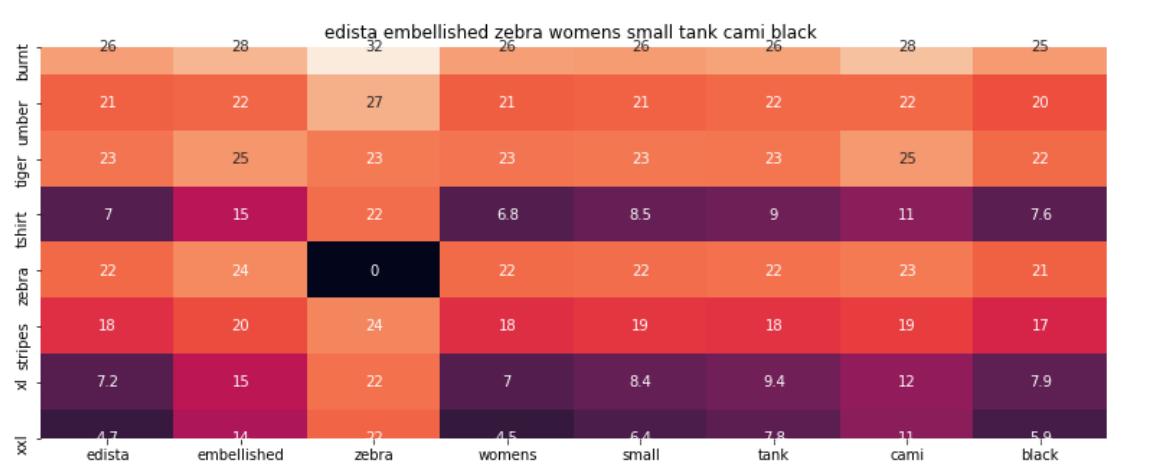
Brand : Concert

euclidean distance from input : 9.144039005224682

---



---



ASIN : B074P8MD22

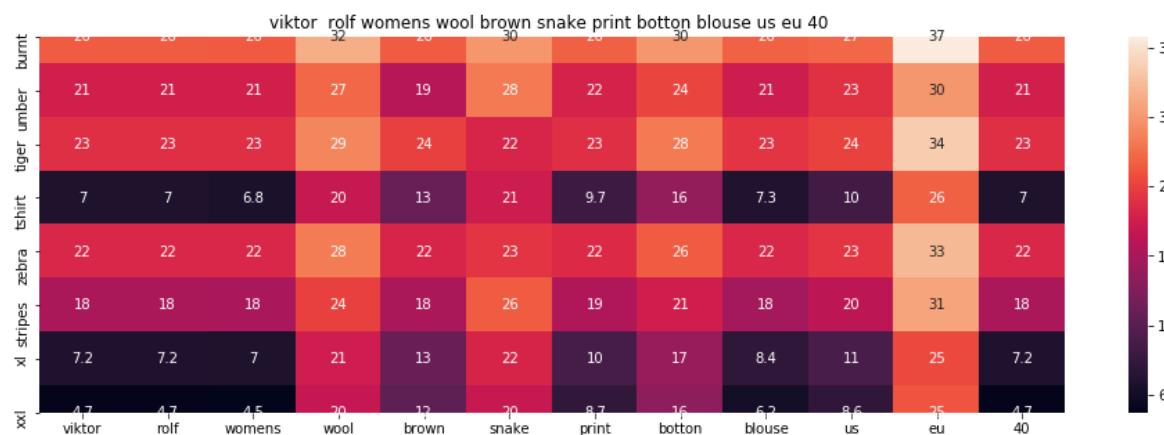
Brand : Edista

euclidean distance from input : 9.152785833576385

---



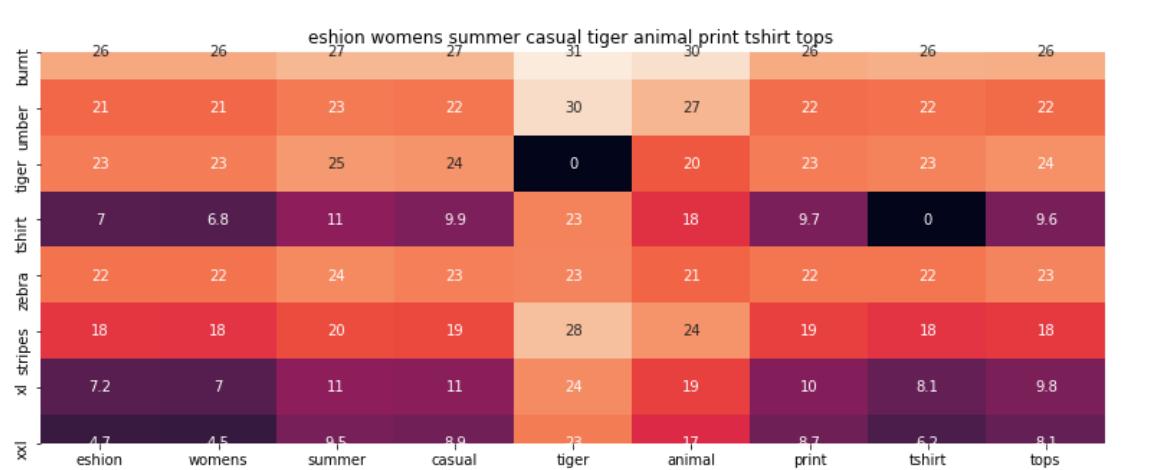
---



ASIN : B00LEHNVZ4

Brand : Viktor &amp; Rolf

euclidean distance from input : 9.153066526166299

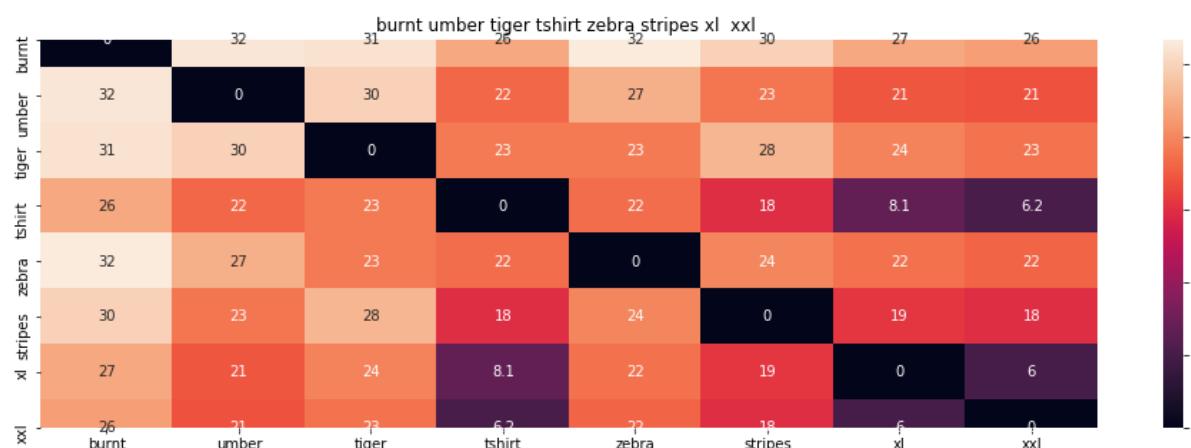


ASIN : B01EXZQR4K

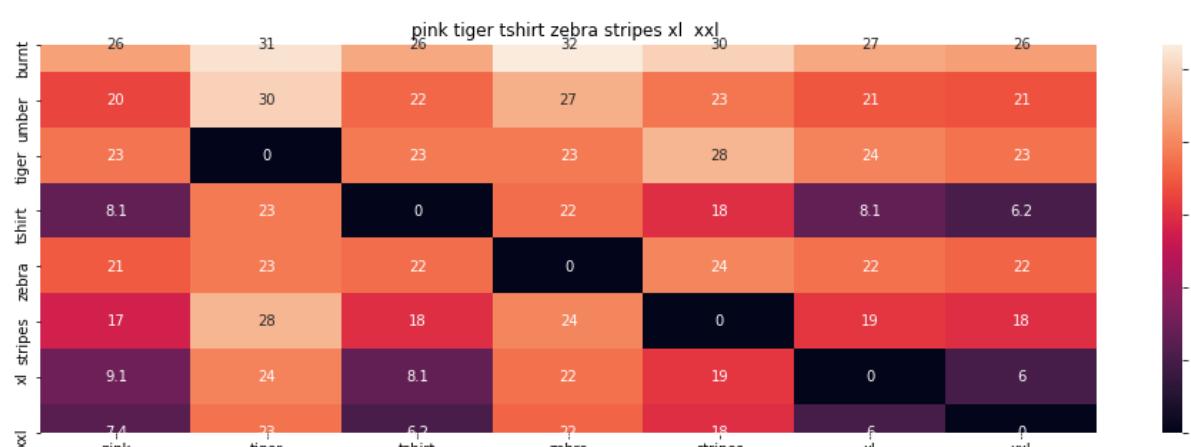
Brand : eshion

euclidean distance from input : 9.176497938344784

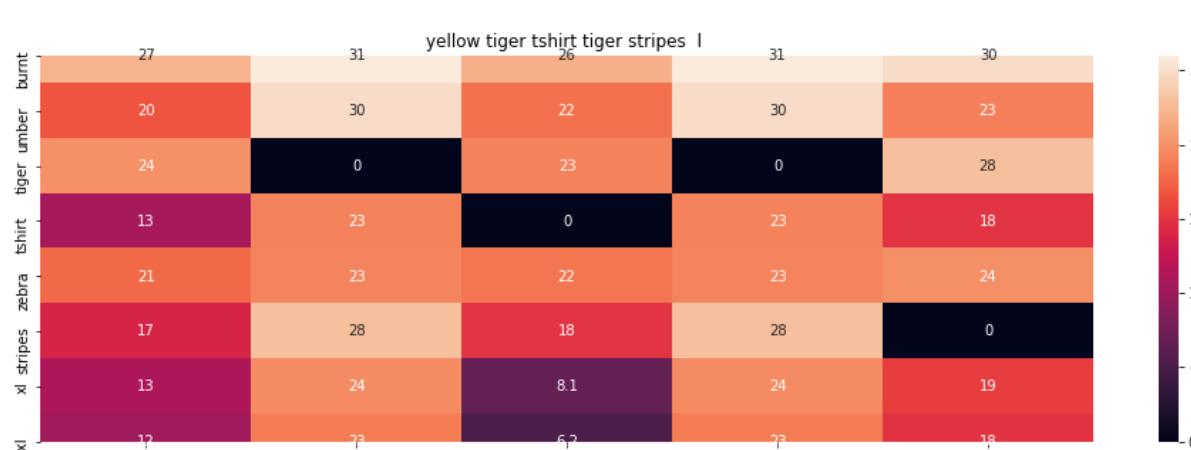
In [149]: `idf_w2v_bottelnack_brand(12566, 5, 1, 2, 20, 23)`



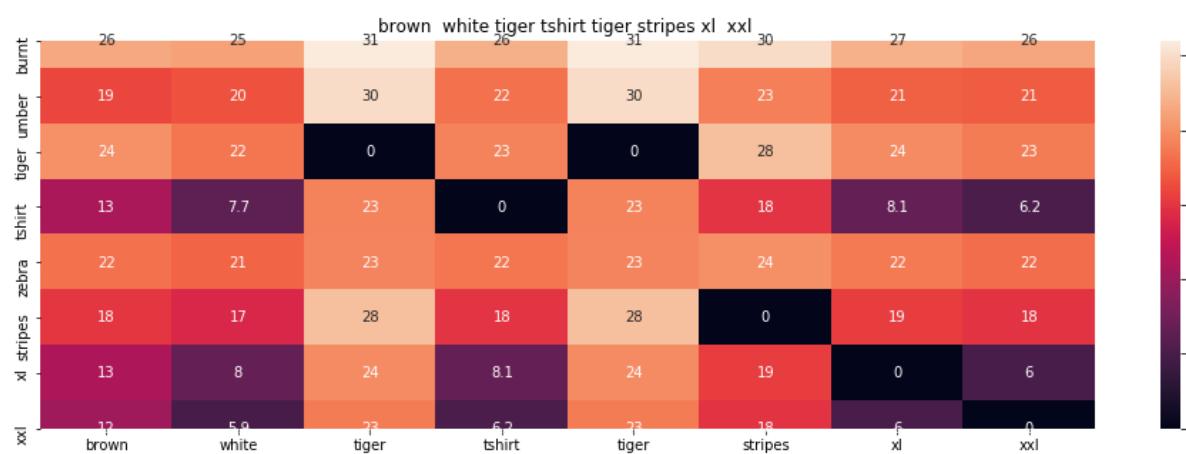
ASIN : B00JXQB5FQ  
 Brand : Si Row  
 euclidean distance from input : 4.518542742256873e-06  
 =====  
 =====



ASIN : B00JXQASS6  
 Brand : Si Row  
 euclidean distance from input : 22.24060945512155  
 =====  
 =====



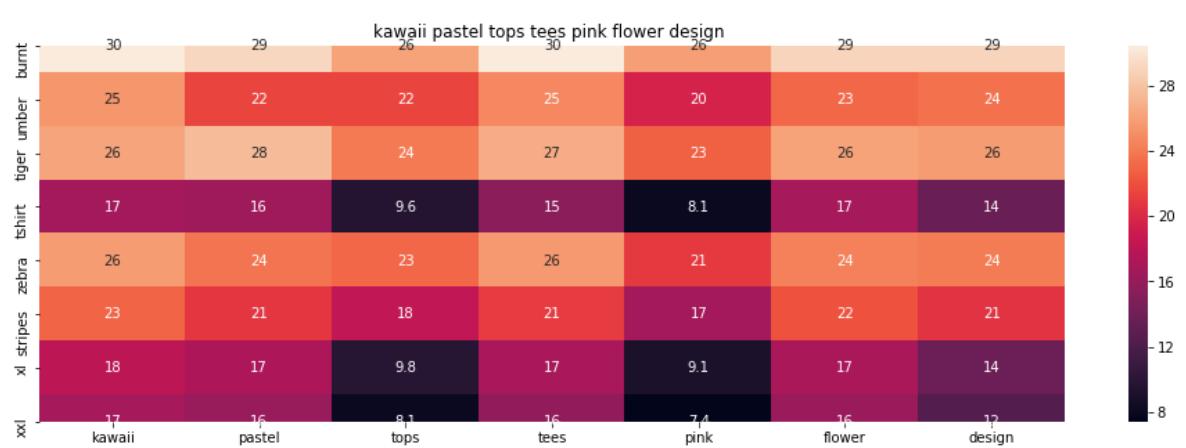
ASIN : B00JXQCUIC  
 Brand : Si Row  
 euclidean distance from input : 30.57513265611032  
 =====  
 =====



ASIN : B00JXQCWTO

Brand : Si Row

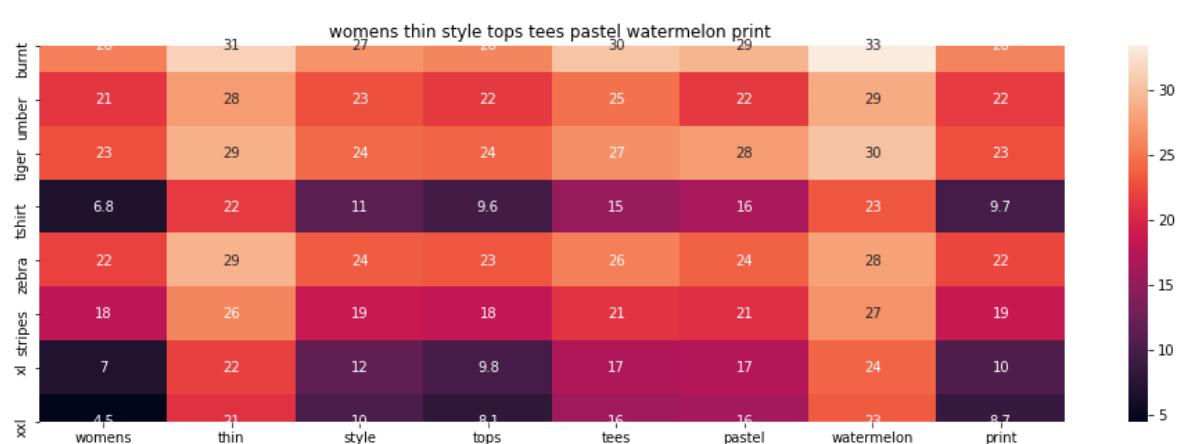
euclidean distance from input : 32.28063637869699



ASIN : B071FCWD97

Brand : Namnoi Clothing Store

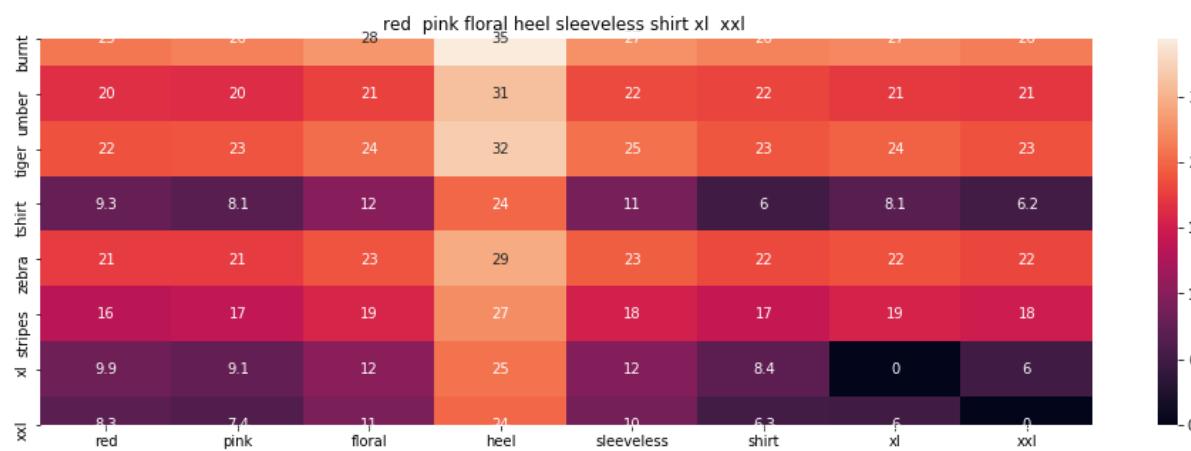
euclidean distance from input : 35.63795817398086



ASIN : B01JUNHBRM

Brand : Namnoi Clothing Store

euclidean distance from input : 35.760449193093585



ASIN : B00JV63QOE

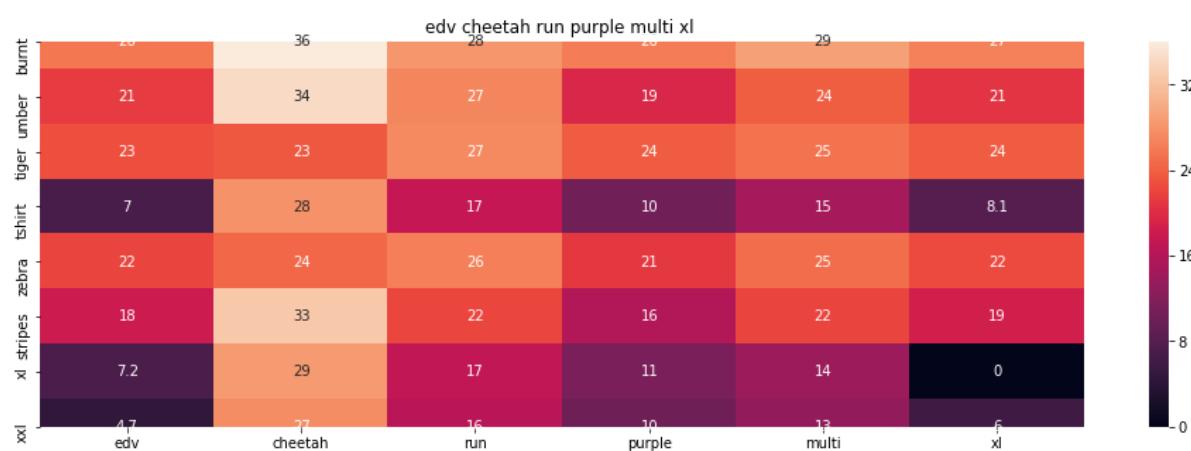
Brand : Si Row

euclidean distance from input : 35.82235249111921

---



---



ASIN : B01CUPYBM0

Brand : Styleco

euclidean distance from input : 35.92212928415008

---



---



ASIN : B071SBCY9W

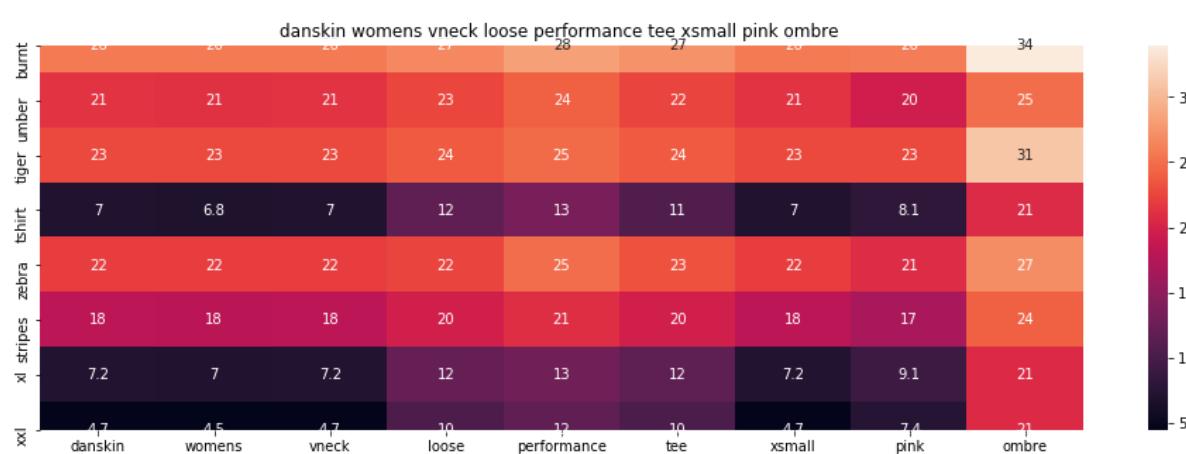
Brand : Namnoi Clothing Store

euclidean distance from input : 35.97082852114011

---



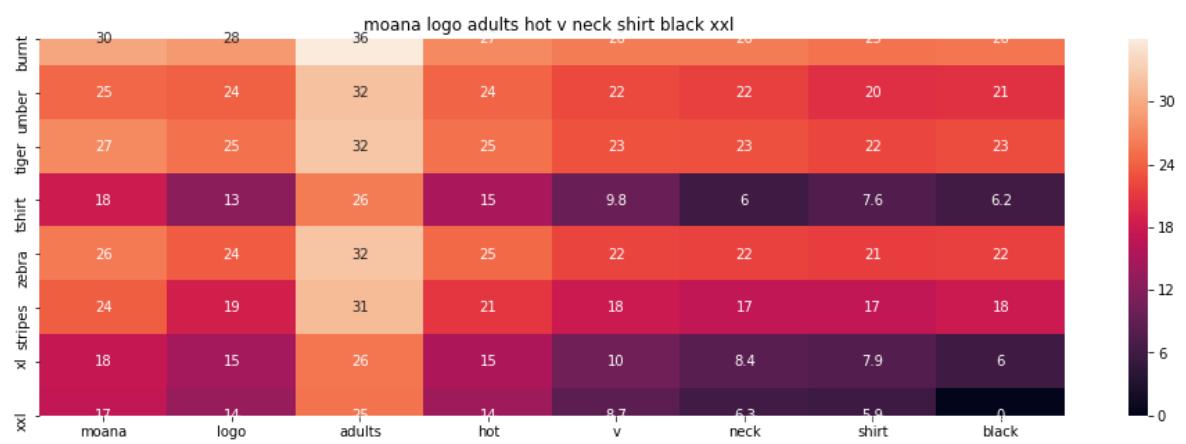
---



ASIN : B01F7PHXY8

Brand : Danskin Now

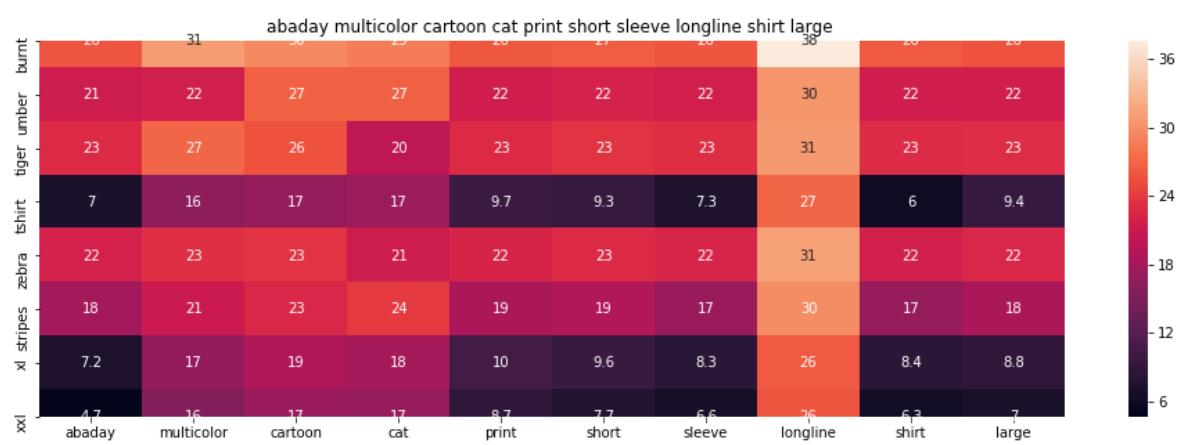
euclidean distance from input : 36.00439900042584



ASIN : B01LX6H43D

Brand : BOBOB

euclidean distance from input : 36.0122614525112



ASIN : B01CR57YY0

Brand : ABADAY

euclidean distance from input : 36.015800401660194

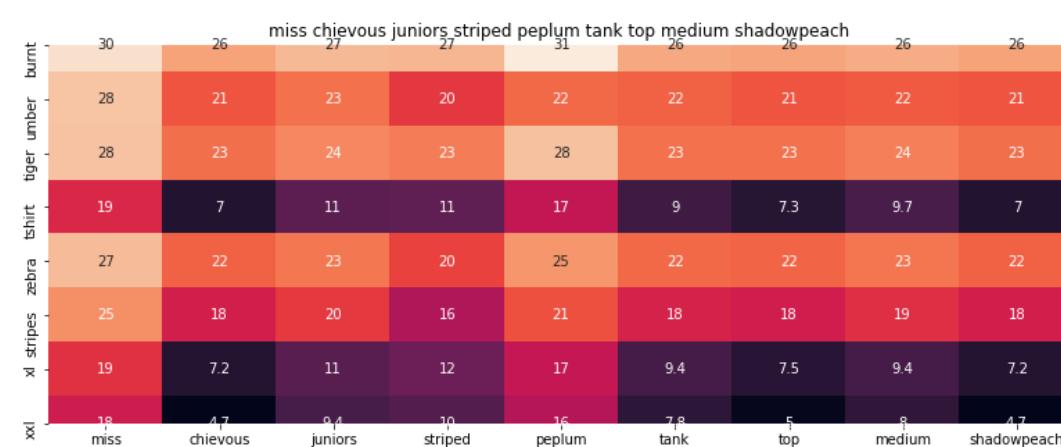


ASIN : B010NN9RXO

Brand : YICHUN

euclidean distance from input : 36.02774966836639

=====



ASIN : B0177DM70S

Brand : MISS CHIEVOUS/AROUND THE WORLD APP

euclidean distance from input : 36.10486935211985

=====



ASIN : B01I80A93G

Brand : Namnoi Clothing Store

euclidean distance from input : 36.172776823272855

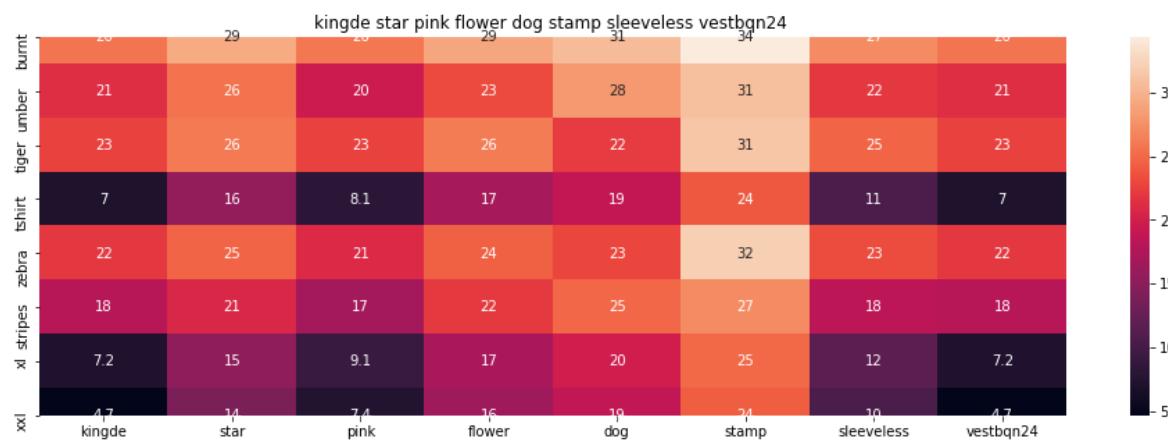
=====



ASIN : B071WYLBZS

Brand : Namnoi Clothing Store

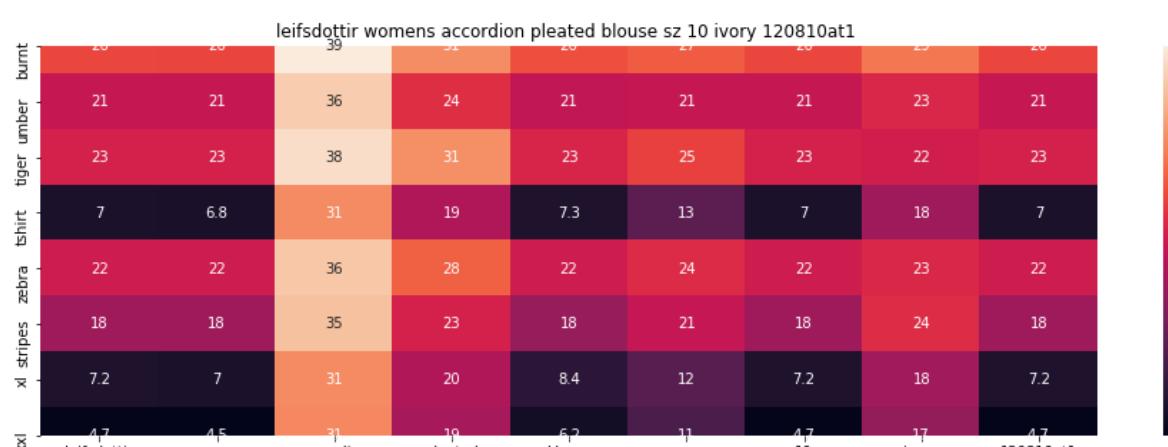
euclidean distance from input : 36.260781889190824



ASIN : B015H3W9BM

Brand : KINGDE

euclidean distance from input : 36.415438305146175



ASIN : B01B6E0DX2

Brand : Leifsdottir

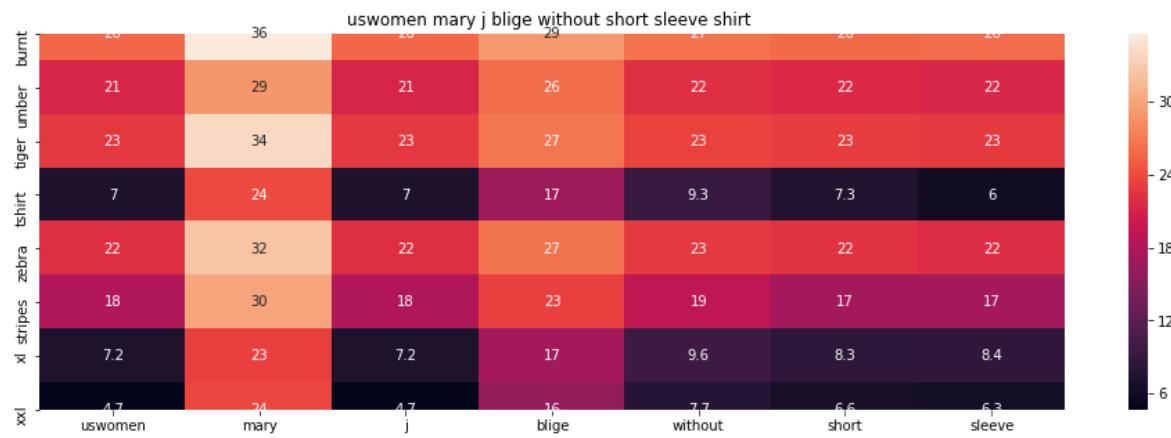
euclidean distance from input : 36.42271593009386



ASIN : B0734GRKZL

Brand : Namnoi Clothing Store

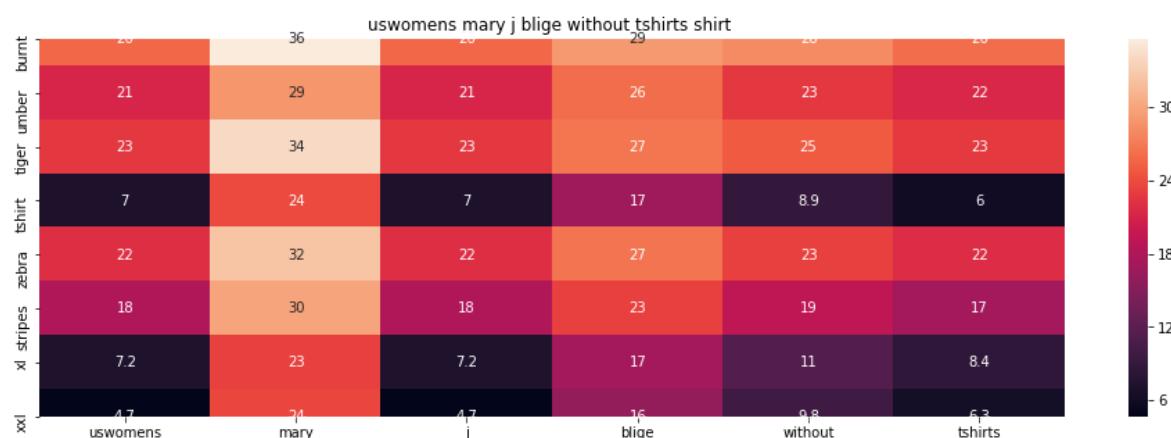
euclidean distance from input : 36.42969984433139



ASIN : B01LYNWX9M

Brand : South Dakota A.

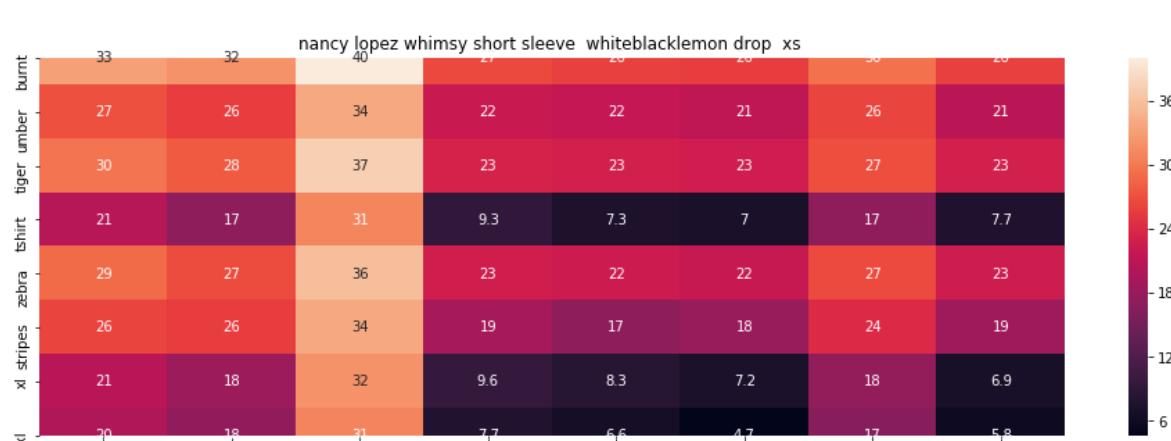
euclidean distance from input : 36.477841241019114



ASIN : B01M0XXFKK

Brand : South Dakota A.

euclidean distance from input : 36.49604892730713

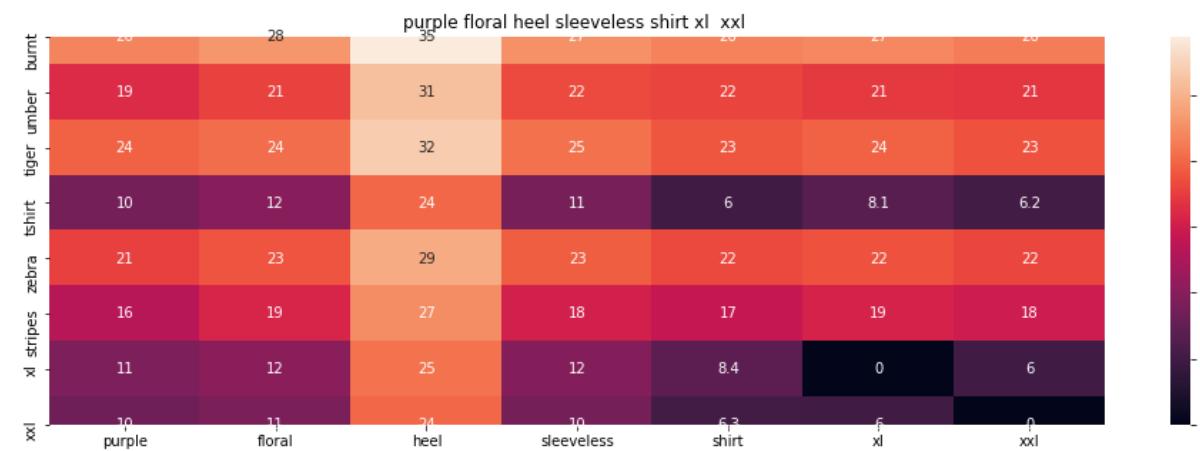


ASIN : B01MPX6IDX

Brand : Nancy Lopez

euclidean distance from input : 36.50816029571639

=====



ASIN : B00JV63VC8

Brand : Si Row

euclidean distance from input : 36.51779033117223

=====