Amazon Apparel Reccomendations

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]:

```
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 [5]:

```
data.columns
```

Out[5]:

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

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

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 apperal, ex: SHIRT/TSHIRT)
5. medium_image_url (url of the image)
6. title (title of the product.)
7. formatted_price (price of the product)

In [6]:

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

In [7]:

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

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

Out[7]:

	asin	brand	color	medium_image_url	product_type_name	
0	B016I2TS4W	FNC7C	None	https://images-na.ssl-images-amazon.com/images...	SHIRT	Minior Como Super Ironm Long R...
1	B01N49AI08	FIG Clothing	None	https://images-na.ssl-images-amazon.com/images...	SHIRT	FIG C Wome Tunic
2	B01JDPCOHO	FIG Clothing	None	https://images-na.ssl-images-amazon.com/images...	SHIRT	FIG C Wome Won
3	B01N19U5H5	Focal18	None	https://images-na.ssl-images-amazon.com/images...	SHIRT	Focal Sailor Bubbl Sleev Blous
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT	Feath Ladie Sleev Resis

Missing data for various features:

Basic stats for the faetures : product_type_name

In [10]:

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

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

In [11]:

```
# name of different prodct 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 [18]:

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

Out[18]:

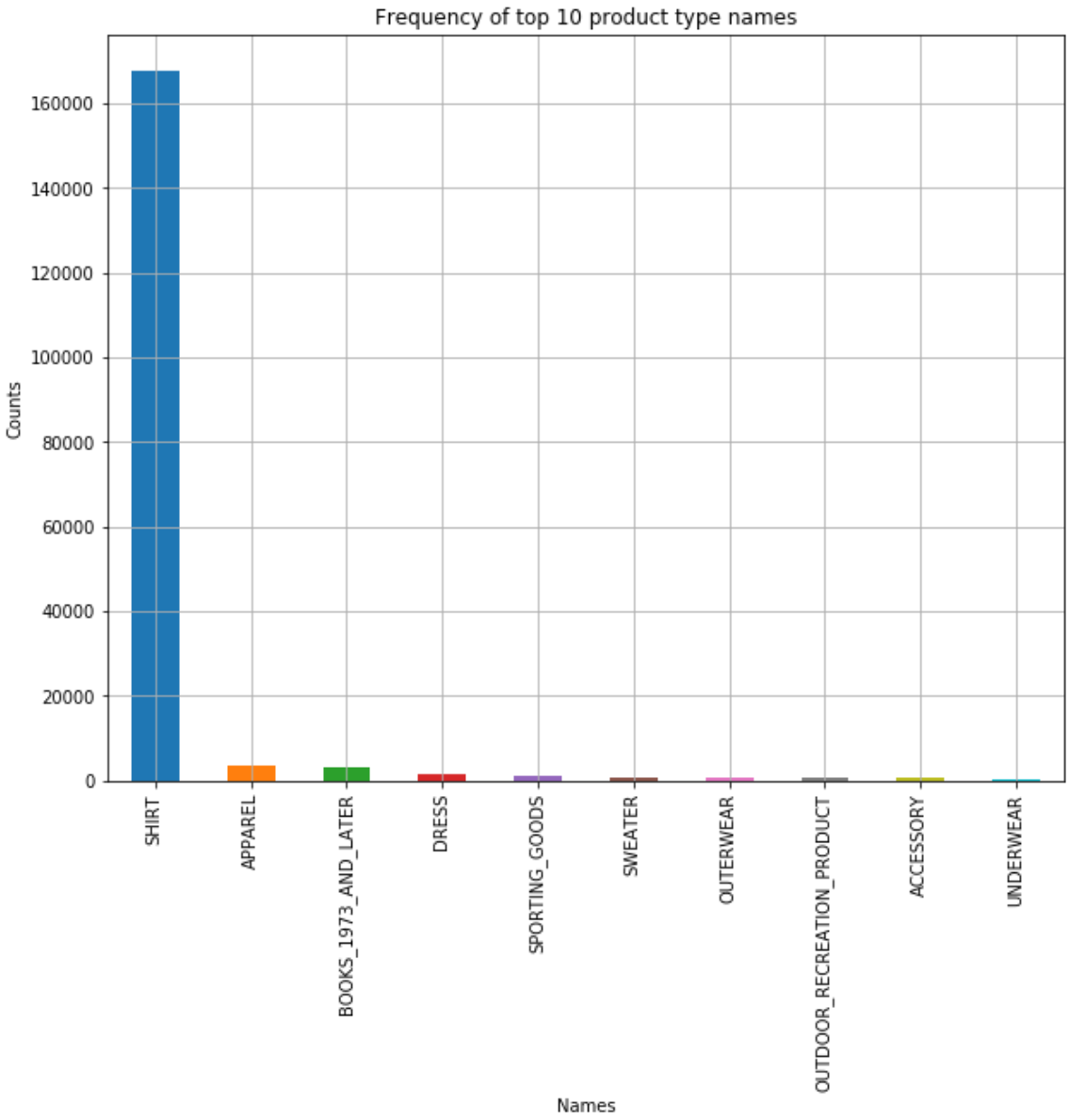
```
[('SHIRT', 167794),
 ('APPAREL', 3549),
 ('BOOKS_1973_AND_LATER', 3336),
 ('DRESS', 1584),
 ('SPORTING_GOODS', 1281),
 ('SWEATER', 837),
 ('OUTERWEAR', 796),
 ('OUTDOOR_RECREATION_PRODUCT', 729),
 ('ACCESSORY', 636),
 ('UNDERWEAR', 425)]
```

In [28]:

```
sorted_names = data['product_type_name'].value_counts()
```

In [46]:

```
plt.figure(figsize=(10, 8))
sorted_names.head(10).plot(kind='bar')
plt.title('Frequency of top 10 product type names')
plt.xlabel('Names')
plt.ylabel('Counts')
plt.grid()
plt.show()
```

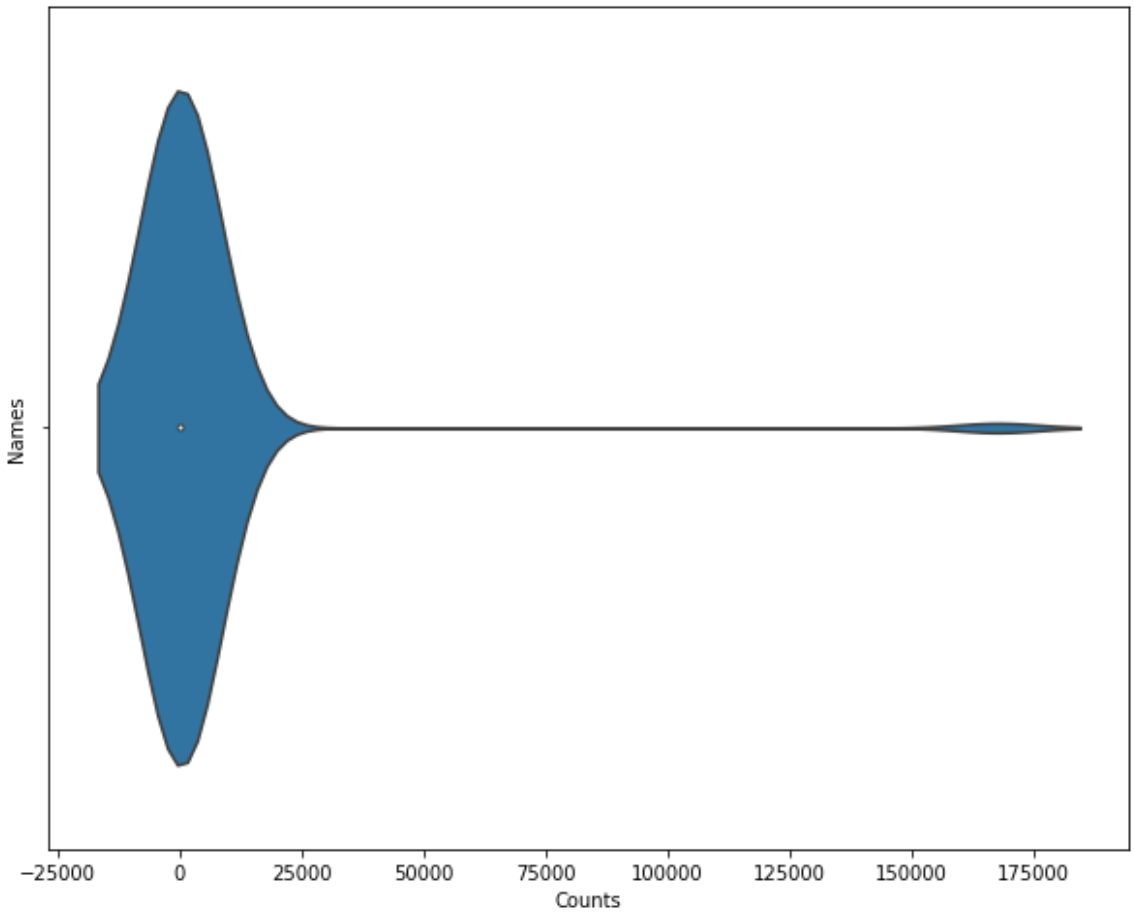


In [52]:

```
plt.figure(figsize=(10,8))
sns.violinplot(sorted_names)
plt.ylabel('Names')
plt.xlabel('Counts')
```

Out[52]:

```
Text(0.5,0,'Counts')
```

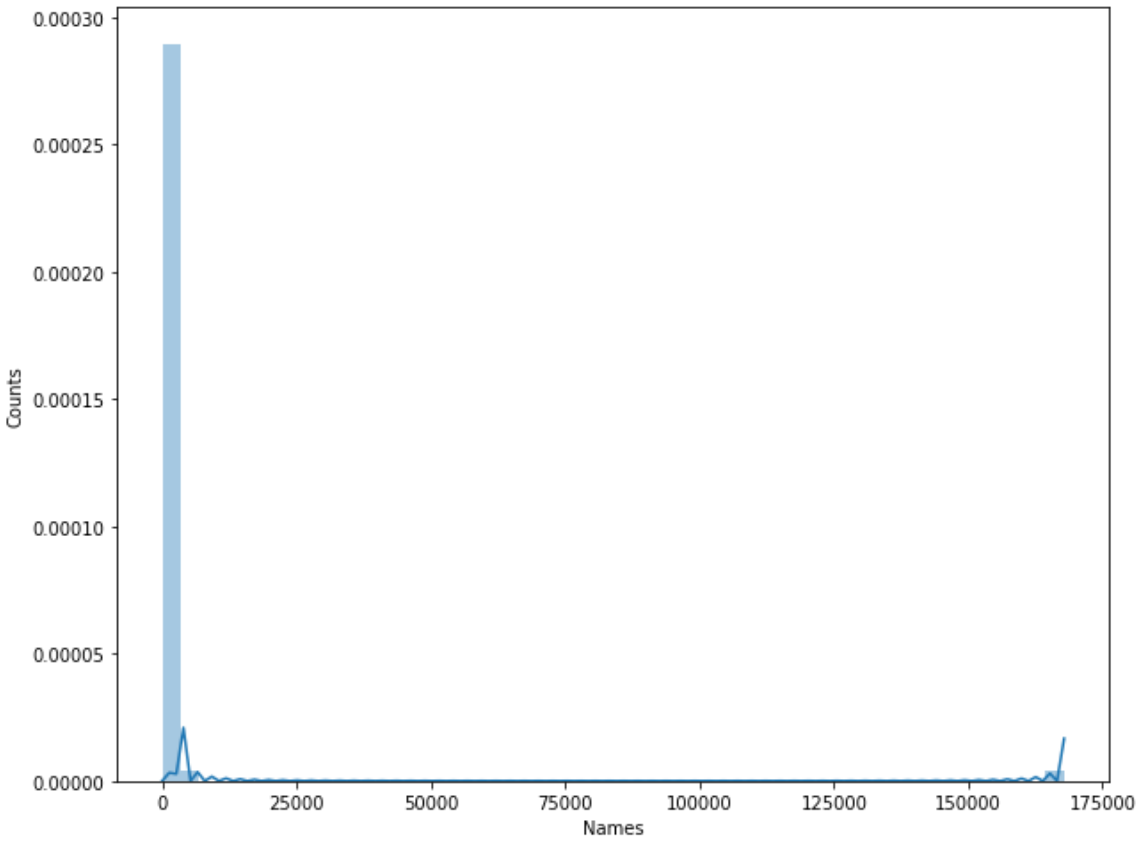


In [53]:

```
plt.figure(figsize=(10,8))
sns.distplot(sorted_names)
plt.xlabel('Names')
plt.ylabel('Counts')
```

Out[53]:

```
Text(0,0.5,'Counts')
```



In [60]:

```
from wordcloud import WordCloud

wordcloud = WordCloud(background_color='black',
                          width=1600,
                          height=800,
                      ).generate_from_frequencies(sorted_names)
fig = plt.figure(figsize=(30,20))
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
fig.savefig("tag.png")
plt.show()
```

Basic stats for feature: brand

In [62]:

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

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

In [63]:

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

Out[63]:

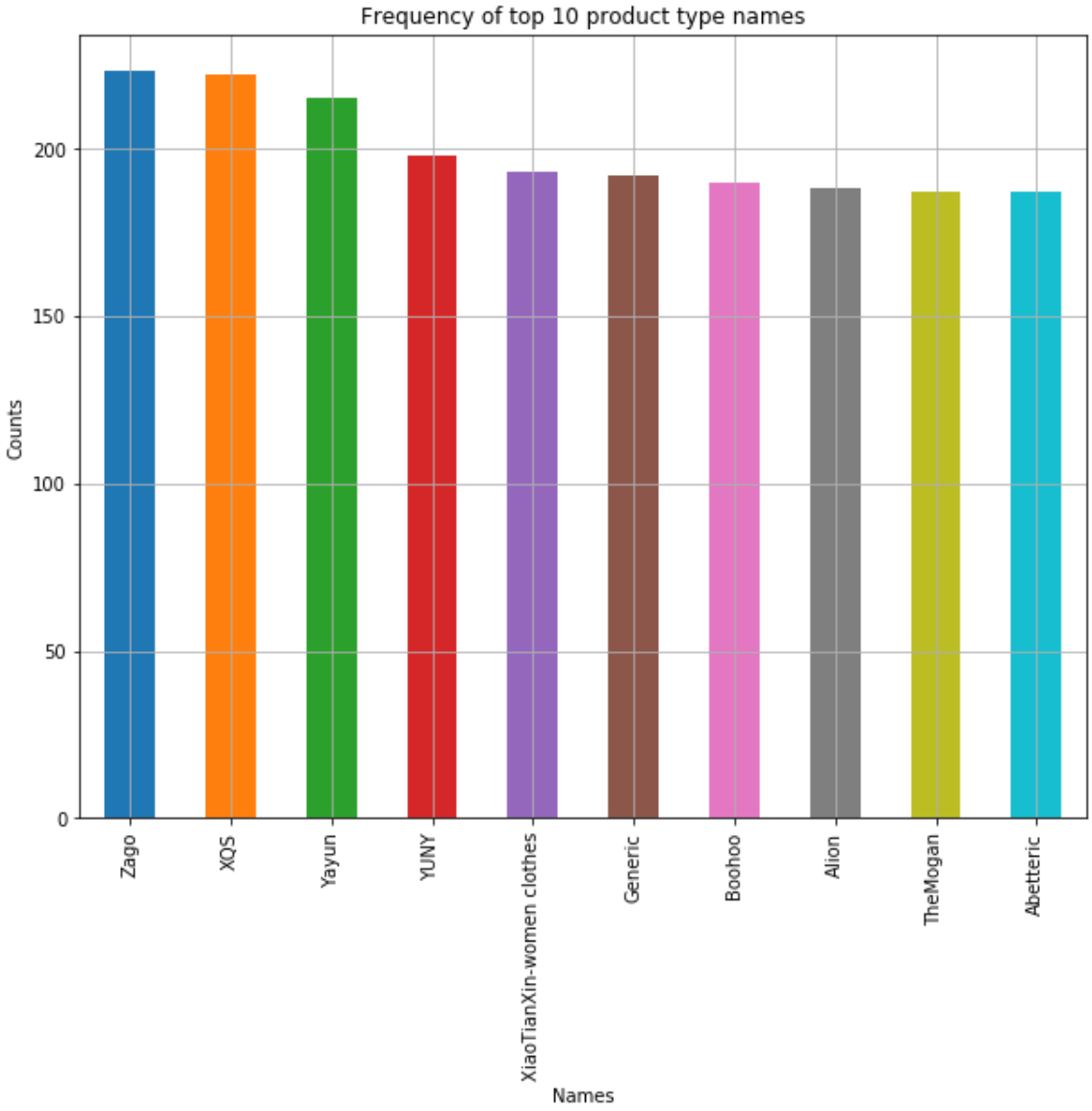
```
[('Zago', 223),
 ('XQS', 222),
 ('Yayun', 215),
 ('YUNY', 198),
 ('XiaoTianXin-women clothes', 193),
 ('Generic', 192),
 ('Boohoo', 190),
 ('Alion', 188),
 ('Abetteric', 187),
 ('TheMogan', 187)]
```

In [64]:

```
sorted_brand = data['brand'].value_counts()
```

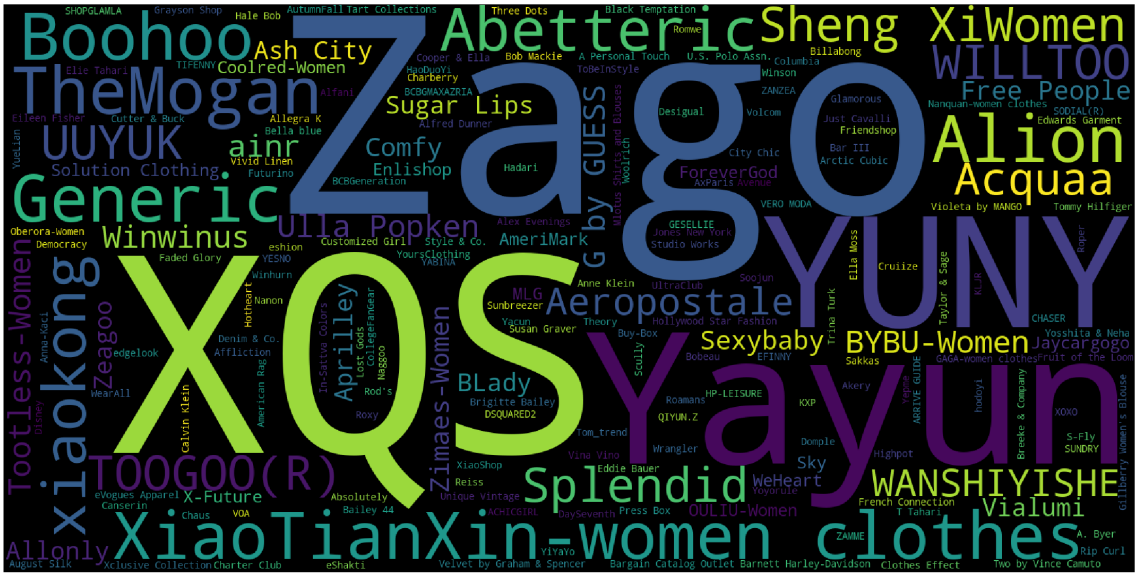
In [65]:

```
plt.figure(figsize=(10, 8))
sorted_brand.head(10).plot(kind='bar')
plt.title('Frequency of top 10 brand names')
plt.xlabel('Names')
plt.ylabel('Counts')
plt.grid()
plt.show()
```


Frequency of top 10 product type names
Counts
0
50
100
150
200
Zago
XQS
Yayun
YUNY
XiaoTianXin-women clothes
Generic
Boohoo
Alion
TheMogan
Abetteric
Names

In [66]:

```
wordcloud = WordCloud(background_color='black',
                            width=1600,
                            height=800,
                      ).generate_from_frequencies(sorted_brand)
fig = plt.figure(figsize=(30,20))
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
fig.savefig("brand.png")
plt.show()
```



Basic stats for feature: color

In [67]:

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

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

In [68]:

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

Out[68]:

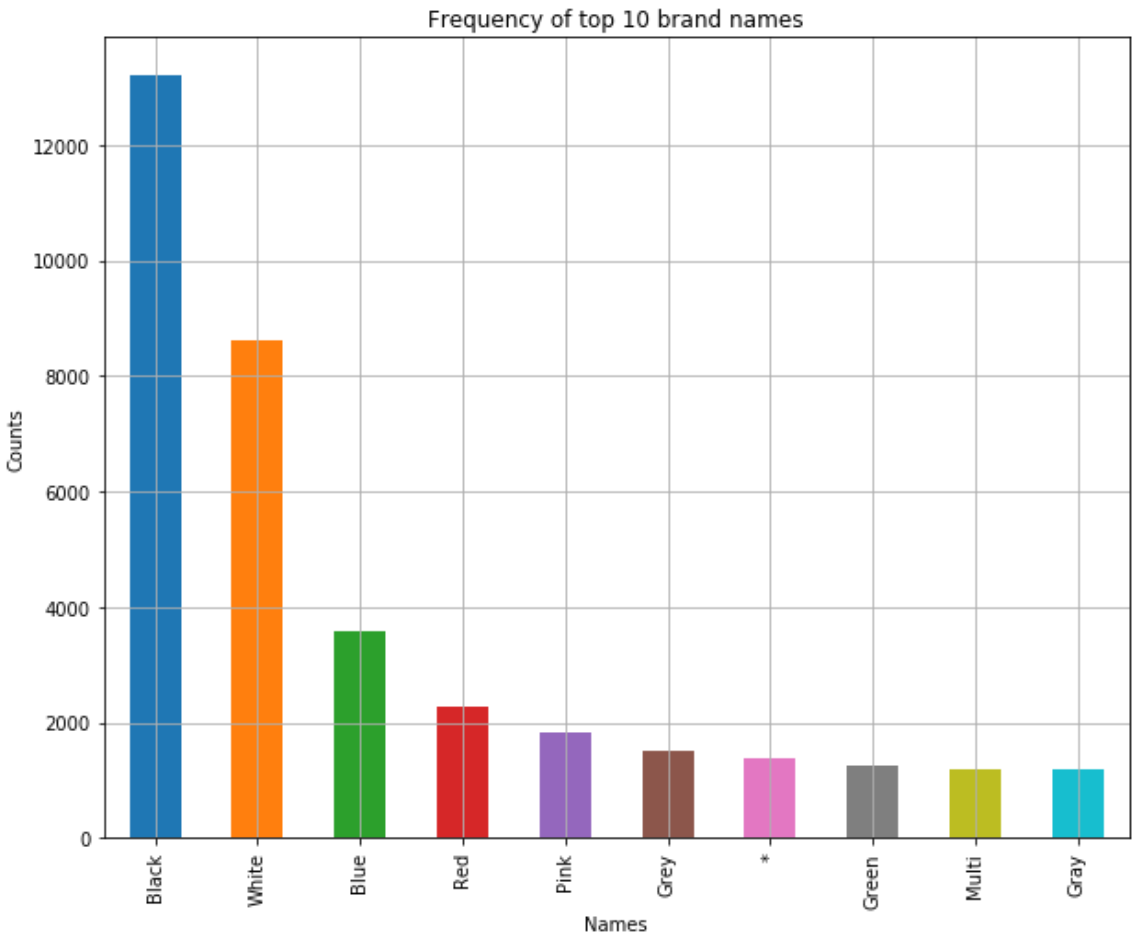
```
[(None, 118182),
 ('Black', 13207),
 ('White', 8616),
 ('Blue', 3570),
 ('Red', 2289),
 ('Pink', 1842),
 ('Grey', 1499),
 ('*', 1388),
 ('Green', 1258),
 ('Multi', 1203)]
```

In [71]:

```
sorted_color = data['color'].value_counts()
```

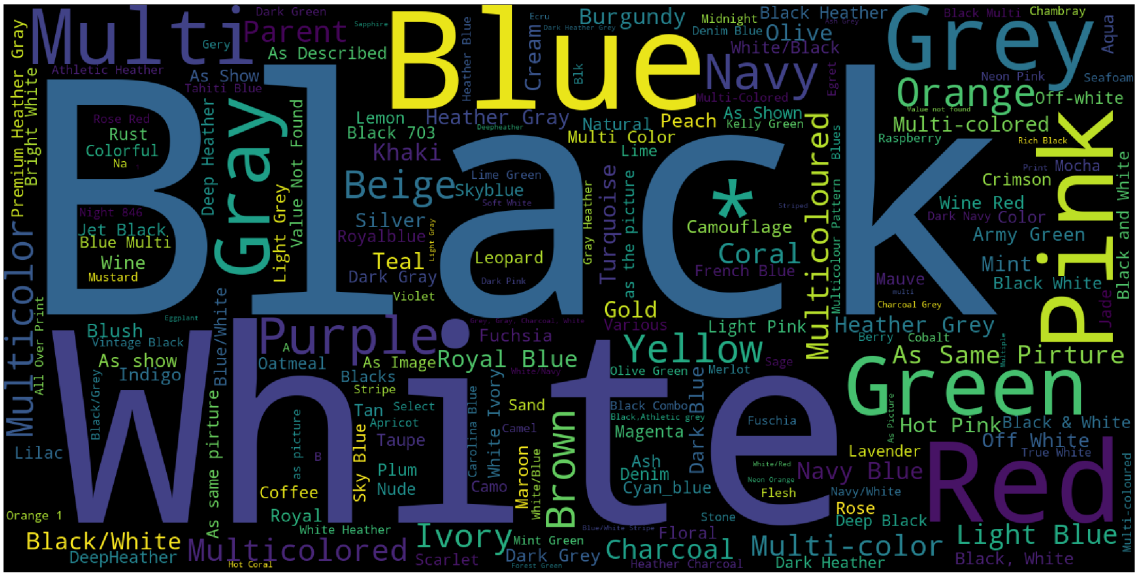
In [72]:

```
plt.figure(figsize=(10, 8))
sorted_color.head(10).plot(kind='bar')
plt.title('Frequency of top 10 brand names')
plt.xlabel('Names')
plt.ylabel('Counts')
plt.grid()
plt.show()
```



In [73]:

```
wordcloud = WordCloud(background_color='black',
                          width=1600,
                          height=800,
                      ).generate_from_frequencies(sorted_color)
fig = plt.figure(figsize=(30,20))
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
fig.savefig("brand.png")
plt.show()
```



Basic stats for feature: formatted_price

In [75]:

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

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

In [76]:

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

Out[76]:

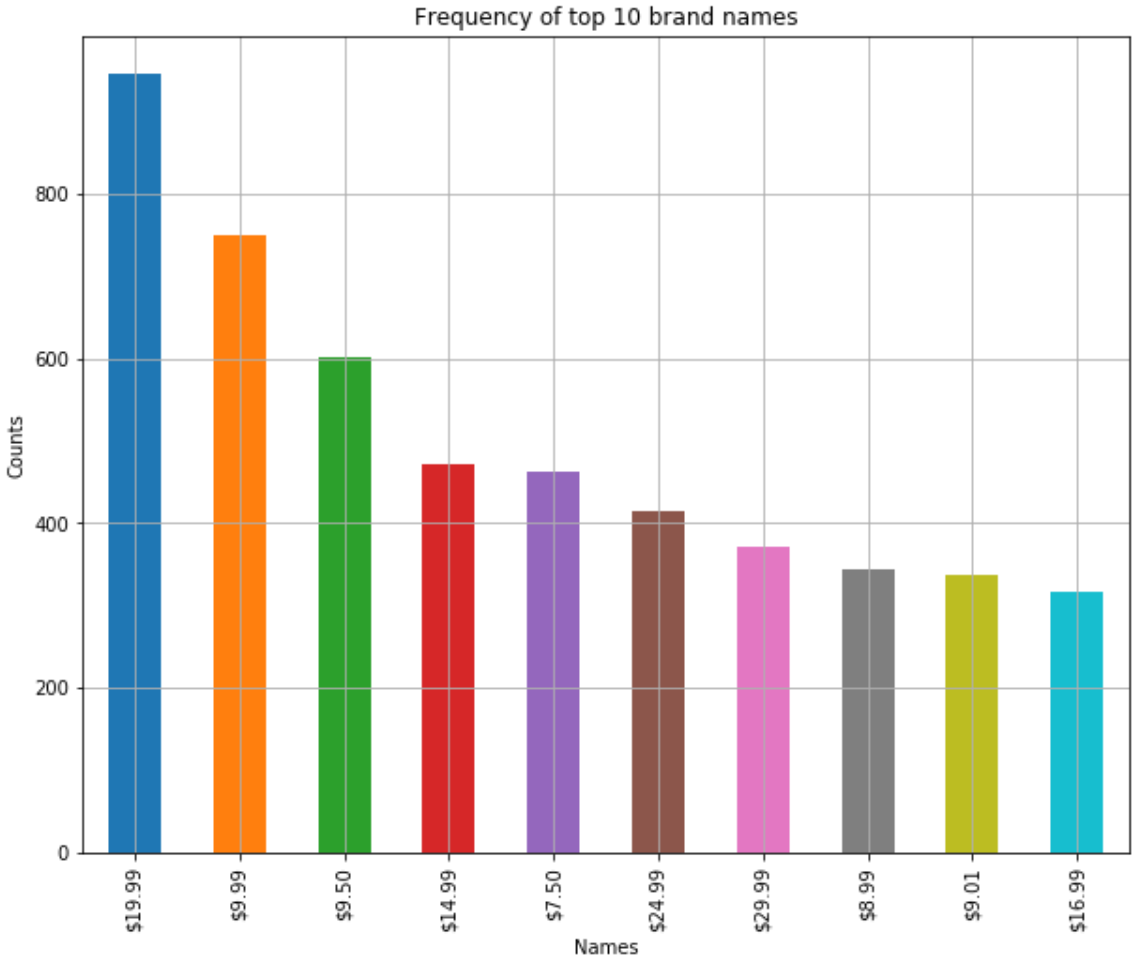
```
[(None, 154743),
 ('$19.99', 945),
 ('$9.99', 749),
 ('$9.50', 601),
 ('$14.99', 472),
 ('$7.50', 463),
 ('$24.99', 414),
 ('$29.99', 370),
 ('$8.99', 343),
 ('$9.01', 336)]
```

In [77]:

```
sorted_price = data['formatted_price'].value_counts()
```

In [78]:

```
plt.figure(figsize=(10, 8))
sorted_price.head(10).plot(kind='bar')
plt.title('Frequency of top 10 brand names')
plt.xlabel('Names')
plt.ylabel('Counts')
plt.grid()
plt.show()
```

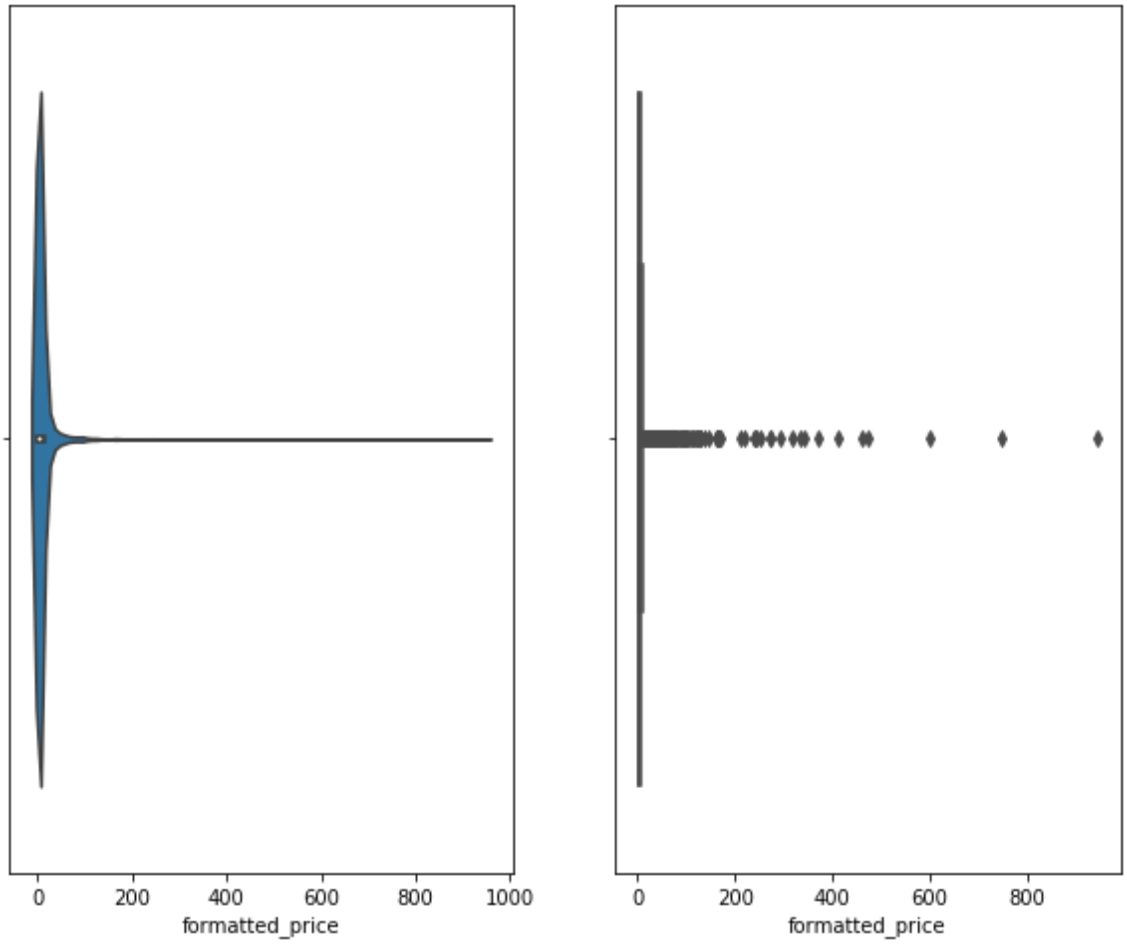


In [84]:

```
plt.figure(figsize=(10, 8))

plt.subplot(1,2,1)
sns.violinplot(sorted_price , )

plt.subplot(1,2,2)
sns.boxplot(sorted_price, palette='gist_rainbow')
```

Out[84]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x17f60fdf28>
```



Basic stats for feature: title

In [85]:

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

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

In [86]:

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

We save data files at every major step in our processing in "pickle" files.

In [87]:

```
# 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 od data after eliminating price', data.shape)
```

```
NUmber od data after eliminating price (28395, 7)
```

In [89]:

```
# eliminate the data that have color = null
data = data.loc[~data['color'].isnull()]
print('After elimination', data.shape)
```

```
After elimination (28385, 7)
```

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

In [90]:

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

In [91]:

```
# 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[91]:

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

Remove near duplicate items:

In [3]:

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

# find number of products that have duplicate titlea
print(sum(data.duplicated('title')))
```

2325

Shirts of same size

 :B00AQ4GMCK	 :B00AQ4GMTS
 :B00AQ4GMLQ	 :B00AQ4GN3I

 :B00G278GZ6	 :B00G278W6O
 :B00G278Z2A	 :B00G2786X8

Remove Duplicates:

In [94]:

```
data.head()
```

Out[94]:

	asin	brand	color	medium_image_url	product_type_name	
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT	Featl Ladie Slee Resi
6	B012YX2ZPI	HX-Kingdom Fashion T-shirts	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	Wom Uniq 100% T - S Olym
11	B001LOUGE4	Fitness Etc.	Black	https://images-na.ssl-images-amazon.com/images...	SHIRT	Ladie Cotto 2x1 R Tank
15	B003BSRPB0	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	Featl Ladie Mois Free Spor
21	B014ICEDNA	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT	Supe Chib Dear Cast Shor

In [4]:

```
# remove all the titles with very few title length

data_sorted = data[data['title'].apply(lambda x: len(x.split())>4)]
print("After removal", data_sorted.shape)
```

```
After removal (27949, 7)
```

In [5]:

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

Out[5]:

	asin	brand	color	medium_image_url	product_type_name
61973	B06Y1KZ2WB	Éclair	Black/Pink	https://images-na.ssl-images-amazon.com/images...	SHIRT
133820	B010RV33VE	xiaoming	Pink	https://images-na.ssl-images-amazon.com/images...	SHIRT
81461	B01DDSDLNS	xiaoming	White	https://images-na.ssl-images-amazon.com/images...	SHIRT
75995	B00X5LYO9Y	xiaoming	Red Anchors	https://images-na.ssl-images-amazon.com/images...	SHIRT
151570	B00WPJG35K	xiaoming	White	https://images-na.ssl-images-amazon.com/images...	SHIRT

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

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

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

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

In [6]:

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

In [8]:

```
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's', 'Shirt', 'X-Large']
    a = data['title'].loc[indices[i]].split()

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

        #store the list of words of jth string in b
        b = data['title'].loc[indices[j]].split()

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

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

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

        if(length - count) > 2:

            stage1_dedupe_asins.append(data_sorted['asin'].loc[indices[i]])

            if j == num_data_points-1: stage1_dedupe_asins.append(data_sorted['asin'].l
oc[indices[j]])

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

In [9]:

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

We removed the duplicates which differ only at the end

In [10]:

```
print('Data :', data.shape[0])
```

```
Data : 17593
```

In [12]:

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

Remove duplicates : Part 2

In the previous cell, we sorted whole data in alphabetical order of titles.The n, 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, X X-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 [14]:

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

In [ ]:

```
# 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])

    a = data['title'].loc[i].split()

    for j in indices:
        b = data['title'].loc[i].split()

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

        count = 0

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

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

In [ ]:

```
# consider the data now
data = data.loc[data['asin'].isin(stage2_dedupe_asins)]
```

In [ ]:

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

In [ ]:

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

Text Preprocessing

In [16]:

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

In [17]:

```
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()))

            #Lower case
            word= word.lower()

            #stop word removal
            if not word in stop_words:
                string += word + " "
        data[column][index] = string
```

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

In [18]:

```
start_time = time.clock()

for index, row in data.iterrows():
    nlp_preprocessing(row['title'], index, 'title')
#pritn time taken
print(time.clock()-start_time, "seconds")
```

```
7.763366871413316 seconds
```

In [19]:

```
data.head()
```

Out[19]:

	asin	brand	color	medium_image_url	product_type_name	
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT	featl ladie slee resis
6	B012YX2ZPI	HX-Kingdom Fashion T-shirts	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	wom uniq cotto spec olym wor..
15	B003BSRPB0	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	featl ladie mois free spor
27	B014ICEJ1Q	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT	supe chibi dear neck
46	B01NACPBG2	Fifth Degree	Black	https://images-na.ssl-images-amazon.com/images...	SHIRT	fifth wom gold grap jun..

In [20]:

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

Stemming:

In [27]:

```
from nltk.stem.porter import *
stemmer = PorterStemmer()
print(stemmer.stem('applied'))
print(stemmer.stem('scientific'))
```

```
appli
scientif
```

Text based product similarity:

In [2]:

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

Out[2]:

	asin	brand	color	medium_image_url	product_type_name	
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT	fe la sl re
6	B012YX2ZPI	HX-Kingdom Fashion T-shirts	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	w ur cc sp ol w
15	B003BSRPB0	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	fe la m fr sp
27	B014ICEJ1Q	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT	su ch de ne
46	B01NACPBG2	Fifth Degree	Black	https://images-na.ssl-images-amazon.com/images...	SHIRT	fif w gc gr ju

In [32]:

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


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


#plotting code to understand the algorithm's decision.
def plot_heatmap(keys, values, labels, url, text):
        # keys: list of words of recommended title
        # values: len(values) ==  len(keys), values(i) represents the occurence of the
 word keys(i)
        # labels: len(labels) == len(keys), the values of labels depends on the model w
e 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, ploting heat map that represents the count of commonly ocurred words in
 title2
        ax = plt.subplot(gs[0])
        # it displays a cell in white color if the word is intersection(lis of words of
title1 and list of words of title2), in black if not
        ax = sns.heatmap(np.array([values]), annot=np.array([labels]))
        ax.set_xticklabels(keys) # set that axis labels as the words of title
        ax.set_title(text) # apparel title

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

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

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

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

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

```
        # 1. bag_of_words
        # 2. tfidf
        # 3. idf

    # we find the common words in both titles, because these only words contribute to t
he distance between two title vec's
    intersection = set(vec1.keys()) & set(vec2.keys())

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

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

    # labels: len(labels) == len(keys), the values of labels depends on the model we ar
e 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 corpu
s
            # tfidf_title_features[doc_id, index_of_word_in_corpus] will give the tfidf
value of word in given document (doc_id)
            if x in  tfidf_title_vectorizer.vocabulary_:
                labels.append(tfidf_title_features[doc_id, tfidf_title_vectorizer.vocab
ulary_[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 val
ue of word in given document (doc_id)
            if x in  idf_title_vectorizer.vocabulary_:
                labels.append(idf_title_features[doc_id, idf_title_vectorizer.vocabular
y_[x]])
            else:
                labels.append(0)

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


# this function gets a list of wrods along with the frequency of each
# word given "text"
def text_to_vector(text):
    word = re.compile(r'\w+')
    words = word.findall(text)
    # words stores list of all words in given string, you can try 'words = text.split
```

```
()' this will also gives same result
    return Counter(words) # Counter counts the occurence of each word in list, it retur
ns dict type object {word1:count}



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

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

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

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

Bag of Words (BoW) on product titles.

In [4]:

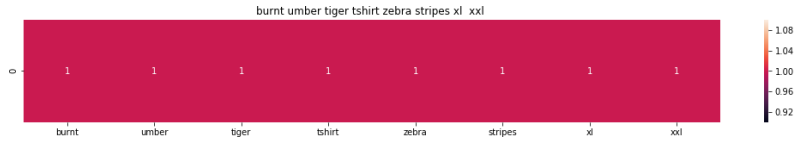
```
from sklearn.feature_extraction.text import CountVectorizer
title_vectorizer = CountVectorizer()
title_features = title_vectorizer.fit_transform(data['title'])
title_features.get_shape()
```

Out[4]:

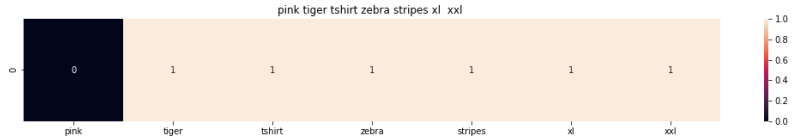
```
(16042, 12609)
```

In [5]:

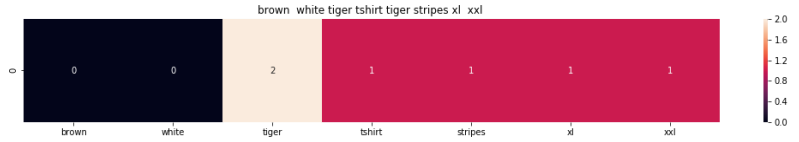
```
def bag_of_words_model (doc_id, num_results):
    pairwise_dist = pairwise_distances(title_features, title_features[doc_id])

    # 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_in
dices[i]], data['medium_image_url'].loc[df_indices[i]], 'bag_of_words')
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print ('Brand:', data['brand'].loc[df_indices[i]])
        print ('Title:', data['title'].loc[df_indices[i]])
        print ('Euclidean similarity with the query image :', pdists[i])
        print('='*60)

#call the bag-of-words model for a product to get similar products.
bag_of_words_model(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 : B00JXQB5FQ
Brand: Si Row
Title: burnt umber tiger tshirt zebra stripes xl xxl
Euclidean similarity with the query image : 0.0
==========

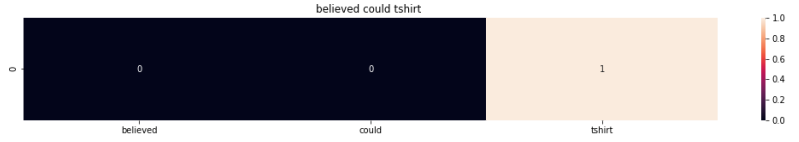


ASIN : B00JXQASS6
Brand: Si Row
Title: pink tiger tshirt zebra stripes xl xxl
Euclidean similarity with the query image : 1.7320508075688772
==========

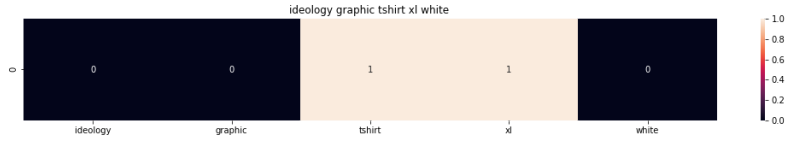


ASIN : B00JXQCWTO
Brand: Si Row
Title: brown white tiger tshirt tiger stripes xl xxl
Euclidean similarity with the query image : 2.449489742783178
==========

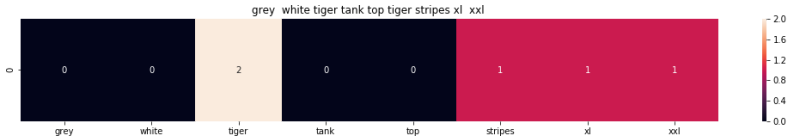
ASIN : B00JXQCUIC
Brand: Si Row
Title: yellow tiger tshirt tiger stripes l
Euclidean similarity with the query image : 2.6457513110645907
==========



ASIN : B07568NZX4
Brand: Rustic Grace
Title: believed could tshirt
Euclidean similarity with the query image : 3.0
==========

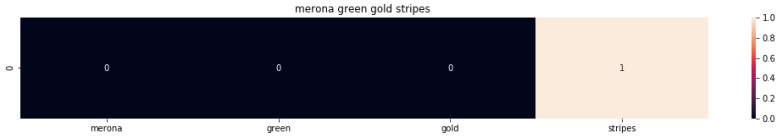


ASIN : B01NB0NKRO
Brand: Ideology
Title: ideology graphic tshirt xl white
Euclidean similarity with the query image : 3.0
==

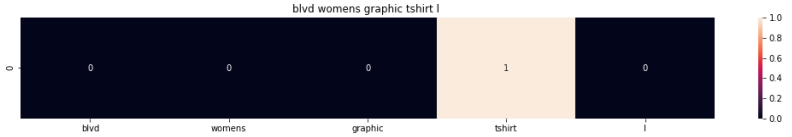


ASIN : B00JXQAFZ2
Brand: Si Row
Title: grey white tiger tank top tiger stripes xl xxl
Euclidean similarity with the query image : 3.0
==

ASIN : B01CLS8LMW
Brand: Awake
Title: morning person tshirt troll picture xl
Euclidean similarity with the query image : 3.1622776601683795
==



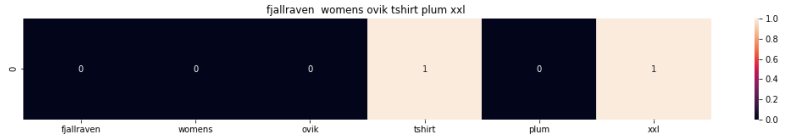
ASIN : B01KVZUB6G
Brand: Merona
Title: merona green gold stripes
Euclidean similarity with the query image : 3.1622776601683795
==



ASIN : B0733R2CJK
Brand: BLVD
Title: blvd womens graphic tshirt l
Euclidean similarity with the query image : 3.1622776601683795
==

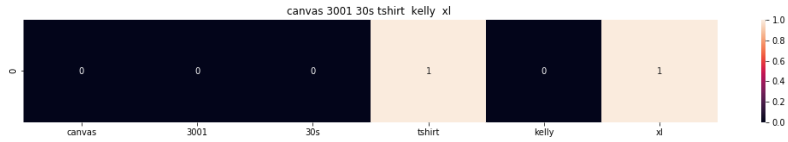
ASIN : B012VQLT6Y
Brand: KM T-shirt
Title: km tiger printed sleeveless vest tshirt
Euclidean similarity with the query image : 3.1622776601683795
==

ASIN : B00JXQC8L6
Brand: Si Row
Title: blue peacock print tshirt l
Euclidean similarity with the query image : 3.1622776601683795
==========

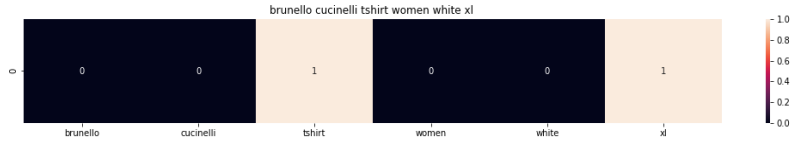


ASIN : B06XC3CZF6
Brand: Fjallraven
Title: fjallraven womens ovik tshirt plum xxl
Euclidean similarity with the query image : 3.1622776601683795
==========

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 lucywaaaaahhhh tshirt size xl
Euclidean similarity with the query image : 3.1622776601683795
============================================================
```

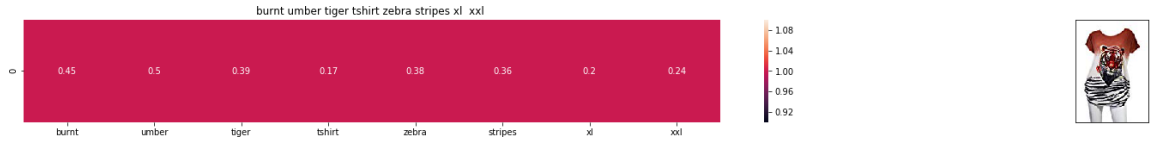
TF-IDF based product similarity:

In [11]:

```
tfidf_title_vectorizer = TfidfVectorizer(min_df = 0)
tfidf_title_features = tfidf_title_vectorizer.fit_transform(data['title'])
```

In [12]:

```
def tfidf_model(doc_id, num_results) :
    pairwise_dist = pairwise_distances(tfidf_title_features, tfidf_title_features[doc_i
d])

        # np.argsort will return indices of 9 smallest distances

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

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

    # data frame indices of 9 smallest dstnaces
    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_i
ndices[i]], data['medium_image_url'].loc[df_indices[i]], 'tfidf')
        print('ASIN :', data['asin'].loc[df_indices[i]])
        print('BRAND :', data['brand'].loc[df_indices[i]])
        print('Euclidian ditance feom the given image :', pdists[i])
        print('='*125)
tfidf_model(12566, 20)
```



ASIN : B00JXQB5FQ
BRAND : Si Row
Euclidian ditance feom the given image : 0.0
==========
==========

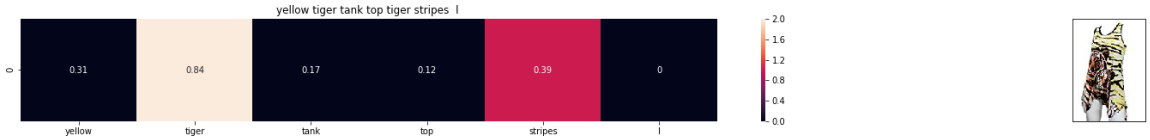
ASIN : B00JXQASS6
BRAND : Si Row
Euclidian ditance feom the given image : 0.7536331912451361
==========
==========

ASIN : B00JXQCWTO
BRAND : Si Row
Euclidian ditance feom the given image : 0.9357643943769645
==========
==========

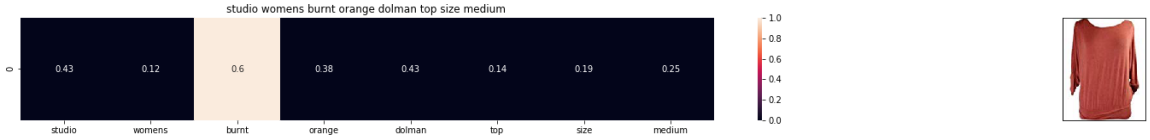
ASIN : B00JXQAFZ2
BRAND : Si Row
Euclidian ditance feom the given image : 0.9586153524200749
==========
==========

ASIN : B00JXQCUIC
BRAND : Si Row
Euclidian ditance feom the given image : 1.000074961446881
==========
==========

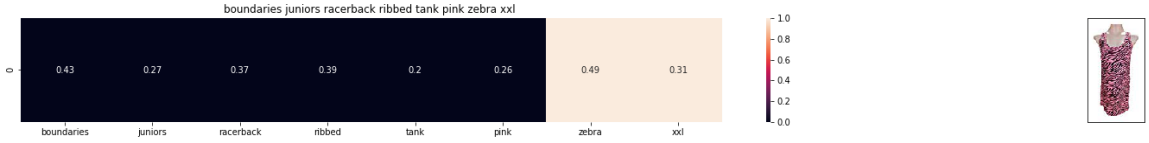
ASIN : B00JXQAO94
BRAND : Si Row
Euclidian ditance feom the given image : 1.023215552457452
==========
==========



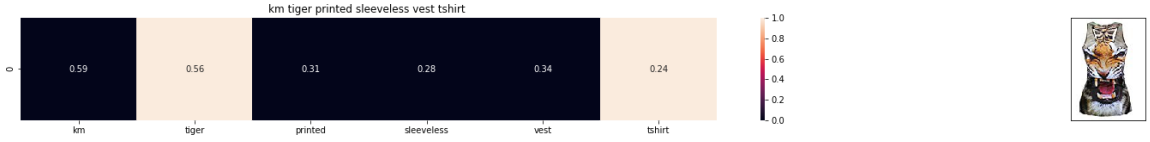
ASIN : B00JXQAUWA
BRAND : Si Row
Euclidian ditance feom the given image : 1.031991846303421
==========
==========



ASIN : B06XSCVFT5
BRAND : Studio M
Euclidian ditance feom the given image : 1.2106843670424716
==========
==========



ASIN : B06Y2GTYPM
BRAND : No Boundaries
Euclidian ditance feom the given image : 1.212168381072083
==========
==========



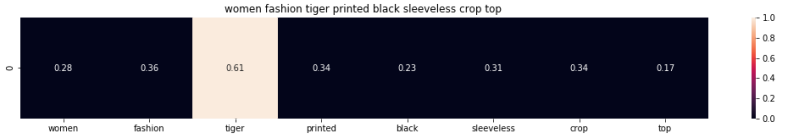
ASIN : B012VQLT6Y
BRAND : KM T-shirt
Euclidian ditance feom the given image : 1.219790640280982
==========
==========

ASIN : B06Y1VN8WQ
BRAND : Black Swan
Euclidian ditance feom the given image : 1.2206849659998316
==========
==========

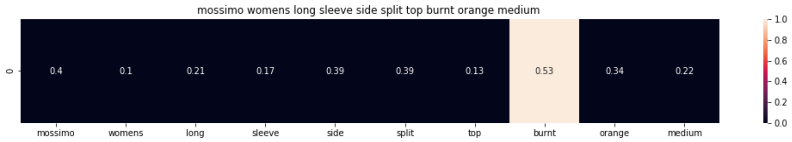
ASIN : B00Z6HEXWI
BRAND : Black Temptation
Euclidian ditance feom the given image : 1.221281392120943
==========
==========

ASIN : B074TR12BH
BRAND : Ultra Flirt
Euclidian ditance feom the given image : 1.2313364094597743
==========
==========

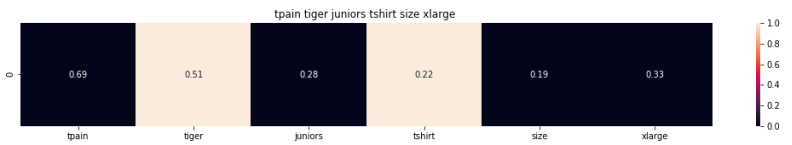
ASIN : B072R2JXKW
BRAND : WHAT ON EARTH
Euclidian ditance feom the given image : 1.2318451972624518
==========
==========



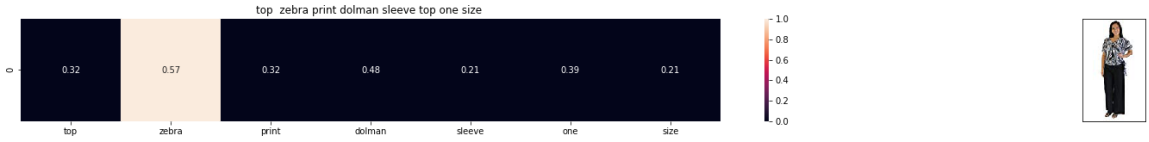
ASIN : B074T8ZYGX
BRAND : MKP Crop Top
Euclidian ditance feom the given image : 1.2340607457359425
==========
==========



ASIN : B071ZDF6T2
BRAND : Mossimo
Euclidian ditance feom the given image : 1.2352785577664824
==========
==========

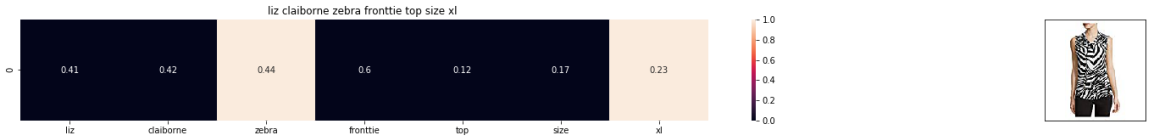


```
ASIN : B01K0H02OG
BRAND : Tultex
Euclidian ditance feom the given image : 1.236457298812782
=====================================================================
==================================================
```




```
ASIN : B00H8A6ZLI
BRAND : Vivian's Fashions
Euclidian ditance feom the given image : 1.24996155052848
=====================================================================
==================================================
```


```
ASIN : B010NN9RXO
BRAND : YICHUN
Euclidian ditance feom the given image : 1.25354614208561
=====================================================================
==================================================
```




```
ASIN : B06XBY5QXL
BRAND : Liz Claiborne
Euclidian ditance feom the given image : 1.2538832938357722
=====================================================================
==================================================
```

IDF based product similarity:

In [13]:

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

In [14]:

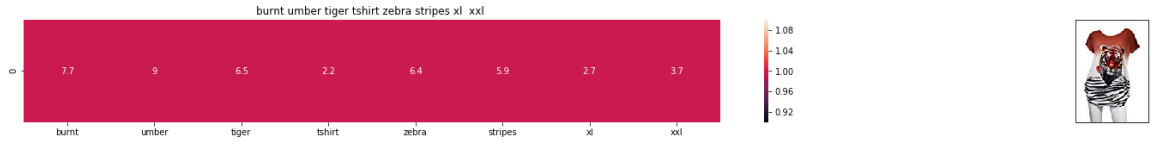
```
def n_containing(word):
    # return the number of documents which had the given word
    return sum(1 for blob in data['title'] if word in blob.split())

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

In [15]:

```
# 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 retu
rn all documents in which the word i present
    for j in idf_title_features[:, idf_title_vectorizer.vocabulary_[i]].nonzero()[0]:

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

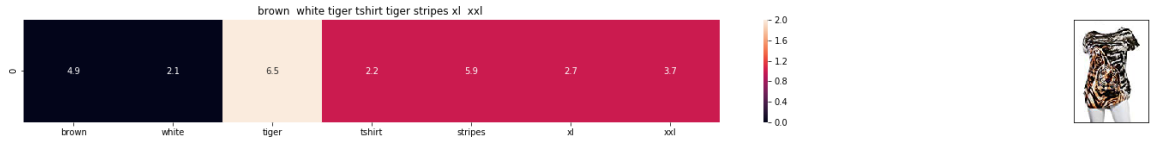
In [16]:

```
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 a
pparels
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y) = <
X, Y> / (||X||*||Y||)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(idf_title_features,idf_title_features[doc_id])

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

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

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



idf_model(12566,20)
# in the output heat map each value represents the idf values of the label word, the co
lor represents the intersection with inputs title
```



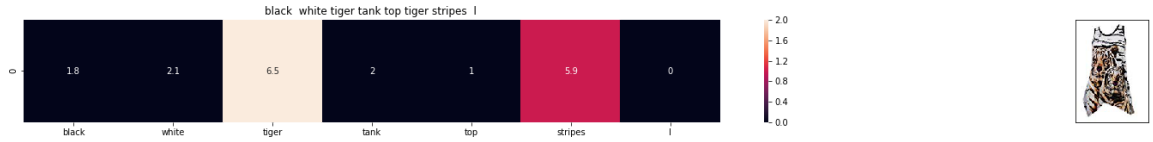
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from the given image : 0.0
==========
==========

ASIN : B00JXQASS6
Brand : Si Row
euclidean distance from the given image : 12.20507131122177
==========
==========



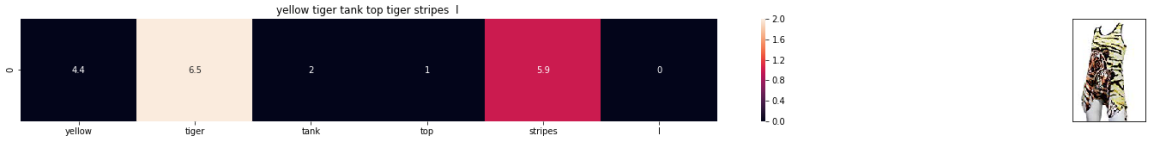
ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from the given image : 14.468362685603465
==========
==========

ASIN : B00JXQAFZ2
Brand : Si Row
euclidean distance from the given image : 14.486832924778964
==========
==========

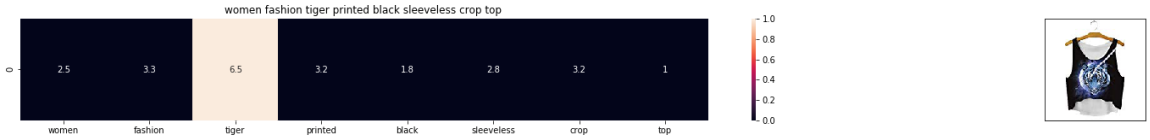


ASIN : B00JXQAO94
Brand : Si Row
euclidean distance from the given image : 14.833392966672909
==========
==========

ASIN : B00JXQCUIC
Brand : Si Row
euclidean distance from the given image : 14.898744516719225
==========
==========



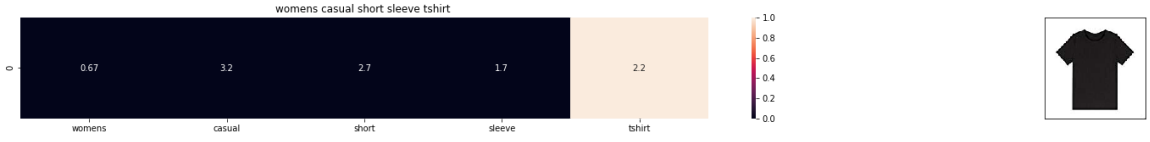
ASIN : B00JXQAUWA
Brand : Si Row
euclidean distance from the given image : 15.224458287343769
==========
==========



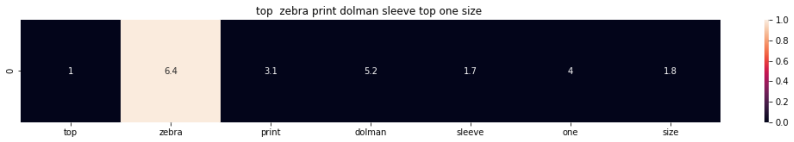
ASIN : B074T8ZYGX
Brand : MKP Crop Top
euclidean distance from the given image : 17.080812955631995
==========
==========

ASIN : B00KF2N5PU
Brand : Vietsbay
euclidean distance from the given image : 17.090168125645416
==========
==========

ASIN : B00JPOZ9GM
Brand : Sofra
euclidean distance from the given image : 17.153215337562703
==========
==========

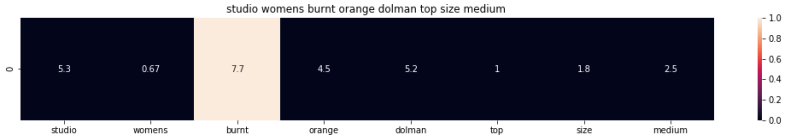


ASIN : B074T9KG9Q
Brand : Rain
euclidean distance from the given image : 17.33671523874989
==========
==========

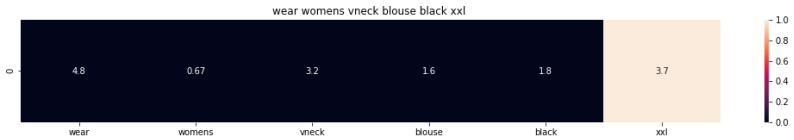


ASIN : B00H8A6ZLI
Brand : Vivian's Fashions
euclidean distance from the given image : 17.410075941001253
==========
==========

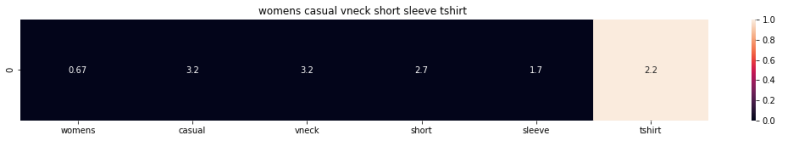
ASIN : B074G5G5RK
Brand : ERMANNO SCERVINO
euclidean distance from the given image : 17.539921335459557
==========
==========



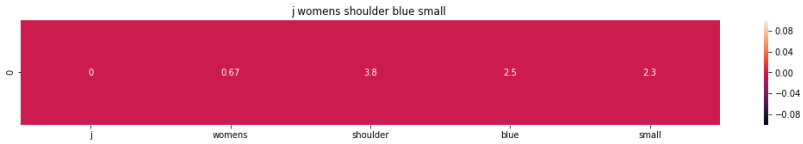
ASIN : B06XSCVFT5
Brand : Studio M
euclidean distance from the given image : 17.61275854366134
==========
==========



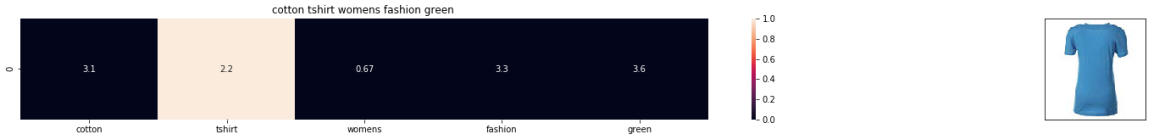
ASIN : B06Y6FH453
Brand : Who What Wear
euclidean distance from the given image : 17.623745282500135
==========
==========



ASIN : B074V45DCX
Brand : Rain
euclidean distance from the given image : 17.634342496835046
==========
==========



ASIN : B07583CQFT
Brand : Very J
euclidean distance from the given image : 17.63753712743611
==========
==========



ASIN : B073GJGVBN
Brand : Ivan Levi
euclidean distance from the given image : 17.7230738913371
==========
==========

ASIN : B012VQLT6Y
Brand : KM T-shirt
euclidean distance from the given image : 17.762588561202364
==========
==========

ASIN : B00ZZMYBRG
Brand : HP-LEISURE
euclidean distance from the given image : 17.779536864674238
==========
==========

Text Semantics based product similarity :

In [17]:

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

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

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

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

In [18]:

```
# Utility functions

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

def get_distance(vec1, vec2):
    # vec1 = np.array(#number_of_words_title1 * 300), each row is a vector of length 30
0 corresponds to each word in give title
    # vec2 = np.array(#number_of_words_title2 * 300), each row is a vector of length 30
0 corresponds to each word in give title

    final_dist = []
    # for each vector in vec1 we caluclate the distance(euclidean) to all vectors in ve
c2
    for i in vec1:
        dist = []
        for j in vec2:
            # np.linalg.norm(i-j) will result the euclidean distance between vectors i,
j
            dist.append(np.linalg.norm(i-j))
        final_dist.append(np.array(dist))
    # final_dist = np.array(#number of words in title1 * #number of words in title2)
    # final_dist[i,j] = euclidean distance between vectors i, j
    return np.array(final_dist)


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

    #s1_vec = np.array(#number_of_words_title1 * 300), each row is a vector(weighted/av
g) of length 300 corresponds to each word in give title
    s1_vec = get_word_vec(sentence1, doc_id1, model)
```

```
    #s2_vec = np.array(#number_of_words_title1 * 300), each row is a vector(weighted/av
g) of length 300 corresponds to each word in give title
    s2_vec = get_word_vec(sentence2, doc_id2, model)

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



    # devide whole figure into 2 parts 1st part displays heatmap 2nd part displays imag
e 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])
    # ploting the heap map based on the pairwise distances
    ax = sns.heatmap(np.round(s1_s2_dist,4), annot=True)
    # set the x axis labels as recommended apparels title
    ax.set_xticklabels(sentence2.split())
    # set the y axis labels as input apparels title
    ax.set_yticklabels(sentence1.split())
    # set title as recommended apparels title
    ax.set_title(sentence2)

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

    plt.show()
```

In [19]:

```
# 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 s
entance
def build_avg_vec(sentence, num_features, doc_id, m_name):
    # sentace: its title of the apparel
    # num_features: the lenght of word2vec vector, its values = 300
    # m_name: model information it will take two values
        # if  m_name == 'avg', we will append the model[i], w2v representation of word
 i
        # if m_name == 'weighted', we will multiply each w2v[word] with the idf(word)

    featureVec = np.zeros((num_features,), dtype="float32")
    # we will intialize a vector of size 300 with all zeros
    # we add each word2vec(wordi) to this fetureVec
    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_ve
ctorizer.vocabulary_[word]] * model[word])
            elif m_name == 'avg':
                featureVec = np.add(featureVec, model[word])
    if(nwords>0):
        featureVec = np.divide(featureVec, nwords)
    # returns the avg vector of given sentance, its of shape (1, 300)
    return featureVec
```

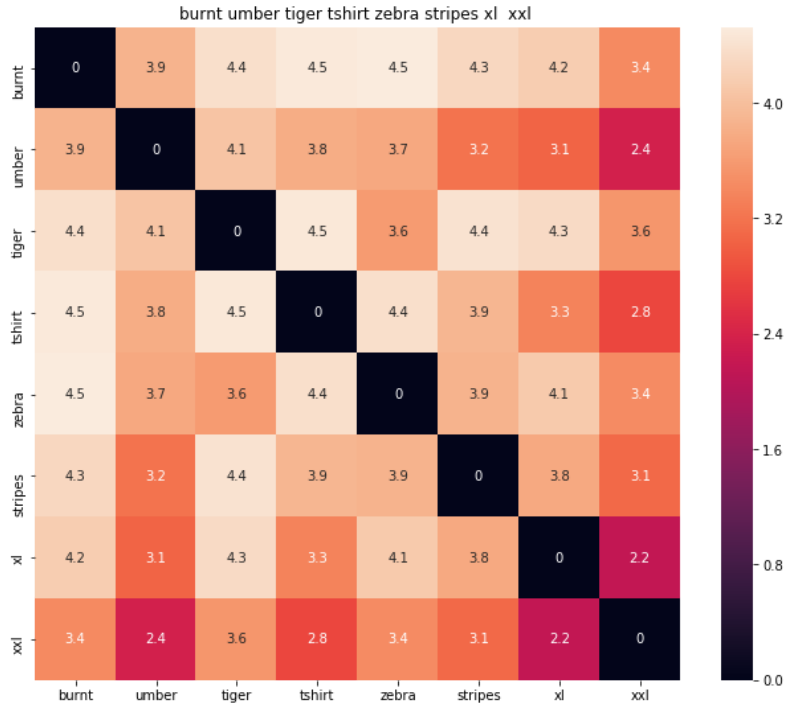
Average Word2Vec product similarity.

In [19]:

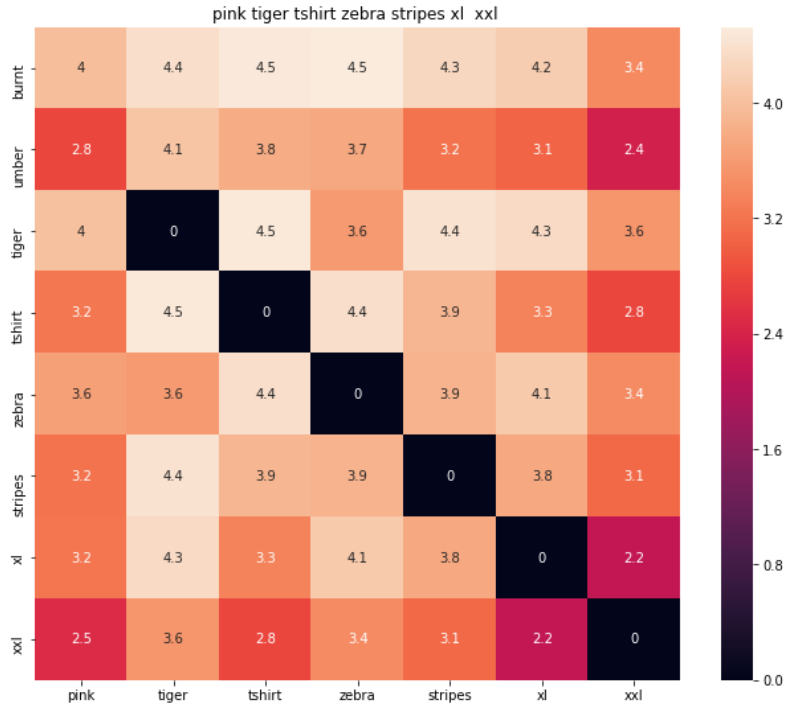
```
doc_id = 0
w2v_title = []

# 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 [20]:

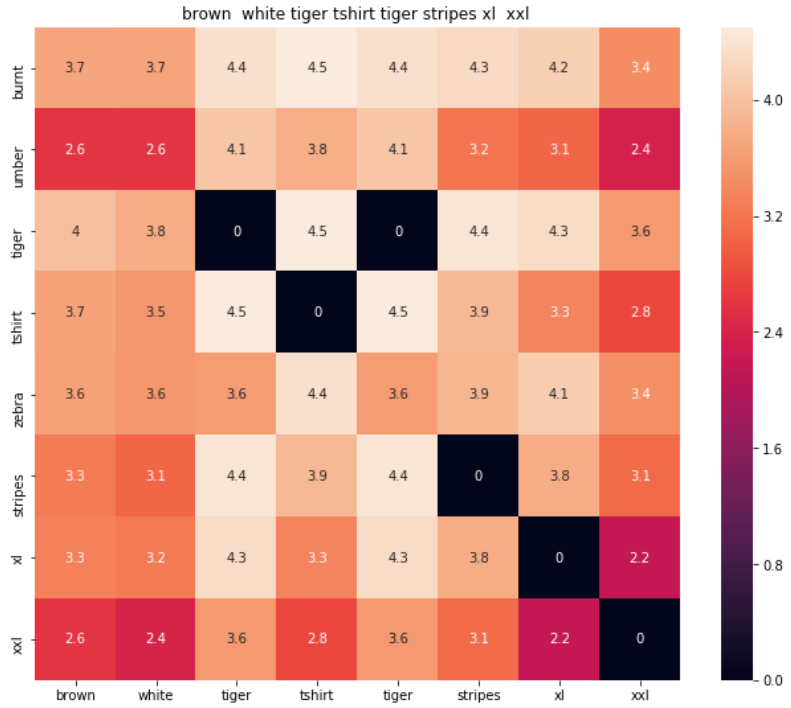
```
def avg_w2v_model(doc_id, num_results) :
    # doc_id: apparel's id in given corpus
    # dist(x, y) = sqrt(dot(x, x) - 2 * dot(x, y) + dot(y, y))
    pairwise_dist = pairwise_distances(w2v_title, w2v_title[doc_id].reshape(1,-1))

    # 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 9 smallest distnaces
    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'].loc[df_indices[i]], indices[0], indices[i], 'avg')
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('BRAND :',data['brand'].loc[df_indices[i]])
        print ('euclidean distance from given input image :', pdists[i])
        print('='*125)


avg_w2v_model(12566, 20)
# in the give heat map, each cell contains the euclidean distance between words i, j
```



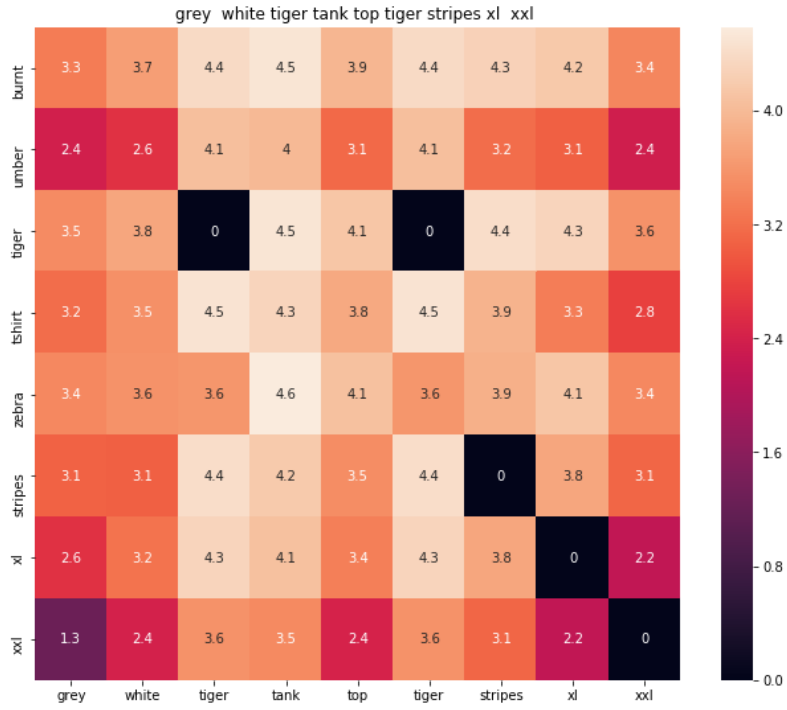
ASIN : B00JXQB5FQ
BRAND : Si Row
euclidean distance from given input image : 0.0
==========
==========



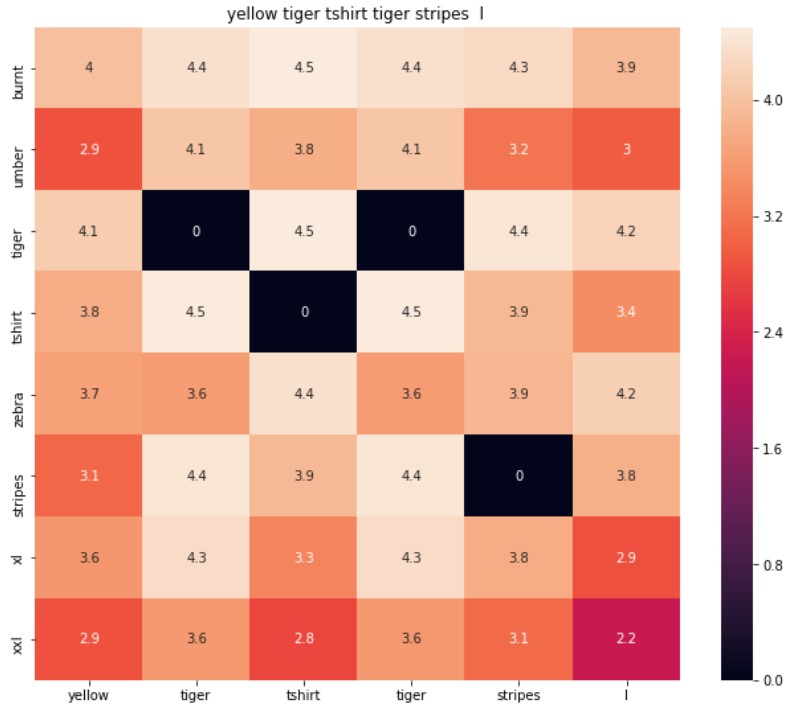
ASIN : B00JXQASS6
BRAND : Si Row
euclidean distance from given input image : 0.5891928
==========
==========



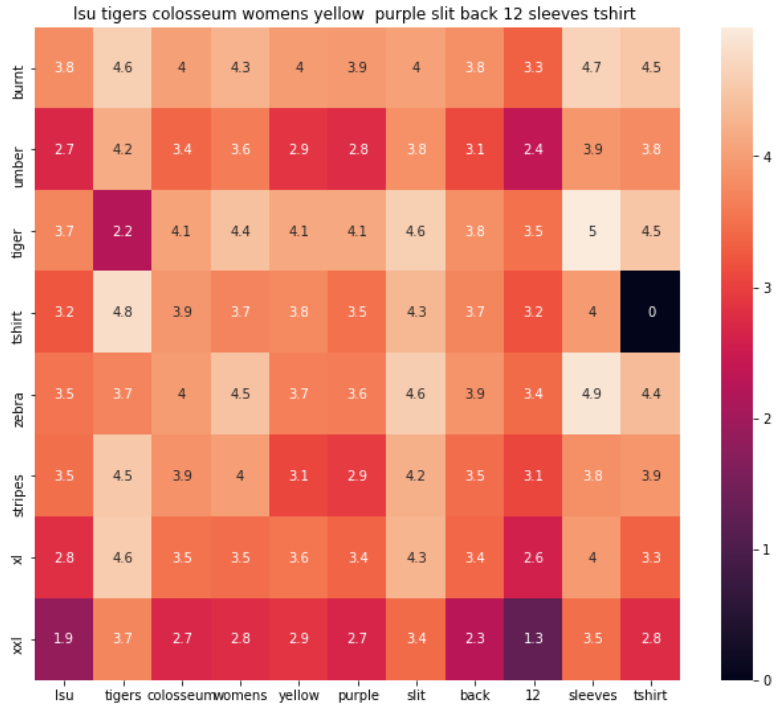
ASIN : B00JXQCWTO
BRAND : Si Row
euclidean distance from given input image : 0.7003436
==========
==========



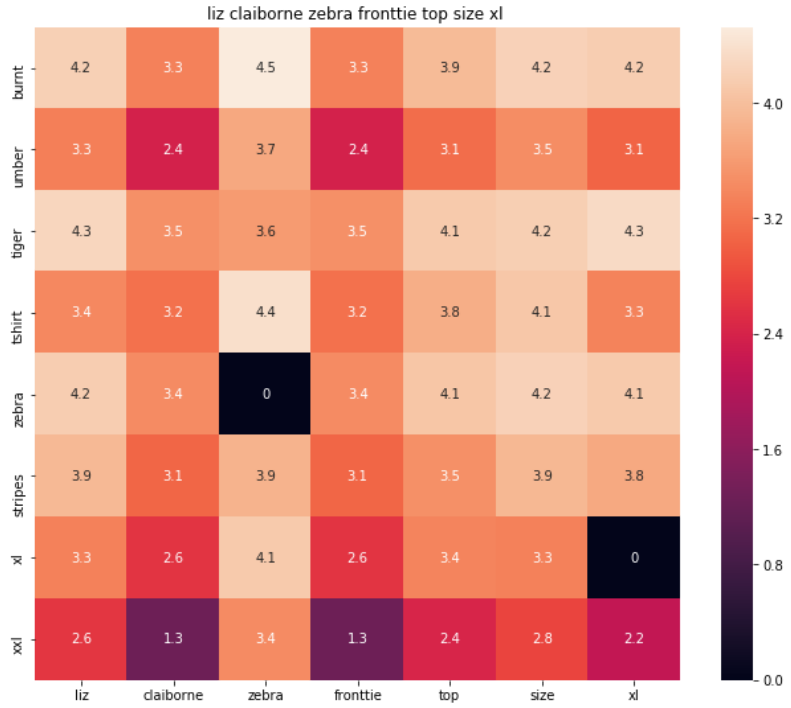
ASIN : B00JXQAFZ2
BRAND : Si Row
euclidean distance from given input image : 0.8928398
==========
==========



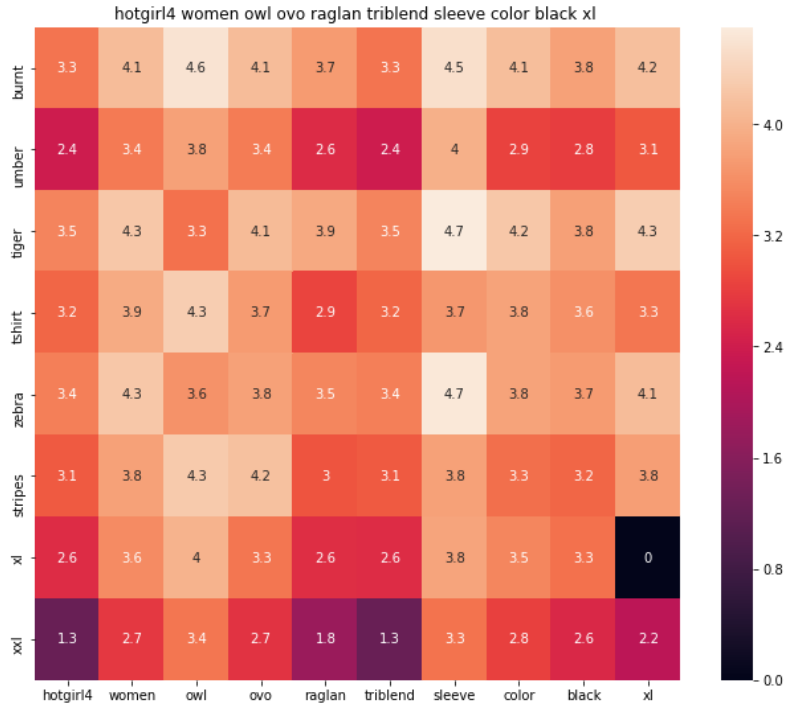
ASIN : B00JXQCUIC
BRAND : Si Row
euclidean distance from given input image : 0.9560129
==========
==========



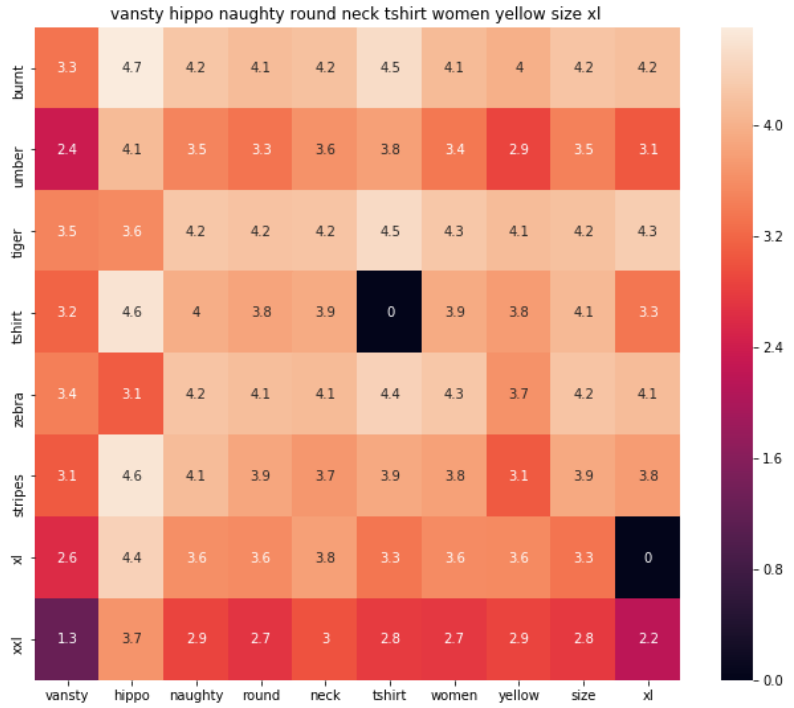
ASIN : B073R5Q8HD
BRAND : Colosseum
euclidean distance from given input image : 1.0229691
==========
==========



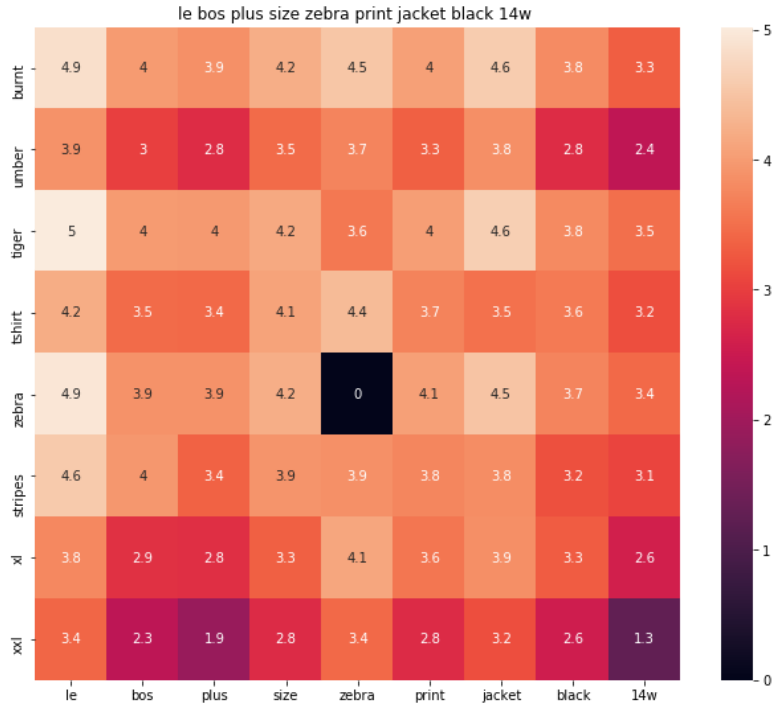
ASIN : B06XBY5QXL
BRAND : Liz Claiborne
euclidean distance from given input image : 1.0669324
==========
==========



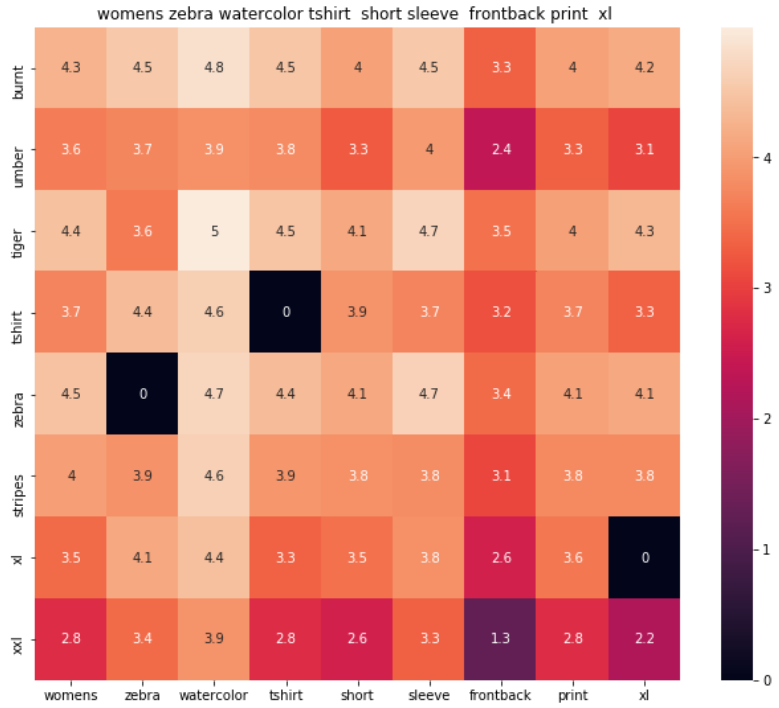
ASIN : B01L8L73M2
BRAND : Hotgirl4 Raglan Design
euclidean distance from given input image : 1.0731406
==========
==========



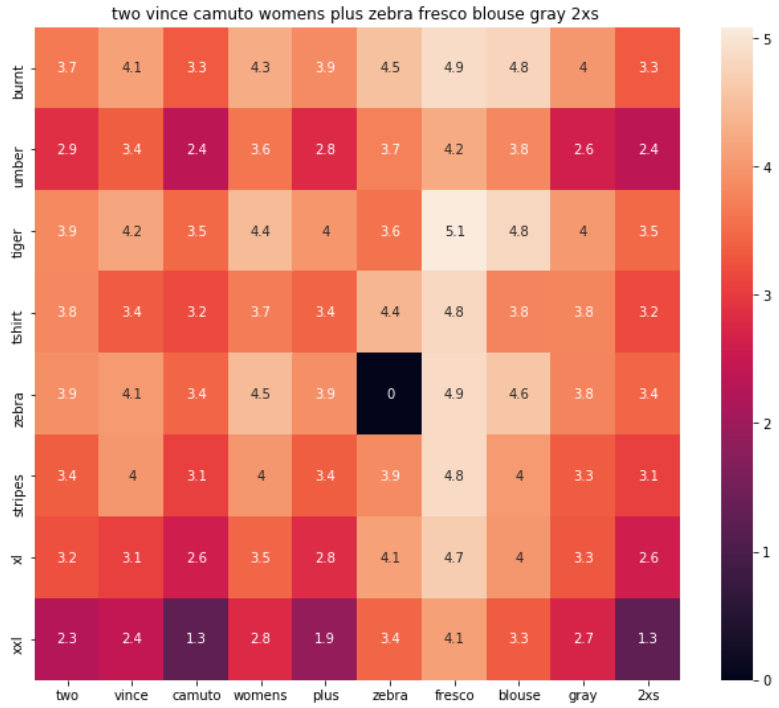
ASIN : B01EJS5H06
BRAND : Vansty
euclidean distance from given input image : 1.0757191
==========
==========



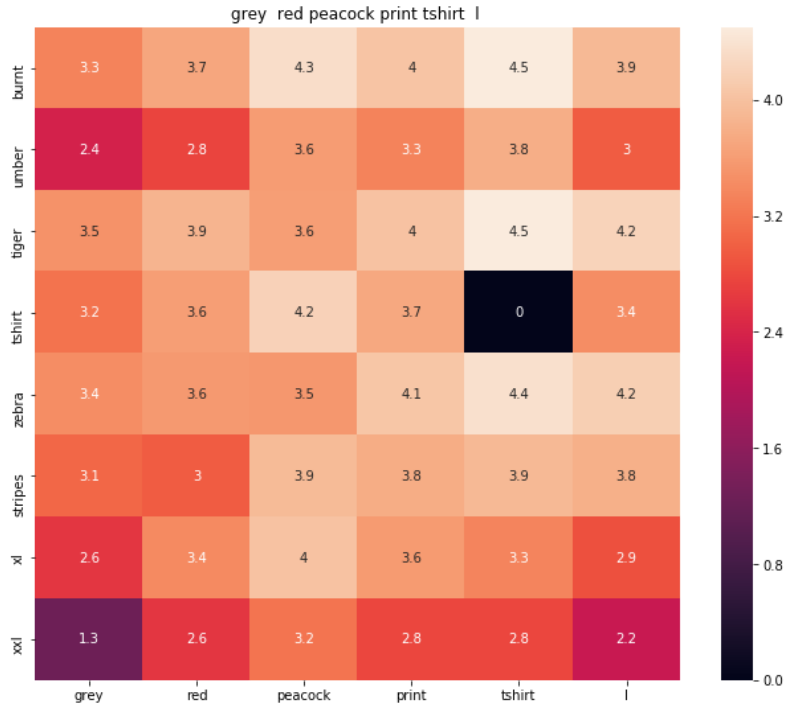
ASIN : B01BO1XRK8
BRAND : Le Bos
euclidean distance from given input image : 1.0839965
==========
==========



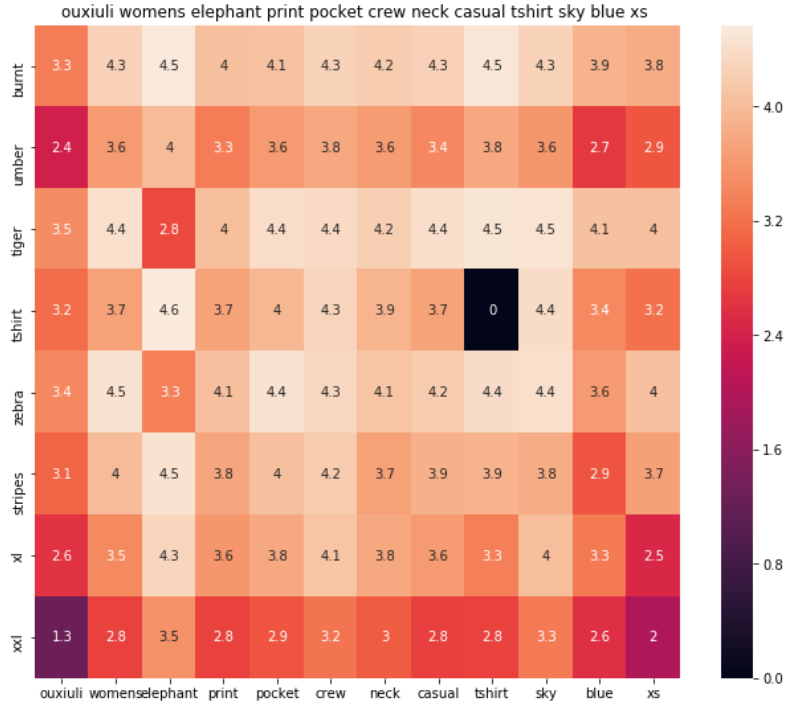
ASIN : B072R2JXKW
BRAND : WHAT ON EARTH
euclidean distance from given input image : 1.0842218
==========
==========



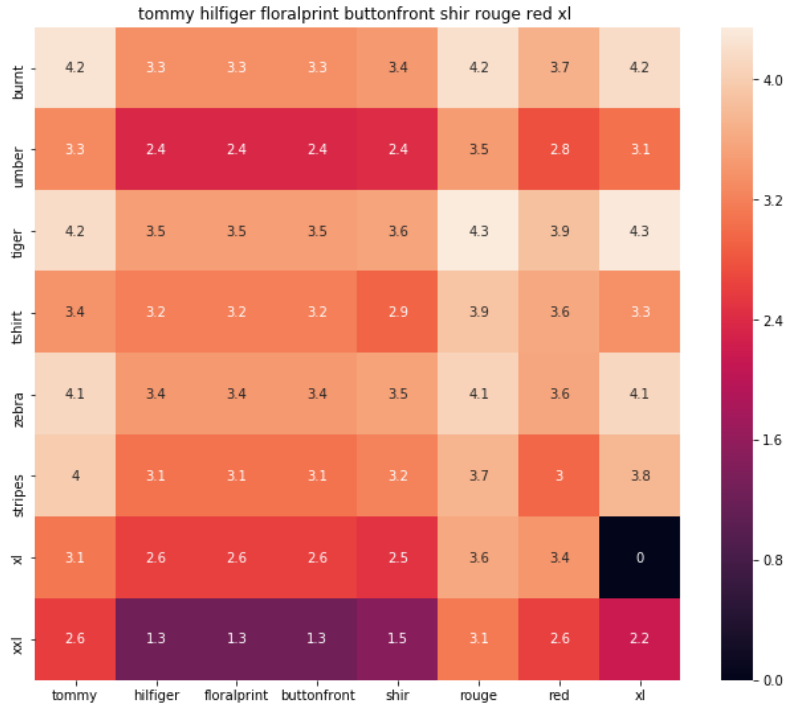
ASIN : B074MJRGW6
BRAND : Two by Vince Camuto
euclidean distance from given input image : 1.0895039
==========
==========



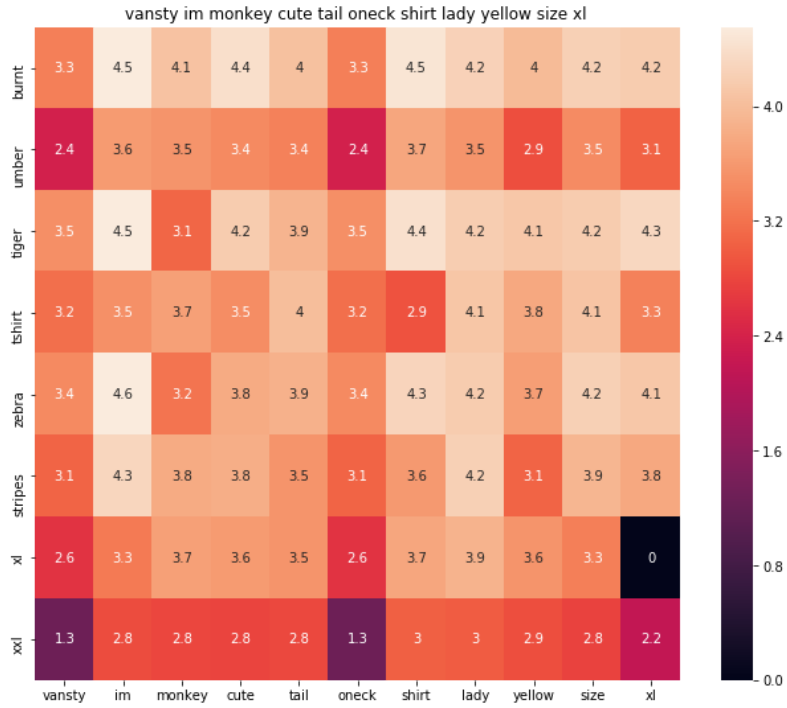
ASIN : B00JXQCFRS
BRAND : Si Row
euclidean distance from given input image : 1.0900588
==========
==========



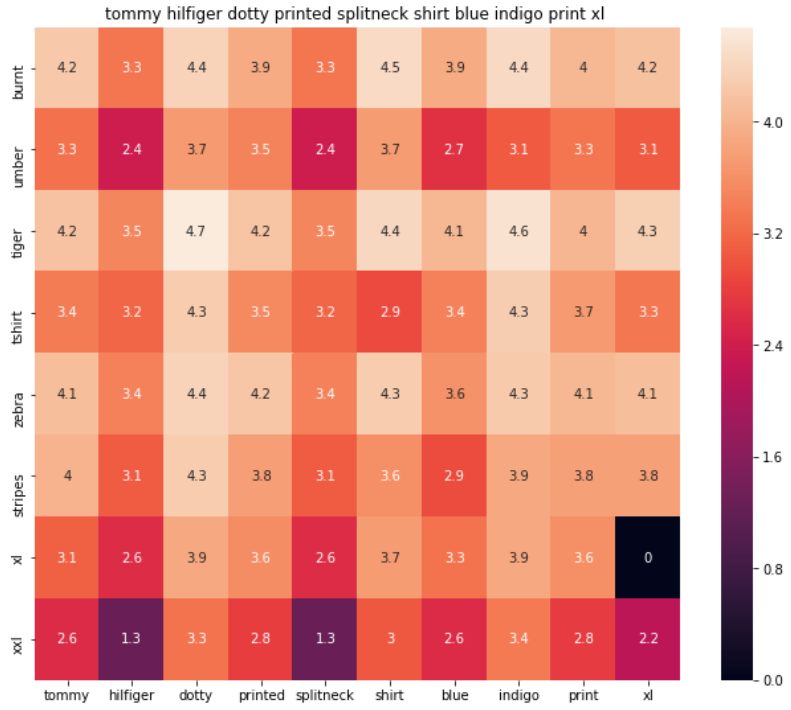
ASIN : B01I53HU6K
BRAND : ouxiuli
euclidean distance from given input image : 1.0920112
==========
==========



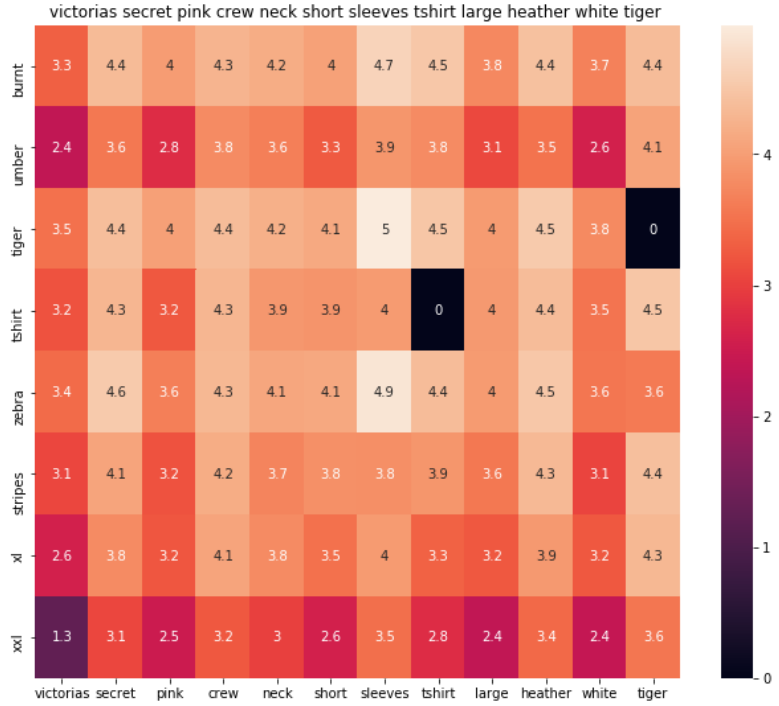
ASIN : B0711NGTQM
BRAND : THILFIGER RTW
euclidean distance from given input image : 1.0923418
==========
==========



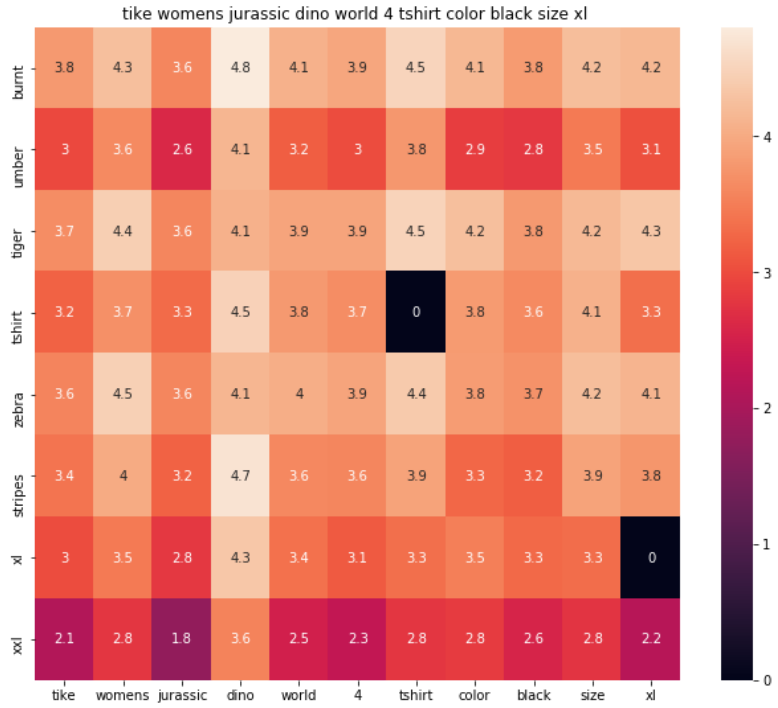
ASIN : B01EFSLO8Y
BRAND : Vansty
euclidean distance from given input image : 1.0934006
==========
==========



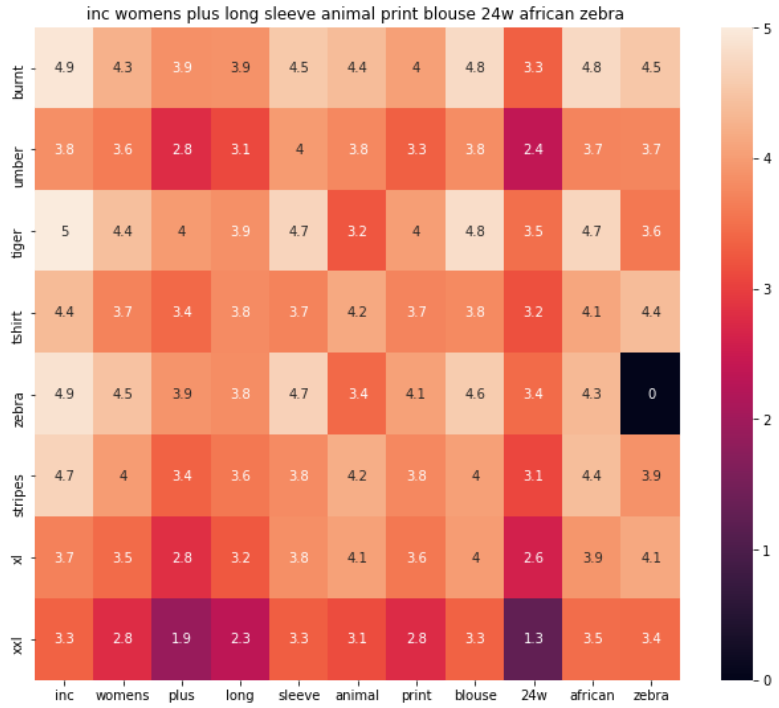
ASIN : B0716TVWQ4
BRAND : THILFIGER RTW
euclidean distance from given input image : 1.0942025
==========
==========



ASIN : B0716MVPGV
BRAND : V.Secret
euclidean distance from given input image : 1.0948305
==========
==========



ASIN : B016OPN4OI
BRAND : TIKE Fashions
euclidean distance from given input image : 1.0951276
==========
==========



```
ASIN : B018WDJCUA
BRAND : INC - International Concepts Woman
euclidean distance from given input image : 1.0966893
================================================================================
==================================================
```

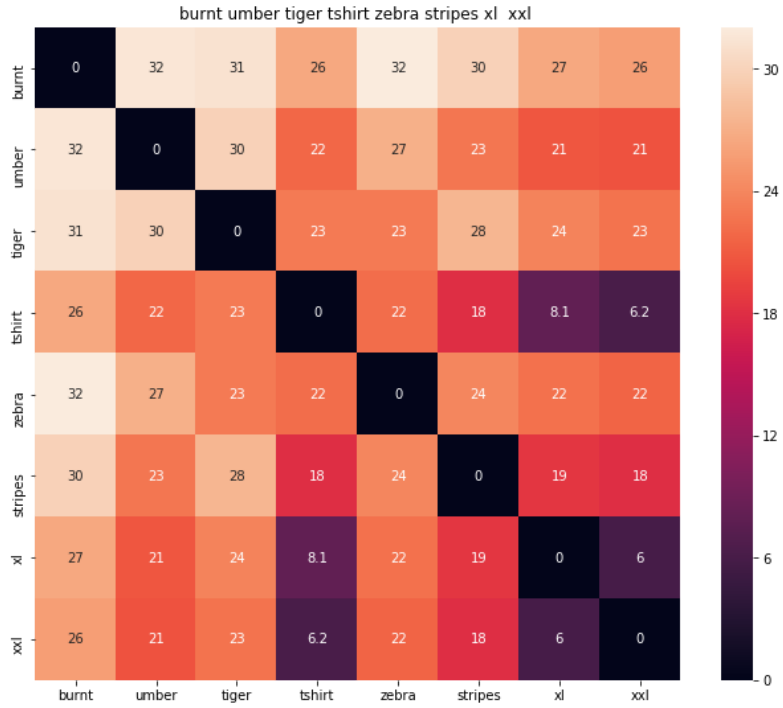
IDF weighted Word2Vec for product similarity:

In [20]:

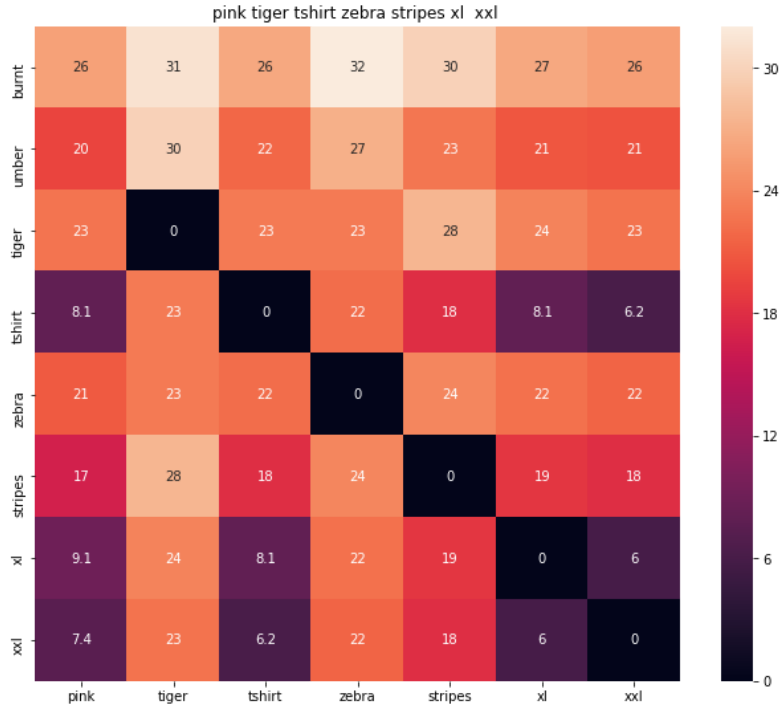
```
doc_id = 0
w2v_title_weight = []
# for every title we build a weighted vector representation
for i in data['title']:
    w2v_title_weight.append(build_avg_vec(i, 300, doc_id,'weighted'))
    doc_id += 1
# w2v_title = np.array(# number of doc in courpus * 300), each row corresponds to a doc
w2v_title_weight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

In [21]:

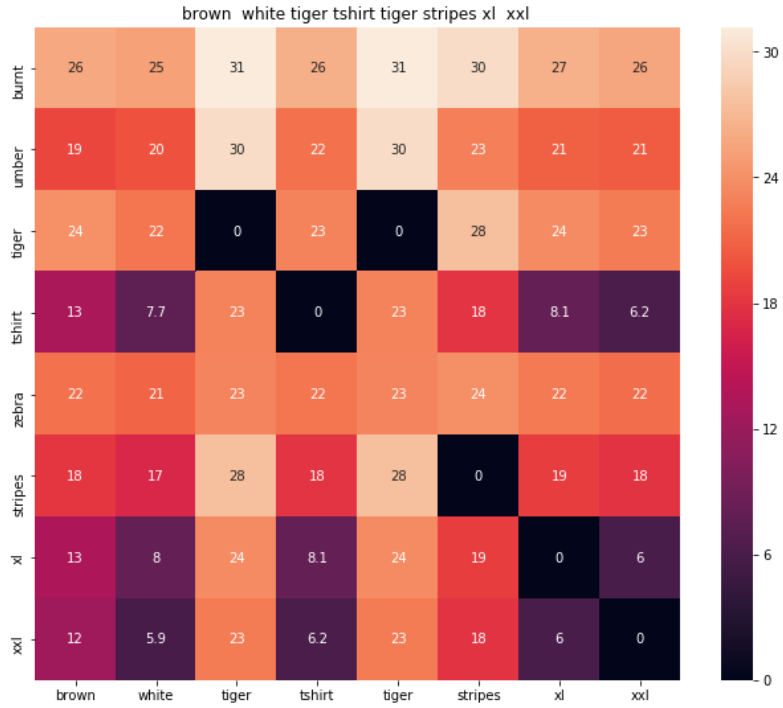
```
def weighted_w2v_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

    # pairwise_dist will store the distance from given input apparel to all remaining a
pparels
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y) = <
X, Y> / (||X||*||Y||)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(w2v_title_weight, w2v_title_weight[doc_id].resha
pe(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'].loc[df_indices[i]], indices[0], indices[i], 'weighted')
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('Brand :',data['brand'].loc[df_indices[i]])
        print('euclidean distance from input :', pdists[i])
        print('='*125)

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



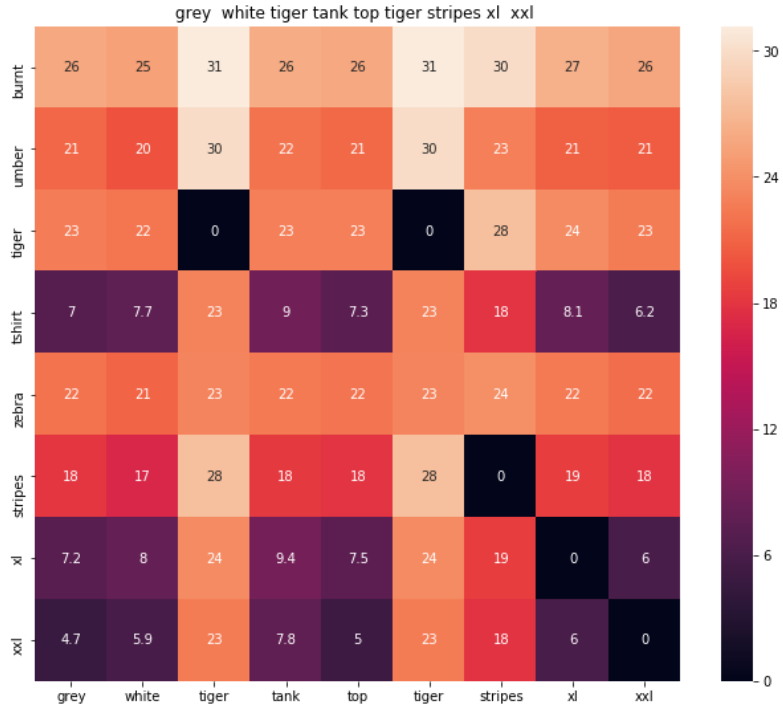
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 0.00390625
==========
==========



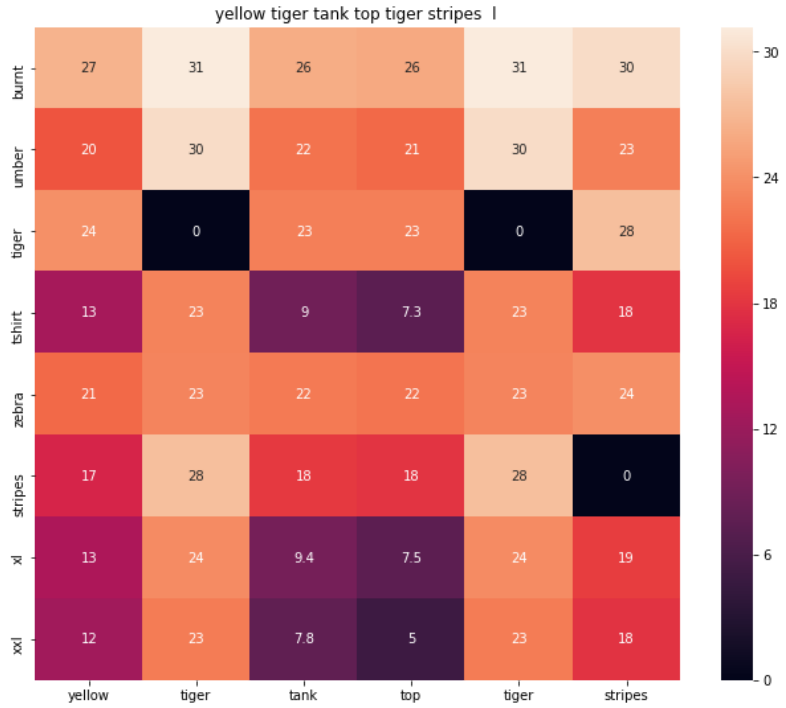
ASIN : B00JXQASS6
Brand : Si Row
euclidean distance from input : 4.0638876
==========
==========



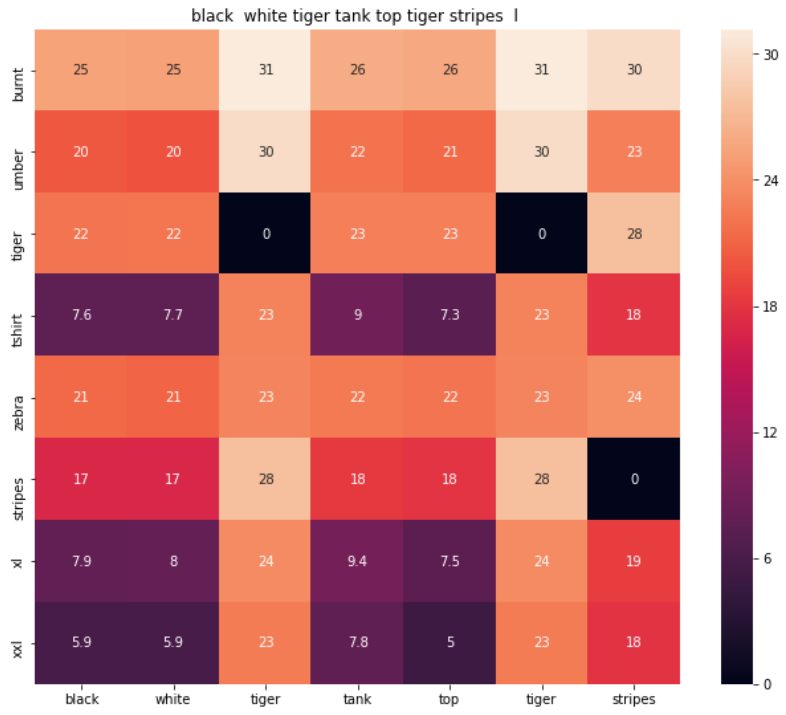
ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 4.770942
==========
==========



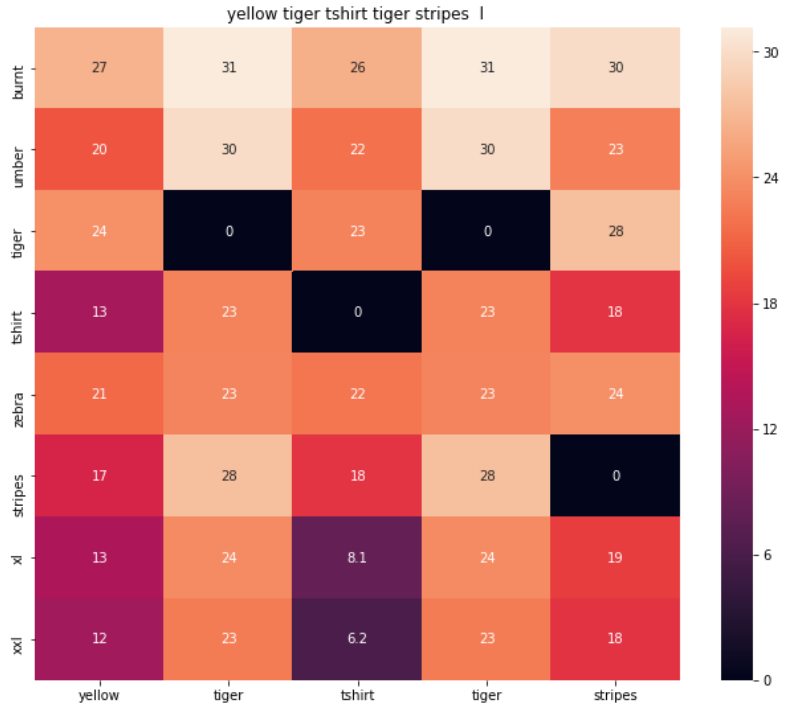
ASIN : B00JXQAFZ2
Brand : Si Row
euclidean distance from input : 5.3601613
==========
==========



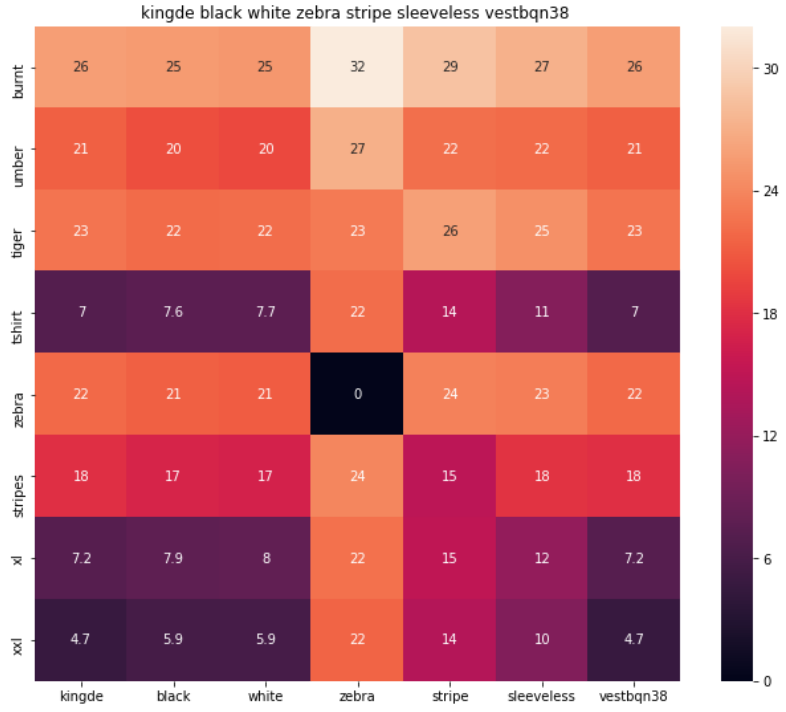
ASIN : B00JXQAUWA
Brand : Si Row
euclidean distance from input : 5.689523
==========
==========



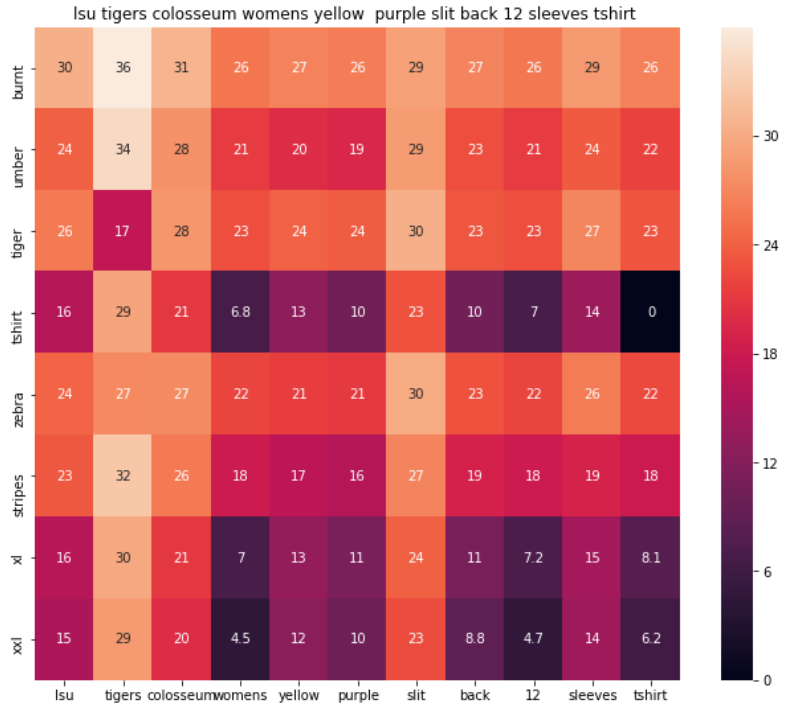
ASIN : B00JXQAO94
Brand : Si Row
euclidean distance from input : 5.6930227
==========
==========



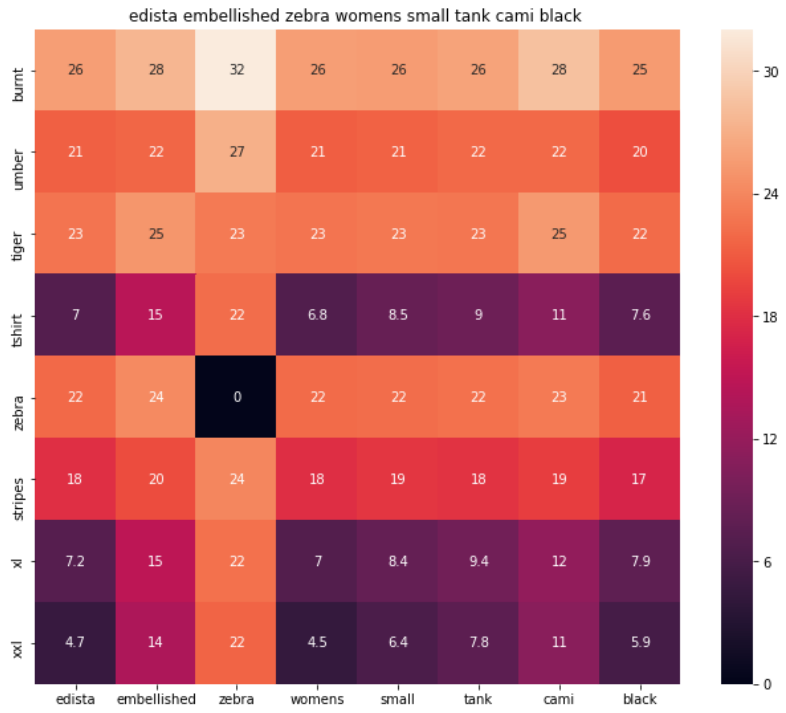
ASIN : B00JXQCUIC
Brand : Si Row
euclidean distance from input : 5.8934426
==========
==========



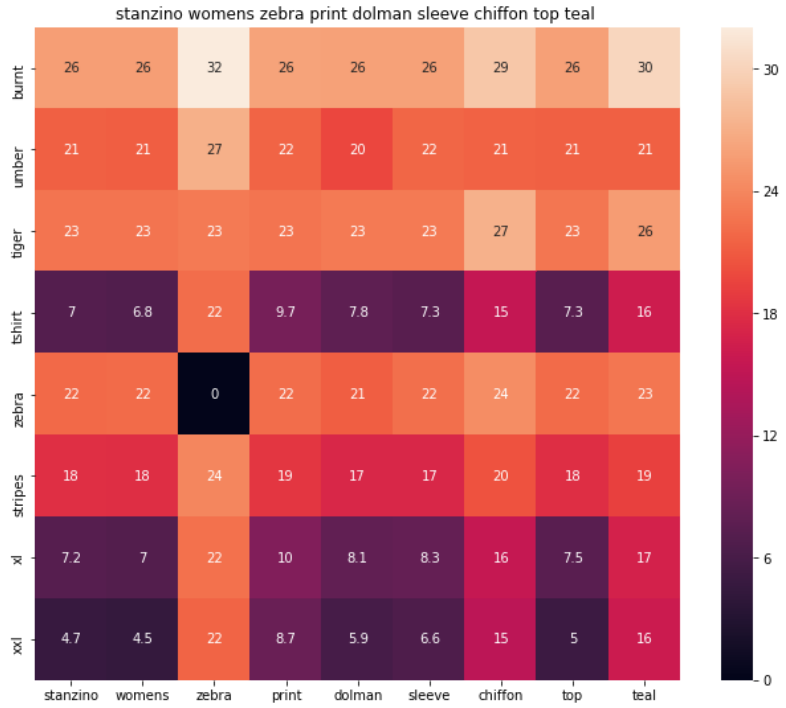
ASIN : B015H41F6G
Brand : KINGDE
euclidean distance from input : 6.13299
==========
==========



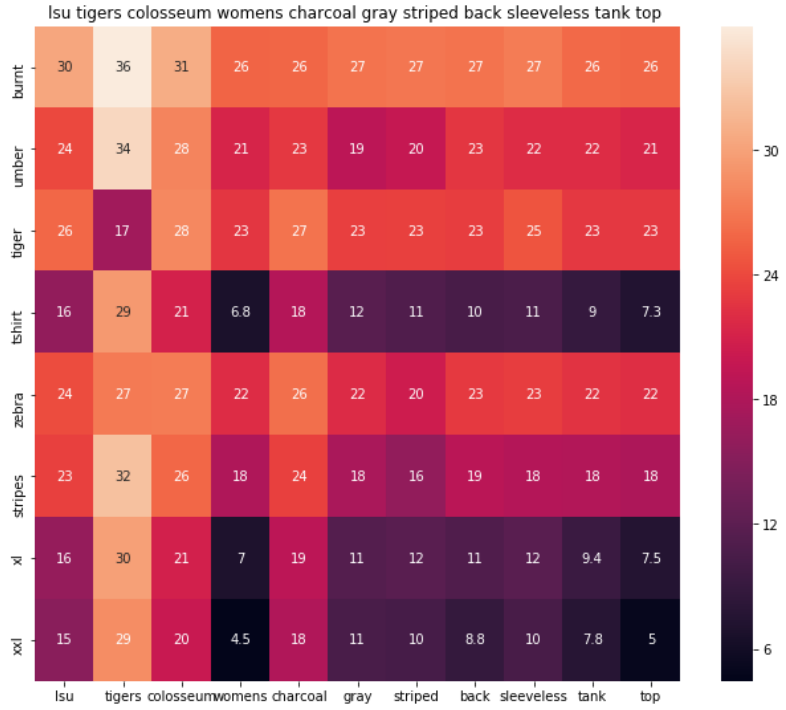
ASIN : B073R5Q8HD
Brand : Colosseum
euclidean distance from input : 6.2567058
==========
==========



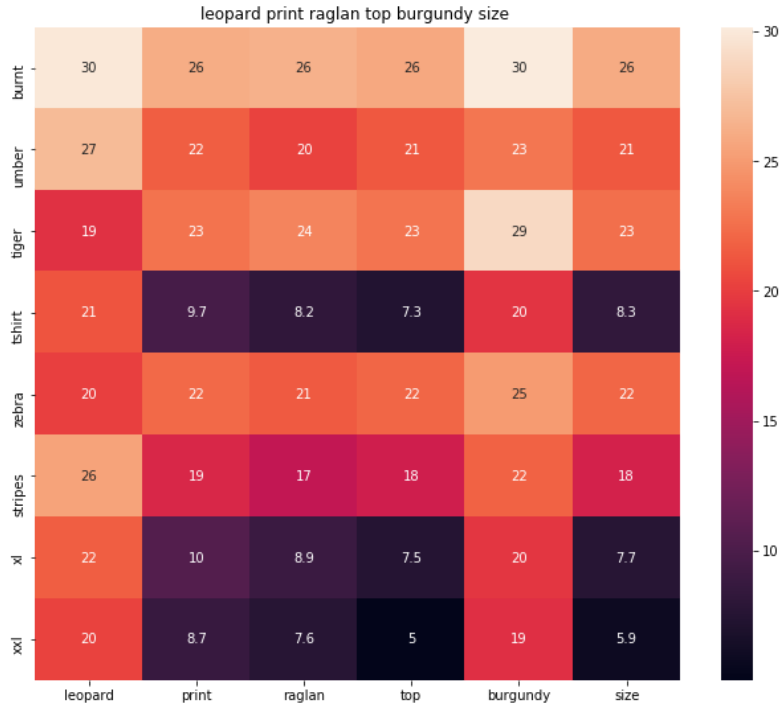
ASIN : B074P8MD22
Brand : Edista
euclidean distance from input : 6.3922043
==========
==========



ASIN : B00C0I3U3E
Brand : Stanzino
euclidean distance from input : 6.414901
==========
==========

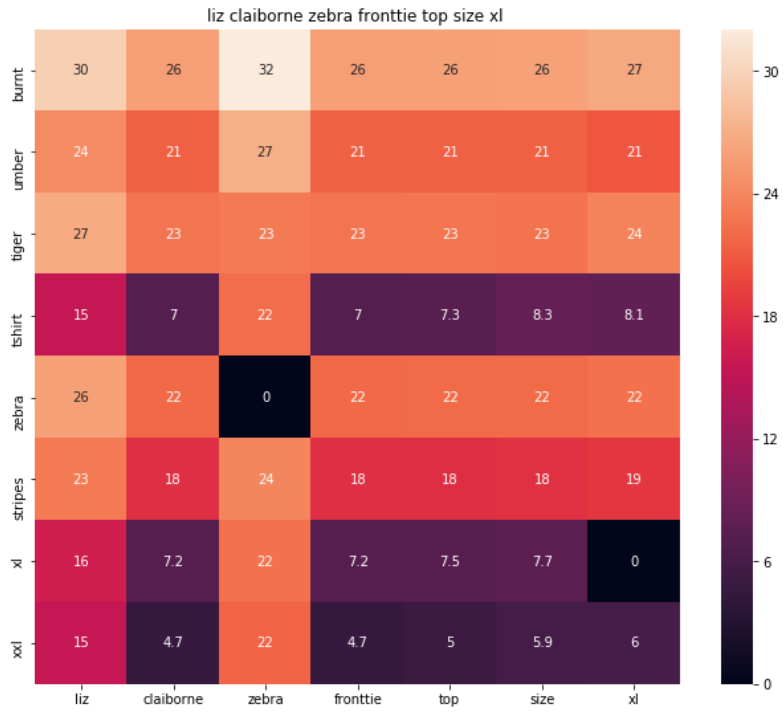


ASIN : B073R4ZM7Y
Brand : Colosseum
euclidean distance from input : 6.4509606
==========
==========



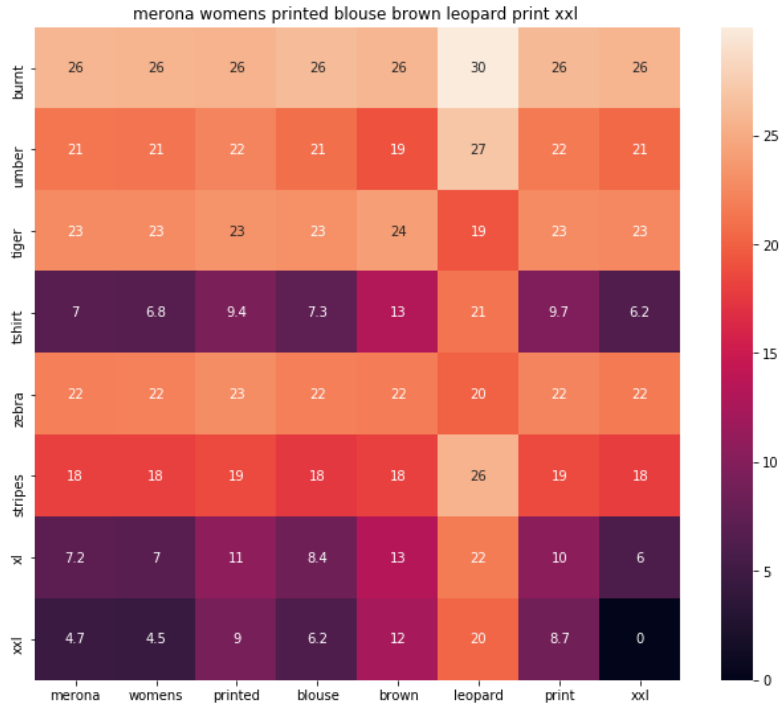
ASIN : B01C6ORLDQ
Brand : 1 Mad Fit
euclidean distance from input : 6.4634094

==========

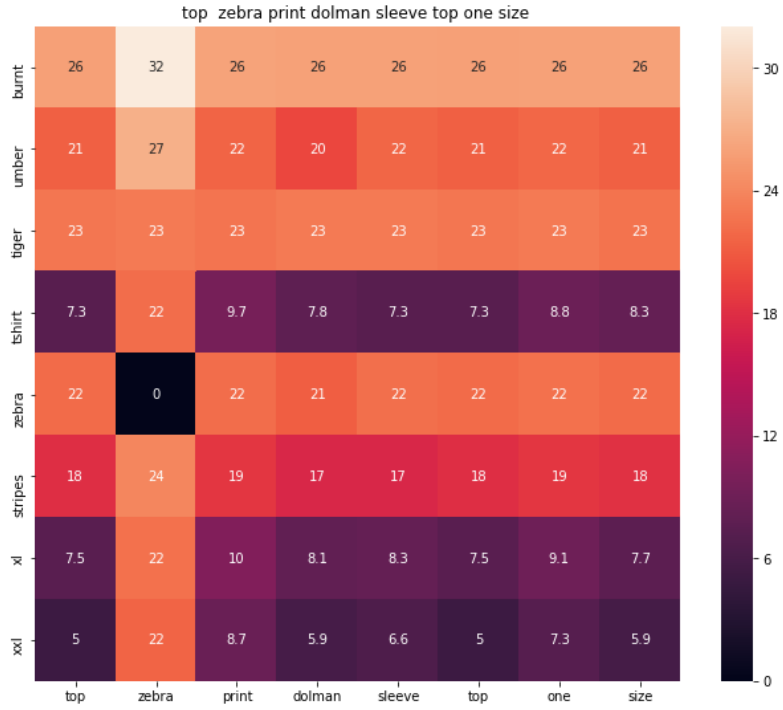


ASIN : B06XBY5QXL
Brand : Liz Claiborne
euclidean distance from input : 6.5392237

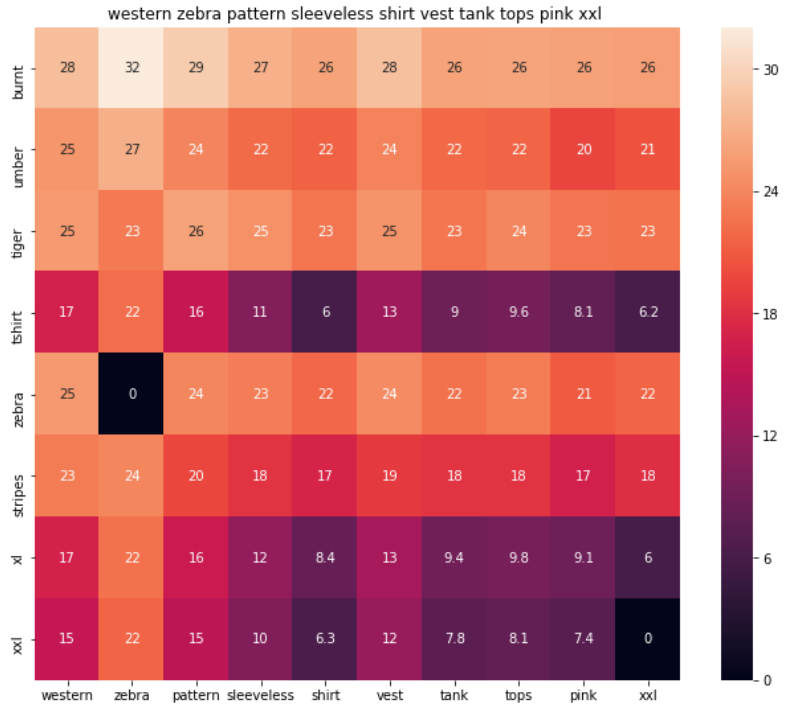
==========



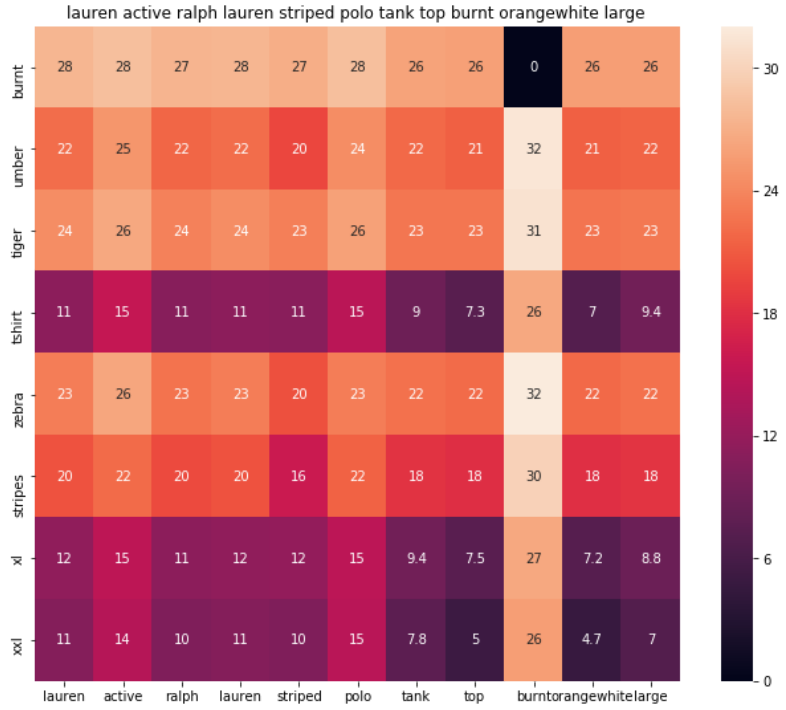
ASIN : B071YF3WDD
Brand : Merona
euclidean distance from input : 6.575504
==========
==========



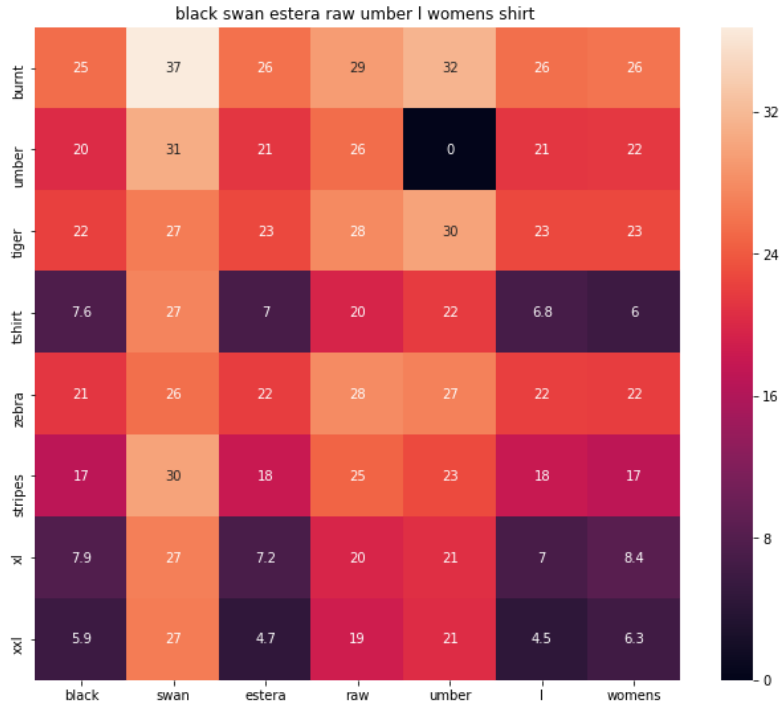
ASIN : B00H8A6ZLI
Brand : Vivian's Fashions
euclidean distance from input : 6.6382155
==========
==========



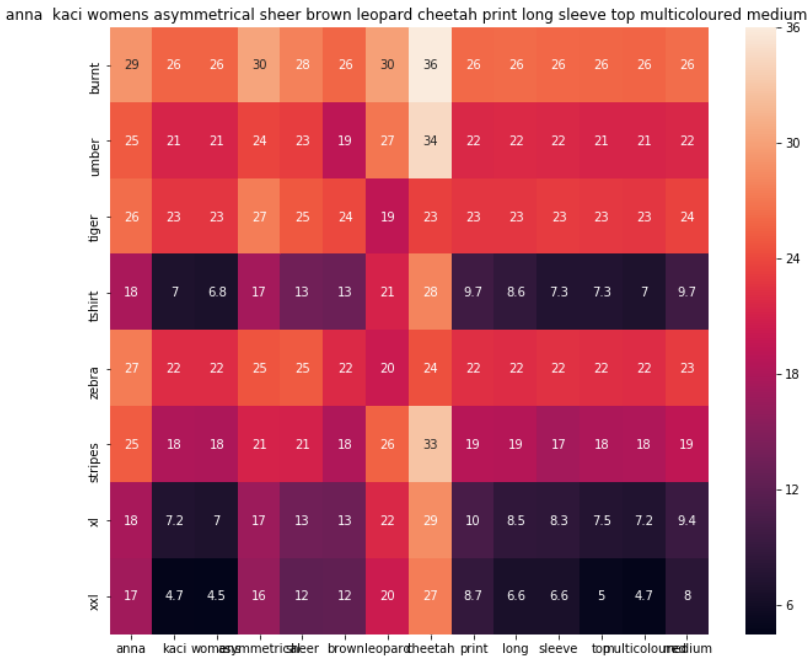
ASIN : B00Z6HEXWI
Brand : Black Temptation
euclidean distance from input : 6.660737
==========
==========



ASIN : B00ILGH5OY
Brand : Ralph Lauren Active
euclidean distance from input : 6.6839056
==========
==========



ASIN : B06Y1VN8WQ
Brand : Black Swan
euclidean distance from input : 6.7057643
==========
==========



ASIN : B00KSNTY7Y
Brand : Anna-Kaci
euclidean distance from input : 6.7061253
==========
==========

Keras and Tensorflow to extract features:

In [28]:

```
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
import PIL.Image
```

```
Using TensorFlow backend.
```

In [27]:

```
# https://gist.github.com/fchollet/f35fbc80e066a49d65f1688a7e99f069
# Code reference: https://blog.keras.io/building-powerful-image-classification-models-u
sing-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 codse takes 160 minutes to run on a high end i7 CPU
# CPU (i7): 0.615 seconds per image.

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

# each image is converted into 25088 length dense-vector


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

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


def save_bottlebeck_features():

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

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

    # build the VGG16 network
    model = applications.VGG16(include_top=False, weights='imagenet')
    generator = datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode=None,
        shuffle=False)

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

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

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


save_bottlebeck_features()

'''
```

Out[27]:

```
"\n# dimensions of our images.\nimg_width, img_height = 224, 224\n\ntop_mo
del_weights_path = 'bottleneck_fc_model.h5'\ntrain_data_dir = 'images2/'\n
nb_train_samples = 16042\nepochs = 50\nbatch_size = 1\n\n\ndef save_bottle
beck_features():\n    \n    #Function to compute VGG-16 CNN for image feat
ure extraction.\n    \n    asins = []\n    datagen = ImageDataGenerator(re
scale=1. / 255)\n    \n    # build the VGG16 network\n    model = applicat
ions.VGG16(include_top=False, weights='imagenet')\n    generator = datage
n.flow_from_directory(\n        train_data_dir,\n        target_size=(img_
width, img_height),\n        batch_size=batch_size,\n        class_mode=No
ne,\n        shuffle=False)\n\n    for i in generator.filenames:\n
asins.append(i[2:-5])\n\n    bottleneck_features_train = model.predict_gen
erator(generator, nb_train_samples // batch_size)\n    bottleneck_features
_train = bottleneck_features_train.reshape((16042,25088))\n    \n    np.sa
ve(open('16k_data_cnn_features.npy', 'wb'), bottleneck_features_train)\n
np.save(open('16k_data_cnn_feature_asins.npy', 'wb'), np.array(asins))\n
\n\nsave_bottlebeck_features()\n\n"
```

Visual features based product similarity.

In [34]:

```
# 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 datset
data = pd.read_pickle('pickels/16k_apperal_data_preprocessed')
df_asins = list(data['asin'])

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

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

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

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

get_similar_products_cnn(12566, 20)
```

Product Title: burnt umber tiger tshirt zebra stripes xl xxl
Euclidean Distance from input image: 0.044194173
Amazon Url: www.amzon.com/dp/B00JXQB5FQ

Product Title: pink tiger tshirt zebra stripes xl xxl
Euclidean Distance from input image: 30.050056
Amazon Url: www.amzon.com/dp/B00JXQASS6

Product Title: yellow tiger tshirt tiger stripes l
Euclidean Distance from input image: 41.261112
Amazon Url: www.amzon.com/dp/B00JXQCUIC

Product Title: brown white tiger tshirt tiger stripes xl xxl
Euclidean Distance from input image: 44.0002
Amazon Url: www.amzon.com/dp/B00JXQCWTO

Product Title: kawaii pastel tops tees pink flower design
Euclidean Distance from input image: 47.38251
Amazon Url: www.amzon.com/dp/B071FCWD97

Product Title: womens thin style tops tees pastel watermelon print
Euclidean Distance from input image: 47.71839
Amazon Url: www.amzon.com/dp/B01JUNHBRM

Product Title: kawaii pastel tops tees baby blue flower design
Euclidean Distance from input image: 47.9021
Amazon Url: www.amzon.com/dp/B071SBCY9W

Product Title: edv cheetah run purple multi xl
Euclidean Distance from input image: 48.046467
Amazon Url: www.amzon.com/dp/B01CUPYBM0

Product Title: danskin womens vneck loose performance tee xsmall pink omb
re
Euclidean Distance from input image: 48.101875
Amazon Url: www.amzon.com/dp/B01F7PHXY8

Product Title: summer alpaca 3d pastel casual loose tops tee design
Euclidean Distance from input image: 48.118896
Amazon Url: www.amzon.com/dp/B01I80A93G

Product Title: miss chievous juniors striped peplum tank top medium shado
wpeach
Euclidean Distance from input image: 48.13128
Amazon Url: www.amzon.com/dp/B0177DM70S

Product Title: red pink floral heel sleeveless shirt xl xxl
Euclidean Distance from input image: 48.16945
Amazon Url: www.amzon.com/dp/B00JV63QQE

Product Title: moana logo adults hot v neck shirt black xxl
Euclidean Distance from input image: 48.25678
Amazon Url: www.amzon.com/dp/B01LX6H43D

Product Title: abaday multicolor cartoon cat print short sleeve longline shirt large
Euclidean Distance from input image: 48.265644
Amazon Url: www.amzon.com/dp/B01CR57YY0

Product Title: kawaii cotton pastel tops tees peach pink cactus design
Euclidean Distance from input image: 48.362583
Amazon Url: www.amzon.com/dp/B071WYLBZS

Product Title: chicago chicago 18 shirt women pink
Euclidean Distance from input image: 48.383648
Amazon Url: www.amzon.com/dp/B01GXAZTRY

Product Title: yichun womens tiger printed summer tshirts tops
Euclidean Distance from input image: 48.449345
Amazon Url: www.amzon.com/dp/B010NN9RXO

Product Title: nancy lopez whimsy short sleeve whiteblacklemon drop xs
Euclidean Distance from input image: 48.478893
Amazon Url: www.amzon.com/dp/B01MPX6IDX

Product Title: womens tops tees pastel peach ice cream cone print
Euclidean Distance from input image: 48.557983
Amazon Url: www.amzon.com/dp/B0734GRKZL

Product Title: uswomens mary j blige without tshirts shirt
Euclidean Distance from input image: 48.614395
Amazon Url: www.amzon.com/dp/B01M0XXFKK

Weighted similarity using IDF features, brand, color and Image using VGG16.

In [29]:

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

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

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

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

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

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

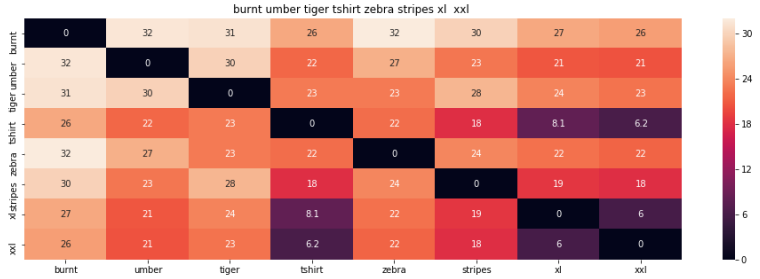
In [30]:

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

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

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

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

    colorscale = [[0, '#1d004d'],[.5, '#f2e5ff'],[1, '#f2e5d1']] # to color the heading
s 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])
    # ploting the heap map based on the pairwise distances
    ax1 = sns.heatmap(np.round(s1_s2_dist,6), annot=True)
    # set the x axis labels as recommended apparels title
    ax1.set_xticklabels(sentance2.split())
    # set the y axis labels as input apparels title
    ax1.set_yticklabels(sentance1.split())
    # set title as recommended apparels title
    ax1.set_title(sentance2)

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

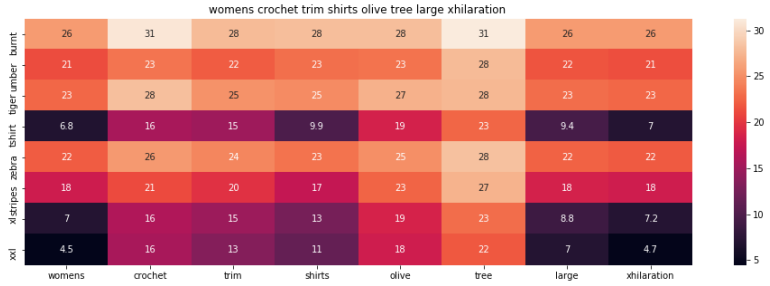
```
    # pass the url it display it
    display_img(url, ax2, fig)

    plt.show()
```

In [35]:

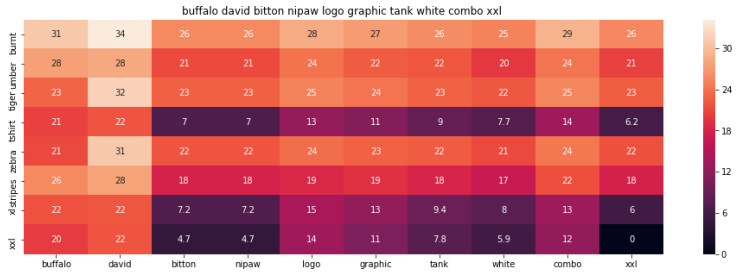
```
def idf_w2v_brand(doc_id, w1, w2, w3, num_results):
    # doc_id: apparel's id in given corpus
    # w1: weight for  w2v features
    # w2: weight for brand and color features

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

    pairwise_dist   = (w1 * idf_w2v_dist +  w2 * ex_feat_dist + w3 * img_dist)/float(w1
+ w2 + w3)

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

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


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

idf_w2v_brand(12566, 5, 5, 5, 20)
# in the give heat map, each cell contains the euclidean distance between words i, j
```



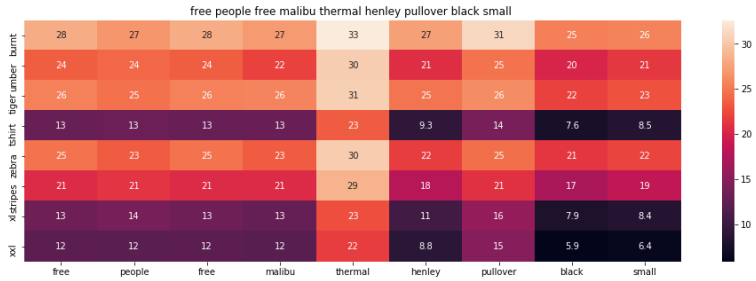
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 0.0013020833333333333
==========
==========



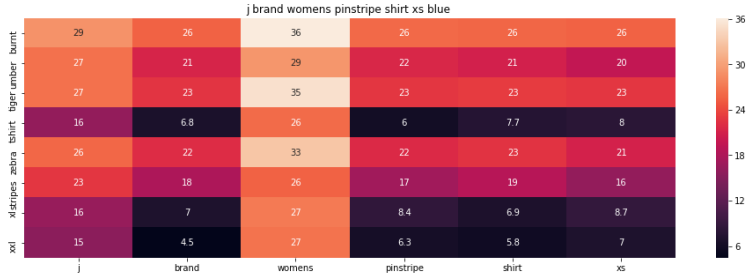
ASIN : B06XBHNM7J
Brand : Xhilaration
euclidean distance from input : 15.982596334062531
==========
==========



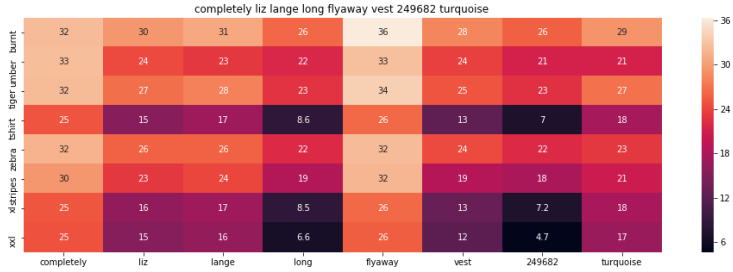
ASIN : B018H5AZXQ
Brand : Buffalo
euclidean distance from input : 16.60389938378584
==========
==========



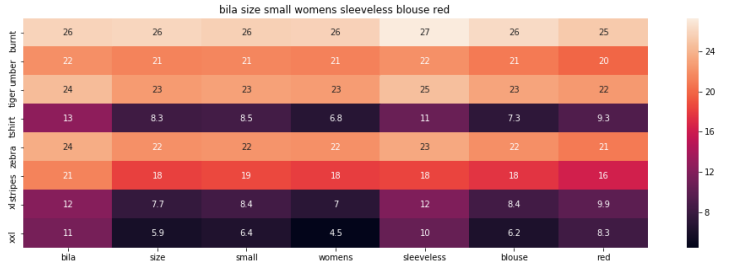
ASIN : B074MXY984
Brand : We The Free
euclidean distance from input : 16.9315793355306
==========
==========



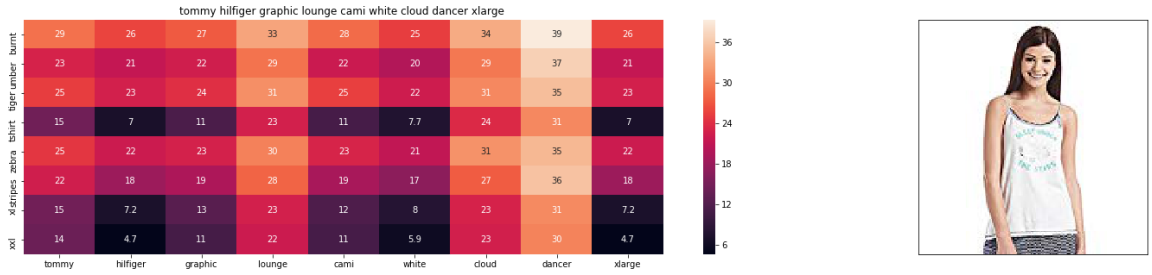
ASIN : B06XYP1X1F
Brand : J Brand Jeans
euclidean distance from input : 16.955509586267063
==========
==========



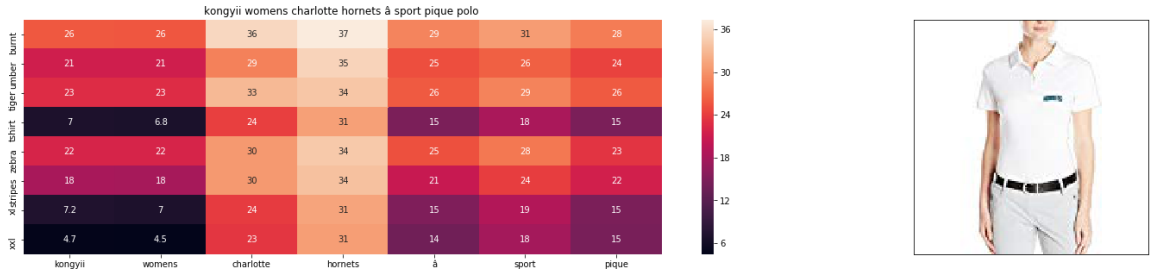
ASIN : B074LTBWSW
Brand : Liz Lange
euclidean distance from input : 16.957030995927767
==========
==========



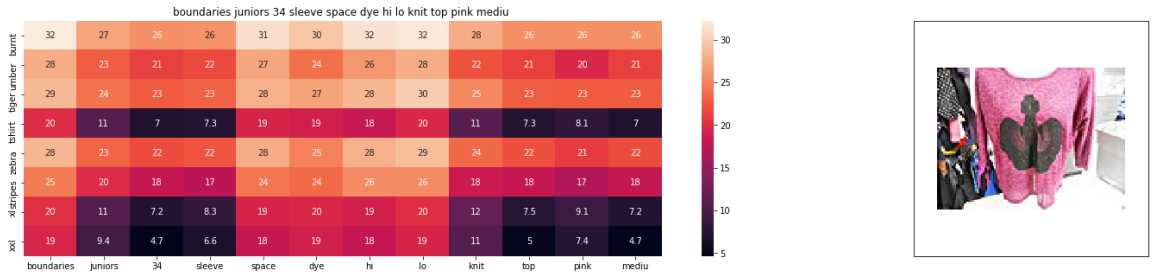
ASIN : B01L7ROZNC
Brand : Bila
euclidean distance from input : 17.094785603444056
==========
==========



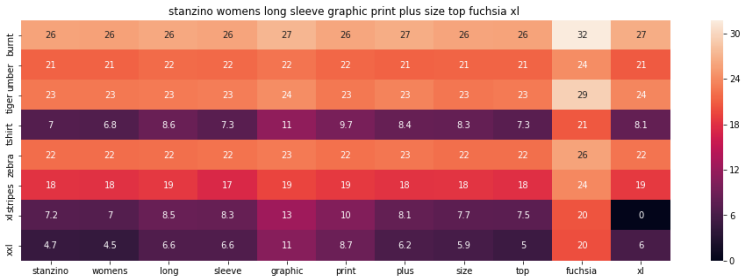
ASIN : B01BMSFYW2
Brand : igertommy hilf
euclidean distance from input : 17.28112604134519
==========
==========



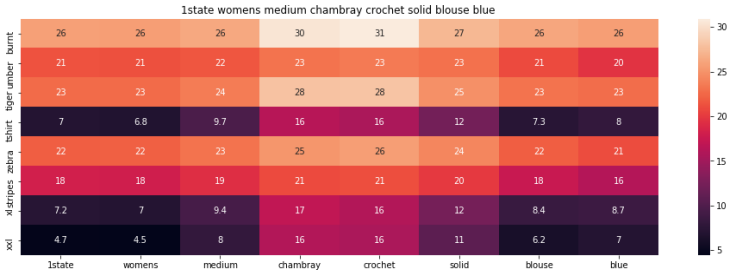
ASIN : B01FJVZST2
Brand : KONGYII
euclidean distance from input : 17.551688170989628
==========
==========



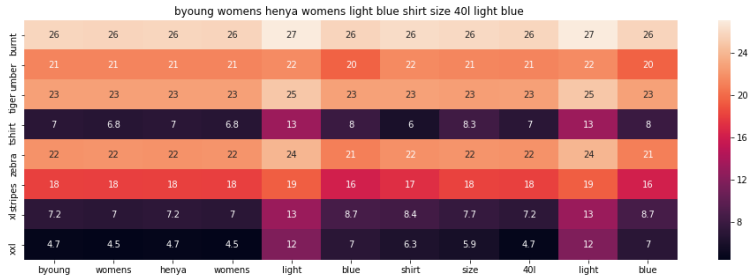
ASIN : B01EXXFS4M
Brand : No Boundaries
euclidean distance from input : 17.564531453691437
==========
==========



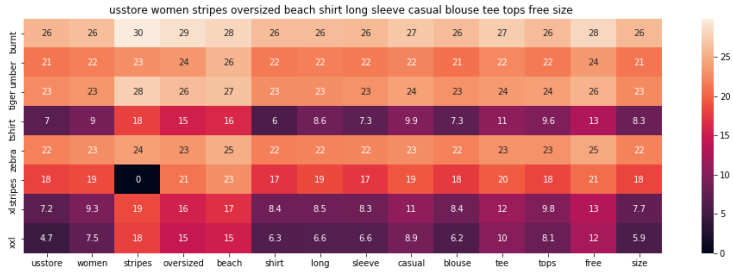
ASIN : B00DP4VHWI
Brand : Stanzino
euclidean distance from input : 17.585476724863327
==========
==========



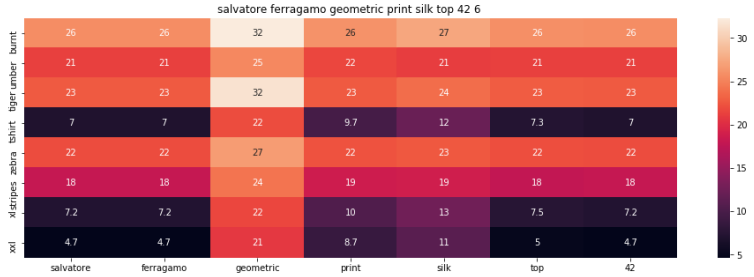
ASIN : B074MK6LV2
Brand : 1.State
euclidean distance from input : 17.59606810466476
==========
==========



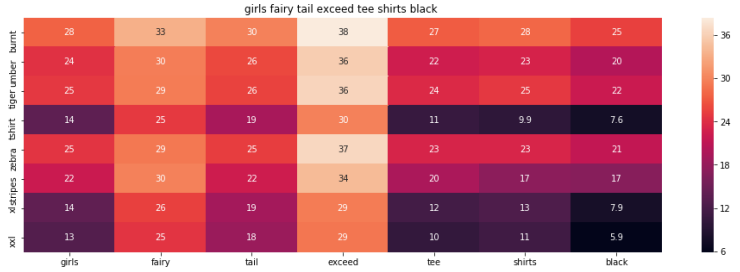
ASIN : B06Y41MRCH
Brand : Byoung
euclidean distance from input : 17.624772135657583
==========
==========



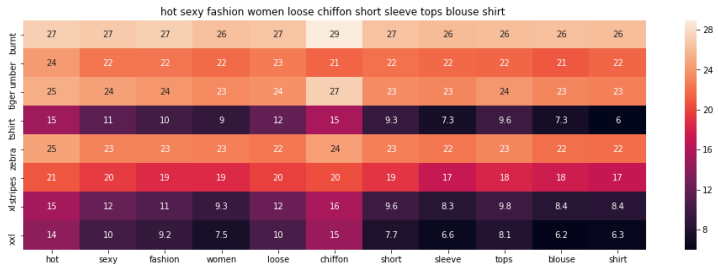
ASIN : B01DNNI1RO
Brand : Usstore
euclidean distance from input : 17.65560150170576
==========
==========



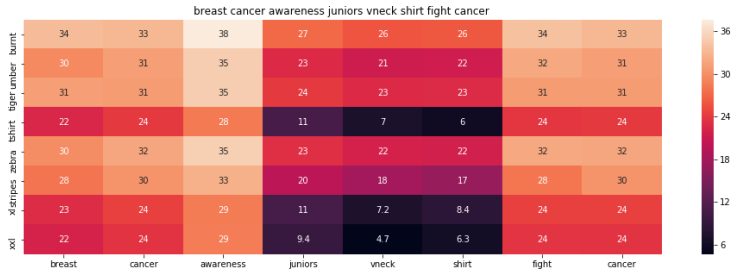
ASIN : B0756JTS1F
Brand : Salvatore Ferragamo
euclidean distance from input : 17.681495666503906
==========
==========



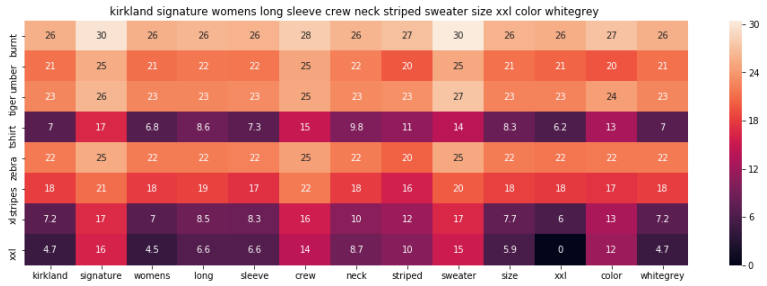
ASIN : B01L9F153U
Brand : ATYPEMX
euclidean distance from input : 17.69363432780929
==========
==========



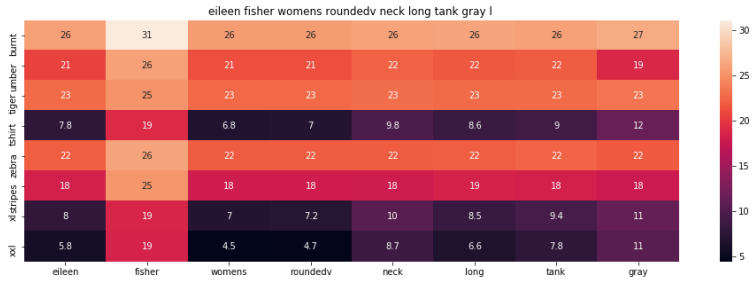
ASIN : B00JMAASRO
Brand : Wotefusi
euclidean distance from input : 17.70576048747726
==========
==========



ASIN : B016CU40IY
Brand : Juiceclouds
euclidean distance from input : 17.709572641611373
==========
==========



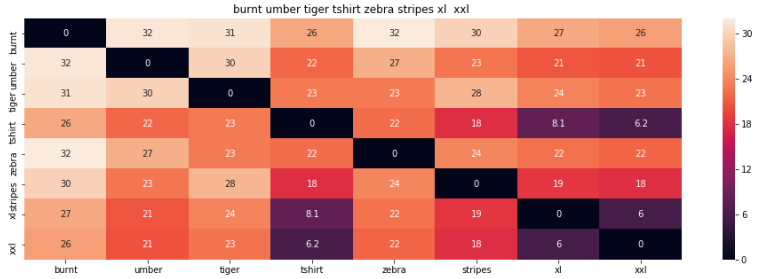
ASIN : B06XTPC3FP
Brand : Kirkland Signature
euclidean distance from input : 17.753979238115267
==========
==========



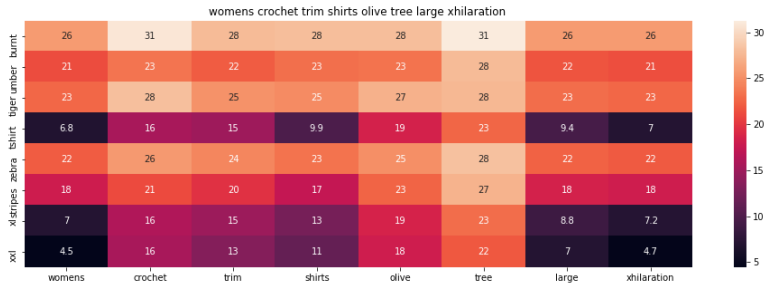
ASIN : B01N2JSRDM
Brand : Eileen Fisher
euclidean distance from input : 17.801957957190787
==========
==========

In [36]:

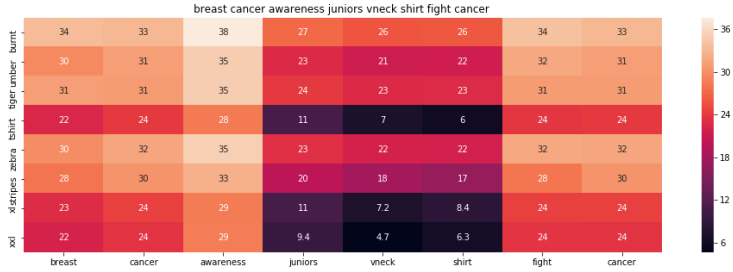
```
# brand and color weight =5
# title vector weight = 5
# image = 50

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



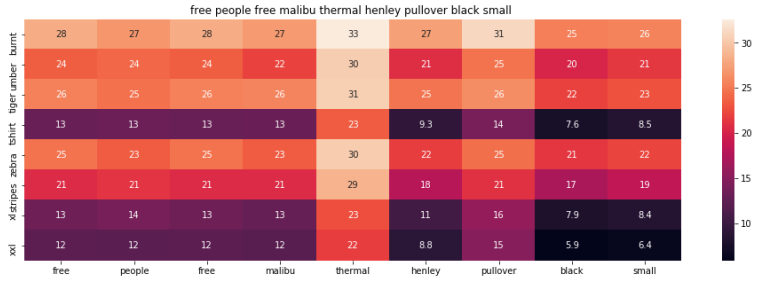
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 0.0003255208333333333
==========
==========



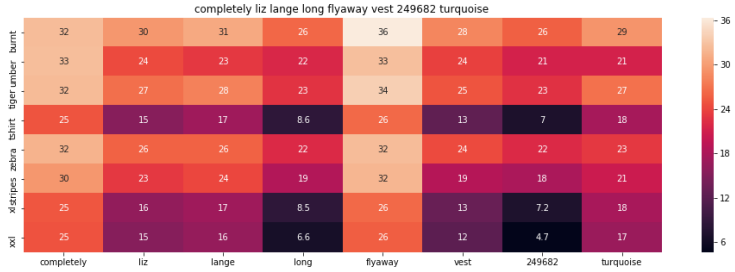
ASIN : B06XBHNM7J
Brand : Xhilaration
euclidean distance from input : 31.88651212062501
==========
==========



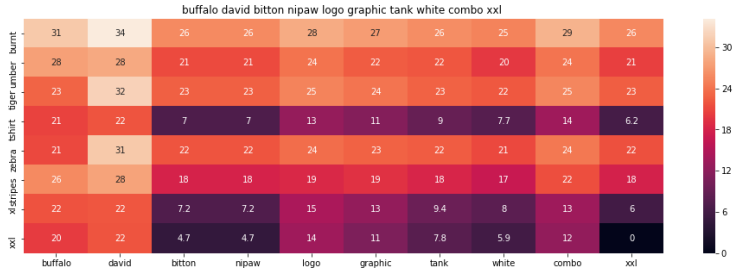
ASIN : B016CU40IY
Brand : Juiceclouds
euclidean distance from input : 33.162255424400236
==========
==========



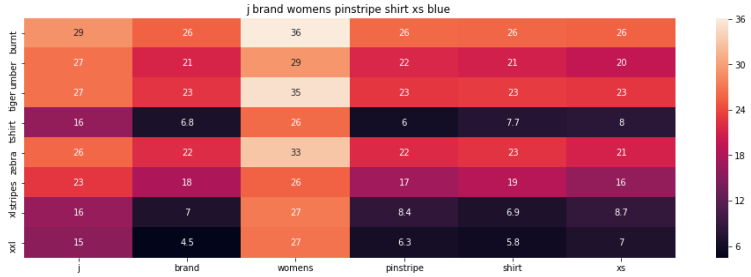
ASIN : B074MXY984
Brand : We The Free
euclidean distance from input : 33.75603326161703
==========
==========



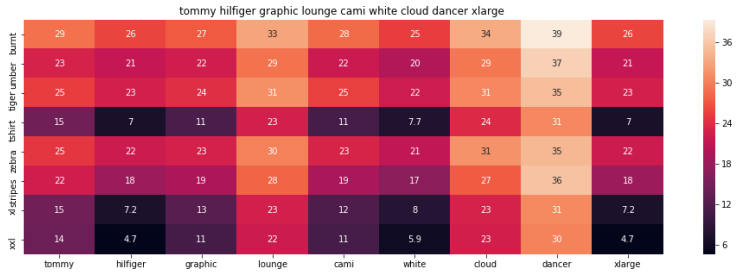
ASIN : B074LTBWSW
Brand : Liz Lange
euclidean distance from input : 33.762396176716315
==========
==========



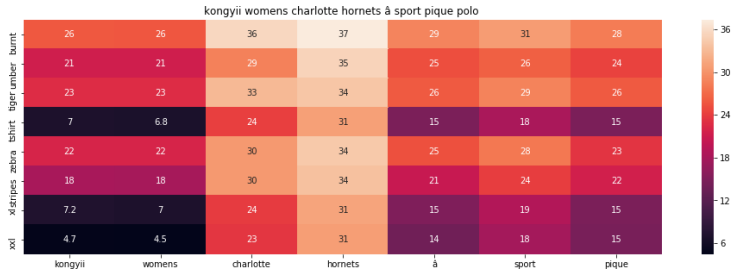
ASIN : B018H5AZXQ
Brand : Buffalo
euclidean distance from input : 33.82252559667889
==========
==========



ASIN : B06XYP1X1F
Brand : J Brand Jeans
euclidean distance from input : 33.93180393058369
==========
==========

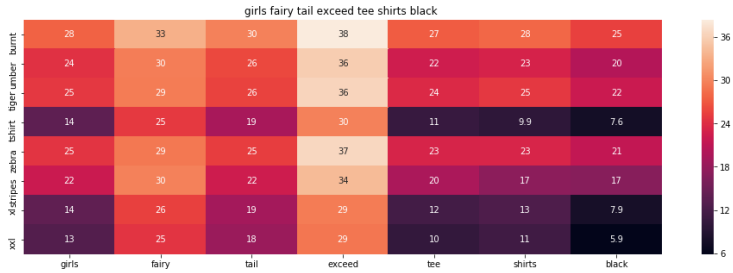


ASIN : B01BMSFYW2
Brand : igertommy hilf
euclidean distance from input : 34.12049869376729
==========
==========

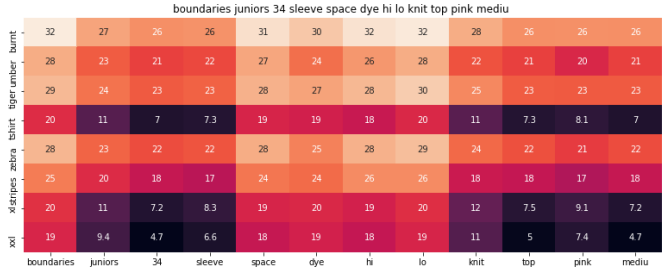


ASIN : B01FJVZST2
Brand : KONGYII
euclidean distance from input : 34.933592615664075
==========
==========

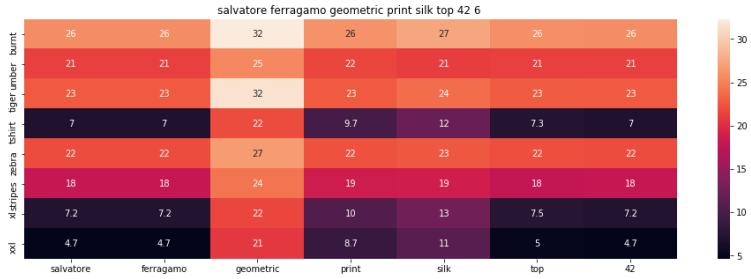
ASIN : B01L7ROZNC
Brand : Bila
euclidean distance from input : 35.05102285756674
==========
==========



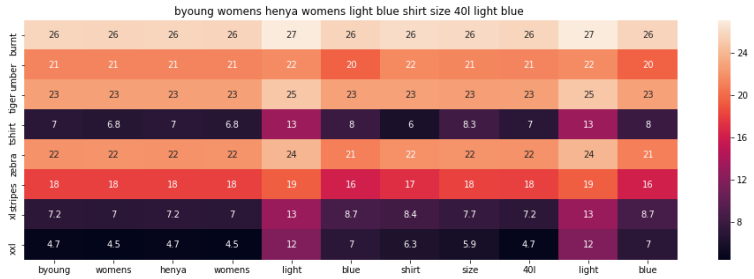
ASIN : B01L9F153U
Brand : ATYPEMX
euclidean distance from input : 35.61336702718344
==========
==========



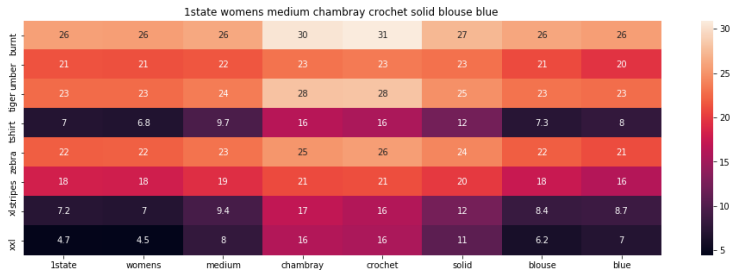
ASIN : B01EXXFS4M
Brand : No Boundaries
euclidean distance from input : 35.658698527078464
==========
==========



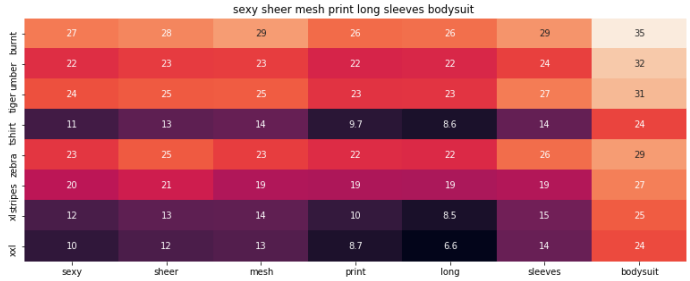
ASIN : B0756JTS1F
Brand : Salvatore Ferragamo
euclidean distance from input : 35.8234561920166
==========
==========



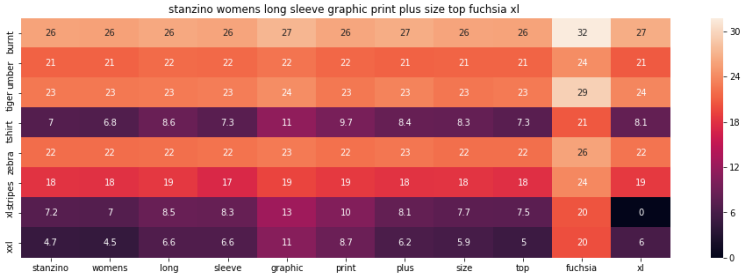
ASIN : B06Y41MRCH
Brand : Byoung
euclidean distance from input : 35.8665211996045
==========
==========



ASIN : B074MK6LV2
Brand : 1.State
euclidean distance from input : 35.93611775451905
==========
==========

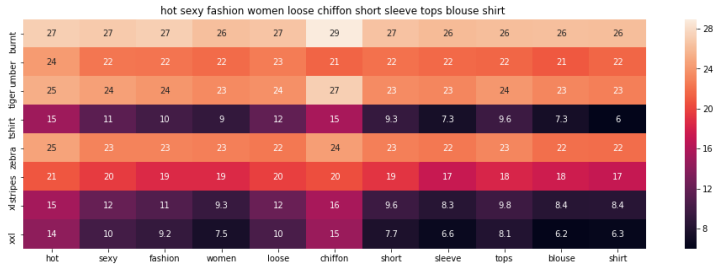


ASIN : B074Z5C98D
Brand : Ariella's closet
euclidean distance from input : 36.04981145510449
==========
==========

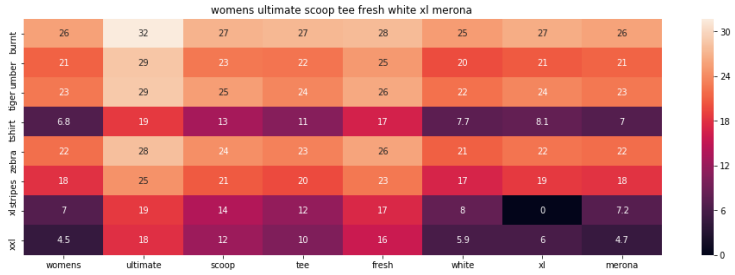


ASIN : B00DP4VHWI
Brand : Stanzino
euclidean distance from input : 36.17867763255047
==========
==========

ASIN : B01M8GB3AL
Brand : Maven West
euclidean distance from input : 36.21196498876873
==========
==========



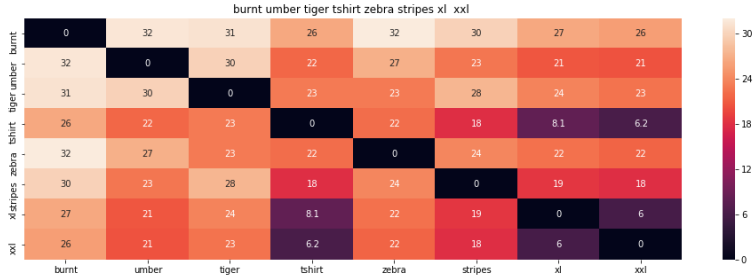
ASIN : B00JMAASRO
Brand : Wotefusi
euclidean distance from input : 36.23066985819744
==========
==========



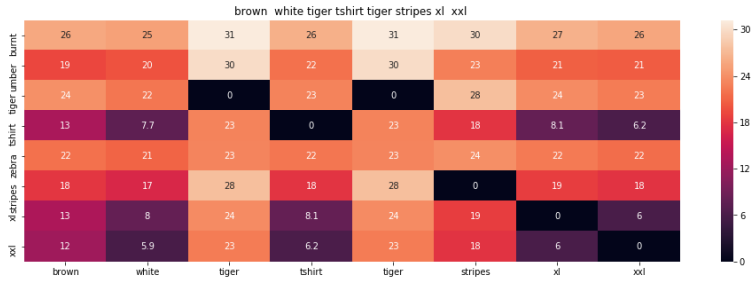
ASIN : B01G7XE50E
Brand : Merona
euclidean distance from input : 36.25837619517254
==========
==========

In [37]:

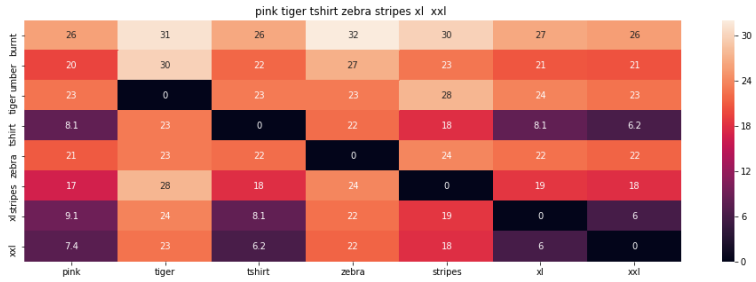
```
# brand and color weight =50
# title vector weight = 5
# image = 5

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



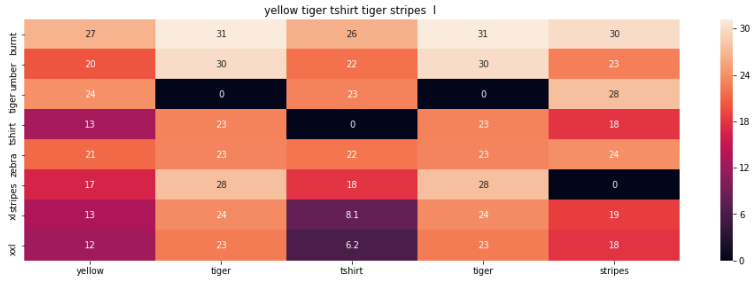
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 0.0003255208333333333
==========
==========



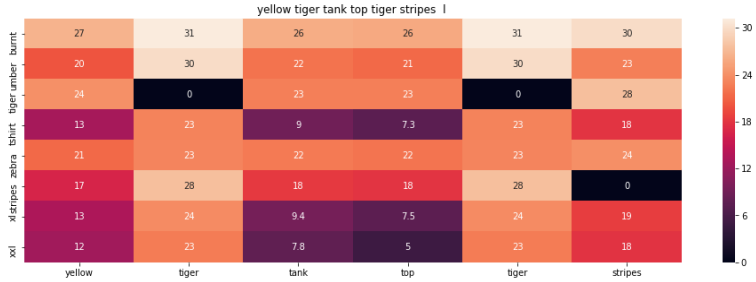
ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 5.026570224761963
==========
==========



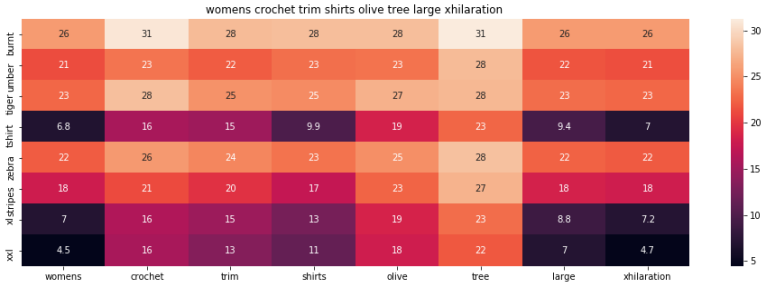
ASIN : B00JXQASS6
Brand : Si Row
euclidean distance from input : 5.559655698441609
==========
==========



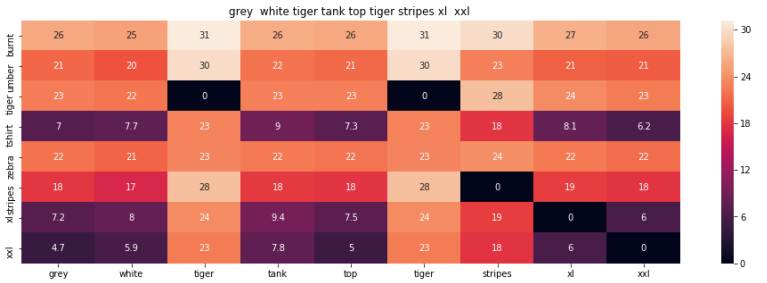
ASIN : B00JXQCUIC
Brand : Si Row
euclidean distance from input : 6.0177165352336015
==========
==========



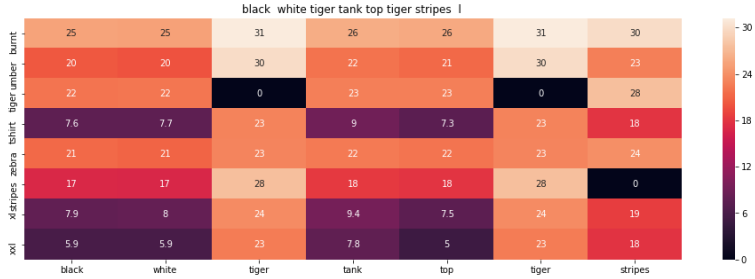
ASIN : B00JXQAUWA
Brand : Si Row
euclidean distance from input : 6.1092127485107826
==========
==========



ASIN : B06XBHNM7J
Brand : Xhilaration
euclidean distance from input : 6.116969427075276
==========
==========

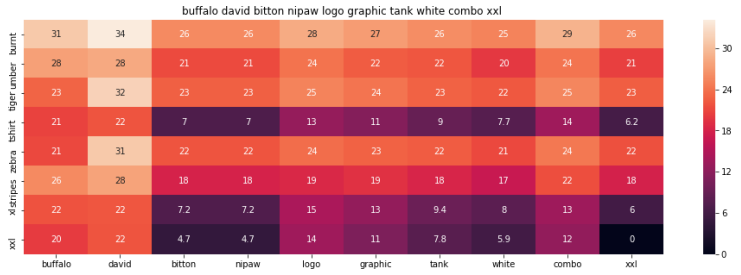


ASIN : B00JXQAFZ2
Brand : Si Row
euclidean distance from input : 6.119070085190876
==========
==========

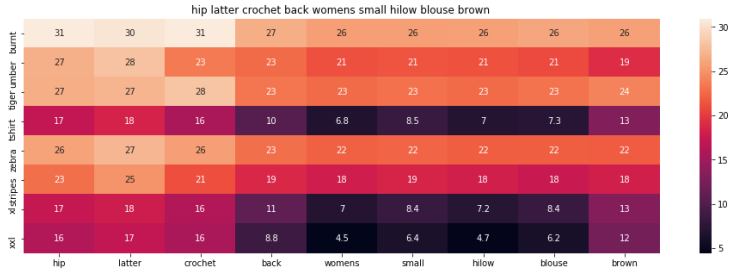


ASIN : B00JXQAO94
Brand : Si Row
euclidean distance from input : 6.255372810664917
==========
==========

ASIN : B01L7ROZNC
Brand : Bila
euclidean distance from input : 6.258009884159458
==========
==========

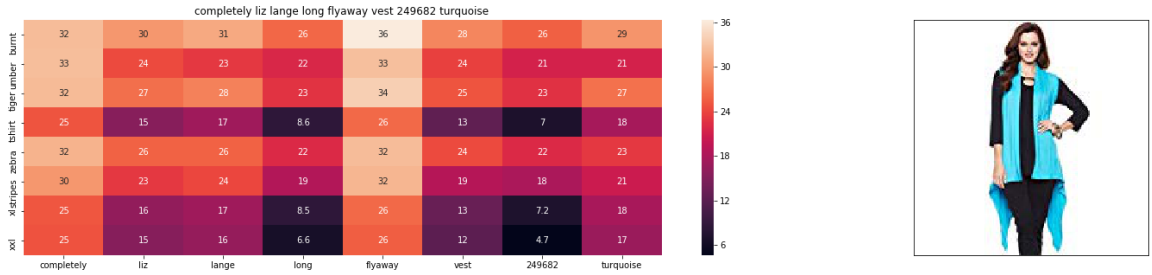


ASIN : B018H5AZXQ
Brand : Buffalo
euclidean distance from input : 6.272295189506102
==========
==========

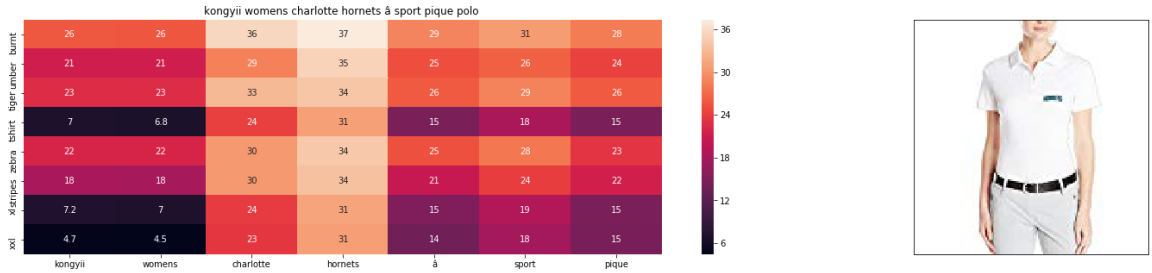


ASIN : B074MJN1K9
Brand : Hip
euclidean distance from input : 6.350509528775867
==========
==========

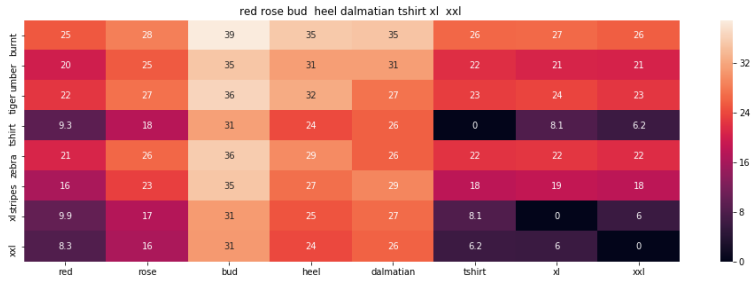
ASIN : B06XK2ZRFH
Brand : Acquaa
euclidean distance from input : 6.35436471027423
==========
==========



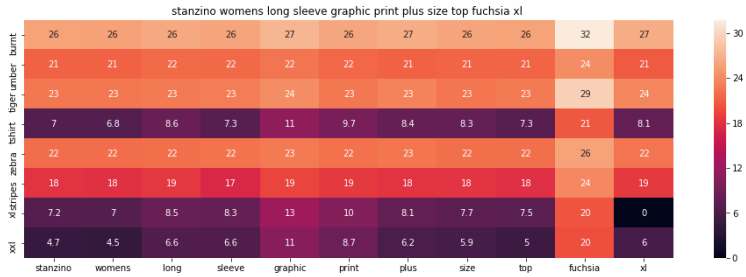
ASIN : B074LTBWSW
Brand : Liz Lange
euclidean distance from input : 6.3605780925415845
==========
==========



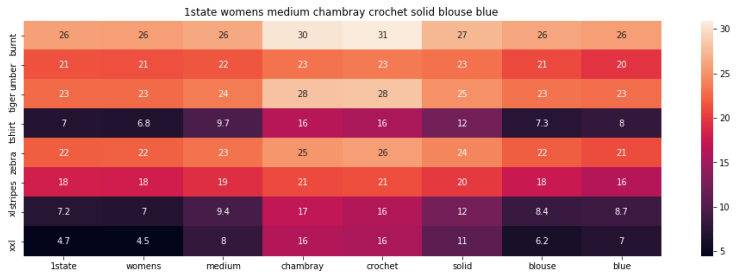
ASIN : B01FJVZST2
Brand : KONGYII
euclidean distance from input : 6.372235526045851
==========
==========



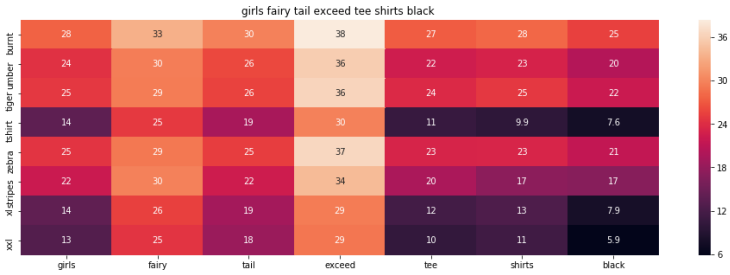
ASIN : B00JXQABB0
Brand : Si Row
euclidean distance from input : 6.380368360184773
==========
==========



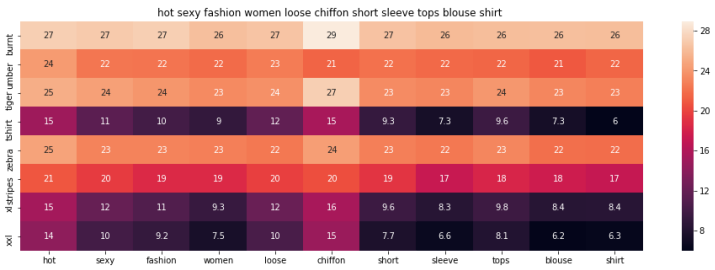
ASIN : B00DP4VHWI
Brand : Stanzino
euclidean distance from input : 6.380682664514276
==========
==========



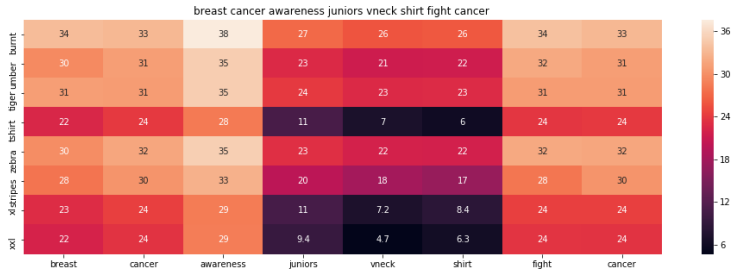
ASIN : B074MK6LV2
Brand : 1.State
euclidean distance from input : 6.383330509464633
==========
==========



ASIN : B01L9F153U
Brand : ATYPEMX
euclidean distance from input : 6.407722065250766
==========
==========



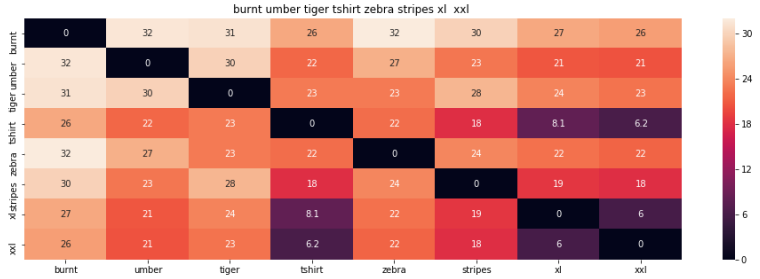
ASIN : B00JMAASRO
Brand : Wotefusi
euclidean distance from input : 6.410753605167758
==========
==========



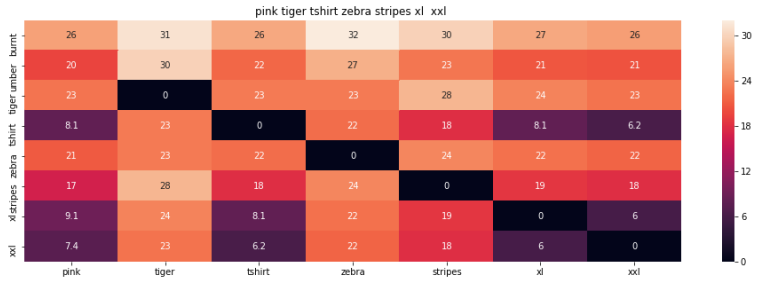
ASIN : B016CU40IY
Brand : Juiceclouds
euclidean distance from input : 6.4117066437012875
==========
==========

In [38]:

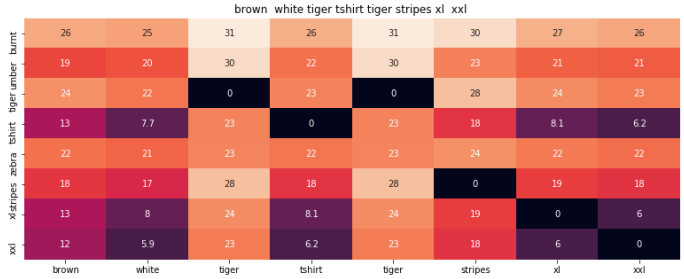
```
# brand and color weight =5
# title vector weight = 50
# image = 5

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



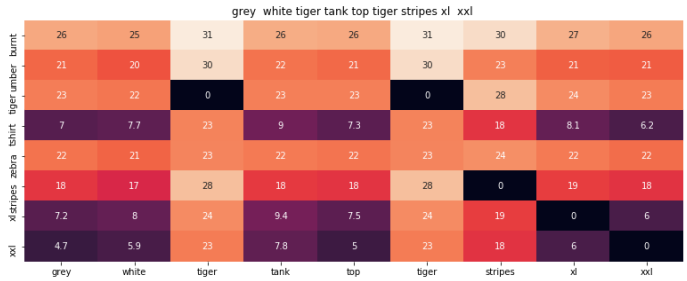
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 0.0032552083333333335
==========
==========



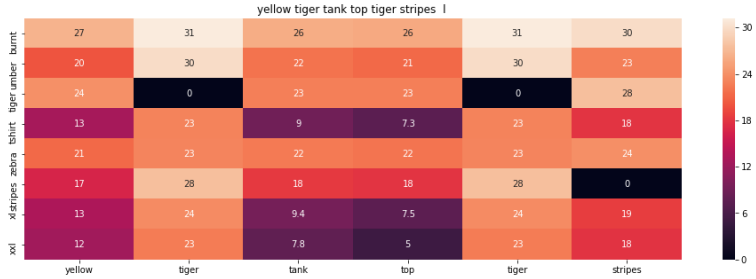
ASIN : B00JXQASS6
Brand : Si Row
euclidean distance from input : 7.546911239654138
==========
==========



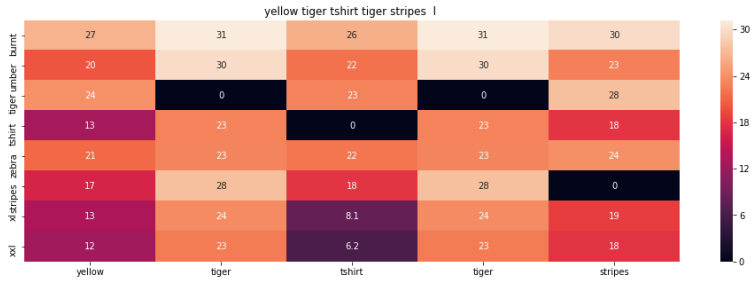
ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 8.604776763916016
==========
==========



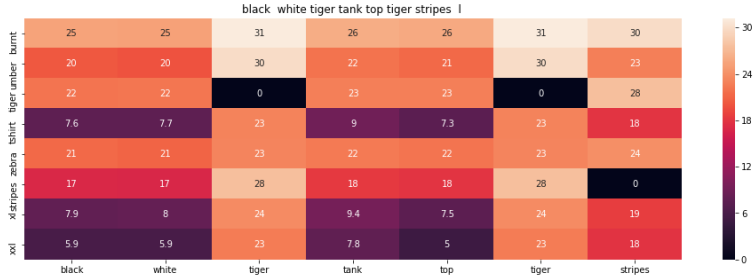
ASIN : B00JXQAFZ2
Brand : Si Row
euclidean distance from input : 9.078530756662603
==========
==========



ASIN : B00JXQAUWA
Brand : Si Row
euclidean distance from input : 9.315695063303227
==========
==========

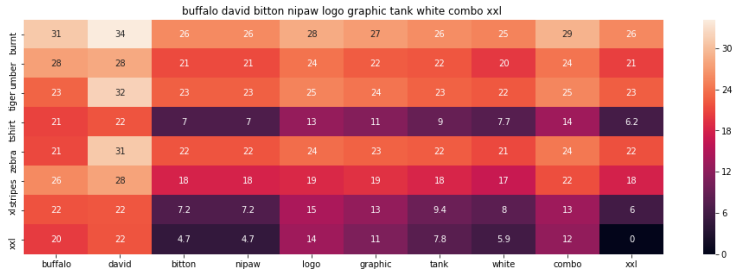


ASIN : B00JXQCUIC
Brand : Si Row
euclidean distance from input : 9.3771381378475
==========
==========

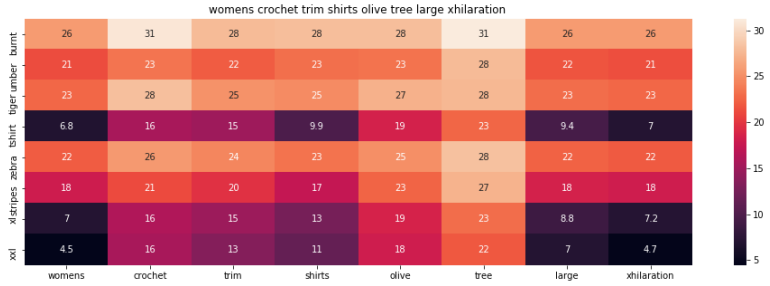


ASIN : B00JXQAO94
Brand : Si Row
euclidean distance from input : 9.46447944644125
==========
==========

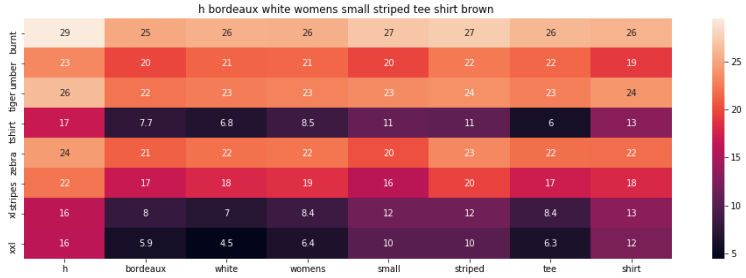
ASIN : B073R5Q8HD
Brand : Colosseum
euclidean distance from input : 9.56130264017986
==========
==========



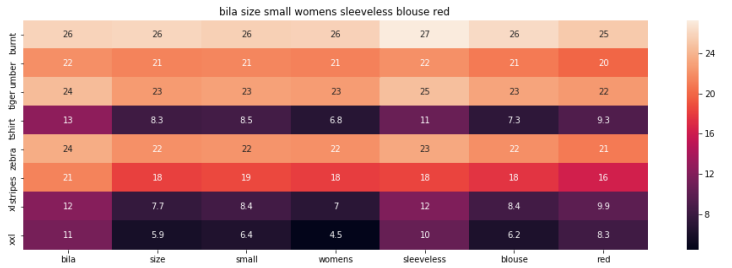
ASIN : B018H5AZXQ
Brand : Buffalo
euclidean distance from input : 9.716875712137051
==========
==========



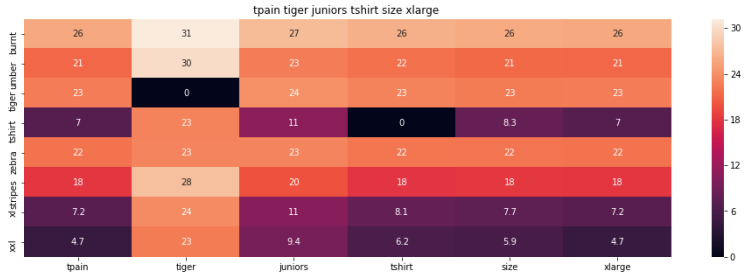
ASIN : B06XBHNM7J
Brand : Xhilaration
euclidean distance from input : 9.944307708800464
==========
==========



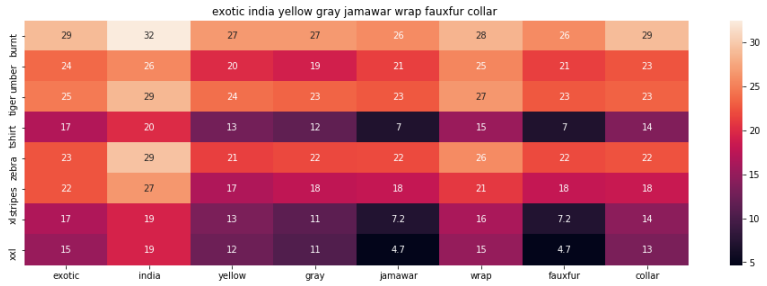
ASIN : B072BVB47Z
Brand : H By Bordeaux
euclidean distance from input : 9.95611821464925
==========
==========



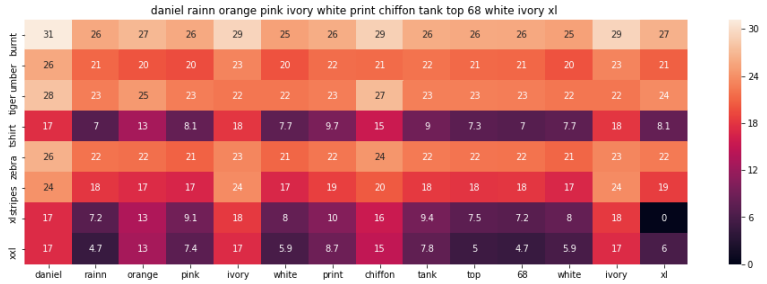
ASIN : B01L7ROZNC
Brand : Bila
euclidean distance from input : 9.975321906944183
==========
==========



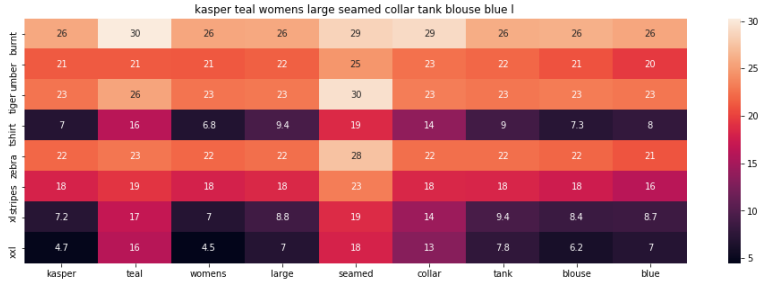
ASIN : B01K0H02OG
Brand : Tultex
euclidean distance from input : 9.983424832562672
==========
==========



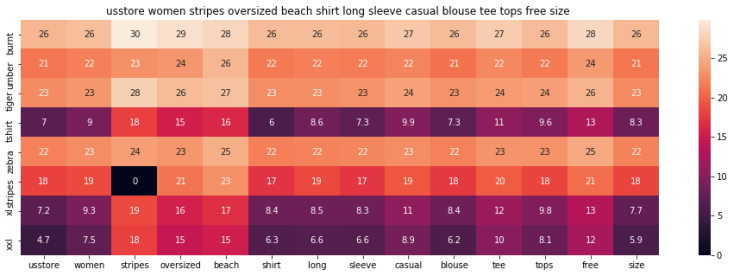
ASIN : B073ZHRBV8
Brand : Exotic India
euclidean distance from input : 10.01042048142416
==========
==========



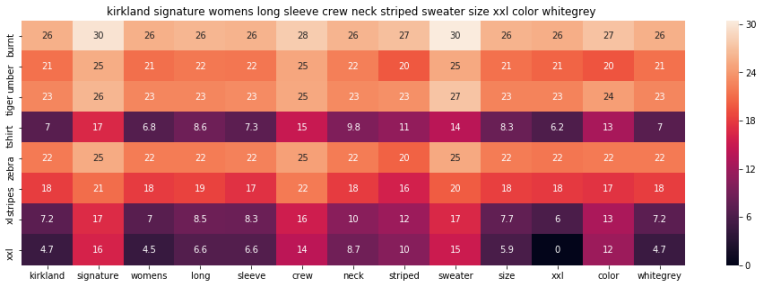
ASIN : B01IPV1SFQ
Brand : Daniel Rainn
euclidean distance from input : 10.047508748372396
==========
==========



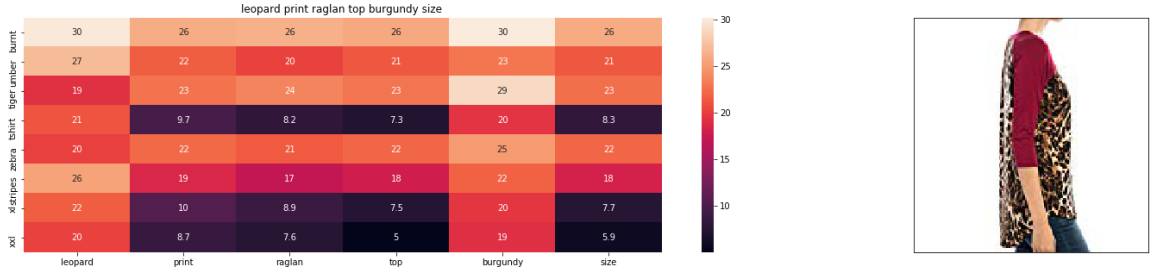
ASIN : B0722DJVQP
Brand : Kasper
euclidean distance from input : 10.065923510134288
==========
==========



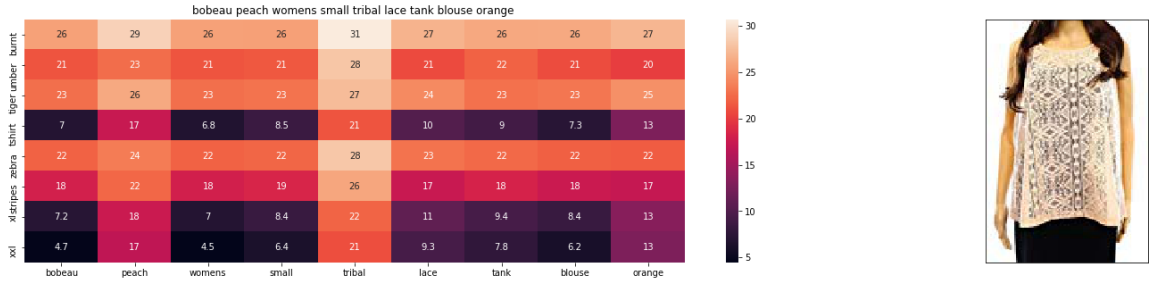
ASIN : B01DNNI1RO
Brand : Usstore
euclidean distance from input : 10.076903025369473
==========
==========



ASIN : B06XTPC3FP
Brand : Kirkland Signature
euclidean distance from input : 10.082541910867521
==========
==========



ASIN : B01C6ORLDQ
Brand : 1 Mad Fit
euclidean distance from input : 10.118006642719735
==========
==========



ASIN : B072JTHCX6
Brand : Bobeau
euclidean distance from input : 10.123612922569183
==========
==========

Observation and Steps followed :

1. First we have tried out some EDA. We selected 7 features out of 19 due to lack of computation power.
2. We striked out the missing values in the columns of various features.
3. Most coomon product type name is shirt which is significantly denser than other counterparts.
4. Then we checked statistics of formatted price, brand and color.
5. We eliminated the duplicate items by their title and also similar semantic titles.
6. Lastly we got 16k points after all the preprocessing and we saved the data in pickle file.
7. Next we tried text preprocessing where we removed stopwords from the title features and stemmed the words.
8. We tried product similarity using BOW vectorization and our measurement metric is euclidian distance. Lesser the eucladian distance bettre the similarity.
9. We tried tfidf vectorization which resulted further good result.
10. Similarly we tried the semantic featurization like avg word2vec .
11. We have encoded the color, brand features using one-hot encoding.
12. Next we tried out similarity of images using CNN (VGG 16 engine).
13. Finally we combined idf featurization on title feature, color, brand, image features using hstack.
14. And observed the similarity assigning weight to each vectorizations.
15. We caliberated different values to the input of similarity function. The model behaved very well as per our weight assignment.