

Amazon Apparel Recommendations



[4.3] Overview of the data

In [1]:

```
#import all the necessary packages.

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

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

In [2]:

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

In [3]:

```
print ('Number of data points : ', data.shape[0], \  
      'Number of features/variables:', data.shape[1])
```

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

Terminology:

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

In [4]:

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

Out[4]:

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

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

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

In [5]:

```
data = data[['asin', 'brand', 'color', 'medium_image_url', 'product_type_name', 'ti
```

In [6]:

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

Number of data points : 183138 Number of features: 7

Out[6]:

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

[5.1] Missing data for various features.

Basic stats for the feature: product_type_name

In [7]:

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

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

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

In [8]:

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

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

In [9]:

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

Out[9]:

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

Basic stats for the feature: brand

In [10]:

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

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

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

In [11]:

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

Out[11]:

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

Basic stats for the feature: color

In [12]:

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

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

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

In [13]:

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

Out[13]:

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

Basic stats for the feature: formatted_price

In [14]:

```
print(data['formatted_price'].describe())
# Only 28,395 (15.5% of whole data) products with price information
```

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

In [15]:

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

Out[15]:

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

Basic stats for the feature: title

In [16]:

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

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

6. Text pre-processing

In [37]:

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

In [38]:

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

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

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

In [39]:

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

5.402977 seconds

In [40]:

```
data.shape
```

Out[40]:

(16042, 7)

In [41]:

```
data.head()
```

Out[41]:

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

In [42]:

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

Stemming

In [43]:

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

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

argu
fish

[8] Text based product similarity

In [44]:

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

Out[44]:

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

◀ ▶

In [45]:

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

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

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

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

    # 1st, plotting heat map that represents the count of commonly occurred words
    ax = plt.subplot(gs[0])
    # it displays a cell in white color if the word is intersection(lis of word
    ax = sns.heatmap(np.array([values]), annot=np.array([labels]))
    ax.set_xticklabels(keys) # set that axis labels as the words of title
    ax.set_title(text) # apparel title

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

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

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

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

    # doc_id : index of the title1
    # vec1 : input 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
    intersection = set(vec1.keys()) & set(vec2.keys())
```

```

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

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

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

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

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

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

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

    # vector1 = dict{word1:#count, word2:#count, etc.}
    vector1 = text_to_vector(text1)

    # vector2 = dict{word1:#count, word2:#count, etc.}
    vector2 = text_to_vector(text2)

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

```

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

In [46]:

```
from sklearn.feature_extraction.text import CountVectorizer
title_vectorizer = CountVectorizer()
title_features = title_vectorizer.fit_transform(data['title'])
title_features.get_shape() # get number of rows and columns in feature matrix.
```

Out[46]:

(16042, 12609)

In [47]:

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

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

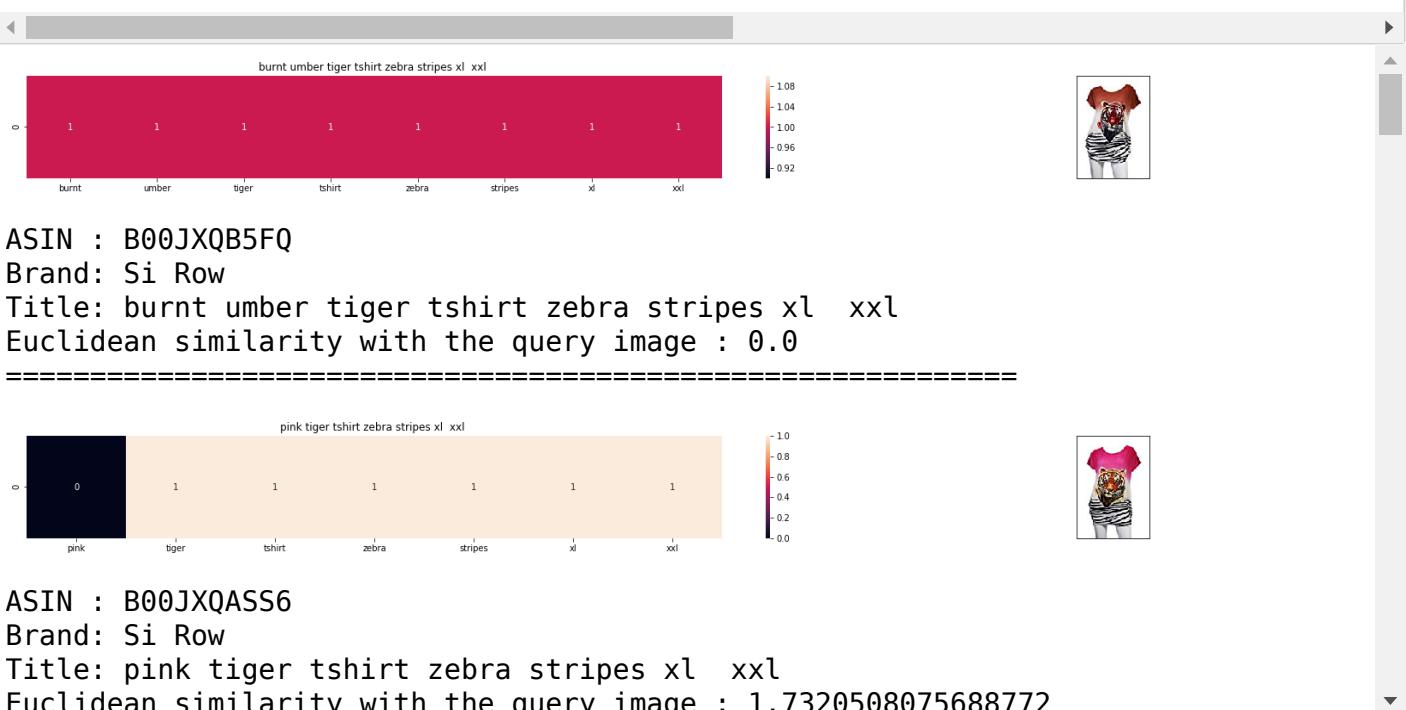
    # np.argsort will return indices of the smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the smallest distances
    pdists = np.sort(pairwise_dist.flatten())[0:num_results]

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

    for i in range(0, len(indices)):
        # we will pass 1. doc_id, 2. title1, 3. title2, url, model
        get_result(indices[i], data['title'].loc[df_indices[0]], data['title'].loc[df_indices[i]])
        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
```



[8.5] TF-IDF based product similarity

In [48]:

```
tfidf_title_vectorizer = TfidfVectorizer(min_df = 0)
tfidf_title_features = tfidf_title_vectorizer.fit_transform(data['title'])
# tfidf_title_features.shape = #data_points * #words_in_corpus
# CountVectorizer().fit_transform(courpus) returns the a sparase matrix of dimension
# tfidf_title_features[doc_id, index_of_word_in_corpus] = tfidf values of the word
```

In [49]:

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

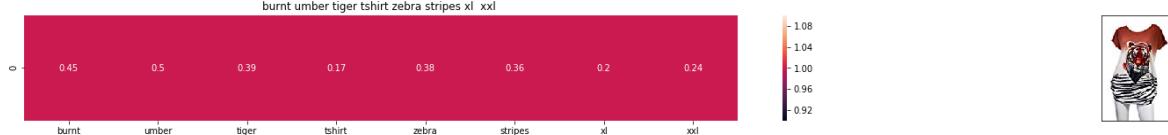
    # pairwise_dist will store the distance from given input apparel to all remaining
    # the metric we used here is cosine, the cosine distance is measured as K(X, Y)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(tfidf_title_features, tfidf_title_features[do

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

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

    for i in range(0,len(indices)):
        # we will pass 1. doc_id, 2. title1, 3. title2, url, model
        get_result(indices[i], data['title'].loc[df_indices[0]], data['title'].loc[
        print('ASIN :', data['asin'].loc[df_indices[i]])
        print('BRAND :', data['brand'].loc[df_indices[i]])
        print('Eucliden distance from the given image :', pdists[i])
        print('*'*125)
tfidf_model(12566, 20)
# in the output heat map each value represents the tfidf values of the label word,
```

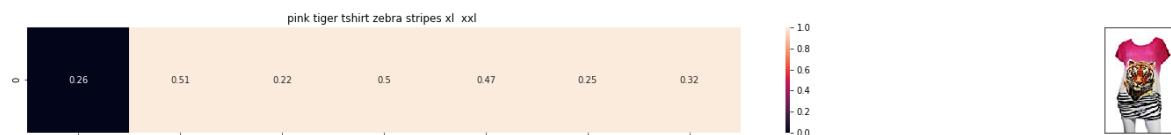
burnt umber tiger tshirt zebra stripes xl xxl



ASIN : B00JXQB5FQ

BRAND : Si Row

Eucliden distance from the given image : 0.0



ASIN : B00JXQASS6

BRAND : Si Row

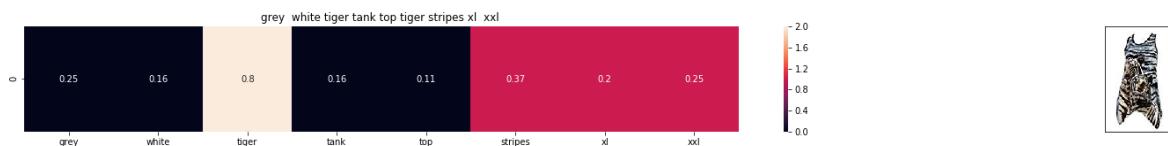
Eucliden distance from the given image : 0.7536331912451361



ASIN : B00JXQCWT0

BRAND : Si Row

Eucliden distance from the given image : 0.9357643943769645



ASIN : B00JXQAFZ2

BRAND : Si Row

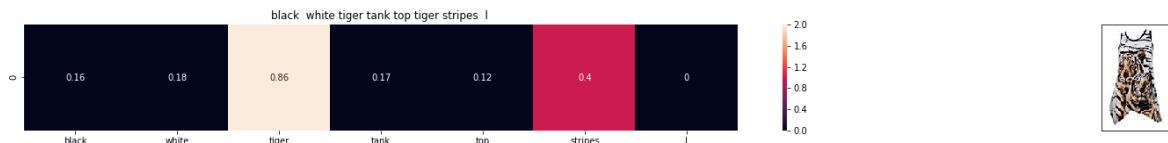
Euclidean distance from the given image : 0.9586153524200749



ASIN : B00JXQCUIC

BRAND : Si Row

Euclidean distance from the given image : 1.000074961446881



ASIN : B00JXQA094

BRAND : Si Row

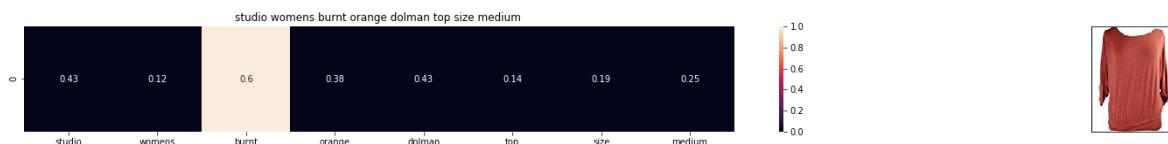
Euclidean distance from the given image : 1.023215552457452



ASIN : B00JXQAUWA

BRAND : Si Row

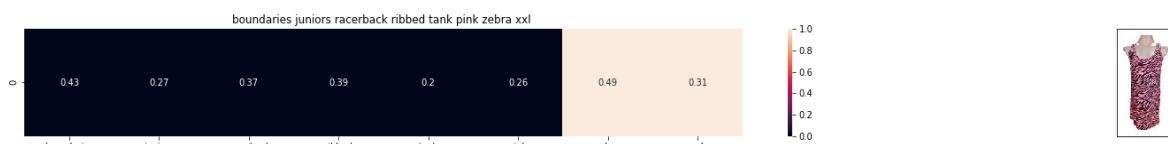
Euclidean distance from the given image : 1.031991846303421



ASIN : B06XSCVFT5

BRAND : Studio M

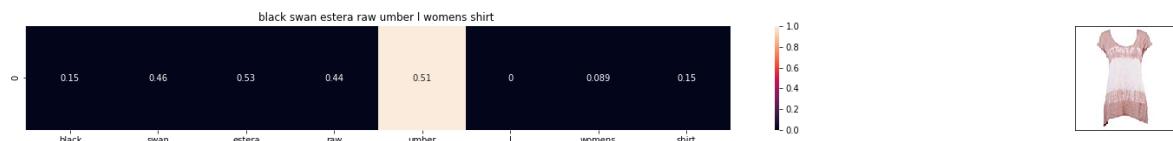
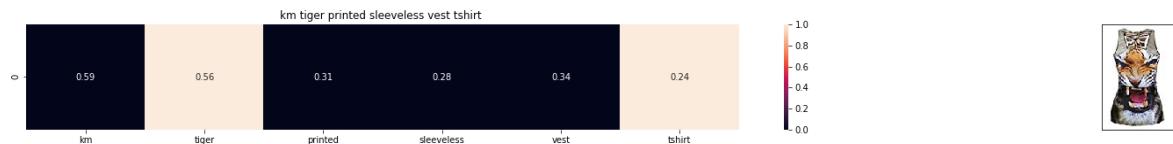
Euclidean distance from the given image : 1.2106843670424716



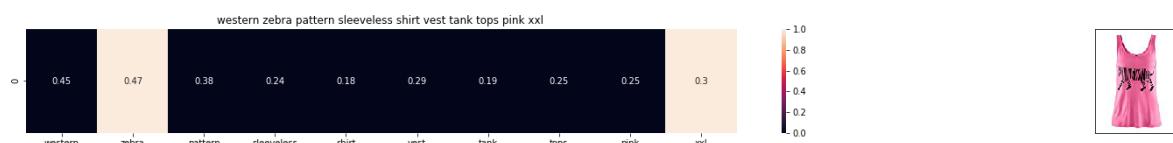
ASIN : B06Y2GTYPM

BRAND : No Boundaries

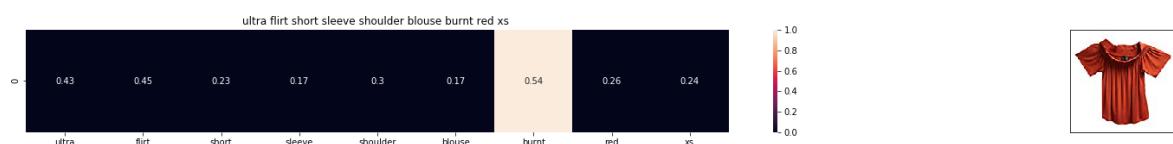
Eucliden distance from the given image : 1.212168381072083



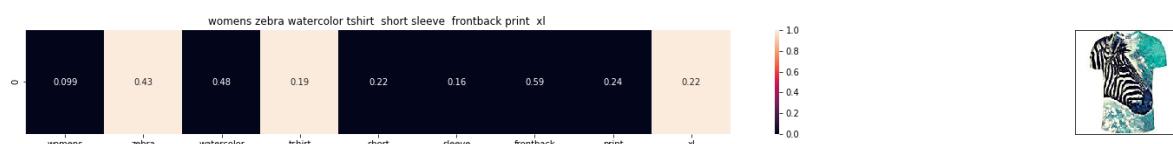
ASIN : B06Y1VN8WQ
BRAND : Black Swan
Eucliden distance from the given image : 1.2206849659998316



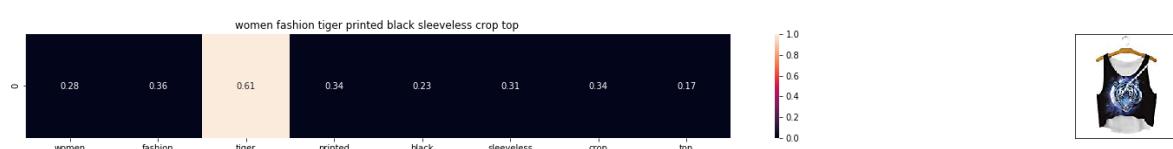
ASIN : B00Z6HEXWI
BRAND : Black Temptation
Eucliden distance from the given image : 1.221281392120943



ASIN : B074TR12BH
BRAND : Ultra Flirt
Eucliden distance from the given image : 1.2313364094597743



ASIN : B072R2JXKW
BRAND : WHAT ON EARTH
Eucliden distance from the given image : 1.2318451972624518



ASIN : B074T8ZYGX

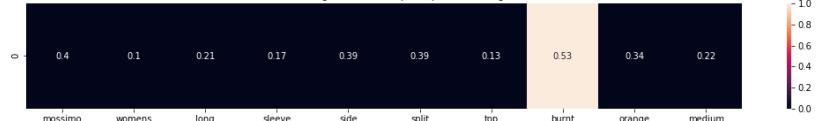
BRAND : MKP Crop Top

Eucliden distance from the given image : 1.2340607457359425

=====

=====

mossimo womens long sleeve side split top burnt orange medium



ASIN : B071ZDF6T2

BRAND : Mossimo

Eucliden distance from the given image : 1.2352785577664824

=====

=====

tpain tiger juniors tshirt size xlarge



ASIN : B01K0H020G

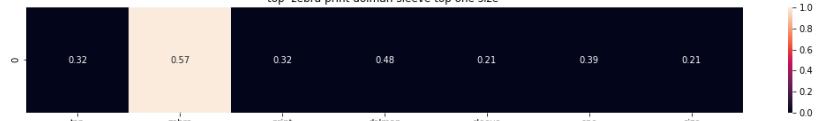
BRAND : Tultex

Eucliden distance from the given image : 1.236457298812782

=====

=====

top zebra print dolman sleeve top one size



ASIN : B00H8A6ZLI

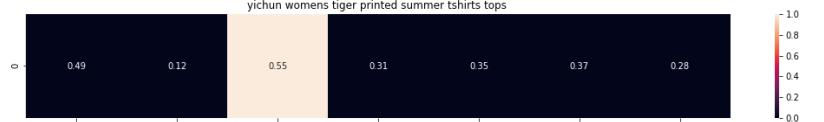
BRAND : Vivian's Fashions

Eucliden distance from the given image : 1.24996155052848

=====

=====

yichun womens tiger printed summer tshirts tops



ASIN : B010NN9RX0

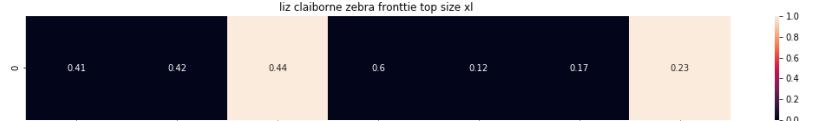
BRAND : YICHUN

Eucliden distance from the given image : 1.25354614208561

=====

=====

liz claiborne zebra fronttie top size xl



ASIN : B06XBY5QXL

BRAND : Liz Claiborne

Eucliden distance from the given image : 1.2538832938357722

=====

=====

[8.5] IDF based product similarity

In [50]:

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

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

In [51]:

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

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

In [52]:

```
# 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_title_features[:, idf_title_vectorizer.vocabulary_[i]].nonzero()[0] will
    # 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 wo
        # 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 [53]:

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

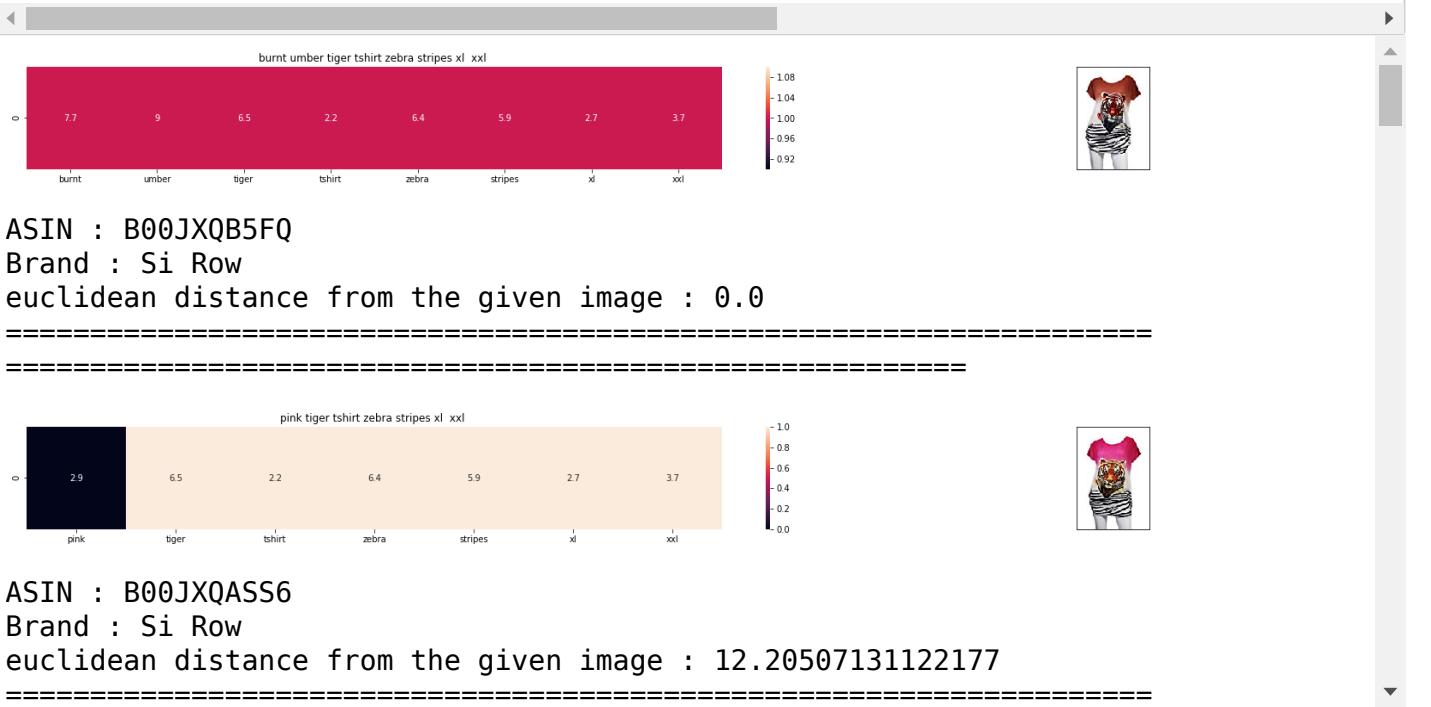
    # pairwise_dist will store the distance from given input apparel to all remaining
    # the metric we used here is cosine, the coside distance is measured as K(X, Y)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(idf_title_features,idf_title_features[doc_id])

    # 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[d
        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, th
```



[9] Text Semantics based product similarity

In [54]:

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

...
# Set values for various parameters
num_features = 300      # Word vector dimensionality
min_word_count = 1        # Minimum word count
num_workers = 4           # Number of threads to run in parallel
context = 10              # Context window size
downsampling = 1e-3       # Downsample setting for frequent words

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

...
```

Out[54]:

```
'\n# Set values for various parameters\nnum_features = 300      # Word v\nector dimensionality\n                           \nmin_word_count = 1      # Mi\nnimum word count\n                           \nnum_workers = 4         # Numb\ner of threads to run in parallel\ncontext = 10              # Context wind\now size\n\ndownsampling = 1e-3       # Downsample setting for frequent words\n#\n# Initialize and train the model (this will take some time)\n\nfrom gensim.\nmodels import word2vec\nprint ("Training model...")\nmodel = word2vec.\nWord2Vec(sen_corpus, workers=num_workers,\n                           size=num_feature\ns, min_count = min_word_count,\n                           window = context)\n\n'
```

In [55]:

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

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

...
model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', bin
...
#if you do NOT have RAM >= 12GB, use the code below.
with open('word2vec_model', 'rb') as handle:
    model = pickle.load(handle)
```

In [56]:

```
# 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
        # 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]])
            elif m_name == 'avg':
                vec.append(model[i])
        else:
            # if the word in our corpus is not there in the google word2vec corpus
            vec.append(np.zeros(shape=(300,)))
    # we will return a numpy array of shape (#number of words in title * 300 ) 300
    # each row represents the word2vec representation of each word (weighted/avg) in title
    return np.array(vec)

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

    final_dist = []
    # for each vector in vec1 we calculate the distance(euclidean) to all vectors in vec2
    for i in vec1:
        dist = []
        for j in vec2:
            # np.linalg.norm(i-j) will result the euclidean distance between vector i and 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)
    s1_vec = get_word_vec(sentence1, doc_id1, model)
    #s2_vec = np.array(#number_of_words_title1 * 300), each row is a vector(weighted)
    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
```

```

gs = gridspec.GridSpec(2, 2, width_ratios=[4,1],height_ratios=[2,1])
fig = plt.figure(figsize=(15,15))

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

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

plt.show()

```

In [57]:

```

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

vocab = model.keys()
# this function will add the vectors of each word and returns the avg vector of given sentence
def build_avg_vec(sentence, num_features, doc_id, m_name):
    # 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
        # if m_name == 'weighted', we will multiply each w2v[word] with the idf(word)
    featureVec = np.zeros((num_features,), dtype="float32")
    # we will intialize a vector of size 300 with all zeros
    # we add each word2vec(wordi) to this fatureVec
    nwords = 0

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

```

[9.2] Average Word2Vec product similarity.

In [58]:

```
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
w2v_title = np.array(w2v_title)
```

In [59]:

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

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

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

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

    for i in range(0, len(indices)):
        heat_map_w2v(data['title'].loc[df_indices[0]], data['title'].loc[df_indices[i]])
        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,
```

[9.4] IDF weighted Word2Vec for product similarity

In [60]:

```
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
w2v_title_weight = np.array(w2v_title_weight)
```

In [61]:

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

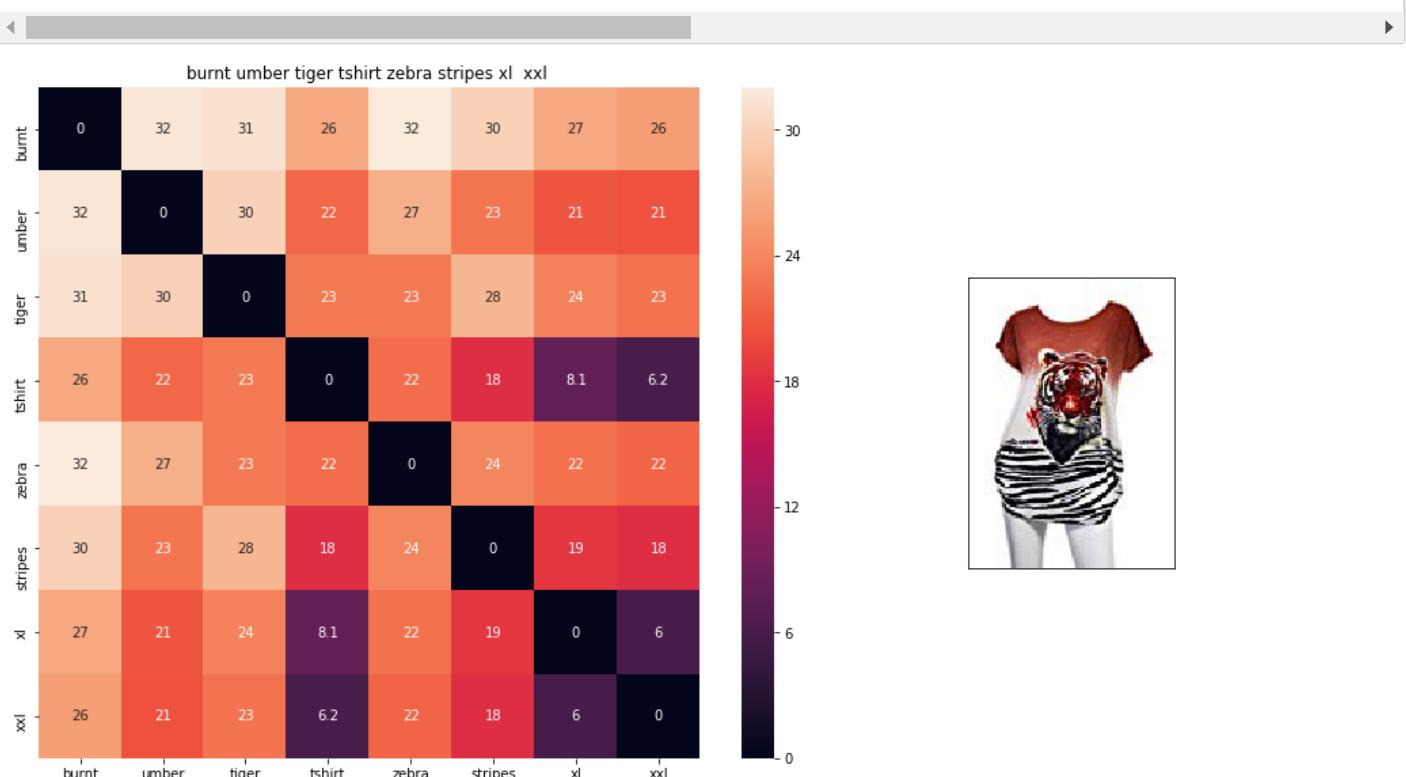
    # pairwise_dist will store the distance from given input apparel to all remaining
    # the metric we used here is cosine, the cosine distance is measured as K(X, Y)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(w2v_title_weight, w2v_title_weight[doc_id]).r

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

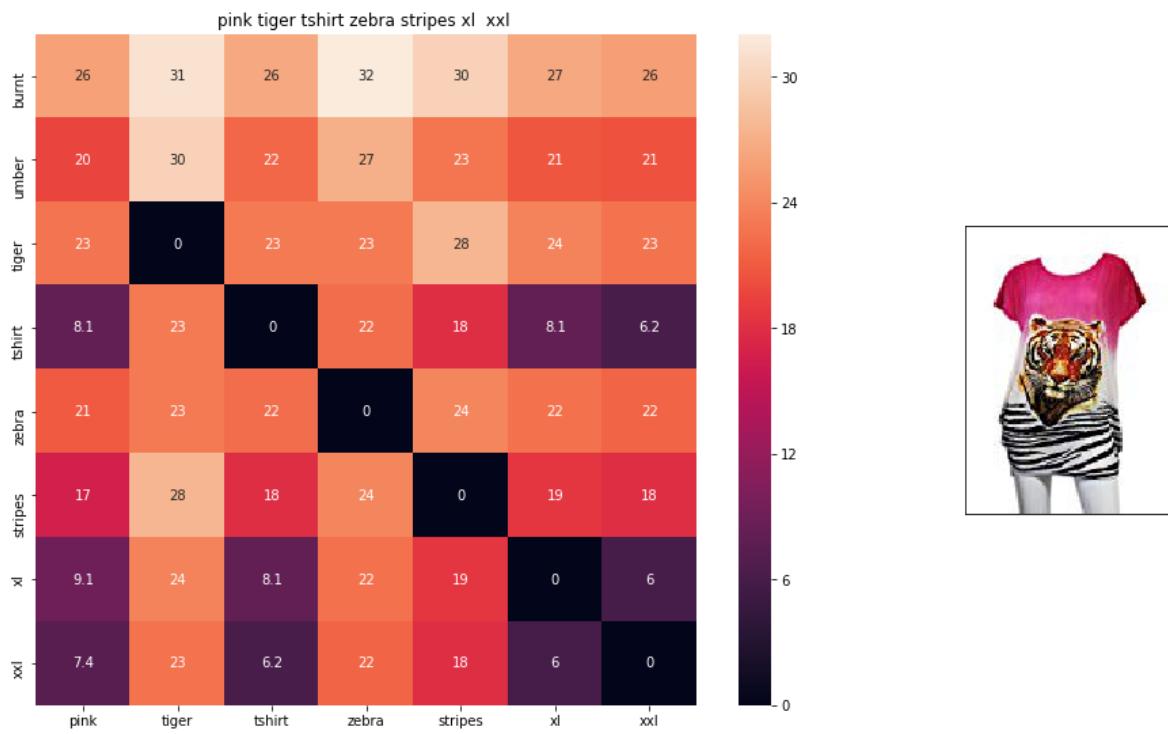
    #data frame indices of the 9 smallest distance's
    df_indices = list(data.index[indices])

    for i in range(0, len(indices)):
        heat_map_w2v(data['title'].loc[df_indices[0]], data['title'].loc[df_indices[i]])
        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,
```



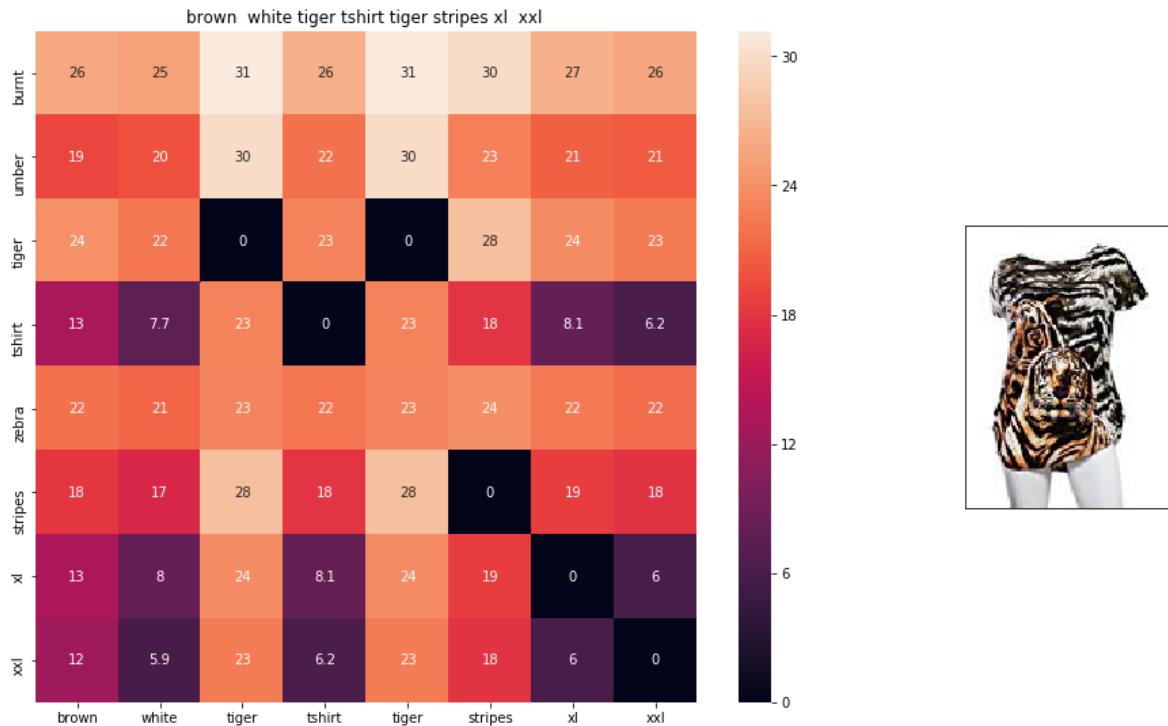
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 0.00390625
=====



ASIN : B00JXQASS6

Brand : Si Row

euclidean distance from input : 4.0638876



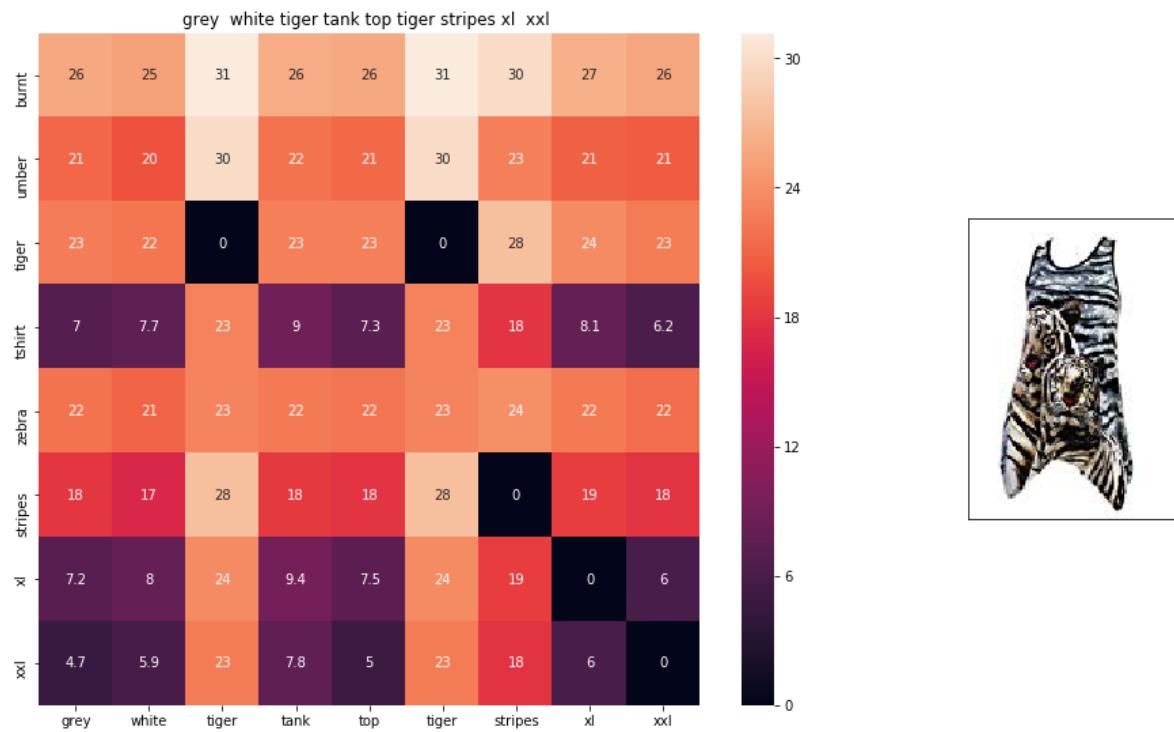
ASIN : B00JXQCWT0

Brand : Si Row

euclidean distance from input : 4.770942

=====

=====



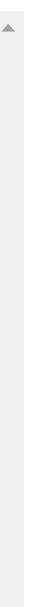
ASIN : B00JXQAFZ2

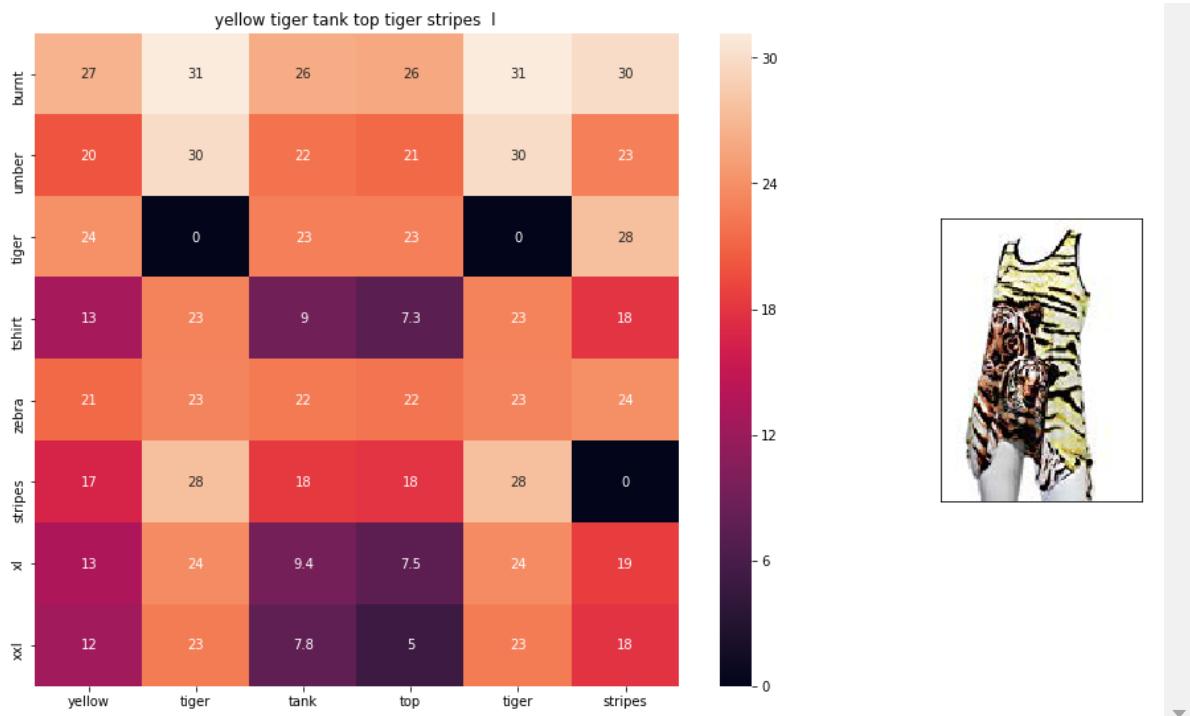
Brand : Si Row

euclidean distance from input : 5.3601613

=====

=====

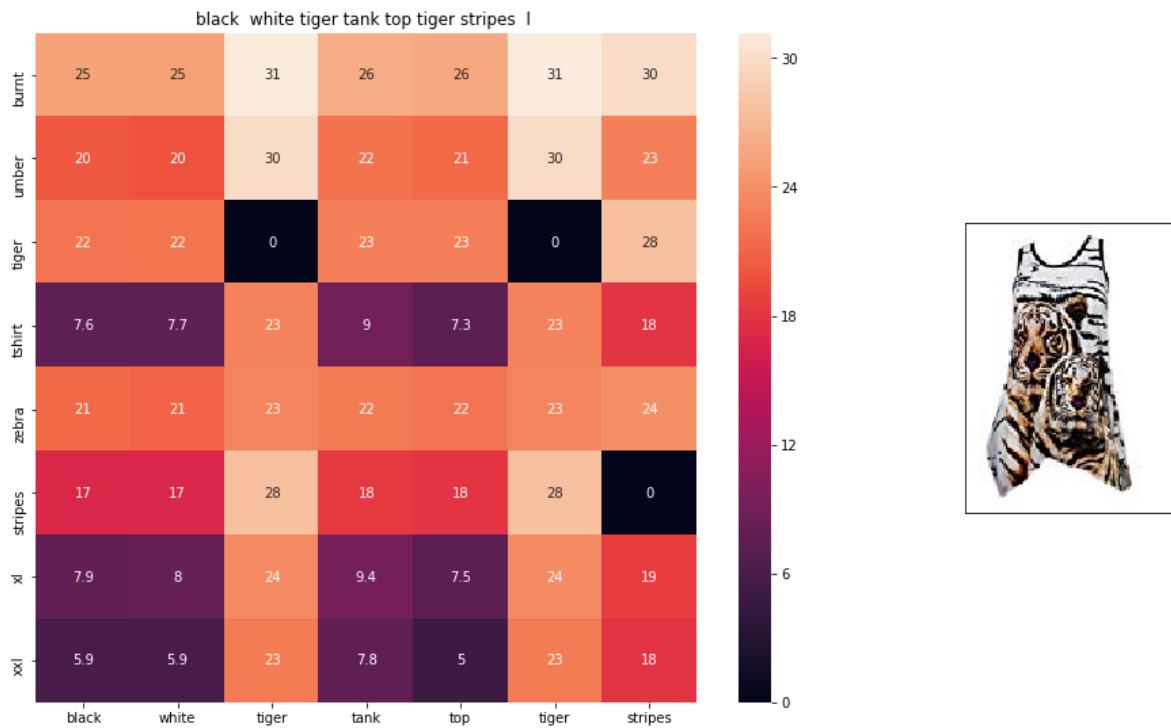




ASIN : B00JXQAUWA

Brand : Si Row

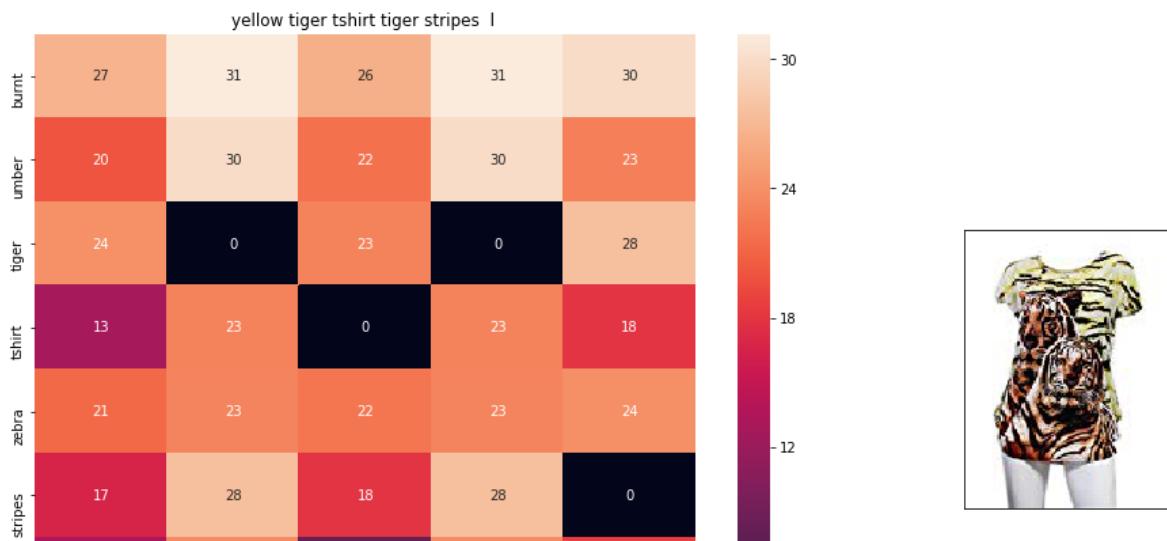
euclidean distance from input : 5.689523



ASIN : B00JXQA094

Brand : Si Row

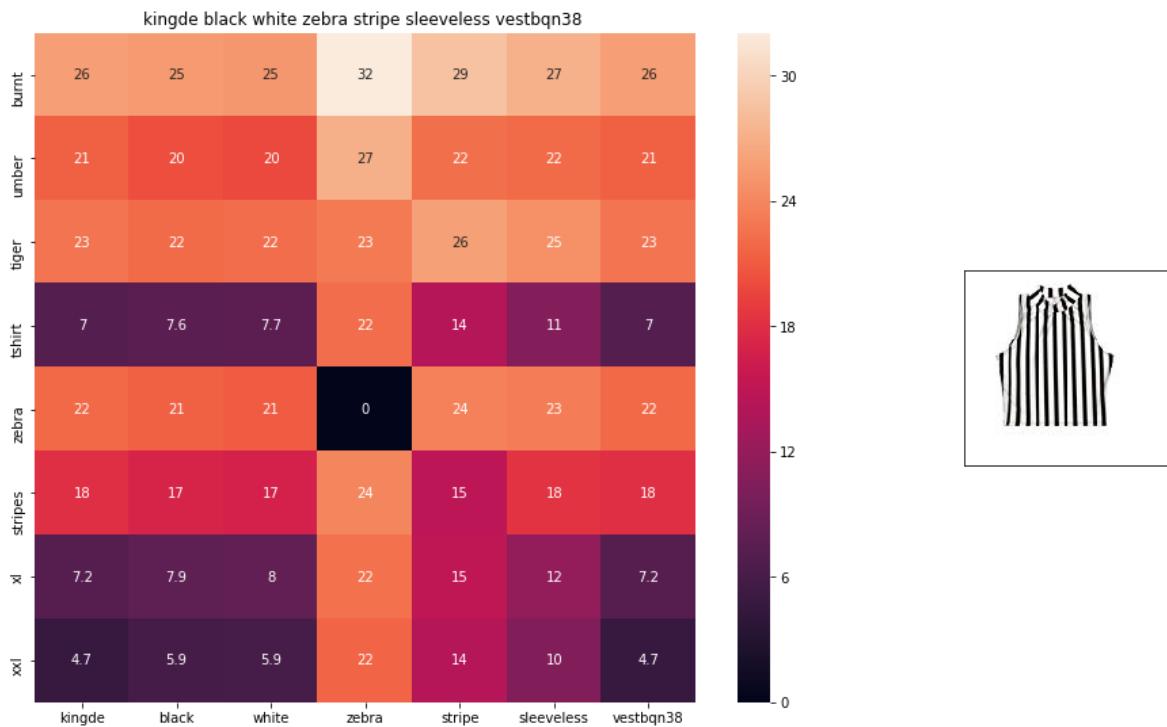
euclidean distance from input : 5.6930227



ASIN : B00JXQCUIC

Brand : Si Row

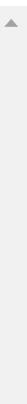
euclidean distance from input : 5.8934426

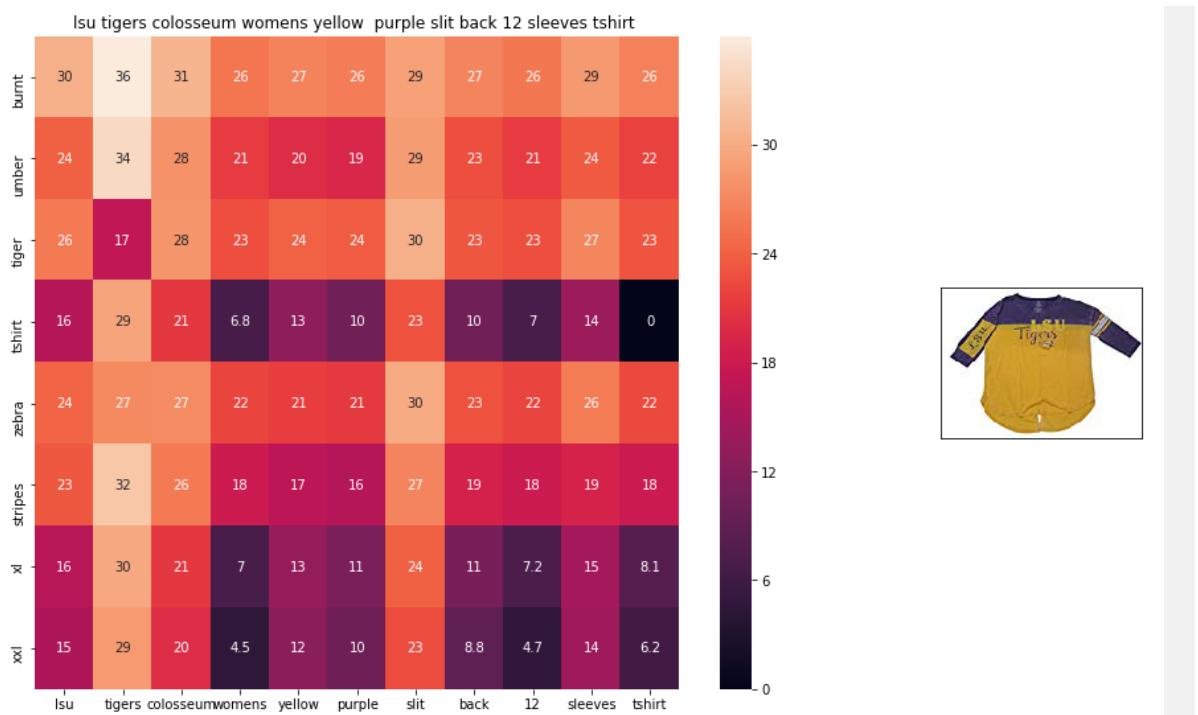


ASIN : B015H41F6G

Brand : KINGDE

euclidean distance from input : 6.13299

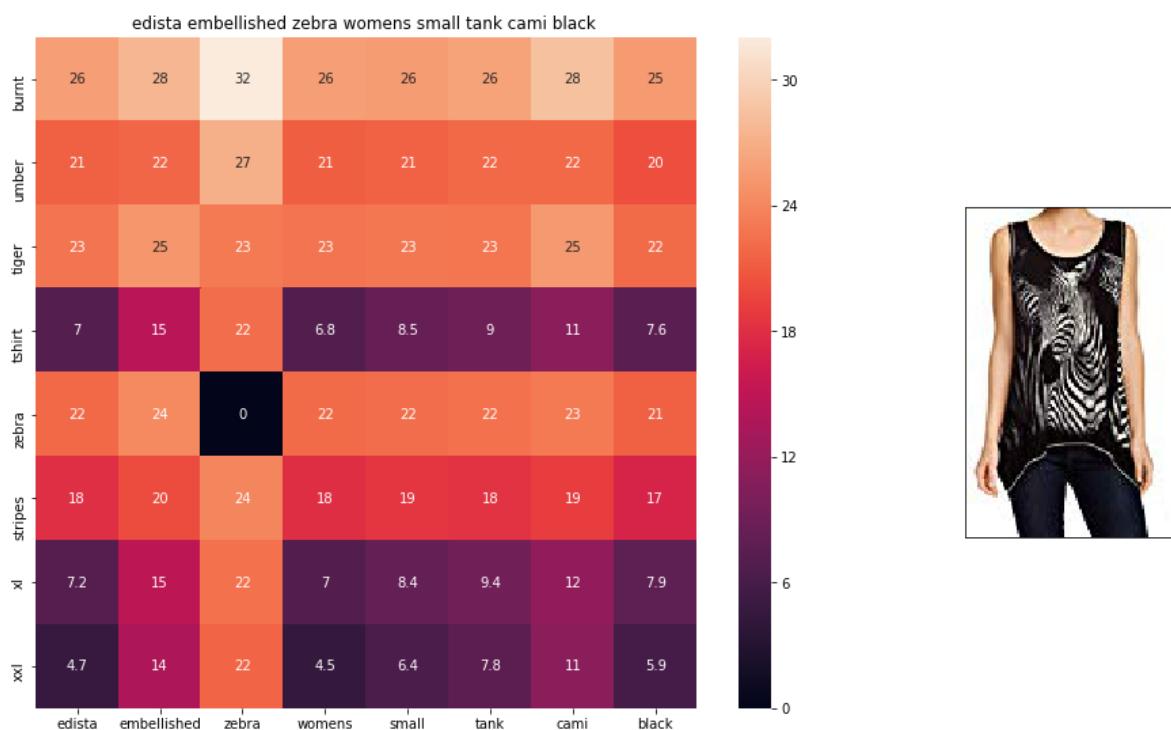




ASIN : B073R5Q8HD

Brand : Colosseum

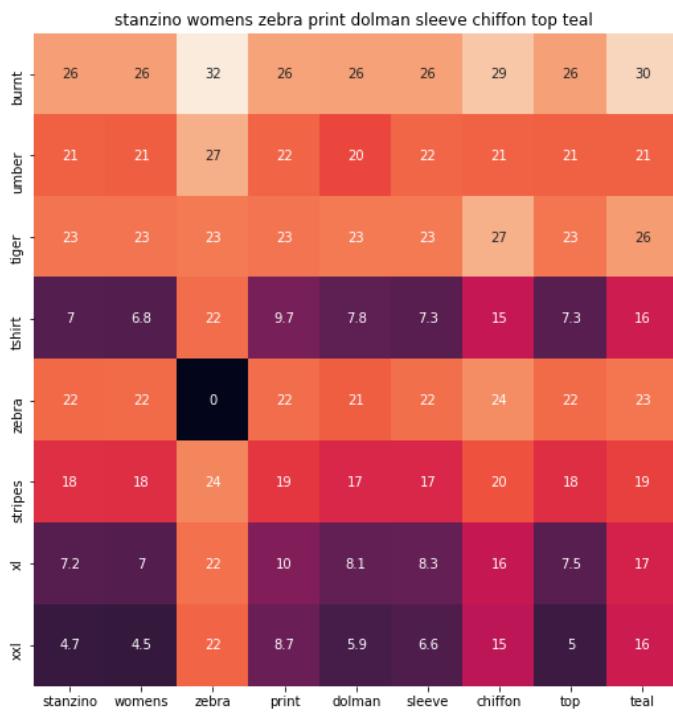
euclidean distance from input : 6.2567058



ASIN : B074P8MD22

Brand : Edista

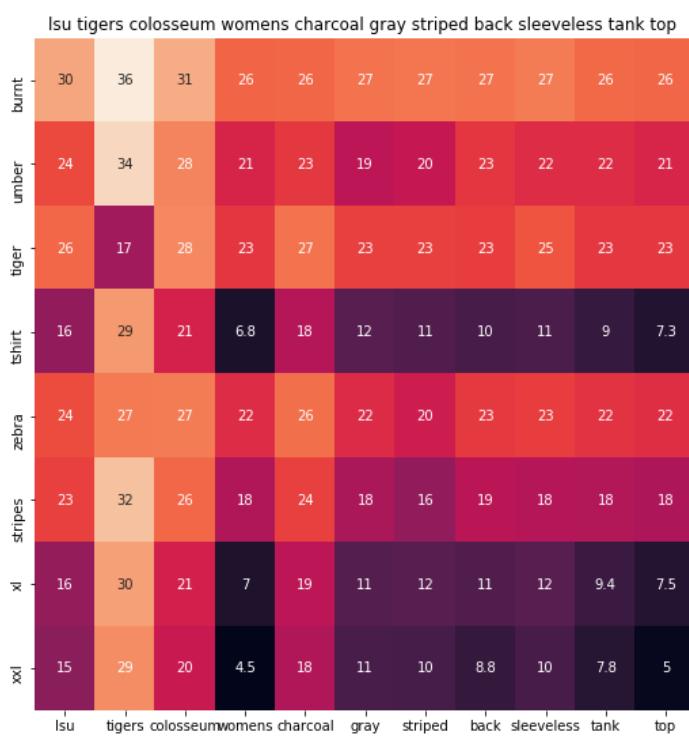
euclidean distance from input : 6.3922043



ASIN : B00C0I3U3E

Brand : Stanzino

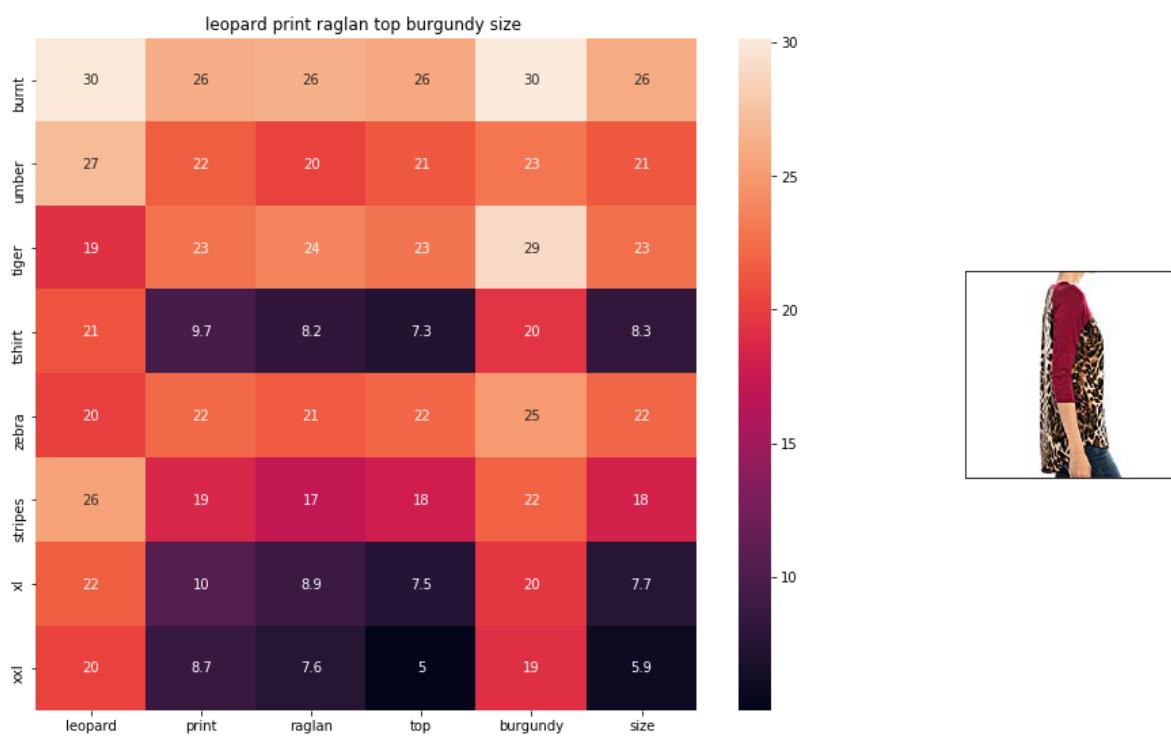
euclidean distance from input : 6.414901



ASIN : B073R4ZM7Y

Brand : Colosseum

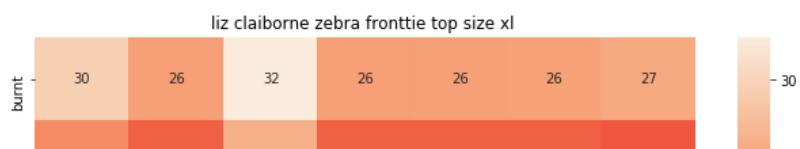
euclidean distance from input : 6.4509606



ASIN : B01C60RLDQ

Brand : 1 Mad Fit

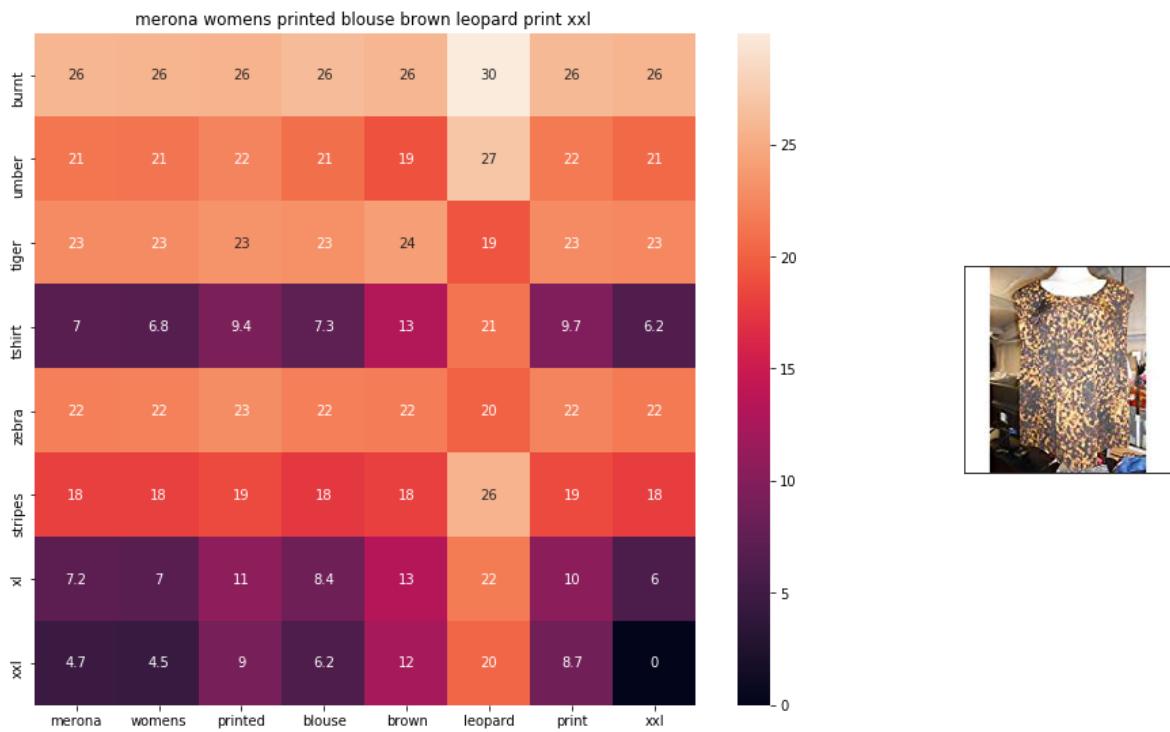
euclidean distance from input : 6.4634094



ASIN : B06XBY5QXL

Brand : Liz Claiborne

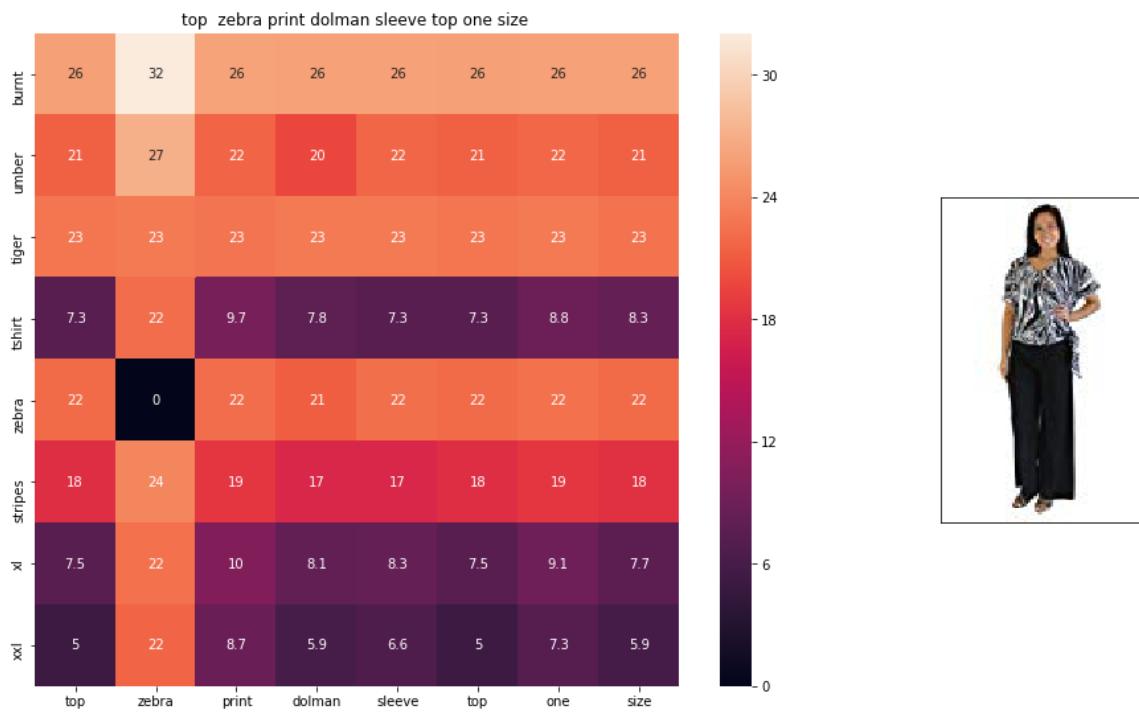
euclidean distance from input : 6.5392237



ASIN : B071YF3WDD

Brand : Merona

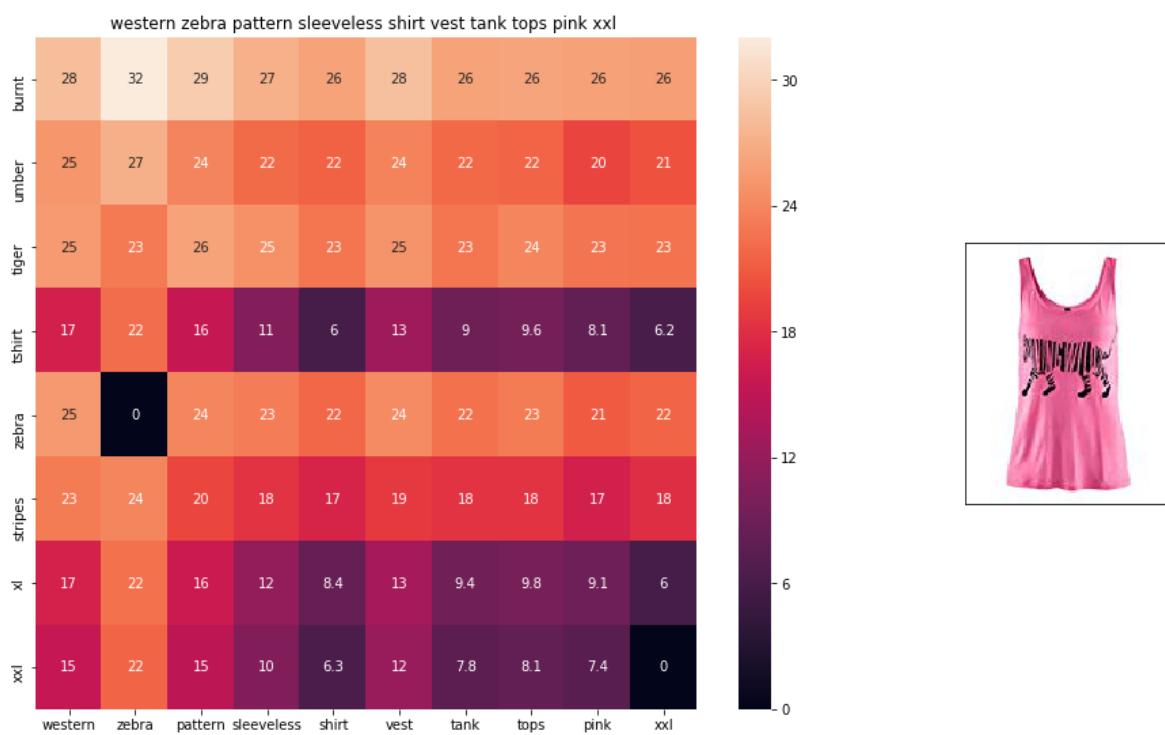
euclidean distance from input : 6.575504



ASIN : B00H8A6ZLI

Brand : Vivian's Fashions

euclidean distance from input : 6.6382155



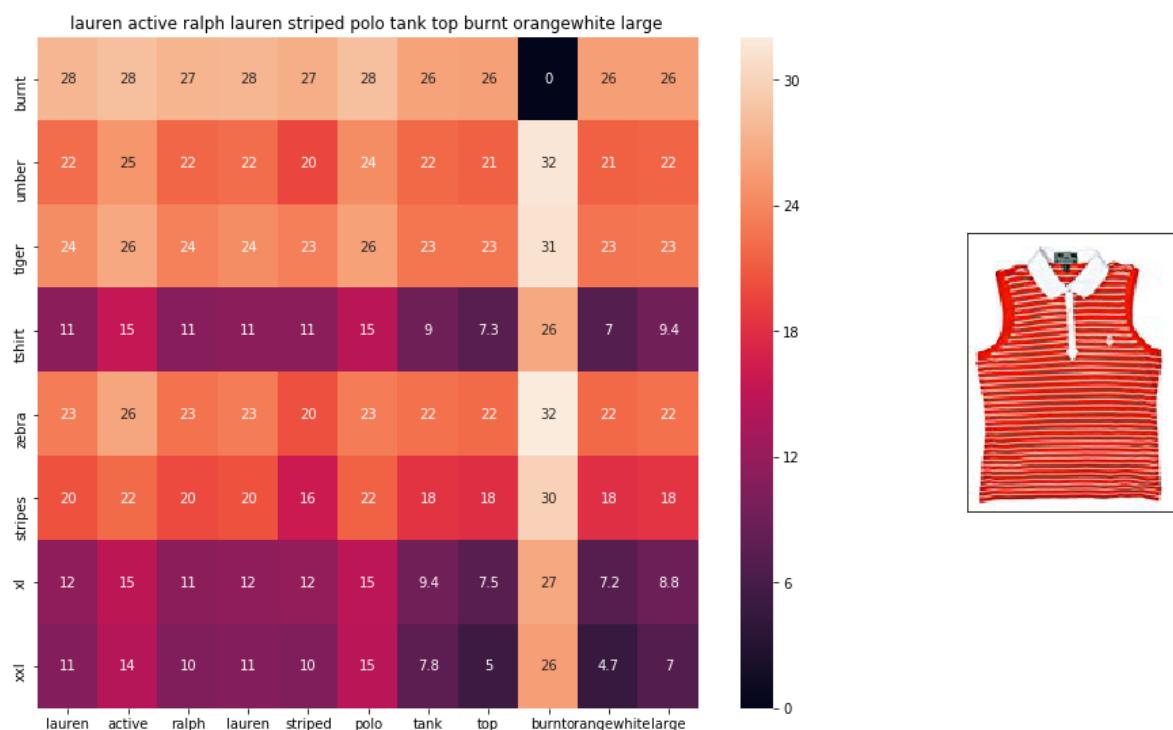
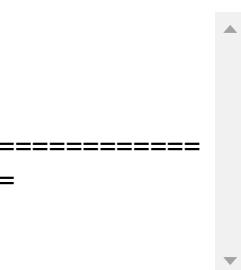
ASIN : B00Z6HEXWI

Brand : Black Temptation

euclidean distance from input : 6.660737

=====

=====



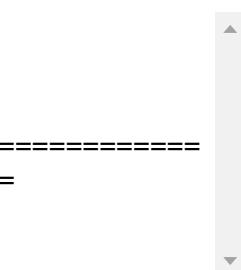
ASIN : B00ILGH50Y

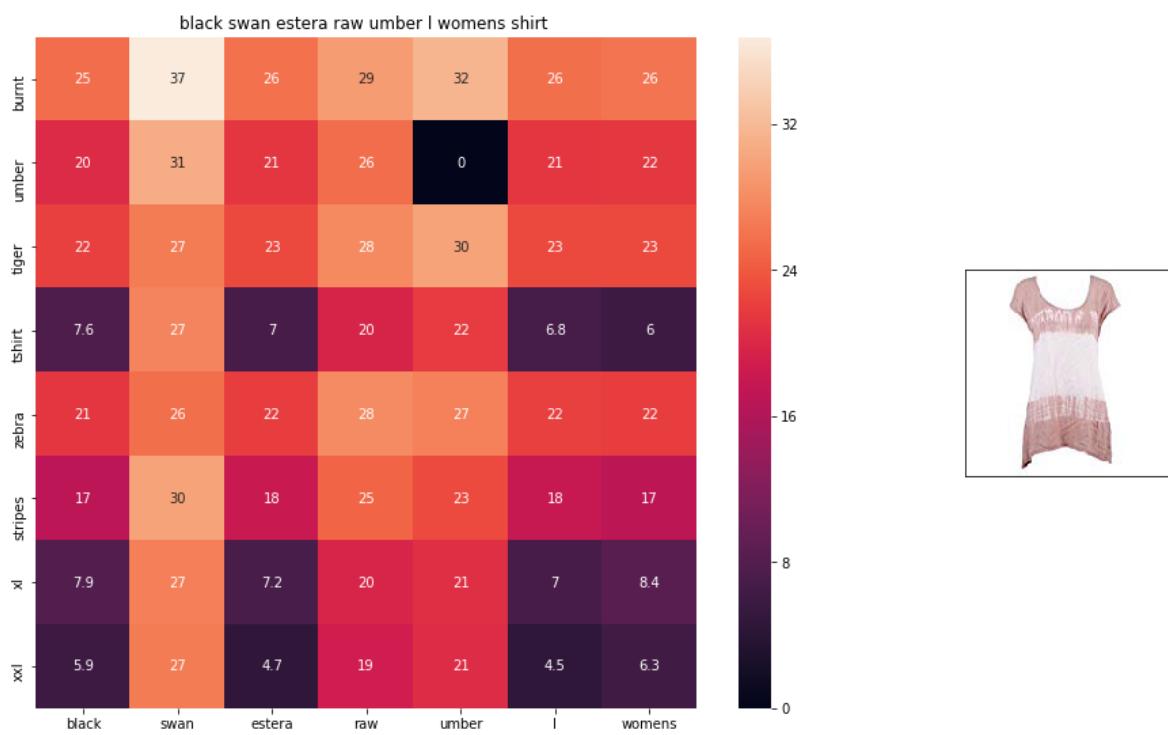
Brand : Ralph Lauren Active

euclidean distance from input : 6.6839056

=====

=====

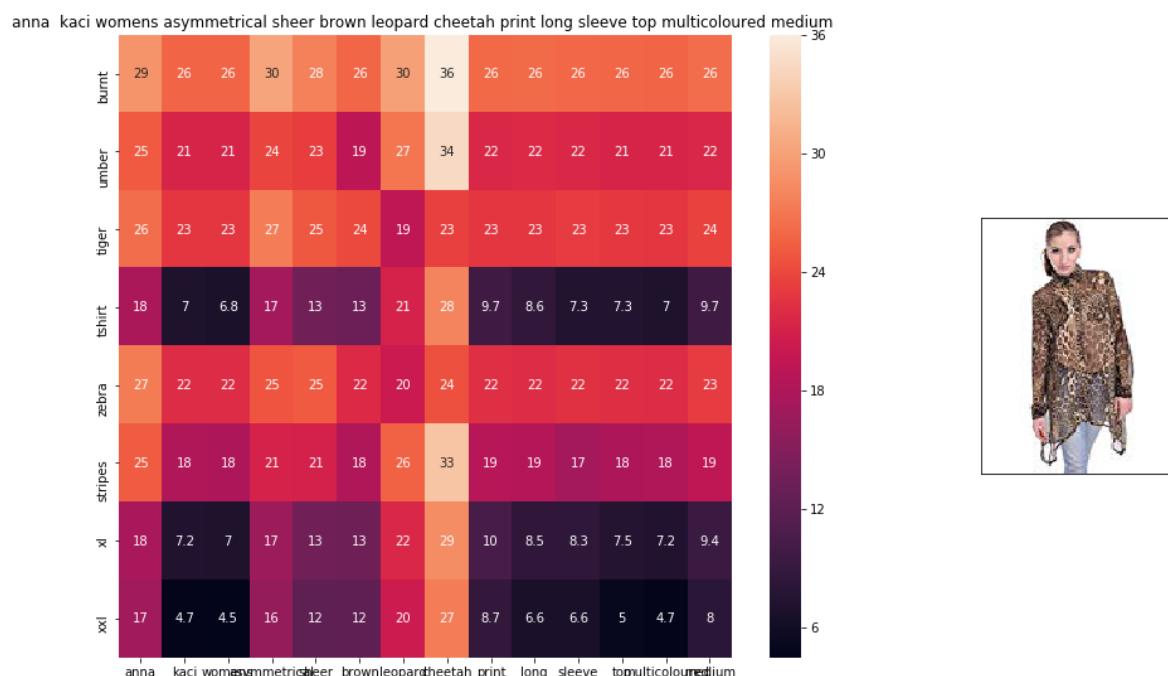




ASIN : B06Y1VN8WQ

Brand : Black Swan

euclidean distance from input : 6.7057643



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

[9.6] Weighted similarity using brand and color.

In [62]:

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

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

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

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

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

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

In [63]:

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

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

    #s1_vec = np.array(#number_of_words_title1 * 300), each row is a vector(weighted)
    s1_vec = get_word_vec(sentance1, doc_id1, model)
    #s2_vec = np.array(#number_of_words_title2 * 300), each row is a vector(weighted)
    s2_vec = get_word_vec(sentance2, doc_id2, model)

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

    data_matrix = [['Asin', 'Brand', 'Color', 'Product type'],
                   [data['asin'].loc[df_id1], brands[doc_id1], colors[doc_id1], types[doc_id1]],
                   [data['asin'].loc[df_id2], brands[doc_id2], colors[doc_id2], types[doc_id2]]]

    colorscale = [[0, '#1d004d'], [.5, '#f2e5ff'], [1, '#f2e5d1']] # to color the heatmap

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

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

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

    # in last 25 * 10:15 grids we display image
    ax2 = plt.subplot(gs[:, 10:16])
    # we dont display grid 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 [80]:

```
def idf_w2v_brand(doc_id,num_results, w1, w2 , w3 = 0):
    # 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
    # the metric we used here is cosine, the coside distance is measured as K(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]).r
    ex_feat_dist = pairwise_distances(extra_features, extra_features[doc_id])
    image_features_dist = pairwise_distances(bottleneck_features_train, bottleneck_
    pairwise_dist = (w1 * idf_w2v_dist + w2 * ex_feat_dist + w3 * image_features

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

Asin Brand Color

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B00JXQB5FQ

Brand : Si Row

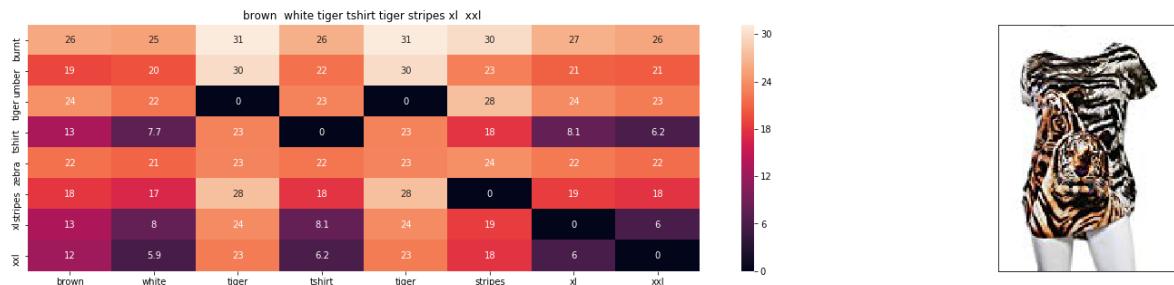
euclidean distance from input : 0.001953125

=====

=====

Asin	Brand	Color
------	-------	-------

B00JXQB5FQ	Si-Row	Brown
------------	--------	-------

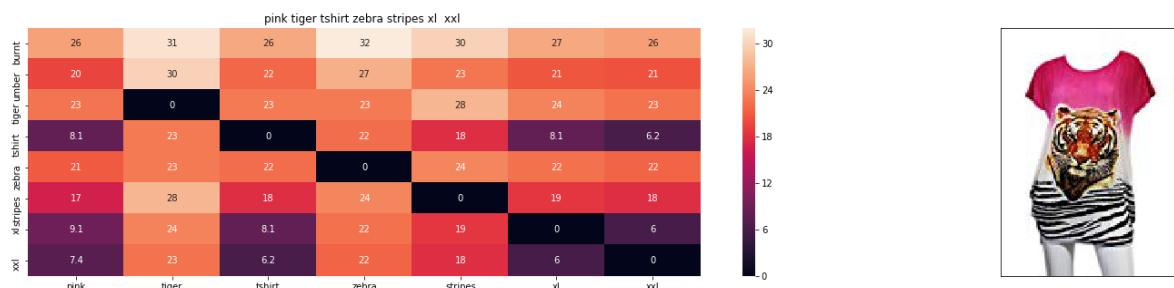


ASIN : B00JXQCWT0

Brand : Si Row

euclidean distance from input : 2.385471153259277

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

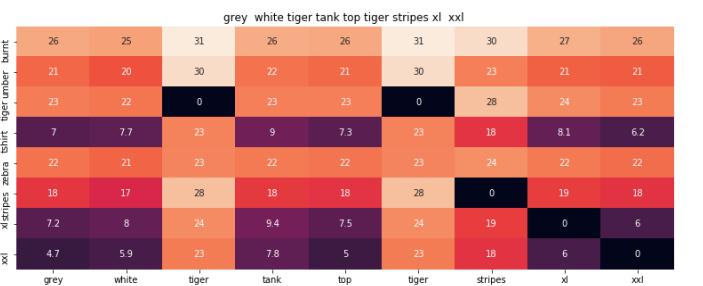


ASIN : B00JXQASS6

Brand : Si Row

euclidean distance from input : 2.7390506746191647

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

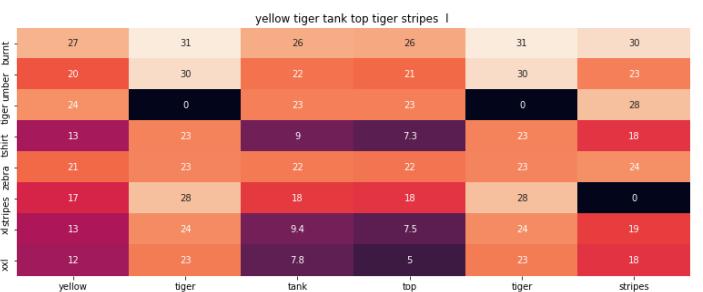


ASIN : B00JXQAFZ2

Brand : Si Row

euclidean distance from input : 3.3871873857397703

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

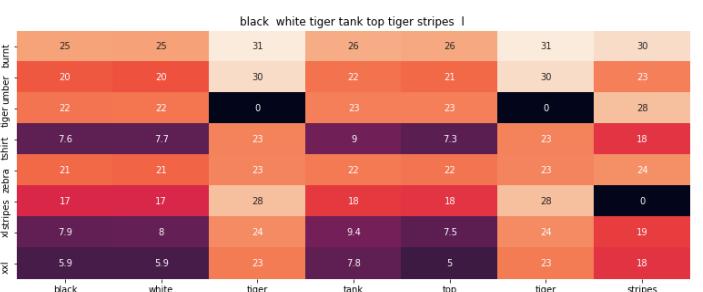


ASIN : B00JXQAUWA

Brand : Si Row

euclidean distance from input : 3.5518684389013915

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

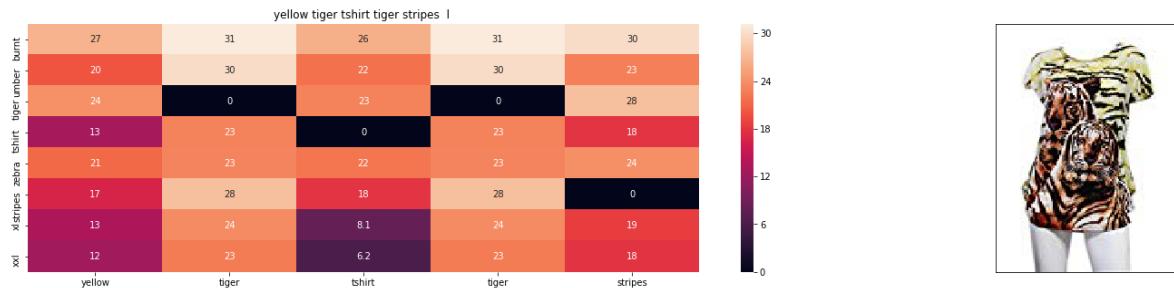


ASIN : B00JXQA094

Brand : Si Row

euclidean distance from input : 3.5536182405371335

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

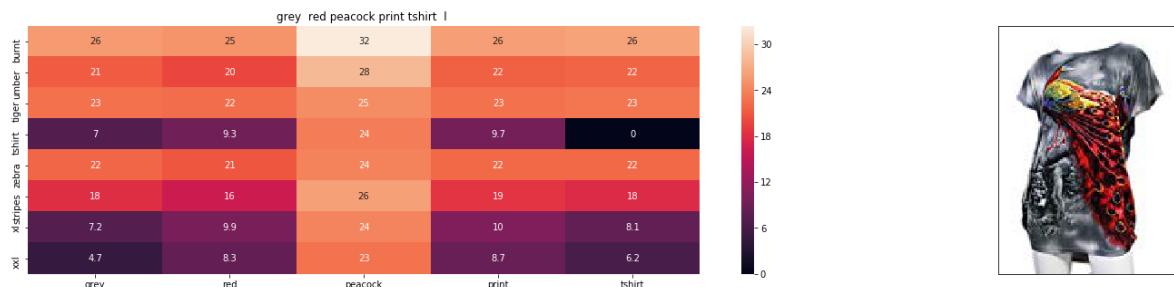


ASIN : B00JXQCUIC

Brand : Si Row

euclidean distance from input : 3.653828048886743

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



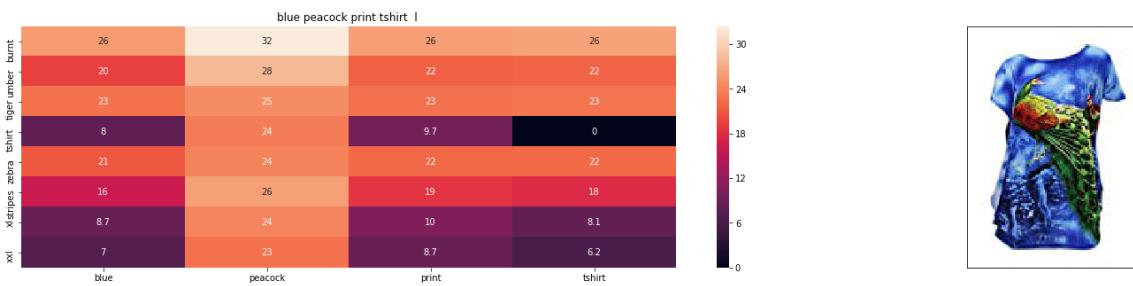
ASIN : B00JXQCFRS

Brand : Si Row

euclidean distance from input : 4.128811645688501

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



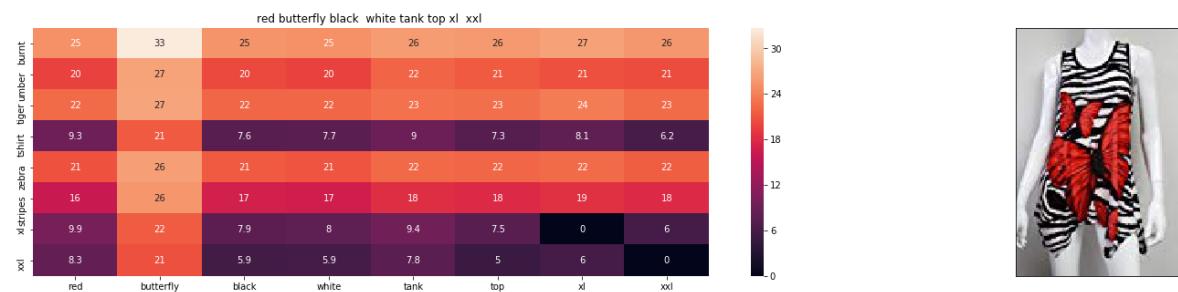


ASIN : B00JXQC8L6

Brand : Si Row

euclidean distance from input : 4.203900146665063

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B00JV63CW2

Brand : Si Row

euclidean distance from input : 4.286586761655298

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

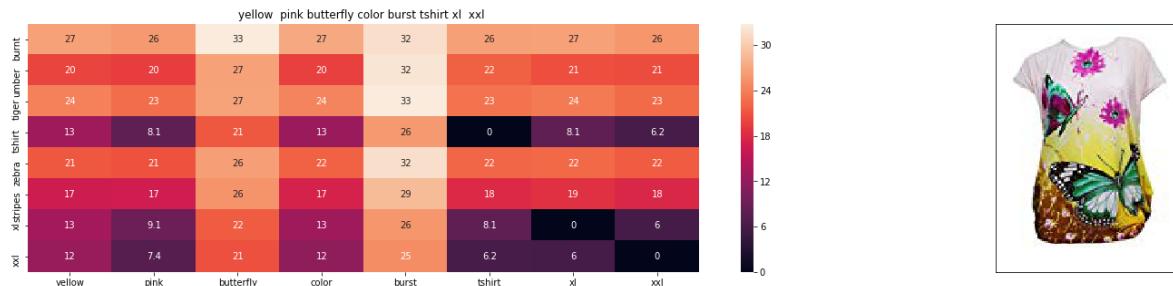


ASIN : B015H41F6G

Brand : KINGDE

euclidean distance from input : 4.389370597243721

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B00JXQBBMI

Brand : Si Row

euclidean distance from input : 4.397909927548852

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

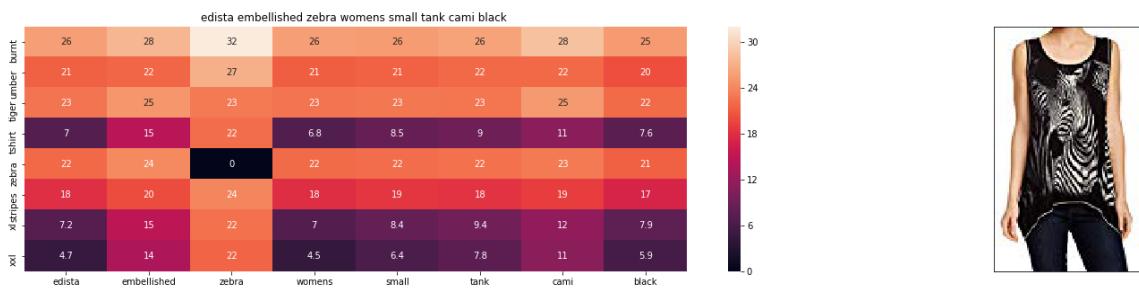


ASIN : B073R5Q8HD

Brand : Colosseum

euclidean distance from input : 4.451228583693917

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

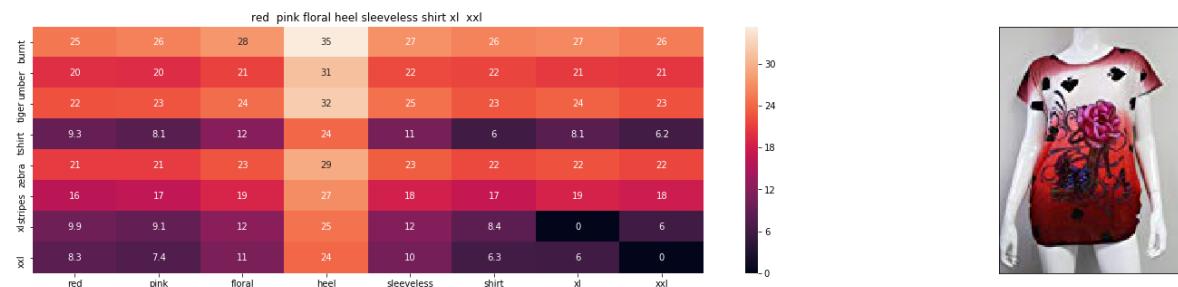


ASIN : B074P8MD22

Brand : Edista

euclidean distance from input : 4.518977797866279

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B00JV63QQE

Brand : Si Row

euclidean distance from input : 4.52937545794436

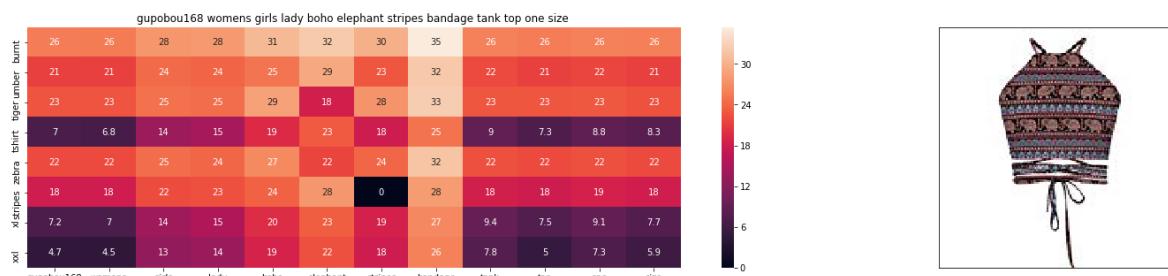
Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

ASIN : B00C0I3U3E

Brand : Stanzino

euclidean distance from input : 4.530326140761788

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

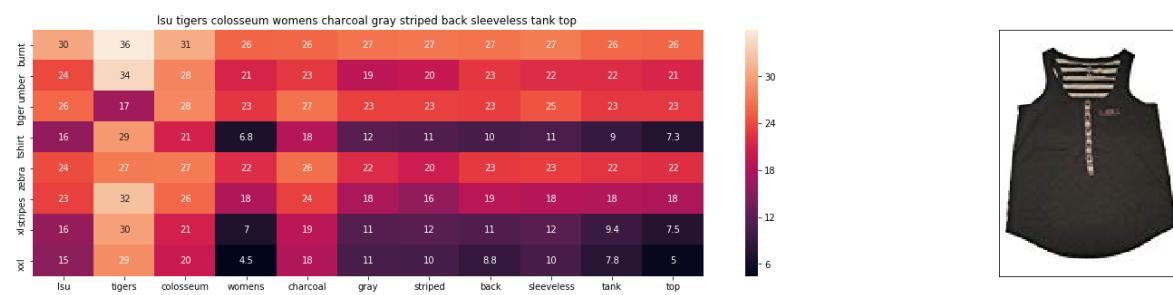


ASIN : B01ER18406

Brand : GuPoBoU168

euclidean distance from input : 4.546817024028215

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

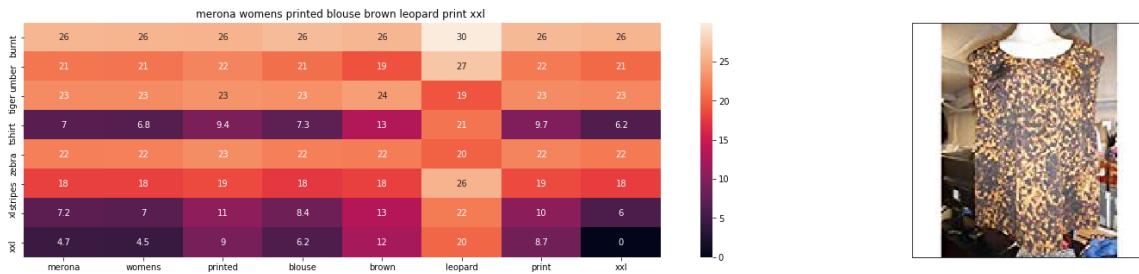


ASIN : B073R4ZM7Y

Brand : Colosseum

euclidean distance from input : 4.548355925918037

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B071YF3WDD

Brand : Merona

euclidean distance from input : 4.610627425551827

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B01C60RLDQ

Brand : 1 Mad Fit

euclidean distance from input : 4.645918274287157

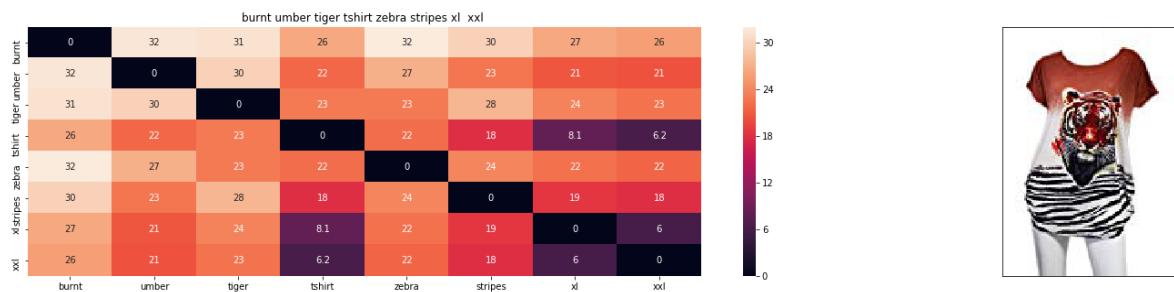
In [65]:

```
# brand and color weight =50
```

```
# title vector weight = 5
```

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

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

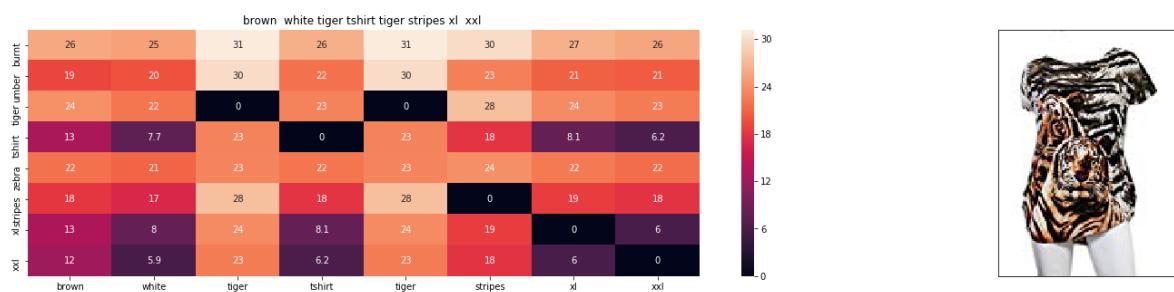


ASIN : B00JXQB5FQ

Brand : Si Row

euclidean distance from input : 0.0003551136363636364

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B00JXQCWT0

Brand : Si Row

euclidean distance from input : 0.43372202786532316

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

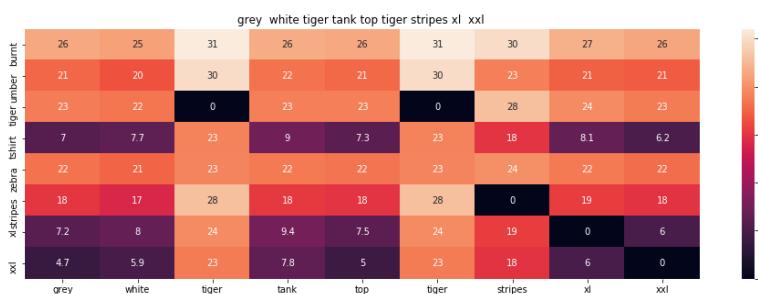


ASIN : B00JXQASS6

Brand : Si Row

euclidean distance from input : 1.655093037326926

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

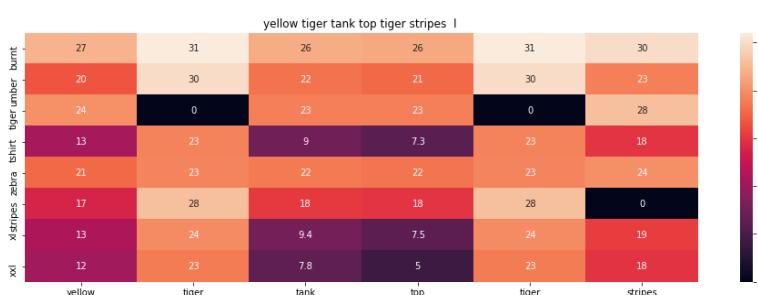


ASIN : B00JXQAFZ2

Brand : Si Row

euclidean distance from input : 1.7729360757124906

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B00JXQAUWA

Brand : Si Row

euclidean distance from input : 1.8028780853782398

=====

=====

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B00JXQA094

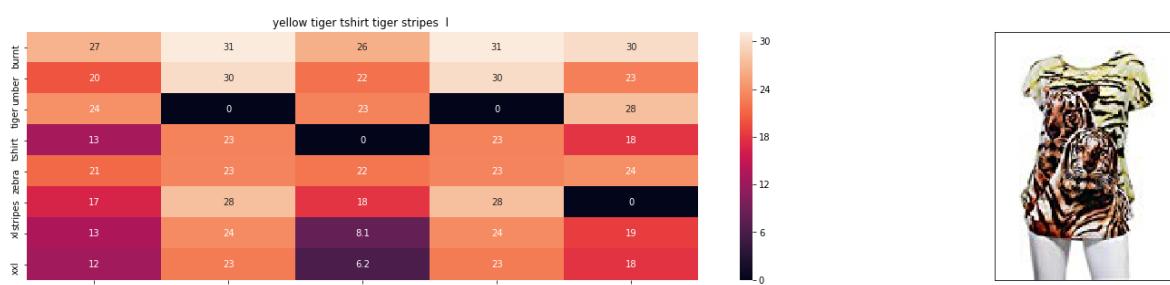
Brand : Si Row

euclidean distance from input : 1.803196231130193

=====

=====

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B00JXQCUIC

Brand : Si Row

euclidean distance from input : 1.8214161962846673

=====

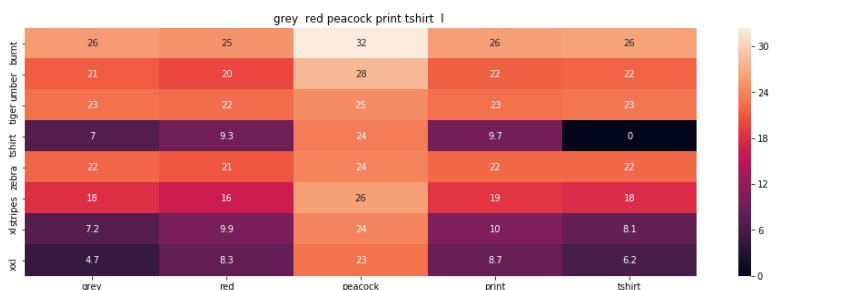
=====

Asin	Brand	Color

B00JXQB5FQ

Si-Row

Brown

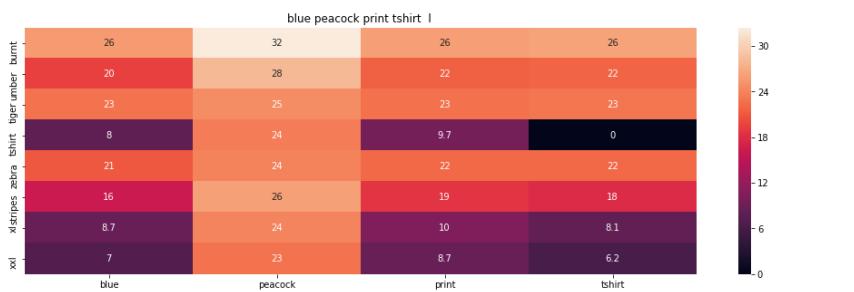


ASIN : B00JXQCFRS

Brand : Si Row

euclidean distance from input : 1.9077768502486234

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

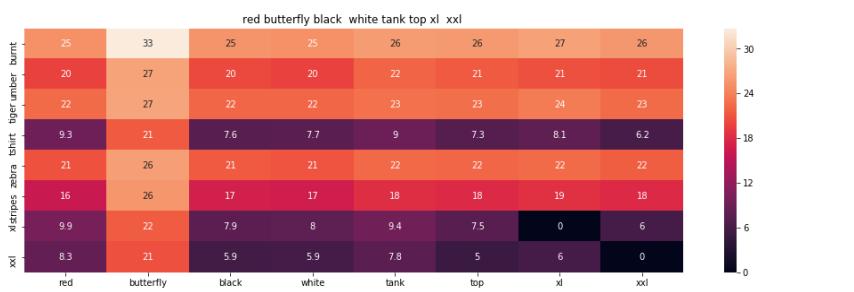


ASIN : B00JXQC8L6

Brand : Si Row

euclidean distance from input : 1.9214293049716347

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B00JV63CW2

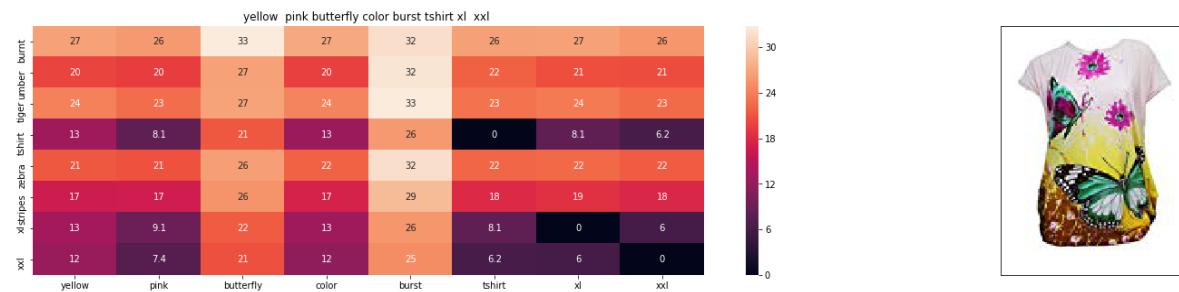
Brand : Si Row

euclidean distance from input : 1.9364632349698592

=====

=====

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B00JXQBBMI

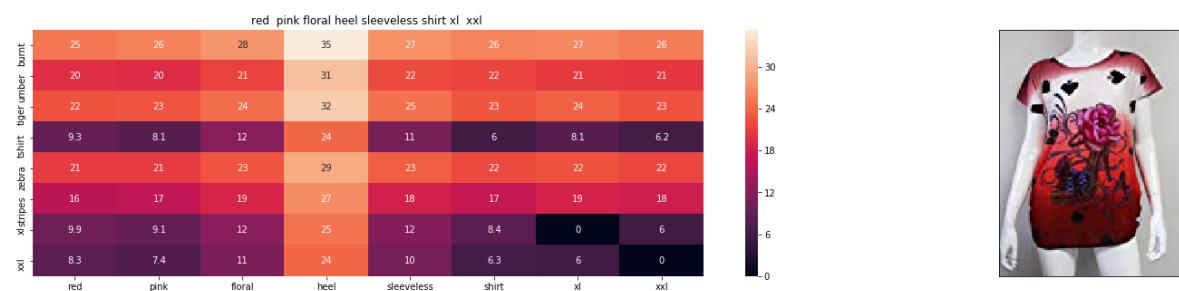
Brand : Si Row

euclidean distance from input : 1.956703810586869

=====

=====

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

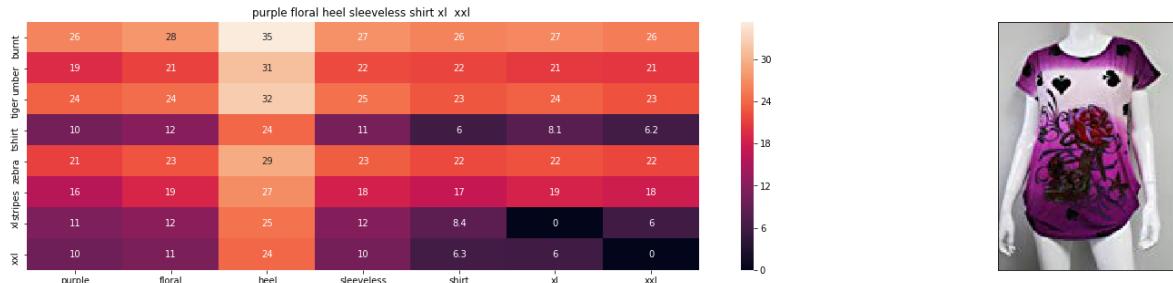


ASIN : B00JV63QQE

Brand : Si Row

euclidean distance from input : 1.9806066342951432

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

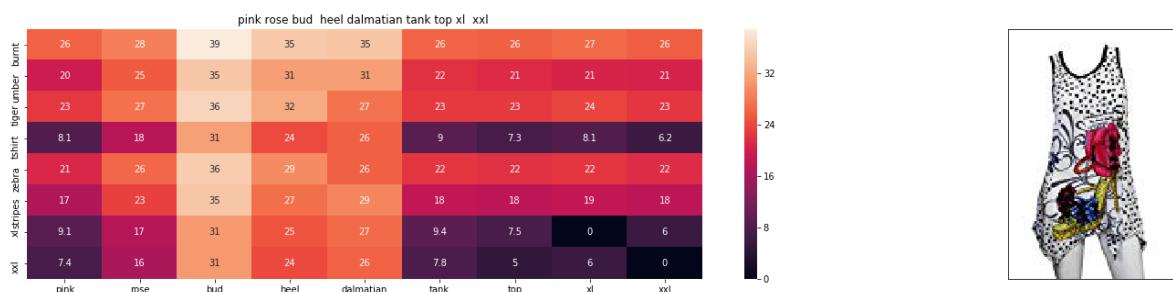


ASIN : B00JV63VC8

Brand : Si Row

euclidean distance from input : 2.0121855999157043

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

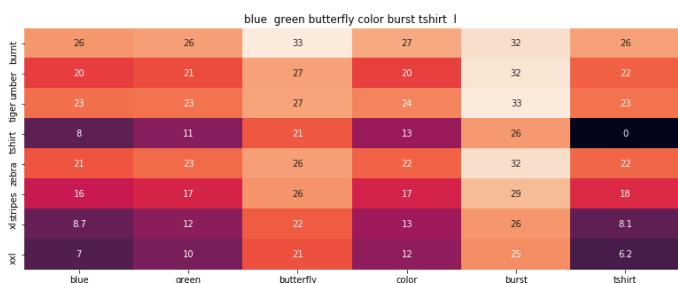


ASIN : B00JXQAX2C

Brand : Si Row

euclidean distance from input : 2.013351787548872

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

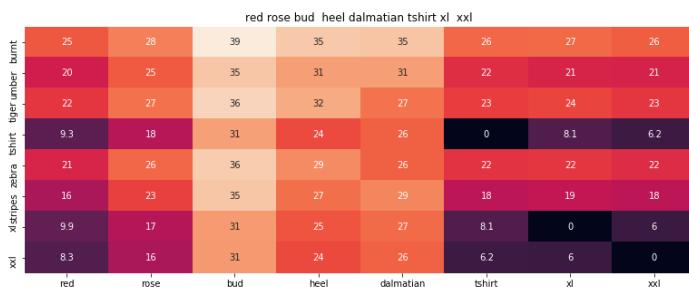


ASIN : B00JXQC0C8

Brand : Si Row

euclidean distance from input : 2.013883348273304

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

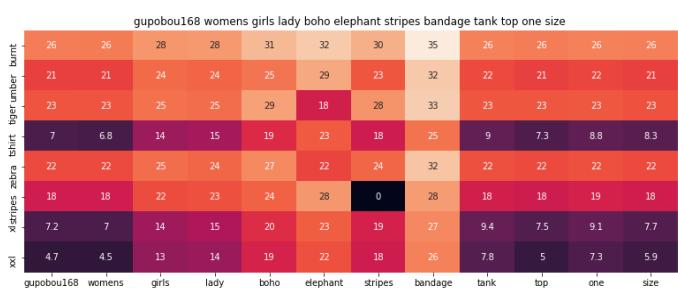


ASIN : B00JXQABB0

Brand : Si Row

euclidean distance from input : 2.0367257554998663

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B01ER18406

Brand : GuPoBoU168

euclidean distance from input : 2.6562041677776853

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

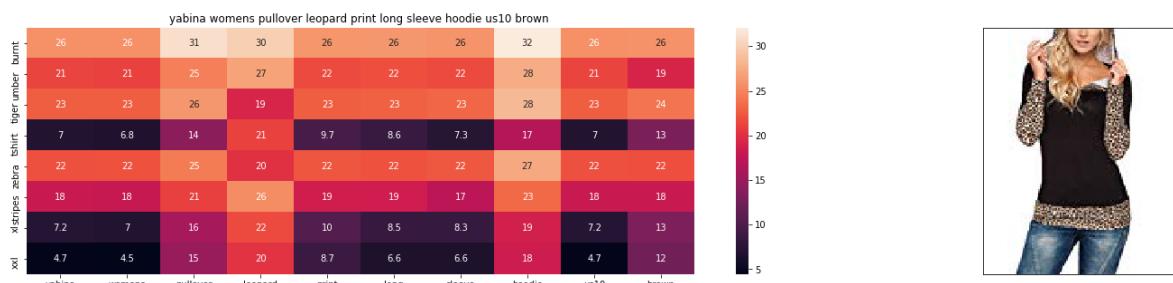


ASIN : B01LZ7BQ4H

Brand : WAYF

euclidean distance from input : 2.684906782301833

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

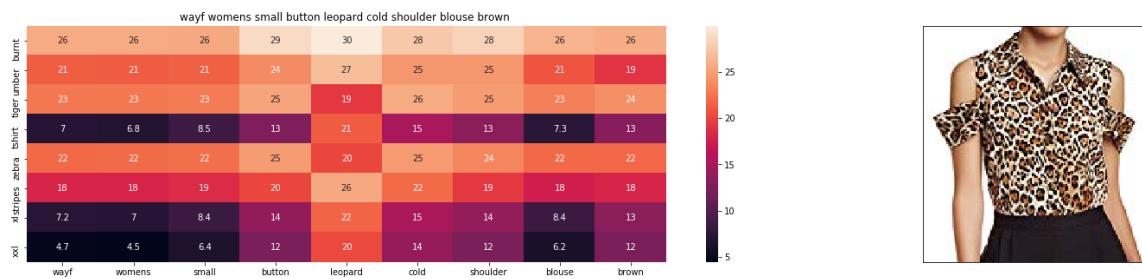


ASIN : B01KJUM6JI

Brand : YABINA

euclidean distance from input : 2.6858381926578345

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B01M06V4X1

Brand : WAYF

euclidean distance from input : 2.694761948650377

[10.2] Keras and Tensorflow to extract features

In [66]:

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

Using TensorFlow backend.

In [67]:

```
# https://gist.github.com/fchollet/f35fbc80e066a49d65f1688a7e99f069
# Code reference: https://blog.keras.io/building-powerful-image-classification-mode

# 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)
    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[67]:

```
"\n# dimensions of our images.\nimg_width, img_height = 224, 224\n\n\np_model_weights_path = 'bottleneck_fc_model.h5'\ntrain_data_dir = 'images2/'\nnb_train_samples = 16042\nepochs = 50\nbatch_size = 1\n\n\n\ndef save_bottlebeck_features():\n    \n    #Function to compute VGG-16 CNN\n    # for image feature extraction.\n    \n    asins = []\n    datagen = ImageDataGenerator(rescale=1. / 255)\n    \n    # build the VGG16 network\n    model = applications.VGG16(include_top=False, weights='imagenet')\n    generator = datagen.flow_from_directory(\n        train_data_dir,\n        target_size=(img_width, img_height),\n        batch_size=batch_size,\n        class_mode=None,\n        shuffle=False)\n\n    for i in generator.filenames:\n        asins.append(i[2:-5])\n\n    bottleneck_features_train = model.predict_generator(generator, nb_train_samples // batch_size)\n    bottleneck_features_train = bottleneck_features_train.reshape((16042,25088))\n    \n    np.save(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\ndef save_bottlebeck_features()\n\n"
```

[10.3] Visual features based product similarity.

In [68]:

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

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

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

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

    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 idx, 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.amazon.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.amazon.com/dp/B00JXQB5FQ



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



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



Product Title: brown white tiger tshirt tiger stripes xl xxl
Euclidean Distance from input image: 44.0002
Amazon Url: www.amazon.com/dp/B00JXQCWT0



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



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



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



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



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



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



Product Title: miss chievous juniors striped peplum tank top medium s
hadowpeach

Euclidean Distance from input image: 48.13128

Amazon Url: www.amazon.com/dp/B0177DM70S



Product Title: red pink floral heel sleeveless shirt xl xxl

Euclidean Distance from input image: 48.16945

Amazon Url: www.amazon.com/dp/B00JV63QQE



Product Title: moana logo adults hot v neck shirt black xxl

Euclidean Distance from input image: 48.25678

Amazon Url: www.amazon.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.amazon.com/dp/B01CR57YY0

Product Title: kawaii cotton pastel tops peach pink cactus design

Euclidean Distance from input image: 48.362583

Amazon Url: www.amazon.com/dp/B071WYLBZS



Product Title: chicago chicago 18 shirt women pink

Euclidean Distance from input image: 48.383648

Amazon Url: www.amazon.com/dp/B01GXAZTRY



Product Title: yichun womens tiger printed summer tshirts tops

Euclidean Distance from input image: 48.449345

Amazon Url: www.amazon.com/dp/B010NN9RX0



Product Title: nancy lopez whimsy short sleeve whiteblacklemon drop xs

Euclidean Distance from input image: 48.478893

Amazon Url: www.amazon.com/dp/B01MPX6IDX



Product Title: womens tops tees pastel peach ice cream cone print

Euclidean Distance from input image: 48.557983

Amazon Url: www.amazon.com/dp/B0734GRKZL



Product Title: uswomens mary j blige without tshirts shirt

Euclidean Distance from input image: 48.614395

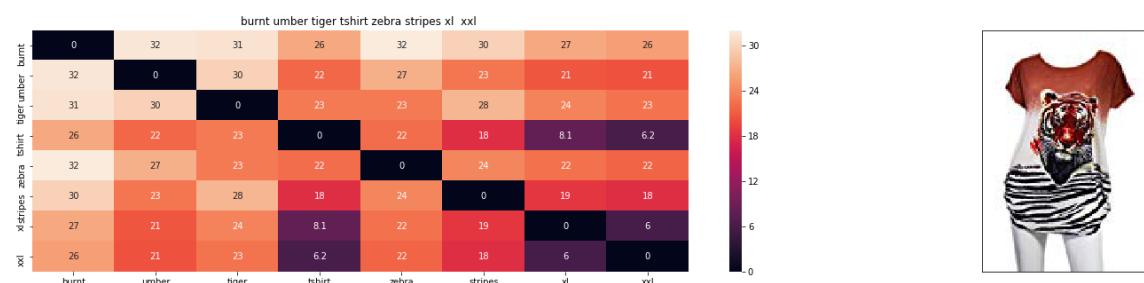
Amazon Url: www.amazon.com/dp/B01M0XXFKK

Eqaul Weightage

In [81]:

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

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B00JXQB5FQ

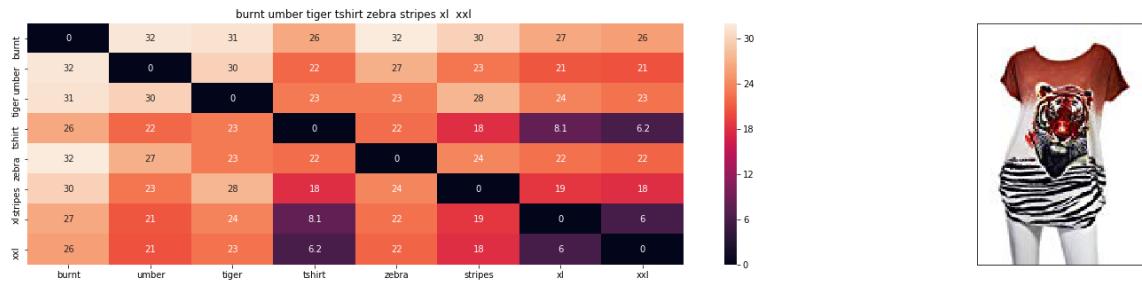
Brand : Si-Row

When Image feature has maximum weightage than 'Title' and 'Brand & Color'

In [82]:

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

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B00JXQB5FQ

Brand : Si Row

When 'Brand & Color' has maximum weightage than 'Image feature'

In [89]:

```
idf_w2v_brand(12566,20, 1,10,1)
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B00JXQB5FQ

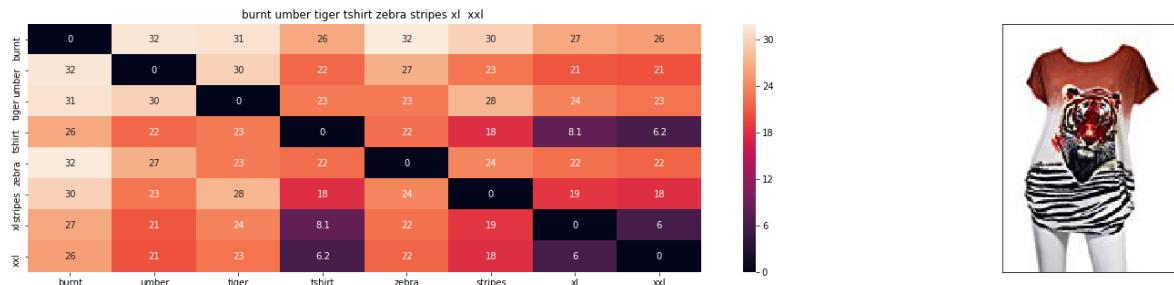
Brand : Si Row

When Image feature has maximum wieghtage, Title has 2nd highest weightage and 'Brand Color' has minimum weightage

In [97]:

```
idf_w2v_brand(12566, 20, 5, 1, 10)
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

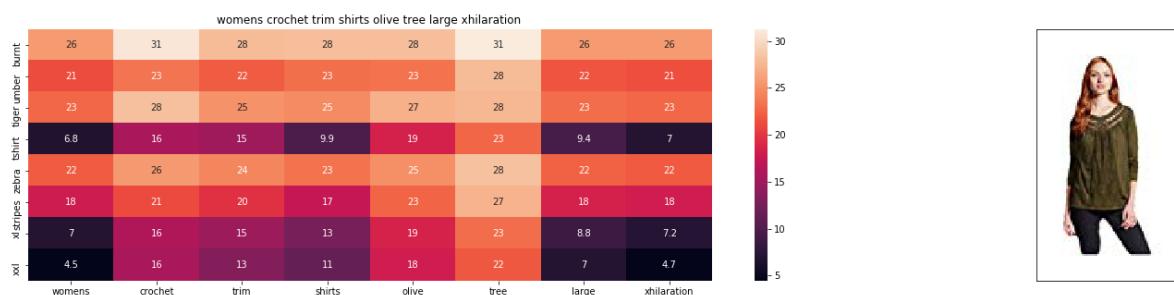


ASIN : B00JXQB5FQ

Brand : Si Row

euclidean distance from input : 0.001220703125

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

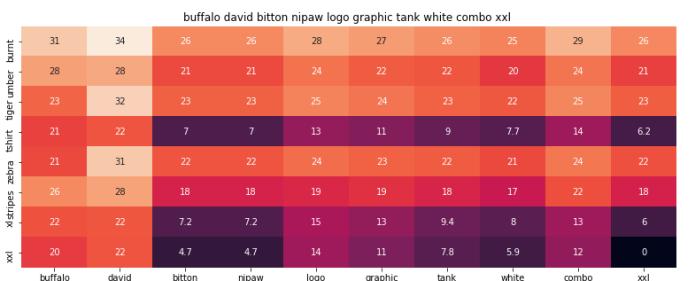


ASIN : B06XBHNM7J

Brand : Xhilaration

euclidean distance from input : 25.897770214125984

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

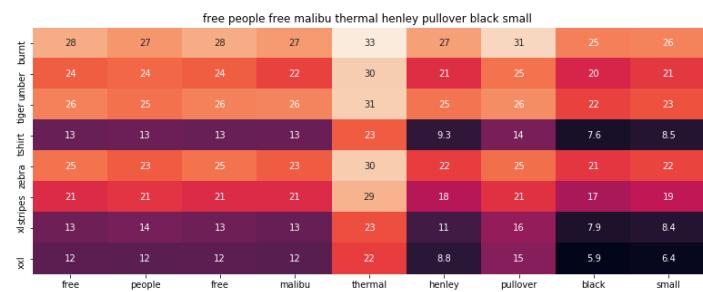


ASIN : B018H5AZXQ

Brand : Buffalo

euclidean distance from input : 27.222195673034065

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

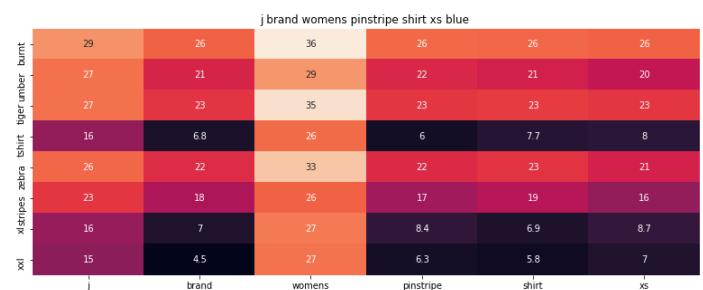


ASIN : B074MXY984

Brand : We The Free

euclidean distance from input : 27.424663305282593

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



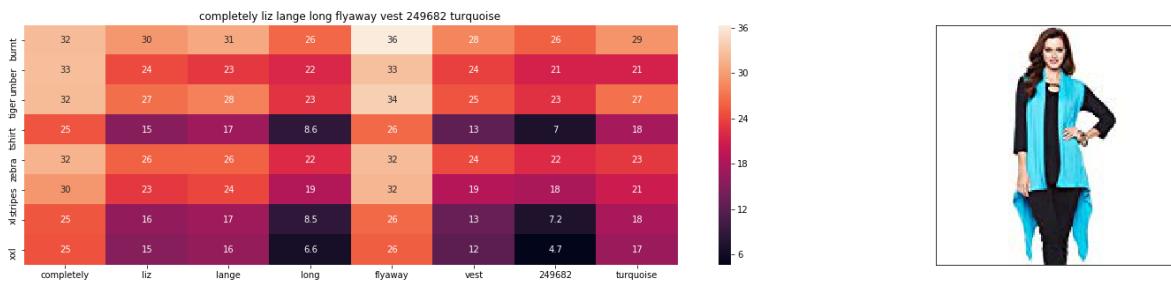
ASIN : B06XYP1X1F

Brand : J Brand Jeans

euclidean distance from input : 27.477273968445584



Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

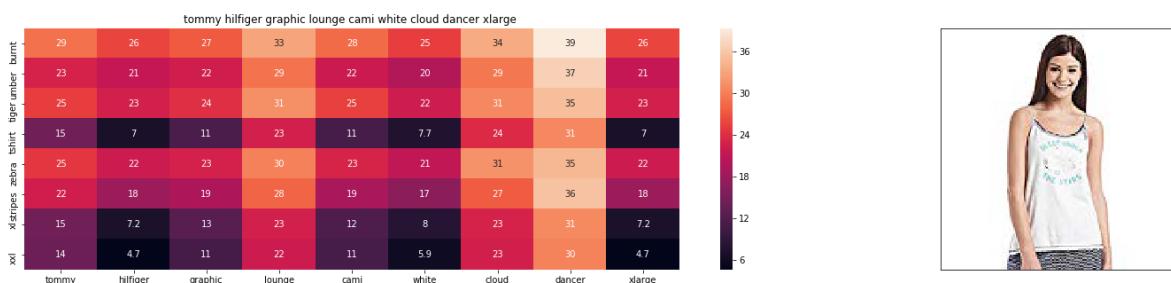


ASIN : B074LTBWSW

Brand : Liz Lange

euclidean distance from input : 27.49141745571839

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

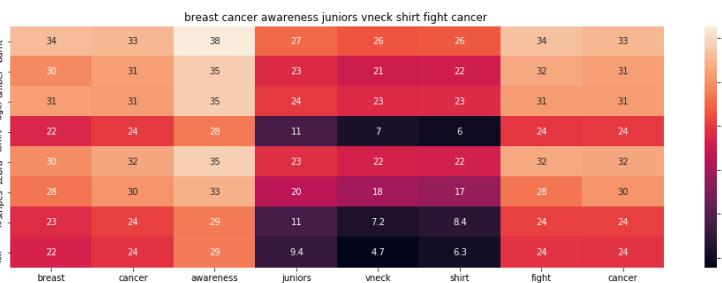


ASIN : B01BMSFYW2

Brand : igertommy hilf

euclidean distance from input : 27.827243832337185

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

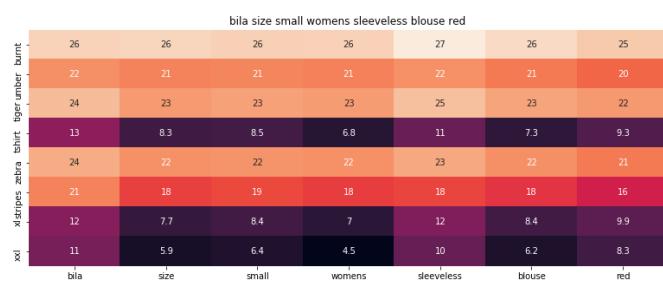


ASIN : B016CU40IY

Brand : Juiceclouds

euclidean distance from input : 27.914144952699594

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

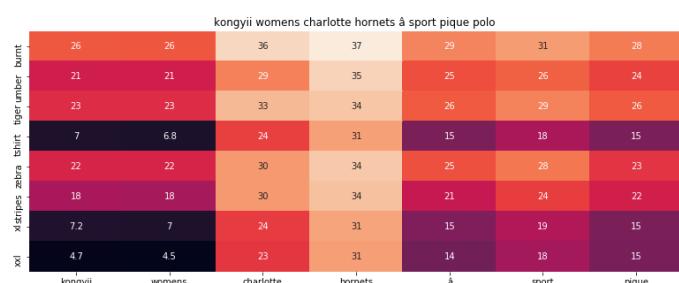


ASIN : B01L7R0ZNC

Brand : Bila

euclidean distance from input : 28.188810546800546

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



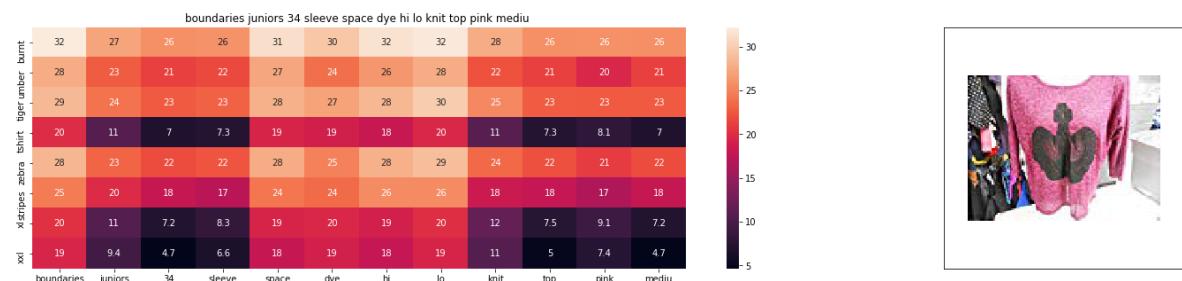
ASIN : B01FJVZST2

Brand : KONGYII



euclidean distance from input : 28.52063341896241

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B01EXXFS4M

Brand : No Boundaries

euclidean distance from input : 28.787794876143806

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

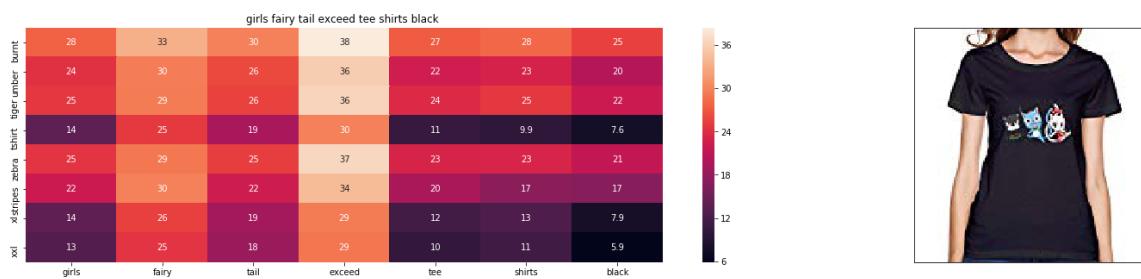


ASIN : B0756JTS1F

Brand : Salvatore Ferragamo

euclidean distance from input : 28.911019802093506

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

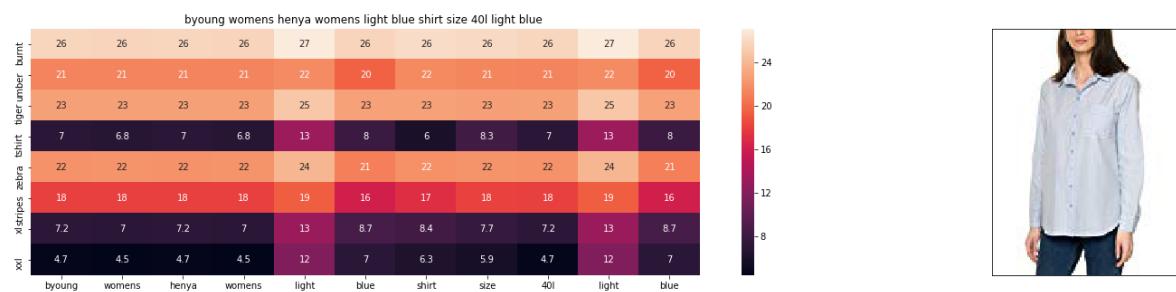


ASIN : B01L9F153U

Brand : ATYPEMIX

euclidean distance from input : 28.92216058532899

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

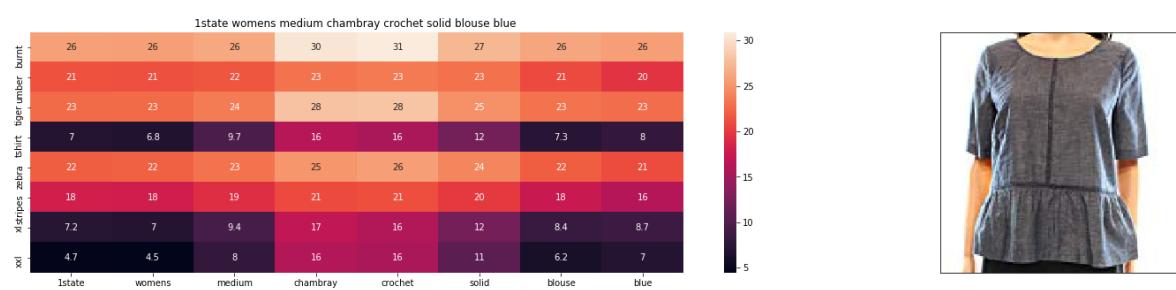


ASIN : B06Y41MRCH

Brand : Byoung

euclidean distance from input : 28.924588012740486

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

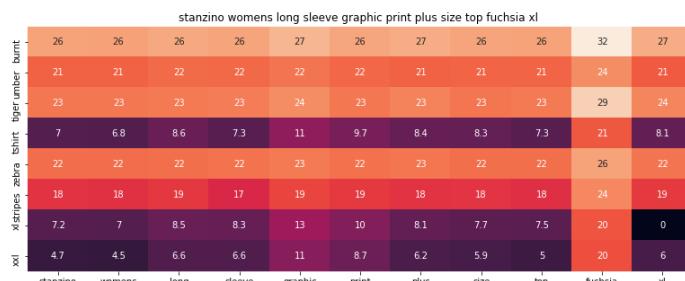


ASIN : B074MK6LV2

Brand : 1.State

euclidean distance from input : 28.97533508102601

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B00DP4VHWI

Brand : Stanzino

euclidean distance from input : 29.067574699327402

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B01DNNI1R0

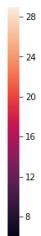
Brand : Usstore

euclidean distance from input : 29.15367822651612

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



hot sexy fashion women loose chiffon short sleeve tops blouse shirt											
27	27	27	26	27	29	27	26	26	26	26	26
24	22	22	22	23	21	22	22	22	21	22	22
25	24	24	23	24	27	23	23	24	23	23	23
15	11	10	9	12	15	9.3	7.3	9.6	7.3	6	
25	23	23	23	22	24	23	22	23	22	22	
21	20	19	19	20	20	19	17	18	18	17	
15	12	11	9.3	12	16	9.6	8.3	9.8	8.4	8.4	
14	10	9.2	7.5	10	15	7.7	6.6	8.1	6.2	6.3	
hot	sexy	fashion	women	loose	chiffon	short	sleeve	tops	blouse	shirt	



ASIN : B00JMAASRO

Brand : Wotefusi

euclidean distance from input : 29.189475019380502

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

sexy sheer mesh print long sleeves bodysuit											
27	28	29	26	26	29	35					
22	23	23	22	22	24	32					
24	25	25	23	23	23	31					
11	13	14	9.7	8.6	14	24					
23	25	23	22	22	26	29					
20	21	19	19	19	19	27					
12	13	14	10	8.5	15	25					
10	12	13	8.7	6.6	14	24					
sexy	sheer	mesh	print	long	sleeves	bodysuit					



ASIN : B074Z5C98D

Brand : Ariella's closet

euclidean distance from input : 29.238545725117426

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

maven west striped sleeveless lace peplum peasant blouse yellow large											
41	29	27	27	27	31	28	26	27	26		
37	25	20	22	21	22	26	21	20	22		
37	27	23	25	24	28	26	23	24	23		
30	17	11	11	10	17	17	7.3	13	9.4		
36	25	20	23	23	25	26	22	21	22		
35	23	16	18	17	21	24	18	17	18		
31	17	12	12	11	17	18	8.4	13	8.8		
30	16	10	10	9.3	16	17	6.2	12	7		
maven	west	striped	sleeveless	lace	peplum	peasant	blouse	yellow	large		



ASIN : B01M8GB3AL

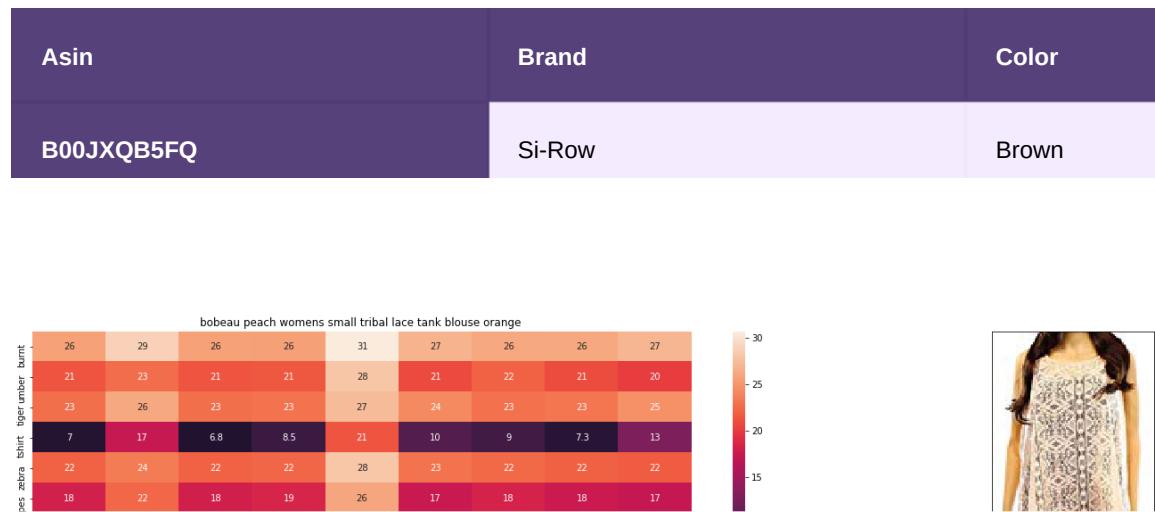
Brand : Maven West
euclidean distance from input : 29.270747232482307

When Title has max weightage, 'Brand&color' has second highest weightage

In [99]:

```
idf_w2v_brand(12566,20, 10,5,1)
```

Brand : Exotic India
euclidean distance from input : 8.214922189938385



Summary and Procedure

- We have considered 16k datapoints for above experiment
- We have built 'idf-w2v-weightage' , 'Brand & Color' and 'Image feature' weightage feature model.
- We have given max and min weightage to these features interchanginly.
- We have view various scenario, we can say that when we gives max importance to 'Brand & Color' model recommended similar brand product
- As when gives max importance to 'Title' model recommended items based on Query title.