Amazon Apparel Recommendations

[4.3] Overview of the data

In [41]:

```
#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 [0]:

```
# 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 [0]:

```
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 [0]:

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

Out[35]:

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

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

1. asin (Amazon standard identification number)
2. brand (brand to which the product belongs to)
3. color (Color information of apparel, it can contain many colors as a value ex: red 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 [0]:

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

In [0]:

```
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[37]:

	asin	brand	color	medium_image_url	product_type_name
0	B016I2TS4W	FNC7C	None	https://images-na.ssl-images-amazon.com/images...	SHIRT
1	B01N49AI08	FIG Clothing	None	https://images-na.ssl-images-amazon.com/images...	SHIRT
2	B01JDPCOHO	FIG Clothing	None	https://images-na.ssl-images-amazon.com/images...	SHIRT
3	B01N19U5H5	Focal18	None	https://images-na.ssl-images-amazon.com/images...	SHIRT
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT

[5.1] Missing data for various features.

Basic stats for the feature: product_type_name

In [0]:

```
# 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 [0]:

```
# 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' 'SLEEPWE
AR'
 'ORCA_SHIRT' 'HANDBAG' 'PET_SUPPLIES' 'SHOES' 'KITCHEN' 'ADUL
T_COSTUME'
 'HOME_BED_AND_BATH' 'MISC_OTHER' 'BLAZER' 'HEALTH_PERSONAL_CA
RE'
 'TOYS_AND_GAMES' 'SWIMWEAR' 'CONSUMER_ELECTRONICS' 'SHORTS'
'HOME'
 'AUTO_PART' 'OFFICE_PRODUCTS' 'ETHNIC_WEAR' 'BEAUTY'
 'INSTRUMENT_PARTS_AND_ACCESSORIES' 'POWERSPORTS_PROTECTIVE_GE
AR' 'SHIRTS'
 'ABIS_APPAREL' 'AUTO_ACCESSORY' 'NONAPPARELMISC' 'TOOLS' 'BAB
Y_PRODUCT'
 'SOCKSHOSIERY' 'POWERSPORTS_RIDING_SHIRT' 'EYEWEAR' 'SUIT'
 'OUTDOOR_LIVING' 'POWERSPORTS_RIDING_JACKET' 'HARDWARE' 'SAFE
TY_SUPPLY'
 'ABIS_DVD' 'VIDEO_DVD' 'GOLF_CLUB' 'MUSIC_POPULAR_VINYL'
 'HOME_FURNITURE_AND_DECOR' 'TABLET_COMPUTER' 'GUILD_ACCESSORI
ES'
 'ABIS_SPORTS' 'ART_AND_CRAFT_SUPPLY' 'BAG' 'MECHANICAL_COMPON
ENTS'
 'SOUND_AND_RECORDING_EQUIPMENT' 'COMPUTER_COMPONENT' 'JEWELR
Y'
 'BUILDING_MATERIAL' 'LUGGAGE' 'BABY_COSTUME' 'POWERSPORTS_VEH
ICLE_PART'
 'PROFESSIONAL_HEALTHCARE' 'SEEDS_AND_PLANTS' 'WIRELESS_ACCESS
ORY']
```

In [0]:

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

Out[40]:

```
[('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 [0]:

```
# 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 [0]:

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

Out[42]:

```
[('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 [0]:

```
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 [0]:

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

Out[44]:

```
[(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 [0]:

```
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 [0]:

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

Out[46]:

```
[(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 [0]:

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

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

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

In [0]:

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

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

In [0]:

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

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

In [0]:

```
# consider products which have color information
# data['color'].isnull() => gives the information about the dataframe row's w
data =data.loc[~data['color'].isnull()]
print('Number of data points After eliminating color=NULL :', data.shape[0])
```

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

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

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

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

In [0]:

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

In [0]:

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


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

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


'''
```

Out[52]:

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

[5.2] Remove near duplicate items

[5.2.1] Understand about duplicates.

In [0]:

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

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

2325

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

 :B00AQ4GMCK  :B00AQ4GMTS

 :B00AQ4GMLQ  :B00AQ4GN3I

These shirts exactly same except in color

 :B00G278GZ6  :B00G278W6O

 :B00G278Z2A  :B00G2786X8

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

[5.2.2] Remove duplicates : Part 1

In [0]:

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

In [0]:

```
data.head()
```

Out[103]:

	asin	brand	color	medium_image_url	product_type_name
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT
6	B012YX2ZPI	HX-Kingdom Fashion T-shirts	White	https://images-na.ssl-images-amazon.com/images...	SHIRT
11	B001LOUGE4	Fitness Etc.	Black	https://images-na.ssl-images-amazon.com/images...	SHIRT
15	B003BSRPB0	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	SHIRT
21	B014ICEDNA	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT

In [0]:

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

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

In [0]:

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

Out[105]:

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

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 X
XL 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 Rainbo
w Animal Print Head Shirt for woman Neon Wolf t-shirt
62. psychedelic colorful Howling Galaxy Wolf T-shirt/Colorful Rainbo
w Animal Print Head Shirt for woman Neon Wolf t-shirt
63. psychedelic colorful Howling Galaxy Wolf T-shirt/Colorful Rainbo
w Animal Print Head Shirt for woman Neon Wolf t-shirt
64. psychedelic colorful Howling Galaxy Wolf T-shirt/Colorful Rainbo
w Animal Print Head Shirt for woman Neon Wolf t-shirt
```

In [0]:

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

In [0]:

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

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

        # store the list of words of jth string in b, ex: b = ['tokidoki', 'T
        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
        count  = 0

        # itertools.zip_longest(a,b): will map the corresponding words in bot
        # example: a =['a', 'b', 'c', 'd']
        # b = ['a', 'b', 'd']
        # itertools.zip_longest(a,b): will give [('a','a'), ('b','b'), ('c',
        for k in itertools.zip_longest(a,b):
            if (k[0] == k[1]):
                count += 1

        # if the number of words in which both strings differ are > 2 , we ar
        # if the number of words in which both strings differ are < 2 , we ar
        if (length - count) > 2: # number of words in which both sensences di
            # if both strings are differ by more than 2 words we include the
            stage1_dedupe_asins.append(data_sorted['asin'].loc[indices[i]])

            # if the comaprision between is between num_data_points, num_data
            if j == num_data_points-1: stage1_dedupe_asins.append(data_sorted

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

In [0]:

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

We removed the dupliactes which differ only at the end.

In [0]:

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

```
Number of data points :  17593
```

In [0]:

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

[5.2.3] Remove duplicates : Part 2

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

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

Examples:

Titles-1
86261.  UltraClub Women's Classic Wrinkle-Free Long Sleeve Oxford Sh
irt, Pink, XX-Large
115042. UltraClub Ladies Classic Wrinkle-Free Long-Sleeve Oxford Lig
ht 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 Te
es
120832. EVALY Women's New University Of UTAH 3/4-Sleeve Raglan Tshir
t
```

In [0]:

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

In [0]:

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

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

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

        b = data['title'].loc[j].split()
        # store the list of words of jth string in b, ex: b = ['tokidoki', 'T

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

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

        # itertools.zip_longest(a,b): will map the corresponding words in bot
        # example: a =['a', 'b', 'c', 'd']
        # b = ['a', 'b', 'd']
        # itertools.zip_longest(a,b): will give [('a','a'), ('b','b'), ('c',
        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 ar
        if (length - count) < 3:
            indices.remove(j)
```

In [0]:

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

In [0]:

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

```
Number of data points after stage two of dedupe:  16042
```

In [0]:

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

6. Text pre-processing

In [0]:

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

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

In [0]:

```
# 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: {'such', 'and', 'hers', 'up', 'she', 'd',
'further', 'all', 'than', 'under', 'is', 'off', 'both', 'mos
t', 'few', 'should', 're', 'very', 'just', 'then', 'didn', 'my
self', 'in', 'too', 's', 'shouldn', 'herself', 'because', 'ho
w', 'itself', 'what', 'shan', 'weren', 'doing', 'them', 'could
n', 'their', 'so', 'ain', 'haven', 'yourself', 'now', 'll', 'i
sn', 'about', 'over', 'into', 'before', 'during', 'on', 'as',
'aren', 'against', 'above', 'down', 'they', 'below', 'me', 'ag
ain', 'for', 'why', 'been', 'yourselves', 'more', 'her', 'tha
t', 'can', 'am', 'was', 'themselves', 'mightn', 'does', 'thos
e', 'only', 'hasn', 'any', 'ma', 'are', 'nor', 'out', 'you',
'ourselves', 'the', 'an', 'has', 'where', 'i', 'while', 'our
s', 'its', 'your', 'had', 'were', 'being', 'no', 'or', 'need
n', 've', 'y', 'a', 'each', 'have', 'through', 'when', 'must
n', 'by', 'won', 'from', 'own', 'will', 'there', 't', 'him',
'these', 'doesn', 'theirs', 'my', 'did', 'of', 'who', 'until',
'wouldn', 'we', 'do', 'having', 'yours', 'other', 'wasn', 'i
t', 'with', 'once', 'here', 'don', 'o', 'whom', 'this', 'if',
'but', 'hadn', 'our', 'some', 'm', 'not', 'between', 'himsel
f', 'same', 'at', 'be', 'he', 'after', 'which', 'to', 'his'}
```

In [0]:

```
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")
```

```
3.5727220000000006 seconds
```

In [0]:

```
data.head()
```

Out[6]:

	asin	brand	color	medium_image_url	product_type_name
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT
6	B012YX2ZPI	HX-Kingdom Fashion T-shirts	White	https://images-na.ssl-images-amazon.com/images...	SHIRT
15	B003BSRPB0	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	SHIRT
27	B014ICEJ1Q	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT
46	B01NACPBG2	Fifth Degree	Black	https://images-na.ssl-images-amazon.com/images...	SHIRT

In [0]:

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

Stemming

In [0]:

```
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 [3]:

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

Out[3]:

	asin	brand	color	medium_image_url	product_type_name
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT
6	B012YX2ZPI	HX-Kingdom Fashion T-shirts	White	https://images-na.ssl-images-amazon.com/images...	SHIRT
15	B003BSRPB0	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	SHIRT
27	B014ICEJ1Q	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT
46	B01NACPBG2	Fifth Degree	Black	https://images-na.ssl-images-amazon.com/images...	SHIRT

In [4]:

```
# 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 occurer
        # labels: len(labels) == len(keys), the values of labels depends on t
                # if model == 'bag of words': labels(i) = values(i)
                # if model == 'tfidf weighted bag of words':labels(i) = tfidf
                # if model == 'idf weighted bag of words':labels(i) = idf(key
        # 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
    ax = plt.subplot(gs[0])
    # it displays a cell in white color if the word is intersection(lis of wo
    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-ax
    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 contr
    intersection = set(vec1.keys()) & set(vec2.keys())

    # we set the values of non intersecting words to zero, this is just to sh
    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 o
    values = [vec2[x] for x in vec2.keys()]

    # labels: len(labels) == len(keys), the values of labels depends on the m
        # 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
            # tfidf_title_features[doc_id, index_of_word_in_corpus] will give
            if x in  tfidf_title_vectorizer.vocabulary_:
                labels.append(tfidf_title_features[doc_id, tfidf_title_vector
            else:
                labels.append(0)
    elif model == 'idf':
        labels = []
        for x in vec2.keys():
            # idf_title_vectorizer.vocabulary_ it contains all the words in t
            # idf_title_features[doc_id, index_of_word_in_corpus] will give t
            if x in  idf_title_vectorizer.vocabulary_:
                labels.append(idf_title_features[doc_id, idf_title_vectorizer
            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 = te
    return Counter(words) # Counter counts the occurence of each word in list



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

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

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

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

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

In [5]:

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

# What is a sparse vector?

# title_features[doc_id, index_of_word_in_corpus] = number of times the word
```

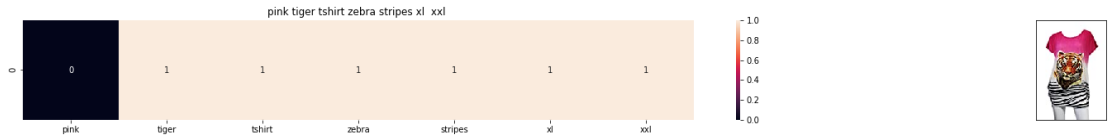
Out[5]:

```
(16042, 12609)
```

In [6]:

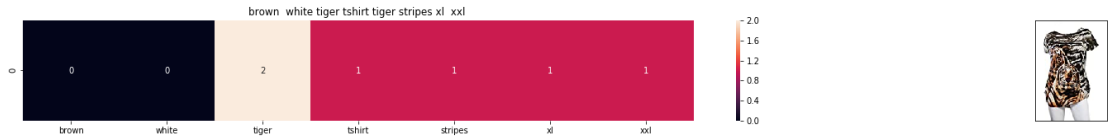
```
def bag_of_words_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

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

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

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

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

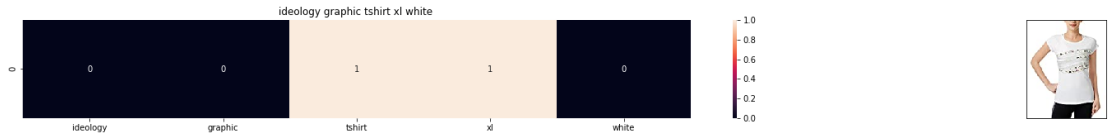
Brand: Si Row
Title: pink tiger tshirt zebra stripes xl xxl
Euclidean similarity with the query image : 1.7320508075688
772
===
=



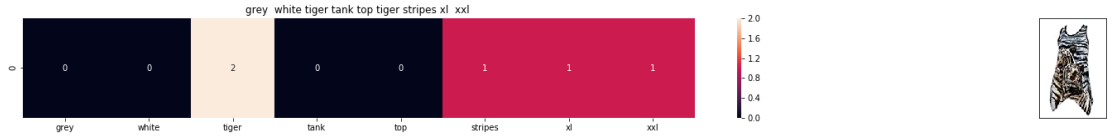
ASIN : B00JXQCWTO
Brand: Si Row
Title: brown white tiger tshirt tiger stripes xl xxl
Euclidean similarity with the query image : 2.449489742783178
===

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
===

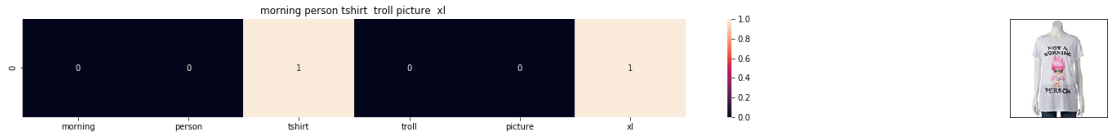


ASIN : B01NB0NKRO
Brand: Ideology
Title: ideology graphic tshirt xl white
Euclidean similarity with the query image : 3.0
===



ASIN : B00JXQAFZ2
Brand: Si Row

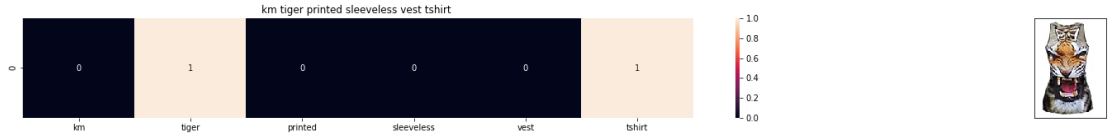
Title: grey white tiger tank top tiger stripes xl xxl
Euclidean similarity with the query image : 3.0
==



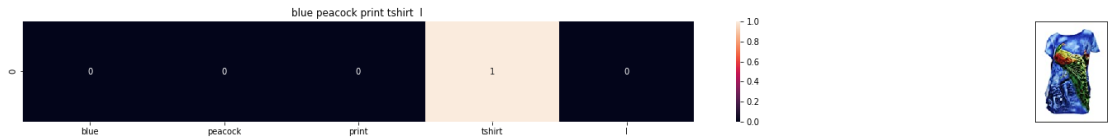
ASIN : B01CLS8LMW
Brand: Awake
Title: morning person tshirt troll picture xl
Euclidean similarity with the query image : 3.1622776601683795
==

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
==



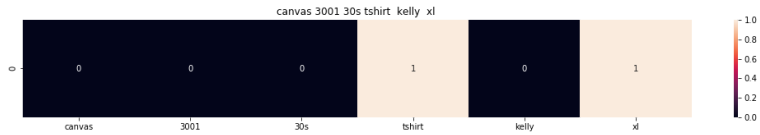
ASIN : B012VQLT6Y
Brand: KM T-shirt
Title: km tiger printed sleeveless vest tshirt
Euclidean similarity with the query image : 3.1622776601683795
==



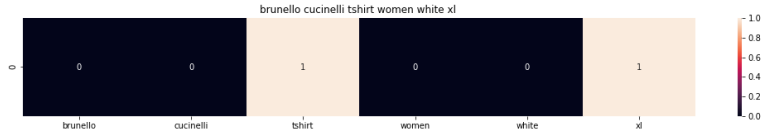
ASIN : B00JXQC8L6
Brand: Si Row
Title: blue peacock print tshirt l
Euclidean similarity with the query image : 3.1622776601683795
==

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 : 3.1622776601683
795
============================================================
=
```

```
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
============================================================
```

[8.5] TF-IDF based product similarity

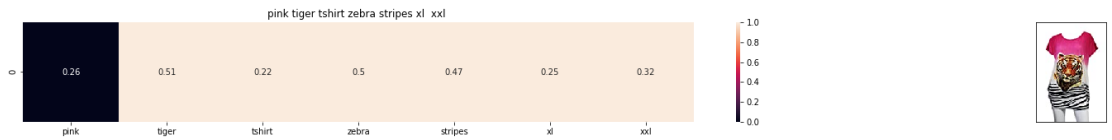
In [7]:

```
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 di
# tfidf_title_features[doc_id, index_of_word_in_corpus] = tfidf values of the
```

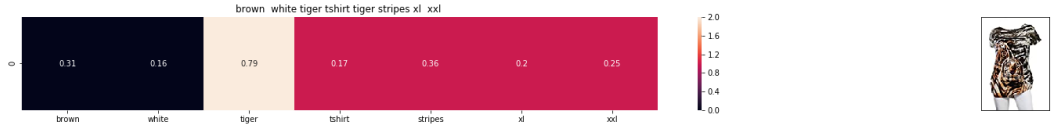
In [8]:

```
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 r
    # the metric we used here is cosine, the coside distance is mesured as K(
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(tfidf_title_features,tfidf_title_featu

    # 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'
        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
```


```
ASIN : B00JXQB5FQ
BRAND : Si Row
Eucliden distance from the given image : 0.0
=============================================================
=============================================================
=
```




```
ASIN : B00JXQASS6
BRAND : Si Row
Eucliden distance from the given image : 0.7536331912451363
=============================================================
=============================================================
=
```

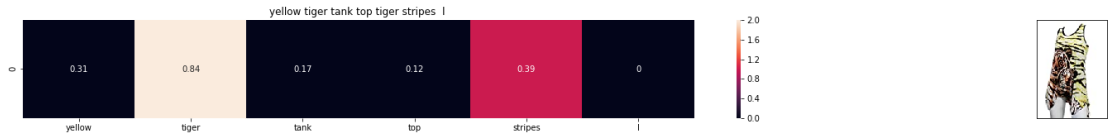


ASIN : B00JXQCWTO
BRAND : Si Row
Eucliden distance from the given image : 0.9357643943769647
==========
==========
=

ASIN : B00JXQAFZ2
BRAND : Si Row
Eucliden distance from the given image : 0.9586153524200749
==========
==========
=

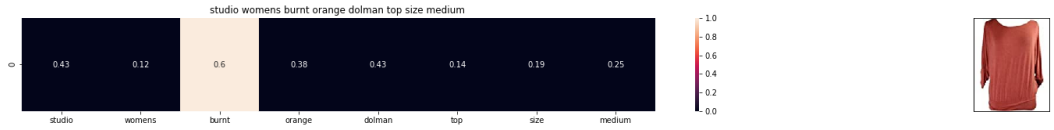
ASIN : B00JXQCUIC
BRAND : Si Row
Eucliden distance from the given image : 1.000074961446881
==========
==========
=

ASIN : B00JXQAO94
BRAND : Si Row
Eucliden distance from the given image : 1.023215552457452
==========
==========
=

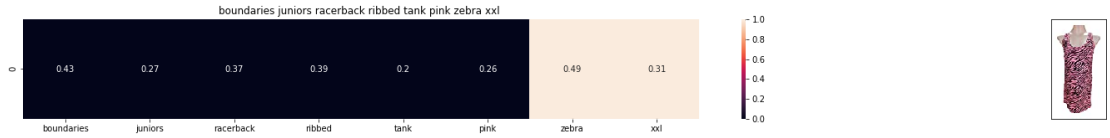


ASIN : B00JXQAUWA
BRAND : Si Row
Eucliden distance from the given image : 1.031991846303421
==========

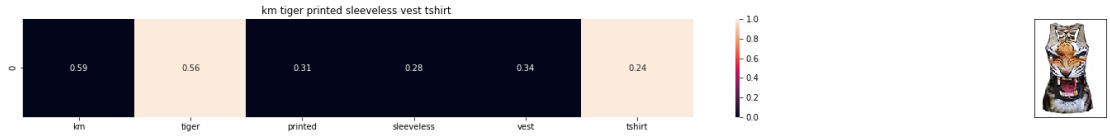
==========



ASIN : B06XSCVFT5
BRAND : Studio M
Eucliden distance from the given image : 1.2106843670424716
==========
==========
=

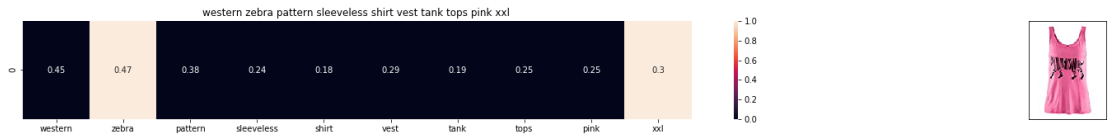


ASIN : B06Y2GTYPM
BRAND : No Boundaries
Eucliden distance from the given image : 1.2121683810720831
==========
==========
=



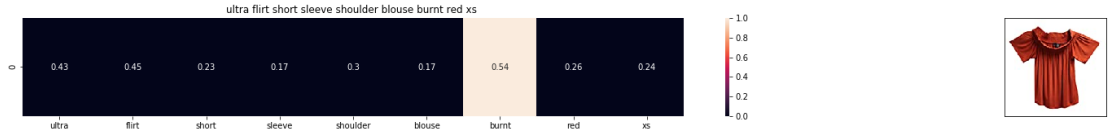
ASIN : B012VQLT6Y
BRAND : KM T-shirt
Eucliden distance from the given image : 1.219790640280982
==========
==========
=

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

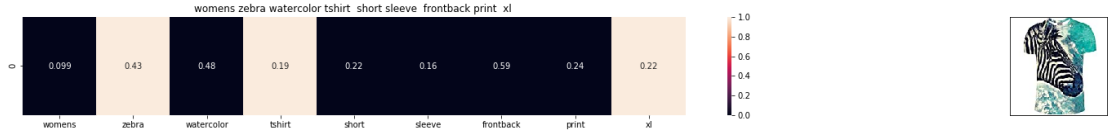


ASIN : B00Z6HEXWI
BRAND : Black Temptation
Eucliden distance from the given image : 1.221281392120943

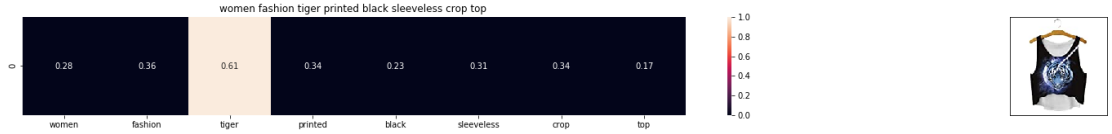
==========
==========
=======



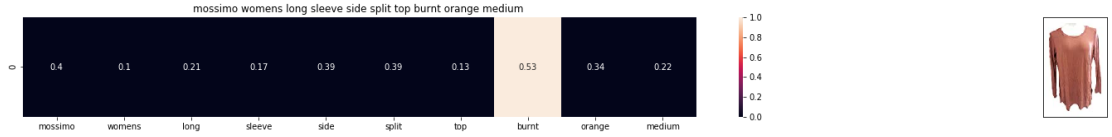
ASIN : B074TR12BH
BRAND : Ultra Flirt
Eucliden distance from the given image : 1.2313364094597743
==========
==========
=



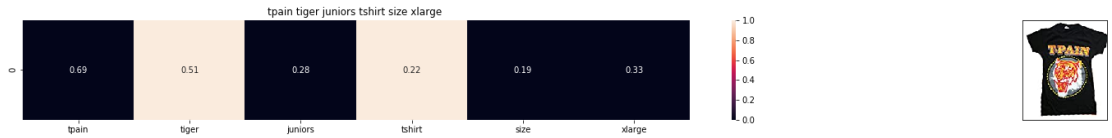
ASIN : B072R2JXKW
BRAND : WHAT ON EARTH
Eucliden distance from the given image : 1.2318451972624516
==========
==========
=



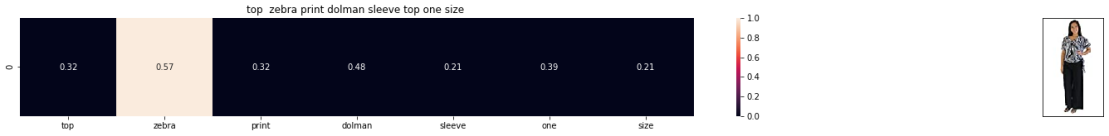
ASIN : B074T8ZYGX
BRAND : MKP Crop Top
Eucliden distance from the given image : 1.2340607457359425
==========
==========
=



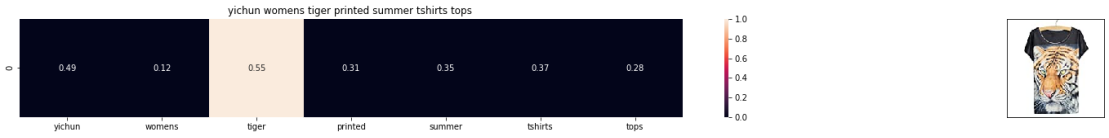
ASIN : B071ZDF6T2
BRAND : Mossimo
Eucliden distance from the given image : 1.2352785577664824
==========
==========
=



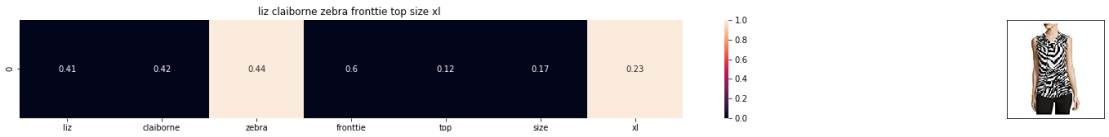
```
ASIN : B01K0H02OG
BRAND : Tultex
Eucliden distance from the given image : 1.236457298812782
============================================================
============================================================
=======
```




```
ASIN : B00H8A6ZLI
BRAND : Vivian's Fashions
Eucliden distance from the given image : 1.24996155052848
============================================================
============================================================
=
```




```
ASIN : B010NN9RXO
BRAND : YICHUN
Eucliden distance from the given image : 1.2535461420856102
============================================================
============================================================
=
```




```
ASIN : B06XBY5QXL
BRAND : Liz Claiborne
Eucliden distance from the given image : 1.2538832938357722
============================================================
============================================================
=
```

[8.5] IDF based product similarity

In [9]:

```
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 di
# idf_title_features[doc_id, index_of_word_in_corpus] = number of times the w
```

In [10]:

```
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 [11]:

```
# 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 wit
    # idf_title_features[:, idf_title_vectorizer.vocabulary_[i]].nonzero()[0]
    for j in idf_title_features[:, idf_title_vectorizer.vocabulary_[i]].nonze

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

In [12]:

```
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 r
    # the metric we used here is cosine, the coside distance is mesured as K(
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(idf_title_features,idf_title_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)):
        get_result(indices[i],data['title'].loc[df_indices[0]], data['title']
        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 wo
```

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
==========
==========

[9] Text Semantics based product similarity

In [13]:

```
# 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[13]:

```
'\n# Set values for various parameters\nnum_features = 300
# Word vector dimensionality                      \nmin_word_c
ount = 1    # Minimum word count                        \nnum_
workers = 4       # Number of threads to run in parallel\ncont
ext = 10          # Context window size
\ndownsampling = 1e-3   # Downsample setting for frequent word
s\n\n# Initialize and train the model (this will take some tim
e)\nfrom gensim.models import word2vec\nprint ("Training mode
l...")\nmodel = word2vec.Word2Vec(sen_corpus, workers=num_work
ers,             size=num_features, min_count = min_word_coun
t,            window = context)\n    \n'
```

In [14]:

```
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.bir
'''

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

In [15]:

```
# 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 representatio
        # if m_name == 'weighted', we will multiply each w2v[word] with the i
    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.vo
            elif m_name == 'avg':
                vec.append(model[i])
        else:
            # if the word in our courpus is not there in the google word2vec
            vec.append(np.zeros(shape=(300,)))
    # we will return a numpy array of shape (#number of words in title * 300
    # each row represents the word2vec representation of each word (weighted/
    return  np.array(vec)

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

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

```
    s2_vec = get_word_vec(sentence2, doc_id2, model)

    # s1_s2_dist = np.array(#number of words in title1 * #number of words in
    # 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 dis
    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 [16]:

```
# 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
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 representatic
        # if m_name == 'weighted', we will multiply each w2v[word] with the i

    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.vocabul
                featureVec = np.add(featureVec, idf_title_features[doc_id, ic
            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
```

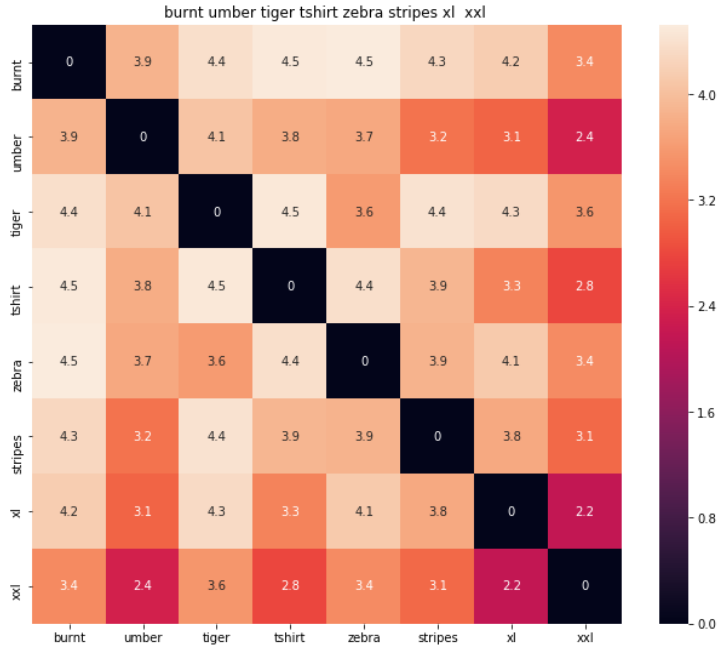
[9.2] Average Word2Vec product similarity.

In [17]:

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

In [18]:

```
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

    # 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_ir
        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 wor
```



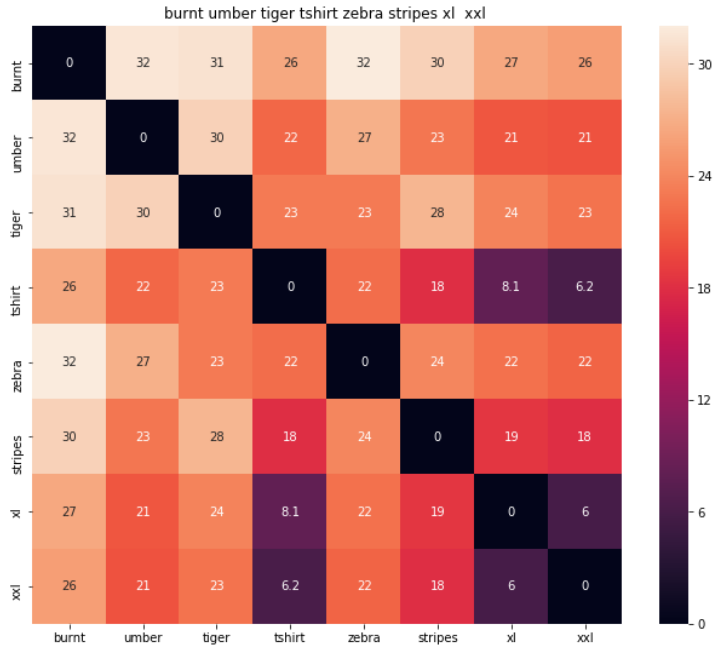
[9.4] IDF weighted Word2Vec for product similarity

In [19]:

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

In [20]:

```
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 r
    # the metric we used here is cosine, the coside distance is mesured as K(
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(w2v_title_weight, w2v_title_weight[doc

    # 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_ir
        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 wor
```



[9.6] Weighted similarity using brand and color.

In [21]:

```
# 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)).tocs
```

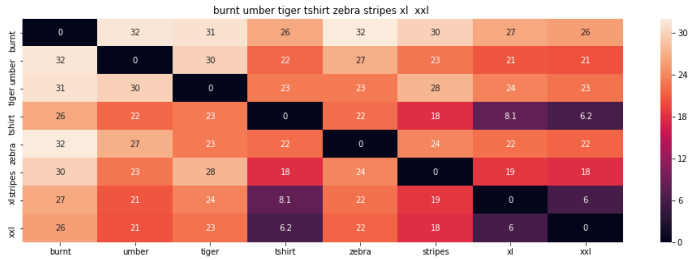
In [22]:

```
def heat_map_w2v_brand(sentance1, sentance2, url, doc_id1, doc_id2, df_id1, d

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

    # s1_s2_dist = np.array(#number of words in title1 * #number of words in
    # 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], ty
                   [data['asin'].loc[df_id2],brands[doc_id2], colors[doc_id2], ty

    colorscale = [[0, '#1d004d'],[.5, '#f2e5ff'],[1, '#f2e5d1']] # to color t

    # 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()
```

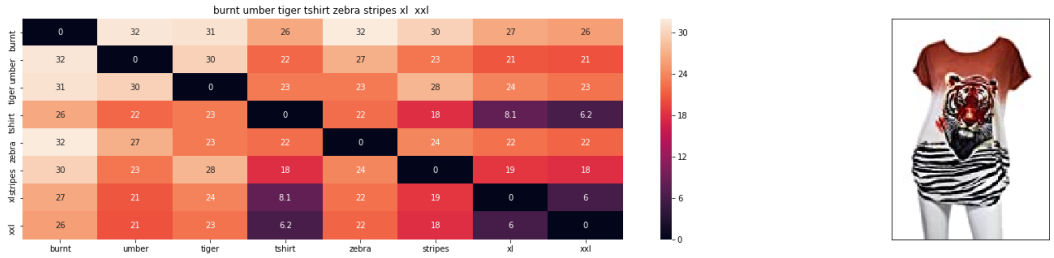
In [23]:

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

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

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

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


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

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



In [67]:

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

idf_w2v_brand(12566, 50, 10, 20)
```



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

[10.2] Keras and Tensorflow to extract features

In [25]:

```
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 [26]:

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



# 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

# 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_s
    bottleneck_features_train = bottleneck_features_train.reshape((16042,2508
```

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


save_bottlebeck_features()

'''
```

Out[26]:

```
"\n# dimensions of our images.\nimg_width, img_height = 224, 2
24\n\ntop_model_weights_path = 'bottleneck_fc_model.h5'\ntrain
_data_dir = 'images2/'\nnb_train_samples = 16042\nepochs = 50
\nbatch_size = 1\n\n\ndef save_bottlebeck_features():\n    \n
#Function to compute VGG-16 CNN for image feature extractio
n.\n    \n    asins = []\n    datagen = ImageDataGenerator(res
cale=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_d
ir,\n        target_size=(img_width, img_height),\n        bat
ch_size=batch_size,\n        class_mode=None,\n        shuffle
=False)\n\n    for i in generator.filenames:\n        asins.ap
pend(i[2:-5])\n\n    bottleneck_features_train = model.predict
_generator(generator, nb_train_samples // batch_size)\n    bot
tleneck_features_train = bottleneck_features_train.reshape((16
042,25088))\n    \n    np.save(open('16k_data_cnn_features.np
y', 'wb'), bottleneck_features_train)\n    np.save(open('16k_d
ata_cnn_feature_asins.npy', 'wb'), np.array(asins))\n    \n\ns
ave_bottlebeck_features()\n\n"
```

[10.3] Visual features based product similarity.

In [28]:

```
#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('pickle/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_

    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[ind
        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: 6.32596e-06
Amazon Url: www.amzon.com/dp/B00JXQB5FQ
```

Assignment

Function for recomendations based on weights

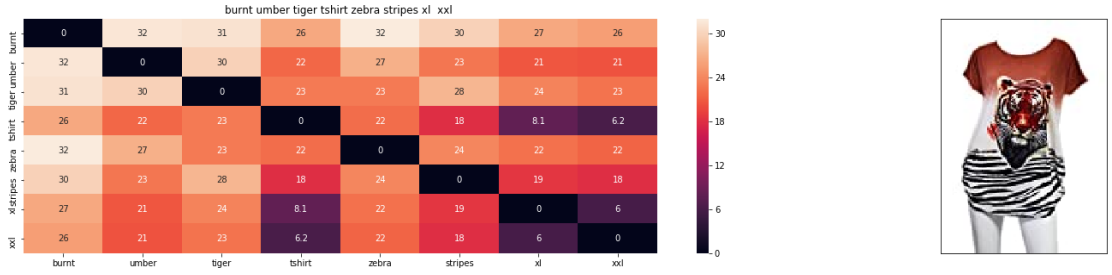
In [79]:

```
def idf_w2v_brand_image(doc_id, w1, w2, w3, w4, num_results):

    idf_w2v_dist  = pairwise_distances(w2v_title_weight, w2v_title_weight[doc
    brand_dist = pairwise_distances(brand_features, brand_features[doc_id])
    color_dist = pairwise_distances(color_features, color_features[doc_id])
    image_dist = pairwise_distances(bottleneck_features_train, bottleneck_fea
    pairwise_dist   = (w1 * idf_w2v_dist +  w2 * brand_dist + w3 * color_dist

    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    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
                           data['medium_image_url'].loc[df_indices[i]],
                           indices[0], indices[i],df_indices[0], df_indices[i
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('Brand :',data['brand'].loc[df_indices[i]])
        print('euclidean distance from input :', pdists[i])
        print('='*125)

# in the give heat map, each cell contains the euclidean distance between wor
```

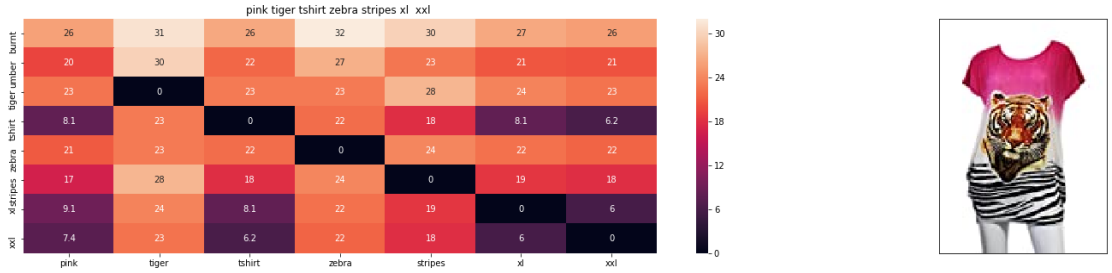
Recommendations 1

In [80]:

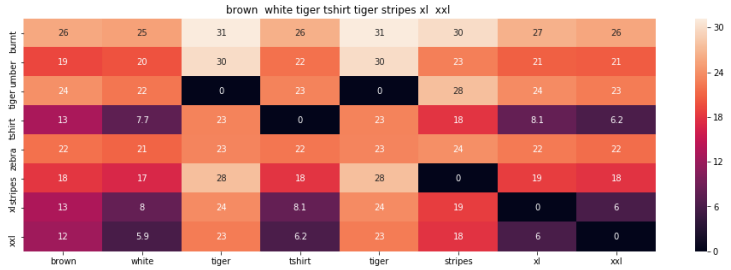
```
idf_w2v_brand_image(12566, 20, 15, 15, 2, 10)
```



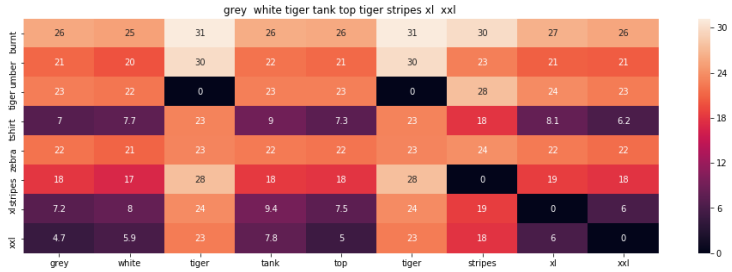
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 2.888168812485394e-07
==========
==========
=



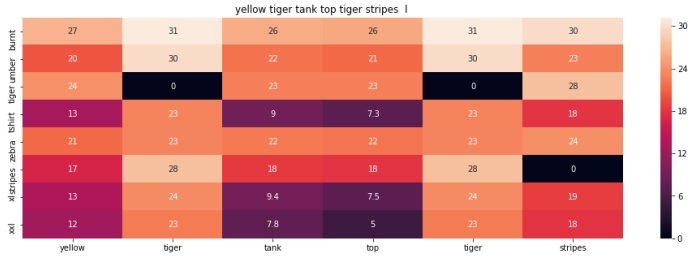
ASIN : B00JXQASS6
Brand : Si Row
euclidean distance from input : 3.8367415209125366
==========
==========
=



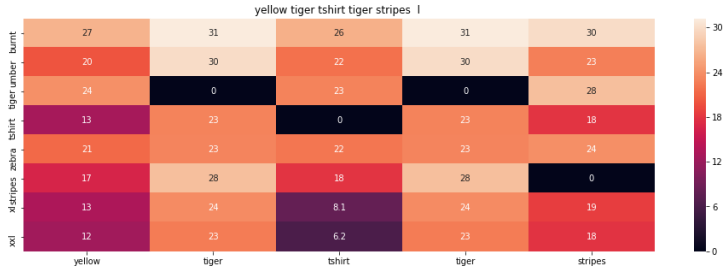
ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 3.9714330526498647
==========
==========
=



ASIN : B00JXQAFZ2
Brand : Si Row
euclidean distance from input : 4.5436468859027705
==========
==========
=



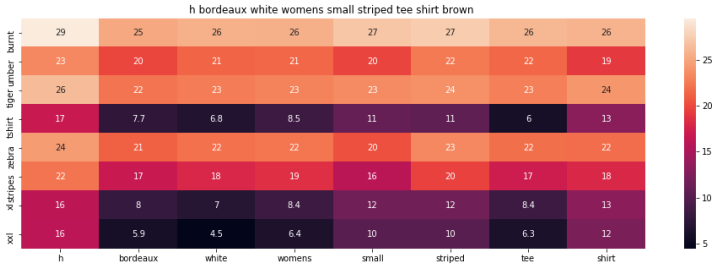
ASIN : B00JXQAUWA
Brand : Si Row
euclidean distance from input : 4.65310463548961
==========
==========
=



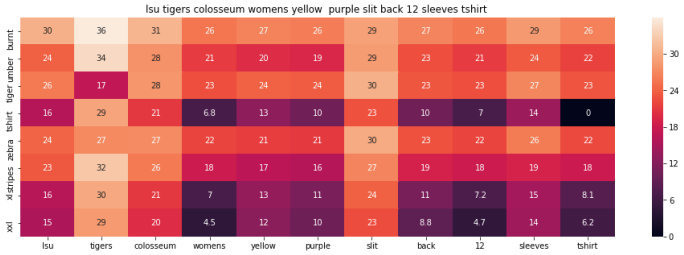
ASIN : B00JXQCUIC
Brand : Si Row
euclidean distance from input : 4.6814648555917735
==========
==========
=

black white tiger tank top tiger stripes l

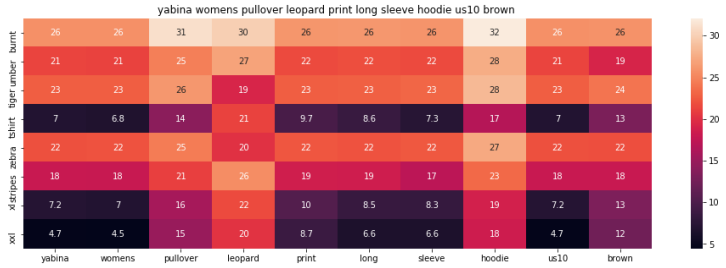
ASIN : B00JXQAO94
Brand : Si Row
euclidean distance from input : 4.721773881282879
==========
==========
=



ASIN : B072BVB47Z
Brand : H By Bordeaux
euclidean distance from input : 5.077845646784856
==========
==========
=



ASIN : B073R5Q8HD
Brand : Colosseum
euclidean distance from input : 5.2187249774134
==========
==========
=

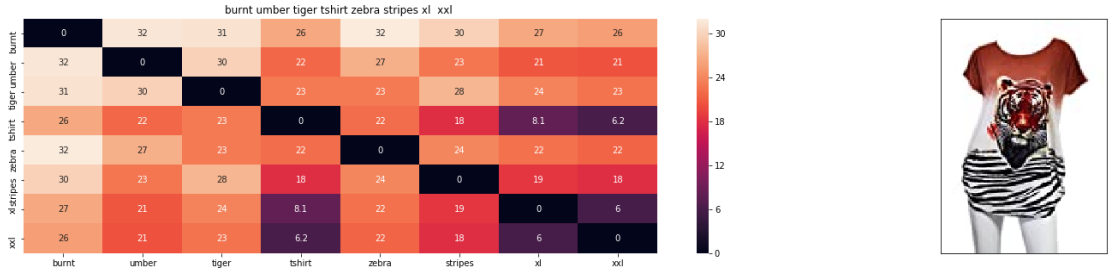


ASIN : B01KJUM6JI
Brand : YABINA
euclidean distance from input : 5.247309384235238
==========
==========
=

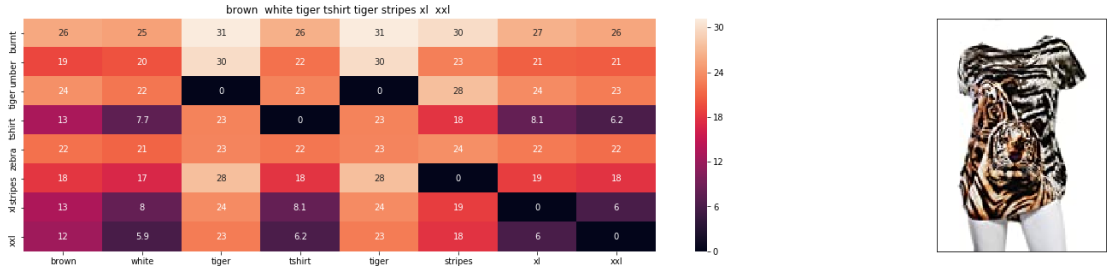
Recommendations 2

In [81]:

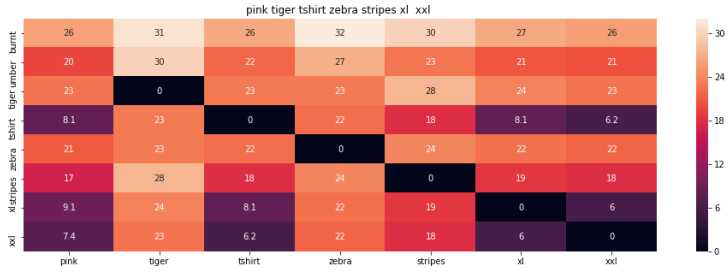
```
idf_w2v_brand_image(12566, 20, 2, 20, 2, 10)
```



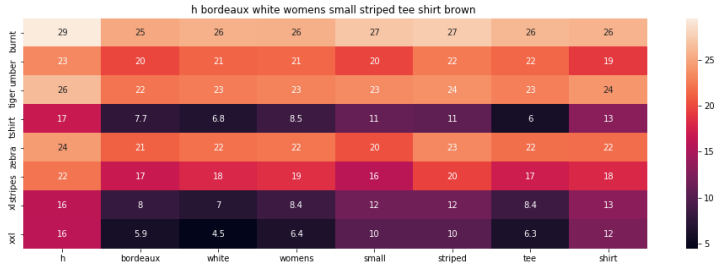
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 3.4132904147554655e-07
==========
==========
=



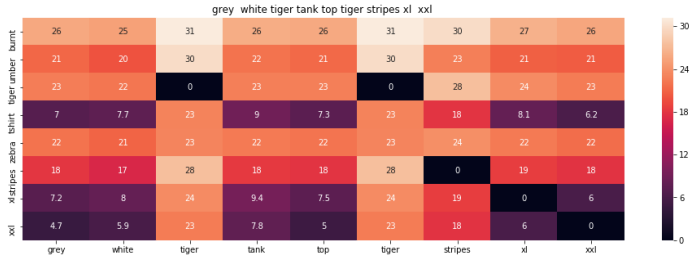
ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 4.693511789495295
==========
==========
=



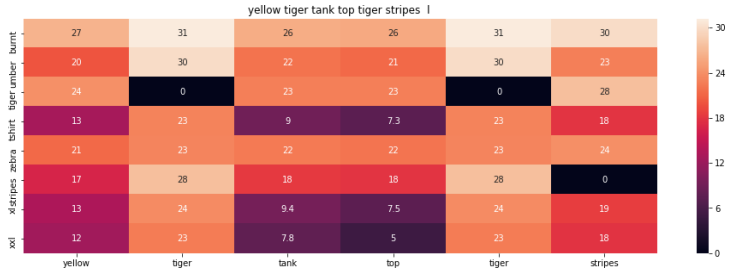
ASIN : B00JXQASS6
Brand : Si Row
euclidean distance from input : 4.695036974984486
==========
==========
=



ASIN : B072BVB47Z
Brand : H By Bordeaux
euclidean distance from input : 5.410181218927557
==========
==========
=



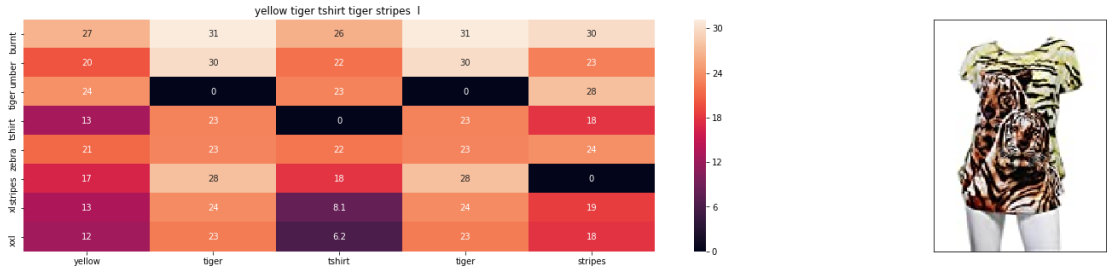
ASIN : B00JXQAFZ2
Brand : Si Row
euclidean distance from input : 5.5304705881547624
==========
==========
=



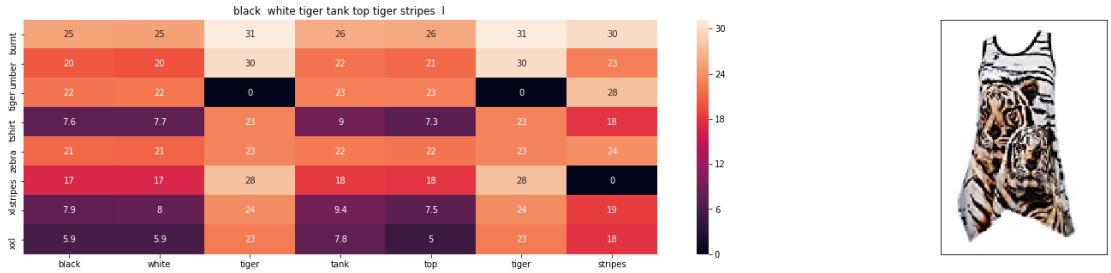
ASIN : B00JXQAUWA
Brand : Si Row
euclidean distance from input : 5.65982974675739
==========
==========
=

yabina womens pullover leopard print long sleeve hoodie us10 brown

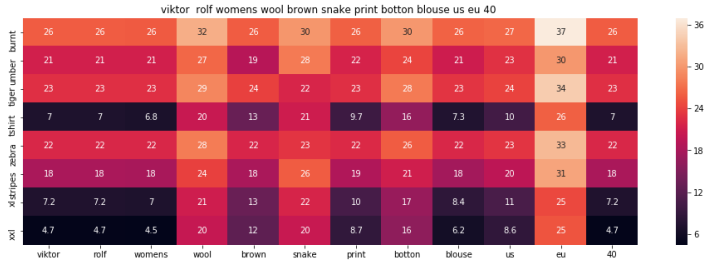
ASIN : B01KJUM6JI
Brand : YABINA
euclidean distance from input : 5.6896233518599315
==========
==========
=



ASIN : B00JXQCUIC
Brand : Si Row
euclidean distance from input : 5.693346370514493
==========
==========
=



ASIN : B00JXQAO94
Brand : Si Row
euclidean distance from input : 5.740984309967618
==========
==========
=

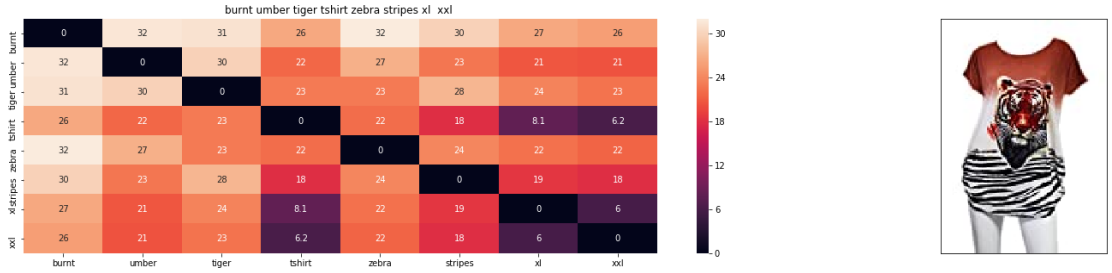


ASIN : B00LEHNVZ4
Brand : Viktor & Rolf
euclidean distance from input : 5.796076861294833
==========
==========
=

Recommendations only with respect to color

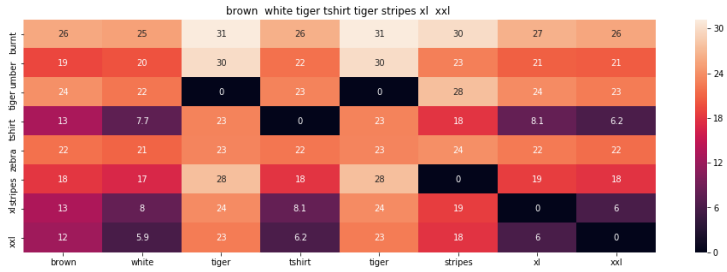
In [82]:

```
idf_w2v_brand_image(12566, 2, 2, 20, 1, 10)
```

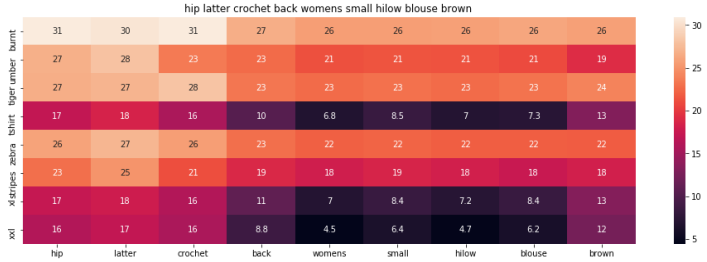


ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 3.0036955649848097e-07
==========
==========
=

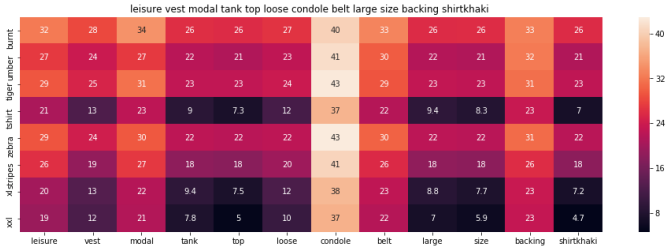
ASIN : B072BVB47Z
Brand : H By Bordeaux
euclidean distance from input : 2.5583835220336915
==========
==========
=



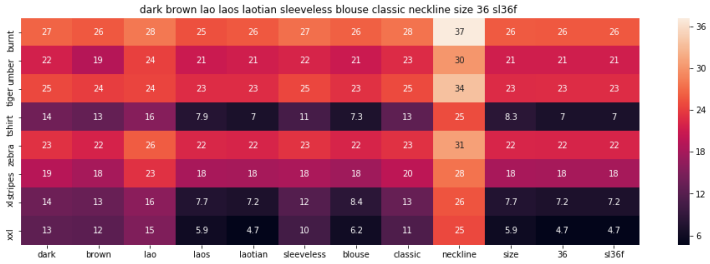
ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 2.603589210510254
==========
==========
=



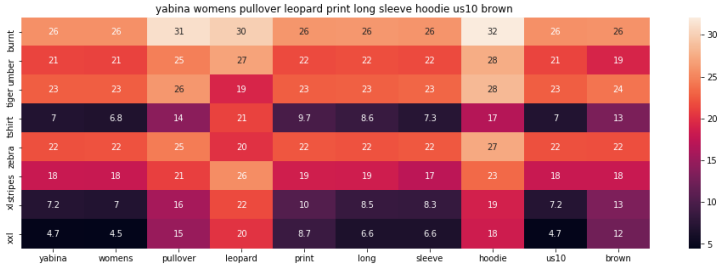
ASIN : B074MJN1K9
Brand : Hip
euclidean distance from input : 2.6279743887021896
==========
==========
=



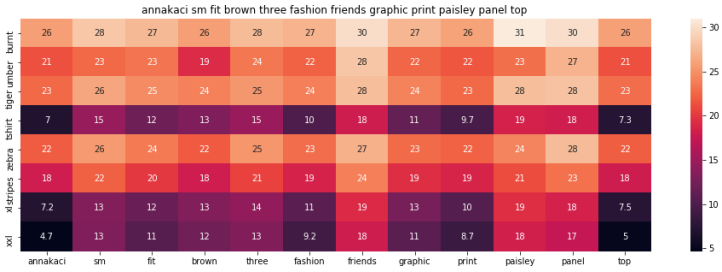
ASIN : B014OUHUZY
Brand : Black Temptation
euclidean distance from input : 2.7642454528808593
==========
==========
=



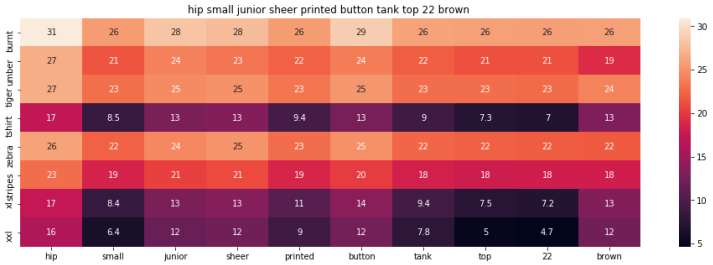
ASIN : B074J7BCYM
Brand : Nanon
euclidean distance from input : 2.7643200613096117
==========
==========
=



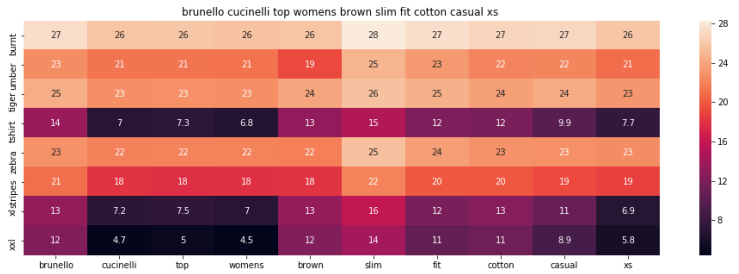
ASIN : B01KJUM6JI
Brand : YABINA
euclidean distance from input : 2.77741790066508
==========
==========
=======



ASIN : B00BTJKAQ0
Brand : Anna-Kaci
euclidean distance from input : 2.7843496322631838
==========
==========
=



ASIN : B071LDTQ1F
Brand : Hip
euclidean distance from input : 2.7984560323789474
==========
==========
=

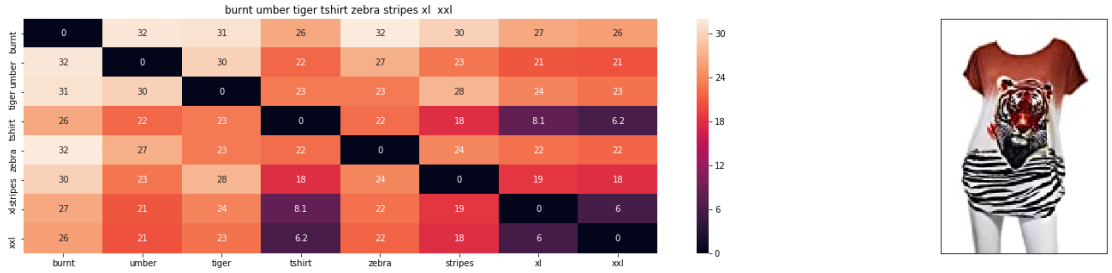


ASIN : B073ZCN5LG
Brand : Brunello Cucinelli
euclidean distance from input : 2.8087984848022463
==========
==========
=

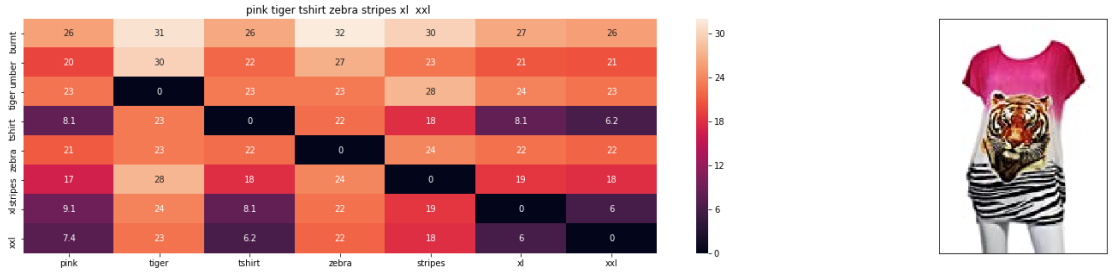
Recommendations only with respect to brand

In [83]:

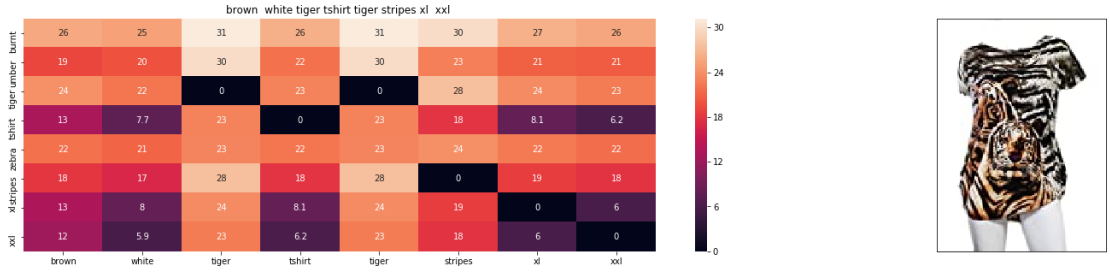
```
idf_w2v_brand_image(12566, 2, 20, 2, 1, 10)
```



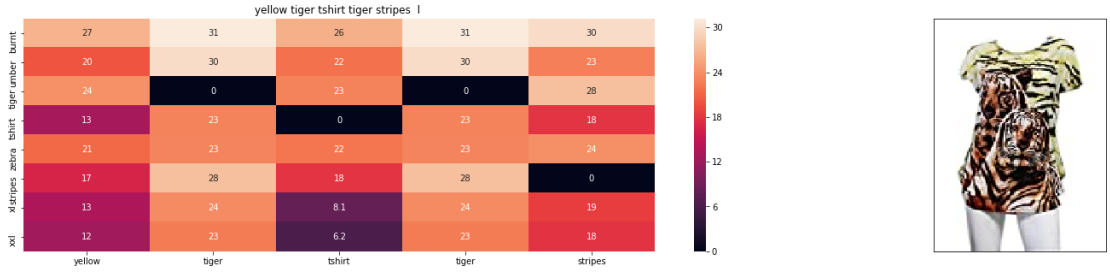
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 3.0036955649848097e-07
==========
==========
=



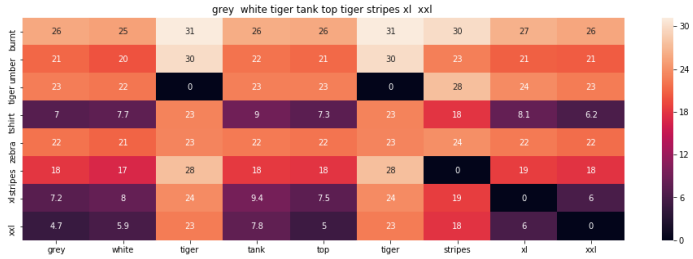
ASIN : B00JXQASS6
Brand : Si Row
euclidean distance from input : 2.3786404724410195
==========
==========
=



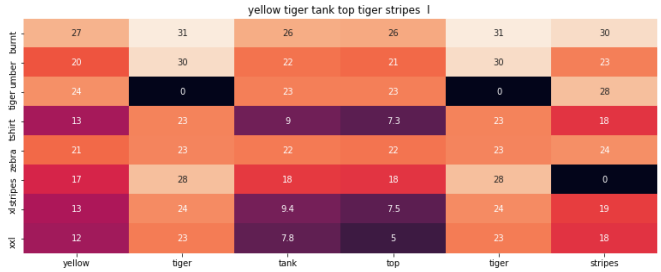
ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 2.603589210510254
==========
==========
=



ASIN : B00JXQCUIC
Brand : Si Row
euclidean distance from input : 2.6716949768355507
==========
==========
=

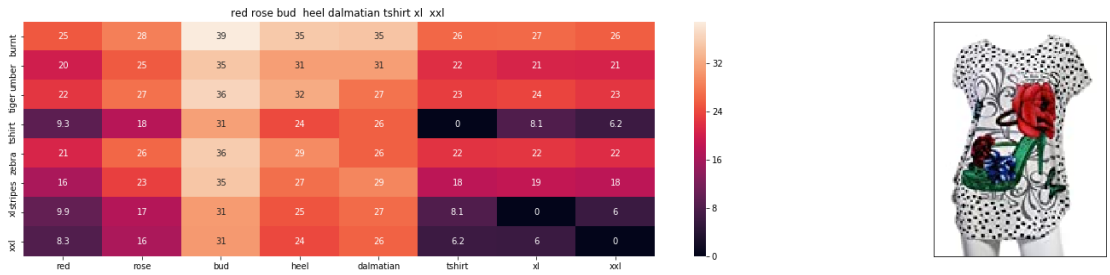


ASIN : B00JXQAFZ2
Brand : Si Row
euclidean distance from input : 2.6990144272139687
==========
==========
=

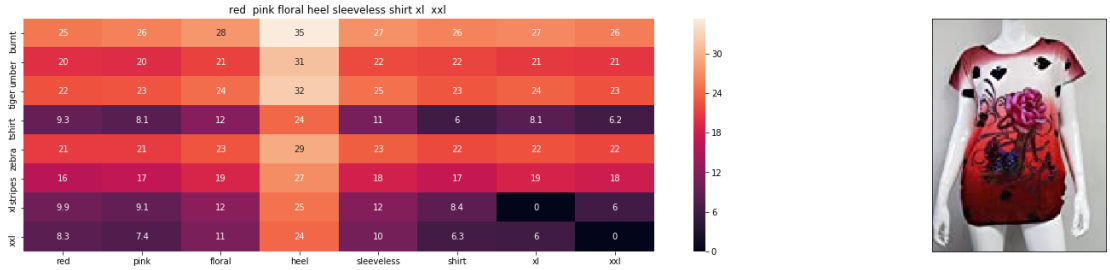


ASIN : B00JXQAUWA
Brand : Si Row
euclidean distance from input : 2.707454635649027
==========
==========
=

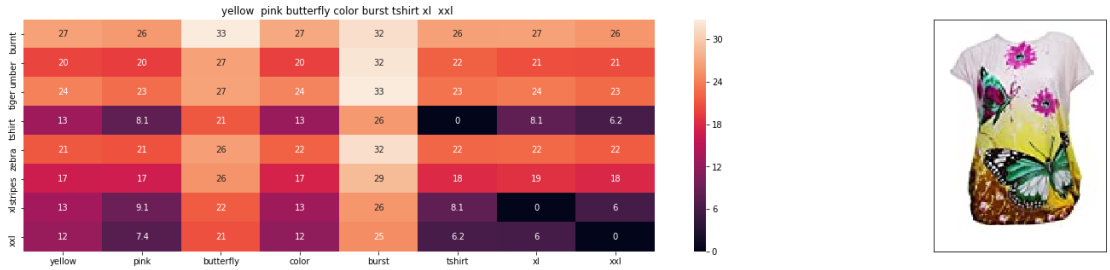
ASIN : B00JXQAO94
Brand : Si Row
euclidean distance from input : 2.777751266508402
==========
==========
=



ASIN : B00JXQABB0
Brand : Si Row
euclidean distance from input : 2.9405042572310585
==========
==========
=



ASIN : B00JV63QQE
Brand : Si Row
euclidean distance from input : 2.9439344330123083
==========

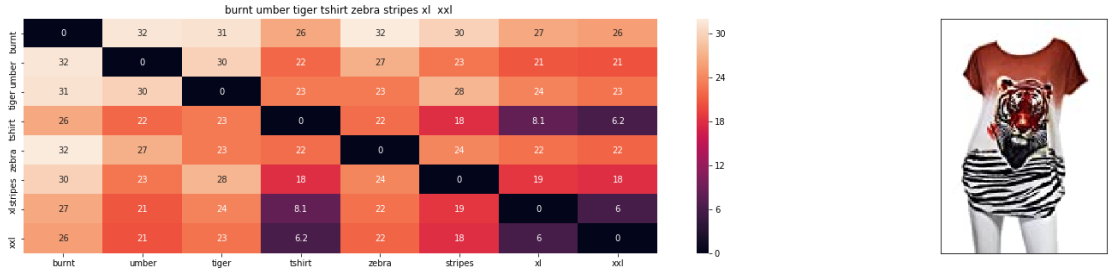


ASIN : B00JXQBBMI
Brand : Si Row
euclidean distance from input : 2.9488167877486364
==========
==========
=

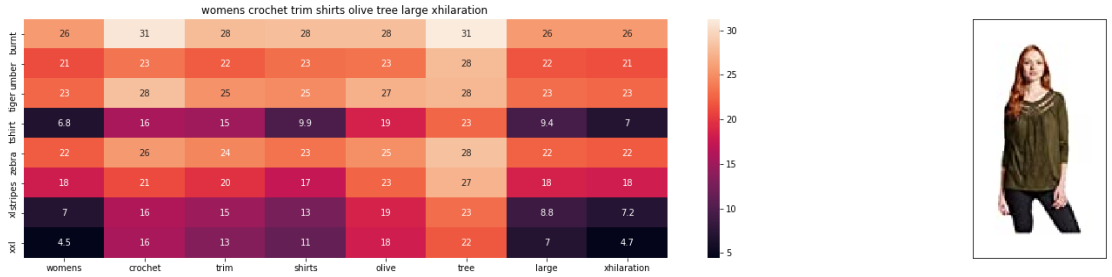
Recommendations only with respect to image

In [85]:

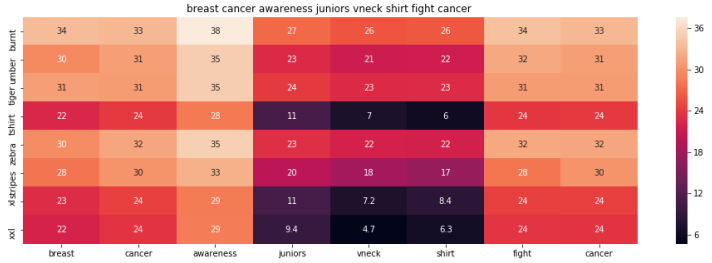
```
idf_w2v_brand_image(12566, 2, 2, 2, 50, 10)
```



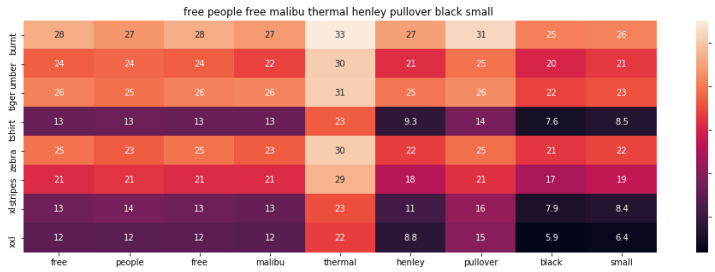
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 6.704677486725684e-06
==========
==========
=



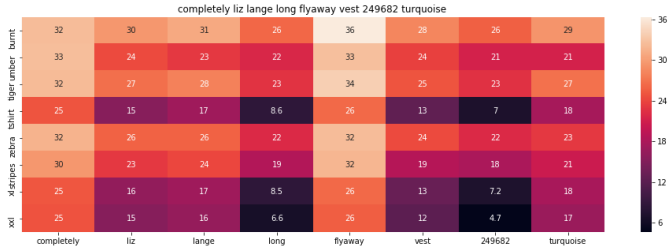
ASIN : B06XBHNM7J
Brand : Xhilaration
euclidean distance from input : 33.61035429801923
==========
==========
=



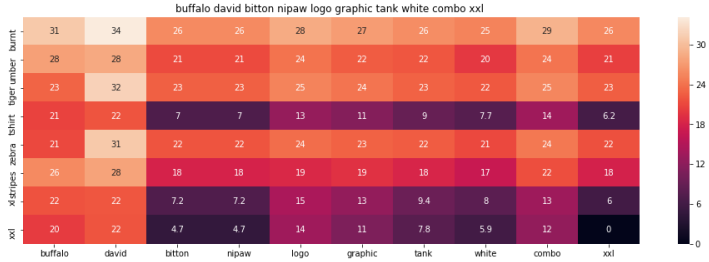
ASIN : B016CU40IY
Brand : Juiceclouds
euclidean distance from input : 34.75514814880835
==========
==========
=



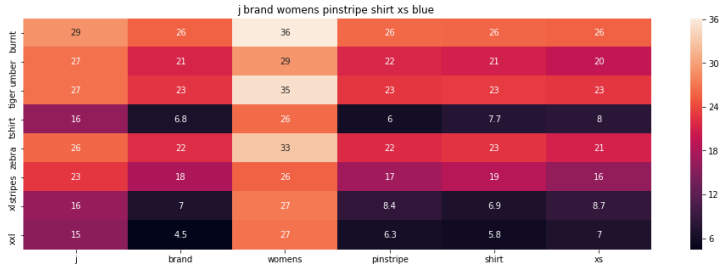
ASIN : B074MXY984
Brand : We The Free
euclidean distance from input : 35.57806262343113
==========
==========
=



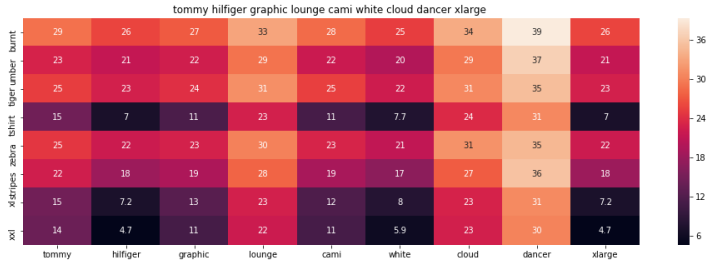
ASIN : B074LTBWSW
Brand : Liz Lange
euclidean distance from input : 35.57848621778417
==========
==========
=



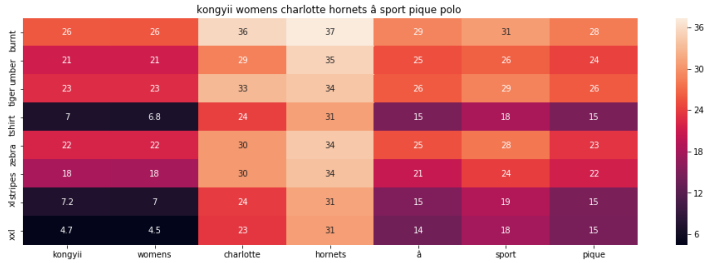
ASIN : B018H5AZXQ
Brand : Buffalo
euclidean distance from input : 35.712046599905655
==========
==========
=



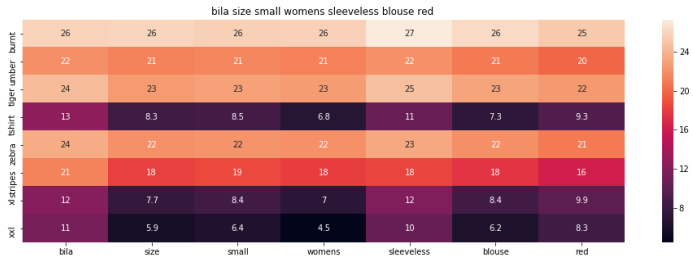
ASIN : B06XYP1X1F
Brand : J Brand Jeans
euclidean distance from input : 35.78138634136745
==========
==========
=======



ASIN : B01BMSFYW2
Brand : igertommy hilf
euclidean distance from input : 35.93889604296003
==========
==========
=



ASIN : B01FJVZST2
Brand : KONGYII
euclidean distance from input : 36.80775142764964
==========
==========
=

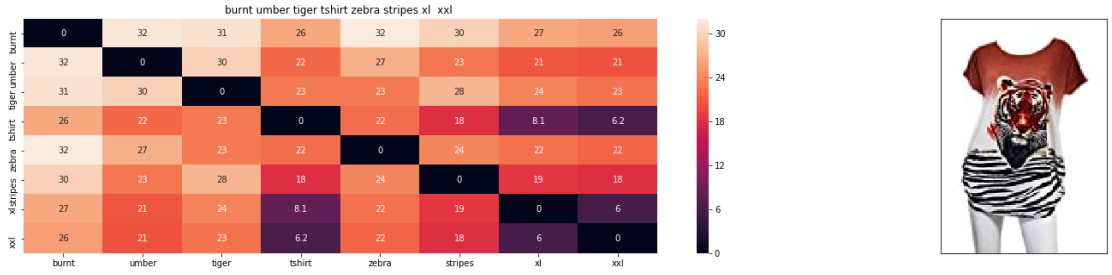


ASIN : B01L7ROZNC
Brand : Bila
euclidean distance from input : 37.023555868284234
==========
==========
=

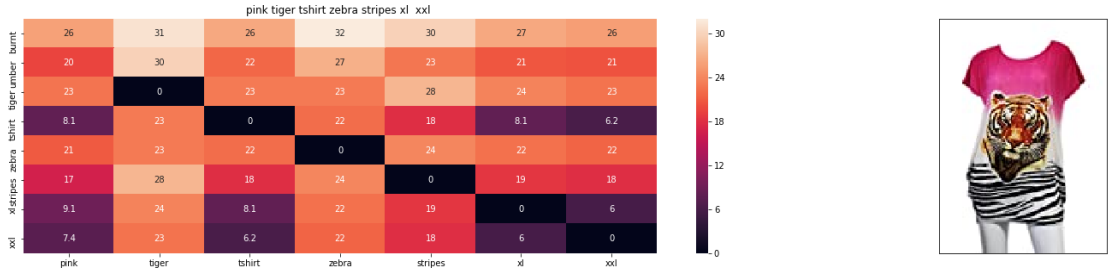
Recommendations only with respect to title

In [86]:

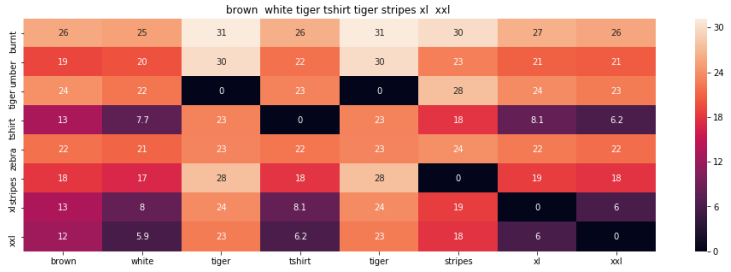
```
idf_w2v_brand_image(12566, 20, 2, 2, 1, 10)
```



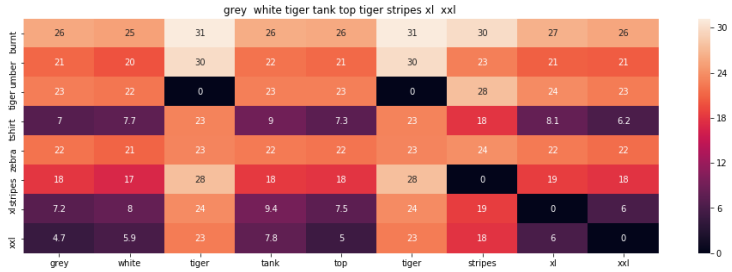
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 3.0036955649848097e-07
==========
==========
=



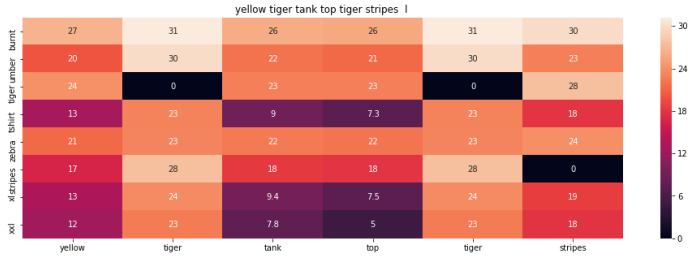
ASIN : B00JXQASS6
Brand : Si Row
euclidean distance from input : 5.304638855009379
==========
==========
=



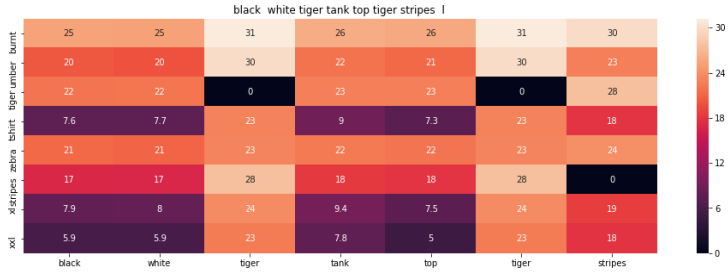
ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 6.0386668395996095
==========
==========
=



ASIN : B00JXQAFZ2
Brand : Si Row
euclidean distance from input : 6.558329956083598
==========
==========
=



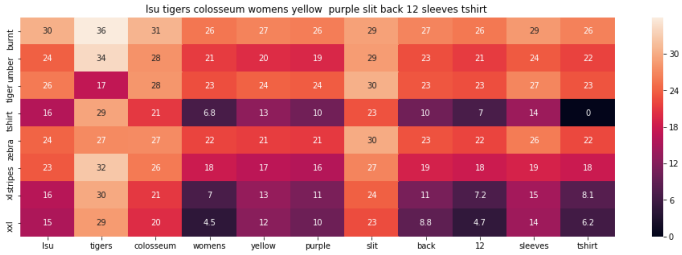
ASIN : B00JXQAUWA
Brand : Si Row
euclidean distance from input : 6.803910858183206
==========
==========
=



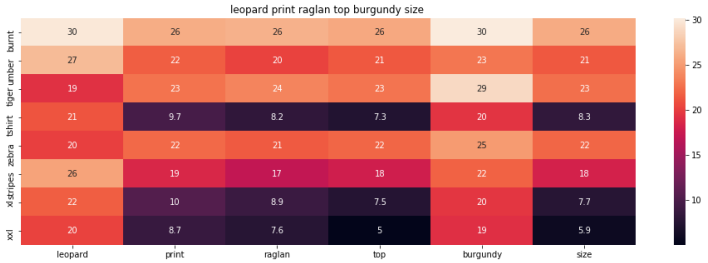
ASIN : B00JXQAO94
Brand : Si Row
euclidean distance from input : 6.876726104765238
==========
==========
=

yellow tiger tshirt tiger stripes I

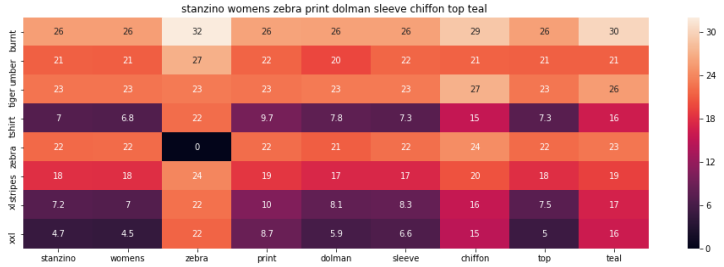
ASIN : B00JXQCUIC
Brand : Si Row
euclidean distance from input : 6.914973327665629
==========
==========
=



ASIN : B073R5Q8HD
Brand : Colosseum
euclidean distance from input : 7.237978005064108
==========
==========
=



ASIN : B01C6ORLDQ
Brand : 1 Mad Fit
euclidean distance from input : 7.60200732424766
==========
==========
=



ASIN : B00C0I3U3E
Brand : Stanzino
euclidean distance from input : 7.641074928892233
==========
==========
=