AAR

September 9, 2020

Amazon Apparel Recommendations

0.0.1 [4.2] Data and Code:

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

0.0.2 [4.3] Overview of the data

[1]:
```
#import all the necessary packages.

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

plotly.offline.init_notebook_mode(connected=True)
```

```
warnings.filterwarnings("ignore")
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```

[2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id
=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redire
ct_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&scope=email%20https%3a%2f%2fwww.googl
eapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%2
0https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2
fwww.googleapis.com%2fauth%2fpeopleapi.readonly&response_type=code

Enter your authorization code:
4/3wHuDMtGQvTBcC7ZyBKifmk5vhgnyeDuLddW4XsGWxiWdQCvZvdr_FM
Mounted at /content/drive
```

[3]:
```
import os
os.chdir("/content/drive/My Drive/AAR")
!ls -l
```

```
total 1632538
-r-------- 1 root root    3243081 Oct 21  2017 16k_apperal_data_preprocessed
-r-------- 1 root root     641760 Oct 23  2017 16k_data_cnn_feature_asins.npy
-r-------- 1 root root 1609846864 Oct 23  2017 16k_data_cnn_features.npy
-r-------- 1 root root   57985422 Oct 21  2017 word2vec_model
```

[4]:
```
data = pd.read_pickle('16k_apperal_data_preprocessed')
```

[5]:
```
# 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')
```

[6]:
```
print ('Number of data points : ', data.shape[0], \
       'Number of features/variables:', data.shape[1])
```

```
Number of data points :  16042 Number of features/variables: 7
```

0.0.3 Terminology:

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

[7]:
```
# each product/item has 19 features in the raw dataset.
data.columns # prints column-names or feature-names.
```

[7]:
```
Index(['asin', 'brand', 'color', 'medium_image_url', 'product_type_name',
       'title', 'formatted_price'],
      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)

[8]:
```
data = data[['asin', 'brand', 'color', 'medium_image_url', 'product_type_name',␣
 ↪'title', 'formatted_price']]
```

[9]:
```
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 :  16042 Number of features: 7
```

[9]:

	asin	...	formatted_price
4	B004GSI2OS	...	$26.26
6	B012YX2ZPI	...	$9.99
15	B003BSRPB0	...	$20.54
27	B014ICEJ1Q	...	$7.39
46	B01NACPBG2	...	$6.95

[5 rows x 7 columns]

0.0.4 [5.1] Missing data for various features.

Basic stats for the feature: product_type_name

[10]:
```
# 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     16042
unique       55
top       SHIRT
freq      12827
Name: product_type_name, dtype: object
```

[11]:
```
# names of different product types
print(data['product_type_name'].unique())
```

```
['SHIRT' 'APPAREL' 'HAT' 'SPORTING_GOODS' 'BOOKS_1973_AND_LATER' 'SWEATER'
 'ACCESSORY' 'UNDERWEAR' 'HANDBAG' 'OUTERWEAR'
 'OUTDOOR_RECREATION_PRODUCT' 'DRESS' 'ADULT_COSTUME' 'SLEEPWEAR'
 'HEALTH_PERSONAL_CARE' 'MISC_OTHER' 'BLAZER' 'SHOES' 'SHORTS' 'AUTO_PART'
 'HOME' 'OFFICE_PRODUCTS' 'PANTS' 'ETHNIC_WEAR' 'SKIRT' 'BEAUTY'
 'AUTO_ACCESSORY' 'INSTRUMENT_PARTS_AND_ACCESSORIES' 'TOOLS'
 'POWERSPORTS_RIDING_SHIRT' 'EYEWEAR' 'BRA' 'POWERSPORTS_RIDING_JACKET'
 'HARDWARE' 'VIDEO_DVD' 'GOLF_CLUB' 'MUSIC_POPULAR_VINYL'
 'GUILD_ACCESSORIES' 'SUIT' 'SAFETY_SUPPLY' 'TOYS_AND_GAMES'
 'POWERSPORTS_PROTECTIVE_GEAR' 'BAG' 'MECHANICAL_COMPONENTS'
 'BABY_PRODUCT' 'SOUND_AND_RECORDING_EQUIPMENT' 'JEWELRY' 'ABIS_APPAREL'
 'OUTDOOR_LIVING' 'BUILDING_MATERIAL' 'LUGGAGE' 'BABY_COSTUME'
 'POWERSPORTS_VEHICLE_PART' 'SHIRTS' 'PET_SUPPLIES']
```

[12]:
```
# find the 10 most frequent product_type_names.
product_type_count = Counter(list(data['product_type_name']))
product_type_count.most_common(10)
```

[12]:
```
[('SHIRT', 12827),
 ('BOOKS_1973_AND_LATER', 1334),
 ('APPAREL', 620),
 ('ACCESSORY', 302),
 ('SPORTING_GOODS', 194),
 ('DRESS', 141),
 ('BLAZER', 93),
 ('SWEATER', 79),
 ('OUTERWEAR', 69),
 ('OUTDOOR_RECREATION_PRODUCT', 53)]
```

Basic stats for the feature: brand

[13]:
```
# there are 10577 unique brands
print(data['brand'].describe())

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

```
count         15997
unique         3543
top       Anna-Kaci
freq            117
Name: brand, dtype: object
```

[14]:
```
brand_count = Counter(list(data['brand']))
brand_count.most_common(10)
```

[14]:
```
[('Anna-Kaci', 117),
 ('Mogul Interior', 80),
 ('Soprano', 78),
 ('BODEN', 75),
```

```
 ('bankhunyabangyai store', 73),
 ('Nanon', 73),
 ('PERI', 73),
 ('DSQUARED2', 73),
 ("H'nan", 72),
 ('No Boundaries', 70)]
```

Basic stats for the feature: color

[ ]:
```
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
```

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

[ ]:
```
[(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

[ ]:
```

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
```

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

[ ]:
```
[(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

[ ]:
```
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
```

[ ]:
```
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.

[ ]:
```
# 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
```

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

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

We brought down the number of data points from 183K to 28K. We are processing only 28K points so that most of the workshop participants can run this code on thier laptops in a reasonable amount of time.

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

[ ]:
```
data.to_pickle('pickels/28k_apparel_data')
```

[ ]:
```
# 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')


'''
```

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

0.0.5 [5.2] Remove near duplicate items

[5.2.1] Understand about duplicates.

[ ]:
```
# 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
```

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

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

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

[ ]:
```
          asin                        brand              color  \
4   B004GSI2OS                   FeatherLite  Onyx Black/ Stone
6   B012YX2ZPI  HX-Kingdom Fashion T-shirts              White
11  B001LOUGE4                 Fitness Etc.              Black
15  B003BSRPB0                   FeatherLite              White
21  B014ICEDNA                         FNC7C             Purple

                                    medium_image_url product_type_name  \
4   https://images-na.ssl-images-amazon.com/images...             SHIRT
6   https://images-na.ssl-images-amazon.com/images...             SHIRT
11  https://images-na.ssl-images-amazon.com/images...             SHIRT
15  https://images-na.ssl-images-amazon.com/images...             SHIRT
21  https://images-na.ssl-images-amazon.com/images...             SHIRT

                                               title formatted_price
4   Featherlite Ladies' Long Sleeve Stain Resistan...          $26.26
6   Women's Unique 100% Cotton T - Special Olympic...           $9.99
11            Ladies Cotton Tank 2x1 Ribbed Tank Top          $11.99
15  FeatherLite Ladies' Moisture Free Mesh Sport S...          $20.54
21  Supernatural Chibis Sam Dean And Castiel Short...           $7.50
```

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

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

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

[ ]:
```
             asin     brand        color  \
61973   B06Y1KZ2WB    Éclair   Black/Pink
133820  B010RV33VE  xiaoming         Pink
81461   B01DDSDLNS  xiaoming        White
75995   B00X5LYO9Y  xiaoming  Red Anchors
```

```
151570  B00WPJG35K  xiaoming        White

                                        medium_image_url product_type_name  \
61973   https://images-na.ssl-images-amazon.com/images...             SHIRT
133820  https://images-na.ssl-images-amazon.com/images...             SHIRT
81461   https://images-na.ssl-images-amazon.com/images...             SHIRT
75995   https://images-na.ssl-images-amazon.com/images...             SHIRT
151570  https://images-na.ssl-images-amazon.com/images...             SHIRT

                                                    title formatted_price
61973   Éclair Women's Printed Thin Strap Blouse Black...          $24.99
133820  xiaoming Womens Sleeveless Loose Long T-shirts...          $18.19
81461   xiaoming Women's White Long Sleeve Single Brea...          $21.58
75995   xiaoming Stripes Tank Patch/Bear Sleeve Anchor...          $15.91
151570  xiaoming Sleeve Sheer Loose Tassel Kimono Woma...          $14.32
```

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

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

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

    previous_i = i

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

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

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

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

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

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

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


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

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

We removed the dupliactes which differ only at the end.

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

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

[ ]:
```
data.to_pickle('pickels/17k_apperal_data')
```

[5.2.3] Remove duplicates : Part 2

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

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

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

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

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

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

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

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

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

[ ]:
```
# 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)]
```

[ ]:
```
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
```

[ ]:
```
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.
```

1 6. Text pre-processing

[ ]:
```
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()
```

[ ]:
```
# 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', 'most', 'few', 'should', 're', 'very',
'just', 'then', 'didn', 'myself', 'in', 'too', 's', 'shouldn', 'herself',
'because', 'how', 'itself', 'what', 'shan', 'weren', 'doing', 'them', 'couldn',
'their', 'so', 'ain', 'haven', 'yourself', 'now', 'll', 'isn', 'about', 'over',
'into', 'before', 'during', 'on', 'as', 'aren', 'against', 'above', 'down',
'they', 'below', 'me', 'again', 'for', 'why', 'been', 'yourselves', 'more',
'her', 'that', 'can', 'am', 'was', 'themselves', 'mightn', 'does', 'those',
```

```
'only', 'hasn', 'any', 'ma', 'are', 'nor', 'out', 'you', 'ourselves', 'the',
'an', 'has', 'where', 'i', 'while', 'ours', 'its', 'your', 'had', 'were',
'being', 'no', 'or', 'needn', 've', 'y', 'a', 'each', 'have', 'through', 'when',
'mustn', 'by', 'won', 'from', 'own', 'will', 'there', 't', 'him', 'these',
'doesn', 'theirs', 'my', 'did', 'of', 'who', 'until', 'wouldn', 'we', 'do',
'having', 'yours', 'other', 'wasn', 'it', 'with', 'once', 'here', 'don', 'o',
'whom', 'this', 'if', 'but', 'hadn', 'our', 'some', 'm', 'not', 'between',
'himself', 'same', 'at', 'be', 'he', 'after', 'which', 'to', 'his'}
```

[ ]:
```
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
```

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

[ ]:

	asin	brand	color	\
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	
6	B012YX2ZPI	HX-Kingdom Fashion T-shirts	White	
15	B003BSRPB0	FeatherLite	White	
27	B014ICEJ1Q	FNC7C	Purple	
46	B01NACPBG2	Fifth Degree	Black	

	medium_image_url	product_type_name	\
4	https://images-na.ssl-images-amazon.com/images...	SHIRT	
6	https://images-na.ssl-images-amazon.com/images...	SHIRT	
15	https://images-na.ssl-images-amazon.com/images...	SHIRT	
27	https://images-na.ssl-images-amazon.com/images...	SHIRT	
46	https://images-na.ssl-images-amazon.com/images...	SHIRT	

	title	formatted_price
4	featherlite ladies long sleeve stain resistant...	$26.26
6	womens unique 100 cotton special olympics wor...	$9.99
15	featherlite ladies moisture free mesh sport sh...	$20.54
27	supernatural chibis sam dean castiel neck tshi...	$7.39
46	fifth degree womens gold foil graphic tees jun...	$6.95

[ ]:
```
data.to_pickle('pickels/16k_apperal_data_preprocessed')
```

1.1 Stemming

[ ]:
```
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
```

2 [8] Text based product similarity

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

[ ]:

	asin	brand	color	\
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	
6	B012YX2ZPI	HX-Kingdom Fashion T-shirts	White	
15	B003BSRPB0	FeatherLite	White	
27	B014ICEJ1Q	FNC7C	Purple	
46	B01NACPBG2	Fifth Degree	Black	

	medium_image_url	product_type_name	\
4	https://images-na.ssl-images-amazon.com/images...	SHIRT	
6	https://images-na.ssl-images-amazon.com/images...	SHIRT	
15	https://images-na.ssl-images-amazon.com/images...	SHIRT	
27	https://images-na.ssl-images-amazon.com/images...	SHIRT	
46	https://images-na.ssl-images-amazon.com/images...	SHIRT	

	title	formatted_price
4	featherlite ladies long sleeve stain resistant...	$26.26
6	womens unique 100 cotton special olympics wor...	$9.99
15	featherlite ladies moisture free mesh sport sh...	$20.54
27	supernatural chibis sam dean castiel neck tshi...	$7.39
46	fifth degree womens gold foil graphic tees jun...	$6.95

[38]:
```
# Utility Functions which we will use through the rest of the workshop.


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

```
    # we will display it in notebook
    plt.imshow(img)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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


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



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

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

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

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

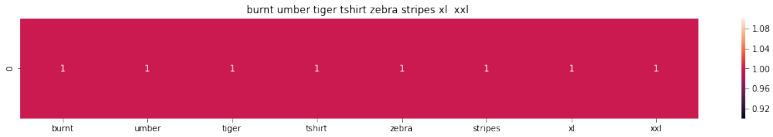
2.1 [8.2] Bag of Words (BoW) on product titles.

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

```
# title_features.shape = #data_points * #words_in_corpus
# CountVectorizer().fit_transform(corpus) returns
# the a sparase matrix of dimensions #data_points * #words_in_corpus

# What is a sparse vector?

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

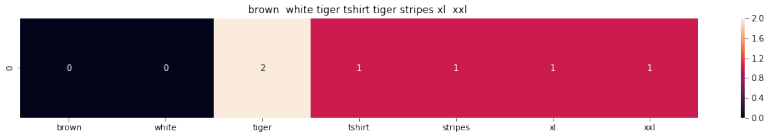
[ ]:
```
(16042, 12609)
```

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

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

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

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

    for i in range(0,len(indices)):
        # we will pass 1. doc_id, 2. title1, 3. title2, url, model
        get_result(indices[i],data['title'].loc[df_indices[0]], data['title'].
 ↪loc[df_indices[i]], data['medium_image_url'].loc[df_indices[i]],␣
 ↪'bag_of_words')
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print ('Brand:', data['brand'].loc[df_indices[i]])
        print ('Title:', data['title'].loc[df_indices[i]])
        print ('Euclidean similarity with the query image :', pdists[i])
        print('='*60)

#call the bag-of-words model for a product to get similar products.
bag_of_words_model(12566, 20) # change the index if you want to.
# In the output heat map each value represents the count value
# of the label word, the color represents the intersection
# with inputs title.

#try 12566
#try 931
```

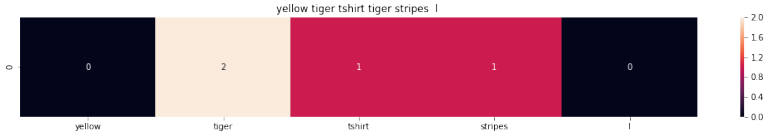


ASIN : B00JXQB5FQ
Brand: Si Row
Title: burnt umber tiger tshirt zebra stripes xl xxl
Euclidean similarity with the query image : 0.0
==========

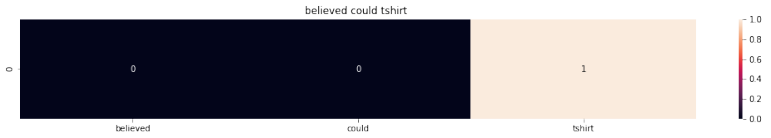
ASIN : B00JXQASS6
Brand: Si Row
Title: pink tiger tshirt zebra stripes xl xxl
Euclidean similarity with the query image : 1.73205080757
==========



ASIN : B00JXQCWTO
Brand: Si Row
Title: brown white tiger tshirt tiger stripes xl xxl
Euclidean similarity with the query image : 2.44948974278
==========

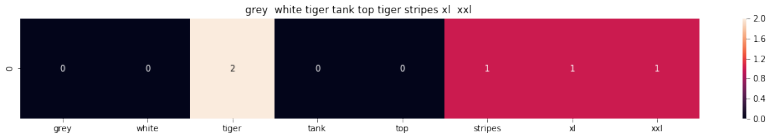


ASIN : B00JXQCUIC
Brand: Si Row
Title: yellow tiger tshirt tiger stripes l
Euclidean similarity with the query image : 2.64575131106
==

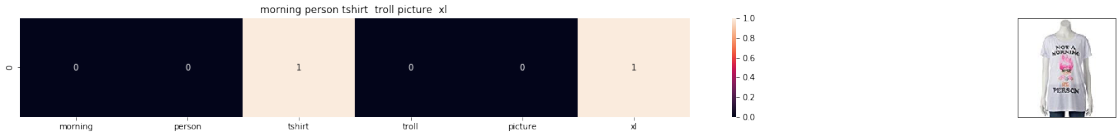


ASIN : B07568NZX4
Brand: Rustic Grace
Title: believed could tshirt
Euclidean similarity with the query image : 3.0
==

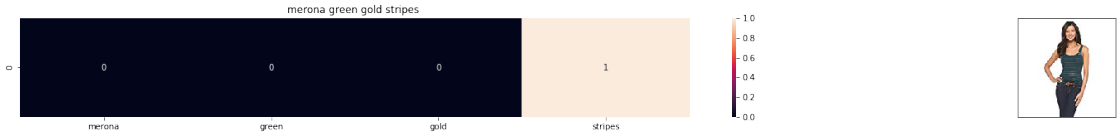
ASIN : B01NBONKRO
Brand: Ideology
Title: ideology graphic tshirt xl white
Euclidean similarity with the query image : 3.0
==



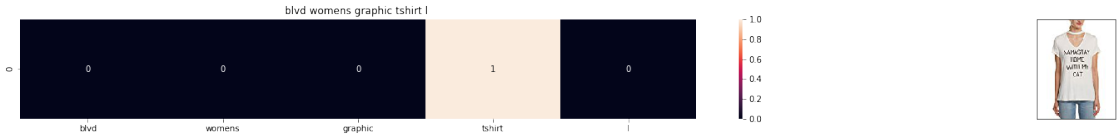
ASIN : B00JXQAFZ2
Brand: Si Row
Title: grey white tiger tank top tiger stripes xl xxl
Euclidean similarity with the query image : 3.0
==



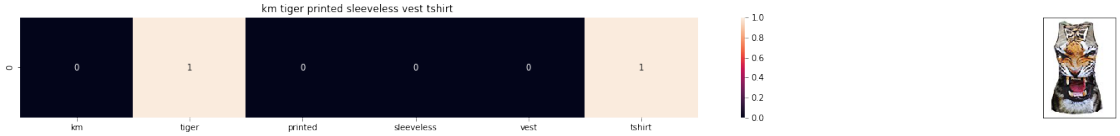
ASIN : B01CLS8LMW
Brand: Awake
Title: morning person tshirt troll picture xl
Euclidean similarity with the query image : 3.16227766017
==

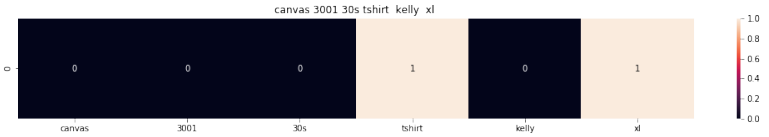


ASIN : B01KVZUB6G
Brand: Merona
Title: merona green gold stripes
Euclidean similarity with the query image : 3.16227766017
==

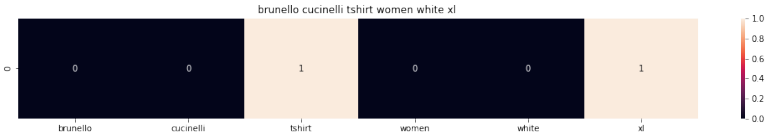


ASIN : B0733R2CJK
Brand: BLVD
Title: blvd womens graphic tshirt l
Euclidean similarity with the query image : 3.16227766017
==



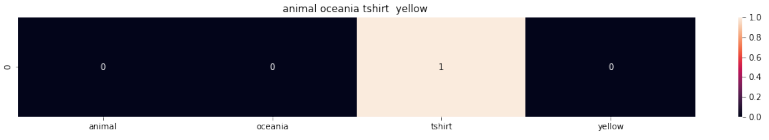


ASIN : B0088PN0LA
Brand: Red House
Title: canvas 3001 30s tshirt kelly xl
Euclidean similarity with the query image : 3.16227766017
==========

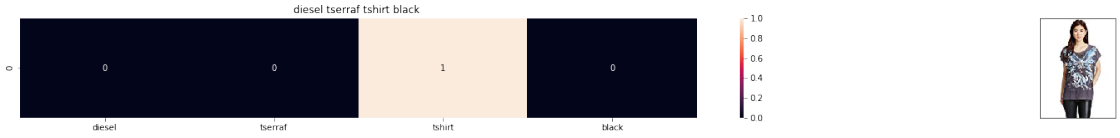


ASIN : B06X99V6WC
Brand: Brunello Cucinelli
Title: brunello cucinelli tshirt women white xl
Euclidean similarity with the query image : 3.16227766017
==========

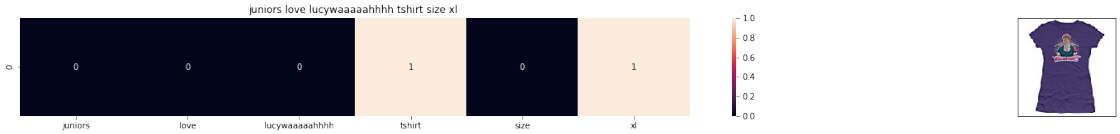
ASIN : B06Y1JPW1Q
Brand: Xhilaration
Title: xhilaration womens lace tshirt salmon xxl
Euclidean similarity with the query image : 3.16227766017
==========



```
ASIN : B06X6GX6WG
Brand: Animal
Title: animal oceania tshirt  yellow
Euclidean similarity with the query image : 3.16227766017
============================================================
```




```
ASIN : B017X8PW9U
Brand: Diesel
Title: diesel tserraf tshirt black
Euclidean similarity with the query image : 3.16227766017
============================================================
```



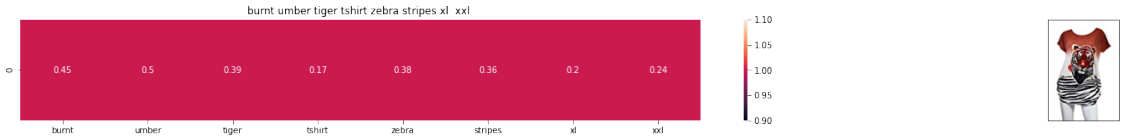

```
ASIN : B00IAA4JIQ
Brand: I Love Lucy
Title: juniors love lucywaaaaahhhh tshirt size xl
Euclidean similarity with the query image : 3.16227766017
============================================================
```

2.2 [8.5] TF-IDF based product similarity

[41]:
```
tfidf_title_vectorizer = TfidfVectorizer(min_df = 0)
tfidf_title_features = tfidf_title_vectorizer.fit_transform(data['title'])
# tfidf_title_features.shape = #data_points * #words_in_corpus
# CountVectorizer().fit_transform(courpus) returns the a sparase matrix of␣
 ↪dimensions #data_points * #words_in_corpus
# tfidf_title_features[doc_id, index_of_word_in_corpus] = tfidf values of the␣
 ↪word in given doc
```

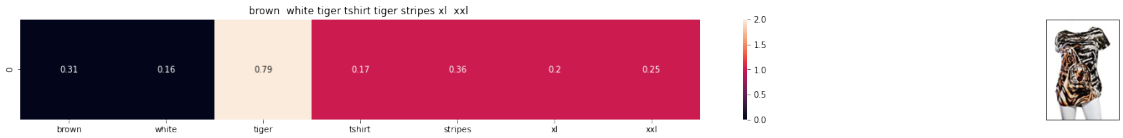
[42]:
```
def tfidf_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

    # pairwise_dist will store the distance from given input apparel to all␣
 ↪remaining apparels
```

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

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

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

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

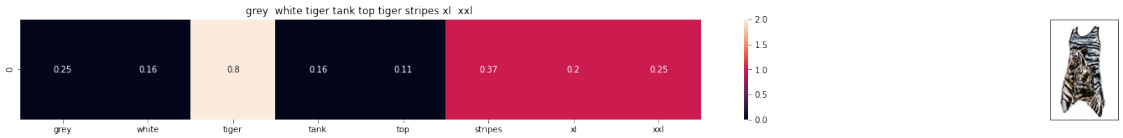



```
ASIN : B00JXQB5FQ
BRAND : Si Row
Eucliden distance from the given image : 0.0
=============================================================================
=============================================
```

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

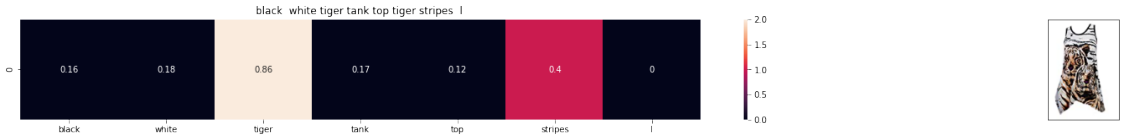


ASIN : B00JXQCWTO
BRAND : Si Row
Eucliden distance from the given image : 0.9357643943769647
==========
==========

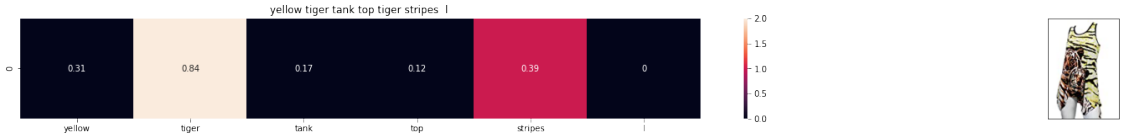


ASIN : B00JXQAFZ2
BRAND : Si Row
Eucliden distance from the given image : 0.9586153524200749
==========
==========

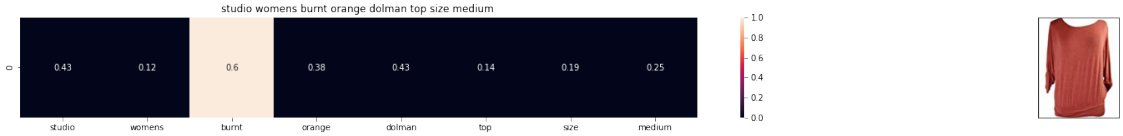
ASIN : B00JXQCUIC
BRAND : Si Row
Eucliden distance from the given image : 1.000074961446881
==========
==========



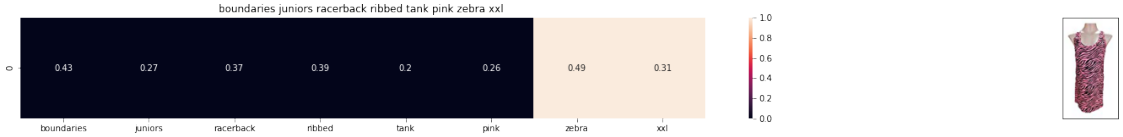
ASIN : B00JXQAO94
BRAND : Si Row
Eucliden distance from the given image : 1.023215552457452
==========
==========



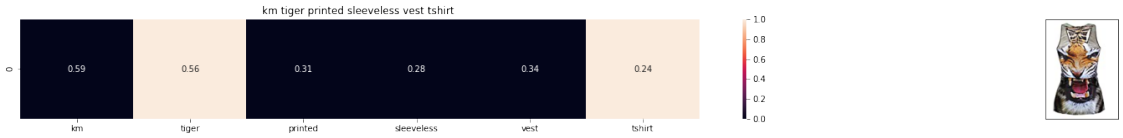
ASIN : B00JXQAUWA
BRAND : Si Row
Eucliden distance from the given image : 1.031991846303421
==========
==========



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

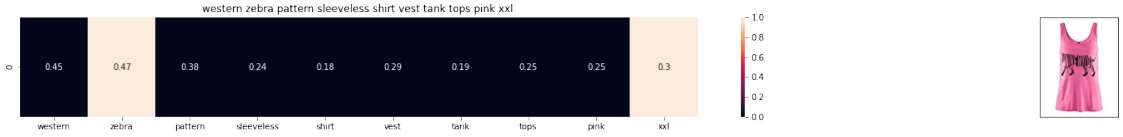


ASIN : B06Y2GTYPM
BRAND : No Boundaries
Eucliden distance from the given image : 1.2121683810720831
==
==



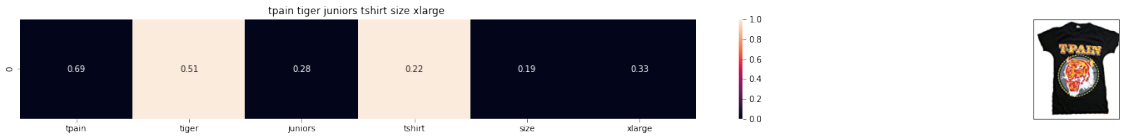
ASIN : B012VQLT6Y
BRAND : KM T-shirt
Eucliden distance from the given image : 1.219790640280982
==
==

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

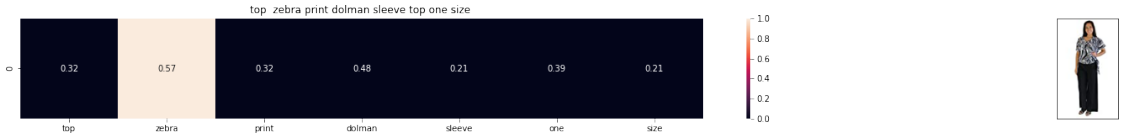


ASIN : B00Z6HEXWI
BRAND : Black Temptation
Eucliden distance from the given image : 1.221281392120943
==
==

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

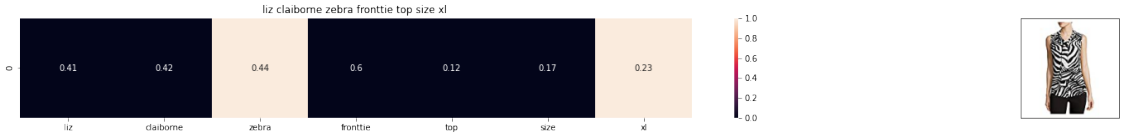


ASIN : B01K0H02OG
BRAND : Tultex
Eucliden distance from the given image : 1.236457298812782
==========
==========



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

2.3 [8.5] IDF based product similarity

[47]:
```
idf_title_vectorizer = CountVectorizer()
idf_title_features = idf_title_vectorizer.fit_transform(data['title'])

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

[62]:
```
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)))
```

[63]:
```
# we need to convert the values into float
idf_title_features  = idf_title_features.astype(np.float)

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

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

        # we replace the count values of word i in document j with  idf_value␣
 ↪of word i
```

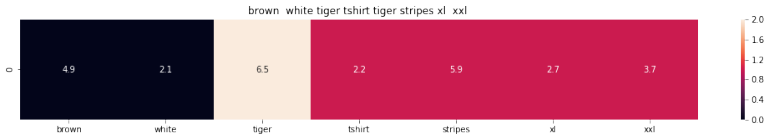
```
        # idf_title_features[doc_id, index_of_word_in_courpus] = idf value of␣
 ↪word
        idf_title_features[j,idf_title_vectorizer.vocabulary_[i]] = idf_val
```

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

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

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

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

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



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

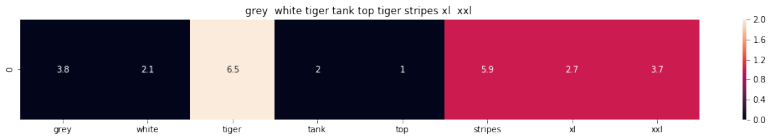

```
ASIN : B00JXQB5FQ
Brand : Si Row
```

euclidean distance from the given image : 0.0

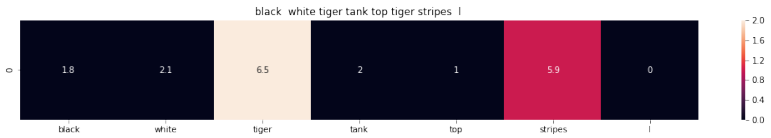
ASIN : B00JXQASS6
Brand : Si Row
euclidean distance from the given image : 12.2050713112



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

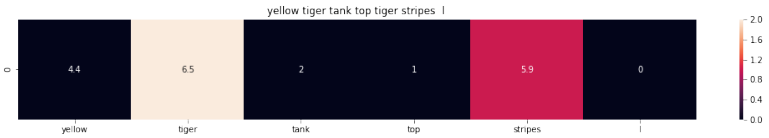


ASIN : B00JXQAFZ2
Brand : Si Row
euclidean distance from the given image : 14.4868329248

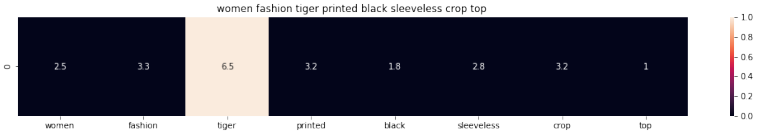


ASIN : B00JXQAO94
Brand : Si Row
euclidean distance from the given image : 14.8333929667
==

ASIN : B00JXQCUIC
Brand : Si Row
euclidean distance from the given image : 14.8987445167
==



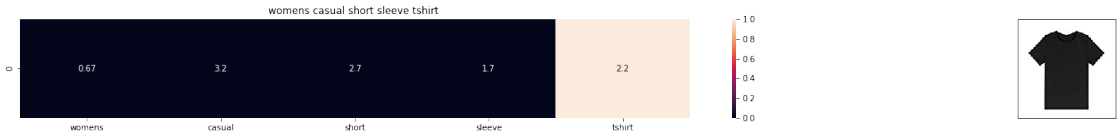
ASIN : B00JXQAUWA
Brand : Si Row
euclidean distance from the given image : 15.2244582873
==



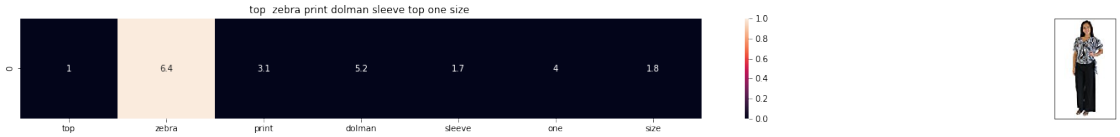
ASIN : B074T8ZYGX
Brand : MKP Crop Top
euclidean distance from the given image : 17.0808129556
==

ASIN : B00KF2N5PU
Brand : Vietsbay
euclidean distance from the given image : 17.0901681256
==========
==========

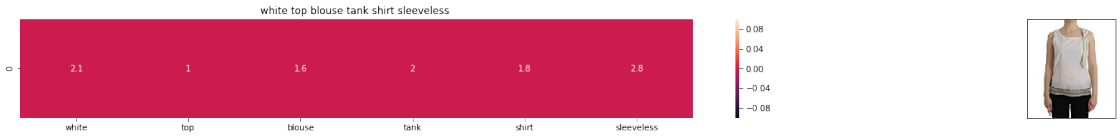
ASIN : B00JPOZ9GM
Brand : Sofra
euclidean distance from the given image : 17.1532153376
==========
==========



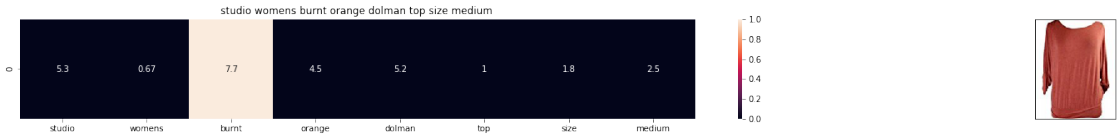
ASIN : B074T9KG9Q
Brand : Rain
euclidean distance from the given image : 17.3367152387
==========
==========



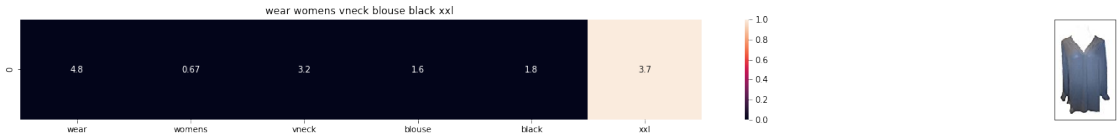
ASIN : B00H8A6ZLI
Brand : Vivian's Fashions
euclidean distance from the given image : 17.410075941
==========
==========



ASIN : B074G5G5RK
Brand : ERMANNO SCERVINO
euclidean distance from the given image : 17.5399213355
==========
==========



ASIN : B06XSCVFT5
Brand : Studio M
euclidean distance from the given image : 17.6127585437
==========
==========



ASIN : B06Y6FH453
Brand : Who What Wear
euclidean distance from the given image : 17.6237452825
==========
==========

```
ASIN : B012VQLT6Y
Brand : KM T-shirt
euclidean distance from the given image : 17.7625885612
================================================================================
============================================
```


```
ASIN : B00ZZMYBRG
Brand : HP-LEISURE
euclidean distance from the given image : 17.7795368647
================================================================================
============================================
```

3 [9] Text Semantics based product similarity

[ ]:
```
# 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)

'''
```

[20]:
```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
```

```
import pickle

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

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

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

[21]:
```
# Utility functions

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

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

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


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

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

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

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

    ax = plt.subplot(gs[0])
    # 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()
```

[22]:
```
# vocab = stores all the words that are there in google w2v model
# vocab = model.wv.vocab.keys() # if you are using Google word2Vec

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

    featureVec = np.zeros((num_features,), dtype="float32")
    # we will intialize a vector of size 300 with all zeros
    # we add each word2vec(wordi) to this fetureVec
    nwords = 0

    for word in sentence.split():
        nwords += 1
        if word in vocab:
```

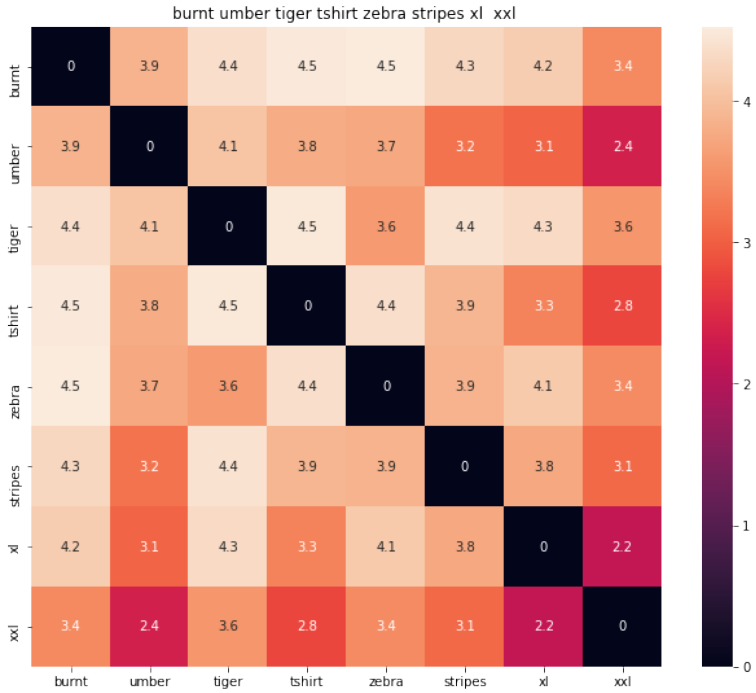
```
            if m_name == 'weighted' and word in  idf_title_vectorizer.
↪vocabulary_:
                featureVec = np.add(featureVec, idf_title_features[doc_id,␣
↪idf_title_vectorizer.vocabulary_[word]] * model[word])
            elif m_name == 'avg':
                featureVec = np.add(featureVec,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

3.0.1 [9.2] Average Word2Vec product similarity.

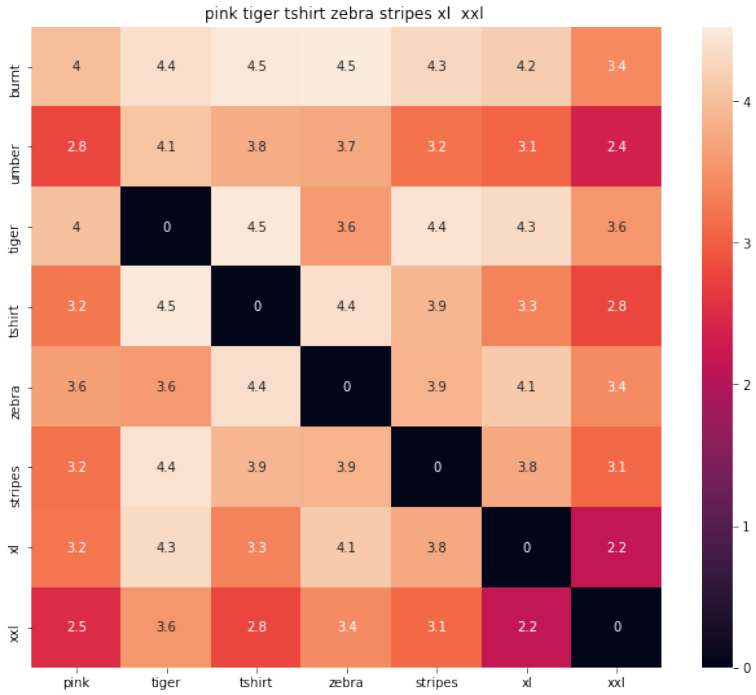
[44]:
```
doc_id = 0
w2v_title = []
# for every title we build a avg vector representation
for i in data['title']:
    w2v_title.append(build_avg_vec(i, 300, doc_id,'avg'))
    doc_id += 1

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

[45]:
```
def avg_w2v_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

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

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

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

    for i in range(0, len(indices)):
        heat_map_w2v(data['title'].loc[df_indices[0]],data['title'].
↪loc[df_indices[i]], data['medium_image_url'].loc[df_indices[i]], indices[0],␣
↪indices[i], 'avg')
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('BRAND :',data['brand'].loc[df_indices[i]])
        print ('euclidean distance from given input image :', pdists[i])
```

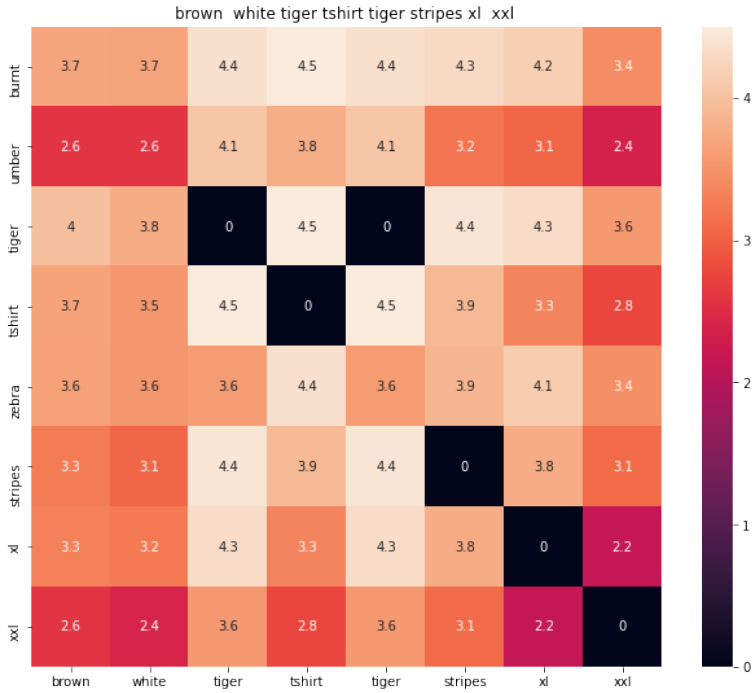
```
        print('='*125)


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



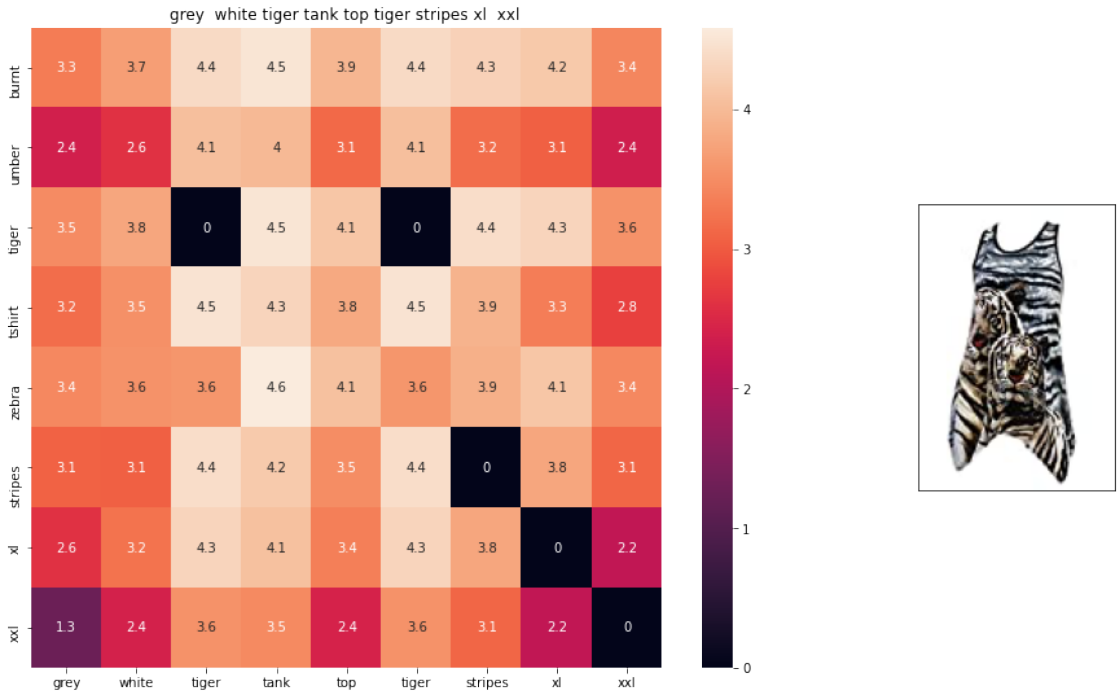
```
ASIN : B00JXQB5FQ
BRAND : Si Row
euclidean distance from given input image : 0.0
============================================================================================
==========================================
```



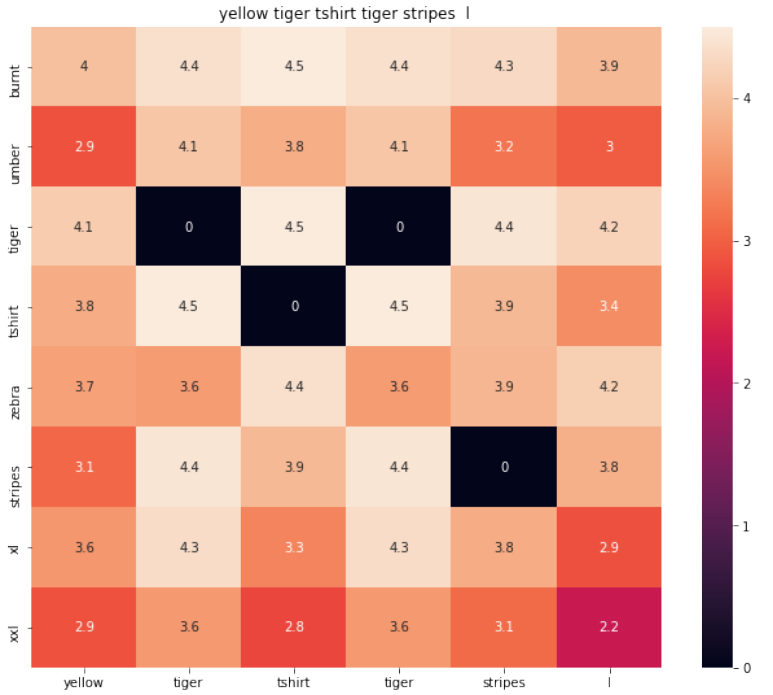
ASIN : B00JXQASS6
BRAND : Si Row
euclidean distance from given input image : 0.5891926
==========
==========



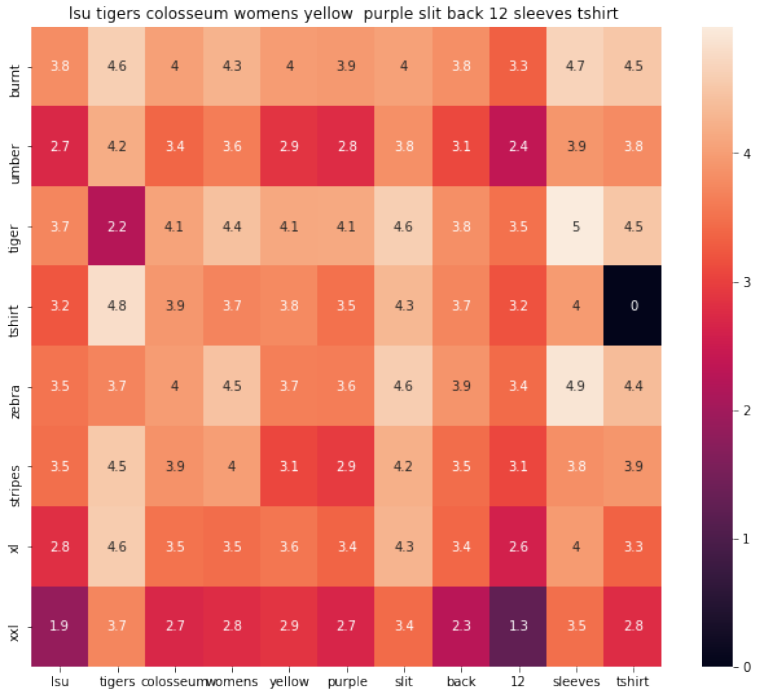
ASIN : B00JXQCWTO
BRAND : Si Row
euclidean distance from given input image : 0.7003438
==========
==========



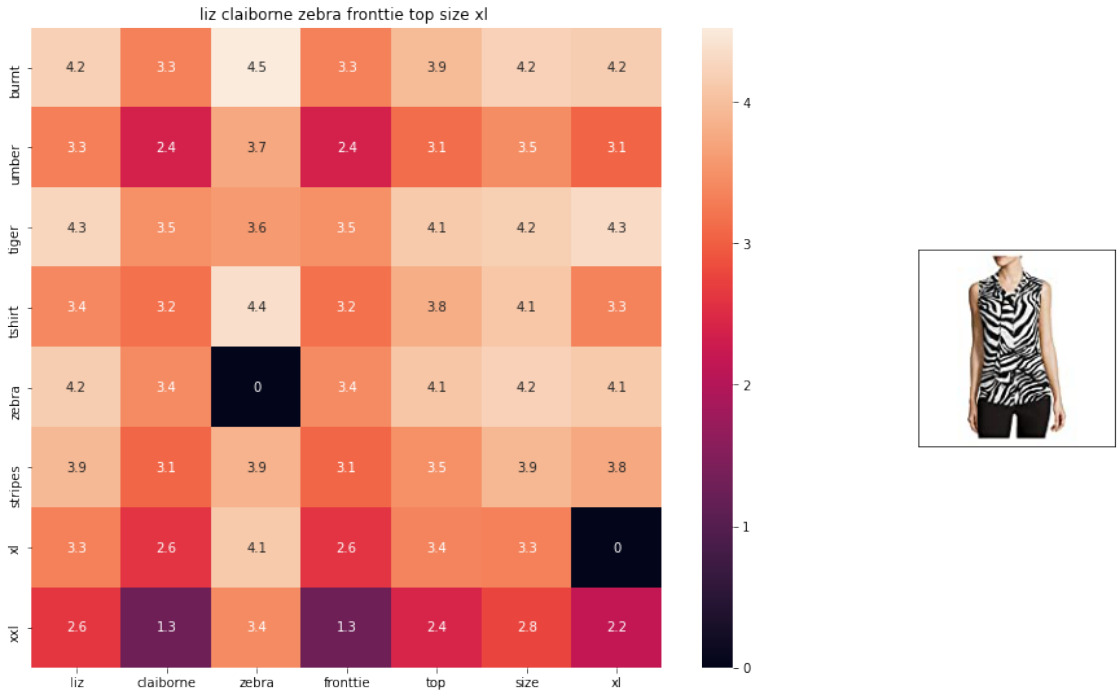
ASIN : B00JXQAFZ2
BRAND : Si Row
euclidean distance from given input image : 0.89283955
==========
==========



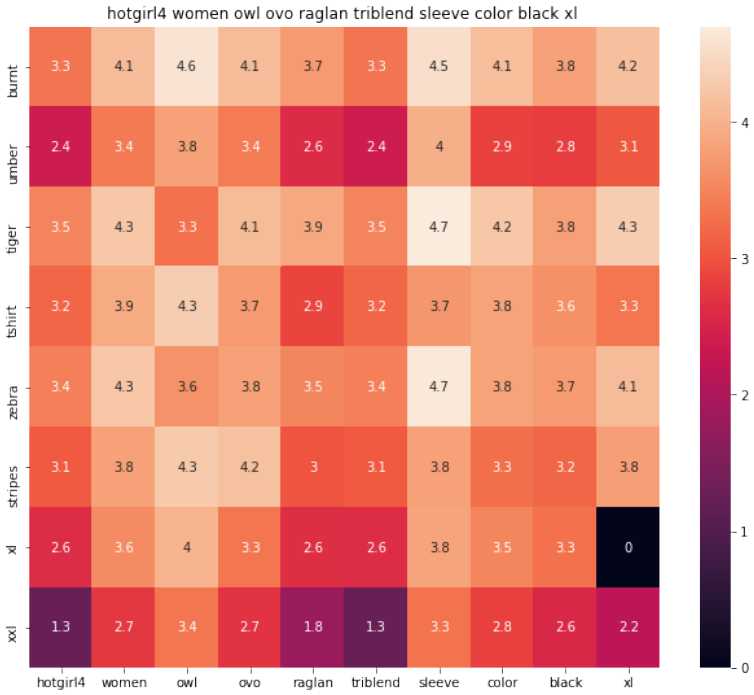
ASIN : B00JXQCUIC
BRAND : Si Row
euclidean distance from given input image : 0.95601255
==========
==========



ASIN : B073R5Q8HD
BRAND : Colosseum
euclidean distance from given input image : 1.022969
==========
==========

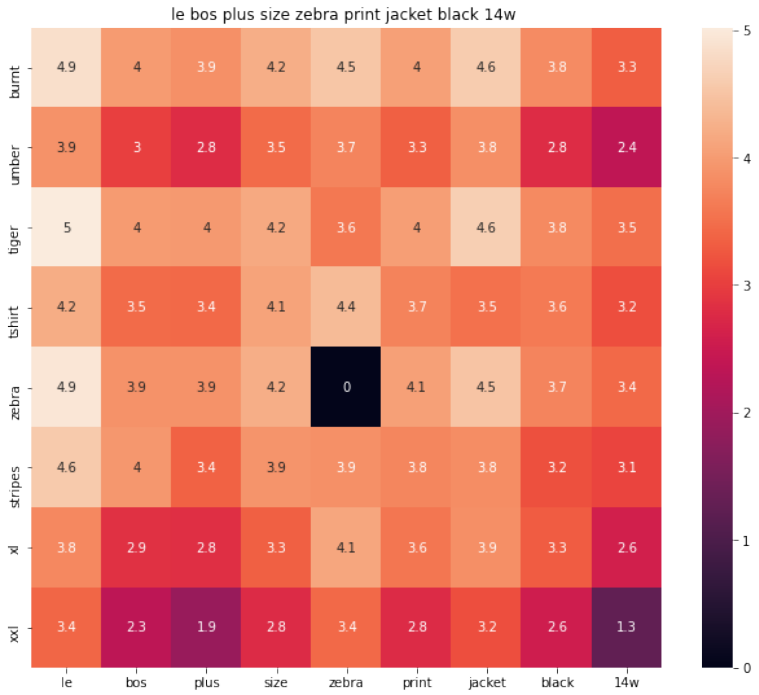


ASIN : B06XBY5QXL
BRAND : Liz Claiborne
euclidean distance from given input image : 1.0669324
==========
==========

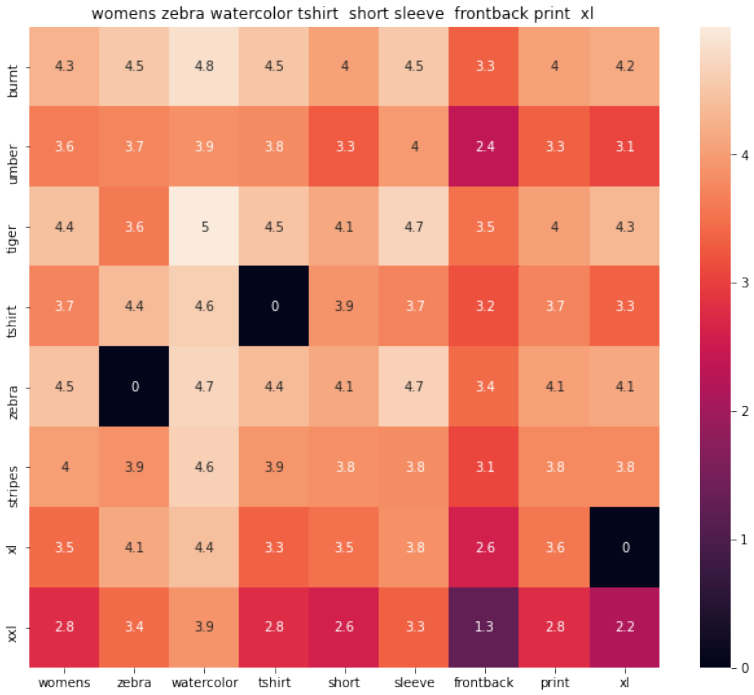


ASIN : B01L8L73M2
BRAND : Hotgirl4 Raglan Design
euclidean distance from given input image : 1.0731405
==========
==========

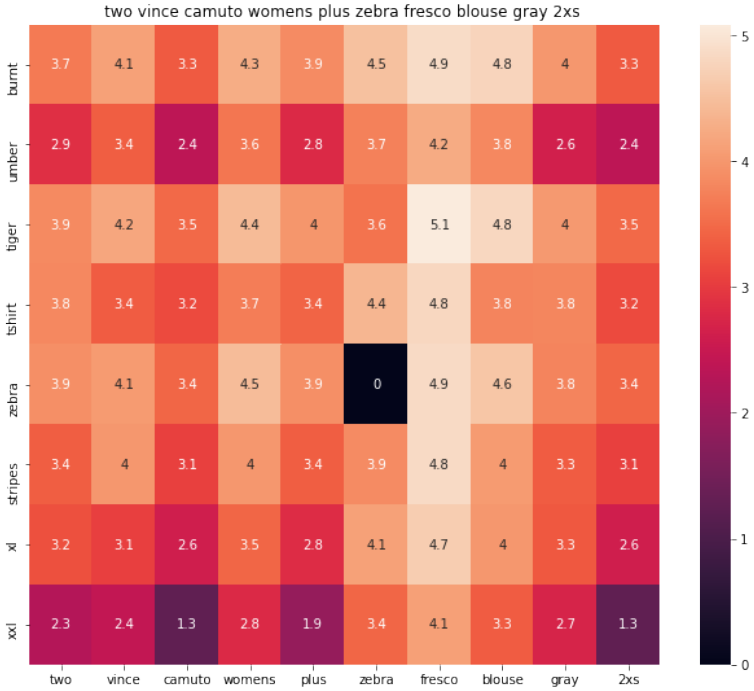
ASIN : B01EJS5H06
BRAND : Vansty
euclidean distance from given input image : 1.075719
==========
==========



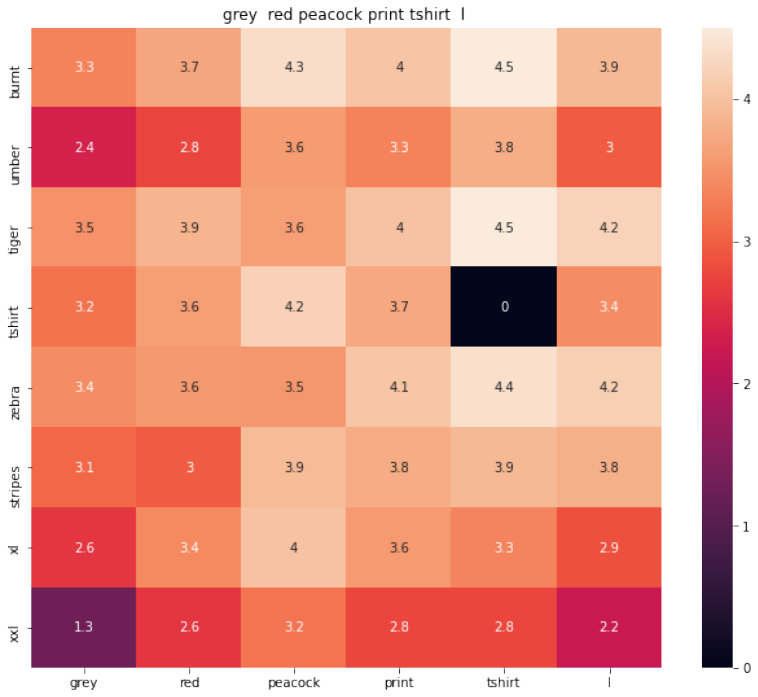
ASIN : B01BO1XRK8
BRAND : Le Bos
euclidean distance from given input image : 1.0839964
==========
==========



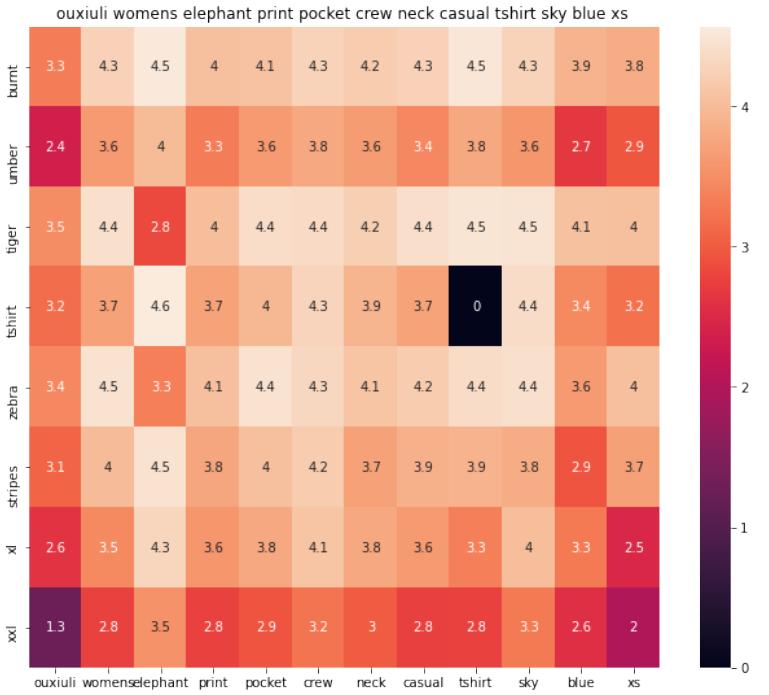
ASIN : B072R2JXKW
BRAND : WHAT ON EARTH
euclidean distance from given input image : 1.0842218
==========
==========



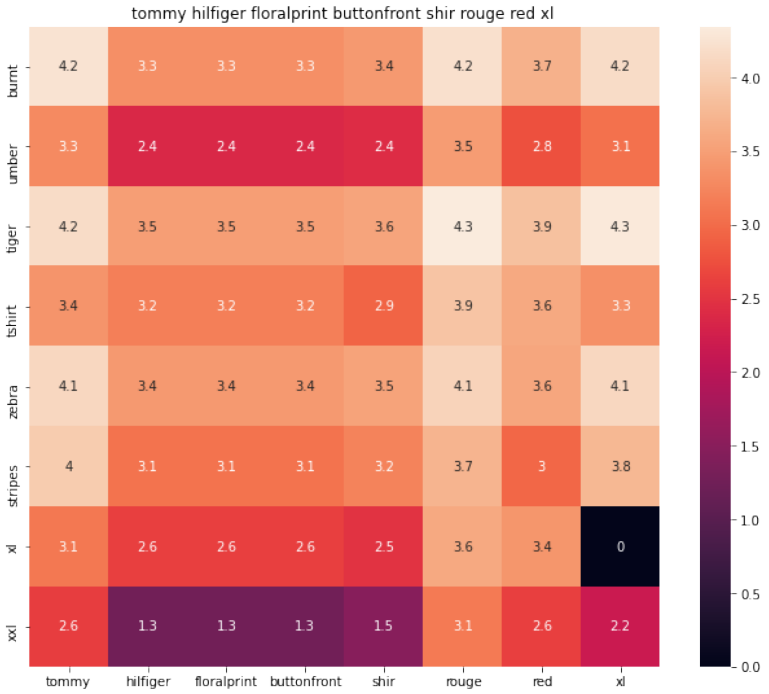
ASIN : B074MJRGW6
BRAND : Two by Vince Camuto
euclidean distance from given input image : 1.0895038
==========
==========



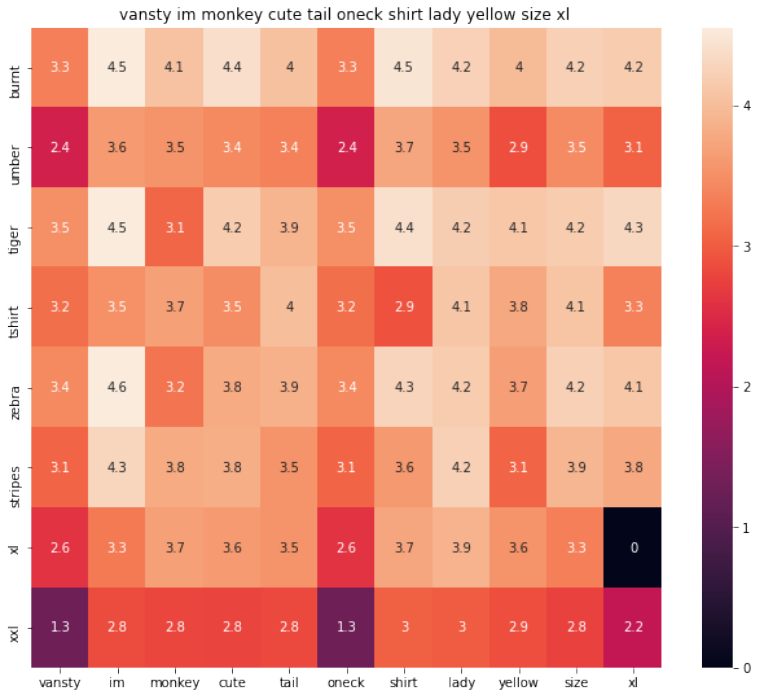
ASIN : B00JXQCFRS
BRAND : Si Row
euclidean distance from given input image : 1.0900588
==========
==========



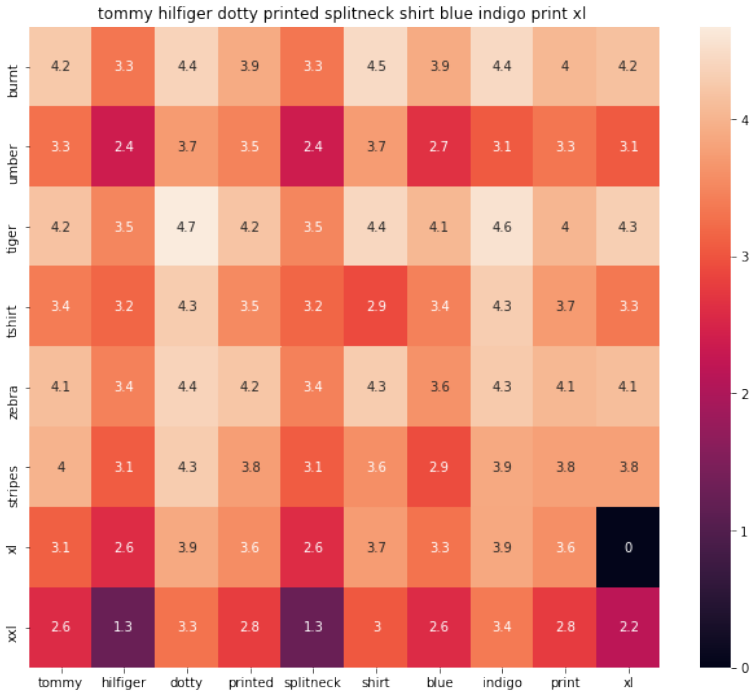
ASIN : B01I53HU6K
BRAND : ouxiuli
euclidean distance from given input image : 1.0920111
==========
==========



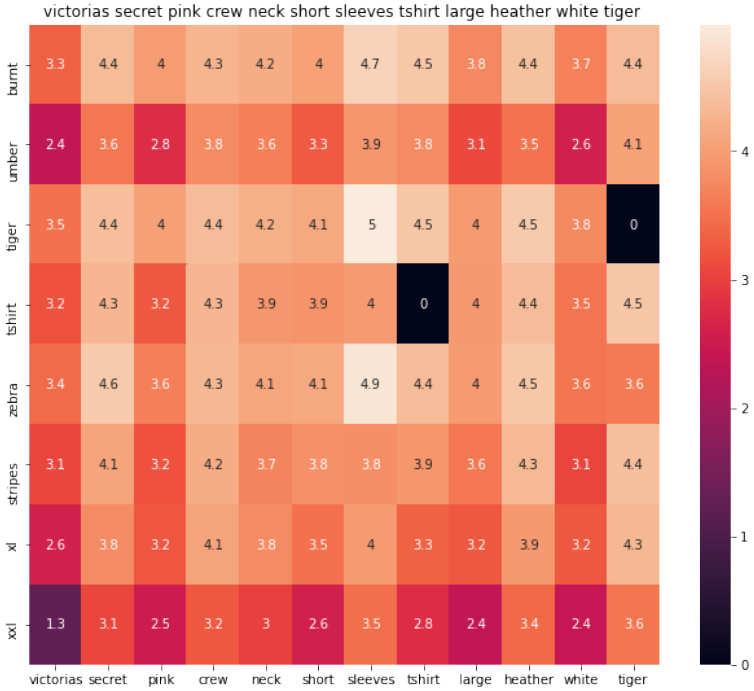
ASIN : B0711NGTQM
BRAND : THILFIGER RTW
euclidean distance from given input image : 1.0923415
==========
==========

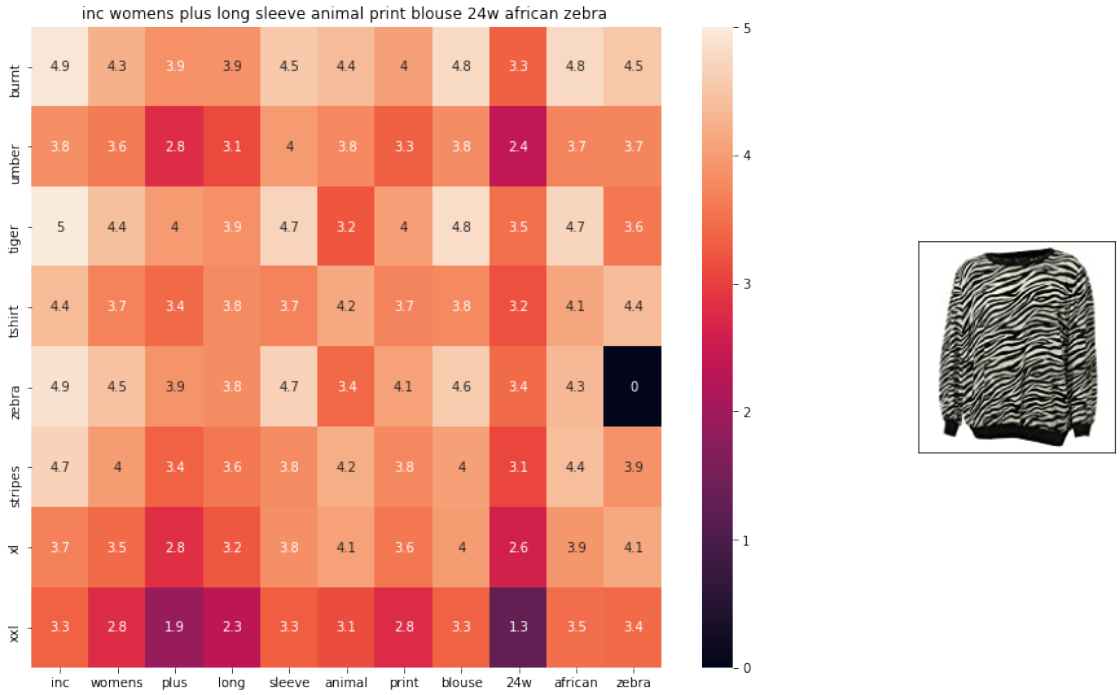


ASIN : B01EFSLO8Y
BRAND : Vansty
euclidean distance from given input image : 1.0934004
==========
==========



ASIN : B0716TVWQ4
BRAND : THILFIGER RTW
euclidean distance from given input image : 1.0942024
==========





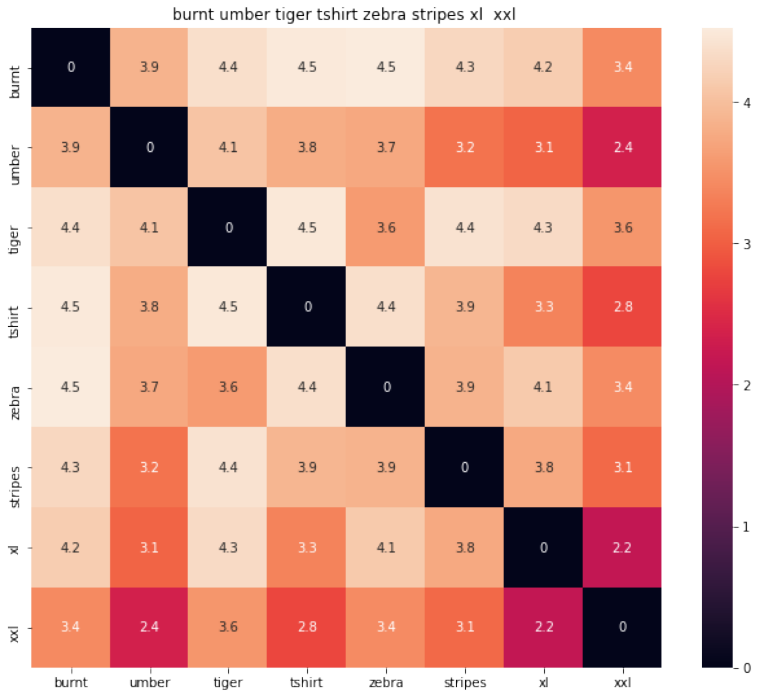

```
ASIN : B018WDJCUA
BRAND : INC - International Concepts Woman
euclidean distance from given input image : 1.0966892
================================================================================
==========================================
```

3.0.2 [9.4] IDF weighted Word2Vec for product similarity

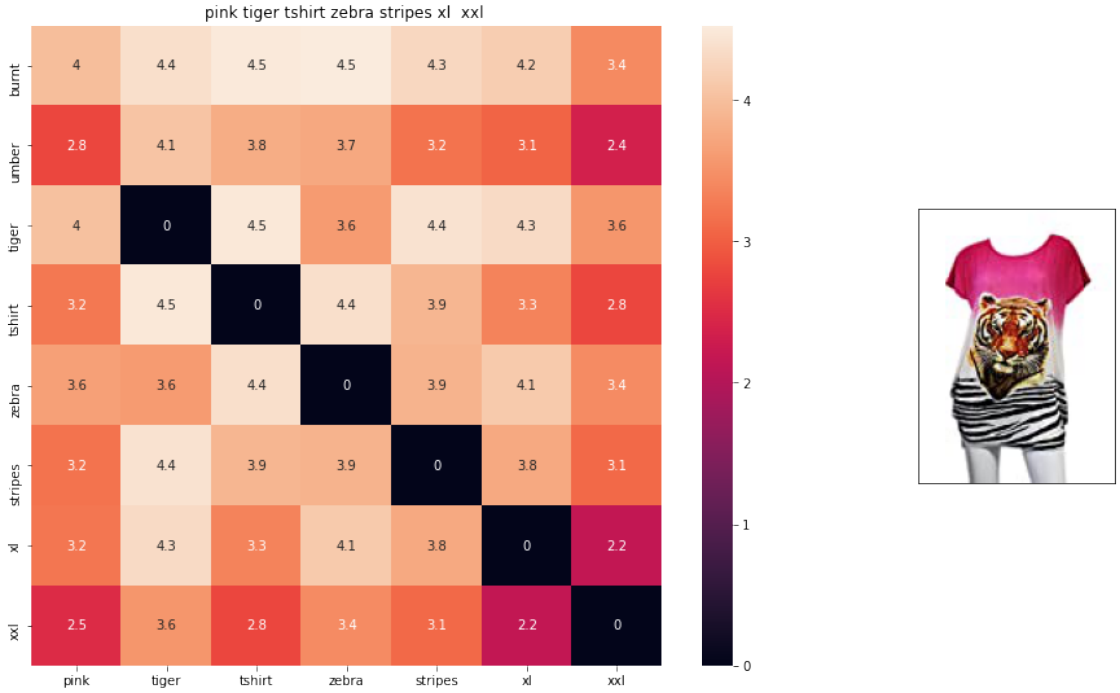
[49]:
```
doc_id = 0
w2v_title_weight = []
# for every title we build a weighted vector representation
for i in data['title']:
    w2v_title_weight.append(build_avg_vec(i, 300, doc_id,'weighted'))
    doc_id += 1
# w2v_title = np.array(# number of doc in courpus * 300), each row corresponds␣
 ↪to a doc
w2v_title_weight = np.array(w2v_title_weight)
```

[50]:
```
def weighted_w2v_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

    # pairwise_dist will store the distance from given input apparel to all␣
 ↪remaining apparels
    # the metric we used here is cosine, the coside distance is mesured as K(X,␣
 ↪Y) = <X, Y> / (||X||*||Y||)
```

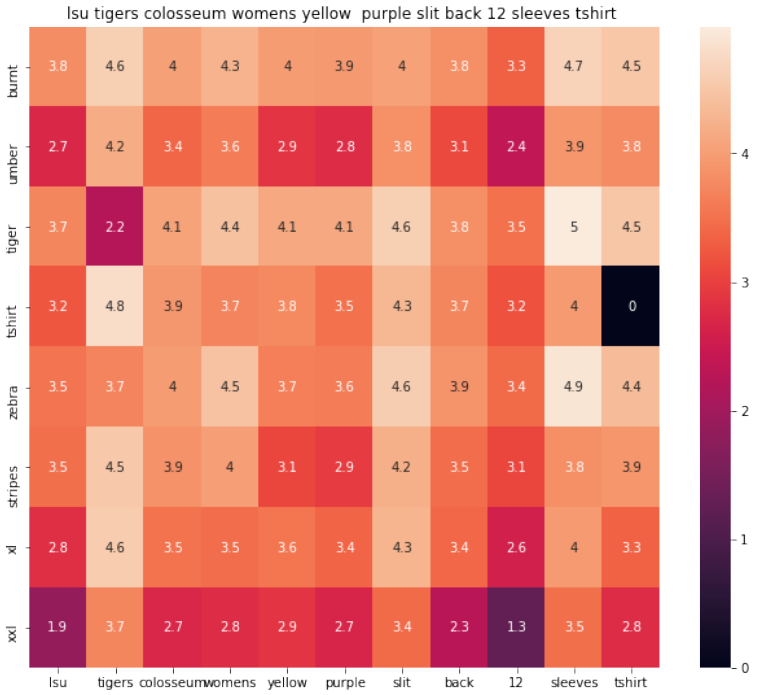
```
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(w2v_title_weight,␣
 ↪w2v_title_weight[doc_id].reshape(1,-1))

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

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

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

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



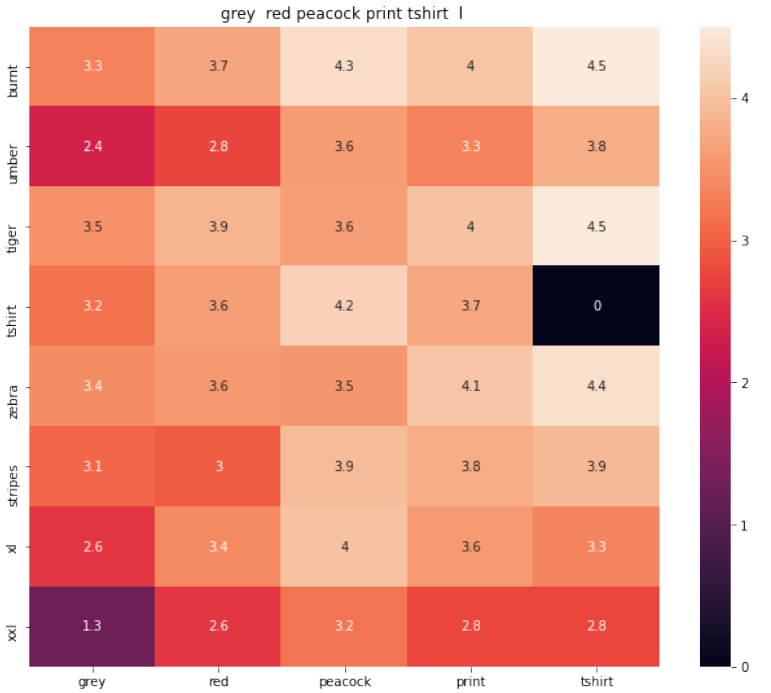
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 0.0
==
==



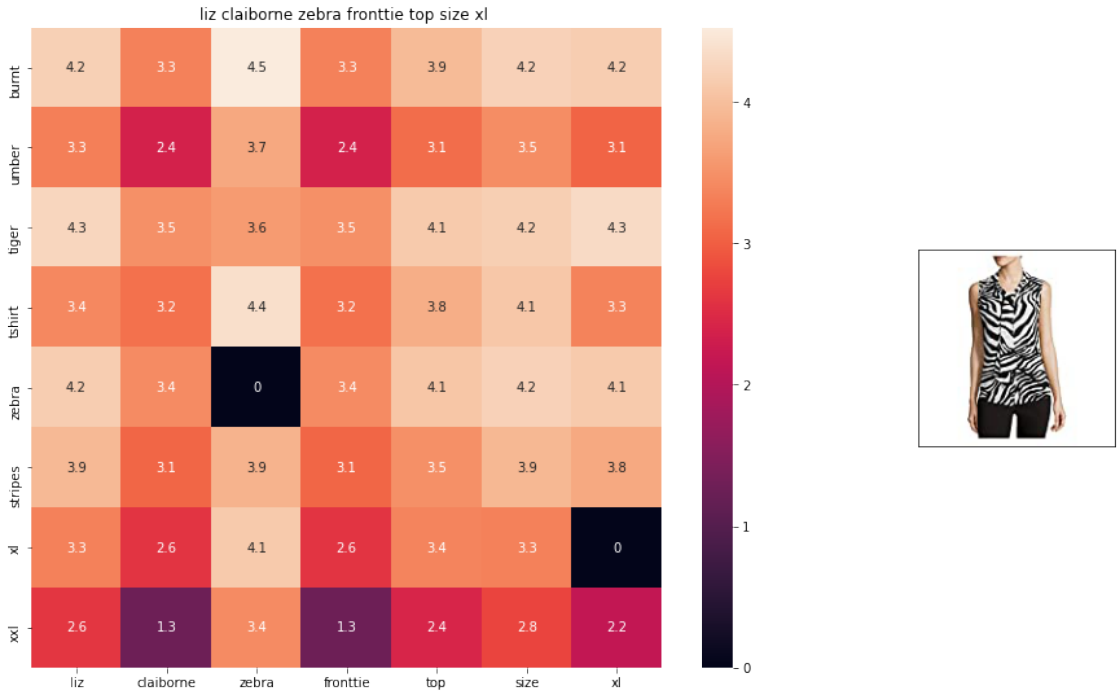
ASIN : B00JXQASS6
Brand : Si Row
euclidean distance from input : 0.5891926
==
==



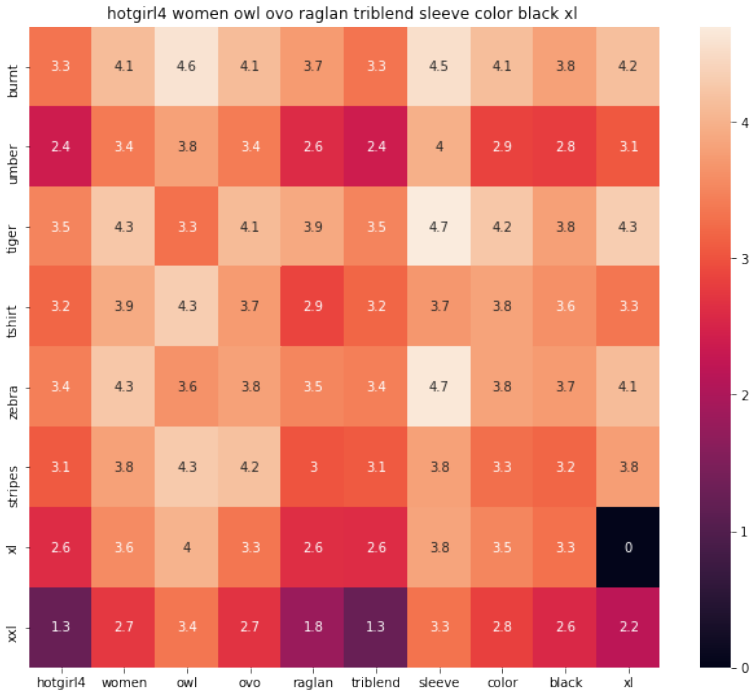
ASIN : B073R5Q8HD
Brand : Colosseum
euclidean distance from input : 1.022969
==========
==========



ASIN : B00JXQCFRS
Brand : Si Row
euclidean distance from input : 1.060353
==========
==========

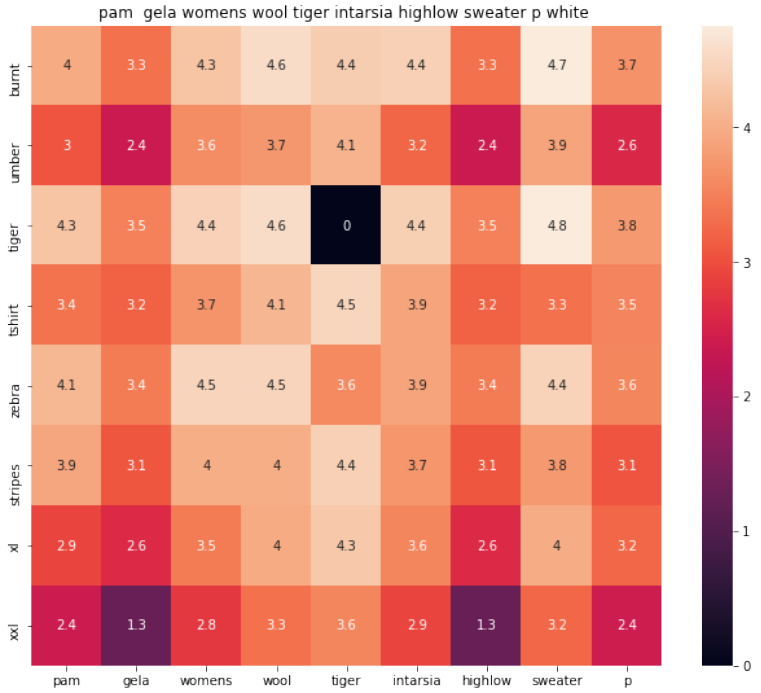


ASIN : B06XBY5QXL
Brand : Liz Claiborne
euclidean distance from input : 1.0669324
==========
==========



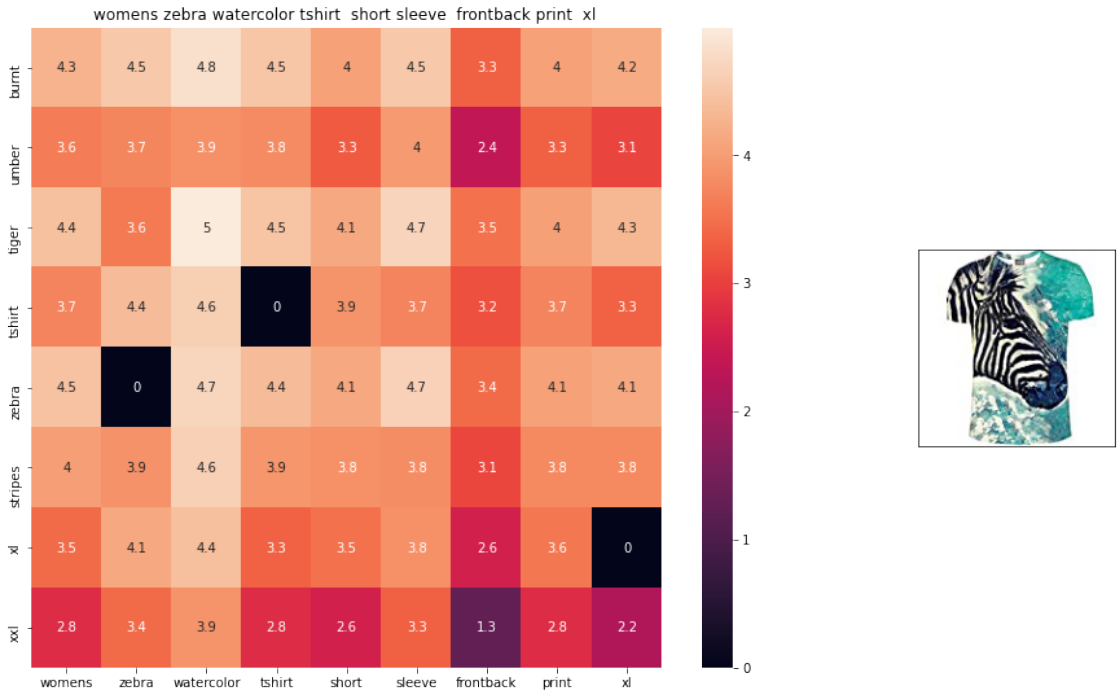
ASIN : B01L8L73M2
Brand : Hotgirl4 Raglan Design
euclidean distance from input : 1.0731405
==========
==========

ASIN : B01EJS5H06
Brand : Vansty
euclidean distance from input : 1.075719
==========
==========

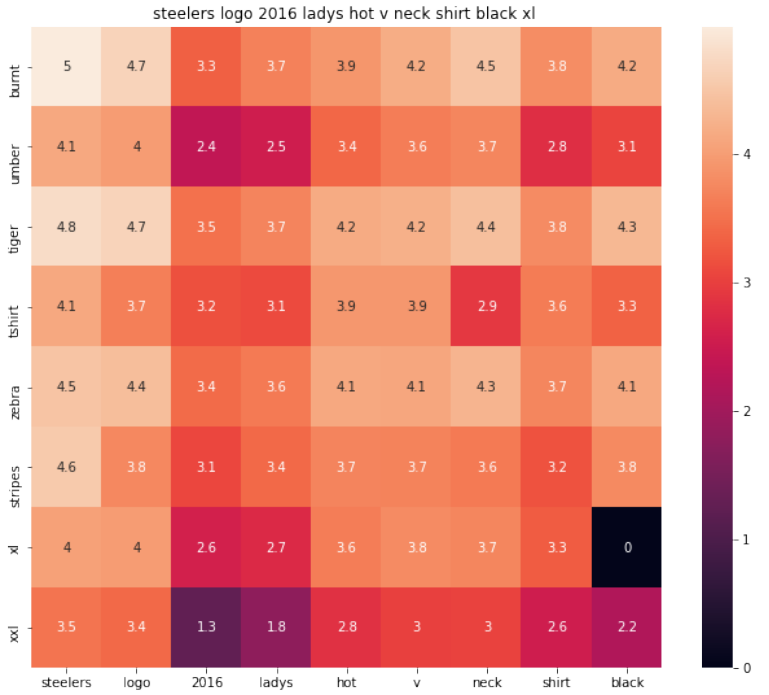


ASIN : B01MPX3HPW
Brand : Pam & Gela
euclidean distance from input : 1.0764749
==========
==========

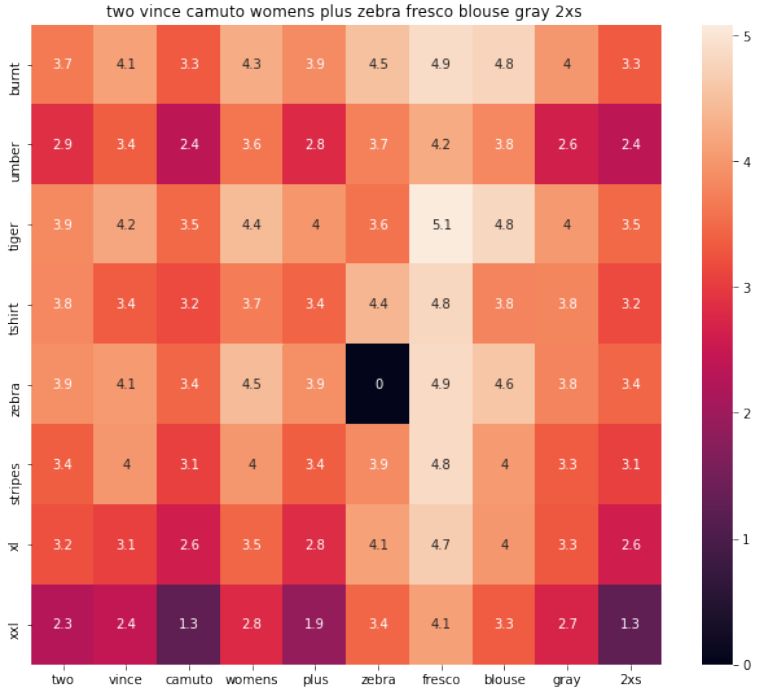
ASIN : B01BO1XRK8
Brand : Le Bos
euclidean distance from input : 1.0839964
==========
==========



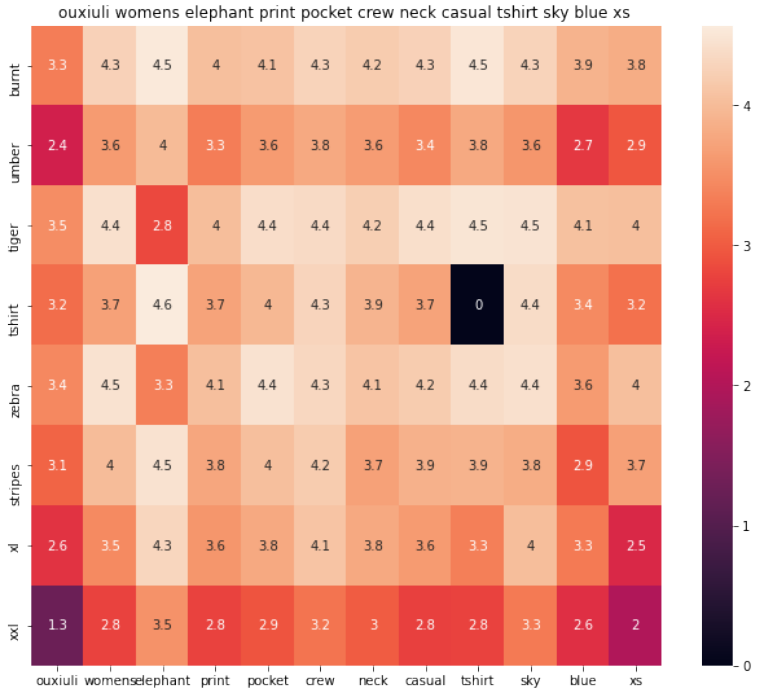
ASIN : B072R2JXKW
Brand : WHAT ON EARTH
euclidean distance from input : 1.0842218
==========
==========



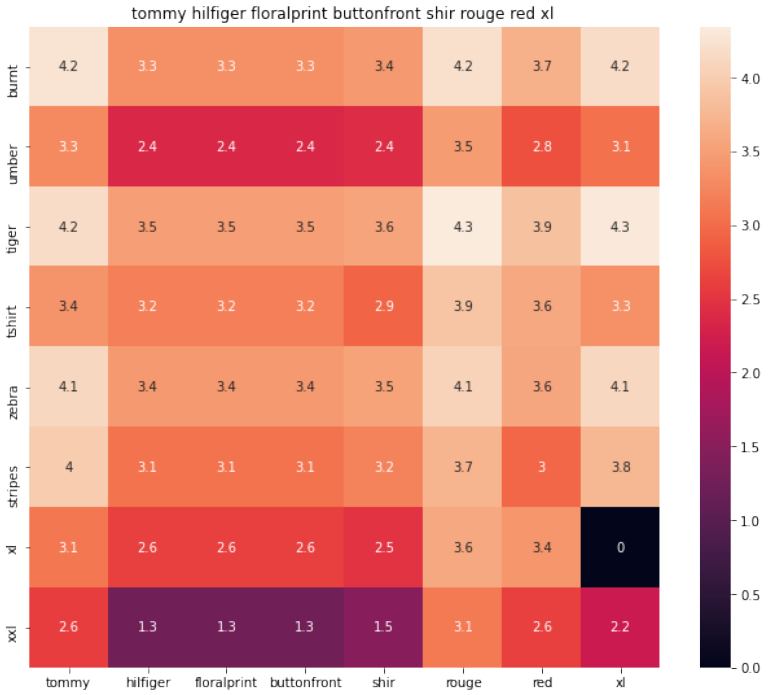
ASIN : B01LX6H2P7
Brand : BOBOB
euclidean distance from input : 1.0865405
==========
==========



ASIN : B074MJRGW6
Brand : Two by Vince Camuto
euclidean distance from input : 1.0895038
==========
==========

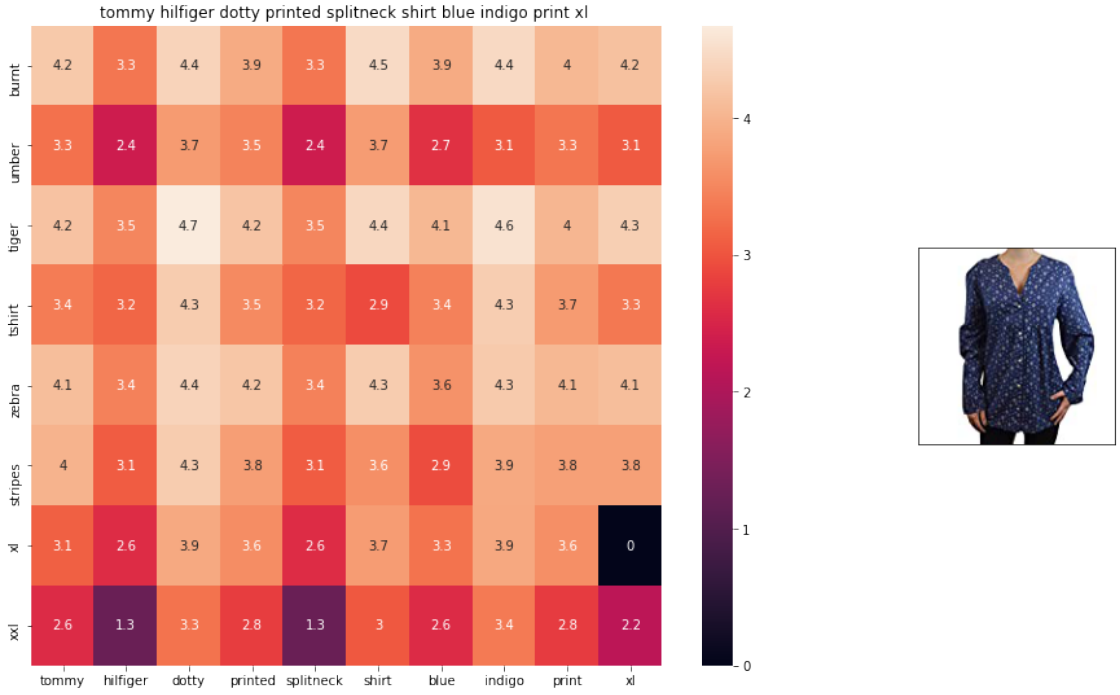


ASIN : B01I53HU6K
Brand : ouxiuli
euclidean distance from input : 1.0920111
==========
==========



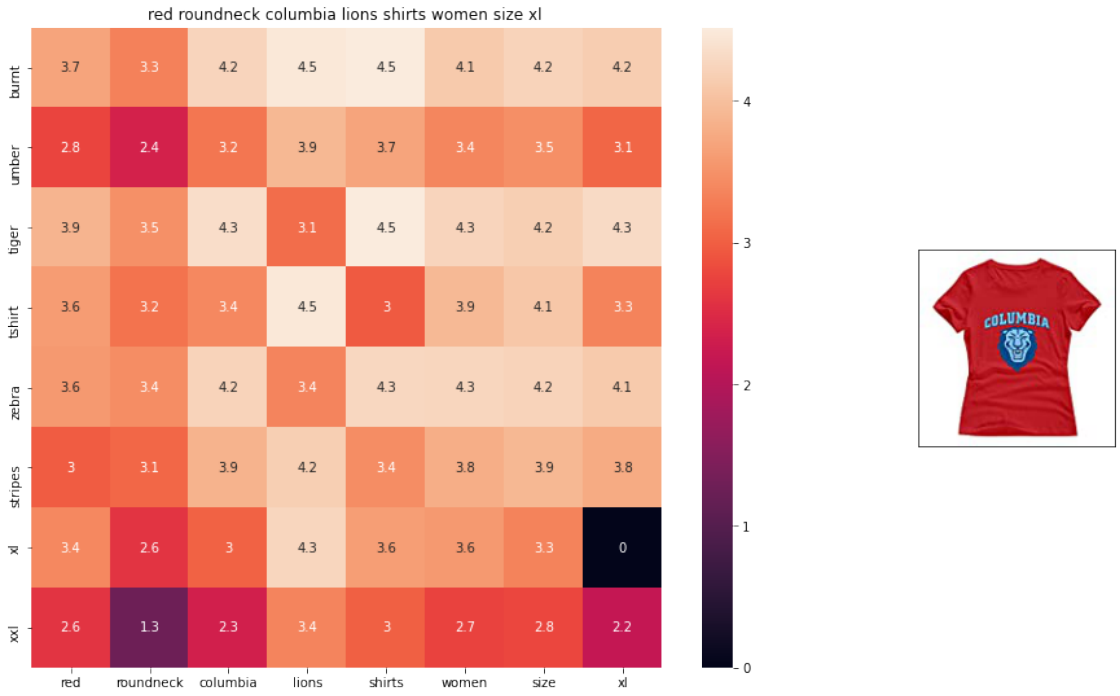
ASIN : B0711NGTQM
Brand : THILFIGER RTW
euclidean distance from input : 1.0923415
==========
==========

ASIN : B01EFSLO8Y
Brand : Vansty
euclidean distance from input : 1.0934004
==========
==========



ASIN : B0716TVWQ4
Brand : THILFIGER RTW
euclidean distance from input : 1.0942024
==========
==========

```
ASIN : B018WDJCUA
Brand : INC - International Concepts Woman
euclidean distance from input : 1.0966892
================================================================================
==========================================
```




```
ASIN : B017Q26FDA
Brand : Tavil
euclidean distance from input : 1.0974625
================================================================================
==========================================
```

3.0.3 [9.6] Weighted similarity using brand and color.

[51]:
```
# some of the brand values are empty.
# Need to replace Null with string "NULL"
data['brand'].fillna(value="Not given", inplace=True )

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

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

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

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

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

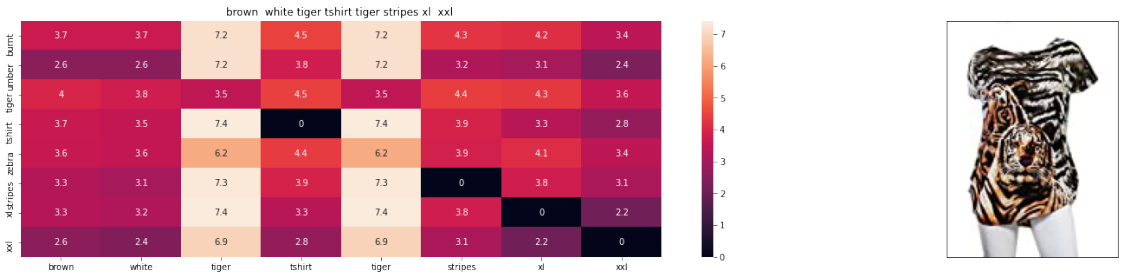
[55]:
```
def heat_map_w2v_brand(sentance1, sentance2, url, doc_id1, doc_id2, df_id1,␣
↪df_id2, model):

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

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

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

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

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

    # we create a table with the data_matrix
    table = ff.create_table(data_matrix, index=True, colorscale=colorscale)
```

```
    # plot it with plotly
    plotly.offline.iplot(table, filename='simple_table')

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

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

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

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

    plt.show()
```

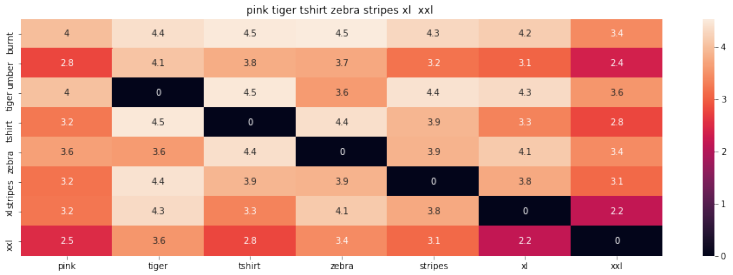
[56]:
```
def idf_w2v_brand(doc_id, w1, w2, num_results):
    # doc_id: apparel's id in given corpus
    # w1: weight for  w2v features
    # w2: weight for brand and color features

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

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

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

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


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

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

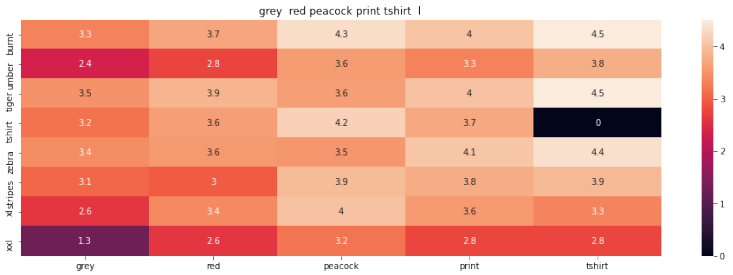

```
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 0.0
=============================================================================
=============================================
```



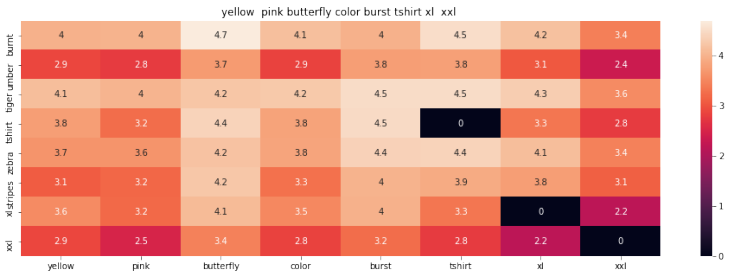
ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 0.6528799057006835
==========
==========

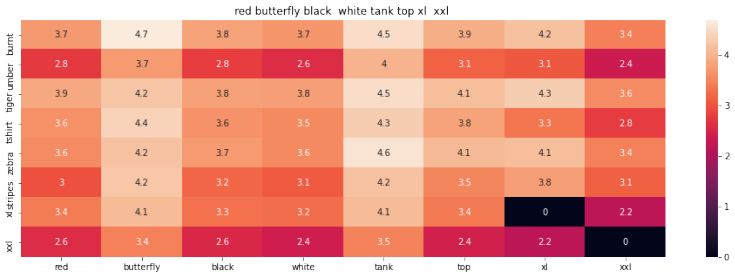


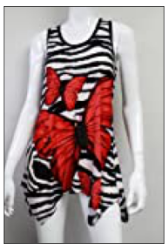
ASIN : B00JXQASS6
Brand : Si Row
euclidean distance from input : 1.0017030956167843
==========
==========



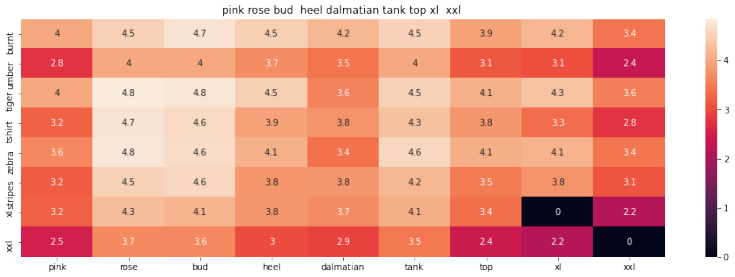
ASIN : B00JXQCFRS
Brand : Si Row
euclidean distance from input : 1.2372833253760007
==========
==========



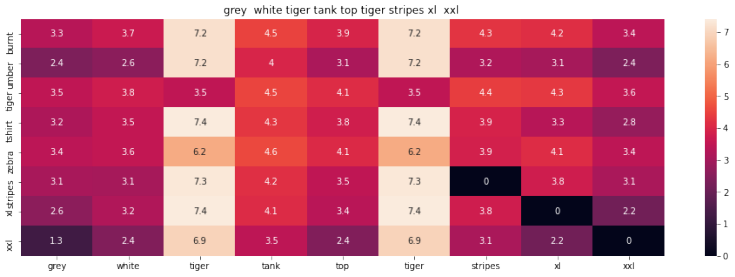




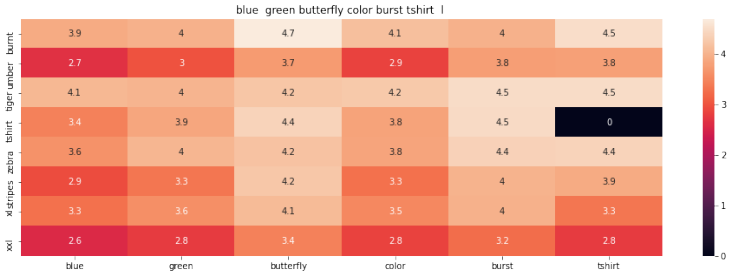
ASIN : B00JV63CW2
Brand : Si Row
euclidean distance from input : 1.2947488786596921
==========
==========



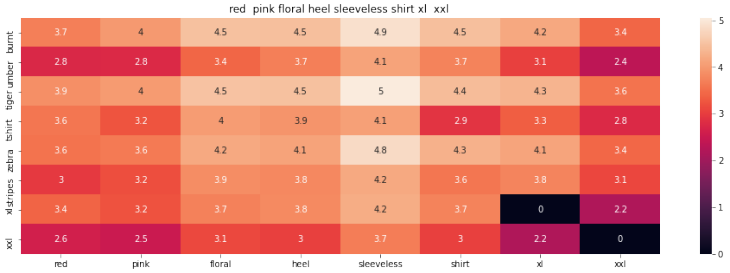
ASIN : B00JXQAX2C
Brand : Si Row
euclidean distance from input : 1.3236358167547848
==========
==========



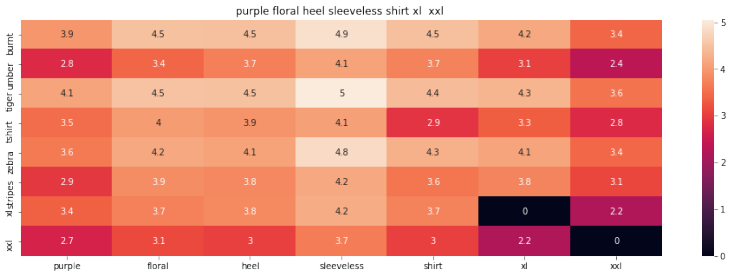
ASIN : B00JXQAFZ2
Brand : Si Row
euclidean distance from input : 1.3293567659277585
==========
==========



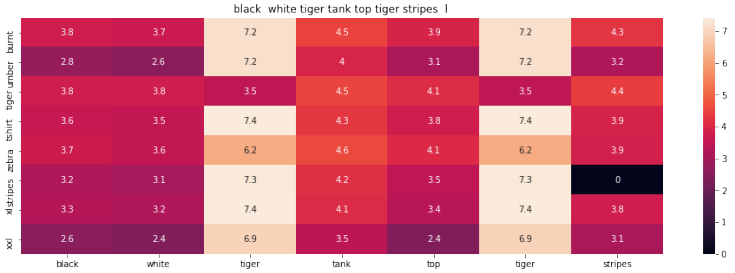
ASIN : B00JXQC0C8
Brand : Si Row
euclidean distance from input : 1.34993848818728
==========
==========



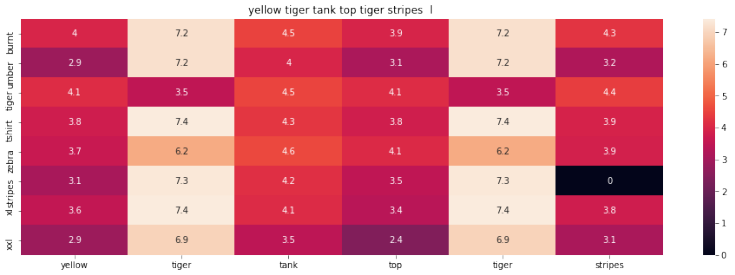
ASIN : B00JV63QQE
Brand : Si Row
euclidean distance from input : 1.3649898530859617
==========
==========



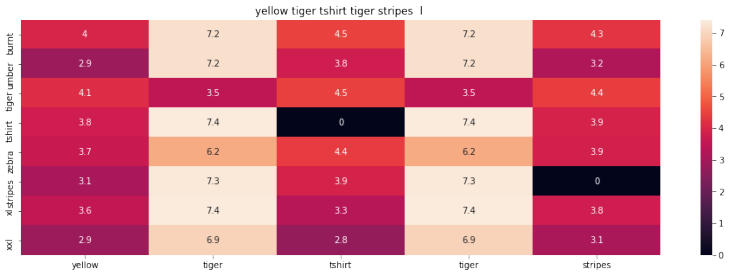
ASIN : B00JV63VC8
Brand : Si Row
euclidean distance from input : 1.4083902837652829
==========
==========



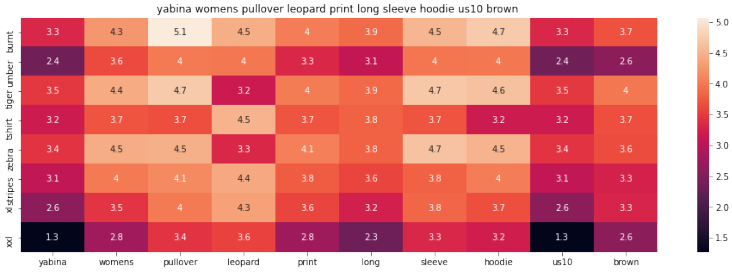
ASIN : B00JXQAO94
Brand : Si Row
euclidean distance from input : 1.5019501687903074
==========
==========



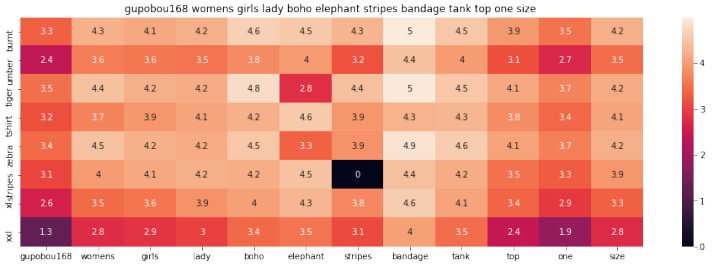
ASIN : B00JXQAUWA
Brand : Si Row
euclidean distance from input : 1.5693789483923581
==========
==========



ASIN : B00JXQCUIC
Brand : Si Row
euclidean distance from input : 1.633365059079614
==========
==========

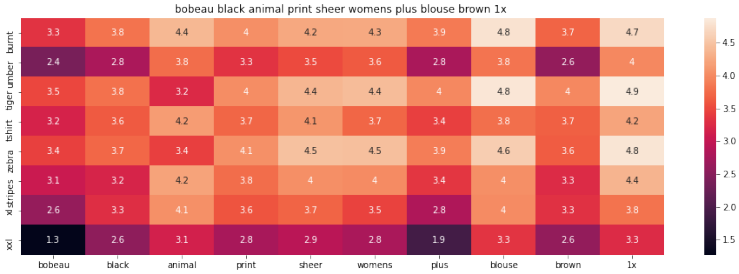


ASIN : B01KJUM6JI
Brand : YABINA
euclidean distance from input : 1.7323767736314868
==========
==========

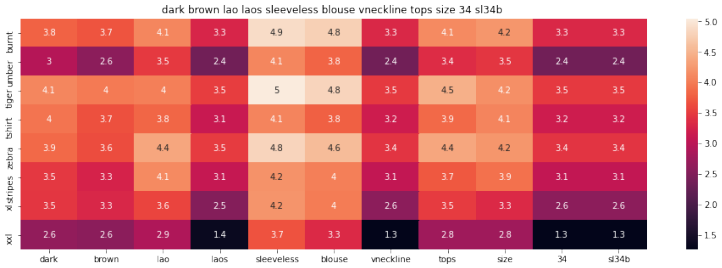


ASIN : B01ER184O6
Brand : GuPoBoU168

```
euclidean distance from input : 1.7352904394029711
================================================================================
==========================================
```

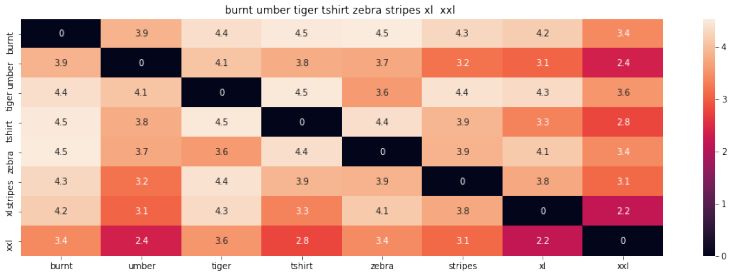


```
ASIN : B074MH886R
Brand : Bobeau
euclidean distance from input : 1.7428852155565355
================================================================================
==========================================
```



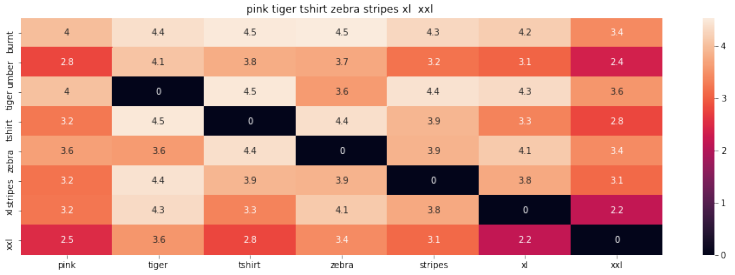
```
ASIN : B074J48RGW
Brand : Nanon
euclidean distance from input : 1.748470886027727
================================================================================
==========================================
```

[57]:
```
# brand and color weight =50
# title vector weight = 5

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

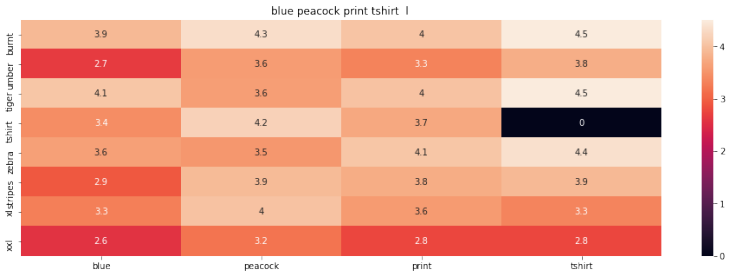


ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 0.0
==========

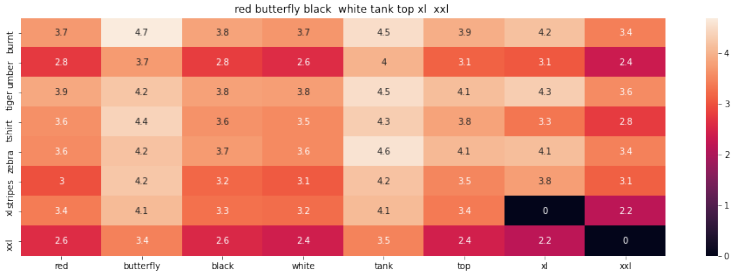
ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 0.11870543740012429
==========



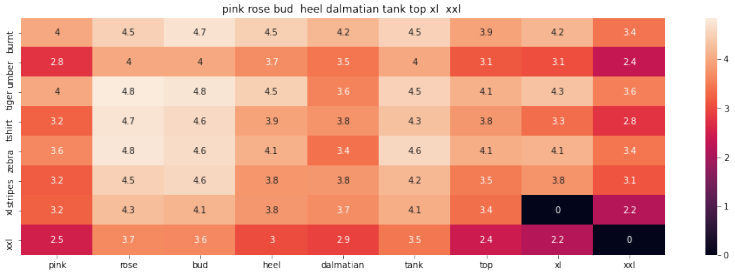
ASIN : B00JXQABB0
Brand : Si Row
euclidean distance from input : 1.388601728247669
==========
==========



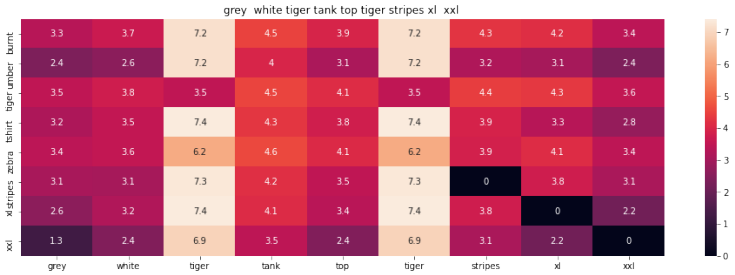
ASIN : B00JXQC8L6
Brand : Si Row
euclidean distance from input : 1.3901508681374728
==========
==========



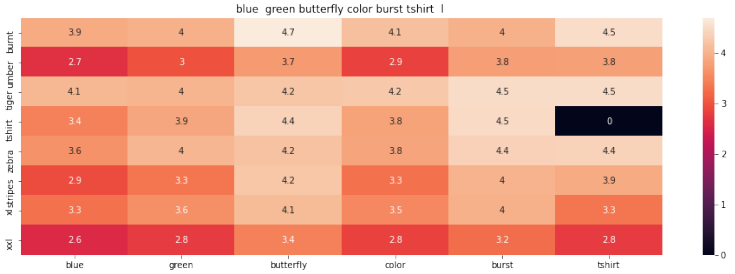
ASIN : B00JV63CW2
Brand : Si Row
euclidean distance from input : 1.39249271078884
==========
==========



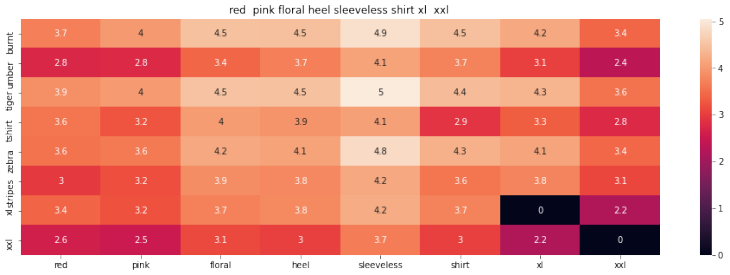
ASIN : B00JXQAX2C
Brand : Si Row
euclidean distance from input : 1.3977448813515843
==========
==========



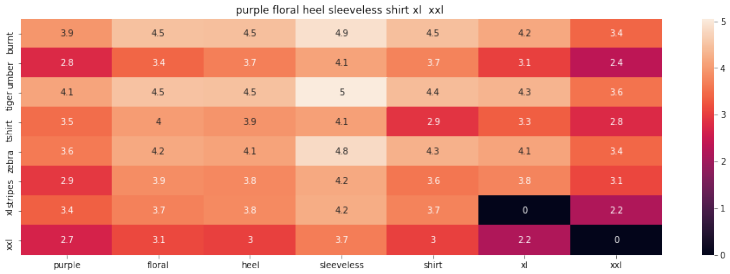
ASIN : B00JXQAFZ2
Brand : Si Row
euclidean distance from input : 1.3987850539284885
==========
==========



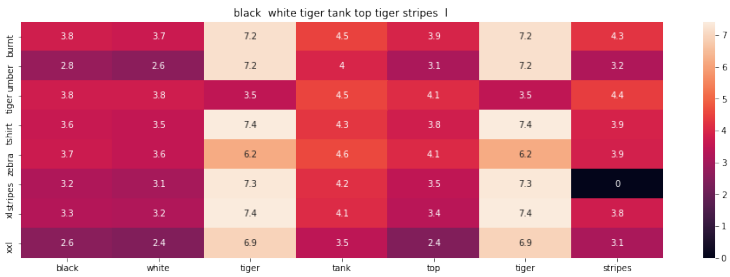
ASIN : B00JXQC0C8
Brand : Si Row
euclidean distance from input : 1.4025271852484014
==========
==========



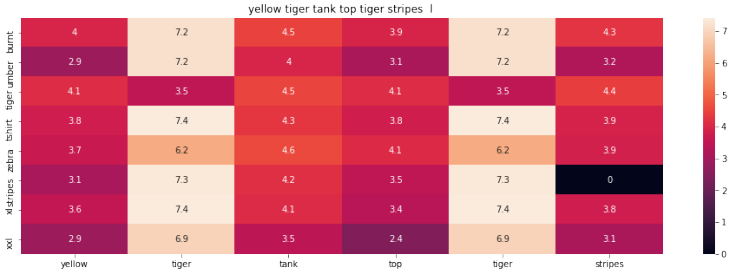
ASIN : B00JV63QQE
Brand : Si Row
euclidean distance from input : 1.4052637970481618
==========
==========



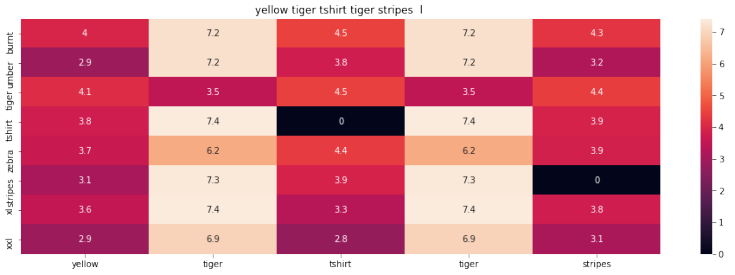
ASIN : B00JV63VC8
Brand : Si Row
euclidean distance from input : 1.413154784444402
==========
==========



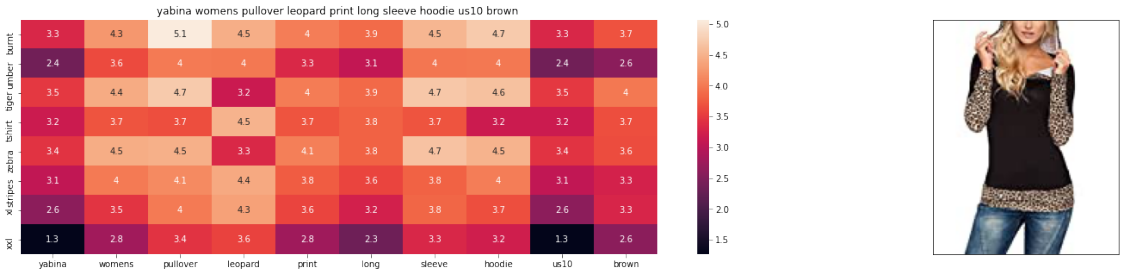
ASIN : B00JXQAO94
Brand : Si Row
euclidean distance from input : 1.43016567263077



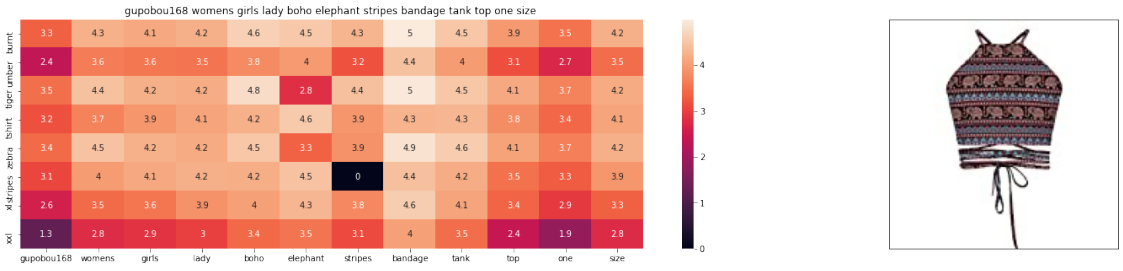
ASIN : B00JXQAUWA
Brand : Si Row
euclidean distance from input : 1.4424254507402339



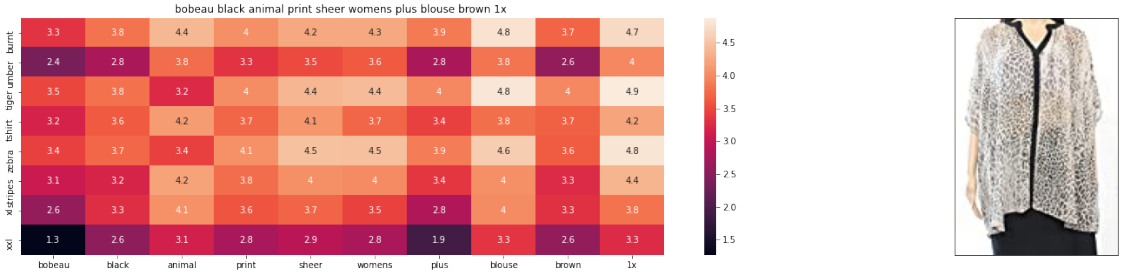
ASIN : B00JXQCUIC
Brand : Si Row
euclidean distance from input : 1.4540592890470077



ASIN : B01KJUM6JI
Brand : YABINA
euclidean distance from input : 2.144487758614644
==========
==========

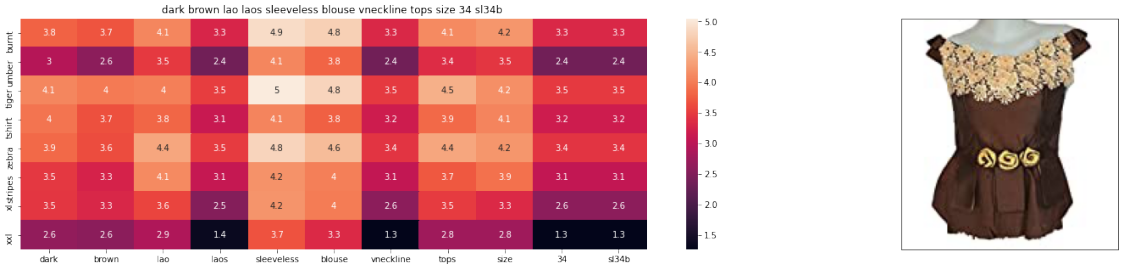


ASIN : B01ER184O6
Brand : GuPoBoU168
euclidean distance from input : 2.145017516027641
==========
==========



ASIN : B074MH886R
Brand : Bobeau

```
euclidean distance from input : 2.146398384419198
================================================================================
============================================
```




```
ASIN : B074J48RGW
Brand : Nanon
euclidean distance from input : 2.1474139608685054
================================================================================
============================================
```

4 [10.2] Keras and Tensorflow to extract features

[ ]:
```
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.
```

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



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

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

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

# each image is converted into 25088 length dense-vector


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

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


def save_bottlebeck_features():

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

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

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

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

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

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

```
save_bottlebeck_features()

'''
```

5 [10.3] Visual features based product similarity.

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

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


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


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

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

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

get_similar_products_cnn(12566, 20)
```

Product Title: burnt umber tiger tshirt zebra stripes xl xxl
Euclidean Distance from input image: 0.0
Amazon Url: www.amzon.com/dp/B00JXQB5FQ

Product Title: pink tiger tshirt zebra stripes xl xxl
Euclidean Distance from input image: 30.0501
Amazon Url: www.amzon.com/dp/B00JXQASS6

Product Title: yellow tiger tshirt tiger stripes l
Euclidean Distance from input image: 41.2611
Amazon Url: www.amzon.com/dp/B00JXQCUIC

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

Product Title: kawaii pastel tops tees pink flower design
Euclidean Distance from input image: 47.3825
Amazon Url: www.amzon.com/dp/B071FCWD97

Product Title: womens thin style tops tees pastel watermelon print
Euclidean Distance from input image: 47.7184
Amazon Url: www.amzon.com/dp/B01JUNHBRM

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

Product Title: edv cheetah run purple multi xl
Euclidean Distance from input image: 48.0465
Amazon Url: www.amzon.com/dp/B01CUPYBM0

Product Title: danskin womens vneck loose performance tee xsmall pink ombre
Euclidean Distance from input image: 48.1019
Amazon Url: www.amzon.com/dp/B01F7PHXY8

Product Title: summer alpaca 3d pastel casual loose tops tee design
Euclidean Distance from input image: 48.1189
Amazon Url: www.amzon.com/dp/B01I80A93G

Product Title: miss chievous juniors striped peplum tank top medium shadowpeach
Euclidean Distance from input image: 48.1313
Amazon Url: www.amzon.com/dp/B0177DM70S

Product Title: red pink floral heel sleeveless shirt xl xxl
Euclidean Distance from input image: 48.1695
Amazon Url: www.amzon.com/dp/B00JV63QQE

Product Title: moana logo adults hot v neck shirt black xxl
Euclidean Distance from input image: 48.2568
Amazon Url: www.amzon.com/dp/B01LX6H43D

Product Title: abaday multicolor cartoon cat print short sleeve longline shirt large
Euclidean Distance from input image: 48.2657
Amazon Url: www.amzon.com/dp/B01CR57YY0

Product Title: kawaii cotton pastel tops tees peach pink cactus design
Euclidean Distance from input image: 48.3626
Amazon Url: www.amzon.com/dp/B071WYLBZS

Product Title: chicago chicago 18 shirt women pink
Euclidean Distance from input image: 48.3836
Amazon Url: www.amzon.com/dp/B01GXAZTRY

Product Title: yichun womens tiger printed summer tshirts tops
Euclidean Distance from input image: 48.4493
Amazon Url: www.amzon.com/dp/B010NN9RXO

Product Title: nancy lopez whimsy short sleeve whiteblacklemon drop xs
Euclidean Distance from input image: 48.4788
Amazon Url: www.amzon.com/dp/B01MPX6IDX

6 Brand, color, type, Idf, CNN features with weight

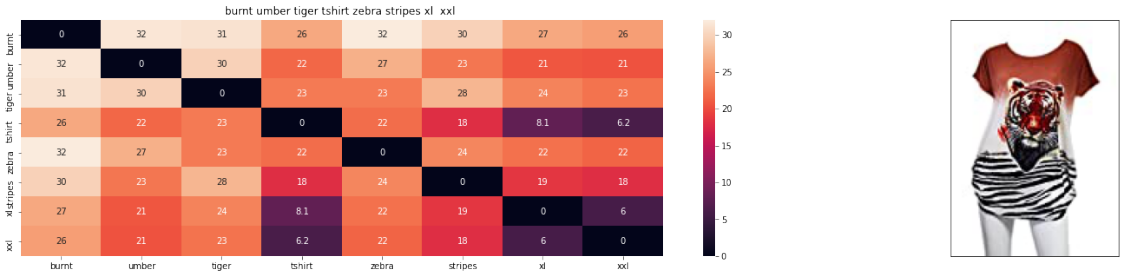
[64]:
```
# some of the brand values are empty.
# Need to replace Null with string "NULL"
data['brand'].fillna(value="Not given", inplace=True )

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

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

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

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

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

[69]:
```
def idf_w2v_brand_with_image(doc_id, b, c, t, i, vgg, num_results):

    # ex_feat_dist = pairwise_distances(extra_features, extra_features[doc_id])

    brand_feat_dist = pairwise_distances(brand_features, brand_features[doc_id])

    type_feat_dist = pairwise_distances(type_features, type_features[doc_id])

    color_feat_dist = pairwise_distances(color_features, color_features[doc_id])

    idf_w2v_dist  = pairwise_distances(w2v_title_weight,␣
 ↪w2v_title_weight[doc_id].reshape(1,-1))

    CNN = pairwise_distances(CNN_feat, CNN_feat[doc_id].reshape(1,-1))

    pairwise_dist   = (i*idf_w2v_dist +  b*brand_feat_dist + t*type_feat_dist +␣
 ↪c*color_feat_dist + vgg*CNN)/float(b+c+t+i+vgg)


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

    df_indices = list(data.index[indices])
```

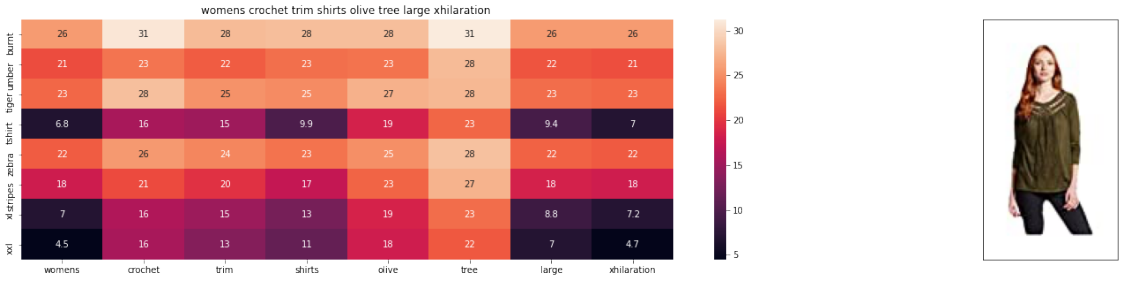
```
    for i in range(0, len(indices)):
        heat_map_w2v_brand(data['title'].loc[df_indices[0]],data['title'].
↪loc[df_indices[i]], data['medium_image_url'].loc[df_indices[i]], indices[0],␣
↪indices[i],df_indices[0], df_indices[i], 'weighted')
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('Brand :',data['brand'].loc[df_indices[i]])
        print('euclidean distance from input :', pdists[i])
        print('='*160)
```

6.1 Equal weights to All Features

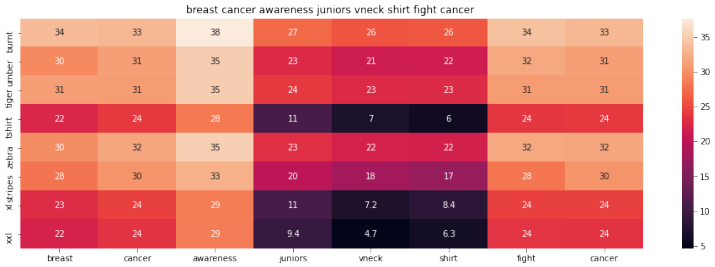
[70]:
```
idf_w2v_brand_with_image(12566,5,5,5,5,5, 10)
```



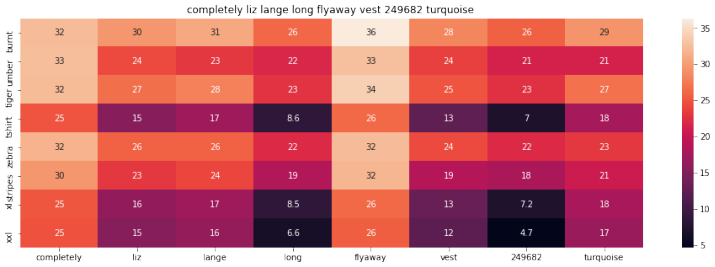

```
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 1.5018477279227227e-06
================================================================================
================================================================================
```



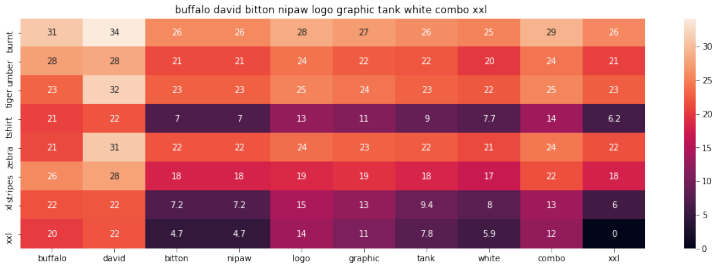

```
ASIN : B06XBHNM7J
Brand : Xhilaration
euclidean distance from input : 8.69540840445969
================================================================================
================================================================================
```



ASIN : B016CU40IY
Brand : Juiceclouds
euclidean distance from input : 9.051878644629516
==========
==========



ASIN : B074LTBWSW
Brand : Liz Lange
euclidean distance from input : 9.107939071799825
==========
==========

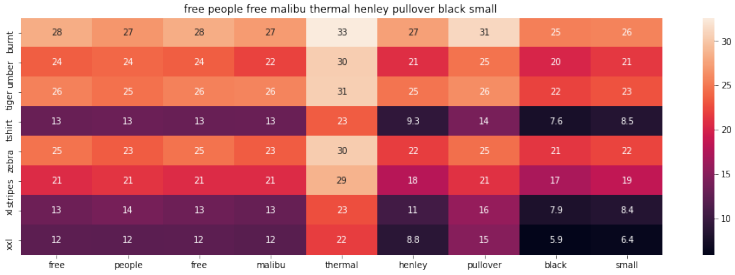


ASIN : B018H5AZXQ
Brand : Buffalo

euclidean distance from input : 9.122009356608126

==========

==========



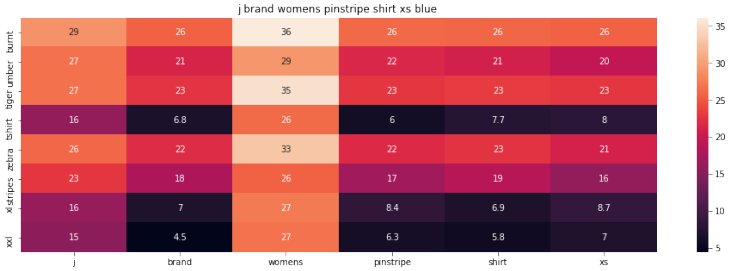
ASIN : B074MXY984

Brand : We The Free

euclidean distance from input : 9.214430316225563

==========

==========



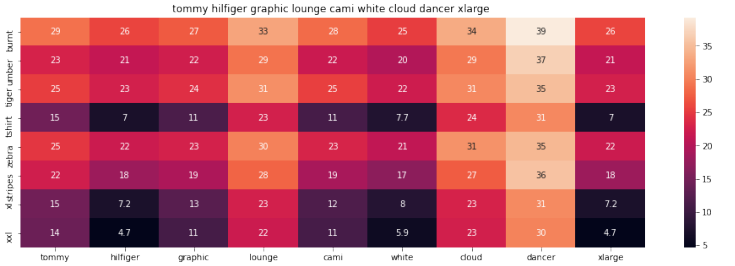
ASIN : B06XYP1X1F

Brand : J Brand Jeans

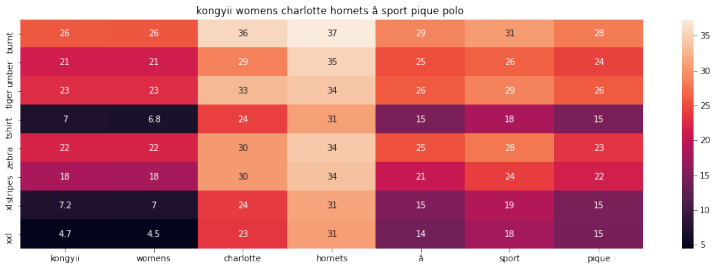
euclidean distance from input : 9.264000568462167

==========

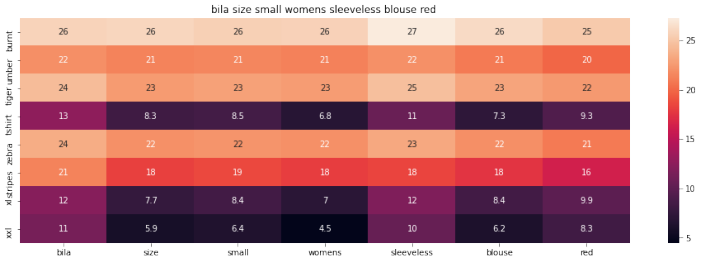
==========



```
ASIN : B01BMSFYW2
Brand : igertommy hilf
euclidean distance from input : 9.296391849590098
================================================================================
================================================================================
```



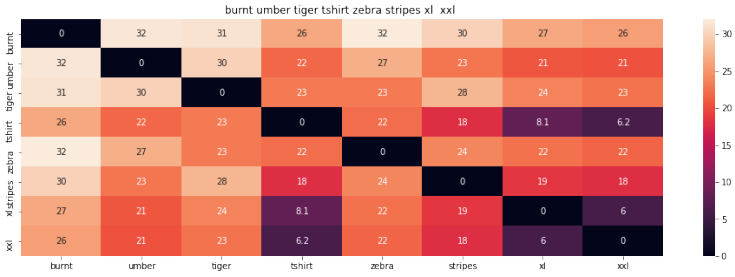
```
ASIN : B01FJVZST2
Brand : KONGYII
euclidean distance from input : 9.368445359869996
================================================================================
================================================================================
```



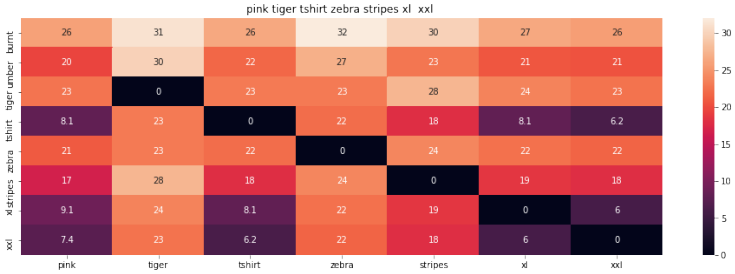
```
ASIN : B01L7ROZNC
Brand : Bila
euclidean distance from input : 9.412861825629273
================================================================================
================================================================================
```

6.2 More weight to idf feature

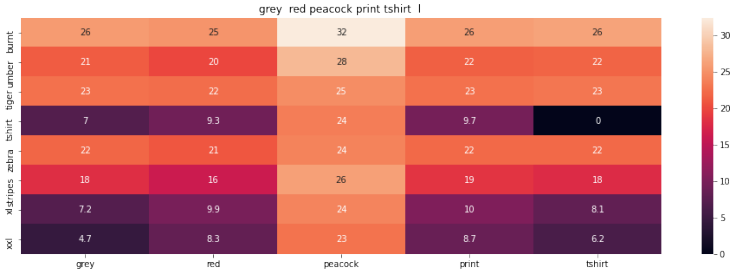
[81]:
```
idf_w2v_brand_with_image(12566,0.05,5,5,50,0.1, 10)
```



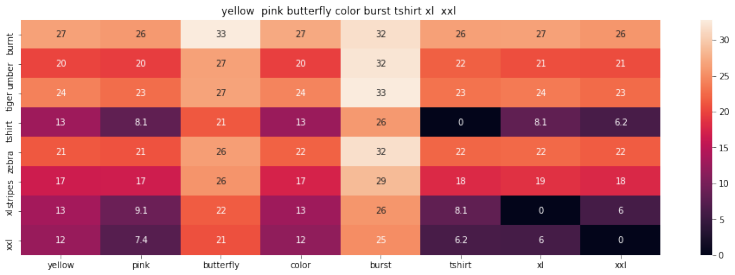
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 1.2484187907146903e-08
==========
==========



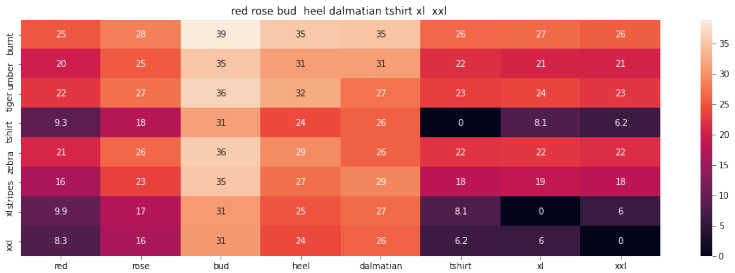
ASIN : B00JXQASS6
Brand : Si Row
euclidean distance from input : 0.6879747300674339
==========
==========



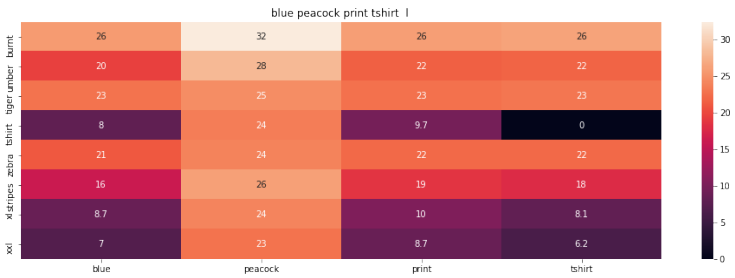
ASIN : B00JXQCFRS
Brand : Si Row
euclidean distance from input : 1.0966084624073198
==========
==========



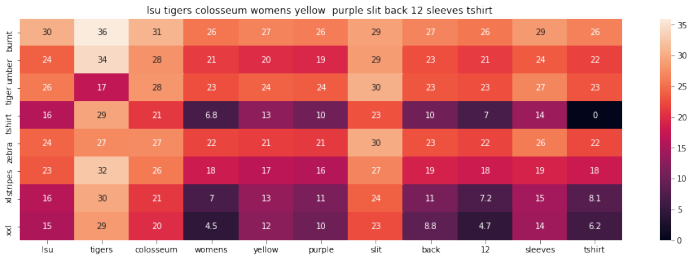
ASIN : B00JXQBBMI
Brand : Si Row
euclidean distance from input : 1.1443977879470268
==========
==========



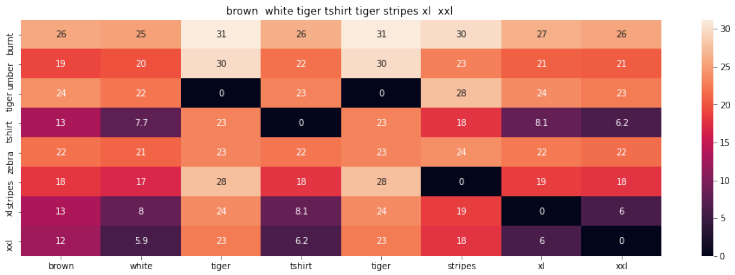
ASIN : B00JXQABB0
Brand : Si Row
euclidean distance from input : 1.1489822898524602
==========
==========



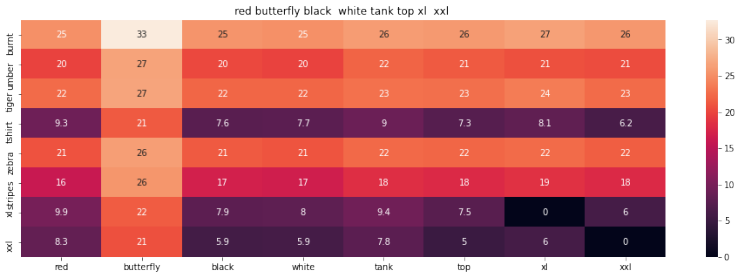
ASIN : B00JXQC8L6
Brand : Si Row
euclidean distance from input : 1.1686656362401127
==========
==========

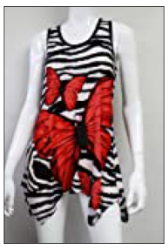


ASIN : B073R5Q8HD
Brand : Colosseum
euclidean distance from input : 1.169234722913854
==========
==========

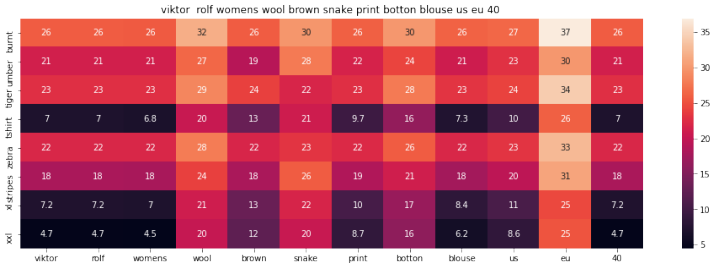


ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 1.1777684377415023
==========
==========





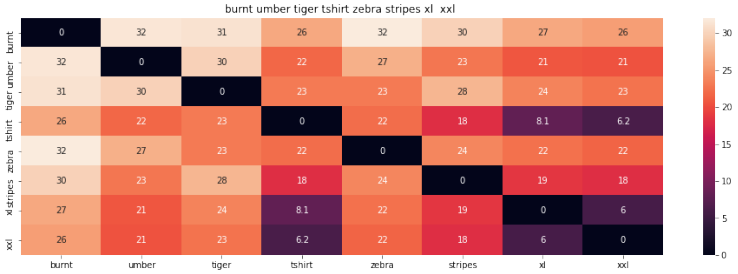
ASIN : B00JV63CW2
Brand : Si Row
euclidean distance from input : 1.1928709843809908
==========
==========



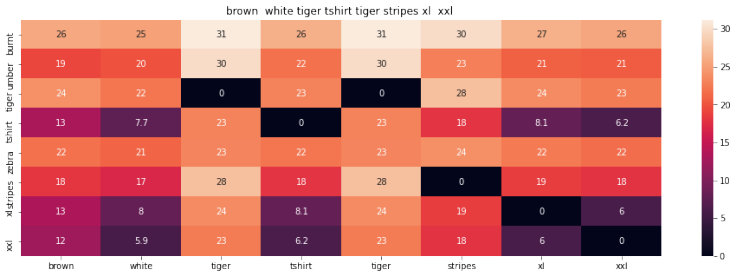
ASIN : B00LEHNVZ4
Brand : Viktor & Rolf
euclidean distance from input : 1.2069615121001855
==========
==========

6.3 More weight to (Brand + Color + type) vectorization

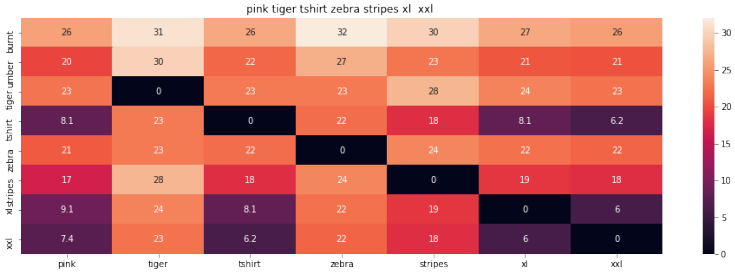
[79]:
```
idf_w2v_brand_with_image(12566,50,50,50,5,5, 10)
```



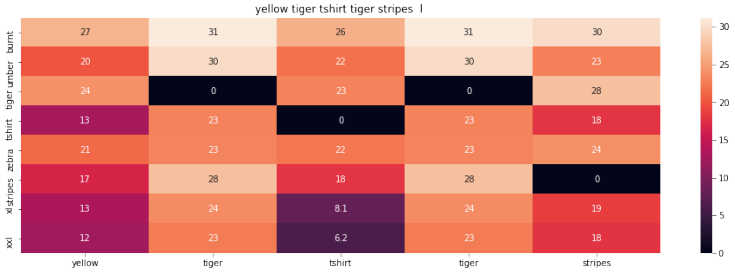
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 2.3466370748792543e-07
==========
==========



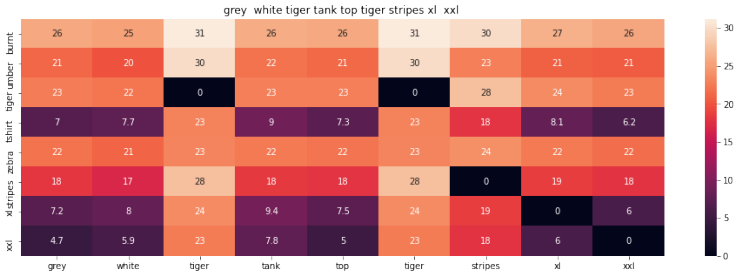
ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 1.7766751646995544
==========
==========



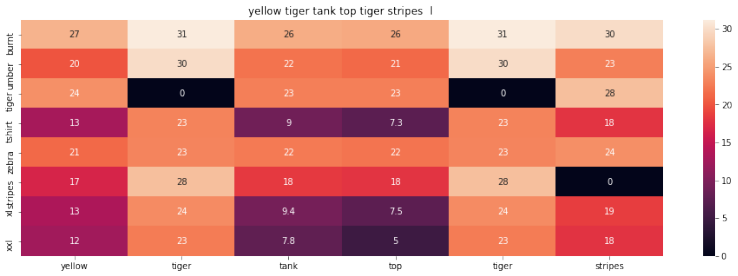
ASIN : B00JXQASS6
Brand : Si Row
euclidean distance from input : 1.9762856141862066
==========
==========



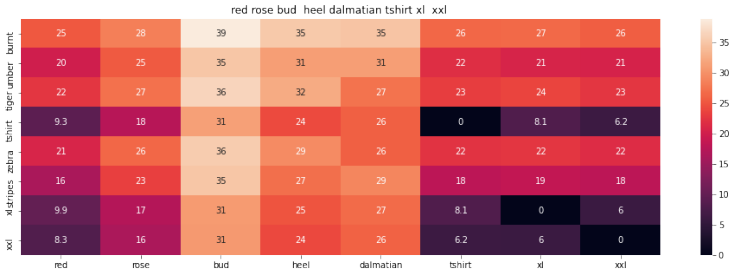
ASIN : B00JXQCUIC
Brand : Si Row
euclidean distance from input : 2.130366098993698
==========
==========



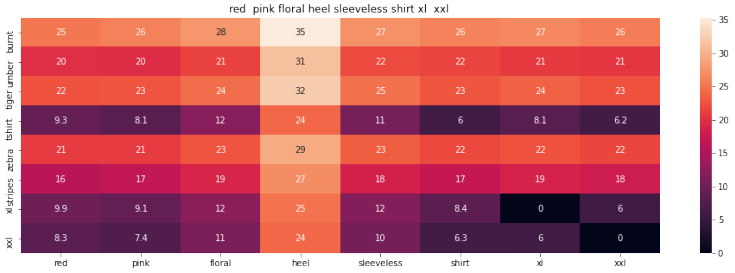
ASIN : B00JXQAFZ2
Brand : Si Row
euclidean distance from input : 2.1660389662918242
==========
==========



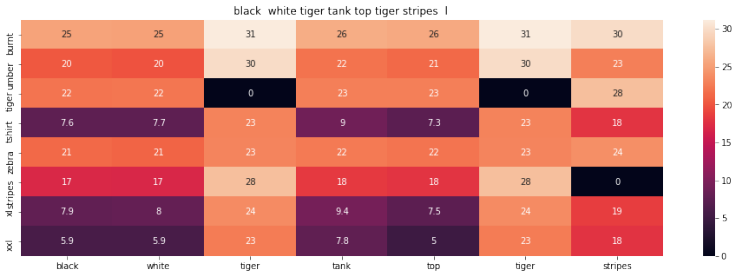
ASIN : B00JXQAUWA
Brand : Si Row
euclidean distance from input : 2.1670491875347286
==========
==========



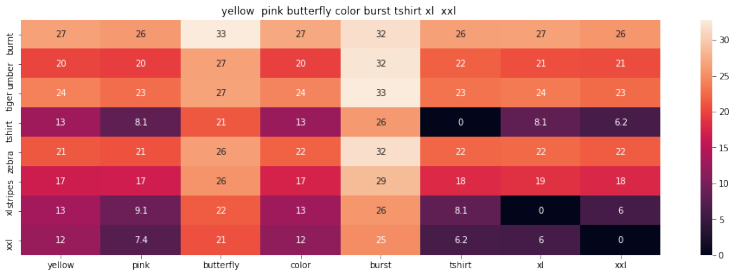
ASIN : B00JXQABB0
Brand : Si Row
euclidean distance from input : 2.1698470265802534
==========
==========



ASIN : B00JV63QQE
Brand : Si Row
euclidean distance from input : 2.2168362857040558
==========
==========



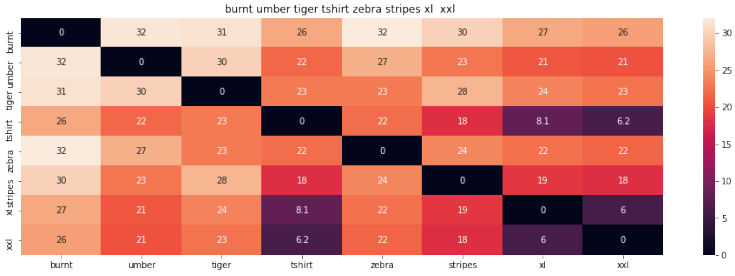
```
ASIN : B00JXQAO94
Brand : Si Row
euclidean distance from input : 2.217535549515167
================================================================================
================================================================================
```



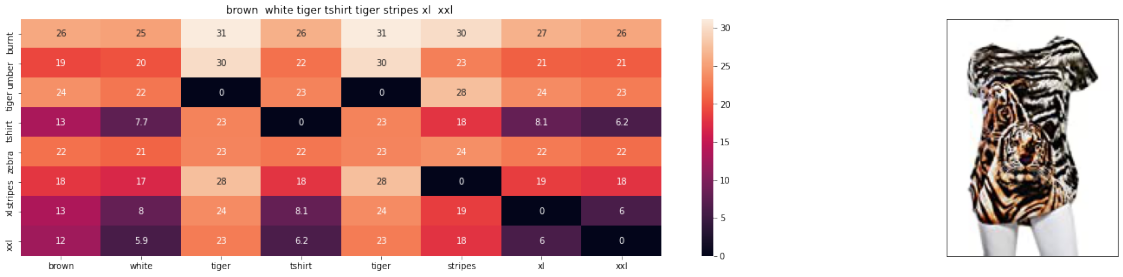
```
ASIN : B00JXQBBMI
Brand : Si Row
euclidean distance from input : 2.231060791128555
================================================================================
================================================================================
```

6.4 More weight to CNN Features

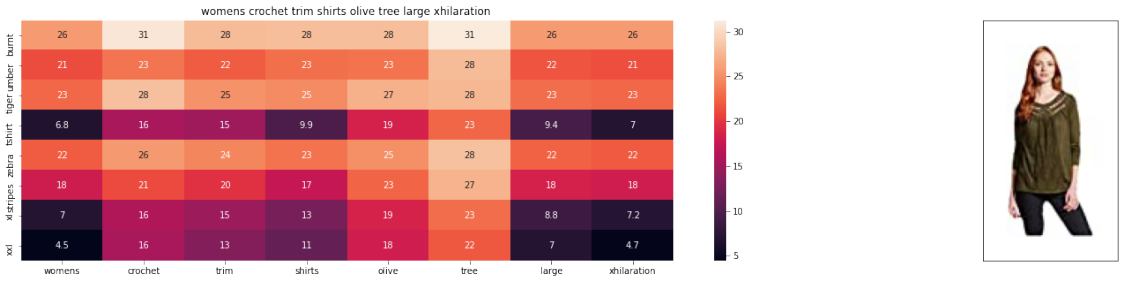
[74]:
```
idf_w2v_brand_with_image(12566,5,5,5,5,50, 10)
```



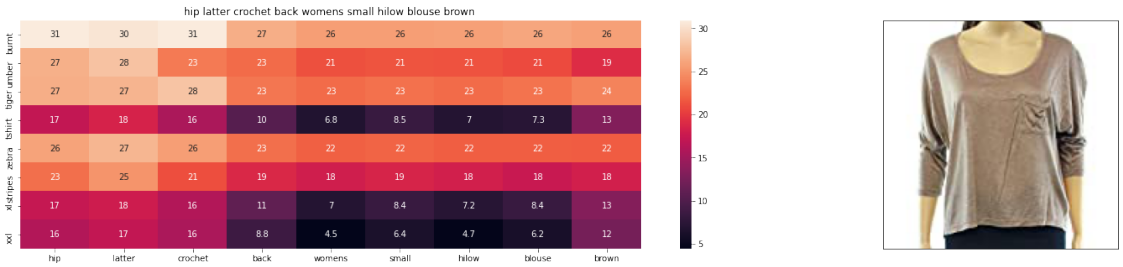
```
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 5.363741989380547e-06
================================================================================
================================================================================
```



ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 3.877291005452474
==========
==========



ASIN : B06XBHNM7J
Brand : Xhilaration
euclidean distance from input : 3.9893853358912708
==========
==========

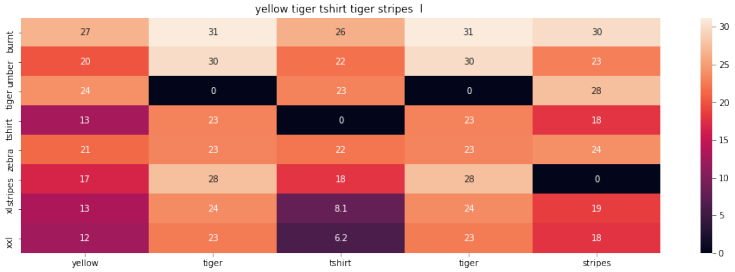


ASIN : B074MJN1K9
Brand : Hip

euclidean distance from input : 4.084150361323262

==========

==========



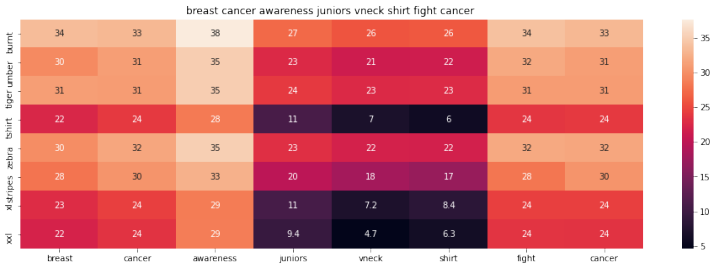
ASIN : B00JXQCUIC

Brand : Si Row

euclidean distance from input : 4.1025966899563935

==========

==========



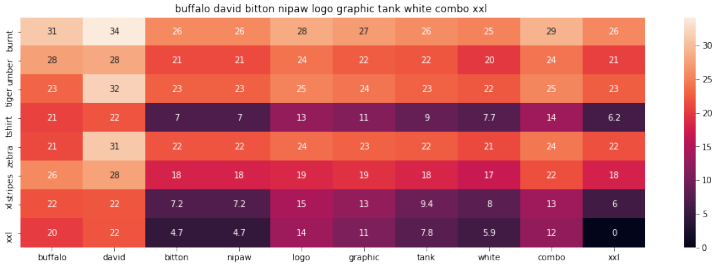
ASIN : B016CU40IY

Brand : Juiceclouds

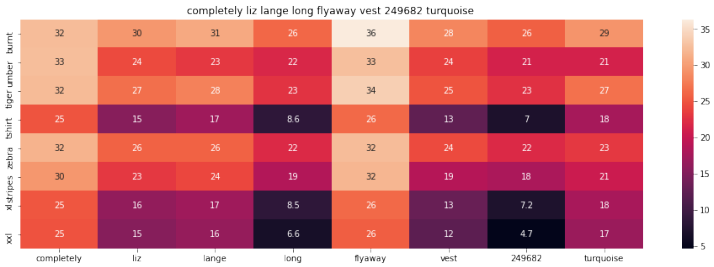
euclidean distance from input : 4.109630466501823

==========

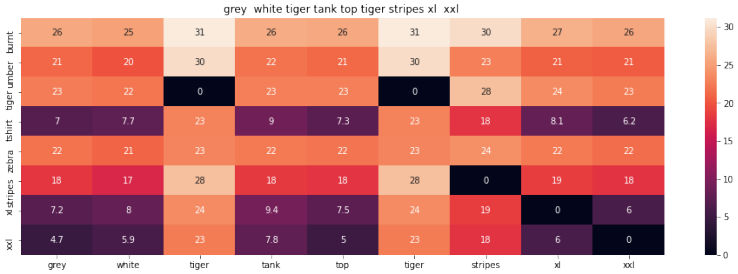
==========



```
ASIN : B018H5AZXQ
Brand : Buffalo
euclidean distance from input : 4.1154988205917915
================================================================================
================================================================================
```



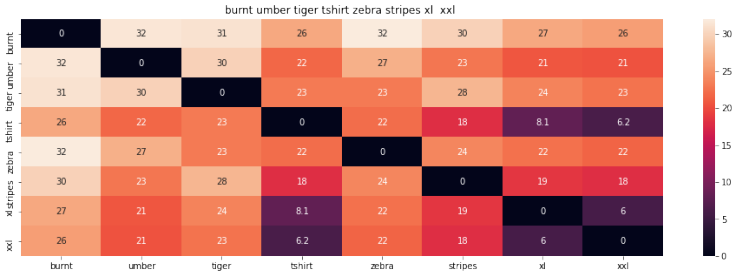
```
ASIN : B074LTBWSW
Brand : Liz Lange
euclidean distance from input : 4.130522308518252
================================================================================
================================================================================
```



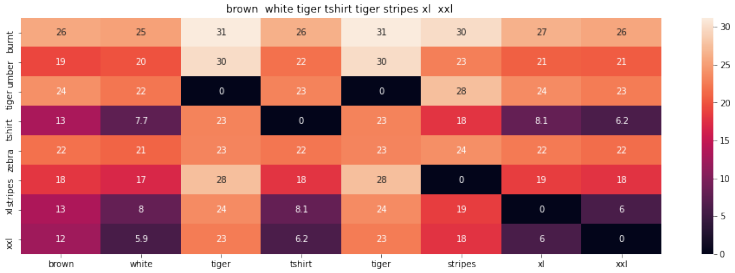
```
ASIN : B00JXQAFZ2
Brand : Si Row
euclidean distance from input : 4.1381643677721485
================================================================================
================================================================================
```

7 More weightage to Color

[92]:
```
idf_w2v_brand_with_image(12566,0.1,100,2,1,0.01, 10)
```

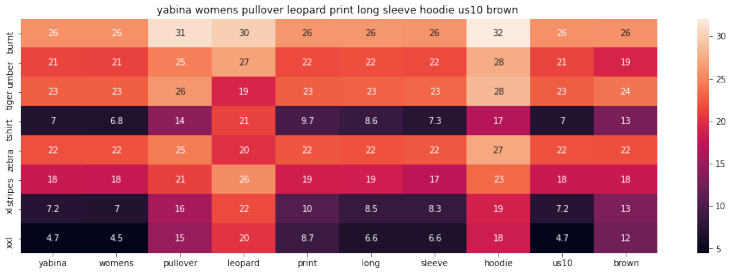


ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 7.282745361198495e-10
==========
==========

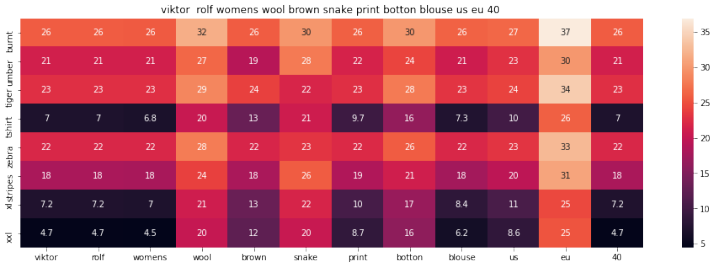


ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 0.01805099642319541
==========
==========

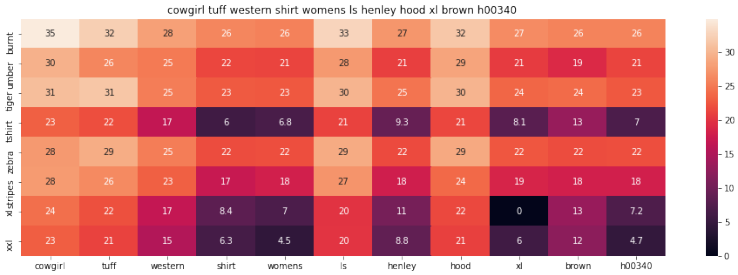
ASIN : B072BVB47Z
Brand : H By Bordeaux
euclidean distance from input : 0.04595411690469613
==========
==========



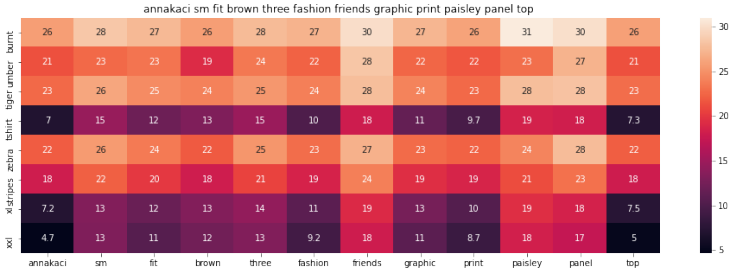
ASIN : B01KJUM6JI
Brand : YABINA
euclidean distance from input : 0.0460320092396074
==========
==========



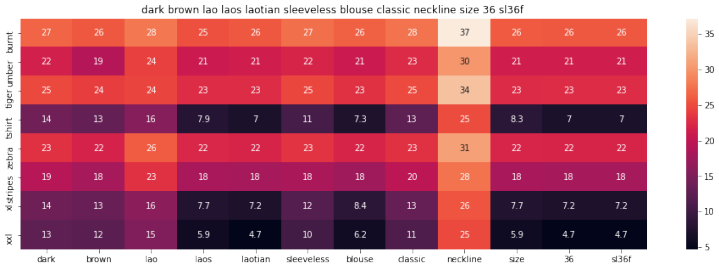
ASIN : B00LEHNVZ4
Brand : Viktor & Rolf
euclidean distance from input : 0.046208431211874824
==========
==========



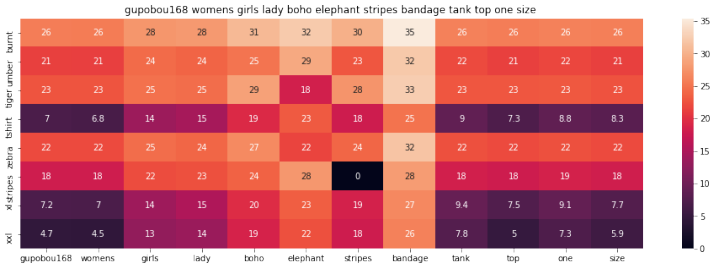
ASIN : B00S6W3MMW
Brand : Cowgirl Tuff
euclidean distance from input : 0.04646422248411415
==========
==========



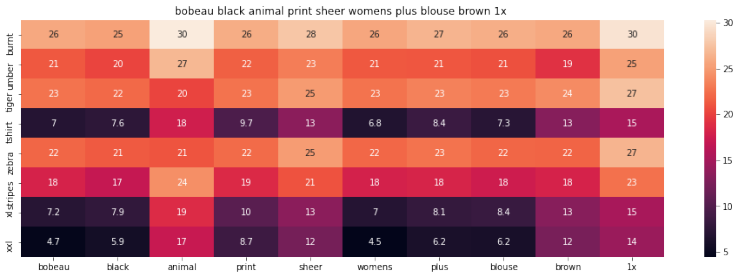
ASIN : B00BTJKAQ0
Brand : Anna-Kaci
euclidean distance from input : 0.046570370315526056
==========
==========



ASIN : B074J7BCYM
Brand : Nanon
euclidean distance from input : 0.04666611918692372
==========
==========



```
ASIN : B01ER18406
Brand : GuPoBoU168
euclidean distance from input : 0.046685624374820246
================================================================================
================================================================================
```



```
ASIN : B074MH886R
Brand : Bobeau
euclidean distance from input : 0.046773509865646906
================================================================================
================================================================================
```

8 WorkFlow

In this case study we try to build a fashion recommendation system with content based design approach. For every product we are given Title, Brand name, Color and image of product,which we leverage build features. Title featurization is done using idf based vectorization, Brand and color are featurized using Bag of words and Images are featurized using VGG16 model's layers without softmax(removing the last layer and keeping the logit as feature). These featurizations are stacked on top of each other to form a vector, and individual weight assignments can be done to stack of features to see which featurization or combination of what all features produce a good result.

[93]:
```
from prettytable import PrettyTable
x = PrettyTable()
```

```
x.field_names = ["Weight Scheme", "Remarks"]
x.add_row(["Equal weights", " This scheme of weights does not produce␣
 ↪satisfactory results as CNN features \n being the longest vector overshadows␣
 ↪others and recommendations suffer. \nVGG16 may be a good edge and pattern␣
 ↪detector it did not take care of color and/or semantics."])
x.add_row(["-----------------------------------------------",␣
 ↪"-------------------------------------------------------------------------------
x.add_row(["More weight to IDF featurization", "Mostly recommendations form␣
 ↪similar brands but relevant recommendations"])
x.add_row(["More weights to Brand+Type+Colour vectorization", "Mostly relevent␣
 ↪recommendations but no recommendations from other brands."])
x.add_row(["More weight to CNN featurizaiton", "no satisfactory results. VGG16␣
 ↪may be a good edge and pattern detector it did not take care of color and/or␣
 ↪semantics."])
x.add_row(["Proportional weights", "Recommendations similar from same brands␣
 ↪but irrelevent recommendations form different brands"])
x.add_row(["More weightage to Color", "Recommendations are pretty relevent in␣
 ↪nature"])

print(x)
```

```
+-----------------------------------------------+-----------------------------
-------------------------------------------------------------------------------
-----------+
|                  Weight Scheme                |
Remarks                                                       |
+-----------------------------------------------+-----------------------------
-------------------------------------------------------------------------------
-----------+
|                  Equal weights                |                        This
scheme of weights does not produce satisfactory results as CNN features
|
|                                               |                          being
the longest vector overshadows others and recommendations suffer.
|
|                                               |             VGG16 may be a
good edge and pattern detector it did not take care of color and/or semantics.
|
| ----------------------------------------------- |            -----------------
--------------------------------------------------------------------------------
|
|         More weight to IDF featurization      |
Mostly recommendations form similar brands but relevant recommendations
|
| More weights to Brand+Type+Colour vectorization |                       Mostly
relevent recommendations but no recommendations from other brands.
|
```

More weight to CNN featurizaiton	no satisfactory results. VGG16 may be a good edge and pattern detector it did not take care of color and/or semantics.
Proportional weights	Recommendations similar from same brands but irrelevent recommendations form different brands
More weightage to Color	Recommendations are pretty relevent in nature