Amazon Apparel Recommendations

[4.2] Data and Code:

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

Overview of the data

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

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

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

Text based product similarity

In [2]:
```
# reloading 16k_apperal_data_preprocessed
# data = pd.read_pickle('pickels/16k_apperal_data_preprocessed')
data = pd.read_pickle('16k_apperal_data_preprocessed')
data.head()
```

Out[2]:

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

Utility Functions

In [3]:
```
# 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 occurenc
e of the word keys(i)
        # labels: len(labels) == len(keys), the values of labels depends on th
e 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 contri
bute 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 sho
w 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 mo
del 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_vectori
zer.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 th
e corpus
            # idf_title_features[doc_id, index_of_word_in_corpus] will give th
e 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 = tex
t.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)
```

In [4]:
```
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 dim
ensions #data_points * #words_in_corpus
# idf_title_features[doc_id, index_of_word_in_corpus] = number of times the wo
rd occured in that doc
```

In [5]:
```
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)))
```

Text + Brand + Color + Image based product similarity

1. Text is IDF-W2V
2. Brand - One-hot encoding
3. Color - One-hot encoding
4. Image - VGG16

Word2Vec:-

In [6]:
```
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.bi
n', binary=True)
'''

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

In [7]:
```
# 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 id
f(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.voc
abulary_[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 c
orpus, 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/a
vg) 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 vect
ors in vec2
    for i in vec1:
        dist = []
        for j in vec2:
            # np.linalg.norm(i-j) will result the euclidean distance between v
ectors 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 t
itle2)
    # 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(we
ighted/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(we
ighted/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 t
itle2)
    # 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 disp
lays 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()
```

In [8]:
```
# 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 o
f 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 id
f(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.vocabula
ry_:
                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
```

IDF weighted Word2Vec

In [9]:
```
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)
```

Weighted similarity using brand and color

In [10]:
```
# Weighted similarity using brand and color
# 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]

# Converting brand and color into one-hot encoding using Count vectorizer
brand_vectorizer = CountVectorizer()
brand_features = brand_vectorizer.fit_transform(brands)

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

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

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

In [11]:
```
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(we
ighted/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(we
ighted/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 t
itle2)
    # 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], typ
es[doc_id1]], # input apparel's features
                [data['asin'].loc[df_id2],brands[doc_id2], colors[doc_id2], typ
es[doc_id2]]] # recommonded apparel's features

    colorscale = [[0, '#1d004d'],[.5, '#f2e5ff'],[1, '#f2e5d1']] # to color th
e 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()
```

Loading image data

In [13]:
```
#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('16k_apperal_data_preprocessed')
df_asins = list(data['asin'])
```

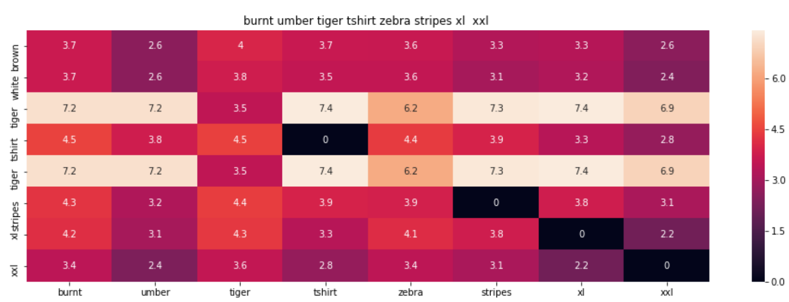
Code with IDF-W2V features + Brand + Color + Image features

In [14]:
```
def idf_w2v_brand(doc_id, w1, w2, w3, num_results):
    # doc_id: apparel's id in given corpus
    # w1: weight for  w2v features
    # w2: weight for brand and color features
    # w3: weight for image features

    # pairwise_dist will store the distance from given input apparel to all re
maining 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
    # computing pair-wise distance using IDF (w2v featurs)
    idf_w2v_dist  = pairwise_distances(w2v_title_weight, w2v_title_weight[doc_
id].reshape(1,-1))
    # computing pair-wise distance using extra features(brand, color)
    ex_feat_dist = pairwise_distances(extra_features, extra_features[doc_id])
    # computing pair-wise distance for image features
    doc_id = asins.index(df_asins[doc_id])
    img_feat_dist = pairwise_distances(bottleneck_features_train, bottleneck_f
eatures_train[doc_id].reshape(1,-1))

    # Computing euclidean distance and multiplying with weights
    # weight1(w1) is for w2v features(idf_w2v_dist)  and w2 is for extra featu
res(ex_feat_dist)
    # Here euclidean distance is computed and is multiplied with weights.
    pairwise_dist   = (w1 * idf_w2v_dist +  w2 * ex_feat_dist + w3 * img_feat_
dist)/float(w1 + w2 + w3)

    # sorting the weighted distances
    # 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], indic
es[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)

# first '5' is weight to Title, next '5' is weight to brand and color and next
'5' is weight of images.
# All are given same weights.
idf_w2v_brand(12566, 5, 5, 5, 20)
# in the give heat map, each cell contains the euclidean distance between word
s i, j
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown

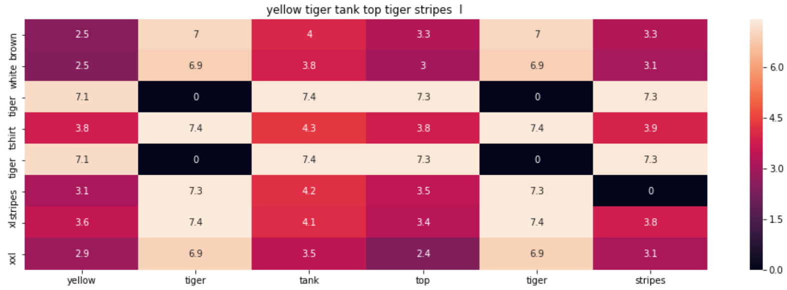
```
ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 0.0
=============================================================================
================================================
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown



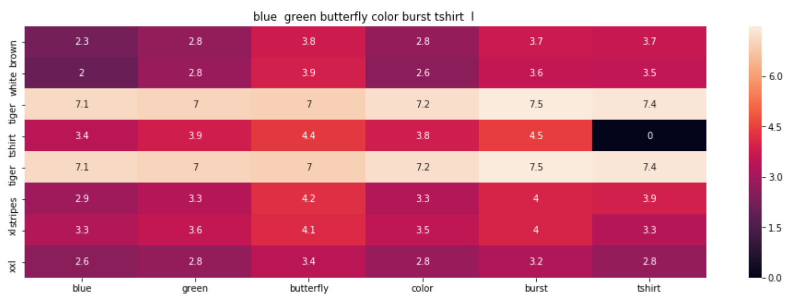
```
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 0.0
=============================================================================
================================================
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown



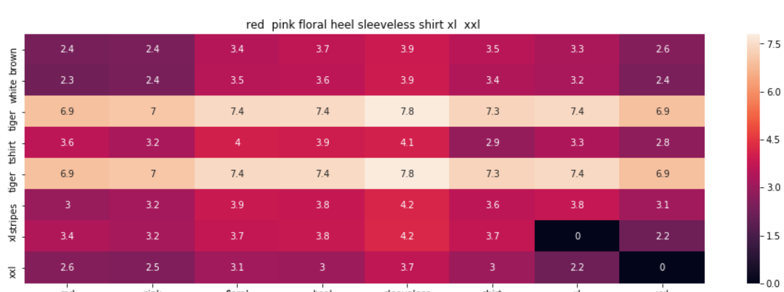
```
ASIN : B00JXQAUWA
Brand : Si Row
euclidean distance from input : 0.4714045207910317
=============================================================================
================================================
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown



```
ASIN : B00JXQC0C8
Brand : Si Row
euclidean distance from input : 0.4714045207910317
=============================================================================
================================================
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown

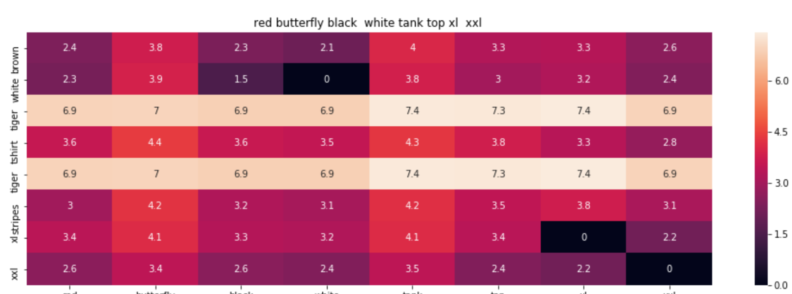


```
ASIN : B00JV63QQE
Brand : Si Row
euclidean distance from input : 0.4714045207910317
==============================================================================
================================================
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown

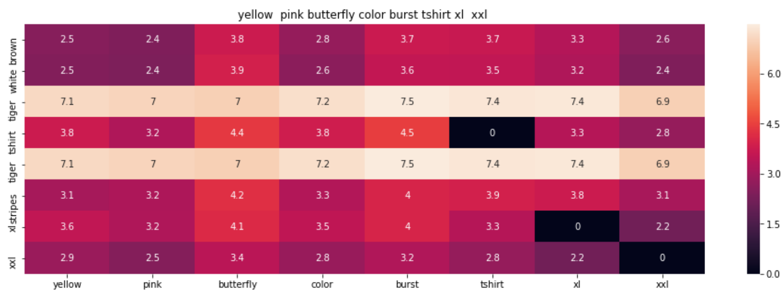
```
ASIN : B00JV63VC8
Brand : Si Row
euclidean distance from input : 0.4714045207910317
==============================================================================
================================================
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown



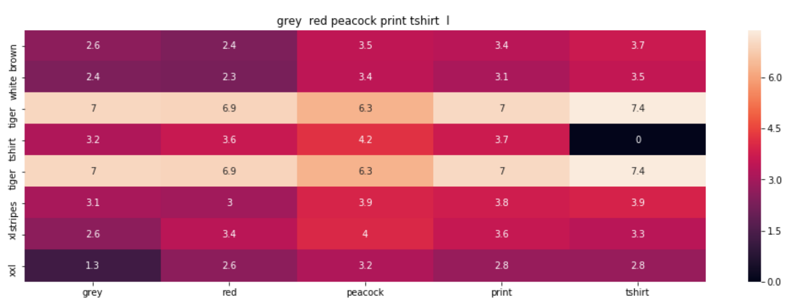
```
ASIN : B00JV63CW2
Brand : Si Row
euclidean distance from input : 0.4714045207910317
==============================================================================
================================================
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown



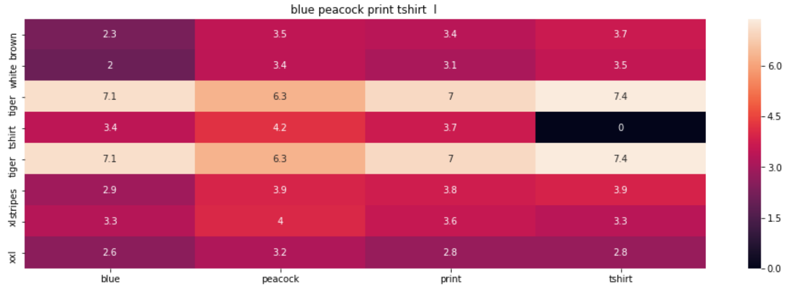
ASIN : B00JXQBBMI
Brand : Si Row
euclidean distance from input : 0.4714045207910317
==

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown



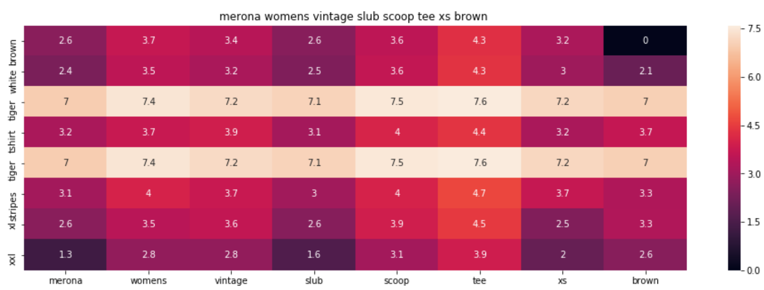
ASIN : B00JXQCFRS
Brand : Si Row
euclidean distance from input : 0.4714045207910317
==

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown



```
ASIN : B00JXQC8L6
Brand : Si Row
euclidean distance from input : 0.4714045207910317
============================================================================================================================================
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown

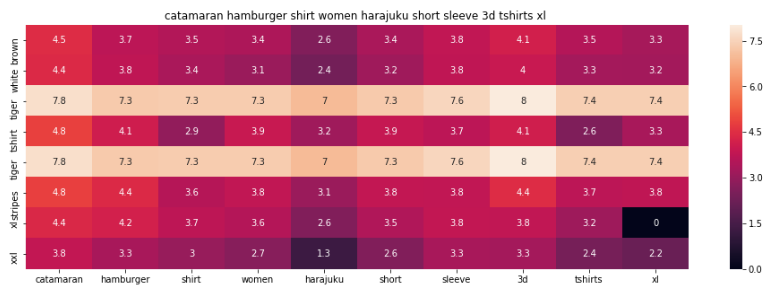


```
ASIN : B01M5KO072
Brand : Merona
euclidean distance from input : 0.7453559924999299
============================================================================================================================================
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown

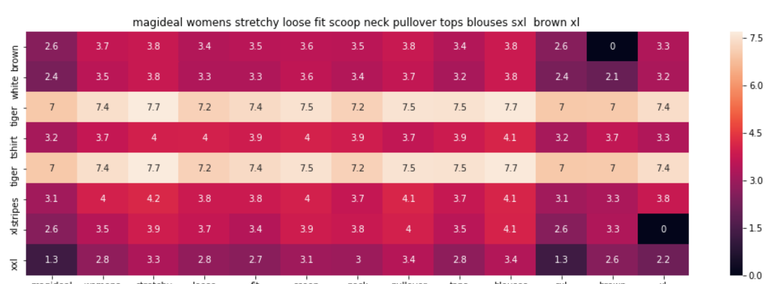
```
ASIN : B00OM4EF60
Brand : Moji
euclidean distance from input : 0.7453559924999299
=============================================================================
================================================
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown



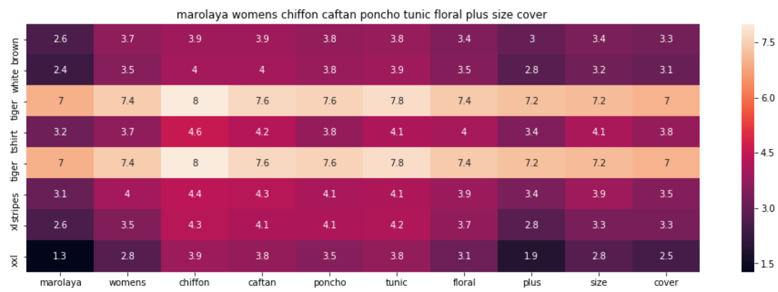
```
ASIN : B01CR325BE
Brand : Catamaran
euclidean distance from input : 0.7453559924999299
=============================================================================
================================================
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown



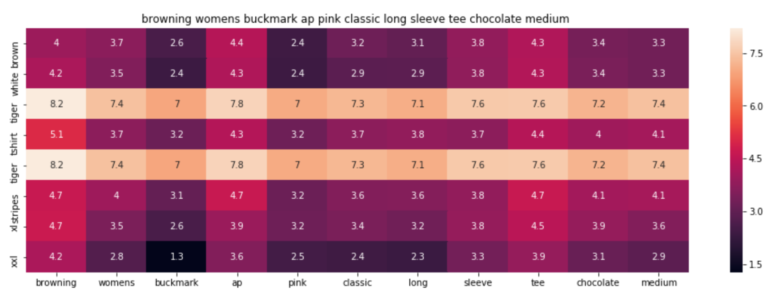
```
ASIN : B07515JFBF
Brand : MagiDeal
euclidean distance from input : 0.7453559924999299
=============================================================================
================================================
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown



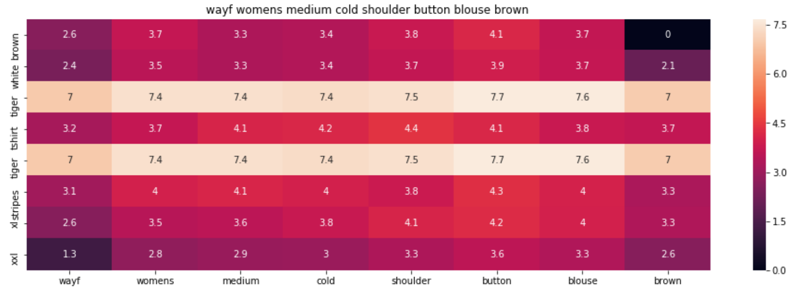
```
ASIN : B01CE40VX0
Brand : Marolaya
euclidean distance from input : 0.7453559924999299
================================================================================
==================================================
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown



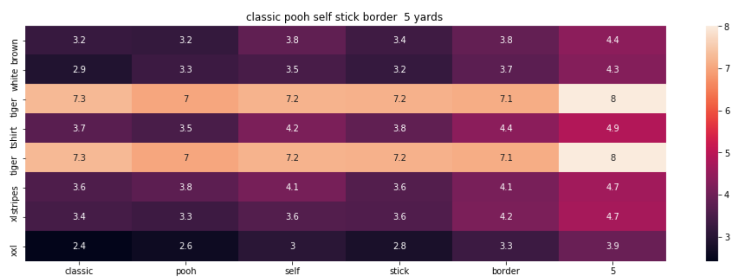
```
ASIN : B00NQFH7MA
Brand : Browning
euclidean distance from input : 0.7453559924999299
================================================================================
==================================================
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown



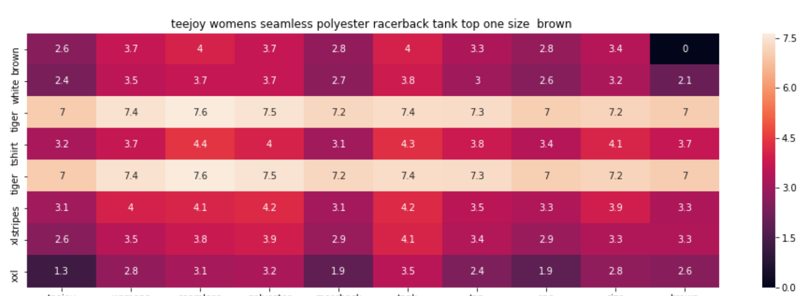
ASIN : B06XDPYJWG
Brand : WAYF
euclidean distance from input : 0.7453559924999299
===

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown



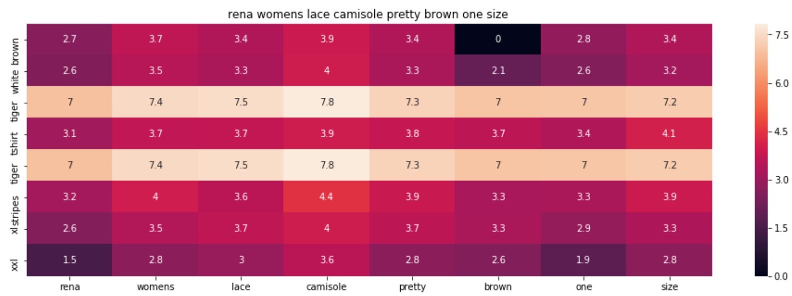
ASIN : B001PO9BNW
Brand : Merona
euclidean distance from input : 0.7453559924999299
===

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown



```
ASIN : B00JPOYCCO
Brand : Sofra
euclidean distance from input : 0.7453559924999299
=============================================================================
================================================
```

Asin	Brand	Color
B00JXQCWTO	Si-Row	Brown



```
ASIN : B01CKFJ1ES
Brand : M. Rena
euclidean distance from input : 0.7453559924999299
=============================================================================
================================================
```

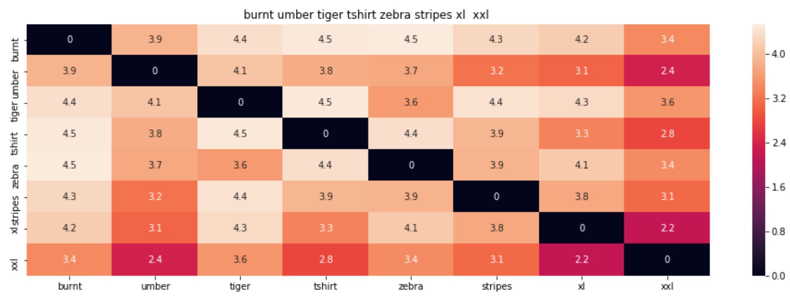
Observation :-

1. For above output, I had given Title weight = 5 , Brand and Color weight = 5 , Image weight = 5 . All are given same weights.
2. In the above output , model gave most of the products of similar brand , color and products similar to the input image.
3. For some products , if the brand is same then color changed and if brand changed then color remained same.
4. Almost all the images matched with the original image either in Brand, color, images or dress shapes.

Changing the weights

In [15]:
```
def idf_w2v_brand(doc_id, w1, w2, w3, num_results):
    # doc_id: apparel's id in given corpus
    # w1: weight for  w2v features
    # w2: weight for brand and color features
    # w3: weight for image features

    # pairwise_dist will store the distance from given input apparel to all re
maining 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
    # computing pair-wise distance using IDF (w2v featurs)
    idf_w2v_dist  = pairwise_distances(w2v_title_weight, w2v_title_weight[doc_
id].reshape(1,-1))
    # computing pair-wise distance using extra features(brand, color)
    ex_feat_dist = pairwise_distances(extra_features, extra_features[doc_id])
    # computing pair-wise distance for image features
    doc_id = asins.index(df_asins[doc_id])
    img_feat_dist = pairwise_distances(bottleneck_features_train, bottleneck_f
eatures_train[doc_id].reshape(1,-1))

    # Computing euclidean distance and multiplying with weights
    # weight1(w1) is for w2v features(idf_w2v_dist)  and w2 is for extra featu
res(ex_feat_dist)
    # Here euclidean distance is computed and is multiplied with weights.
    pairwise_dist   = (w1 * idf_w2v_dist +  w2 * ex_feat_dist + w3 * img_feat_
dist)/float(w1 + w2 + w3)

    # sorting the weighted distances
    # 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], indic
es[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)

# first '5' is weight to Title, next '30' is weight to brand and color and nex
t '50' is weight of images and getting 20 images
# All are given same weights.
idf_w2v_brand(12566, 5, 30, 50, 20)
# in the give heat map, each cell contains the euclidean distance between word
s i, j
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

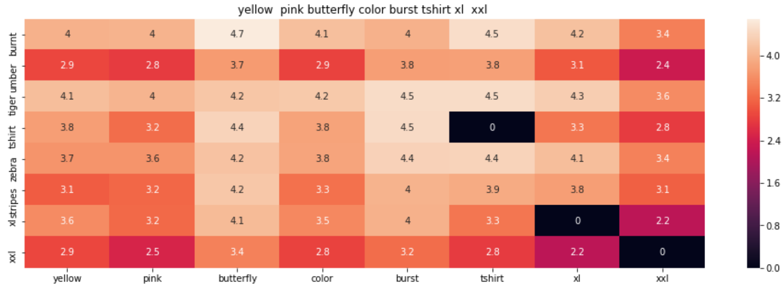


```
ASIN : B00JXQB5FQ
Brand : Si Row
euclidean distance from input : 0.0
============================================================================================================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

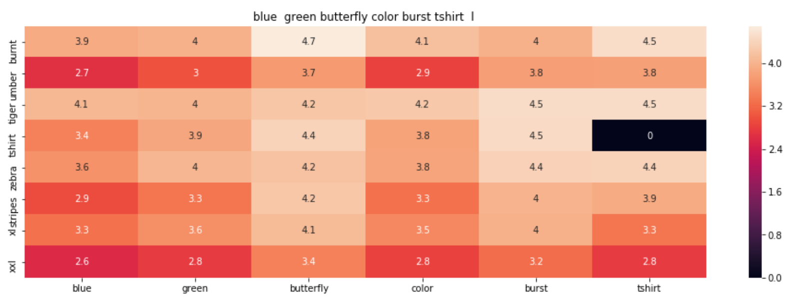
```
ASIN : B00JXQCWTO
Brand : Si Row
euclidean distance from input : 0.0
============================================================================================================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



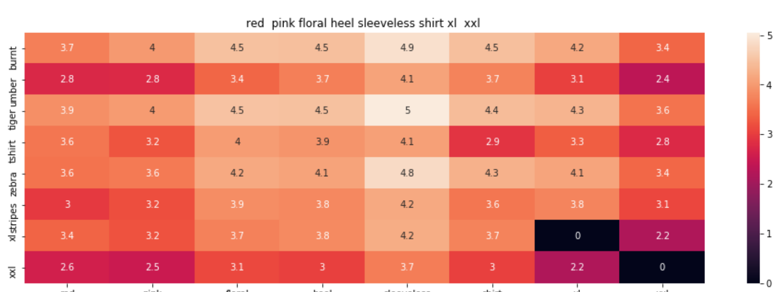
```
ASIN : B00JXQBBMI
Brand : Si Row
euclidean distance from input : 0.4991341984846218
============================================================================
===============================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



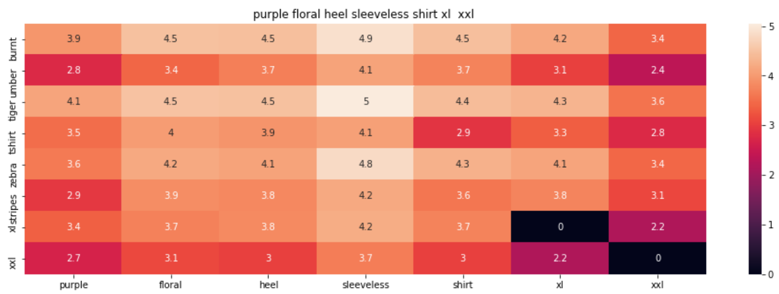
```
ASIN : B00JXQC0C8
Brand : Si Row
euclidean distance from input : 0.4991341984846218
============================================================================
===============================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



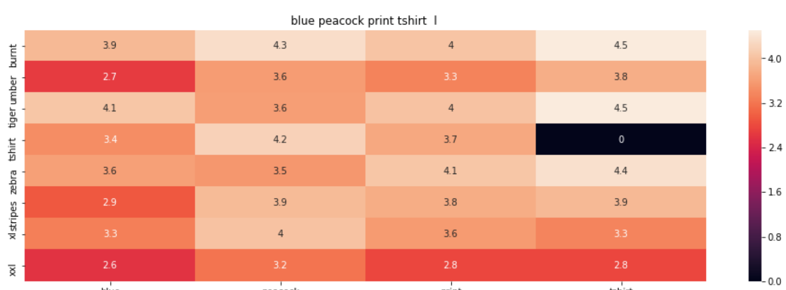
```
ASIN : B00JV63QQE
Brand : Si Row
euclidean distance from input : 0.4991341984846218
==============================================================================
================================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



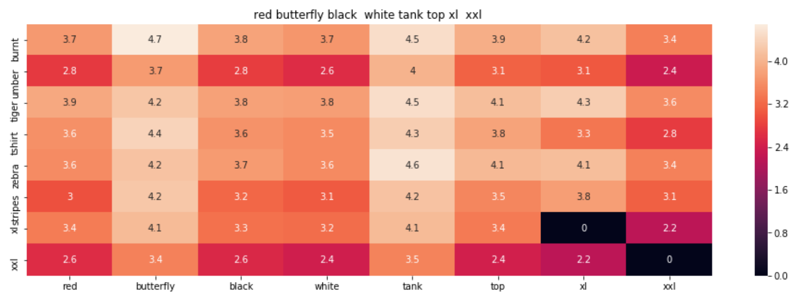
```
ASIN : B00JV63VC8
Brand : Si Row
euclidean distance from input : 0.4991341984846218
==============================================================================
================================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



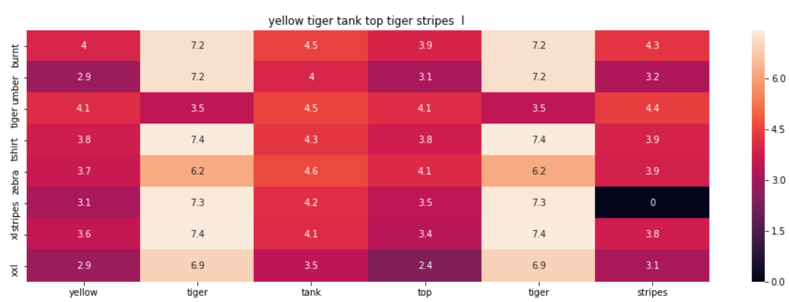
```
ASIN : B00JXQC8L6
Brand : Si Row
euclidean distance from input : 0.4991341984846218
==============================================================================
================================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



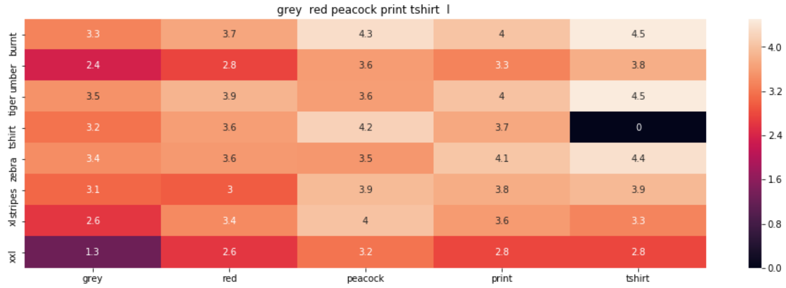
```
ASIN : B00JV63CW2
Brand : Si Row
euclidean distance from input : 0.4991341984846218
=============================================================================
================================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



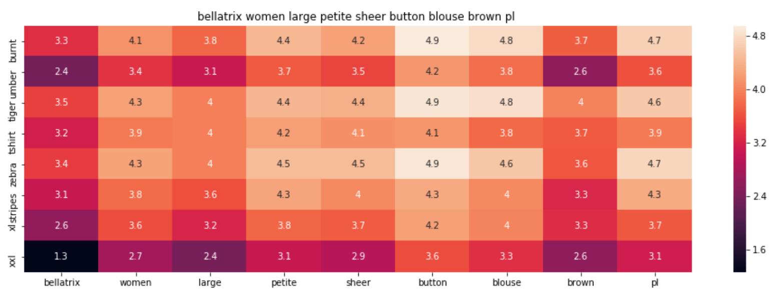
```
ASIN : B00JXQAUWA
Brand : Si Row
euclidean distance from input : 0.4991341984846218
=============================================================================
================================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



ASIN : B00JXQCFRS
Brand : Si Row
euclidean distance from input : 0.4991341984846218
===
==

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

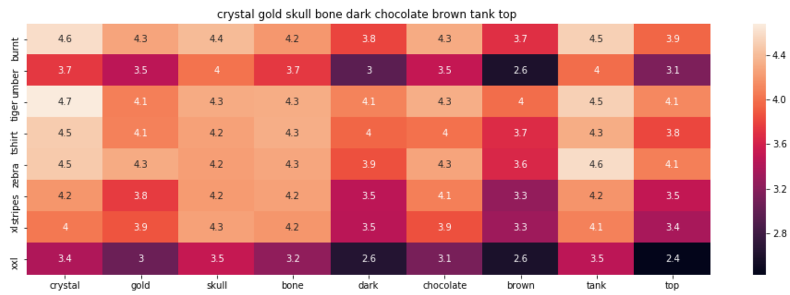


ASIN : B074QVMXSQ
Brand : bellatrix
euclidean distance from input : 0.7892004626469847
===
==

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown

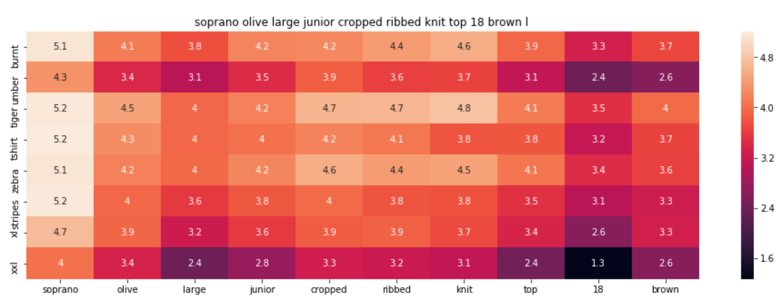
```
ASIN : B00OM4EF60
Brand : Moji
euclidean distance from input : 0.7892004626469847
================================================================================
==================================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



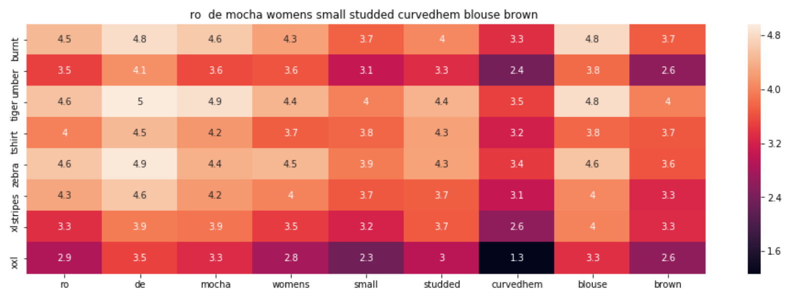
```
ASIN : B004YR55O6
Brand : stonepowerss
euclidean distance from input : 0.7892004626469847
================================================================================
==================================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



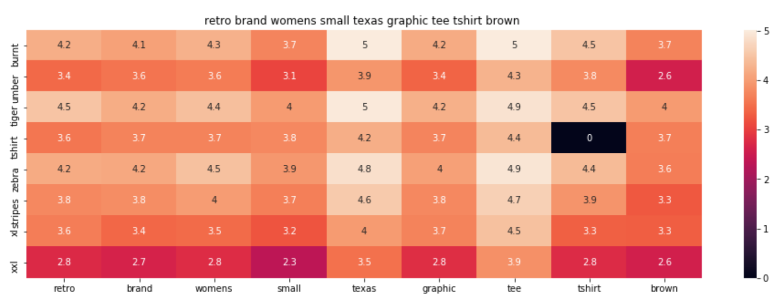
```
ASIN : B07288KFHF
Brand : Soprano
euclidean distance from input : 0.7892004626469847
================================================================================
==================================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



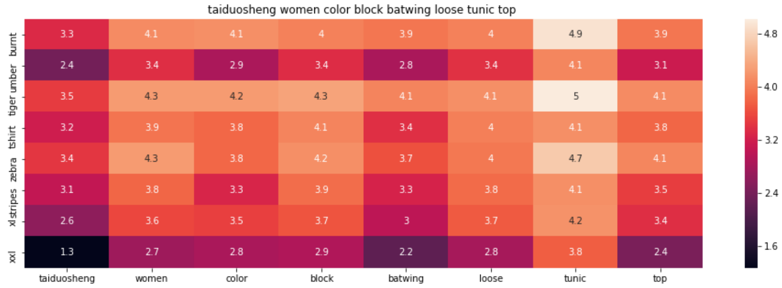
```
ASIN : B01M35MN7L
Brand : Rode
euclidean distance from input : 0.7892004626469847
===============================================================================
=================================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



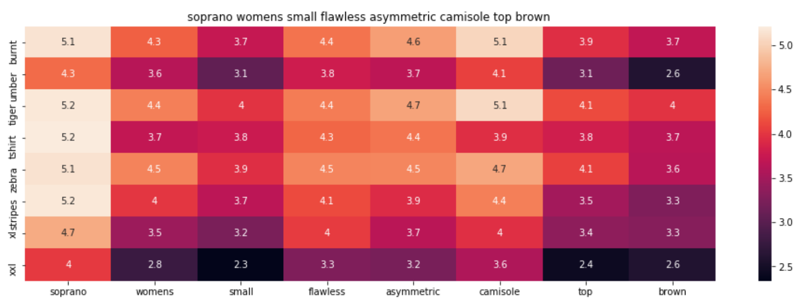
```
ASIN : B072Y1RTRY
Brand : Retro
euclidean distance from input : 0.7892004626469847
===============================================================================
=================================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



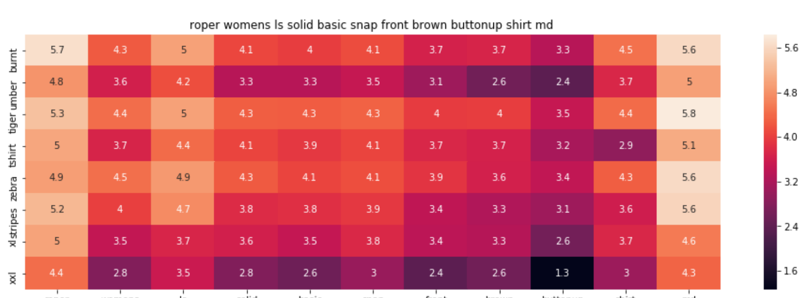
```
ASIN : B01L8TH6B2
Brand : Taiduosheng
euclidean distance from input : 0.7892004626469847
=============================================================================
================================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



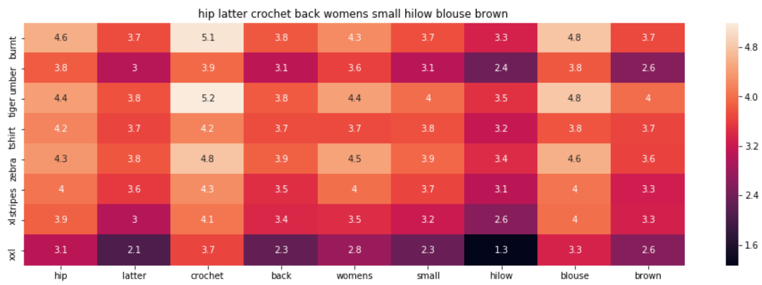
```
ASIN : B0758356K3
Brand : Soprano
euclidean distance from input : 0.7892004626469847
=============================================================================
================================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



```
ASIN : B00JLQAOZ0
Brand : Roper
euclidean distance from input : 0.7892004626469847
=============================================================================
================================================
```

Asin	Brand	Color
B00JXQB5FQ	Si-Row	Brown



```
ASIN : B074MJN1K9
Brand : Hip
euclidean distance from input : 0.7892004626469847
=============================================================================
================================================
```

Observations :-

1. In this assignment I took IDF-W2V based Text, Brand, Color, Images information, to recommend similar products.
2. All 4 features are given weights and built weighted Eucledian distance based similarity using those 4 features.
3. Model performed well by recommending the similar products of same brand , same color and similar images to original image.
4. Even if the recommended product image did not match original image exactly, model tried to show products of same brand or similar color images which have same color as the original image (in above case it is brown color).

In [ ]: