Amazon Apparel Recommendations

[4.2] Data and Code:

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

[4.3] Overview of the data

In [1]:

```
# Plots and visuals:
from PIL import Image
from io import BytesIO
import matplotlib.pyplot as plt
import seaborn as sns
import plotly
import plotly.figure_factory as ff
from plotly.graph_objs import Scatter, Layout
from matplotlib import gridspec

# data objects:
import requests
from bs4 import BeautifulSoup
import numpy as np
from scipy.sparse import hstack
import pandas as pd
from collections import Counter

# Working with text:
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import re
from gensim.models import Word2Vec
from gensim.models import KeyedVectors

# Metrics
from sklearn.metrics.pairwise import cosine_similarity, pairwise_distances

# Mics:
import warnings
import math
import time
import os
from tqdm import tqdm
import pickle

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

```
c:\users\byron\applications\pythonmaster\lib\site-packages\gensim\utils.p
y:1212: UserWarning:

detected Windows; aliasing chunkize to chunkize_serial
```

In [2]:

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

In [3]:

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

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

Terminology:

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

In [4]:

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

Out[4]:

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

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

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

In [5]:

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

In [6]:

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

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

Out[6]:

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

[5.1] Missing data for various features.

Basic stats for the feature: product_type_name

In [7]:

```
# We have total 72 unique type of product_type_names
print(data['product_type_name'].describe())
# 91.62% (167794/183138) of the products are shirts,
```

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

In [8]:

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

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

In [9]:

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

Out[9]:

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

Basic stats for the feature: brand

In [10]:

```
# there are 10577 unique brands
print(data['brand'].describe())
# 183138 - 182987 = 151 missing values.
```

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

In [11]:

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

Out[11]:

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

Basic stats for the feature: color

In [12]:

```
print(data['color'].describe())
# we have 7380 unique colors
# 7.2% of products are black in color
# 64956 of 183138 products have color information. That's approx 35.4%.
```

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

In [13]:

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

Out[13]:

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

Basic stats for the feature: formatted_price

In [14]:

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

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

In [15]:

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

Out[15]:

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

Basic stats for the feature: title

In [16]:

```
print(data['title'].describe())
# All of the products have a title.
# Titles are fairly descriptive of what the product is.
# We use titles extensively in this workshop
# as they are short and informative.
```

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

In [17]:

```
url = data[data['medium_image_url'].duplicated()].loc[27,'medium_image_url']
data.query('medium_image_url == @url')
```

Out[17]:

	asin	brand	color	medium_image_url	product_type_name	title	fc
21	B014ICEDNA	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT	Supernatural Chibis Sam Dean And Castiel Short...	
27	B014ICEJ1Q	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT	Supernatural Chibis Sam Dean And Castiel O Nec...	
2121	B014ICEP24	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT	Supernatural Chibis Sam Dean And Castiel 100% ...	

In [18]:

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

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

In [19]:

```
# 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_sub = data.loc[~data['formatted_price'].isnull()]
print('Number of data points After eliminating price=NULL :', data_sub.shape[0])
```

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

In [20]:

```
# 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_sub =data_sub.loc[~data_sub['color'].isnull()]
print('Number of data points After eliminating color=NULL :', data_sub.shape[0])
```

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

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

In [21]:

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

In [22]:

```
data[data['medium_image_url'].isnull()]
```

Out[22]:

	asin	brand	color	medium_image_url	product_type_name	title	formatted_price

[5.2] Remove near duplicate items

[5.2.1] Understand about duplicates.

In [23]:

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

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

```
2325
```

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

:B00AQ4GMCK

:B00AQ4GMTS

:B00AQ4GMLQ

:B00AQ4GN3I

These shirts exactly same except in color

:B00G278GZ6

:B00G278W6O

:B00G278Z2A

:B00G2786X8

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

[5.2.2] Remove duplicates : Part 1

In [24]:

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

In [25]:

```
data_sub.head()
```

Out[25]:

	asin	brand	color	medium_image_url	product_type_name	title
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT	Featherlite Ladies' Long Sleeve Stain Resistan...
6	B012YX2ZPI	HX-Kingdom Fashion T-shirts	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	Women's Unique 100% Cotton T - Special Olympic...
11	B001LOUGE4	Fitness Etc.	Black	https://images-na.ssl-images-amazon.com/images...	SHIRT	Ladies Cotton Tank 2x1 Ribbed Tank Top
15	B003BSRPB0	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	FeatherLite Ladies' Moisture Free Mesh Sport S...
21	B014ICEDNA	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT	Supernatural Chibis Sam Dean And Castiel Short...

In [26]:

```
# Remove All products with very few words in title
data_sorted = data_sub[data_sub['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
```

In [27]:

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

Out[27]:

	asin	brand	color	medium_image_url	product_type_name	t
61973	B06Y1KZ2WB	Éclair	Black/Pink	https://images-na.ssl-images-amazon.com/images...	SHIRT	Éc Wome Prin Thin St Blo Blac
133820	B010RV33VE	xiaoming	Pink	https://images-na.ssl-images-amazon.com/images...	SHIRT	xiaom Wom Sleevel Lo Long shir
81461	B01DDSDLNS	xiaoming	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	xiaom Wome W L Sle Sir Bre
75995	B00X5LYO9Y	xiaoming	Red Anchors	https://images-na.ssl-images-amazon.com/images...	SHIRT	xiaom Stri T Patch/B Sle Anch
151570	B00WPJG35K	xiaoming	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	xiaom Sle Sh Lo Tas Kim Wom

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

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

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

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

In [28]:

```
indices = list(data_sorted.index.values)
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_sub['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', 'Quee
n', 'of', 'Diamonds', 'Women's', 'Shirt', 'Small']
        b = data_sub['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 appended 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 consider
ing it as those two apperals are different
        # if the number of words in which both strings differ are < 2 , we are consider
ing 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]])

            # if the comaprision between is between num_data_points, num_data_points-1
 strings and they differ in more than 2 words we include both
            if j == num_data_points-1: stage1_dedupe_asins.append(data_sorted['asin'].l
oc[indices[j]])

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

In [29]:

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

We removed the dupliactes which differ only at the end.

In [30]:

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

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

In [31]:

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

[5.2.3] Remove duplicates : Part 2

```
    In the previous cell, we sorted whole data in alphabetical order of  titles.The
    n, we removed titles which are adjacent and very similar title

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

    Examples:

    Titles-1
    86261.  UltraClub Women's Classic Wrinkle-Free Long Sleeve Oxford Shirt, Pink, X
    X-Large
    115042. UltraClub Ladies Classic Wrinkle-Free Long-Sleeve Oxford Light Blue XXL

    TItles-2
    75004.  EVALY Women's Cool University Of UTAH 3/4 Sleeve Raglan Tee
    109225. EVALY Women's Unique University Of UTAH 3/4 Sleeve Raglan Tees
    120832. EVALY Women's New University Of UTAH 3/4-Sleeve Raglan Tshirt
```

In [32]:

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

In [33]:

```
# This code snippet takes significant amount of time.
# O(n^2) time.
# Takes about an hour to run on a decent computer.
if not os.path.exists('pickels/16k_apperal_data'):
    indices = list(data.index.values)

    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

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

6. Text pre-processing

In [34]:

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

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

In [35]:

```
# different titles only else everything is the same:
url = data[data['medium_image_url'].duplicated()].loc[703,'medium_image_url']
data.query('medium_image_url == @url')
```

Out[35]:

	asin	brand	color	medium_image_url	product_type_name	
471	B01M9J1FJC	FARYSAYS	Black	https://images-na.ssl-images-amazon.com/images...	BOOKS_1973_AND_LATER	FARYS Wom Sexy Shou Blous
703	B01M4R97JB	FARYSAYS	Black	https://images-na.ssl-images-amazon.com/images...	BOOKS_1973_AND_LATER	FARYS Wom Sexy (Shou Blc L

In [36]:

```
data.drop_duplicates(subset = ['brand','color','product_type_name','medium_image_url'],
keep = 'first', inplace = True)
```

In [37]:

```
# different titles only else everything is the same:
url = data[data['medium_image_url'].duplicated()].loc[1033,'medium_image_url']
data.query('medium_image_url == @url')
```

Out[37]:

	asin	brand	color	medium_image_url	product_type_name	title	fo
60	B014ICB9A0	FNC7C	Black	https://images-na.ssl-images-amazon.com/images...	APPAREL	Supernatural Chibis Sam Dean And Castiel O Nec...	
1033	B014ICBNQU	FNC7C	Black	https://images-na.ssl-images-amazon.com/images...	SHIRT	Supernatural Chibis Sam Dean And Castiel Round...	

In [38]:

```
data.drop_duplicates(subset = ['brand','color','medium_image_url'],keep = 'first', inpl
ace = True)
```

In [39]:

```
# different titles only else everything is the same:
url = data[data['medium_image_url'].duplicated()].loc[1381,'medium_image_url']
data.query('medium_image_url == @url')
```

Out[39]:

	asin	brand	color	medium_image_url	product_type_name	title
770	B003BSQPX4	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	FeatherLite Ladies Long Sleeve Oxford Shirt, W...
1381	B003BSQPW0	FeatherLite	Blue	https://images-na.ssl-images-amazon.com/images...	SHIRT	FeatherLite Ladies Long Sleeve Oxford Shirt, L...

In [40]:

```
data.drop_duplicates(subset = ['brand','medium_image_url','product_type_name'],keep =
'first', inplace = True)
```

In [41]:

```
# different titles only else everything is the same:
url = data[data['medium_image_url'].duplicated()].loc[8084,'medium_image_url']
data.query('medium_image_url == @url')
```

Out[41]:

	asin	brand	color	medium_image_url	product_type_name	title	form
6339	B071KXR4MJ	Belle du Jour	Multi	https://images-na.ssl-images-amazon.com/images...	SHIRT	Self Esteem Juniors Printed Scarf Tank To Chil...	
8084	B0719GH839	Belle Du Jour Self Esteem	Chili Pepper Navy	https://images-na.ssl-images-amazon.com/images...	APPAREL	Belle du Jour Womens Chiffon Printed Tank Top ...	

In [42]:

```
data.drop_duplicates(subset = ['medium_image_url'],keep = 'first', inplace = True)
```

In [43]:

```
data[data['medium_image_url'].duplicated()]
```

Out[43]:

	asin	brand	color	medium_image_url	product_type_name	title	formatted_price

In [44]:

```
data.shape
```

Out[44]:

```
(15528, 7)
```

In [45]:

```
images_obtained = list()
for file in os.listdir('images'):
    images_obtained.append(file[0:-5])

images_requested = list()
for record in data.iterrows():
    images_requested.append(record[1]['asin'])

S1 = set(images_obtained)
S2 = set(images_requested)
```

In [46]:

```
actual = len(S1.intersection(S2))
expected = data.shape[0]
```

In [47]:

```
def image_download(url):
    filename = data.query('medium_image_url == @url')['asin'].values[0]
    response = requests.get(url)
    img = Image.open(BytesIO(response.content))
    img.save('images/'+str(filename)+'.jpeg')

if expected != actual:
    data['medium_image_url'].apply(image_download)
```

In [48]:

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

In [49]:

```
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")
```

```
4.263814708000005 seconds
```

In [50]:

```
data.shape
```

Out[50]:

```
(15528, 7)
```

In [51]:

```
data.head()
```

Out[51]:

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

In [52]:

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

Stemming

In [53]:

```
from nltk.stem.porter import *
stemmer = PorterStemmer()
print(stemmer.stem('arguing'))
print(stemmer.stem('fishing'))
# We tried using stemming on our titles and it did not work very well.
```

```
argu
fish
```

[8] Text based product similarity

In [54]:

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

Out[54]:

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

In [55]:

```
# 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 t
he distance between two title vec's
    intersection = set(vec1.keys()) & set(vec2.keys())

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

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

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

    if model == 'bag_of_words':
        labels = values
    elif model == 'tfidf':
        labels = []
        for x in vec2.keys():
            # tfidf_title_vectorizer.vocabulary_ it contains all the words in the corpu
s
            # tfidf_title_features[doc_id, index_of_word_in_corpus] will give the tfidf
value of word in given document (doc_id)
            if x in  tfidf_title_vectorizer.vocabulary_:
                labels.append(tfidf_title_features[doc_id, tfidf_title_vectorizer.vocab
ulary_[x]])
            else:
                labels.append(0)
    elif model == 'idf':
        labels = []
        for x in vec2.keys():
            # idf_title_vectorizer.vocabulary_ it contains all the words in the corpus
            # idf_title_features[doc_id, index_of_word_in_corpus] will give the idf val
ue of word in given document (doc_id)
            if x in  idf_title_vectorizer.vocabulary_:
                labels.append(idf_title_features[doc_id, idf_title_vectorizer.vocabular
y_[x]])
            else:
                labels.append(0)

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


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

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

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

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

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

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

In [56]:

```
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
```

Out[56]:

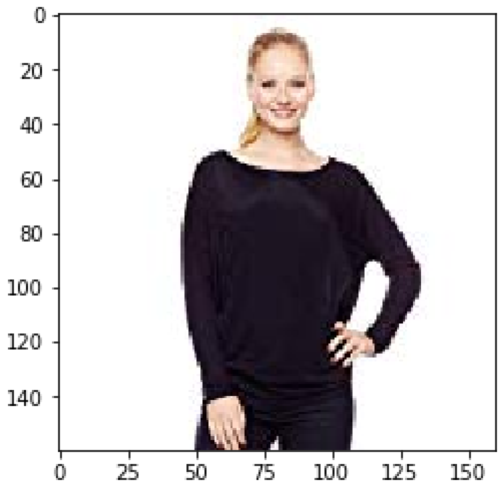
```
(15528, 12512)
```

In [57]:

```
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 cosine 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],metric='cosine')

    # 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)
```

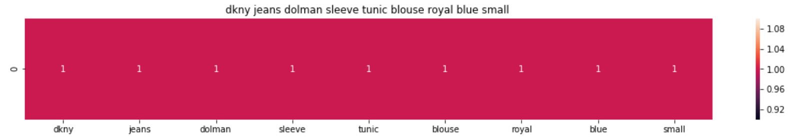
In [58]:

```
# query point:
def display_img_querypoint(url=data.iloc[12566,:]['medium_image_url']):
    # 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)
display_img_querypoint()
```

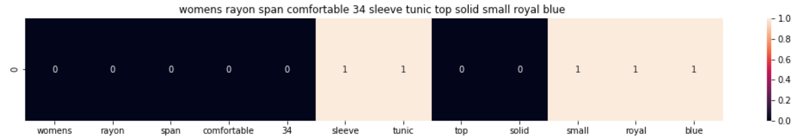


In [59]:

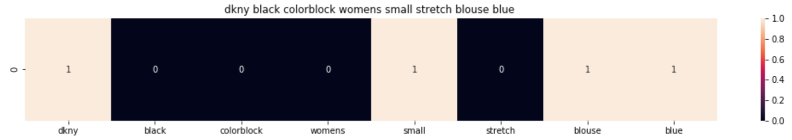
```
#call the bag-of-words model for a product to get similar products.
bag_of_words_model(12566, 5) # change the index if you want to.
# In the output heat map each value represents the count value
# of the label word, the color represents the intersection
# with inputs title.
```



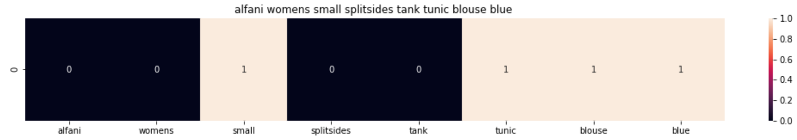
```
ASIN : B00ESZLHCI
Brand: DKNY Jeans
Title: dkny jeans dolman sleeve tunic blouse royal blue small
Euclidean similarity with the query image : 0.0
=============================================================
```



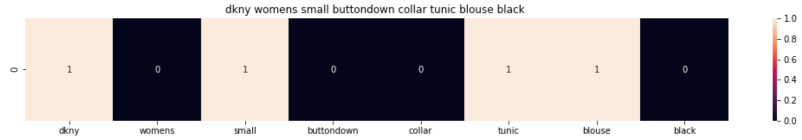
```
ASIN : B010UKQ15W
Brand: NioBe
Title: womens rayon span comfortable 34 sleeve tunic top solid small royal
blue
Euclidean similarity with the query image : 0.5188747756753118
=============================================================
```



```
ASIN : B01J298XVW
Brand: DKNY
Title: dkny black colorblock womens small stretch blouse blue
Euclidean similarity with the query image : 0.5285954792089684
=============================================================
```



```
ASIN : B072FP1NQV
Brand: Alfani
Title: alfani womens small splitsides tank tunic blouse blue
Euclidean similarity with the query image : 0.5285954792089684
=============================================================
```



```
ASIN : B0716YWPN1
Brand: DKNY
Title: dkny womens small buttondown collar tunic blouse black
Euclidean similarity with the query image : 0.5285954792089684
=============================================================
```

[8.5] TF-IDF based product similarity

In [60]:

```
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 g
iven doc
```

In [61]:

```
tfidf_title_features.shape
```

Out[61]:

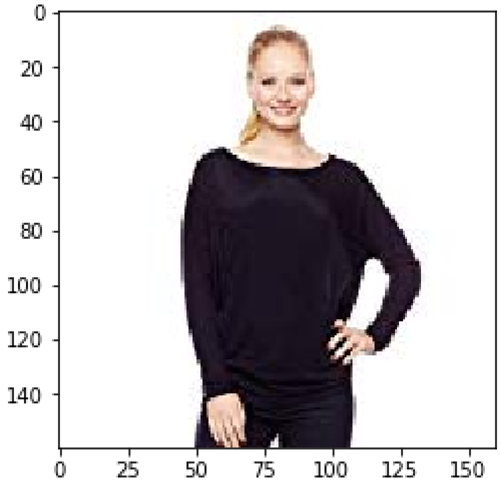
```
(15528, 12512)
```

In [62]:

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

    # 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_i
ndices[i]], data['medium_image_url'].loc[df_indices[i]], 'tfidf')
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('BRAND :',data['brand'].loc[df_indices[i]])
        print ('Eucliden distance from the given image :', pdists[i])
        print('='*125)
# in the output heat map each value represents the tfidf values of the label word, the
 color represents the intersection with inputs title
```

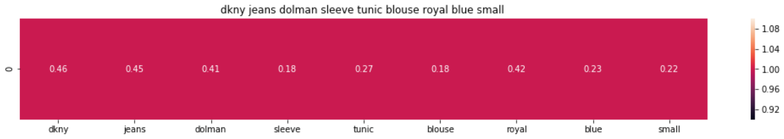
In [63]:

```
# query point:
def display_img_querypoint(url=data.iloc[12566,:]['medium_image_url']):
    # 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)
display_img_querypoint()
```

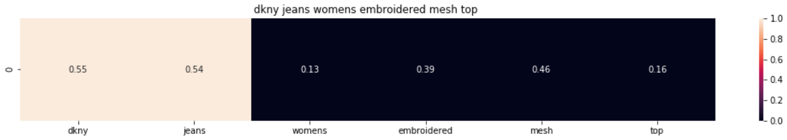


In [64]:

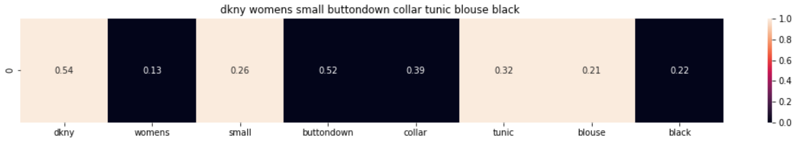
```
tfidf_model(12566, 5)
```



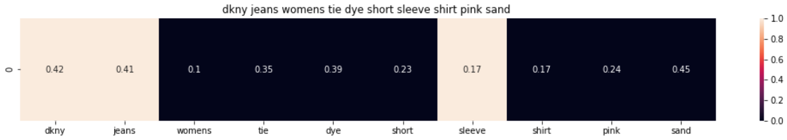
```
ASIN : B00ESZLHCI
BRAND : DKNY Jeans
Eucliden distance from the given image : 0.0
==========================================================================
===================================================
```



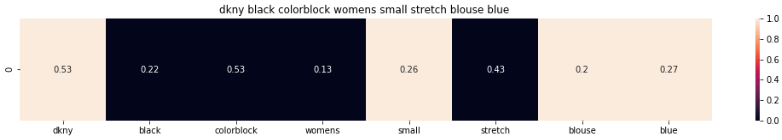
```
ASIN : B00VUL8GY0
BRAND : DKNY Jeans
Eucliden distance from the given image : 0.50552764226898
==========================================================================
===================================================
```



```
ASIN : B0716YWPN1
BRAND : DKNY
Eucliden distance from the given image : 0.5667578412051616
==========================================================================
===================================================
```



```
ASIN : B00S5E1P2A
BRAND : DKNY Jeans
Eucliden distance from the given image : 0.5873162644035862
==========================================================================
===================================================
```



```
ASIN : B01J298XVW
BRAND : DKNY
Eucliden distance from the given image : 0.5977076127219004
==========================================================================
===================================================
```

[8.5] IDF based product similarity

In [65]:

```
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 occure
d in that doc
```

In [66]:

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

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

In [67]:

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

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

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

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

In [68]:

```
idf_title_features.shape
```

Out[68]:

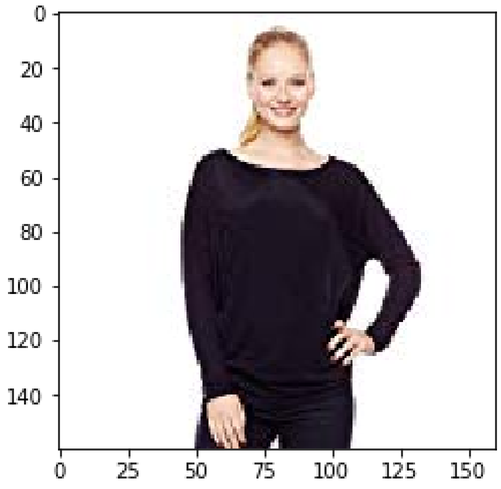
```
(15528, 12512)
```

In [69]:

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

    # pairwise_dist will store the distance from given input apparel to all remaining a
pparels
    # the metric we used here is cosine, the cosine 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],me
tric='cosine')

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

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

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

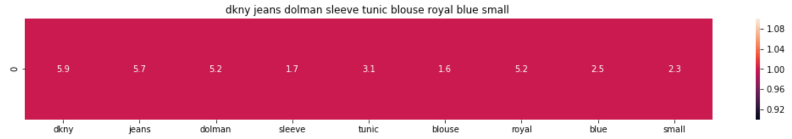
In [70]:

```
# query point:
def display_img_querypoint(url=data.iloc[12566,:]['medium_image_url']):
    # 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)
display_img_querypoint()
```

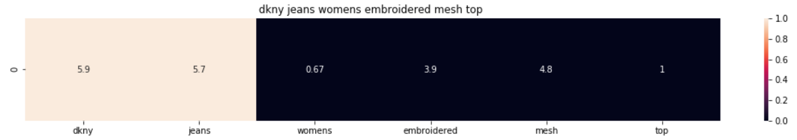


In [71]:

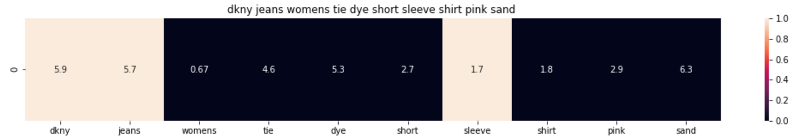
```
idf_model(12566,5)
```



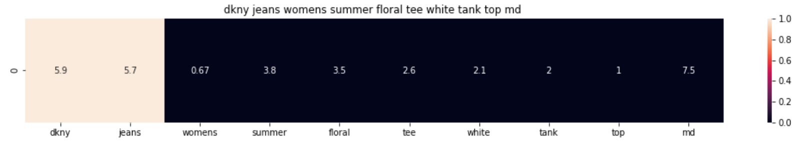
```
ASIN : B00ESZLHCI
Brand : DKNY Jeans
euclidean distance from the given image : 0.0
===========================================================================
===================================================
```



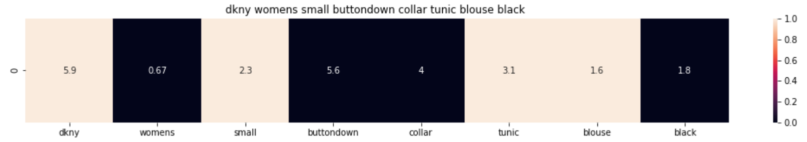
```
ASIN : B00VUL8GY0
Brand : DKNY Jeans
euclidean distance from the given image : 0.4631865308811107
===========================================================================
===================================================
```



```
ASIN : B00S5E1P2A
Brand : DKNY Jeans
euclidean distance from the given image : 0.5656388545742195
===========================================================================
===================================================
```



```
ASIN : B00R2KS8PU
Brand : DKNY Jeans
euclidean distance from the given image : 0.5685773183938205
===========================================================================
===================================================
```



```
ASIN : B0716YWPN1
Brand : DKNY
euclidean distance from the given image : 0.5760782849568237
===========================================================================
===================================================
```

[9] Text Semantics based product similarity

In [72]:

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

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

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

'''
```

Out[72]:

```
'\n# Set values for various parameters\nnum_features = 300    # Word vecto
r dimensionality                      \nmin_word_count = 1    # Minimum wo
rd count                        \nnum_workers = 4       # Number of thread
s to run in parallel\ncontext = 10          # Context window size
\ndownsampling = 1e-3   # Downsample setting for frequent words\n\n# Initi
alize and train the model (this will take some time)\nfrom gensim.models i
mport word2vec\nprint ("Training model...")\nmodel = word2vec.Word2Vec(sen
_corpus, workers=num_workers,              size=num_features, min_count = m
in_word_count,             window = context)\n    \n'
```

In [73]:

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

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

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

In [74]:

```
# Utility functions

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

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

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

            dist.append(np.linalg.norm(i-j))
        final_dist.append(np.array(dist))
    # final_dist = np.array(#number of words in title1 * #number of words in title2)
    # final_dist[i,j] = euclidean distance between vectors i, j
    return np.array(final_dist)


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

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

```python
    #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()
```

In [75]:

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

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

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

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

[9.2] Average Word2Vec product similarity.

In [76]:

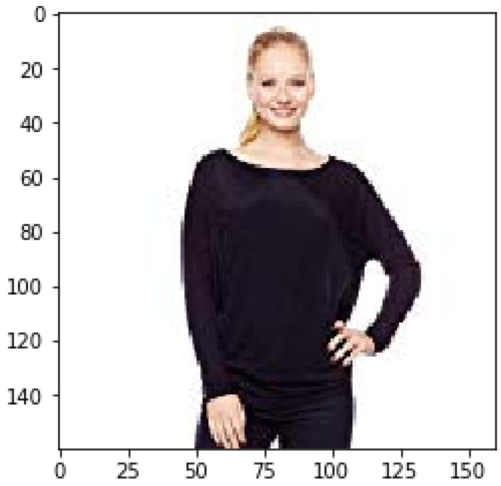
```
doc_id = 0
w2v_title = []
# for every title we build a avg vector representation
for i in data['title']:
    w2v_title.append(build_avg_vec(i, 300, doc_id,'avg'))
    doc_id += 1

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

In [77]:

```
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)
```

In [78]:

```
# query point:
def display_img_querypoint(url=data.iloc[12566,:]['medium_image_url']):
    # 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)
display_img_querypoint()
```



In [79]:

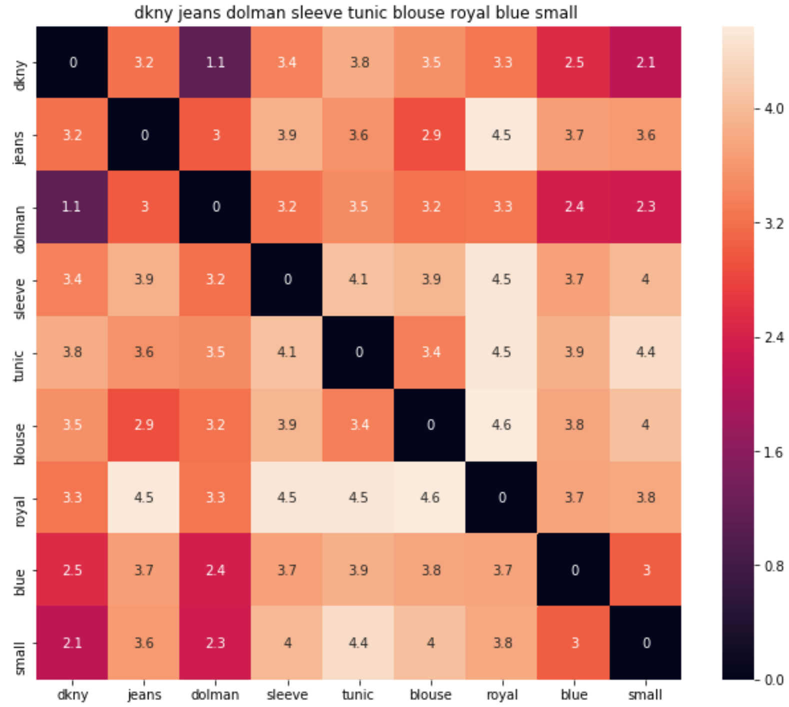
```
data.iloc[12566,:]
```

Out[79]:

```
asin                                                           B00ESZLHCI
brand                                                          DKNY Jeans
color                                                          Royal Blue
medium_image_url     https://images-na.ssl-images-amazon.com/images...
product_type_name                                                   SHIRT
title                dkny jeans dolman sleeve tunic blouse royal bl...
formatted_price                                                    $42.00
Name: 145393, dtype: object
```

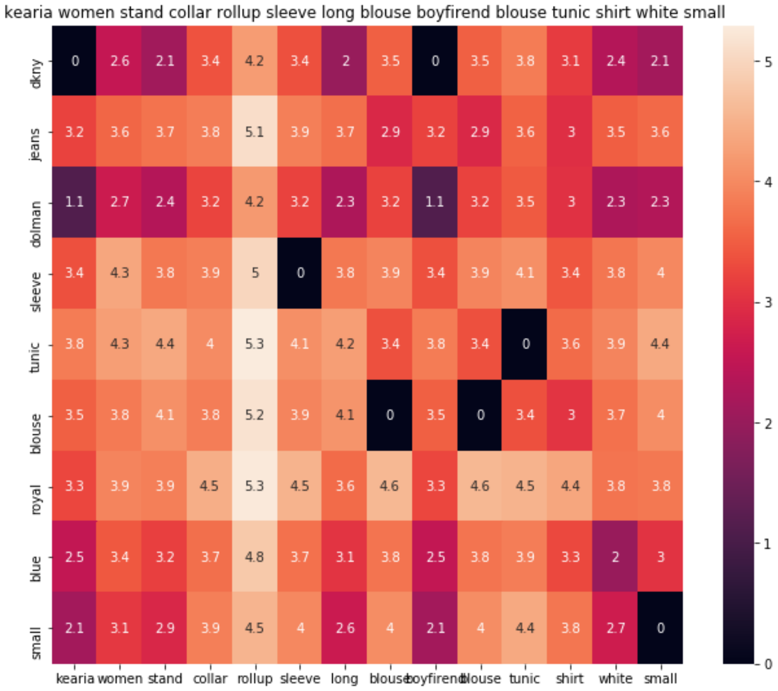
In [80]:

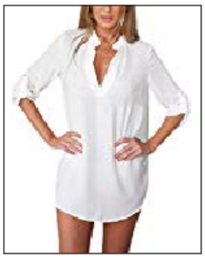
```
avg_w2v_model(12566, 5)
```



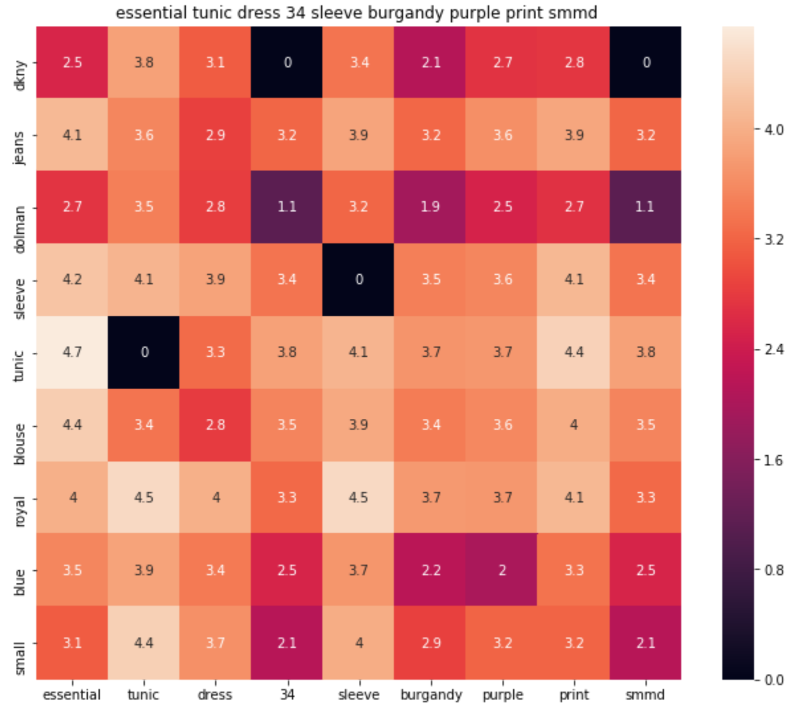
	dkny	jeans	dolman	sleeve	tunic	blouse	royal	blue	small
dkny	0	3.2	1.1	3.4	3.8	3.5	3.3	2.5	2.1
jeans	3.2	0	3	3.9	3.6	2.9	4.5	3.7	3.6
dolman	1.1	3	0	3.2	3.5	3.2	3.3	2.4	2.3
sleeve	3.4	3.9	3.2	0	4.1	3.9	4.5	3.7	4
tunic	3.8	3.6	3.5	4.1	0	3.4	4.5	3.9	4.4
blouse	3.5	2.9	3.2	3.9	3.4	0	4.6	3.8	4
royal	3.3	4.5	3.3	4.5	4.5	4.6	0	3.7	3.8
blue	2.5	3.7	2.4	3.7	3.9	3.8	3.7	0	3
small	2.1	3.6	2.3	4	4.4	4	3.8	3	0

```
ASIN : B00ESZLHCI
BRAND : DKNY Jeans
euclidean distance from given input image : 0.00069053395
============================================================================
====================================================
```



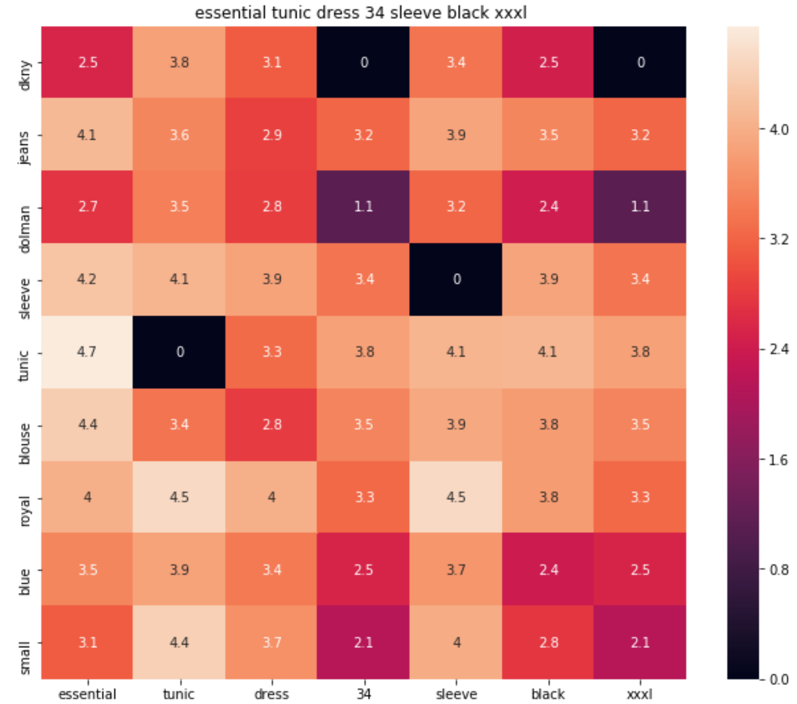


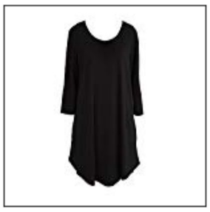
```
ASIN : B01CJP5A12
BRAND : Kearia
euclidean distance from given input image : 0.7071495
===========================================================================
===================================================
```



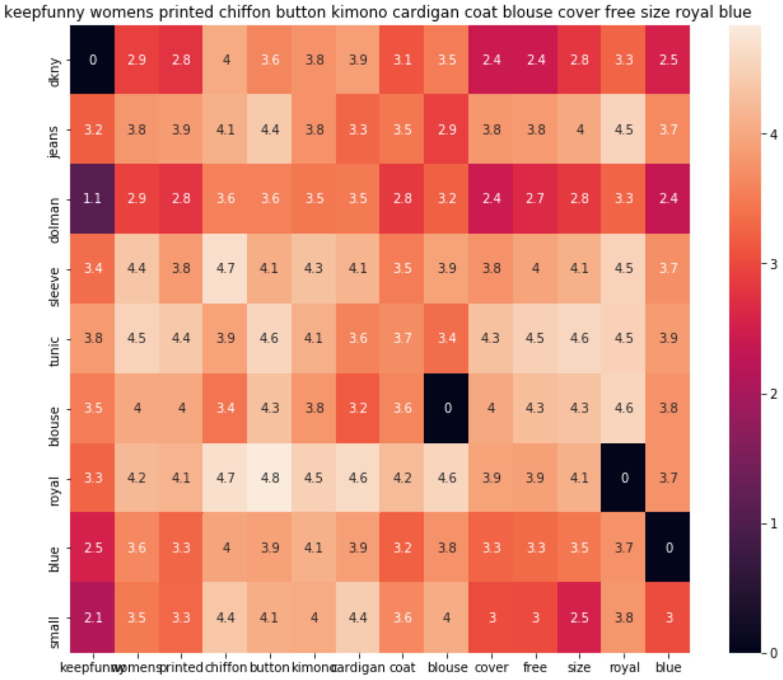
	essential	tunic	dress	34	sleeve	burgandy	purple	print	smmd
dkny	2.5	3.8	3.1	0	3.4	2.1	2.7	2.8	0
jeans	4.1	3.6	2.9	3.2	3.9	3.2	3.6	3.9	3.2
dolman	2.7	3.5	2.8	1.1	3.2	1.9	2.5	2.7	1.1
sleeve	4.2	4.1	3.9	3.4	0	3.5	3.6	4.1	3.4
tunic	4.7	0	3.3	3.8	4.1	3.7	3.7	4.4	3.8
blouse	4.4	3.4	2.8	3.5	3.9	3.4	3.6	4	3.5
royal	4	4.5	4	3.3	4.5	3.7	3.7	4.1	3.3
blue	3.5	3.9	3.4	2.5	3.7	2.2	2	3.3	2.5
small	3.1	4.4	3.7	2.1	4	2.9	3.2	3.2	2.1

```
ASIN : B0748PB38B
BRAND : Mountain Mamas
euclidean distance from given input image : 0.73200744
===========================================================================
===================================================
```





```
ASIN : B074P6LPRM
BRAND : Mountain Mamas
euclidean distance from given input image : 0.7424394
===========================================================================
===================================================
```



```
ASIN : B01IV2EEAK
BRAND : KEEPFUNNY
euclidean distance from given input image : 0.7458595
==========================================================================
===================================================
```

[9.4] IDF weighted Word2Vec for product similarity

In [81]:

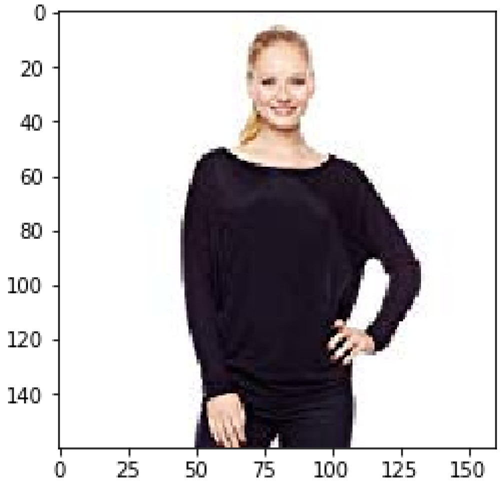
```
doc_id = 0
w2v_title_weight = []
# for every title we build a weighted vector representation
for i in data['title']:
    w2v_title_weight.append(build_avg_vec(i, 300, doc_id,'weighted'))
    doc_id += 1
# w2v_title = np.array(# number of doc in courpus * 300), each row corresponds to a doc
w2v_title_weight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

In [82]:

```
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)
```

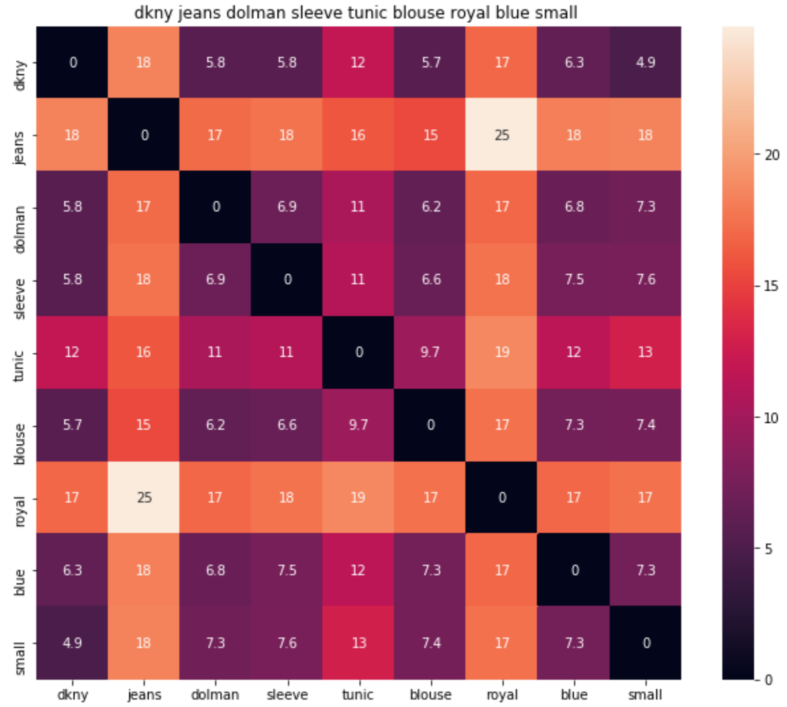
In [83]:

```
# query point:
def display_img_querypoint(url=data.iloc[12566,:]['medium_image_url']):
    # 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)
display_img_querypoint()
```

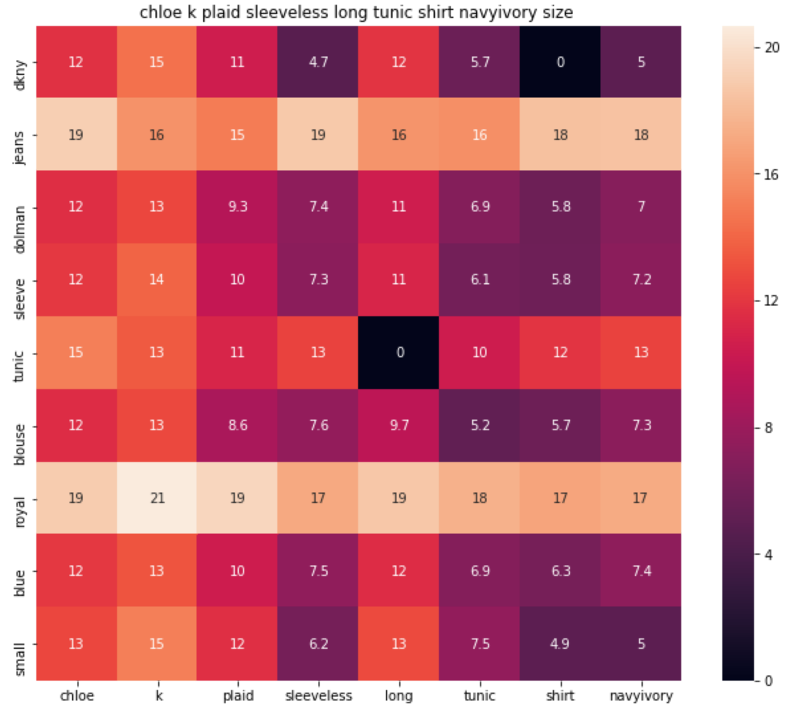


In [84]:

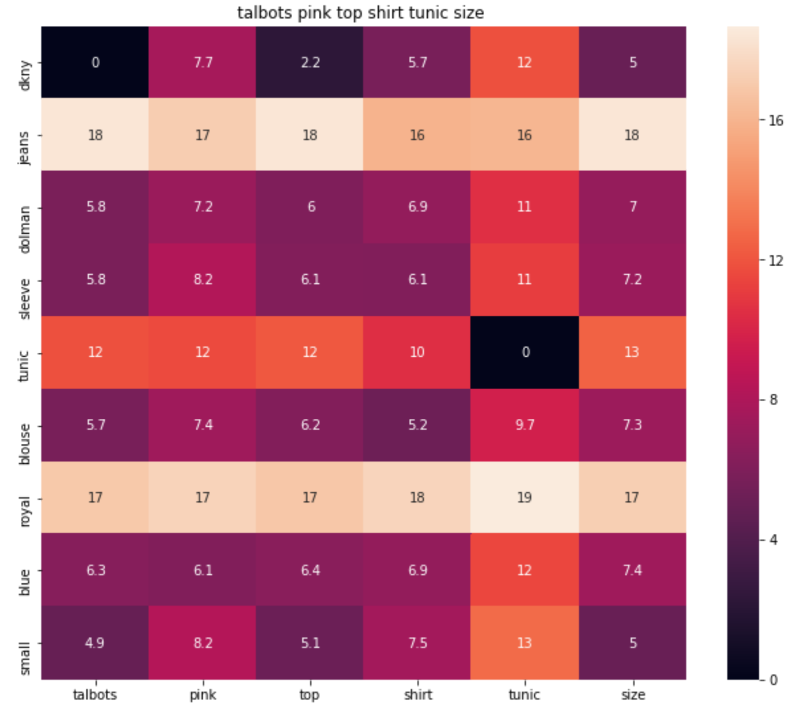
```
weighted_w2v_model(12566, 5)
```



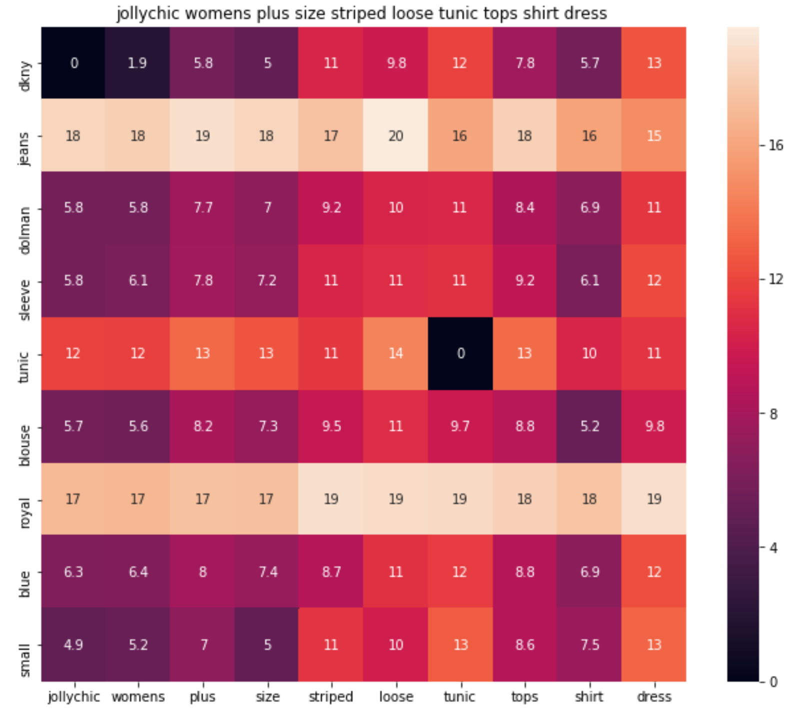
```
ASIN : B00ESZLHCI
Brand : DKNY Jeans
euclidean distance from input : 0.001953125
===========================================================================
====================================================
```



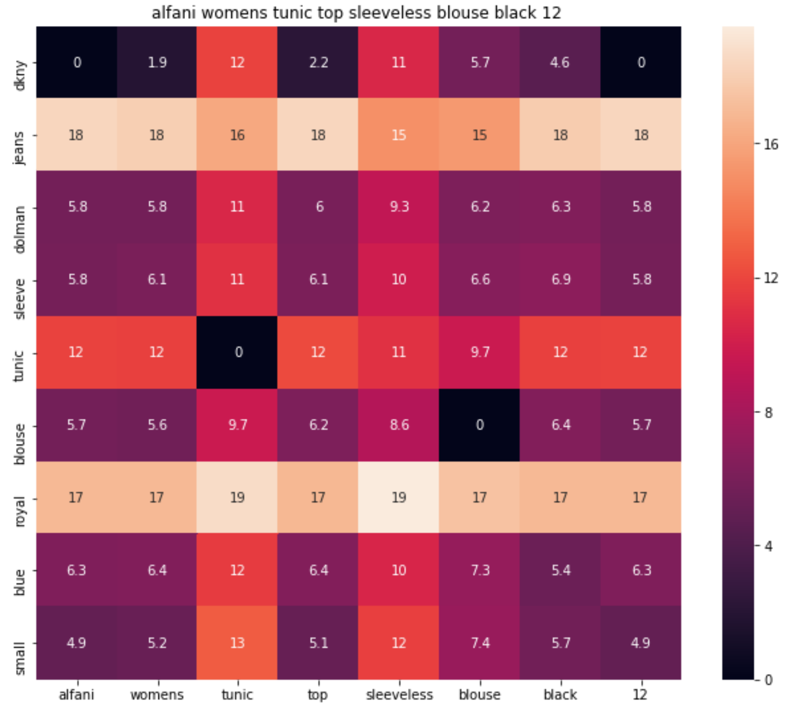
```
ASIN : B0732LF4Q3
Brand : Chloe K.
euclidean distance from input : 2.7857335
===========================================================================
====================================================
```



```
ASIN : B073Y2VZXN
Brand : Talbots
euclidean distance from input : 2.854287
===========================================================================
===================================================
```



```
ASIN : B071XNDLFH
Brand : JOLLYCHIC
euclidean distance from input : 2.8957968
===========================================================================
===================================================
```



```
ASIN : B071V8VJJT
Brand : Alfani
euclidean distance from input : 2.914971
==========================================================================
===================================================
```

[9.6] Weighted similarity using brand and color.

In [85]:

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

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

In [86]:

```
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()
```

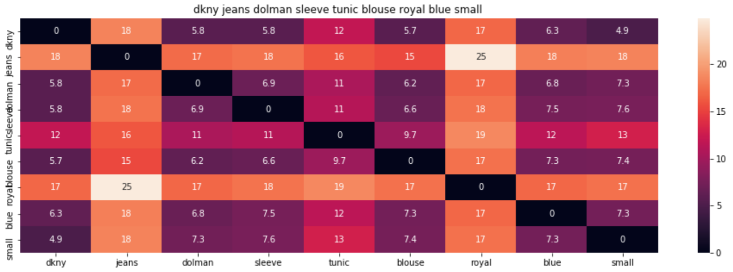
In [87]:

```
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 a
pparels
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y) = <
X, Y> / (||X||*||Y||)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    idf_w2v_dist  = pairwise_distances(w2v_title_weight, w2v_title_weight[doc_id].resha
pe(1,-1))
    ex_feat_dist = pairwise_distances(extra_features, extra_features[doc_id])
    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_indice
s[i]], data['medium_image_url'].loc[df_indices[i]], indices[0], indices[i],df_indices[0
], df_indices[i], 'weighted')
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('Brand :',data['brand'].loc[df_indices[i]])
        print('euclidean distance from input :', pdists[i])
        print('='*125)
```

In [88]:

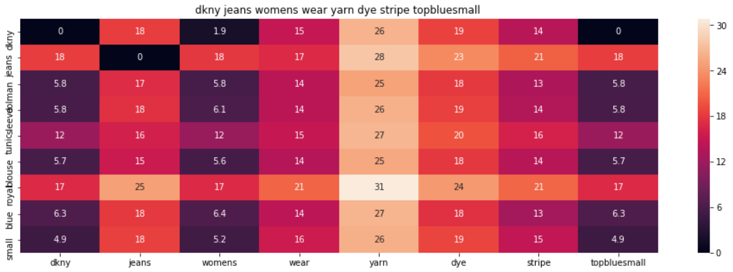
```
# brand and color weight =50
# title vector weight = 5
idf_w2v_brand(12566, 5, 50, 10)
```

Asin	Brand	Color
B00ESZLHCI	DKNY-Jeans	Royal-Blue
B00ESZLHCI	DKNY-Jeans	Royal-Blue



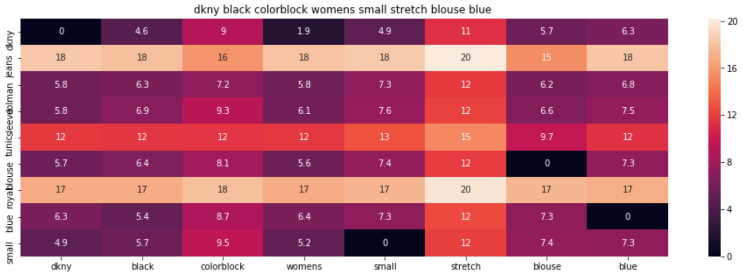
```
ASIN : B00ESZLHCI
Brand : DKNY Jeans
euclidean distance from input : 0.00017755681818182
===========================================================================
====================================================
```

Asin	Brand	Color
B00ESZLHCI	DKNY-Jeans	Royal-Blue
B01N2GT7I1	DKNY-Jeans	Blue



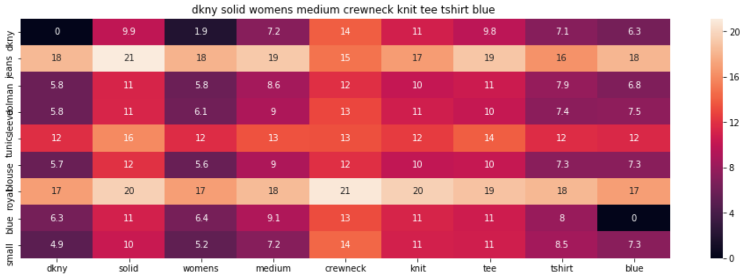
```
ASIN : B01N2GT7I1
Brand : DKNY Jeans
euclidean distance from input : 1.4039134632457386
===========================================================================
====================================================
```

Asin	Brand	Color
B00ESZLHCI	DKNY-Jeans	Royal-Blue
B01J298XVW	DKNY	Blue



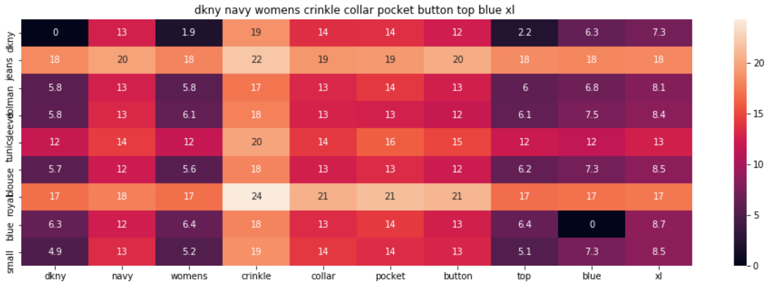
```
ASIN : B01J298XVW
Brand : DKNY
euclidean distance from input : 1.6128510218350935
===========================================================================
===================================================
```

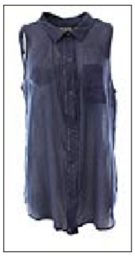
Asin	Brand	Color
B00ESZLHCI	DKNY-Jeans	Royal-Blue
B071JRTC33	DKNY	Blue



```
ASIN : B071JRTC33
Brand : DKNY
euclidean distance from input : 1.6338371280140709
===========================================================================
===================================================
```

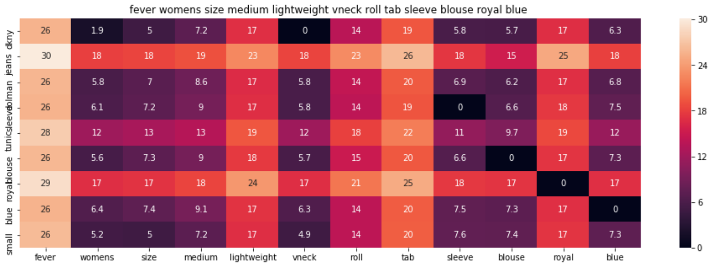
Asin	Brand	Color
B00ESZLHCI	DKNY-Jeans	Royal-Blue
B01G3OAK6A	DKNY	Blue

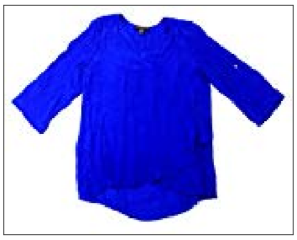




```
ASIN : B01G3OAK6A
Brand : DKNY
euclidean distance from input : 1.6481017376110167
===========================================================================
===================================================
```

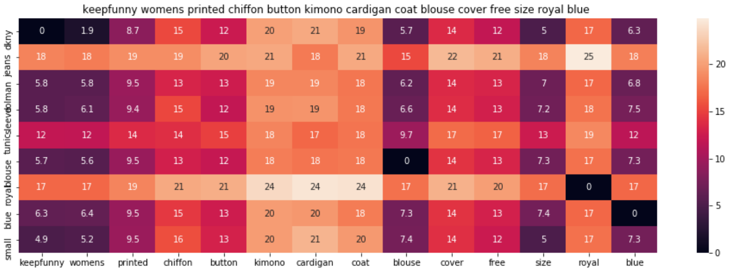
Asin	Brand	Color
B00ESZLHCI	DKNY-Jeans	Royal-Blue
B073ZNQT5M	F	Royal-Blue





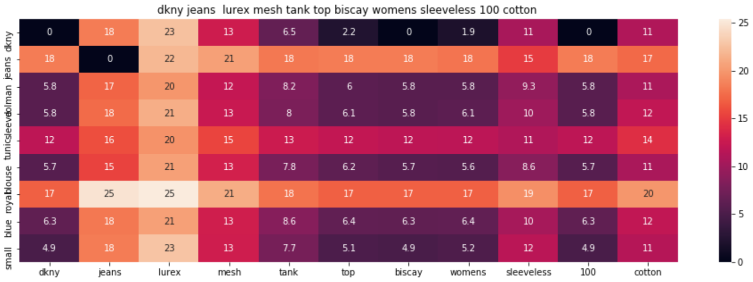
```
ASIN : B073ZNQT5M
Brand : F
euclidean distance from input : 1.685158331072487
===========================================================================
===================================================
```

Asin	Brand	Color
B00ESZLHCI	DKNY-Jeans	Royal-Blue
B01IV2EEAK	KEEPFUNNY	Royal-Blue



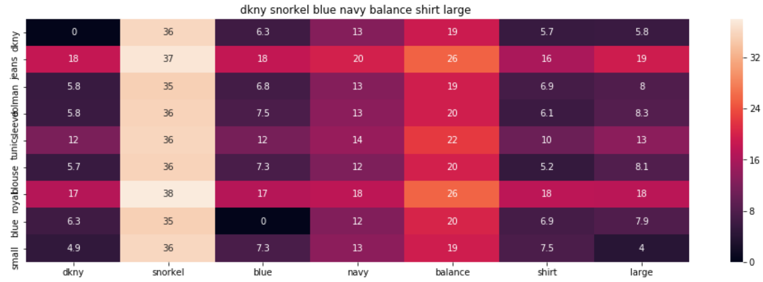
```
ASIN : B01IV2EEAK
Brand : KEEPFUNNY
euclidean distance from input : 1.8580381285904477
============================================================================
====================================================
```

Asin	Brand	Color
B00ESZLHCI	DKNY-Jeans	Royal-Blue
B0756J6PQ7	DKNY-Jeans	Orange



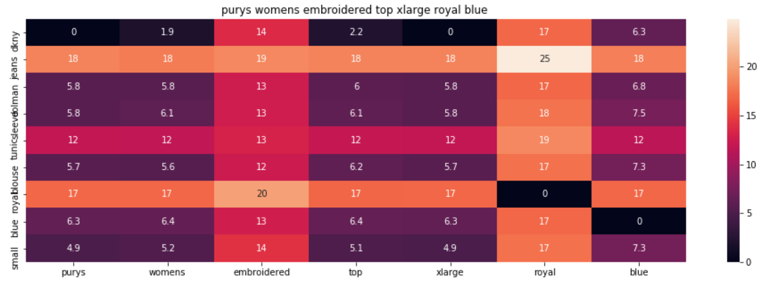
```
ASIN : B0756J6PQ7
Brand : DKNY Jeans
euclidean distance from input : 1.8685866855251512
============================================================================
====================================================
```

Asin	Brand	Color
B00ESZLHCI	DKNY-Jeans	Royal-Blue
B07529ZGK8	DKNY	Blue



```
ASIN : B07529ZGK8
Brand : DKNY
euclidean distance from input : 1.8728316570446106
===========================================================================
===================================================
```

Asin	Brand	Color
B00ESZLHCI	DKNY-Jeans	Royal-Blue
B00XHF54TC	Purys	Royal-Blue



```
ASIN : B00XHF54TC
Brand : Purys
euclidean distance from input : 1.8936487263743167
===========================================================================
===================================================
```

[10.2] Keras and Tensorflow to extract features

In [89]:

```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dropout, Flatten, Dense
from keras import applications
```

```
Using TensorFlow backend.
```

In [90]:

```
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, l
oad_img

datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        rescale=1./255,
        horizontal_flip=True,
        fill_mode='nearest')

img = load_img('images/B000RZ4X7Y.jpeg')  # this is a PIL image
x = img_to_array(img)  # this is a Numpy array with shape (3, 150, 150)
x = x.reshape((1,) + x.shape)  # this is a Numpy array with shape (1, 3, 150, 150)

# the .flow() command below generates batches of randomly transformed images
# and saves the results to the `preview/` directory
i = 1
for batch in datagen.flow(x, batch_size=1,
                          save_to_dir='preview', save_prefix='img', save_format='jpeg'
):
    i += 1
    if i > 10:
        break  # otherwise the generator would loop indefinitely
```

In [91]:

```
# 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 = 'kerasimages/train'
nb_train_samples = 15528
epochs = 50
batch_size = 1

def save_bottlebeck_features():

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

    asins = []

    #transformation on the images:
        #->only performing normalization
    datagen = ImageDataGenerator(rescale=1./255)

    generator = datagen.flow_from_directory(train_data_dir,
                                            target_size=(img_width, img_height),
                                            batch_size=batch_size,
                                            class_mode=None,
                                            shuffle=False)

    # build the VGG16 network
    model = applications.VGG16(include_top=False, weights='imagenet')

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

    bottleneck_features_train = model.predict_generator(generator, nb_train_samples)
    bottleneck_features_train = bottleneck_features_train.reshape((15528,25088))

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

if not os.path.exists('16k_data_cnn_bottleneck_features.npy'):
    save_bottlebeck_features()
```

[10.3] Visual features based product similarity.

In [92]:

```
#load the features and corresponding ASINS info.
bottleneck_features_train = np.load('16k_data_cnn_bottleneck_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()

for i in list(data['asin']):
    df_asins.append(i[0:])


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 = df_asins.index(df_asins[doc_id])
    pairwise_dist = pairwise_distances(bottleneck_features_train, bottleneck_features_t
rain[doc_id].reshape(1,-1))

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

    for i in range(len(indices)):
        rows = data[['medium_image_url','title']].loc[data['asin']==df_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/'+ df_asins[indices[i]])
```

In [93]:

```
get_similar_products_cnn(12566, 10)
```

```
Product Title:  dkny jeans dolman sleeve tunic blouse royal blue small
Euclidean Distance from input image: 0.044194173
Amazon Url: www.amzon.com/dp/B00ESZLHCI
```

```
Product Title:  nobody cares women black short sleeve tshirt
Euclidean Distance from input image: 41.06858
Amazon Url: www.amzon.com/dp/B01HBDJVW6
```

```
Product Title:  ella moss mint womens small splitback seamed blouse blue
Euclidean Distance from input image: 43.310043
Amazon Url: www.amzon.com/dp/B074QVLHBM
```

```
Product Title:  rick morty fingerless gloves
Euclidean Distance from input image: 44.03959
Amazon Url: www.amzon.com/dp/B01MTP0QZY
```

```
Product Title:  towi lady little big town pain killer vneck tshirt black
Euclidean Distance from input image: 44.31086
Amazon Url: www.amzon.com/dp/B0159XJ3NM
```

```
Product Title:  frame navy sleeveless silk blouse
Euclidean Distance from input image: 44.372547
Amazon Url: www.amzon.com/dp/B06XKYXBKP
```

```
Product Title:  devon  jones womens executive club polo
Euclidean Distance from input image: 44.417816
Amazon Url: www.amzon.com/dp/B0009MDHAE
```

```
Product Title:  ca fashion womens printed half sleeve tshirt tops tee
Euclidean Distance from input image: 45.01516
Amazon Url: www.amzon.com/dp/B00FZMBGHY
```

```
Product Title:  pepin womens charlotte fringe tee black size small
Euclidean Distance from input image: 45.251247
Amazon Url: www.amzon.com/dp/B0716Z8WFL
```

```
Product Title:  kenneth cole new york womens white aurore foldover back to
p blouse large
Euclidean Distance from input image: 45.37145
Amazon Url: www.amzon.com/dp/B017VCIYF2
```

In [97]:

```
def weighted_final(doc_id, w1, w2, w3, w4, num_results):
    idf_w2v_dist    = pairwise_distances(w2v_title_weight, w2v_title_weight[doc_id].reshape(1,-1))
    brand_feat_dist = pairwise_distances(brand_features, brand_features[doc_id].reshape(1,-1))
    color_feat_dist = pairwise_distances(color_features, color_features[doc_id].reshape(1,-1))
    image_feat_dist = pairwise_distances(bottleneck_features_train, bottleneck_features_train[doc_id].reshape(1,-1))
    #Avg weighted sum:
    pairwise_dist   = (w1*idf_w2v_dist + w2*brand_feat_dist + w3*color_feat_dist + w4*image_feat_dist)/float(w1 + w2 + w3 + w4)

    # 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(len(indices)):
        rows = data[['medium_image_url','title']].loc[data['asin']==df_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/'+ df_asins[indices[i]])
```

In [119]:

```
weighted_final(doc_id=12566, w1=30, w2=30, w3=5, w4=0.5, num_results=10)
```

```
Product Title:  dkny jeans dolman sleeve tunic blouse royal blue small
Euclidean Distance from input image: 0.0012319211684565508
Amazon Url: www.amzon.com/dp/B00ESZLHCI
```

```
Product Title:  dkny jeans  lurex mesh tank top biscay womens sleeveless 1
00 cotton
Euclidean Distance from input image: 2.0448386292492042
Amazon Url: www.amzon.com/dp/B0756J6PQ7
```

```
Product Title:  dkny jeans womens embroidered mesh top
Euclidean Distance from input image: 2.2780360156343185
Amazon Url: www.amzon.com/dp/B00VUL8GY0
```

```
Product Title:  dkny jeans womens tie dye short sleeve shirt pink sand
Euclidean Distance from input image: 2.3316732878064266
Amazon Url: www.amzon.com/dp/B00S5E1P2A
```

```
Product Title:  dkny womens small buttondown collar tunic blouse black 
Euclidean Distance from input image: 2.3408050324241505
Amazon Url: www.amzon.com/dp/B0716YWPN1
```

```
Product Title:  dkny womens cotton printed pullover top blue 
Euclidean Distance from input image: 2.4145067890252663
Amazon Url: www.amzon.com/dp/B00JA13ZN4
```

```
Product Title:  dkny jeans women sharkbite scoopneck tank xl azalea red 
Euclidean Distance from input image: 2.5235828952934907
Amazon Url: www.amzon.com/dp/B00YCKS1BO
```

```
Product Title:  dkny black colorblock womens small stretch blouse blue 
Euclidean Distance from input image: 2.5932949444719853
Amazon Url: www.amzon.com/dp/B01J298XVW
```

```
Product Title:  dkny womens petite knit racerback pullover tank top black
p
Euclidean Distance from input image: 2.604040838022985
Amazon Url: www.amzon.com/dp/B071RR7P1L
```

```
Product Title:  versace jeans white short sleeves womens blouse top us 40
Euclidean Distance from input image: 2.619129392745544
Amazon Url: www.amzon.com/dp/B01LI5ED76
```