# Creating a Dataframe and indexing embedding to Faiss for searching.

In [ ]:

```
!wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.128 Safari/537.36" --header="Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9" --header="Accept-Language: en-IN,en-GB;q=0.9,en-US;q=0.8,en;q=0.7" --header="Referer: https://www.kaggle.com/" "https://storage.googleapis.com/kaggle-data-sets/930393/1613771/bundle/archive.zip?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccount.com%2F20210422%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20210422T055341Z&X-Goog-Expires=259199&X-Goog-SignedHeaders=host&X-Goog-Signature=a8d3db4154d5897288f3da7f13a0dedd7716729e33386e91a9a625f60374ec86f39477b70d9e4ab3b1b6afa68d01216c8a9f254838ab3f556d9c4f5fef89d6afddd8dee59a2583ba0a7aa1f408f56d88a42d07de560360b0792ef9ea31cb287f3ac5691040ff7419f8b46116aff0cc89ffaebf9063cccacbef18907b0c3ed74ca9b87a3b217add4bf64c6d98e0b4687721218958b784157886a64ddb81dc2d7bde3b31a2e3c76a6e571ea91d9a025e757e6ea7e4e3504a707b05ba0d15282950fdc654fcd282cecfbbf58d6b6c87aa8c85f1f3d418b31004460534b9345808b9859c875c33e4458090d662fc643cf87a2736e862ee9d354f2638e5610a1663f5" -c -O 'archive.zip'
```

```
--2021-04-22 05:55:16--  https://storage.googleapis.com/kaggle-data-sets/930393/1613771/bundle/archive.zip?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccount.com%2F20210422%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20210422T055341Z&X-Goog-Expires=259199&X-Goog-SignedHeaders=host&X-Goog-Signature=a8d3db4154d5897288f3da7f13a0dedd7716729e33386e91a9a625f60374ec86f39477b70d9e4ab3b1b6afa68d01216c8a9f254838ab3f556d9c4f5fef89d6afddd8dee59a2583ba0a7aa1f408f56d88a42d07de560360b0792ef9ea31cb287f3ac5691040ff7419f8b46116aff0cc89ffaebf9063cccacbef18907b0c3ed74ca9b87a3b217add4bf64c6d98e0b4687721218958b784157886a64ddb81dc2d7bde3b31a2e3c76a6e571ea91d9a025e757e6ea7e4e3504a707b05ba0d15282950fdc654fcd282cecfbbf58d6b6c87aa8c85f1f3d418b31004460534b9345808b9859c875c33e4458090d662fc643cf87a2736e862ee9d354f2638e5610a1663f5
Resolving storage.googleapis.com (storage.googleapis.com)... 108.177.126.128, 108.177.127.128, 172.217.218.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|108.177.126.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11894512126 (11G) [application/zip]
Saving to: 'archive.zip'

archive.zip         100%[===================>]  11.08G  29.7MB/s    in 4m 29s

2021-04-22 05:59:46 (42.2 MB/s) - 'archive.zip' saved [11894512126/11894512126]
```

In [ ]:

```
!mkdir data
!unzip -q '/content/archive.zip' -d '/content/data'
!rm -rf '/content/archive.zip'
```

In [ ]:

```
import os
#get data from json file
folder = []
json_file =[]
for (root,dirs,files) in os.walk('/content/data/', topdown=False):
  for directory in dirs:
    for file in os.listdir(root+directory):
      if file.endswith('.json'):
        folder.append(directory)
        json_file.append(file)
```

In [ ]:

```
fashion = pd.read_csv("info.csv")
```

In [ ]:

```
import tensorflow as tf
import cv2
import numpy as np
tf.keras.backend.clear_session()

model_embedding = tf.keras.applications.DenseNet121(include_top=False, weights='imagenet
', input_tensor=None, input_shape=(520,520,3),pooling=None,)
def load_img(path):
  # Reading an image
  image = cv2.imread(path)
  # resizing because pre-trained model image shape is 520x520
  image = cv2.resize(image,(520,520),interpolation=cv2.INTER_AREA)
  # Converting to RBG because it will be saved as a correct image even if it is saved aft
er being converted to a PIL
  image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
  # Preprocessed numpy.array or a tf.Tensor with type float32.
  image = tf.image.convert_image_dtype(image,tf.float32)[tf.newaxis, ...]
  return image

def get_embeddings(path):
  image = load_img(path)
  # Retuns 1024 dimension array/ vector with predicted values
  img_embedding = model_embedding.predict(image,steps=1)
  # Removes dimensions of size 1 from the shape of a tensor.
  img_embedding = tf.squeeze(img_embedding, axis=None, name=None)
  # Computes the mean of elements across dimensions of a tensor. [ Normalize ]
  img_embedding = tf.reduce_mean(img_embedding, axis=(0,1), keepdims=False, name=None).n
umpy()
  # Converting to List
  img_embedding = img_embedding.tolist()
  return img_embedding
```

In [ ]:

```
#genrate docs and embedding to transfer to Elastic search
super_cat = []
url = []
file_name = []
embedding = []
for row in fashion.iterrows():
  path = root + row[1]['cat']+'/' + row[1]['file_name']
  em = get_embeddings(path)
  super_cat.append(row[1]['super_cat'])
  url.append(row[1]['url'])
  file.append(row[1]['file_name'])
  embedding.append(em)
  c+=1
  if c%5000==0:
    print(c)
```

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

## reading the data from saved location.

In [ ]:

```
upper_ware = pd.read_csv("/content/drive/MyDrive/Applied AI Course/Assignments/29. Self C
ase Study 2/upper_ware.csv")
lower_ware = pd.read_csv("/content/drive/MyDrive/Applied AI Course/Assignments/29. Self C
ase Study 2/lower_ware.csv")
foot_ware = pd.read_csv("/content/drive/MyDrive/Applied AI Course/Assignments/29. Self Ca
se Study 2/foot_ware.csv")
```

**Generating list of embeddings as Faiss takes only nxd matrices dtype float32**

In [ ]:

```
upper_img_embedding = upper_ware.img_embedding.tolist()
lower_img_embedding = lower_ware.img_embedding.tolist()
foot_img_embedding = foot_ware.img_embedding.tolist()
```

In [ ]:

```
!pip install wget
!pip install faiss-cpu --no-cache
```

In [ ]:

```
# As the embedding stored in from os strings converting them to list by using EVAL method
upper_list_ = []
lower_list_ = []
foot_list_  = []
for i in range(len(upper_img_embedding)):
  upper_list_.append(eval(upper_img_embedding[i]))

for i in range(len(lower_img_embedding)):
  lower_list_.append(eval(lower_img_embedding[i]))

for i in range(len(foot_img_embedding)):
  foot_list_.append(eval(foot_img_embedding[i]))

# Converting the list to Arrya and chaning type to float32 as the FAISS takes only array
with type float32

upper_ =np.array([np.array(xi) for xi in upper_list_])
upper_ = upper_.astype('float32')

lower_ =np.array([np.array(xi) for xi in lower_list_])
lower_ = lower_.astype('float32')

foot_ =np.array([np.array(xi) for xi in foot_list_])
foot_ = foot_.astype('float32')

import faiss
## Creating three index for upper ware, lower ware and foot ware.
upper_index = faiss.IndexFlatL2(1024)   # build the index with len of list[ as embedding
length is 1024 , have to pass argument as 1024] L2 distance
lower_index = faiss.IndexFlatL2(1024)
foot_index = faiss.IndexFlatL2(1024)
print(upper_index.is_trained)
print(lower_index.is_trained)
print(foot_index.is_trained)

## Adding embeddings to INDEX.
upper_index.add(upper_)
lower_index.add(lower_)
foot_index.add(foot_)                 # add vectors to the index
```

```
True
True
True
```

**Now let's test the Faiss seach engine**

In [ ]:

```
## generating embedding
path = "/content/9a6e150b28e55527140ae9374ba022d0.jpg"
embedding = get_embeddings(path)
```

In [ ]:

```
from google.colab.patches import cv2_imshow
img = cv2.imread("/content/9a6e150b28e55527140ae9374ba022d0.jpg")
img = cv2.resize(img,(200,200))
cv2_imshow(img)
```

In [ ]:

```
# Embedding is list type. Now we have convert it into array and change dtype to float32
embedding = np.asarray(embedding)
embedding = embedding.astype('float32')
```

In [ ]:

```
# search query should be in same shape of index (1x1024). So, let's print the shape of ar
ray
embedding.shape
```

Out[ ]:

```
(1024,)
```

In [ ]:

```
# Convert it to row vector by adding a new axis
upper_vec = embedding[np.newaxis, :]
print(upper_vec.shape)
```

```
(1, 1024)
```

In [ ]:

```
## As the image have only upper ware. Now we gonna search for similar dress in upper inde
x
D, I = upper_index.search(upper_vec,10)    #Embedding and K nearest neighbers.

# D is Disatnce and I is index value
print("Disatnce", D)
print("**********************")
print("Index",I)
```

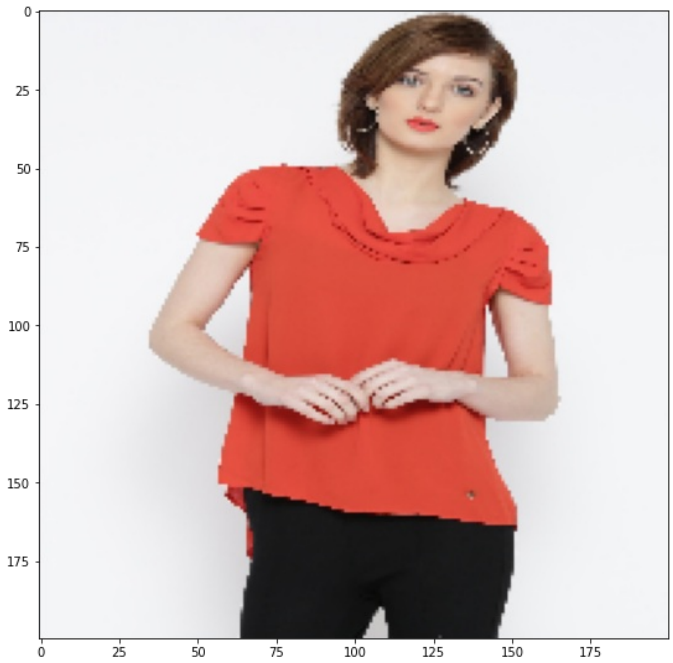
```
Disatnce [[16.62631  16.857439 16.857439 17.207136 17.452564 17.452564 17.452564
  17.606245 17.606245 17.965452]]
**********************
Index [[16135  4653  7183   215  5584  8248   983 13433  4210 11637]]
```

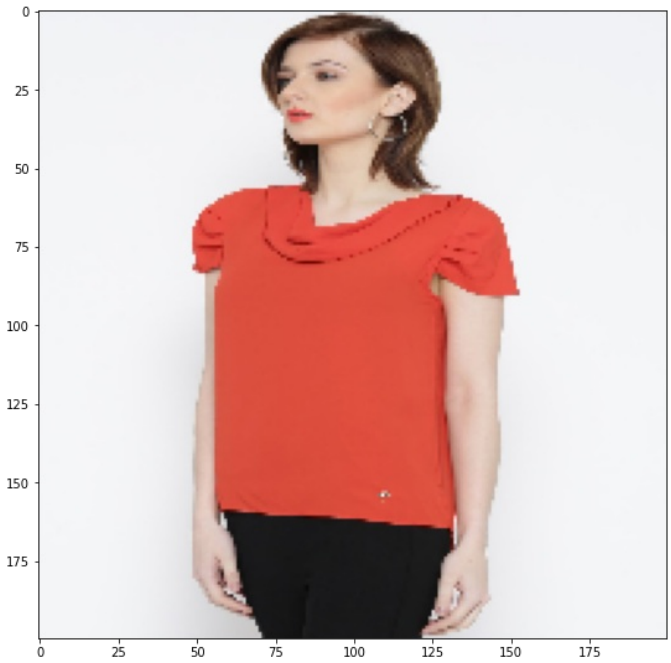
**As we know there are 1024 columns and some rows in INDEX. So, we have to make sure that the search index should be a row vector with shape 1x1024.**

In [ ]:

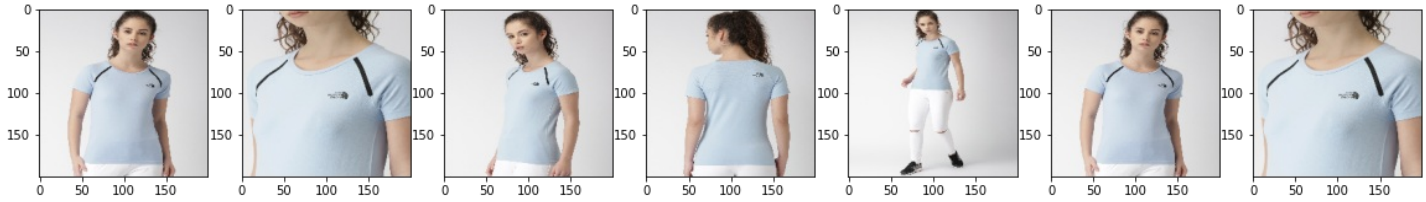
```
### Calling the funtion print_recommendations with arguments as search index values and u
pperware dataframe.
print_recommendations(I,upper_ware)
```

```
recomendation 1
You can buy product from below link
https://www.myntra.com/tops/rinascimento/rinascimento-women-red-top/1386159/buy
```

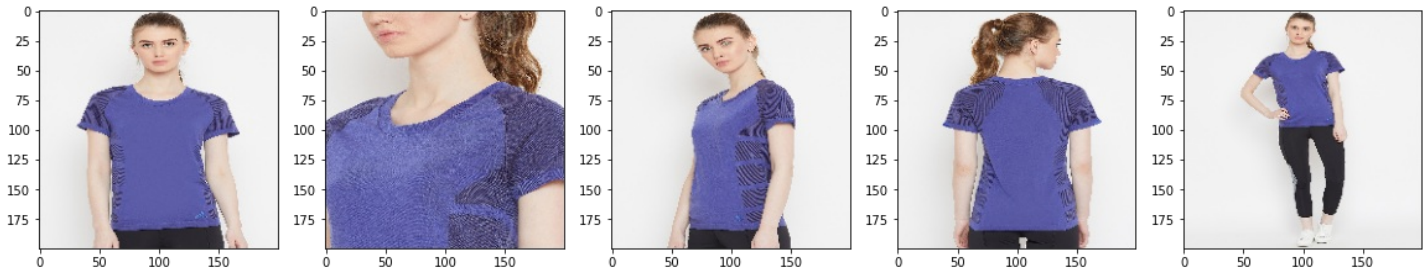




recomendation 2
You can buy product from below link
https://www.myntra.com/tshirts/the-north-face/the-north-face-women-blue-active-fit-fast-hike-t-shirt/7014428/buy



recomendation 3
You can buy product from below link
https://www.myntra.com/tshirts/adidas/adidas-women-blue-cru-pk-running-t-shirt/6824009/buy



In [ ]:

```
from PIL import Image
import wget
import matplotlib.pyplot as plt

def get_image_links(df,url,img_emb):
  image_links = []
  # Creating a new dataframe based on buy url link
  df_new = img_emb[img_emb['url'] == url]
  df_new.reset_index(inplace = True)
  if df_new.shape[0] > 4:
    for i in range(int(df_new.shape[0]/2)):
```

```
      # Retreving the image url's and storing them to image_links list
      image_links.append(df_new["img_url"][i])
  else:
    for i in range(int(df_new.shape[0]/2)):
      image_links.append(df_new["img_url"][i])
  # returning the final list
  return image_links



def url_to_image(link):
  # Downloading the images and saving img info to filename variable
  filename = wget.download(link)
  # Reading img information
  img = cv2.imread(filename,cv2.IMREAD_UNCHANGED)
  # Resizing img
  img = cv2.resize(img,(200,200))
  # Changing to original colour formate RGB
  img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
  #return img
  return img


def plot_images(li):
  # Creating subplots
  fig, ax = plt.subplots(1,len(li), figsize=(20,20))
  for i in range(len(li)):
    #ploting the images
    ax[i].imshow(li[i].astype(np.uint8))
  plt.show()



def print_recommendations(I,img_emb):
  # We want to print only top 4 recommendation. So, counting it and stopping once "c" rea
ches to 4
  c = 1
  # Storing unique URL of all index values because we have duplicate buy "url" associated
with unique image url's
  url_exist = []

  # Faiss returns list of list. So, saving all index value under one list
  returned_embedding = []
  for i in range(I.shape[1]):
    returned_embedding.append(I[0][i])

  # Filtering the dataframe based on index values and storing all buy url's in URL list
  new_img_emb = img_emb[img_emb.index.isin(returned_embedding, level=None)]
  info = new_img_emb["url"].tolist()

  # For every url in info list
  for url in info:
    ## check if already we have printed the buy url it not
    if url not in url_exist:
      url_exist.append(url)
      # Retieving the img url from original dataframe.
      images_links = get_image_links(new_img_emb,url,img_emb)

      images = []
      ## for every image url from image_links list
      for link in images_links:
        # retrieving the images using wget
        img = url_to_image(link)
        ## Appending to images list
        images.append(img)

      print("Recomendation",c)
      print("You can buy product from below link")
      print(url)
      print("\n")
      # Plot recommanded images
      plot_images(images)
```

```
        print("\n")
        c+=1
        if c==4:
          break
```