NLP Code

April 4, 2021

1 Project Phase 2

2 Team Purple

2.1 Notebook Description

Executive Summary - This notebook builds a recommendation system for the outfit combinations file using Product ID and/or free text. - The notebook allows a user to enter inputs – either product IDs or product descriptions and details - and returns recommended outfits. - We tried different vectorization techniques, using Word2Vec, Word2Vec weighted average by TF-IDF, 1-Hot encoding - We then calculated a similarity score between the user's input and the existing Product ID, descriptions and Full names in the database, in order to find the best match for the given input

> We finally choose Word2vec embeddings (Skipgram) as our final model. It uses the 'Spacy' library to generate document vectors by averaging individual word vectors

[1]:
```
# Importing relevant libraries

import pandas as pd
import re
import nltk
import spacy
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import numpy as np
from numpy import array, argmax, asarray, zeros
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import cosine_similarity
from fuzzywuzzy import fuzz
from tabulate import tabulate
from scipy.spatial.distance import cosine
from keras.preprocessing.text import Tokenizer
from gensim.test.utils import common_texts
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
import warnings
warnings.filterwarnings("ignore")
nlp = spacy.load("en_core_web_md")
```

```
C:\Users\rajat\Anaconda3\lib\site-packages\fuzzywuzzy\fuzz.py:11: UserWarning:
Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove
this warning
  warnings.warn('Using slow pure-python SequenceMatcher. Install python-
Levenshtein to remove this warning')
Using TensorFlow backend.
WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard
installation.
WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard
installation.
WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard
installation.
WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard
installation.
WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard
installation.
WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard
installation.
WARNING:root:Limited tf.summary API due to missing TensorBoard installation.
```

[2]:
```
# Reading Outfit Combinations Provided by Experts
## We have merged Product Description from Part 1 on Product ID as unique key
data = pd.read_csv("outfit_combinations_description.csv")

# Replacing Null values in product description with blank space
data['product description'] = data['product description'].fillna('')
data.head()
```

[2]:
```
                    outfit_id                   product_id outfit_item_type  \
0  01DDBHC62ES5K80P0KYJ56AM2T  01DMBRYVA2P5H24WK0HTK4R0A1           bottom
1  01DDBHC62ES5K80P0KYJ56AM2T  01DMBRYVA2PEPWFTT7RMP5AA1T              top
2  01DDBHC62ES5K80P0KYJ56AM2T  01DMBRYVA2S5T9W793F4CY41HE       accessory1
3  01DDBHC62ES5K80P0KYJ56AM2T  01DMBRYVA2ZFDYRYY5TRQZJTBD             shoe
4  01DMHCX50CFX5YNG99F3Y65GQW  01DMBRYVA2P5H24WK0HTK4R0A1           bottom

                brand               product_full_name product description
0        Eileen Fisher                  Slim Knit Skirt        A nice skirt
1        Eileen Fisher               Rib Mock Neck Tank         A nice tank
2  kate spade new york  medium margaux leather satchel          A nice bag
3           Tory Burch         Penelope Mid Cap Toe Pump         A nice shoe
4        Eileen Fisher                  Slim Knit Skirt        A nice skirt
```

[3]:
```
# Defining Recommendation Function
## This function returns a randomly chosen set of outfit combinations for a␣
↪given product ID

def recommendation(prod_id):
    df_product = data[data['product_id'] == prod_id].reset_index(drop = True)
    outfit_type = df_product.loc[0,"outfit_item_type"]
    print(f"Outfit type for product id {prod_id} is :",outfit_type,"\n")

    outfit_id_show = list(np.random.choice(a = list(df_product['outfit_id']),␣
 ↪size = 1))
    df_outfit = data[data['outfit_id'] == outfit_id_show[0]]
    print(f"Matching Outfit ID is :",outfit_id_show[0],"\n")
    output = df_outfit['outfit_item_type']+ ": "+␣
 ↪df_outfit['product_full_name']+ " (" + df_outfit['product_id'] + ")"
    return output
```

3 Product ID Input

if user has the Product ID, they can enter it in the cell below, and get the corresponding outfit recommendations

[4]:
```
  # User Input
prod_id = "01DMBRYVA2ZFDYRYY5TRQZJTBD"            # <----- ENTER INPUT HERE



# Removes any white spaces to give a contiguous string for exact match
prod_id = ''.join(prod_id.split())

## Assigns productID similarity score to each row using Fuzzywuzzy library
## Chooses list of top 3 unique scores as suggested product IDs, in case of no␣
 ↪exact match
## User may choose one of the suggested 3 IDs, and re-enter in the user input␣
 ↪space (in Line 2 above)

df = data.copy()
df['fuzz_score'] = data["product_id"].apply(lambda x: fuzz.ratio(x,prod_id.
 ↪upper()))
df = df.sort_values(by = 'fuzz_score', ascending = False)
matches = list(pd.Series(df['product_id'].unique())[:3])

# If a perfect match is found then recommendation function is called
if (df['fuzz_score'] == 100).any():
    output = pd.DataFrame(recommendation(prod_id),columns = ["Recommended␣
 ↪Outfit Combination:"]).reset_index(drop = True)
    print(tabulate(output, showindex=False, headers=df.columns))

# If a perfect match is not found then similar product IDs are suggested
else:
   print(f'{prod_id} not found\n\nSuggested Product IDs {matches}')
```

```
Outfit type for product id 01DMBRYVA2ZFDYRYY5TRQZJTBD is : shoe

Matching Outfit ID is : 01DMHRX35M2DPVYVQ1PNER4S4B

outfit_id
------------------------------------------------------------
onepiece: Chemelle Midi Dress (01DMBRYVA2Q2ST7MNYR6EEY4TK)
shoe: Penelope Mid Cap Toe Pump (01DMBRYVA2ZFDYRYY5TRQZJTBD)
accessory1: Crystal Clutch (01DMHCNT41E14QWP503V7CT9G6)
```

4 Product Description Input

if user does not have a Prouct ID, they can enter the prodcut's Brand and/or description in the cell below, and get the corresponding outfit recommendations

[25]:
```
# Brand and Description input
## ENTER Brand and product description information below
## In case any information is missing, jut enter blank string, i.e., ''

brand = "Reformation"
description = "Sexy silky, a-line mini skirt zipper Benson skirt"
```

4.0.1 Cleaning the input text

[26]:
```
# Stores the descriotion in a temporary test variable
test_desc = description

# Remove Punctuations from the input text
punctuation = "!@#$%^&*()_+<>?:.,;"

for c in test_desc:
    if c in punctuation:
        test_desc = test_desc.replace(c, "")

# Remove Stopwords from input text
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(test_desc)
test_desc = [w for w in word_tokens if not w in stop_words]
test_desc = []
for w in word_tokens:
    if w not in stop_words:
        test_desc.append(w)
test_desc = ' '.join(test_desc)
test_desc
```

[26]:
```
'Sexy silky a-line mini skirt zipper Benson skirt'
```

4.0.2 Determine Outfit Type

- Find the most relevant words (eg: common nouns) associated with each outfit item type
- When a test query/document is submitted on user interface, this query is parsed to check with what outfit item type(s) it matches using regular expression
- Once we know the possible outfit item types, we find the most similar product by filtering dataset on these outfit item types only.

[27]:
```
# Regex to identify right category for filtering
shoe=r'(boot|sandal|pump|mule|sneaker|loafer|slingback|flat|slide|croc)'
top=r'(shirt|sweater|top|blouse|turtleneck|jersey|tee|bodysuit|neck|sleeve|jacket|coat|cardiga
bottom=r'(leg|pant|skirt|jean|rise|midi|short|trouser)'
onepiece =␣
 ↪r'(dress|jumpsuit|wrap|stretch|maxi|midi|larina|francoise|polka|shirt|sweater|top|blouse|tu
accessory1=r'(bag|tote|croc|tori|clutch|mini|scarf|cabinet|top|bucket|backpack|hammock|belt|la
accessory2=r'(bag|tote|croc|tori|clutch|mini|scarf|cabinet|top|bucket|backpack|hammock|belt|la
accessory3= r'(coat)'
```

[28]:
```
# Determining Potential outfit categories using the above Regex

# outfitTypes is a dictionary to map 'outfit item type' with it's regular␣
 ↪expression created above
outfitTypes={'top':top,'bottom':bottom,'shoe':shoe,'onepiece':
 ↪onepiece,'accessory1':accessory1,'accessory2':accessory2,'accessory3':
 ↪accessory3}
outfits = [outfit for outfit in outfitTypes if re.
 ↪search(outfitTypes[outfit],test_desc,flags=re.IGNORECASE)]
outfits
```

[28]:
```
['bottom', 'onepiece', 'accessory1', 'accessory2']
```

[29]:
```
# Filtering data to the outfit categories found above
## We use this filtering to search in a subset of the dataframe (only for the␣
 ↪categories identified)
## This will make search faster and give more accurate results

outfit_data = data.copy()

if outfits != []:
    outfit_data = data[data['outfit_item_type'].isin(outfits)].reset_index(drop␣
 ↪= True)
outfit_data.head()
```

[29]:
```
                    outfit_id                  product_id outfit_item_type  \
0  01DDBHC62ES5K80P0KYJ56AM2T  01DMBRYVA2P5H24WK0HTK4R0A1           bottom
1  01DDBHC62ES5K80P0KYJ56AM2T  01DMBRYVA2S5T9W793F4CY41HE       accessory1
2  01DMHCX50CFX5YNG99F3Y65GQW  01DMBRYVA2P5H24WK0HTK4R0A1           bottom
```

```
3  01DMHCX50CFX5YNG99F3Y65GQW  01DMHCNT41E14QWP503V7CT9G6       accessory1
4  01DMHRX35M2DPVYVQ1PNER4S4B  01DMBRYVA2Q2ST7MNYR6EEY4TK         onepiece

                 brand               product_full_name product description
0        Eileen Fisher                  Slim Knit Skirt        A nice skirt
1  kate spade new york  medium margaux leather satchel          A nice bag
2        Eileen Fisher                  Slim Knit Skirt        A nice skirt
3                 Nina                   Crystal Clutch       A nice clutch
4            Equipment               Chemelle Midi Dress        A nice dress
```

4.0.3 Determine Brand

- Find close matches to the brand entered by the user, based on FuzzyWuzzy library, with cutoff of 85
- If a close match is found, we filter the outfit data further for the given brand

[30]:
```
# Filtering data with specific brand if a brand match is found in the data

brand_data = outfit_data.copy()

if brand != "":
    brand_data['fuzz_score'] = brand_data["brand"].str.lower().apply(lambda x:␣
 ↪fuzz.ratio(x,brand.lower()))
    brand_data = brand_data[brand_data['fuzz_score'] > 85].
 ↪drop('fuzz_score',axis=1)

brand_data.head()
```

[30]:
```
                     outfit_id                  product_id outfit_item_type  \
8   01DQ63P636Q4BQVCKT6Z4S41G5  01DPKMGJ33SDFXM7XHGPQJWQ12           bottom
10  01DQ86EH3GMXAVKNECH2Z6FCSV  01DPKNJ6J1NQPQ1D3DBKWK5ARS         onepiece
12  01DQ8KWVX1GBJTPTVDAC6NQ9B4  01DPKNJA2K022V3KP077611MKC           bottom
14  01DQ8ME3M3QS9MQGZCQHXDHE1R  01DPKMH0D252JKMAA27MFCT5GM           bottom
15  01DQ8MQAVBFSGHJXCF5JCYJ7A6  01DPKMKG14KT68YQY0MWA1CAA8           bottom

          brand               product_full_name  \
8   Reformation                    Benson Skirt
10  Reformation                  Rosamund Dress
12  Reformation                    Everett Pant
14  Reformation                     Marlon Pant
15  Reformation  Julia Crop High Cigarette Jean

                                   product description
8   Sexy silky. This is an a-line mini skirt with …
10  Let your dress do the work. This is a midi len…
12  It's cold. Put some pants on. This is a high r…
14  Let your pants do the talking. This is a slim …
```

```
15  Better butts. This is a high rise, rigid jean …
```

4.0.4 Declaring Relevant functions

[31]:
```
## Finding most similar document's product ID and relevant outfit combination␣
 ↪for the dataset filtered  from above
## We calculate 2 separate cosine similarity scores: one for Description and 1␣
 ↪for Product_Full_Name

def subset_prodid(test_doc):
    df2 = brand_data.copy()

    test=[]
    score_desc=[]
    score_full_name = []
    for idx,row in df2.iterrows():
        descr=row['product description']
        full_name = row['product_full_name']
        org=nlp(descr)
        org2 = nlp(full_name)
        score_desc.append(test_doc.similarity(org))
        score_full_name.append(test_doc.similarity(org2))
        test.append(test_doc)
    df2['test_doc']=test
    df2['score_sim_full_name'] = score_full_name
    df2['score_sim_desc']=score_desc

    # If the user input is longer than 20 characters, we give higher weightage␣
 ↪to Description
    if (len_test_desc>20):
        df2['score_sim'] = 0.7*df2['score_sim_desc'] + 0.3*␣
 ↪df2['score_sim_full_name']

    # If the user input is less than 20 characters, we take the maximum of␣
 ↪Similarity scores received from Description and Full_name
    # This gives higher weightage to Full_Name as Full Name column is generally␣
 ↪15-20 characters
    else:
        df2['score_sim'] = df2[['score_sim_desc','score_sim_full_name']].
 ↪max(axis = 1)
    df2 = df2.sort_values(by='score_sim',ascending=False).reset_index()
    df2.head()
    tar_prodid=df2.loc[0,"product_id"]
    tar_prodid
    return tar_prodid
```

```
# This function generates a word2vec vector, does a weighted average TF-IDF␣
↪score, to give higher weightage to relevant words

def word2vec_TFIDF(data,X):
    X = X.transform(data)

    tf_idf_lookup_table = pd.DataFrame(X.toarray(), columns=vectorizer.
↪get_feature_names())


    DOCUMENT_SUM_COLUMN = "DOCUMENT_TF_IDF_SUM"

    # sum the tf idf scores for each document
    tf_idf_lookup_table[DOCUMENT_SUM_COLUMN] = tf_idf_lookup_table.sum(axis=1)
    available_tf_idf_scores = tf_idf_lookup_table.columns # a list of all the␣
↪columns we have
    available_tf_idf_scores = list(map( lambda x: x.lower(),␣
↪available_tf_idf_scores)) # lowercase everything

    row_vectors = []
    for idx, row in enumerate(data): # iterate through each review
        tokens = nlp(row) # have spacy tokenize the review text

        # initially start a running total of tf-idf scores for a document
        total_tf_idf_score_per_document = 0

        # start a running total of initially all zeroes (300 is picked since␣
↪that is the word embedding size used by word2vec)
        running_total_word_embedding = np.zeros(300)
        for token in tokens: # iterate through each token

        # if the token has a pretrained word embedding it also has a tf-idf␣
↪score
            if token.has_vector and token.text.lower() in␣
↪available_tf_idf_scores:

                tf_idf_score = tf_idf_lookup_table.loc[idx, token.text.lower()]
                #print(f"{token} has tf-idf score of {tf_idf_lookup_table.
↪loc[idx, token.text.lower()]}")
                running_total_word_embedding += tf_idf_score * token.vector

                total_tf_idf_score_per_document += tf_idf_score

        # divide the total embedding by the total tf-idf score for each document
        document_embedding = running_total_word_embedding /␣
↪total_tf_idf_score_per_document
```

```
        row_vectors.append(document_embedding)
    return row_vectors
```

5 Method 1 - Using Word Embeddings from Spacy

6 (Final Output)

[32]:
```
# Calculating the total # of characters of the input query
len_test_desc = sum(len(word) for word in test_desc)

# Calculating scores on filtered subset of the original dataframe and returns␣
↪outfit recommendations
prod_id = subset_prodid(nlp(test_desc))
output = pd.DataFrame(recommendation(prod_id),columns = ["Recommended Outfit␣
↪Combination:"]).reset_index(drop = True)
print(tabulate(output, showindex=False, headers=df.columns))
```

```
Outfit type for product id 01DPKMGJ33SDFXM7XHGPQJWQ12 is : bottom

Matching Outfit ID is : 01DQ63P636Q4BQVCKT6Z4S41G5

outfit_id
-------------------------------------------------------------
shoe: Pointed-toe flats in suede (01DPCRZWX4S2Z8Q5HYDFM4HNEG)
top: Ashlynn Blouse (01DPET2NWSA221STZF740BZ9SW)
bottom: Benson Skirt (01DPKMGJ33SDFXM7XHGPQJWQ12)
```

7 Other Methods Tried

8 Method 2 - Using Weighted Average Word Embeddings

[13]:
```
brand_data.head(1)
```

[13]:
```
                    outfit_id                  product_id outfit_item_type  \
0  01DDBHC62ES5K80P0KYJ56AM2T  01DMBRYVA2ZFDYRYY5TRQZJTBD             shoe

        brand          product_full_name product description
0  Tory Burch  Penelope Mid Cap Toe Pump         A nice shoe
```

[19]:
```
# pd.set_option('display.max_colwidth', None)

data_list = list(brand_data['product_full_name'] + ' ' + brand_data['product␣
↪description'])

vectorizer = TfidfVectorizer()
```

```
X = vectorizer.fit(data_list)
train_vec = word2vec_TFIDF(data_list,X)

test_vec = word2vec_TFIDF([test_desc],X)
test_vec = [list(test_vec[0])]

sim_score = []

for i in range(brand_data.shape[0]):
     train_vec[i] = list(train_vec[i])
     score =  float(cosine_similarity([train_vec[i]],test_vec))
     sim_score.append(score)

max_row = sim_score.index(max(sim_score))
recommendation(data.loc[max_row,"product_id"])
```

```
Outfit type for product id 01DMHCNT41E14QWP503V7CT9G6 is : accessory1

Matching Outfit ID is : 01DMHRX35M2DPVYVQ1PNER4S4B
```

[19]:
```
8     onepiece: Chemelle Midi Dress (01DMBRYVA2Q2ST7…
9     shoe: Penelope Mid Cap Toe Pump (01DMBRYVA2ZFD…
10    accessory1: Crystal Clutch (01DMHCNT41E14QWP50…
dtype: object
```

9 Method 3 - One Hot Encoding + Cosine Similarity

[20]:
```
vectorizer = CountVectorizer(stop_words="english", binary=True)
H = vectorizer.fit(data_list)
train_vec = H.transform(data_list)
train_vec_df = pd.DataFrame(train_vec.toarray(), columns=vectorizer.
 ↪get_feature_names())
train_vec_df.head()
```

[20]:
```
   01  06  100  100mm  105  105mm  12  15mm  1774  19  …  wrapped  wraps  \
0   0   0    0      0    0      0   0     0     0   0  …        0      0
1   0   0    0      0    0      0   0     0     0   0  …        0      0
2   0   0    0      0    0      0   0     0     0   0  …        0      0
3   0   0    0      0    0      0   0     0     0   0  …        0      0
4   0   0    0      0    0      0   0     0     0   0  …        0      0

   www  years  young  zebra  zip  zipper  zippers  zoom
0    0      0      0      0    0       0        0     0
1    0      0      0      0    0       0        0     0
2    0      0      0      0    0       0        0     0
```

```
3    0         0         0         0      0          0           0        0
4    0         0         0         0      0          0           0        0

[5 rows x 1269 columns]
```

```
[23]: test_vec = H.transform(test_desc).toarray()
      test_vec = [list(test_vec[0])]
```

```
[24]: sim_score = []

      for i in range(brand_data.shape[0]):
          train_vec = list(train_vec_df.iloc[i,:])
          score =  float(cosine_similarity([train_vec],test_vec))
          sim_score.append(score)

      max_row = sim_score.index(max(sim_score))
      recommendation(brand_data.loc[max_row,"product_id"])
```

```
Outfit type for product id 01DMBRYVA2ZFDYRYY5TRQZJTBD is : shoe

Matching Outfit ID is : 01DDBHC62ES5K80P0KYJ56AM2T
```

```
[24]: 0    bottom: Slim Knit Skirt (01DMBRYVA2P5H24WK0HTK…
      1    top: Rib Mock Neck Tank (01DMBRYVA2PEPWFTT7RMP…
      2    accessory1: medium margaux leather satchel (01…
      3    shoe: Penelope Mid Cap Toe Pump (01DMBRYVA2ZFD…
      dtype: object
```

Cleaning

April 4, 2021

1 Group Project (Team Purple)

[1]:
```
# Importing relevant libraries
import pandas as pd
import numpy as np
import re
from collections import Counter
import nltk
import spacy
import functools
from sklearn.feature_extraction.text import TfidfVectorizer
nlp = spacy.load("en_core_web_sm")
```

2 Data Exploration

[2]:
```
# Reading data into pandas
full_data = pd.read_csv("Full Data.csv")
tagged_data = pd.read_csv("Tagged Product Attributes.csv")
```

2.0.1 Preparing Full Data File

[3]:
```
full_data.head(2)
```

[3]:
```
                   product_id            brand     mpn    product_full_name  \
0  01DSE9TC2DQXDG6GWKW9NMJ416  Banana Republic  514683     Ankle-Strap Pump
1  01DSE9SKM19XNA6SJP36JZC065  Banana Republic  526676  Petite Tie-Neck Top

                                         description brand_category  \
0  A modern pump, in a rounded silhouette with an…        Unknown
1  Dress it down with jeans and sneakers or dress…        Unknown

                         created_at                        updated_at deleted_at  \
0  2019-11-11 22:37:15.719107+00  2019-12-19 20:40:30.786144+00        NaN
1  2019-11-11 22:36:50.682513+00  2019-12-19 20:40:30.786144+00        NaN

                                brand_canonical_url  \
0  https://bananarepublic.gap.com/browse/product…
1  https://bananarepublic.gap.com/browse/product…

                                              details            labels  \
0  A modern pump, in a rounded silhouette with an…  {"Needs Review"}
1  Dress it down with jeans and sneakers or dress…  {"Needs Review"}

   bc_product_id
0            NaN
1            NaN
```

[4]:
```
# Deleting irrelevant columns in full data
full_data.drop(['mpn', 'created_at', 'updated_at', 'deleted_at',
                'brand_canonical_url', 'bc_product_id'],axis = 1,inplace = True)
```

[5]:
```
# Checking NA values
full_data.isnull().sum()
```

[5]:
```
product_id              0
brand                   0
product_full_name       0
description          7974
brand_category        238
details              9866
labels                  0
dtype: int64
```

[6]:
```
# Replacing NA values with Unknown
full_data=full_data.fillna("Unknown")
```

[7]:
```
# Dropping duplicate rows based on product id (This makes product_id unique)
full_data = full_data.drop_duplicates(subset="product_id")
full_data.head()
```

[7]:
```
                   product_id              brand  \
0  01DSE9TC2DQXDG6GWKW9NMJ416    Banana Republic
1  01DSE9SKM19XNA6SJP36JZC065    Banana Republic
2  01DSJX8GD4DSAP76SPR85HRCMN              Loewe
3  01DSJVKJNS6F4KQ1QM6YYK9AW2           Converse
4  01DSK15ZD4D5A0QXA8NSD25YXE  Alexander McQueen

                                 product_full_name  \
0                                 Ankle-Strap Pump
1                              Petite Tie-Neck Top
2                  52MM Padded Leather Round Sunglasses
3  Baby's & Little Kid's All-Star Two-Tone Mid-To…
4                            64MM Rimless Sunglasses
```

```
                                        description  \
0  A modern pump, in a rounded silhouette with an…
1  Dress it down with jeans and sneakers or dress…
2    Padded leather covers classic round sunglasses.
3  The iconic mid-top design gets an added dose o…
4  Hexagonal shades offer a rimless view with int…

                                     brand_category  \
0                                           Unknown
1                                           Unknown
2  JewelryAccessories/SunglassesReaders/RoundOval…
3  JustKids/Shoes/Baby024Months/BabyGirl,JustKids…
4     JewelryAccessories/SunglassesReaders/RoundOval

                                            details            labels
0  A modern pump, in a rounded silhouette with an…  {"Needs Review"}
1  Dress it down with jeans and sneakers or dress…  {"Needs Review"}
2  100% UV protection\nCase and cleaning cloth in…  {"Needs Review"}
3  Canvas upper\nRound toe\nLace-up vamp\nSmartFO…  {"Needs Review"}
4  100% UV protection\nGradient lenses\nAdjustabl…  {"Needs Review"}
```

2.0.2 Preparing Tagged Products File

[8]:
```
# Retaining only the relevant labels
tagged_data = tagged_data[tagged_data['attribute_name'].isin(["style",␣
↪"occasion","fit","Primary Color"])]
```

[9]:
```
# Converting misspelled labels to a standard format
tagged_data['attribute_value'] = tagged_data['attribute_value'].
↪replace({"semifitted": "Semi-Fitted",
                                                                    ␣
↪"straightregular": "Straight / Regular",
                                                                    ␣
↪"fittedtailored": "Fitted / Tailored",
                                                                     "daytonight":
↪ "Day to Night",
                                                                     "nightout":␣
↪"Night Out",
                                                                    ␣
↪"businesscasual": "Business Casual",
                                                                         })
tagged_data['attribute_name'] = tagged_data['attribute_name'].replace({"Primary␣
↪Color": "color",
                                                                         })
tagged_data.head()
```

```
[9]:                      product_id             product_color_id attribute_name  \
     1  01DVA7QRXM928ZM0WWR7HFNTC1  01DVA7QRXXR9F0TWVE1HMC5ZQ3          color
     2  01DPGV4YRP3Z8J85DASGZ1Y99W  01DPGVGBK6YGNYGNF2S6FSH02T          style
     3  01E1JM43NQ3H17PB22EV3074NX  01E1JM5WFWWCCCH3JTTTCYQCEQ          style
     6  01E2C3YN4KQ36A0REWZJ89ZN73  01E2C3YN56ZCJ8TN45V3EC8CPS          color
     8  01E223GDRKR84THXZ54GJEW60Y  01E223GKFAFZ5HTVBQJ82TAEZH            fit

       attribute_value          file
     1          Blacks  initial_tags
     2          Casual  initial_tags
     3          Modern  initial_tags
     6          Blacks  initial_tags
     8     Semi-Fitted  initial_tags
```

```
[10]: # Grouping attribute value based on product id

      tagged_data['attribute_value'] = tagged_data['attribute_value'].str.lower()
      tagged_data['attribute_name'] = tagged_data['attribute_name'].str.lower()

      tagged_data1 = tagged_data.groupby(['product_id'])['attribute_value'].
       ↪apply(set).reset_index()
      tagged_data2 = tagged_data.groupby(['product_id'])['attribute_name'].apply(set).
       ↪reset_index()

      tagged_data3 = pd.merge(tagged_data2,tagged_data1,on='product_id', how='left')
      tagged_data3.head()
```

```
[10]:                     product_id                   attribute_name  \
     0  01DPC9GSTT72KHNN0MNDNKH7RD             {occasion, style}
     1  01DPCB2KEAVXXKFVM7FXBNE4VY      {color, occasion, style}
     2  01DPCDEF6SYX2E1NT5X7HJBFGY                {color, style}
     3  01DPCG1C1P0MQAV9NMS3N1TDAA  {color, occasion, style, fit}
     4  01DPCHNEW5F2RHJQ3NJMVPK6SE  {color, occasion, style, fit}

                                  attribute_value
     0    {business casual, work, classic, day to night}
     1  {day to night, browns, blacks, work, modern, w…
     2  {burgundies, classic, beiges, blacks, pinks, g…
     3  {glam, semi-fitted, weekend, romantic, night o…
     4  {burgundies, classic, day to night, casual, an…
```

```
[11]: # Merge the attribute value as label in the full_data
      full_data = pd.merge(full_data,tagged_data3,on='product_id', how='left')
      full_data.drop("labels",axis = 1,inplace = True)
      full_data.rename(columns = {'attribute_value':'labels','attribute_name':
       ↪'category'}, inplace = True)
      full_data.head()
```

```
[11]:                    product_id              brand  \
      0  01DSE9TC2DQXDG6GWKW9NMJ416    Banana Republic
      1  01DSE9SKM19XNA6SJP36JZC065    Banana Republic
      2  01DSJX8GD4DSAP76SPR85HRCMN              Loewe
      3  01DSJVKJNS6F4KQ1QM6YYK9AW2           Converse
      4  01DSK15ZD4D5A0QXA8NSD25YXE  Alexander McQueen

                                   product_full_name  \
      0                                Ankle-Strap Pump
      1                             Petite Tie-Neck Top
      2               52MM Padded Leather Round Sunglasses
      3  Baby's & Little Kid's All-Star Two-Tone Mid-To…
      4                           64MM Rimless Sunglasses

                                         description  \
      0  A modern pump, in a rounded silhouette with an…
      1  Dress it down with jeans and sneakers or dress…
      2    Padded leather covers classic round sunglasses.
      3  The iconic mid-top design gets an added dose o…
      4  Hexagonal shades offer a rimless view with int…

                                      brand_category  \
      0                                          Unknown
      1                                          Unknown
      2  JewelryAccessories/SunglassesReaders/RoundOval…
      3  JustKids/Shoes/Baby024Months/BabyGirl,JustKids…
      4     JewelryAccessories/SunglassesReaders/RoundOval

                                             details category labels
      0  A modern pump, in a rounded silhouette with an…      NaN    NaN
      1  Dress it down with jeans and sneakers or dress…      NaN    NaN
      2  100% UV protection\nCase and cleaning cloth in…      NaN    NaN
      3  Canvas upper\nRound toe\nLace-up vamp\nSmartFO…      NaN    NaN
      4  100% UV protection\nGradient lenses\nAdjustabl…      NaN    NaN
```

```
[12]: # Creating a new df with labels
      df = full_data[full_data['labels'].notnull()]
      df.head()
```

```
[12]:                     product_id             brand  \
      15  01E5ZXP5H0BTEZT9QD2HRZJ47A            A.L.C.
      33  01DSECZPAGJJC1EDC79JRBF4WK   Banana Republic
      43  01E607BHRQAJDZ76MJFN7RPRK1      Simon Miller
      44  01E5ZXJ6G03R7177X723CT04W0            A.L.C.
      70  01E6074PQA697JZ1SBM6NM8TBG      Simon Miller

                                product_full_name  \
15  Lennox High Waist Cotton & Linen Pants
33                   Mock-Neck Sweater Top
43                      Rost Belted Shorts
44             Minelli Silk Sleeveless Top
70     Nepa Mismatched Button Rib Cardigan

                                          description brand_category  \
15  High-rise trousers tailored from a cool Italia…        Unknown
33  Designed to be worn with high-waisted bottoms,…        Unknown
43  Cinched at the natural waist and pleated for f…        Unknown
44  Painterly brushes of color that convey the flu…        Unknown
70  The West Coast-based label channels beachy vib…        Unknown

                                              details  \
15  True to size. High rise.\n31" inseam; 14" leg …
33  Designed to be worn with high-waisted bottoms,…
43  True to size. XS=0-2, S=4-6, M=6-8, L=8-10, XL…
44  True to size.\n25 1/2" length (size Medium)\nF…
70  True to size. XS=0-2, S=4-6, M=6-8, L=8-10, XL…

                            category  \
15         {occasion, fit, style}
33  {color, occasion, style, fit}
43         {occasion, fit, style}
44         {occasion, fit, style}
70         {occasion, fit, style}

                                               labels
15  {classic, work, semi-fitted, modern, business …
33  {classic, day to night, blacks, whites, work, …
43  {oversized, casual, androgynous, modern, weeke…
44  {day to night, casual, modern, boho, relaxed, …
70  {fitted / tailored, day to night, casual, mode…
```

3 Data Cleaning

3.0.1 1. Stopword removal

[13]:
```python
# Stopwords including custom stopwords
from nltk.corpus import stopwords

custom_stopwords =  [ 'ever','always','every','even','though','here','was'
                      'there','ve','re', "'m","'ve", "n't",'not','yourself',
                      'yup','yours','you','yet','yes','yep','or','yeah','yea',
                      'nor','no',"weren't", "mustn't","needn't","shouldn't",
                      "won't","wouldn't","weren't","wasn't","shan't","mightn't",
                       "isn't", "haven't","hasn't","doesn't","aren't","couldn't",␣
↪"don't","didn't","hadn't","mustn't",'on','your','yet','why','whose','we']


stopwords = stopwords.words('english') + custom_stopwords
stopwords[0:10]
```

[13]:
```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

[14]:
```
# Removing stopwords from details and description columns
df['details'] = df["details"].apply(lambda x: ' '.join([word for word in x.
↪split() if word.lower() not in (stopwords)]))
df['description'] = df["description"].apply(lambda x: ' '.join([word for word␣
↪in x.split() if word.lower() not in (stopwords)]))
df.head(2)
```

```
C:\Users\jayan\AppData\Local\Continuum\anaconda3\lib\site-
packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\jayan\AppData\Local\Continuum\anaconda3\lib\site-
packages\ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[14]:
```
                    product_id            brand  \
15  01E5ZXP5H0BTEZT9QD2HRZJ47A           A.L.C.
33  01DSECZPAGJJC1EDC79JRBF4WK  Banana Republic

                         product_full_name  \
15  Lennox High Waist Cotton & Linen Pants
33                   Mock-Neck Sweater Top

                                          description brand_category  \
15  High-rise trousers tailored cool Italian cotto…        Unknown
33  Designed worn high-waisted bottoms, oh-so-now …        Unknown
```

```
                                             details  \
15  True size. High rise. 31" inseam; 14" leg open…
33  Designed worn high-waisted bottoms, oh-so-now …

                         category  \
15        {occasion, fit, style}
33  {color, occasion, style, fit}

                                              labels
15  {classic, work, semi-fitted, modern, business …
33  {classic, day to night, blacks, whites, work, …
```

3.0.2 2. Replacing numbers using regex

[15]:
```
df['details'] = df['details'].apply(lambda x: re.
 ↪sub(r'$\d+\W+|\b\d+\b|\W+\d+$','Number', x, flags=re.IGNORECASE))
df['description'] = df['description'].apply(lambda x: re.
 ↪sub(r'$\d+\W+|\b\d+\b|\W+\d+$','Number', x, flags=re.IGNORECASE))
```

```
C:\Users\jayan\AppData\Local\Continuum\anaconda3\lib\site-
packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
C:\Users\jayan\AppData\Local\Continuum\anaconda3\lib\site-
packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

3.0.3 3. Punctuation Removal (Using Regex)

A lot of punctuations can make it difficult during word token creation and lemmatization process. Thus, we remove common punctuations like !,* etc. as they do not add much value. However, hyphen and hash can lead to some interesting word combinations and special word meanings so we retain these 2 punctuations

[16]:
```
# Removing punctuations except for hyphens and hashtag
import string
remove = string.punctuation
remove = remove.replace("-", "") # don't remove hyphens
```

```
pattern = r'[{}]'.format(remove) # create the pattern
df['details'] = df['details'].apply(lambda x: re.sub(pattern,'',x))
df['description'] = df['description'].apply(lambda x: re.sub(pattern,'',x))
df.head()
```

```
C:\Users\jayan\AppData\Local\Continuum\anaconda3\lib\site-
packages\ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\jayan\AppData\Local\Continuum\anaconda3\lib\site-
packages\ipykernel_launcher.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
```

[16]:

```
                    product_id            brand  \
15  01E5ZXP5H0BTEZT9QD2HRZJ47A           A.L.C.
33  01DSECZPAGJJC1EDC79JRBF4WK  Banana Republic
43  01E607BHRQAJDZ76MJFN7RPRK1     Simon Miller
44  01E5ZXJ6G03R7177X723CT04W0           A.L.C.
70  01E6074PQA697JZ1SBM6NM8TBG     Simon Miller

                         product_full_name  \
15  Lennox High Waist Cotton & Linen Pants
33                    Mock-Neck Sweater Top
43                       Rost Belted Shorts
44               Minelli Silk Sleeveless Top
70     Nepa Mismatched Button Rib Cardigan

                                          description brand_category  \
15  High-rise trousers tailored cool Italian cotto…        Unknown
33  Designed worn high-waisted bottoms oh-so-now m…        Unknown
43  Cinched natural waist pleated fullness long wo…        Unknown
44  Painterly brushes color convey flutter butterf…        Unknown
70  West Coast-based label channels beachy vibes c…        Unknown

                                              details  \
15  True size High rise Number inseam Number leg o…
33  Designed worn high-waisted bottoms oh-so-now m…
43  True size XSNumber-Number SNumber-Number MNumb…
```

```
44  True size Number NumberNumber length size Medi…
70  True size XSNumber-Number SNumber-Number MNumb…

                           category  \
15         {occasion, fit, style}
33  {color, occasion, style, fit}
43         {occasion, fit, style}
44         {occasion, fit, style}
70         {occasion, fit, style}

                                               labels
15  {classic, work, semi-fitted, modern, business …
33  {classic, day to night, blacks, whites, work, …
43  {oversized, casual, androgynous, modern, weeke…
44  {day to night, casual, modern, boho, relaxed, …
70  {fitted / tailored, day to night, casual, mode…
```

[17]:
```
# Replacing , and \ by " " for brand category

df['brand_category'] = df['brand_category'].apply(lambda x: re.sub(r'(\,|\/)','␣
↪',x))
df = df.reset_index(drop = True)
df.head()
```

```
C:\Users\jayan\AppData\Local\Continuum\anaconda3\lib\site-
packages\ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[17]:
```
                   product_id            brand  \
0  01E5ZXP5H0BTEZT9QD2HRZJ47A           A.L.C.
1  01DSECZPAGJJC1EDC79JRBF4WK  Banana Republic
2  01E607BHRQAJDZ76MJFN7RPRK1     Simon Miller
3  01E5ZXJ6G03R7177X723CT04W0           A.L.C.
4  01E6074PQA697JZ1SBM6NM8TBG     Simon Miller

                      product_full_name  \
0  Lennox High Waist Cotton & Linen Pants
1                    Mock-Neck Sweater Top
2                       Rost Belted Shorts
3                Minelli Silk Sleeveless Top
4     Nepa Mismatched Button Rib Cardigan
```

```
                                         description brand_category  \
0  High-rise trousers tailored cool Italian cotto…        Unknown
1  Designed worn high-waisted bottoms oh-so-now m…        Unknown
2  Cinched natural waist pleated fullness long wo…        Unknown
3  Painterly brushes color convey flutter butterf…        Unknown
4  West Coast-based label channels beachy vibes c…        Unknown

                                             details  \
0  True size High rise Number inseam Number leg o…
1  Designed worn high-waisted bottoms oh-so-now m…
2  True size XSNumber-Number SNumber-Number MNumb…
3  True size Number NumberNumber length size Medi…
4  True size XSNumber-Number SNumber-Number MNumb…

                          category  \
0         {occasion, fit, style}
1  {color, occasion, style, fit}
2         {occasion, fit, style}
3         {occasion, fit, style}
4         {occasion, fit, style}

                                              labels
0  {classic, work, semi-fitted, modern, business …
1  {classic, day to night, blacks, whites, work, …
2  {oversized, casual, androgynous, modern, weeke…
3  {day to night, casual, modern, boho, relaxed, …
4  {fitted / tailored, day to night, casual, mode…
```

3.0.4 4. Tokenization and Lemmatization

Creating word tokens on which we run lemmatization to bring different words to their base form. The reason for choosing lemmatization here over stemming is that lemmatization using a dictionary based on lemma and hence can assign the words correctly to their root forms unlike stemming where words may be cut-down and do not have a real meaning.

[18]:
```python
# Lemmatizing details and description

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

def lemmatization_sentences(sentence):
    tokens = sentence.split()
    lemma = [lemmatizer.lemmatize(token) for token in tokens]
    return ' '.join(lemma)
```

```
df['description'] = df['description'].apply(lambda x:␣
↪lemmatization_sentences(x))
df['details'] = df['details'].apply(lambda x: lemmatization_sentences(x))
df.head(2)
```

```
[18]:                    product_id           brand  \
0  01E5ZXP5H0BTEZT9QD2HRZJ47A          A.L.C.
1  01DSECZPAGJJC1EDC79JRBF4WK  Banana Republic

                        product_full_name  \
0  Lennox High Waist Cotton & Linen Pants
1                      Mock-Neck Sweater Top

                                        description brand_category  \
0  High-rise trouser tailored cool Italian cotton…        Unknown
1  Designed worn high-waisted bottom oh-so-now mo…        Unknown

                                            details  \
0  True size High rise Number inseam Number leg o…
1  Designed worn high-waisted bottom oh-so-now mo…

                    category  \
0        {occasion, fit, style}
1  {color, occasion, style, fit}

                                             labels
0  {classic, work, semi-fitted, modern, business …
1  {classic, day to night, blacks, whites, work, …
```

3.1 Create Dataframe for each category

```
[19]: # Creating 4 dataframes
for var in ['fit','occasion','color','style']:
    df['{}'.format(var)] = [var in i for i in df['category']]


df_fit = df[df['fit']==True].
↪drop(['category','fit','occasion','color','style'],axis = 1)
df_occasion = df[df['occasion']==True].
↪drop(['category','fit','occasion','color','style'],axis = 1)
df_color = df[df['color']==True].
↪drop(['category','fit','occasion','color','style'],axis = 1)
df_style = df[df['style']==True].
↪drop(['category','fit','occasion','color','style'],axis = 1)
```

```
# Creating fit dataframe

df_fit = df_fit.reset_index(drop = True)
df_fit1 = pd.DataFrame(df_fit['labels'].values.tolist()) \
         .rename(columns = lambda x: 'labels{}'.format(x+1)) \
          .fillna('Unknown')
df_fit = df_fit.merge(df_fit1,left_on = df_fit.index,right_on = df_fit1.
 ↪index,how = 'left')
df_fit = df_fit.drop("key_0",axis = 1)


# Creating occasion dataframe

df_occasion = df_occasion.reset_index(drop = True)
df_occasion1 = pd.DataFrame(df_occasion['labels'].values.tolist()) \
         .rename(columns = lambda x: 'labels{}'.format(x+1)) \
          .fillna('Unknown')
df_occasion = df_occasion.merge(df_occasion1,left_on = df_occasion.
 ↪index,right_on = df_occasion1.index,how = 'left')
df_occasion = df_occasion.drop("key_0",axis = 1)


# Creating color dataframe

df_color = df_color.reset_index(drop = True)
df_color1 = pd.DataFrame(df_color['labels'].values.tolist()) \
         .rename(columns = lambda x: 'labels{}'.format(x+1)) \
          .fillna('Unknown')
df_color = df_color.merge(df_color1,left_on = df_color.index,right_on =␣
 ↪df_color1.index,how = 'left')
df_color = df_color.drop("key_0",axis = 1)

# Creating style dataframe
df_style = df_style.reset_index(drop = True)
df_style1 = pd.DataFrame(df_style['labels'].values.tolist()) \
         .rename(columns = lambda x: 'labels{}'.format(x+1)) \
          .fillna('Unknown')
df_style = df_style.merge(df_style1,left_on = df_style.index,right_on =␣
 ↪df_style1.index,how = 'left')
df_style = df_style.drop("key_0",axis = 1)
```

[20]:
```
df_color.head()
```

[20]:
```
                   product_id              brand  \
0  01DSECZPAGJJC1EDC79JRBF4WK    Banana Republic
1  01DVA59VHYAPT4PVX32NXW91G5               Tibi
2  01DVA4XY7A0QMMSK3V3SBR52J9  Alexandre Birman
```

```
3  01DVBP9AHVQTZXJSBNJ0N2NYJP           Khaite
4  01DVBR93Y7KANZE3C09YCTVXDF  Lauren Manoogian

                    product_full_name  \
0                  Mock-Neck Sweater Top
1                     Juan Embossed Mules
2  Clarita Bow-Embellished Suede Sandals
3                     Leather ankle boots
4                      Alpaca-blend scarf

                                          description  \
0  Designed worn high-waisted bottom oh-so-now mo…
1  Tibis Juan embossed mule made shiny black leat…
2  Alexandre Birmans Clarita sandal quickly risen…
3  Heel measure approximately 50mm Number inch Bl…
4  Brown alpaca-blend Number alpaca Number polyam…

                 brand_category  \
0                       Unknown
1              women:SHOES:MULES
2            women:SHOES:SANDALS
3           Shoes   Boots   Ankle
4  Accessories   Scarves   Scarves

                                              details  \
0  Designed worn high-waisted bottom oh-so-now mo…
1  seen Pre-Fall ‘Number runway Heel measure appr…
2  Heel height measure approximately 50mm Number …
3     Fits true size take normal size Italian sizing
4       item measurement are Length 136cm Width 32cm

                                               labels       labels1  \
0  {classic, day to night, blacks, whites, work, …       classic
1  {classic, day to night, blacks, androgynous, w…       classic
2  {classic, day to night, casual, neutrals, week…       classic
3  {classic, day to night, androgynous, blacks, w…       classic
4  {day to night, oversized, browns, casual, andr…  day to night

        labels2      labels3  … labels12 labels13 labels14 labels15  \
0  day to night       blacks  …  Unknown  Unknown  Unknown  Unknown
1  day to night       blacks  …  Unknown  Unknown  Unknown  Unknown
2  day to night       casual  …  Unknown  Unknown  Unknown  Unknown
3  day to night  androgynous  …  Unknown  Unknown  Unknown  Unknown
4     oversized       browns  …  Unknown  Unknown  Unknown  Unknown

  labels16 labels17 labels18 labels19 labels20 labels21
0  Unknown  Unknown  Unknown  Unknown  Unknown  Unknown
```

```
1  Unknown  Unknown  Unknown  Unknown  Unknown  Unknown
2  Unknown  Unknown  Unknown  Unknown  Unknown  Unknown
3  Unknown  Unknown  Unknown  Unknown  Unknown  Unknown
4  Unknown  Unknown  Unknown  Unknown  Unknown  Unknown

[5 rows x 28 columns]
```

```
[21]: # Finding relevant labels in each dataframe
fit = ['semi-fitted','relaxed','straight / regular','fitted /␣
 ↪tailored','oversized']
occasion = ['day to night','work','weekend','night␣
 ↪out','vacation','coldweather','workout']
color␣
 ↪=['blacks','pinks','whites','reds','greens','blues','silvers','neutrals','beiges','grays','
        'browns','multi','oranges','teal']
style = ['business␣
 ↪casual','classic','modern','boho','glam','romantic','casual','androgynous','edgy','retro','


# Creating a new column that tells that each document had the relevant label␣
 ↪(So it will assign yes and no)
for var in fit:
    df_fit['{}'.format(var)] = functools.reduce(np.logical_or,␣
 ↪[df_fit['labels{}'.format(i)].str.contains(var) for i in range(1,22)])

for var in occasion:
    df_occasion['{}'.format(var)] = functools.reduce(np.logical_or,␣
 ↪[df_occasion['labels{}'.format(i)].str.contains(var) for i in range(1,22)])

for var in color:
    df_color['{}'.format(var)] = functools.reduce(np.logical_or,␣
 ↪[df_color['labels{}'.format(i)].str.contains(var) for i in range(1,22)])

for var in style:
    df_style['{}'.format(var)] = functools.reduce(np.logical_or,␣
 ↪[df_style['labels{}'.format(i)].str.contains(var) for i in range(1,22)])

#drop original labels columns
df_fit.drop(['labels{}'.format(i) for i in range(1,22)] + ['labels'],axis =␣
 ↪1,inplace = True)
df_occasion.drop(['labels{}'.format(i) for i in range(1,22)] + ['labels'],axis␣
 ↪= 1,inplace = True)
df_color.drop(['labels{}'.format(i) for i in range(1,22)] + ['labels'],axis =␣
 ↪1,inplace = True)
df_style.drop(['labels{}'.format(i) for i in range(1,22)] + ['labels'],axis =␣
 ↪1,inplace = True)
```

[22]:
```
# Exporting files
df_fit.to_csv("style.csv")
df_occasion.to_csv("occasion.csv")
df_color.to_csv("color.csv")
df_style.to_csv("fit.csv")
```

Model Building

April 4, 2021

[10]:
```
# Importing relevant libraries
import pandas as pd
import numpy as np
import re
from collections import Counter
import nltk
import keras
import spacy
import functools
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import GradientBoostingClassifier
from gensim.test.utils import common_texts, get_tmpfile
from sklearn.feature_extraction.text import CountVectorizer
from keras.layers.recurrent import SimpleRNN, LSTM
from keras.layers import Flatten, Masking
from sklearn.linear_model import LogisticRegression
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from gensim.models import Word2Vec
from nltk import word_tokenize
from keras.preprocessing.text import Tokenizer
from random import randint
from numpy import array, argmax, asarray, zeros
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Embedding
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import warnings
from sklearn import linear_model
warnings.filterwarnings("ignore")
```

[11]:
```
# Reading 4 Data Frames from part 1
df_color = pd.read_csv("color.csv",index_col = 0)
df_fit = pd.read_csv("fit.csv",index_col = 0)
df_occasion = pd.read_csv("occasion.csv",index_col = 0)
df_style = pd.read_csv("style.csv", index_col = 0)
```

[12]:
```
# transforming T/F to binary (we start getting labels from column 6 onwards␣
↪that is why we just choose those to convert them to 1 and 0)
df_color.iloc[:,6:]= df_color.iloc[:,6:].astype(int)
df_fit.iloc[:,6:]= df_fit.iloc[:,6:].astype(int)
df_occasion.iloc[:,6:]= df_occasion.iloc[:,6:].astype(int)
df_style.iloc[:,6:]= df_style.iloc[:,6:].astype(int)
```

[13]:
```
df_color.head()
```

[13]:
```
                   product_id             brand  \
0  01DSECZPAGJJC1EDC79JRBF4WK   Banana Republic
1  01DVA59VHYAPT4PVX32NXW91G5              Tibi
2  01DVA4XY7A0QMMSK3V3SBR52J9  Alexandre Birman
3  01DVBP9AHVQTZXJSBNJ0N2NYJP            Khaite
4  01DVBR93Y7KANZE3C09YCTVXDF  Lauren Manoogian

                       product_full_name  \
0                   Mock-Neck Sweater Top
1                     Juan Embossed Mules
2  Clarita Bow-Embellished Suede Sandals
3                      Leather ankle boots
4                       Alpaca-blend scarf

                                         description  \
0  Designed worn high-waisted bottom oh-so-now mo…
1  Tibis Juan embossed mule made shiny black leat…
2  Alexandre Birmans Clarita sandal quickly risen…
3  Heel measure approximately 50mm Number inch Bl…
4  Brown alpaca-blend Number alpaca Number polyam…

                 brand_category  \
0                       Unknown
1             women:SHOES:MULES
2           women:SHOES:SANDALS
3            Shoes   Boots   Ankle
4  Accessories   Scarves   Scarves

                                             details  blacks  pinks  whites  \
0  Designed worn high-waisted bottom oh-so-now mo…       1      0       1
1  seen Pre-Fall ‘Number runway Heel measure appr…       1      0       0
2  Heel height measure approximately 50mm Number …       0      0       0
```

```
3     Fits true size take normal size Italian sizing       1       0        0
4       item measurement are Length 136cm Width 32cm       0       0        0

   reds   …  grays  golds  navy  yellows  burgundies  purples  browns  multi  \
0     0   …      0      0     0        0           0        0       0      0
1     0   …      0      0     0        0           0        0       0      0
2     0   …      0      0     0        0           0        0       0      0
3     0   …      0      0     0        0           0        0       0      0
4     0   …      0      0     0        0           0        0       1      0

   oranges  teal
0        0     0
1        0     0
2        0     0
3        0     0
4        0     0

[5 rows x 25 columns]
```

[5]:
```
doc_color = df_color.brand + df_color.product_full_name + df_color.description␣
↪+ df_color.brand_category + df_color.details
doc_fit = df_fit.brand + df_fit .product_full_name + df_fit.description +␣
↪df_fit.brand_category + df_fit.details
doc_occasion = df_occasion.brand + df_occasion.product_full_name + df_occasion.
↪description + df_occasion.brand_category + df_occasion.details
doc_style = df_style.brand + df_style.product_full_name + df_style.description␣
↪+ df_style.brand_category + df_style.details
```

0.0.1 1. Count_Vectorizer with Logistic

[6]:
```
# 42 models trained

def logistic_model(doc,df,columns):

    vectorizer = CountVectorizer(feature_name)
    X = vectorizer.fit(doc)
    #X = vectorizer.transform(X_test) juse for test
    X = X.toarray()
    X = StandardScaler().fit_transform(X)# same for this (separtely)
    data = pd.DataFrame(X, columns=vectorizer.get_feature_names())

    models = []

    for col in columns:
        y = df[col].values
        #base_accuracy = y.sum()/len(y)
        #base_accuracy = max(base_accuracy,1-base_accuracy)

        data["TARGET"] = y

        train_df, test_df = train_test_split(data)
        X_train = train_df.loc[:, ~train_df.columns.isin(['TARGET'])]
        X_test = test_df.loc[:, ~test_df.columns.isin(['TARGET'])]

        y_train = train_df["TARGET"]
        y_test = test_df["TARGET"]

        clf =linear_model.LogisticRegression(C=0.001,random_state=None).
 ↪fit(X_train, y_train)

        #models.append[clf]
        #y_pred = clf.predict(X_test)

        #acc = np.mean(y_pred == y_test)
        models.append(clf)
    return X_test


columns_color =␣
 ↪['blacks','pinks','whites','reds','greens','blues','silvers','neutrals','oranges',
              ␣
 ↪'beiges','grays','golds','navy','yellows','burgundies','purples','browns','multi','teal']
columns_fit   = ['business␣
 ↪casual','classic','modern','boho','glam','romantic','casual','androgynous','edgy','retro','
columns_occasion = ['day to night','work','weekend','night␣
 ↪out','vacation','coldweather','workout']
columns_style = ['semi-fitted','relaxed','straight / regular','fitted /␣
 ↪tailored','oversized']

model_color = logistic_model(doc_color,df_color,columns_color)
#model_fit = logistic_model(doc_fit,df_fit,columns_fit)
#model_occasion = logistic_model(doc_occasion,df_occasion,columns_occasion)
#model_style = logistic_model(doc_style,df_style,columns_style)

#model_list = model_color+model_fit+model_occasion+model_style
```

```
       ␣
↪---------------------------------------------------------------------------


       NameError                                 Traceback (most recent call␣
↪last)
```

```
        <ipython-input-6-186831e47d5b> in <module>
         42 columns_style = ['semi-fitted','relaxed','straight /␣
↪regular','fitted / tailored','oversized']
         43
    ---> 44 model_color = logistic_model(doc_color,df_color,columns_color)
         45 #model_fit = logistic_model(doc_fit,df_fit,columns_fit)
         46 #model_occasion =␣
↪logistic_model(doc_occasion,df_occasion,columns_occasion)


        <ipython-input-6-186831e47d5b> in logistic_model(doc, df, columns)
          3 def logistic_model(doc,df,columns):
          4
    ----> 5     vectorizer = CountVectorizer(feature_name)
          6     X = vectorizer.fit(doc)
          7     #X = vectorizer.transform(X_test) juse for test


        NameError: name 'feature_name' is not defined
```

[7]:
```
model_color
```

```
      ␣
↪---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call␣
↪last)

        <ipython-input-7-0f330dda1959> in <module>
    ----> 1 model_color


        NameError: name 'model_color' is not defined
```

[8]:
```
ylabel =␣
 ↪['blacks','pinks','whites','reds','greens','blues','silvers','neutrals','oranges',
                 ␣
 ↪'beiges','grays','golds','navy','yellows','burgundies','purples','browns','multi','teal'
'business␣
 ↪casual','classic','modern','boho','glam','romantic','casual','androgynous','edgy','retro','
'day to night','work','weekend','night out','vacation','coldweather','workout'
'semi-fitted','relaxed','straight / regular','fitted / tailored','oversized']

dftest = pd.DataFrame(ylabel,columns = ["color"])
```

[9]:
```
test_brand = "Forever 21"
test_product_full_name = "Jeans size 34 M,"
test_description = "This is a slim jeans"
test_brand_category = "Denim Jeans"
test_details = "Blue color"

test_docs = test_brand +" " + test_product_full_name + " " + test_description +␣
↪" " + test_brand_category +  " " + test_details

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Remove Punctuations
punctuation = "!@#$%^&*()_+<>?:.,;"

for c in test_docs:
    if c in punctuation:
        test_docs = test_docs.replace(c, "")



# Remove Stopwords
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(test_docs)
test_docs = [w for w in word_tokens if not w in stop_words]
test_docs = []
for w in word_tokens:
    if w not in stop_words:
        test_docs.append(w)

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(test_docs)
X = X.toarray()
X = StandardScaler().fit_transform(X)
test_data = pd.DataFrame(X, columns=vectorizer.get_feature_names())

test_data
#y_pred = []
#model_list[0]
#model_list[0].predict(test_data)

#test
```

[9]:
```
         21        34      blue     color     denim   forever     jeans  \
0 -0.288675 -0.288675 -0.288675 -0.288675 -0.288675  3.464102 -0.547723
1  3.464102 -0.288675 -0.288675 -0.288675 -0.288675 -0.288675 -0.547723
2 -0.288675 -0.288675 -0.288675 -0.288675 -0.288675 -0.288675  1.825742
```

```
3  -0.288675 -0.288675 -0.288675 -0.288675 -0.288675 -0.288675 -0.547723
4  -0.288675  3.464102 -0.288675 -0.288675 -0.288675 -0.288675 -0.547723
5  -0.288675 -0.288675 -0.288675 -0.288675 -0.288675 -0.288675 -0.547723
6  -0.288675 -0.288675 -0.288675 -0.288675 -0.288675 -0.288675 -0.547723
7  -0.288675 -0.288675 -0.288675 -0.288675 -0.288675 -0.288675 -0.547723
8  -0.288675 -0.288675 -0.288675 -0.288675 -0.288675 -0.288675  1.825742
9  -0.288675 -0.288675 -0.288675 -0.288675  3.464102 -0.288675 -0.547723
10 -0.288675 -0.288675 -0.288675 -0.288675 -0.288675 -0.288675  1.825742
11 -0.288675 -0.288675  3.464102 -0.288675 -0.288675 -0.288675 -0.547723
12 -0.288675 -0.288675 -0.288675  3.464102 -0.288675 -0.288675 -0.547723

        size      slim      this
0  -0.288675 -0.288675 -0.288675
1  -0.288675 -0.288675 -0.288675
2  -0.288675 -0.288675 -0.288675
3   3.464102 -0.288675 -0.288675
4  -0.288675 -0.288675 -0.288675
5  -0.288675 -0.288675 -0.288675
6  -0.288675 -0.288675  3.464102
7  -0.288675  3.464102 -0.288675
8  -0.288675 -0.288675 -0.288675
9  -0.288675 -0.288675 -0.288675
10 -0.288675 -0.288675 -0.288675
11 -0.288675 -0.288675 -0.288675
12 -0.288675 -0.288675 -0.288675
```

```
[ ]: nlp = spacy.load("en_core_web_md")
```

```
[ ]: def MAX_SEQUENCE_LENGTH(list1):
         max = 0
         for i in list1:
             if max<len(i):
                 max=len(i)
         return max

     def integer_encode_documents(docs, tokenizer):
         return tokenizer.texts_to_sequences(docs)

     def load_glove_vectors():
         embeddings_index = {}
         with open('glove.6B.100d.txt', encoding = 'utf-8') as f:
             for line in f:
                 values = line.split()
                 word = values[0]
                 coefs = asarray(values[1:], dtype='float32')
                 embeddings_index[word] = coefs
         return embeddings_index
```

```
def make_binary_classification_rnn_model(plot=False):
    model = Sequential()
    model.add(Embedding(VOCAB_SIZE, 100, weights=[embedding_matrix],␣
↪input_length=MAX_SEQUENCE_LENGTH, trainable=False))
    model.add(Masking(mask_value=0.0)) # masking layer, masks any words that␣
↪don't have an embedding as 0s.
    model.add(SimpleRNN(units=64, input_shape=(1, MAX_SEQUENCE_LENGTH)))
    model.add(Dense(16))
    model.add(Dense(2, activation='softmax'))

    # Compile the model
    model.compile(
    optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    # summarize the model
    model.summary()
    return model

def make_lstm_classification_model(plot = False):
    model = Sequential()
    model.add(Embedding(VOCAB_SIZE, 100, weights=[embedding_matrix],␣
↪input_length=MAX_SEQUENCE_LENGTH, trainable=False))
    model.add(Masking(mask_value=0.0)) # masking layer, masks any words that␣
↪don't have an embedding as 0s.
    model.add(LSTM(units=32, input_shape=(1, MAX_SEQUENCE_LENGTH)))
    model.add(Dense(2, activation='softmax'))

    # Compile the model
    model.compile(
    optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    # summarize the model
    model.summary()

    return model
```

[ ]:
```
doc_color
```

0.0.2 2. Glove + LSTM Model

[ ]:
```
# Tokenize Text
tokenizer = Tokenizer(num_words=5000, oov_token="UNKNOWN_TOKEN")
tokenizer.fit_on_texts(list(doc_color))

# integer encode the documents
encoded_docs = integer_encode_documents(doc_color, tokenizer)
```

```
# padding to create equal length sequences
MAX_SEQUENCE_LENGTH = 1000
padded_docs = tf.keras.preprocessing.sequence.pad_sequences(encoded_docs,␣
 ↪maxlen=MAX_SEQUENCE_LENGTH, padding='post')

encoder = LabelEncoder()
labels = to_categorical(encoder.fit_transform(df_color['blacks']))

# train-test split
X_train, X_test, y_train, y_test = train_test_split(padded_docs, labels,␣
 ↪test_size=0.2)

VOCAB_SIZE = int(len(tokenizer.word_index) * 1.1)

# Load in GloVe Vectors
embeddings_index = load_glove_vectors()
embeddings_index

# # create a weight matrix for words in training docs
embedding_matrix = zeros((VOCAB_SIZE, 100))
for word, i in tokenizer.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None: # check that it is an actual word that we␣
 ↪have embeddings for
        embedding_matrix[i] = embedding_vector

# define model
model = make_lstm_classification_model()

# fit the model
history = model.fit(X_train, y_train,validation_split = 0.1, epochs=5,␣
 ↪verbose=1)

# evaluate the model
loss, accuracy = model.evaluate(X_test, y_test, verbose=1)
print('Accuracy: %f' % (accuracy*100))
```

```
[ ]: test_docs = [
         "Employees look like they hate their job. Milkshake was like drinking milk.␣
      ↪Food was cold and not warm at all",
         "This Mcdonalds is not only in the business of making crappy food and␣
      ↪providing even crappier service watch out for the racket they have in the␣
      ↪parking lot . If your not careful reading the sign at the the front of the␣
      ↪entrance it is going to cost you $195.00 in parking fees. went in to to ask␣
      ↪the management they just blew me off. lucky they are in vegas where they␣
      ↪dont count on repeat businesssss.",
         "There are better stores without fruit flies in Griffin, GA.",
         "Slowest drive-thru ever. Better option is to go to the location on␣
      ↪arlington"
     ]

     test_docs = list(
         map(lambda doc: " ".join([token.text for token in nlp(doc) if not token.
      ↪is_stop]), test_docs))

     encoded_test_sample = integer_encode_documents(test_docs, tokenizer)

     padded_test_docs = keras.preprocessing.sequence.
      ↪pad_sequences(encoded_test_sample, maxlen=MAX_SEQUENCE_LENGTH,␣
      ↪padding='post')

     model.predict_classes(padded_test_docs)
     prediction = model.predict_classes(padded_test_docs)
     encoder.inverse_transform(prediction)
```

0.0.3 3. Word2Vec (Equal Weights)

[ ]:
```
# Tokenize Text
tokenizer = Tokenizer(num_words=5000, oov_token="UNKNOWN_TOKEN")
tokenizer.fit_on_texts(list(doc_color))

# integer encode the documents
encoded_docs = integer_encode_documents(doc_color, tokenizer)

# padding to create equal length sequences
MAX_SEQUENCE_LENGTH = 1000
padded_docs = tf.keras.preprocessing.sequence.pad_sequences(encoded_docs,␣
 ↪maxlen=MAX_SEQUENCE_LENGTH, padding='post')

encoder = LabelEncoder()
labels = to_categorical(encoder.fit_transform(df_color['blacks']))

# train-test split
X_train, X_test, y_train, y_test = train_test_split(padded_docs, labels,␣
 ↪test_size=0.2)

VOCAB_SIZE = int(len(tokenizer.word_index) * 1.1)

# Load in GloVe Vectors
embedding_matrix = []
for i in doc_color:
    embedding_matrix.append(nlp(i).vector)
```

```
embedding_matrix = np.asarray(embedding_matrix)
# embeddings_index = load_glove_vectors()
# embeddings_index

#create a weight matrix for words in training docs
# embedding_matrix = zeros((VOCAB_SIZE, 100))
# for word, i in tokenizer.word_index.items():
#     embedding_vector = embeddings_index.get(word)
#     if embedding_vector is not None: # check that it is an actual word that␣
↪we have embeddings for
#         embedding_matrix[i] = embedding_vector


# define model
model = make_lstm_classification_model2()

# fit the model
history = model.fit(X_train, y_train,validation_split = 0.1, epochs=5,␣
↪verbose=1)

# evaluate the model
loss, accuracy = model.evaluate(X_test, y_test, verbose=1)
print('Accuracy: %f' % (accuracy*100))
```

[ ]:
```
nlp(doc).vector
```

[ ]:
```
# test = "black shoes green belt"
# nlp(test).vector
```

0.0.4 3. GLOVE (Unequal Weights)

[ ]:
```
# vectorizer = TfidfVectorizer()
# X = vectorizer.fit_transform(doc)
# X = X.toarray()
# X.shape
```

0.0.5 5. Self Trained Corpus

[ ]:
```
# doc = list(doc.values)
# doc = [word_tokenize(review) for review in doc]
# model = Word2Vec(doc, min_count=5)
# words = list(model.wv.vocab)
# vectors = []
# for word in words:
#     vectors.append(model[word].tolist())
# data = np.array(vectors)
# data
```

0.1 Model 1 count vectorizer

0.1.1 5. Using count vectorization to find out more words that lemmatization could not remove and assigning them to base form for the purpose of dimensionality reduction

[ ]:
```
# converting remaning unchanged words to their base form manually
# doc1 = re.sub(r'wearability|wearable|wearin|wearing','wear',doc1)
# doc1 = re.sub(r'transitioning|transitioned|transitional','transition',doc1)
```

[ ]:
```
list(df_style.columns)
```

[ ]:
```
# # Count vectorization for full data

# # Subset of the broader category




# doc = df_occasion.brand + df_occasion.product_full_name + df_occasion.
 ↪description + df_occasion.brand_category + df_occasion.details

# vectorizer = CountVectorizer()
# X = vectorizer.fit_transform(doc)
# X = X.toarray()
# columns = ['day to night','work','weekend','night␣
 ↪out','vacation','coldweather','workout']


# data = pd.DataFrame(X, columns=vectorizer.get_feature_names())

# accuracy = []
# for col in columns:
#     y = df_occasion[col].values
#     base_accuracy = y.sum()/len(y)
#     base_accuracy = max(base_accuracy,1-base_accuracy)

#     data["TARGET"] = y

#     train_df, test_df = train_test_split(data)
#     X_train = train_df.loc[:, ~train_df.columns.isin(['TARGET'])]
#     X_test = test_df.loc[:, ~test_df.columns.isin(['TARGET'])]

#     y_train = train_df["TARGET"]
#     y_test = test_df["TARGET"]
#     lr.fit(X_train, y_train)
#     y_pred = lr.predict(X_test)

#     acc = np.mean(y_pred == y_test)
#     accuracy.append([col,acc,base_accuracy])
# accuracy
# # X= X.toarray()
# # countVector = pd.DataFrame(X, columns=vectorizer.get_feature_names())
# # pd.set_option('display.max_columns', None)
# # countVector.head()
```

[ ]:
```
# df_color.head()
```

[ ]:
```
# doc = df_color.brand + df_color.product_full_name + df_color.description +␣
↪df_color.brand_category + df_color.details

# vectorizer = CountVectorizer()
# X = vectorizer.fit_transform(doc)
# X = X.toarray()
# columns = ['blacks',
#  'pinks',
#  'whites',
#  'reds',
#  'greens',
#  'blues',
#  'silvers',
#  'neutrals',
#  'beiges',
#  'grays',
#  'golds',
#  'navy',
#  'yellows',
#  'burgundies',
#  'purples',
#  'browns',
#  'multi',
#  'oranges',
#  'teal']


# data = pd.DataFrame(X, columns=vectorizer.get_feature_names())
# accuracy = []
# for col in columns:
#     y = df_color[col].values
#     base_accuracy = y.sum()/len(y)
#     base_accuracy = max(base_accuracy,1-base_accuracy)
#     data["TARGET"] = y

#     train_df, test_df = train_test_split(data)
#     X_train = train_df.loc[:, ~train_df.columns.isin(['TARGET'])]
#     X_test = test_df.loc[:, ~test_df.columns.isin(['TARGET'])]

#     y_train = train_df["TARGET"]
#     y_test = test_df["TARGET"]

#     lr.fit(X_train, y_train)
#     y_pred = lr.predict(X_test)

#     acc = np.mean(y_pred == y_test)
#     accuracy.append([col,acc,base_accuracy])
# accuracy
```

```
[ ]: # doc = df_fit.brand + df_fit.product_full_name + df_fit.description + df_fit.
     ↪brand_category + df_fit.details

     # vectorizer = CountVectorizer()
     # X = vectorizer.fit_transform(doc)
     # X = X.toarray()
     # columns = ['business casual',
     #  'classic',
     #  'modern',
     #  'boho',
     #  'glam',
     #  'romantic',
     #  'casual',
     #  'androgynous',
     #  'edgy',
     #  'retro',
     #  'athleisure']


     # data = pd.DataFrame(X, columns=vectorizer.get_feature_names())
     # accuracy = []
     # for col in columns:
     #     y = df_fit[col].values
     #     base_accuracy = y.sum()/len(y)
     #     base_accuracy = max(base_accuracy,1-base_accuracy)

     #     data["TARGET"] = y

     #     train_df, test_df = train_test_split(data)
     #     X_train = train_df.loc[:, ~train_df.columns.isin(['TARGET'])]
     #     X_test = test_df.loc[:, ~test_df.columns.isin(['TARGET'])]

     #     y_train = train_df["TARGET"]
     #     y_test = test_df["TARGET"]

     #     lr.fit(X_train, y_train)
     #     y_pred = lr.predict(X_test)

     #     acc = np.mean(y_pred == y_test)
     #     accuracy.append([col,acc,base_accuracy])
     # accuracy
```

[ ]:
```
# doc = df_style.brand + df_style.product_full_name + df_style.description +␣
 ↪df_style.brand_category + df_style.details

# vectorizer = CountVectorizer()
# X = vectorizer.fit_transform(doc)
# X = X.toarray()
# columns = [
#  'semi-fitted',
#  'relaxed',
#  'straight / regular',
#  'fitted / tailored',
#  'oversized']


# data = pd.DataFrame(X, columns=vectorizer.get_feature_names())
# accuracy = []
# for col in columns:
#     y = df_style[col].values
#     base_accuracy = y.sum()/len(y)
#     base_accuracy = max(base_accuracy,1-base_accuracy)
#     data["TARGET"] = y

#     train_df, test_df = train_test_split(data)
#     X_train = train_df.loc[:, ~train_df.columns.isin(['TARGET'])]
#     X_test = test_df.loc[:, ~test_df.columns.isin(['TARGET'])]

#     y_train = train_df["TARGET"]
#     y_test = test_df["TARGET"]

#     clf =linear_model.LogisticRegression(C=0.001,random_state=None)
#     clf.fit(X_train, y_train)
#     y_pred = lr.predict(X_test)

#     acc = np.mean(y_pred == y_test)
#     accuracy.append([col,acc,base_accuracy])
# accuracy
```

```
[ ]: # doc = df_occasion.brand + df_occasion.product_full_name + df_occasion.
     ↪description + df_occasion.brand_category + df_occasion.details

     # vectorizer = CountVectorizer()
     # X = vectorizer.fit_transform(doc)
     # X = X.toarray()
     # columns = ['day to night','work','weekend','night␣
     ↪out','vacation','coldweather','workout']


     # data = pd.DataFrame(X, columns=vectorizer.get_feature_names())

     # n=[100,200,300]
     # max_depth=[2,4,6,8]
     # for i in n:
     #     for j in max_depth:
     #         accuracy = []
     #         for col in columns:
     #             y = df_occasion[col].values
     #             base_accuracy = y.sum()/len(y)
     #             base_accuracy = max(base_accuracy,1-base_accuracy)

     #             data["TARGET"] = y

     #             train_df, test_df = train_test_split(data)
     #             X_train = train_df.loc[:, ~train_df.columns.isin(['TARGET'])]
     #             X_test = test_df.loc[:, ~test_df.columns.isin(['TARGET'])]

     #             y_train = train_df["TARGET"]
     #             y_test = test_df["TARGET"]

     #             rf = RandomForestClassifier(max_depth=j, n_estimators = i, ␣
     ↪n_jobs = -1,max_features = 10).fit(X_train, y_train)
     #             y_pred = rf.predict(X_test)

     #             acc = np.mean(y_pred == y_test)
     #             accuracy.append([col,acc,base_accuracy])
     #         print(f"max_depth: {j}, estimators: {i}\n{accuracy}\n")
```

```
[ ]: # # Boosted Trees
     # doc = df_occasion.brand + df_occasion.product_full_name + df_occasion.
     ↪description + df_occasion.brand_category + df_occasion.details

     # vectorizer = CountVectorizer()
     # X = vectorizer.fit_transform(doc)
     # X = X.toarray()
     # columns = ['day to night','work','weekend','night␣
      ↪out','vacation','coldweather','workout']


     # data = pd.DataFrame(X, columns=vectorizer.get_feature_names())

     # n=[100,200,300]
     # max_depth=[3,4]
     # for i in n:
     #     for j in max_depth:
     #             accuracy = []
     #             for col in columns:
     #                 y = df_occasion[col].values
     #                 base_accuracy = y.sum()/len(y)
     #                 base_accuracy = max(base_accuracy,1-base_accuracy)

     #                 data["TARGET"] = y

     #                 train_df, test_df = train_test_split(data)
     #                 X_train = train_df.loc[:, ~train_df.columns.isin(['TARGET'])]
     #                 X_test = test_df.loc[:, ~test_df.columns.isin(['TARGET'])]

     #                 y_train = train_df["TARGET"]
     #                 y_test = test_df["TARGET"]

     #                 bt=GradientBoostingClassifier(n_estimators=i, learning_rate=0.
      ↪1,max_depth=j).fit(X_train, y_train)
     #                 y_pred = bt.predict(X_test)

     #                 acc = np.mean(y_pred == y_test)
     #                 accuracy.append([col,acc,base_accuracy])
     #             print(f"max_depth: {j}, estimators: {i}\n{accuracy}\n")
```

0.2 Model 2 Word2Vec

[ ]:
```
# df.head()
```

[ ]:
```
# ## TF-IDF Weighted Average Word Embeddings

# vectorizer = TfidfVectorizer()
# X = vectorizer.fit_transform(doc1)
# X = X.toarray()
# tf_idf = pd.DataFrame(X, columns=vectorizer.get_feature_names())

# # sum the tf idf scores for each document
# tf_idf["TF_IDF_SUM"] = tf_idf.sum(axis=1)
# tf_idf_scores = list(map( lambda x: x.lower(), tf_idf.columns))
# tf_idf_scores
```

1 TESTING SECTION FOR EVALUATION (FOR PROFESSOR)

[ ]:
```
test_brand = "Forever 21"
test_product_full_name = "Jeans size 34 M"
test_description = "This is a slim jeans"
test_brand_category = "Denim Jeans"
test_details = "Blue color"




# test_brand = str(input("Enter Brand: "))
# test_product_full_name = str(input("Product_Full_Name: "))
# test_description = str(input("Product Description: "))
# test_brand_category = str(input("Brand Category: "))
# test_details = str(input("Details: "))

# MAX_SEQUENCE_LENGTH = 4

# test_docs = test_brand +" " + test_product_full_name + " " + test_description␣
 ↪+ " " + test_brand_category +  " " + test_details
# test_docs = list(map(lambda doc: " ".join([token.text for token in nlp(doc)␣
 ↪if not token.is_stop]), test_docs))

# encoded_test_sample = integer_encode_documents(test_docs, tokenizer)
# padded_test_docs = keras.preprocessing.sequence.
 ↪pad_sequences(encoded_test_sample, maxlen=MAX_SEQUENCE_LENGTH,␣
 ↪padding='post')


# VOCAB_SIZE = int(len(tokenizer.word_index) * 1.1)




# from keras.layers.recurrent import SimpleRNN
# from keras.layers import Flatten, Masking


# def load_glove_vectors():
#     embeddings_index = {}

#     with open('glove.6B.100d.txt',encoding = "utf8") as f:
```

```
#         for line in f:
#             values = line.split()
#             word = values[0]
#             coefs = asarray(values[1:], dtype='float32')
#             embeddings_index[word] = coefs
#     print('Loaded %s word vectors.' % len(embeddings_index))
#     return embeddings_index

# labels = 1

# from keras.utils import to_categorical
# from sklearn.preprocessing import LabelEncoder
# encoder = LabelEncoder()
# #labels = to_categorical(encoder.fit_transform(labels))

# embeddings_index = load_glove_vectors()

# embedding_matrix = zeros((VOCAB_SIZE, 100))
# for word, i in tokenizer.word_index.items():
#     embedding_vector = embeddings_index.get(word)
#     if embedding_vector is not None: # check that it is an actual word that␣
 ↪we have embeddings for
#         embedding_matrix[i] = embedding_vector
# embedding_matrix

# model = Sequential()
# model.add(Embedding(VOCAB_SIZE, 100, weights=[embedding_matrix],␣
 ↪input_length=MAX_SEQUENCE_LENGTH, trainable=False))
# model.add(Masking(mask_value=0.0)) # masking layer, masks any words that␣
 ↪don't have an embedding as 0s.
# model.add(SimpleRNN(units=64, input_shape=(1, MAX_SEQUENCE_LENGTH)))
# model.add(Dense(32))
# model.add(Dense(9, activation='softmax'))



# prediction = model.predict_classes(padded_test_docs)
# encoder.inverse_transform(prediction)
```

[ ]:
```
# occasion_vectors = []
# for idx, occasion in enumerate(occasions): # iterate through each document
#     tokens = nlp(occasion) # have spacy tokenize the review text

#     # initially start a running total of tf-idf scores for a document
#     total_tf_idf_score_per_document = 0
#     # start a running total of initially all zeroes (300 is picked since that␣
↪is the word embedding size used by word2vec)
#     running_total_word_embedding = np.zeros(300)
#     for token in tokens: # iterate through each token

#     # if the token has a pretrained word embedding it also has a tf-idf score
#         if token.has_vector and token.text.lower() in available_tf_idf_scores:

#             tf_idf_score = tf_idf_lookup_table.loc[idx, token.text.lower()]
#             #print(f"{token} has tf-idf score of {tf_idf_lookup_table.
↪loc[idx, token.text.lower()]}")
#             running_total_word_embedding += tf_idf_score * token.vector

#             total_tf_idf_score_per_document += tf_idf_score

#     # divide the total embedding by the total tf-idf score for each document
#     document_embedding = running_total_word_embedding /␣
↪total_tf_idf_score_per_document
#     occasion_vectors.append(document_embedding)
# occasion_vectors
```

ProjectFilterOutfits

April 4, 2021

1 TEAM PURPLE

1.1 APPENDIX to 'NLP Part2 Team Purple Code.ipynb'

Notebook Description

- In this notebook we will find the most relevant words (eg: common nouns) associated with each outfit item type. When a test query/document is submitted on user interface, this query is parsed to check with what outfit item type(s) it matches using regular expression.
- Once we know the possible outfit item types through this notebook, we find the most similar product by filtering dataset on these outfit item types only.
- This rationale has reduced false positives to a mimimum since without this logic finding exact/similar products using description was leading to irrevalant matches at times.

[1]:
```
##Importing required libraries

import pandas as pd
import numpy as np
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from functools import reduce
import re
```

[3]:
```
# Reading input file

data = pd.read_csv("outfit_combinations.csv")
data.head()
```

[3]:
```
                  outfit_id                  product_id outfit_item_type  \
0  01DDBHC62ES5K80P0KYJ56AM2T  01DMBRYVA2P5H24WK0HTK4R0A1           bottom
1  01DDBHC62ES5K80P0KYJ56AM2T  01DMBRYVA2PEPWFTT7RMP5AA1T              top
2  01DDBHC62ES5K80P0KYJ56AM2T  01DMBRYVA2S5T9W793F4CY41HE       accessory1
3  01DDBHC62ES5K80P0KYJ56AM2T  01DMBRYVA2ZFDYRYY5TRQZJTBD             shoe
4  01DMHCX50CFX5YNG99F3Y65GQW  01DMBRYVA2P5H24WK0HTK4R0A1           bottom

            brand           product_full_name
0      Eileen Fisher              Slim Knit Skirt
1      Eileen Fisher            Rib Mock Neck Tank
```

```
2  kate spade new york  medium margaux leather satchel
3            Tory Burch       Penelope Mid Cap Toe Pump
4         Eileen Fisher                 Slim Knit Skirt
```

1.1.1 TFIDF Vectorized Scores

- This function will take product full name for a particular outfit item type at a time
- It will then find the TF-IDF vectorized score of words in descending order found in the product full name corresponding to outfit item type.

[4]:
```
# Function to generate TF-IDF Vector

vectorizer = TfidfVectorizer(token_pattern=r'\b[a-zA-Z0-9]{3,}\b',
                             min_df=0.001,
                            stop_words=stopwords.words('english'))

def vectorizeProductFullName(df,outfitType):
    X = vectorizer.fit_transform(df)
    terms = vectorizer.get_feature_names()
    tf_idf = pd.DataFrame(X.toarray().transpose(), index=terms)
    tf_idf = tf_idf.sum(axis=1)
    outfit = pd.DataFrame(tf_idf, columns=[outfitType+'_score'])
    outfit["term"] = terms
    outfit["category"] = outfitType
    outfit.sort_values(by=outfitType+'_score', ascending=False, inplace=True)
    return outfit
```

Category: Shoe

[5]:
```
## filter dataset on shoe outfit and get the product_full_name
shoe_df = data[data['outfit_item_type']=='shoe']['product_full_name'].tolist()
```

[6]:
```
## Get the TF-IDF score for the words in shoe outfit
shoe = vectorizeProductFullName(shoe_df,'shoe')
shoe.reset_index(drop=True,inplace=True)
```

[7]:
```
## Here we will find most relevant words associated with shoes outfit
shoe.head(30)
```

[7]:
```
   shoe_score      term category
0  118.846684   leather     shoe
1   96.593574     boots     shoe
2   81.333578     ankle     shoe
3   71.811042     suede     shoe
4   61.301217   sandals     shoe
5   61.222516     pumps     shoe
6   44.062204     mules     shoe
```

```
7    43.332791       effect     shoe
8    42.119832        snake     shoe
9    40.608446     sneakers     shoe
10   33.949594    slingback     shoe
11   28.300546          toe     shoe
12   27.144380       sandal     shoe
13   26.359449     metallic     shoe
14   22.343143     embossed     shoe
15   21.870261       slides     shoe
16   20.483479          low     shoe
17   19.839709        point     shoe
18   19.791738        flats     shoe
19   19.728040        slide     shoe
20   18.534218         romy     shoe
21   18.414387         calf     shoe
22   17.681100        print     shoe
23   17.618928         star     shoe
24   17.241900  embellished     shoe
25   16.982267      paneled     shoe
26   16.527276         mule     shoe
27   16.487841         pump     shoe
28   15.635099         croc     shoe
29   15.513265          top     shoe
```

Category: Top

[8]:
```
## filter dataset on top outfit and get the product_full_name
top_df = data[data['outfit_item_type']=='top']['product_full_name'].tolist()
```

[11]:
```
## Get the TF-IDF score for the words in top outfit
top = vectorizeProductFullName(top_df,'top')
top.reset_index(drop=True,inplace=True)
```

[12]:
```
## Here we will find most relevant words associated with top outfit
top.head(30)
```

[12]:
```
    top_score       term category
0   66.810834      shirt      top
1   58.766229    sweater      top
2   56.776740       silk      top
3   52.190902     cotton      top
4   51.345017        top      top
5   43.847669       wool      top
6   42.782341      blend      top
7   41.965295     blouse      top
8   41.783438  turtleneck      top
9   36.613681      satin      top
```

```
10  31.027171     striped      top
11  29.511357      jersey      top
12  25.452379      draped      top
13  24.806841     printed      top
14  24.233264     cropped      top
15  22.971262         tee      top
16  22.681126      ribbed      top
17  22.433799        knit      top
18  21.919626    cashmere      top
19  21.765437        neck      top
20  20.436818        tank      top
21  20.294304       print      top
22  19.563236      sleeve      top
23  19.091479       crepe      top
24  18.309011     stretch      top
25  17.773349  sweatshirt      top
26  15.673111         tie      top
27  15.184315       voile      top
28  14.346460         boy      top
29  14.032869    bodysuit      top
```

Category: Bottom

```
[13]: ## filter dataset on bottom outfit and get the product_full_name
      bottom_df = data[data['outfit_item_type']=='bottom']['product_full_name'].
       ↪tolist()
```

```
[14]: ## Get the TF-IDF score for the words in bottom outfit
      bottom = vectorizeProductFullName(bottom_df,'bottom')
      bottom.reset_index(drop=True,inplace=True)
```

```
[15]: ## Here we will find most relevant words associated with bottom outfit
      bottom.head(30)
```

```
[15]:     bottom_score     term category
      0      80.074348      leg   bottom
      1      75.137667    pants   bottom
      2      71.064973    skirt   bottom
      3      64.417121     wide   bottom
      4      62.404956     rise   bottom
      5      61.726302    jeans   bottom
      6      61.051533     high   bottom
      7      50.211487     midi   bottom
      8      41.212341   cotton   bottom
      9      38.242021  cropped   bottom
      10     35.937517     slim   bottom
      11     34.538493     wool   bottom
```

```
12      32.114202  straight   bottom
13      31.806149    belted   bottom
14      31.467942   leather   bottom
15      29.416779     track   bottom
16      28.598333     crepe   bottom
17      27.687615     satin   bottom
18      26.620643     blend   bottom
19      23.435965  corduroy   bottom
20      22.863763      silk   bottom
21      22.834806       mid   bottom
22      22.766995    skinny   bottom
23      22.457793     twill   bottom
24      21.877219   striped   bottom
25      20.696511   stretch   bottom
26      20.655689      mini   bottom
27      18.092428    shorts   bottom
28      16.889618    jersey   bottom
29      16.050515  cashmere   bottom
```

Category: Onepiece

[16]:
```
## filter dataset on onepiece and get the product_full_name
onepiece_df = data[data['outfit_item_type']=='onepiece']['product_full_name'].
 ↪tolist()
```

[17]:
```
## Get the TF-IDF score for the words in onepiece
onepiece = vectorizeProductFullName(onepiece_df,'onepiece')
onepiece.reset_index(drop=True,inplace=True)
```

[18]:
```
## Here we will find most relevant words associated with accessory1
onepiece.head(30)
```

[18]:
```
    onepiece_score       term  category
0        28.838090      dress  onepiece
1        16.207471       mini  onepiece
2        13.048706     cotton  onepiece
3        12.389663      linen  onepiece
4        12.140031   jumpsuit  onepiece
5        11.556358       wrap  onepiece
6        11.330612      crepe  onepiece
7        10.406386       silk  onepiece
8         9.981422       midi  onepiece
9         9.513179     floral  onepiece
10        9.365523    stretch  onepiece
11        8.576385       maxi  onepiece
12        7.948648      print  onepiece
13        6.430161     sleeve  onepiece
```

```
14        5.994903       shirt  onepiece
15        5.667145       blend  onepiece
16        5.614349      belted  onepiece
17        5.263993      jersey  onepiece
18        4.832116      tiered  onepiece
19        4.824012      larina  onepiece
20        4.765196        long  onepiece
21        4.537669      draped  onepiece
22        4.269563       satin  onepiece
23        4.196457         dot  onepiece
24        4.196457       polka  onepiece
25        4.103499         tie  onepiece
26        4.098677    metallic  onepiece
27        4.093156       denim  onepiece
28        4.004016  embroidered  onepiece
29        3.893421     leopard  onepiece
```

Category: Accessory1

[19]:
```
## filter dataset on accessory1 and get the product_full_name
accessory1_df =␣
 ↪data[data['outfit_item_type']=='accessory1']['product_full_name'].tolist()
```

[20]:
```
## Get the TF-IDF score for the words in accessory1
accessory1 = vectorizeProductFullName(accessory1_df,'accessory1')
accessory1.reset_index(drop=True,inplace=True)
```

[21]:
```
## Here we will find most common and relevant words associated with accessory1
accessory1.head(30)
```

[21]:
```
    accessory1_score       term    category
0         131.773751    leather  accessory1
1         111.024354        bag  accessory1
2          77.260215   shoulder  accessory1
3          60.858545       tote  accessory1
4          46.544087      small  accessory1
5          38.989623       croc  accessory1
6          38.046061     clutch  accessory1
7          36.532774       mini  accessory1
8          35.516223   textured  accessory1
9          34.775623      large  accessory1
10         33.240551     effect  accessory1
11         29.848660       wool  accessory1
12         28.514196       tori  accessory1
13         27.238913      scarf  accessory1
14         27.187178   embossed  accessory1
15         26.932921    cabinet  accessory1
```

```
16          26.503119     bucket  accessory1
17          25.687628        top  accessory1
18          22.753099   backpack  accessory1
19          22.504129       silk  accessory1
20          21.375611    hammock  accessory1
21          18.619449     handle  accessory1
22          18.592815  oversized  accessory1
23          17.926676      blend  accessory1
24          17.833235     blazer  accessory1
25          17.332524    shopper  accessory1
26          17.236131        two  accessory1
27          16.826932    printed  accessory1
28          16.695555       lazo  accessory1
29          16.341685       belt  accessory1
```

Category: Accessory 2

[22]:
```
## filter dataset on accessory2 and get the product_full_name
accessory2_df =␣
↪data[data['outfit_item_type']=='accessory2']['product_full_name'].tolist()
```

[23]:
```
## Get the TF-IDF score for the words in accessory1
accessory2 = vectorizeProductFullName(accessory2_df,'accessory2')
accessory2.reset_index(drop=True,inplace=True)
```

[24]:
```
## Here we will find most relevant words associated with accessory2
accessory2.head(30)
```

[24]:
```
    accessory2_score       term    category
0          69.166653       wool  accessory2
1          65.922846     jacket  accessory2
2          63.089966       coat  accessory2
3          58.555551   cardigan  accessory2
4          49.313984       wrap  accessory2
5          48.041975      blend  accessory2
6          42.934633   cashmere  accessory2
7          40.659035    leather  accessory2
8          36.495422     cotton  accessory2
9          35.404966        bag  accessory2
10         33.015664     belted  accessory2
11         29.696126       knit  accessory2
12         29.575888     ribbed  accessory2
13         26.482867     double  accessory2
14         26.086540     blazer  accessory2
15         25.071639  oversized  accessory2
16         23.883585      twill  accessory2
17         21.575179   shoulder  accessory2
```

```
18         21.309340       faced  accessory2
19         20.980369      trench  accessory2
20         20.822656     sweater  accessory2
21         19.000000        name  accessory2
22         18.972982  reversible  accessory2
23         18.544662       shirt  accessory2
24         17.974091       denim  accessory2
25         17.625283      hoodie  accessory2
26         17.523345       woven  accessory2
27         16.861367        silk  accessory2
28         16.328328    breasted  accessory2
29         16.083695       scarf  accessory2
```

Category: Accessory3

[9]:
```python
## filter dataset on accessory2 and get the product_full_name
accessory3_df =␣
↪data[data['outfit_item_type']=='accessory3']['product_full_name'].tolist()
```

[10]:
```python
## Get the TF-IDF score for the words in accessory1
accessory3 = vectorizeProductFullName(accessory3_df,'accessory3')
accessory3.reset_index(drop=True,inplace=True)
```

[11]:
```python
## Here we will find most relevant words associated with accessory2
accessory3.head(30)
```

[11]:
```
   accessory3_score        term    category
0               0.5  asymmetric  accessory3
1               0.5        coat  accessory3
2               0.5      cotton  accessory3
3               0.5      trench  accessory3
```

1.1.2 Regular Expressions

- In below cell we have prepared regular expression for each of the outfit item types using the relevant words (preferrably proper nouns) found in above cells
- If a relevant word appears in more than one outfit type we have included it in regular expressions of all the outfit item types.
- Similarly, we have included unique words corresponding to each outfit item type (from above cells). So, if a user enters an product description unique to an outfit item type we narrow down our search to that specific outfit type.

[12]:
```python
#Regular expressions for each of the outfit item types

shoe=r'(boot|sandal|pump|mule|sneaker|loafer|slingback|flat|slide|croc)'
top=r'(shirt|sweater|top|blouse|turtleneck|jersey|tee|bodysuit|neck|sleeve|jacket|coat|cardiga
bottom=r'(leg|pant|skirt|jean|rise|midi|short|trouser)'
```

```
onepiece =␣
 ↪r'(dress|jumpsuit|wrap|stretch|maxi|midi|larina|francoise|polka|shirt|sweater|top|blouse|tu
accessory1=r'(bag|tote|croc|tori|clutch|mini|scarf|cabinet|top|bucket|backpack|hammock|belt|la
accessory2=r'(bag|tote|croc|tori|clutch|mini|scarf|cabinet|top|bucket|backpack|hammock|belt|la
accessory3= r'(coat)'
```

[13]:
```
# Test description
# 'bucket' belongs to Accessory1 and Accessory2 only
# Thus, our output should recognize both these categories

description = 'bucket'
```

[14]:
```
# outfitTypes is a dictionary to map 'outfit item type' with it's regular␣
 ↪expression created above
outfitTypes={'top':top,'bottom':bottom,'shoe':shoe,'onepiece':
 ↪onepiece,'accessory1':accessory1,'accessory2':accessory2,'accessory3':
 ↪accessory3}

# Parse test description and return its corresponding outfit item types in a␣
 ↪list called outfits
outfits = [outfit for outfit in outfitTypes if re.
 ↪search(outfitTypes[outfit],description,flags=re.IGNORECASE)]
```

[15]:
```
# Since bucket is common to accessory1 and accessory2. The outfits list below␣
 ↪is used in main notebook to narrow down
# the dataset
outfits
```

[15]:
```
['accessory1', 'accessory2']
```

APPENDIX - Rule-based prediction for Category

April 4, 2021

1 APPENDIX - Rule-based prediction for Category

[8]:
```
# Import relevant libraries
import pandas as pd
import numpy as np
import re
```

[2]:
```
# read input file into dataframe

data=pd.read_csv('Full Data + Tagged Product Combined.csv')
data.head()
```

[2]:
```
                     product_id            brand       mpn  \
0  01DPGV4YRP3Z8J85DASGZ1Y99W            Frame  LWAX0056
1  01DPGV4YRP3Z8J85DASGZ1Y99W            Frame  LWAX0056
2  01DSE8Z2ZDAZKZ2SKCS1E3B3HK  Banana Republic    491075
3  01DSE8Z2ZDAZKZ2SKCS1E3B3HK  Banana Republic    491075
4  01E2C3YN4KQ36A0REWZJ89ZN73   FREDA SALVADOR   5229129

              product_full_name  \
0    Les Second - Medium--NOIR
1    Les Second - Medium--NOIR
2  Madison 12-Hour Loafer Pump
3  Madison 12-Hour Loafer Pump
4                   Ace Bootie

                                         description brand_category  \
0  Minimal, Modern Styling Meets Refined Luxury I…    Accessories
1  Minimal, Modern Styling Meets Refined Luxury I…    Accessories
2  Everything you love about our original Madison…        Unknown
3  Everything you love about our original Madison…        Unknown
4  Edgy style and expert craftsmanship combine on…        Unknown

                         created_at                         updated_at  \
0     2019-10-06 15:31:31.730524+00     2019-12-19 20:40:30.786144+00
1  2019-10-06 15:31:31.730000+00:00  2020-04-06 23:19:53.216000+00:00
2  2019-11-11 22:22:21.664000+00:00  2020-03-25 23:24:44.823000+00:00
```

```
3    2019-11-11 22:22:21.664425+00    2019-12-19 20:40:30.786144+00
4  2020-03-01 22:37:32.169000+00:00  2020-04-15 21:46:03.512000+00:00

                        deleted_at  \
0                              NaN
1  2020-04-06 23:19:53.216000+00:00
2  2020-03-23 21:06:15.953000+00:00
3                              NaN
4  2020-03-18 23:00:31.558000+00:00

                                brand_canonical_url  \
0  https://frame-store.com/products/les-second-me…
1  https://frame-store.com/products/les-second-me…
2  https://bananarepublic.gap.com/browse/product…
3  https://bananarepublic.gap.com/browse/product…
4  https://shop.nordstrom.com/s/freda-salvador-ac…

                                            details labels  bc_product_id  \
0                                              NaN     {}            NaN
1                                              NaN     []          185.0
2  Everything you love about our original Madison…     []          431.0
3  Everything you love about our original Madison…     {}            NaN
4  True to size.\n2 1/4" (57mm) heel (size 8.5)\n…     []         1051.0

                product_id-2            product_color_id attribute_name  \
0  01DPGV4YRP3Z8J85DASGZ1Y99W  01DPGVGBK6YGNYGNF2S6FSH02T          style
1  01DPGV4YRP3Z8J85DASGZ1Y99W  01DPGVGBK6YGNYGNF2S6FSH02T          style
2  01DSE8Z2ZDAZKZ2SKCS1E3B3HK  01DSE8ZG8Y3FR8KWE2TY1QDWBF     shoe_width
3  01DSE8Z2ZDAZKZ2SKCS1E3B3HK  01DSE8ZG8Y3FR8KWE2TY1QDWBF     shoe_width
4  01E2C3YN4KQ36A0REWZJ89ZN73  01E2C3YN56ZCJ8TN45V3EC8CPS  Primary Color

  attribute_value          file
0          Casual  initial_tags
1          Casual  initial_tags
2          Medium  initial_tags
3          Medium  initial_tags
4          Blacks  initial_tags
```

[3]:
```
# check null values
data.isnull().sum()
```

[3]:
```
product_id                0
brand                     0
mpn                       0
product_full_name         0
description           18496
brand_category         5870
```

```
created_at                0
updated_at                0
deleted_at            94172
brand_canonical_url      34
details               34370
labels                    0
bc_product_id         75961
product_id-2          44105
product_color_id      44105
attribute_name        44105
attribute_value       44105
file                  44105
dtype: int64
```

[5]:
```
# drop unnecessary columns
data.drop(columns=['mpn', 'created_at', 'updated_at', 'deleted_at',
                'brand_canonical_url',␣
 ↪'bc_product_id','product_id-2','product_color_id','file'],inplace=True)
```

[6]:
```
data.head()
```

[6]:
```
                        product_id              brand  \
0       01DPGV4YRP3Z8J85DASGZ1Y99W              Frame
1       01DPGV4YRP3Z8J85DASGZ1Y99W              Frame
2       01DSE8Z2ZDAZKZ2SKCS1E3B3HK  Banana Republic
3       01DSE8Z2ZDAZKZ2SKCS1E3B3HK  Banana Republic
4       01E2C3YN4KQ36A0REWZJ89ZN73    FREDA SALVADOR
…                                …                …
203113  01DPETKRJG1XH8XZESV7JSF4VP           J.Crew
203114  01DSE9X9C73BXXGXNAPMKGJTD1  Banana Republic
203115  01DSP4DTZF3P5EF1RR53AZ2TP2          Shinola
203116  01DPH1M5PXCK323CFYDMMJ4P9C     Sole Society
203117  01E4EFN1J438ZN2V6ZMNAV79Q2            FRAME

                                 product_full_name  \
0                          Les Second - Medium--NOIR
1                          Les Second - Medium--NOIR
2                        Madison 12-Hour Loafer Pump
3                        Madison 12-Hour Loafer Pump
4                                         Ace Bootie
…                                                  …
203113  Riley sandals in sunwashed pink patent leather
203114                         Glitz Short Necklace
203115           Vinton Stainless Steel Bracelet Watch
203116                                        Nadina
203117                            Le Francoise Skinny
```

```
                                              description  \
0       Minimal, Modern Styling Meets Refined Luxury I…
1       Minimal, Modern Styling Meets Refined Luxury I…
2       Everything you love about our original Madison…
3       Everything you love about our original Madison…
4       Edgy style and expert craftsmanship combine on…
…                                                    …
203113  Our design team created this strappy sandal sp…
203114  Dress it up or dress it down, our jewelry coll…
203115  From the Vinton Collection. Featuring a matte …
203116  Details    \nSlide into style with this trend-…
203117  We took our heritage Francoise jean and reinve…

                                           brand_category  \
0                                             Accessories
1                                             Accessories
2                                                 Unknown
3                                                 Unknown
4                                                 Unknown
…                                                       …
203113                                              shoes
203114                                            Unknown
203115  JewelryAccessories/Watches/ForHim,TheMensStore…
203116                                              Shoes
203117                                              Jeans

                                                  details  \
0                                                     NaN
1                                                     NaN
2       Everything you love about our original Madison…
3       Everything you love about our original Madison…
4       True to size.\n2 1/4" (57mm) heel (size 8.5)\n…
…                                                     …
203113                                                NaN
203114  Dress it up or dress it down, our jewelry coll…
203115  Argonite 715 Swiss quartz movement\nPolished s…
203116                                                NaN
203117                                                NaN

                          labels attribute_name attribute_value
0                             {}          style          Casual
1                             []          style          Casual
2                             []     shoe_width          Medium
3                             {}     shoe_width          Medium
4                             []  Primary Color          Blacks
…                            …              …               …
203113       {"Needs Attributes"}           NaN             NaN
```

```
203114              {"Needs Review"}            NaN             NaN
203115              {"Needs Review"}            NaN             NaN
203116              {"Needs Review"}            NaN             NaN
203117   [{'value': 'Needs Review'}]            NaN             NaN

[203118 rows x 9 columns]
```

1.0.1 Regex to identify Category: Occasion

[30]:
```
pattern=r'Casual|Weekend|Day\s?(?:to)\s?Night|Night\s?out|work(?:out)'



data['occasion']=np.where(data.brand.str.contains(pattern,flags=re.
 ↪IGNORECASE),True,
                         np.where(data.product_full_name.str.
 ↪contains(pattern,flags=re.IGNORECASE),True,
                         np.where(data.description.str.
 ↪contains(pattern,flags=re.IGNORECASE),True,
                         np.where(data.brand_category.str.
 ↪contains(pattern,flags=re.IGNORECASE),True,
                         np.where(data.details.str.contains(pattern,flags=re.
 ↪IGNORECASE),True,False)))))
```

1.0.2 Regex to identify Category: Style

[17]:
```
pattern2=r'Androgynous|Athleisure|Boho|Business\s?(?:
 ↪Casual)|Classic|Edgy|Glam|Modern|Retro|Romantic'



data['style']=np.where(data.brand.str.contains(pattern2,flags=re.
 ↪IGNORECASE),True,
                         np.where(data.product_full_name.str.
 ↪contains(pattern2,flags=re.IGNORECASE),True,
                         np.where(data.description.str.
 ↪contains(pattern2,flags=re.IGNORECASE),True,
                         np.where(data.brand_category.str.
 ↪contains(pattern2,flags=re.IGNORECASE),True,
                         np.where(data.details.str.contains(pattern2,flags=re.
 ↪IGNORECASE),True,False)))))
```

1.0.3 Regex to identify Category: Fit

[19]:
```
pattern3=r'Fitted|Tailored|Semi\s?-?\s?Fitted|Straight\s?
 ↪|Regular|Relaxed|Oversized'
```

```
data['fit']=np.where(data.brand.str.contains(pattern3,flags=re.IGNORECASE),True,
                        np.where(data.product_full_name.str.
↪contains(pattern3,flags=re.IGNORECASE),True,
                        np.where(data.description.str.
↪contains(pattern3,flags=re.IGNORECASE),True,
                        np.where(data.brand_category.str.
↪contains(pattern3,flags=re.IGNORECASE),True,
                        np.where(data.details.str.contains(pattern3,flags=re.
↪IGNORECASE),True,False)))))
```

1.0.4 Regex to identify Category: Color

[22]:
```
pattern4=r'Beiges|Blacks|Blues|Browns|Burgundies|Golds|Grays|Greens|Multi|Navy|Neutrals|Orange
            |Reds|Silvers|Teal|Whites|Yellows'

data['color']=np.where(data.brand.str.contains(pattern4,flags=re.
↪IGNORECASE),True,
                        np.where(data.product_full_name.str.
↪contains(pattern4,flags=re.IGNORECASE),True,
                        np.where(data.description.str.
↪contains(pattern4,flags=re.IGNORECASE),True,
                        np.where(data.brand_category.str.
↪contains(pattern4,flags=re.IGNORECASE),True,
                        np.where(data.details.str.contains(pattern4,flags=re.
↪IGNORECASE),True,False)))))
```

[33]:
```
data['labels'] = data.apply(lambda x:␣
↪list([x['occasion'],x['style'],x['fit'],x['color']]),axis=1)
data
```

[33]:
```
                        product_id            brand  \
0       01DPGV4YRP3Z8J85DASGZ1Y99W            Frame
1       01DPGV4YRP3Z8J85DASGZ1Y99W            Frame
2       01DSE8Z2ZDAZKZ2SKCS1E3B3HK  Banana Republic
3       01DSE8Z2ZDAZKZ2SKCS1E3B3HK  Banana Republic
4       01E2C3YN4KQ36AOREWZJ89ZN73   FREDA SALVADOR
…                              …               …
203113  01DPETKRJG1XH8XZESV7JSF4VP           J.Crew
203114  01DSE9X9C73BXXGXNAPMKGJTD1  Banana Republic
203115  01DSP4DTZF3P5EF1RR53AZ2TP2          Shinola
203116  01DPH1M5PXCK323CFYDMMJ4P9C     Sole Society
203117  01E4EFN1J438ZN2V6ZMNAV79Q2            FRAME

                           product_full_name  \
0                       Les Second - Medium--NOIR
1                       Les Second - Medium--NOIR
```

```
2                            Madison 12-Hour Loafer Pump
3                            Madison 12-Hour Loafer Pump
4                                             Ace Bootie
…                                                  …
203113  Riley sandals in sunwashed pink patent leather
203114                             Glitz Short Necklace
203115           Vinton Stainless Steel Bracelet Watch
203116                                           Nadina
203117                             Le Francoise Skinny

                                           description  \
0       Minimal, Modern Styling Meets Refined Luxury I…
1       Minimal, Modern Styling Meets Refined Luxury I…
2       Everything you love about our original Madison…
3       Everything you love about our original Madison…
4       Edgy style and expert craftsmanship combine on…
…                                                  …
203113  Our design team created this strappy sandal sp…
203114  Dress it up or dress it down, our jewelry coll…
203115  From the Vinton Collection. Featuring a matte …
203116  Details    \nSlide into style with this trend-…
203117  We took our heritage Francoise jean and reinve…

                                        brand_category  \
0                                          Accessories
1                                          Accessories
2                                              Unknown
3                                              Unknown
4                                              Unknown
…                                                  …
203113                                           shoes
203114                                         Unknown
203115  JewelryAccessories/Watches/ForHim,TheMensStore…
203116                                           Shoes
203117                                           Jeans

                                               details  \
0                                                  NaN
1                                                  NaN
2       Everything you love about our original Madison…
3       Everything you love about our original Madison…
4       True to size.\n2 1/4" (57mm) heel (size 8.5)\n…
…                                                  …
203113                                             NaN
203114  Dress it up or dress it down, our jewelry coll…
203115  Argonite 715 Swiss quartz movement\nPolished s…
203116                                             NaN
```

```
203117                                                 NaN

                               labels attribute_name attribute_value  occasion  \
0           [True, True, True, True]          style          Casual      True
1           [True, True, True, True]          style          Casual      True
2        [False, True, False, False]     shoe_width          Medium     False
3        [False, True, False, False]     shoe_width          Medium     False
4        [False, True, False, False]  Primary Color          Blacks     False
…                                …            …               …         …
203113      [True, True, True, True]            NaN             NaN      True
203114  [False, False, False, False]            NaN             NaN     False
203115   [False, True, False, False]            NaN             NaN     False
203116      [True, True, True, True]            NaN             NaN      True
203117      [True, True, True, True]            NaN             NaN      True

        style    fit  color
0        True   True   True
1        True   True   True
2        True  False  False
3        True  False  False
4        True  False  False
…         …     …      …
203113   True   True   True
203114  False  False  False
203115   True  False  False
203116   True   True   True
203117   True   True   True

[203118 rows x 13 columns]
```