Problem Statement

Given crawled data from PDP, we need to create semantic relationship between words of a product or we can say that we need to come up with feature vector for each of these words to see if we can use this information to cluster similar words/products.

```
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re
import glob
import os
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns
import string
import spacy
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from tqdm import tqdm
from wordcloud import WordCloud, STOPWORDS
```

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/abhishek_khatri_macys_com/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
def multiple_file_joiner(direc):
    '''
    Function can be used to merge all the csv files in a directory
    '''
    os.chdir(direc)
    all_filenames = [i for i in glob.glob('*.csv')]
    combined_csv = pd.concat([pd.read_csv(file) for file in all_filenames]).reset_index(dr
    return combined_csv
```

Reading Data

```
# Reading raw data
raw_data_df = multiple_file_joiner('/appl/taxonomy/data/Abhishek/comp_AI_data/PDP')
```

```
raw_data_df[raw_data_df['product_name'].str.contains('caterina')]
```

	product_name	brand	price	currency	breadcrumbs	pdp_url	variants	images	descr

```
raw_data_df.head()
```

	product_name	brand	price	currency	breadcrumbs	
0	Leather Heeled Booties	ADRIENNE VITTADINI	39	USD	Women > Shoes > View All	https://www.marshalls.com/us/st
1	Buckle Wedge Suede Booties	SOLE SOCIETY	29	USD	Women > Shoes > View All	https://www.marshalls.com/us/st
2	Suede Whipstitch Booties	1.STATE	29	USD	Women > Shoes > View All	https://www.marshalls.com/us/st
3	Leather Snake Booties	VINCE CAMUTO	49.99	USD	Women > Shoes > View All	https://www.marshalls.com/us/st
4	Suede Booties	SOLE SOCIETY	39.99	USD	Women > Shoes > View All	https://www.marshalls.com/us/st

```
raw_data_df[raw_data_df['product_name'] == 'Leather Heeled Booties']
```

	product_name	brand	price	currency	breadcrumbs	
0	Leather Heeled Booties	ADRIENNE VITTADINI	39	USD	Women > Shoes > View All	https://www.marshalls.com/us
201	Leather Heeled Booties	ADRIENNE VITTADINI	39	USD	Women > Shoes > View All	https://www.marshalls.com/us

```
print('Following columns we have in our data!!!')
raw_data_df.columns
```

```
Following columns we have in our data!!!
Index(['product_name', 'brand', 'price', 'currency', 'breadcrumbs', 'pdp_url',
       'variants', 'images', 'description'],
      dtype='object')
```

```
print('Shape of data is: {}'.format(raw_data_df.shape))
```

```
Shape of data is: (101902, 9)
```

Checking for null entries

```
print('Looking for null values!!!')
print(raw_data_df.isnull().sum())
```

```
Looking for null values!!!
product_name     0
brand            0
price            0
currency         0
breadcrumbs      0
pdp_url          0
variants         0
images          28
description     28
dtype: int64
```

28 data null data points for images and description, dropping them for now...

```
raw_data_df.dropna(inplace = True)
print(raw_data_df.isnull().sum())
print('Shape of data is: {}'.format(raw_data_df.shape))
```

```
product_name    0
brand           0
price           0
currency        0
breadcrumbs     0
pdp_url         0
variants        0
images          0
description     0
dtype: int64
Shape of data is: (101846, 9)
```

```
raw_data_df.head()
```

	product_name	brand	price	currency	breadcrumbs	
0	Leather Heeled Booties	ADRIENNE VITTADINI	39	USD	Women > Shoes > View All	https://www.marshalls.com/us/st
1	Buckle Wedge Suede Booties	SOLE SOCIETY	29	USD	Women > Shoes > View All	https://www.marshalls.com/us/st
2	Suede Whipstitch Booties	1.STATE	29	USD	Women > Shoes > View All	https://www.marshalls.com/us/st
3	Leather Snake Booties	VINCE CAMUTO	49.99	USD	Women > Shoes > View All	https://www.marshalls.com/us/st
					Women >	

```
raw_data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 101846 entries, 0 to 101901
Data columns (total 9 columns):
 #   Column        Non-Null Count   Dtype 
---  ------        --------------   ----- 
 0   product_name  101846 non-null  object
 1   brand         101846 non-null  object
 2   price         101846 non-null  object
 3   currency      101846 non-null  object
 4   breadcrumbs   101846 non-null  object
 5   pdp_url       101846 non-null  object
 6   variants      101846 non-null  object
 7   images        101846 non-null  object
 8   description   101846 non-null  object
dtypes: object(9)
memory usage: 7.8+ MB
```

Selecting subset of features

```
#Selecting only the text based columns from all the columns present
upd_raw_data_df = raw_data_df[['product_name', 'brand', 'price', 'breadcrumbs', 'descripti
del raw_data_df
```

```
print('After removing rows for which columns had null values we are left out with {} data
```

```
After removing rows for which columns had null values we are left out with 101846 dat
```

```
upd_raw_data_df.head()
```

	product_name	brand	price	breadcrumbs	description
0	Leather Heeled Booties	ADRIENNE VITTADINI	39	Women > Shoes > View All	lace up vamp, quilted panel, stacked block hee...
1	Buckle Wedge Suede Booties	SOLE SOCIETY	29	Women > Shoes > View All	stacked wedge, strap and buckle detail3.25in. ...
2	Suede Whipstitch Booties	1.STATE	29	Women > Shoes > View All	whipstitch accent3.25in. heel3.5in. shaft heig...
3	Leather Snake	VINCE	49.99	Women > Shoes	snake embossed design, block

Checking duplicates

```
upd_raw_data_df[upd_raw_data_df['product_name'] == 'Leather Heeled Booties']
```

	product_name	brand	price	breadcrumbs	description
0	Leather Heeled Booties	ADRIENNE VITTADINI	39	Women > Shoes > View All	lace up vamp, quilted panel, stacked block hee...
201	Leather Heeled	ADRIENNE	39	Women > Shoes	lace up vamp, quilted panel,

```
upd_raw_data_df[upd_raw_data_df['product_name'] == 'Buckle Wedge Suede Booties']
```

	product_name	brand	price	breadcrumbs	description
1	Buckle Wedge Suede Booties	SOLE SOCIETY	29	Women > Shoes > View All	stacked wedge, strap and buckle detail3.25in. ...
202	Buckle Wedge Suede Booties	SOLE SOCIETY	29	Women > Shoes > View All	stacked wedge, strap and buckle detail3.25in. ...
310	Buckle Wedge Suede Booties	SOLE SOCIETY	39.99	Women > Shoes > View All	stacked wedge, strap and buckle detail3.25in. ...

```
dedup_raw_data = upd_raw_data_df.drop_duplicates(subset = {'product_name'}, keep = 'first'
print('We are left with {} rows after removing duplicate entries!!!!'.format(dedup_raw_dat
```

```
We are left with 47630 rows after removing duplicate entries!!!!
```

```
print('Number of unique levels for brand and breadcrums are: {}, {}'.format(dedup_raw_data
```

```
Number of unique levels for brand and breadcrums are: 1041, 3411
```

Text Data Analysis

```
data_df = dedup_raw_data.rename(columns = {'product_name' : 'title'})
```

```
data_df.shape
```

```
(47630, 5)
```

```
#del dedup_raw_data
```

Text Pre Processing

Let us start by looking at some of the entries that we have both for title and description

```
data_df['title'].unique()
```

```
array(['Leather Heeled Booties', 'Buckle Wedge Suede Booties',
       'Suede Whipstitch Booties', ..., 'Classic Short Spill Seam Boot',
       'Customizable Bailey Bow Mini Boot',
       'Mini Bailey Button Bling II Boot'], dtype=object)
```

```
data_df['description'].unique()
```

```
array(['lace up vamp, quilted panel, stacked block heel3.5in. heel4in. shaft height,
       'stacked wedge, strap and buckle detail3.25in. heel3.5in. shaft height, 12in.
       'whipstitch accent3.25in. heel3.5in. shaft height, 10.75in. shaft circumferenc
       ...,
       'A puffer jacket in shoe form. Packed with very grown-up cold weather technolc
       'Plush wool seams lend this Classic boot a laid-back, California vibe and the
       'For those who take glamour seriously, one solid Swarovski® crystal doubles as
      dtype=object)
```

```
#Taking data points where we have alteast 2 words in title
data_df = data_df[data_df['title'].apply(lambda x: len(x.split()) >= 2)]
```

```
data_df.shape
```

```
(47626, 5)
```

```
#Converting into lower case
data_df['title'] = data_df['title'].apply(lambda x : x.lower())
data_df['description'] = data_df['description'].apply(lambda x : x.lower())
data_df['brand'] = data_df['brand'].apply(lambda x : x.lower())
```

```
data_df[data_df['title'].str.contains('corral')]
```

	title	brand	price	breadcrumbs	description
14338	corral boots q0157	corral boots	80.99	Shoes > Boots > Corral Boots	the corral® q0157 cowgirl boots offers a beaut...
14500	corral boots l5474	corral boots	74.99	Shoes > Boots > Corral Boots	the corral™ l5474 cowgirl boots are perfect fo...
14840	corral boots l5562	corral boots	74.99	Shoes > Boots > Corral Boots	the corral™ l5562 cowgirl boots set your look ...
15175	corral boots e1581	corral boots	35.40	Shoes > Sneakers & Athletic Shoes > Corral...	the corral® e1581 slip-on shoes have a comfort...

```
#Removing digits and words containing digits
data_df['title'] = data_df['title'].apply(lambda x: re.sub('\w*\d\w*',' ', x))
data_df['description'] = data_df['description'].apply(lambda x: re.sub('\w*\d\w*',' ', x))
```

	...	...	...	...	...	...

```
#Remove Punctuations
data_df['title'] = data_df['title'].apply(lambda x: re.sub('[%s]' % re.escape(string.punct
data_df['description'] = data_df['description'].apply(lambda x: re.sub('[%s]' % re.escape(


#Removing special character after a word, like if you see in above example we see in descr
#removing all such characters from title and description
data_df['description'] = [' '.join(word if word[-1]>='a' and word[-1]<='z' else word[0:-1]
data_df['title'] = [' '.join(word if word[-1]>='a' and word[-1]<='z' else word[0:-1] for w


data_df[data_df['title'].str.contains('corral')]
```

This is one thing that needs to be discussed, since we removed words with digits from title as well, now all the disctinct entries for corral boots have same title and going forward we'll keep only one such data point, so is this pre processing step is fine or should we include all data pts by not eliminating such words from title. At some point we'll use title also for our moel and that was the reason why we are cleaning our title as well.....

boots boots Shoes > Boots > Corral Boots your look apart w...

```
#removing additional spaces between words
data_df['title'] = data_df['title'].str.replace('\s+', ' ')
data_df['description'] = data_df['description'].str.replace('\s+', ' ')
```

16163 boots boots 119.99 Shoes > Boots > Corral Boots overlays and

```
print('10 Most frequent products')
Counter(list(data_df['title'])).most_common(10)
```

```
10 Most frequent products
[('corral boots', 162),
 ('giuseppe zanotti', 129),
 ('gabor gabor', 99),
 ('fly london', 89),
 ('new balance', 74),
 ('blundstone', 65),
 ('rieker', 55),
 ('twisted x', 48),
 ('see by chloe', 29),
 ('gbg los angeles', 24)]
```

```
data_df = data_df.drop_duplicates(subset = {'title'}, keep = 'first')
```

```
data_df.reset_index(drop = True, inplace = True)
```

```
#Removing brand details from title and description
data_df['description'] = [' '.join(word for word in desc.split(' ') if word not in set(bra
data_df['title'] = [' '.join(word for word in title.split(' ') if word not in set(brand.sp
```

```
#removing titles from description
#data_df['description'] = [' '.join(word for word in desc.split(' ') if word not in set(ti
```

```
data_df.shape
```

```
(45185, 5)
```

```
data_df = data_df[data_df['title'].apply(lambda x: len(x.split()) >= 2)]
```

```
data_df.shape
```

```
(30294, 5)
```

One thought that came to mind is that can we even remove the words we have in title from the description ??? Open question this is, I am not sure about it... At times it makes sense, at times it doesn't... After trying various techniques with the first method itself, we'll see where are we going wrong, we'll try to have a discussion on this and then we'll re-visit the pre processing part of text...

One important thing that was observed that we have data points in title with mixed information, have seen that like may be boots and sneaker or sandal and sneaker in one data entry... will see when we'll get this information and show that in notebook as well.....

Data Pre-Processing steps done

1. Taken data points where we have alteast 2 words in title.
2. Converted into lower case.
3. Removed digits and words containing digits.
4. Removed Punctuations.
5. Removed special character after a word.
6. Removed additional spaces between words.
7. Again removed duplicate entries based on title.
8. Removed brand details from title and description.
9. Taken rows with title have 2 words or more.
10. Removed words in title from description

Stop words Removal

```
stop = set(stopwords.words('english'))


data_df['description'] = [' '.join(word for word in data.split() if word not in stop) for
data_df['title'] = [' '.join(word for word in data.split() if word not in stop) for data i


remove_words = ['women', 'womens']


data_df['description'] = [' '.join(word for word in data.split() if word not in remove_wor
data_df['title'] = [' '.join(word for word in data.split() if word not in remove_words) fo
```

Stop Word Removal and Lemmatization

```
nlp = spacy.load('en_core_web_sm', disable = ['parser', 'net'])
data_df['title'] = data_df['title'].apply(lambda x : ' '.join([token.lemma_ for token in l
data_df['description'] = data_df['description'].apply(lambda x : ' '.join([token.lemma_ fo
```

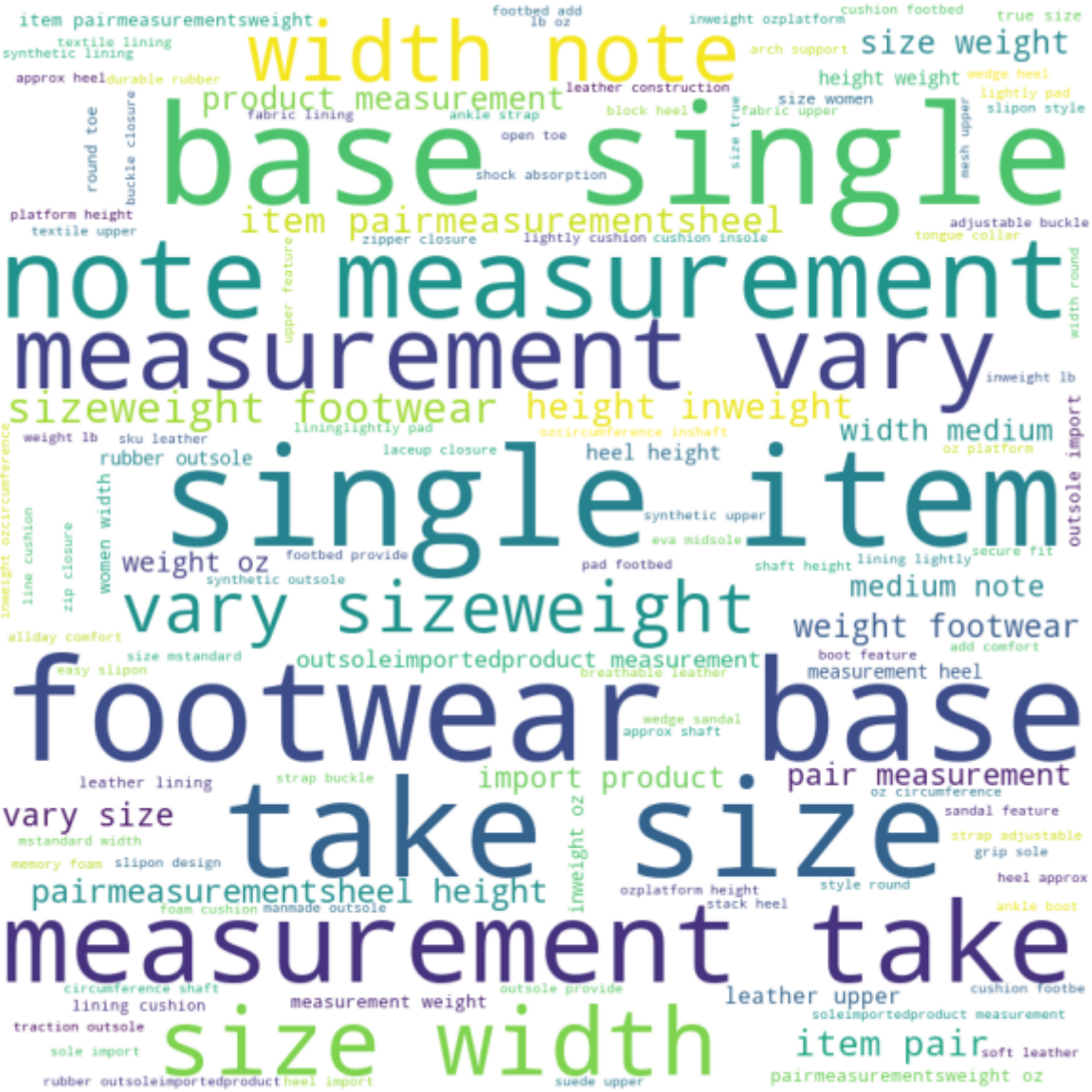
```
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)

for val in data_df['title']:
    tokens = val.split()
    for words in tokens:
        comment_words = comment_words + words + ' '

wordcloud = WordCloud(width = 800, height = 800, background_color = 'white', stopwords = s

plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad = 0)

plt.show()
```


```
data_df[data_df['title'].str.contains('sneaker boot')]
```

	title	brand	price	description
18	cozy lined sneaker boots	mia	29.99	faux fur comfort shaft height shaft circumfere...
331	cozy lined suede sneaker booties	splendid	49.99	fleece warm cushioned shaft height shaft circu...
523	fringe front suede sneaker booties	seychelles	29.99	detail tie shaft height shaft circumference ta...
570	dausette sneaker boot	timberland	79.99	metallic leather upper paired doublerolled cus...
17500	shearling lined sneaker boot	blackstone	65.40	offers topnotch urban appeal suede upper bold ...
37267	womens lanyn wedge sneaker boot	dr. scholl's	59.99	get best style worlds dr scholls bootmicrosued...
37279	womens whoa sneaker boot	dr. scholl's	44.99	get best style worlds dr scholls bootmicrosued...
37311	womens madison high top wedge sneaker boot	dr. scholl's	54.99	sleek bootie hidden lift dr scholls bootlining...
37334	womens skyla bay slip sneaker boot	timberland	79.99	get best style worlds bootsuede upper slipon s...
37532	womens insane sneaker boot	dr. scholl's	54.99	get best worlds dr scholls bootfabric upper ca...
37638	womens whoa slip sneaker bootie	dr. scholl's	54.99	standout slipon ecoconscious designsustainably...
37730	womens rosalind knit sneaker boot lab	dr. scholl's orig collection	39.99	chelsea like none forefront technology design ...
37829	womens hensley mediumwide sneaker boot	ryka	84.99	casual cool bootie goto year roundperforated s...
37865	womens taffy mediumwide sneaker boot	soul naturalizer	69.99	bootfabric upper hightop style round toepullon...
38261	womens gwyn mediumwide wedge sneaker boot	ryka	69.99	made women made winter youll rocking kicks sea...
38351	womens madison hi wedge sneaker boot	dr. scholl's	64.99	sleek bootie hidden liftsustainably crafted ec...
38871	womens bateman slip resistant waterproof sneak...	harley davidson	159.99	bootwaterproof leather upper hightop style rou...
39215	womens mimi wedge sneaker boot	dr. scholl's	84.99	laceup hidden allday comfortsustainably crafte...
39218	womens good wedge sneaker boot	dr. scholl's	79.99	sporty vibessustainably crafted soft linings m...
40028	womens dausette lace sneaker boot	timberland	49.98	tough like youleather upper style round toelac...
40043	womens worries sneaker bootie	dr. scholl's	25.98	genuine nubuck comfort

```
from wordcloud import WordCloud, STOPWORDS
```

```
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)

for val in text_data_df['description']:
    tokens = val.split()
    for words in tokens:
        comment_words = comment_words + words + ' '

wordcloud = WordCloud(width = 800, height = 800, background_color = 'white', stopwords = s

plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad = 0)

plt.show()
```



Adding 2 more features, which are word count of title and description

```
def word_count(val):
    value = []
```

```
    for i in val:
        count = len(i.split())
        value.append(count)
    return value


data_df['word_count_title'] = word_count(data_df['title'])
data_df['word_count_description'] = word_count(data_df['description'])


data_df.head()
```

	title	brand	price	description	word_count_title	word_count_description
0	leather heeled booties	adrienne vittadini	39	lace vamp quilted panel stacked block shaft he...	3	16
1	buckle wedge suede booties	sole society	29	stacked wedge strap buckle shaft height shaft ...	4	18

Text Feature Engg

For text based features, we'll try different feature engg techniques.

1. We'll vectorize using tf-idf method.
2. We'll create features based on co-occurence matrix.
3. We'll vectorize using W2V method.
4. We'll vectorize using ti-idf based W2V method.

Computing Feature Vector using tf-idf

```
vectoriz_tfid_description = TfidfVectorizer(min_df = 10)
description_tfidf = vectoriz_tfid_description.fit_transform(data_df['description'])
print('Shape of matrix is: {}'.format(description_tfidf.shape))
```

```
Shape of matrix is: (30057, 5043)
```

```
vectoriz_tfid_description = TfidfVectorizer(min_df = 10)
title_tfidf = vectoriz_tfid_description.fit_transform(data_df['title'])
print('Shape of matrix is: {}'.format(title_tfidf.shape))
```

```
Shape of matrix is: (30057, 1025)
```

Considering only description

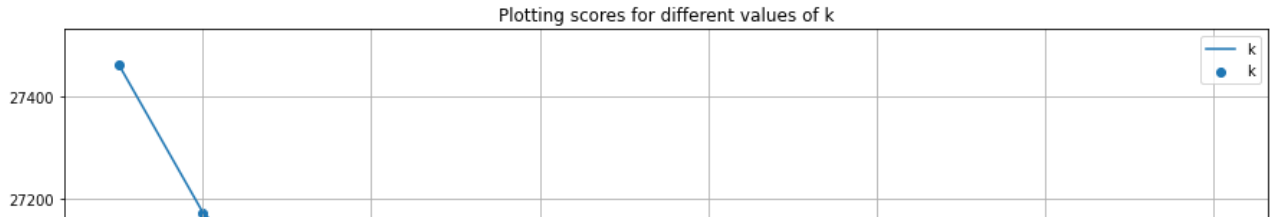
K-Means

Using description to create vectors

```
from sklearn.cluster import KMeans
k = [3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16]
score = []
for i in tqdm(range(len(k))):
    kmeans = KMeans(n_clusters = k[i], random_state = 1).fit(description_tfidf)
    score.append(kmeans.inertia_)
```

```
 100%|██████████| 11/11 [01:04<00:00,  5.83s/it]
```

```
plt.figure(figsize = (15, 10))

plt.plot(k, score, label = 'k')
plt.scatter(k, score, label = 'k')

plt.title('Plotting scores for different values of k')
plt.xlabel('Number of clusters(k)')
plt.ylabel('Loss Score')
plt.legend()
plt.grid()
plt.show()
```




```
#Taking 12 clusters
kmeans = KMeans(n_clusters = 12, random_state = 1).fit(description_tfidf)
kmeans.labels_.shape
```

```
(30057,)
```

```
df = pd.DataFrame(data_df['title'], columns = ['title'])
df['cluster_name'] = kmeans.labels_
df = df.groupby(['cluster_name'])
```

```
from collections import Counter

for key, data in df:
    print('Number of data points in cluster {} are {}'.format(key + 1, data.shape[0]))
    print('Number of words in cluster {} are {}'.format(key + 1, len(' '.join(list(data['t
    counter = Counter(' '.join(list(data['title'])).split())
    most_occur = counter.most_common(10)
    print('Top 10 words for cluster {} are: {} \n\n'.format(key + 1, [most_occur[i][0] for
```

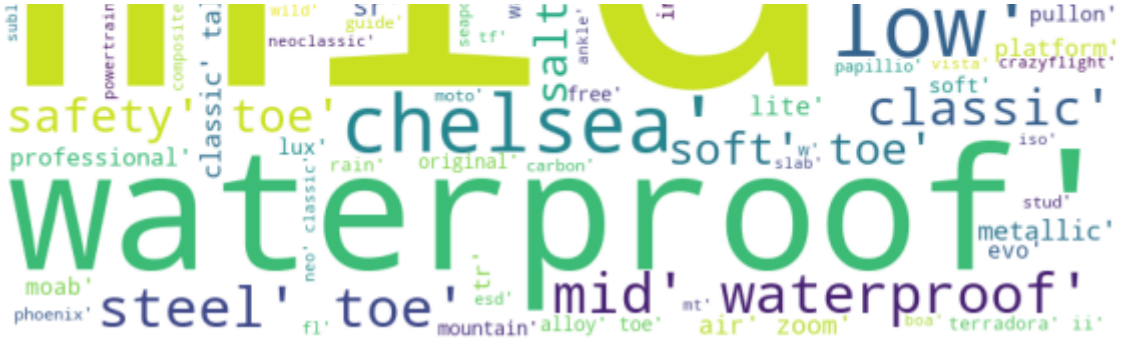
```
Top 10 words for cluster 1 are: ['sandal', 'boot', 'bootie', 'wedge', 'pump', 'jcr


Number of data points in cluster 2 are 2266
Number of words in cluster 2 are 6684
Top 10 words for cluster 2 are: ['dr', 'mid', 'waterproof', 'toe', 'gtx', 'ii', 'b


Number of data points in cluster 3 are 4169
Number of words in cluster 3 are 11467
Top 10 words for cluster 3 are: ['sandal', 'wedge', 'pump', 'womens', 'bootie', 's


Number of data points in cluster 4 are 1271
Number of words in cluster 4 are 3642
Top 10 words for cluster 4 are: ['boot', 'bootie', 'riding', 'wide', 'calf', 'wedg


Number of data points in cluster 5 are 2851
Number of words in cluster 5 are 8104
Top 10 words for cluster 5 are: ['sneaker', 'sandal', 'slipon', 'flat', 'slide', '


Number of data points in cluster 6 are 2071
Number of words in cluster 6 are 9018
Top 10 words for cluster 6 are: ['womens', 'sandal', 'boot', 'mediumwide', 'sneake


Number of data points in cluster 7 are 3703

Number of words in cluster 7 are 13107
Top 10 words for cluster 7 are: ['womens', 'sandal', 'sneaker', 'slipon', 'wedge',
```

```
Number of data points in cluster 8 are 2545
Number of words in cluster 8 are 11204
Top 10 words for cluster 8 are: ['boot', 'sandal', 'bootie', 'pump', 'available',


Number of data points in cluster 9 are 1563
Number of words in cluster 9 are 6424
Top 10 words for cluster 9 are: ['sneaker', 'sandal', 'flat', 'leather', 'availabl


Number of data points in cluster 10 are 2851
Number of words in cluster 10 are 7968
Top 10 words for cluster 10 are: ['sandal', 'boot', 'slide', 'sneaker', 'dr', 'lar


Number of data points in cluster 11 are 1119
Number of words in cluster 11 are 2859
Top 10 words for cluster 11 are: ['sandal', 'wedge', 'pump', 'bootie', 'platform',


Number of data points in cluster 12 are 520
Number of words in cluster 12 are 2116
Top 10 words for cluster 12 are: ['leather', 'booties', 'boots', 'suede', 'comfort
```

```
for key, data in df:
    print('For cluster number {} word cloud is:'.format(key))

    wordcloud = WordCloud(width = 800, height = 800, background_color = 'white', stopwords
    plt.figure(figsize = (8, 8), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.title('Word Cloud for cluster number {}'.format(key))
    plt.tight_layout(pad = 0)

    plt.show()
```

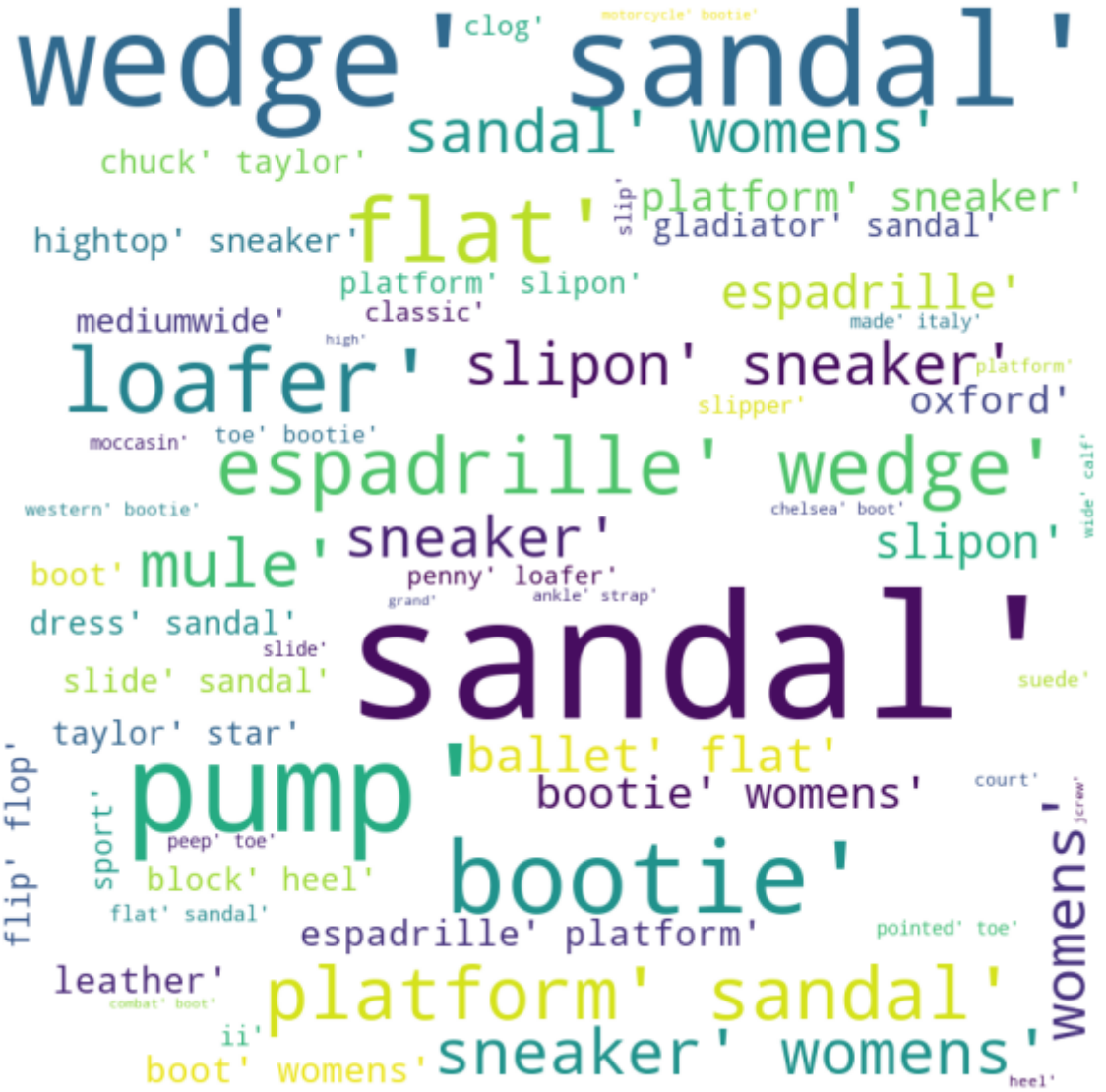
```
For cluster number 0 word cloud is:
```


```
For cluster number 1 word cloud is:
```



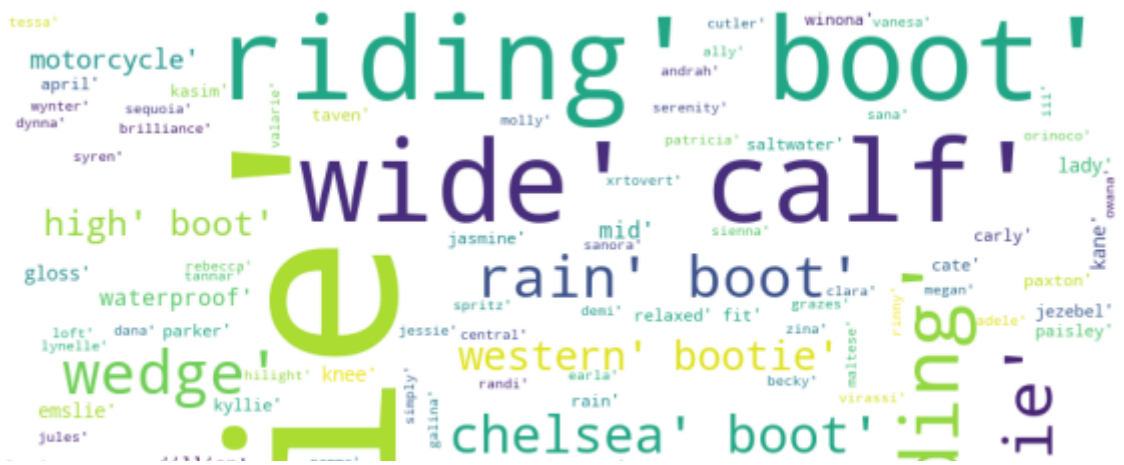
For cluster number 2 word cloud is:

Word Cloud for cluster number 2



For cluster number 3 word cloud is:

Word Cloud for cluster number 3

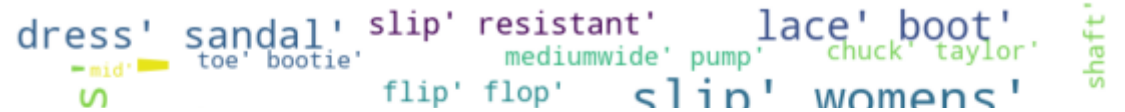


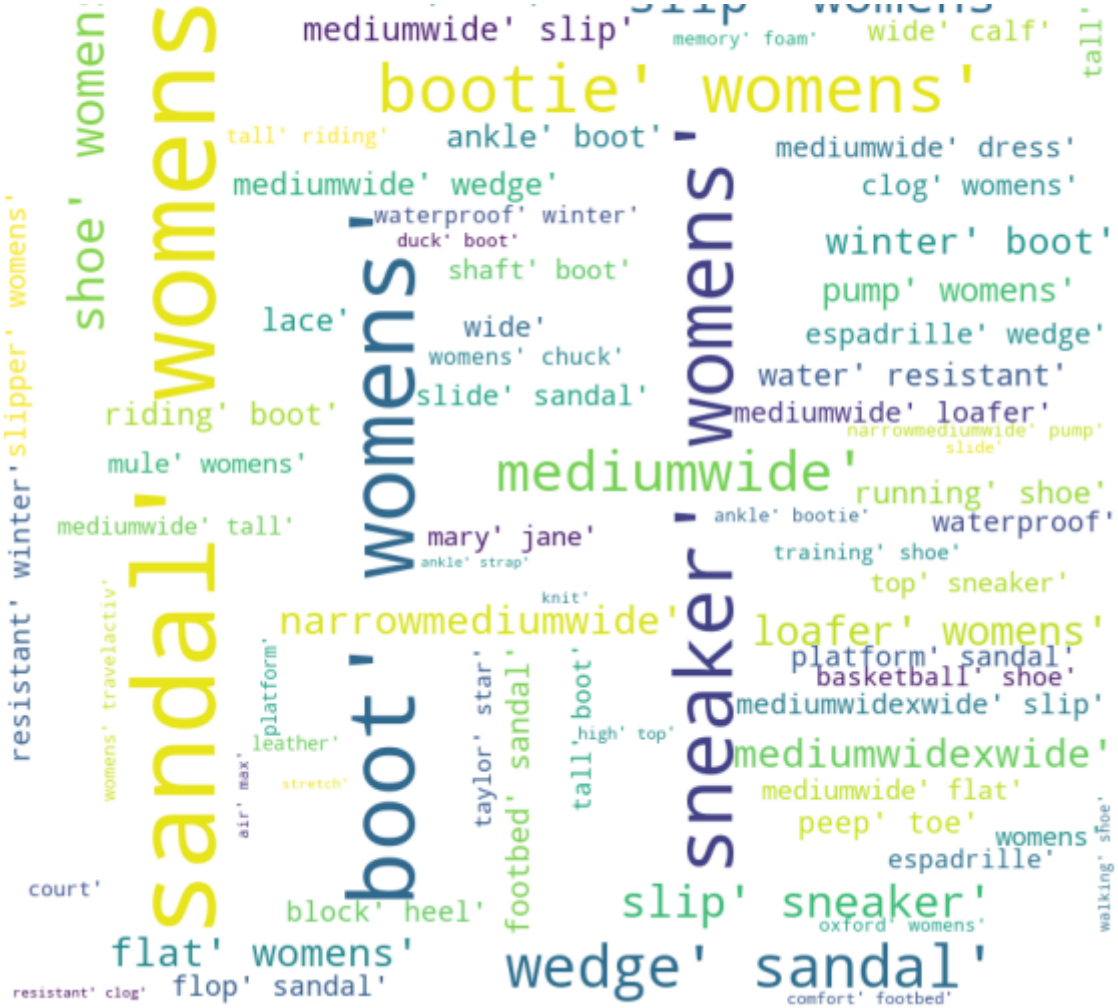
For cluster number 4 word cloud is:

Word Cloud for cluster number 4

For cluster number 5 word cloud is:

Word Cloud for cluster number 5

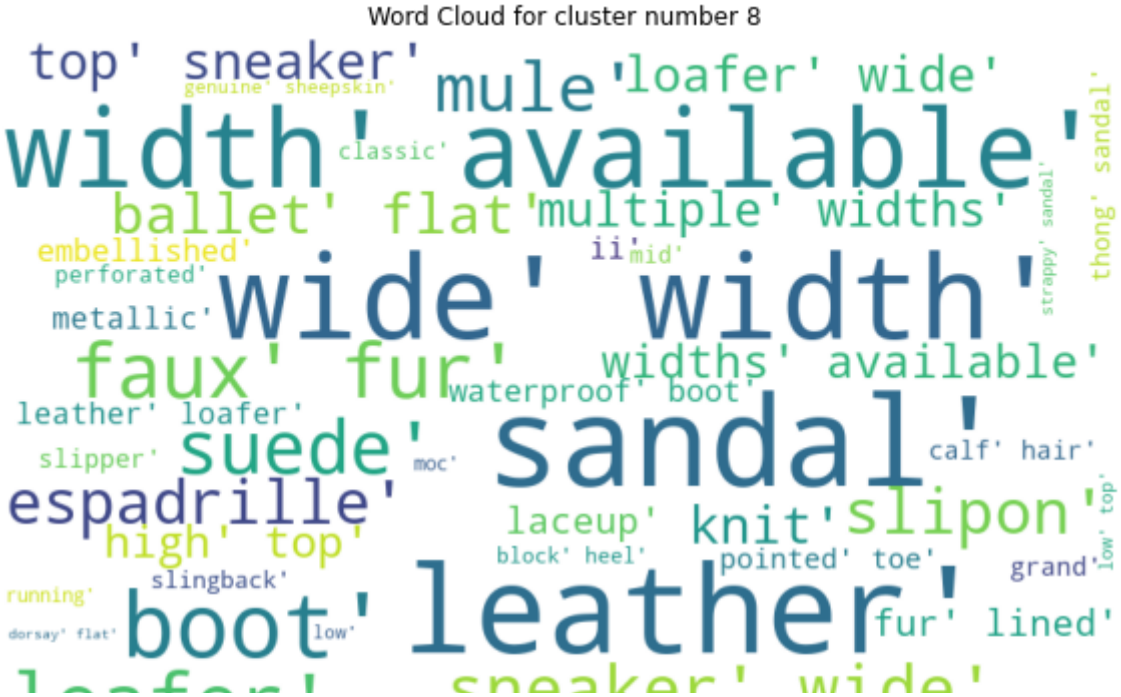




For cluster number 6 word cloud is:

For cluster number 7 word cloud is:

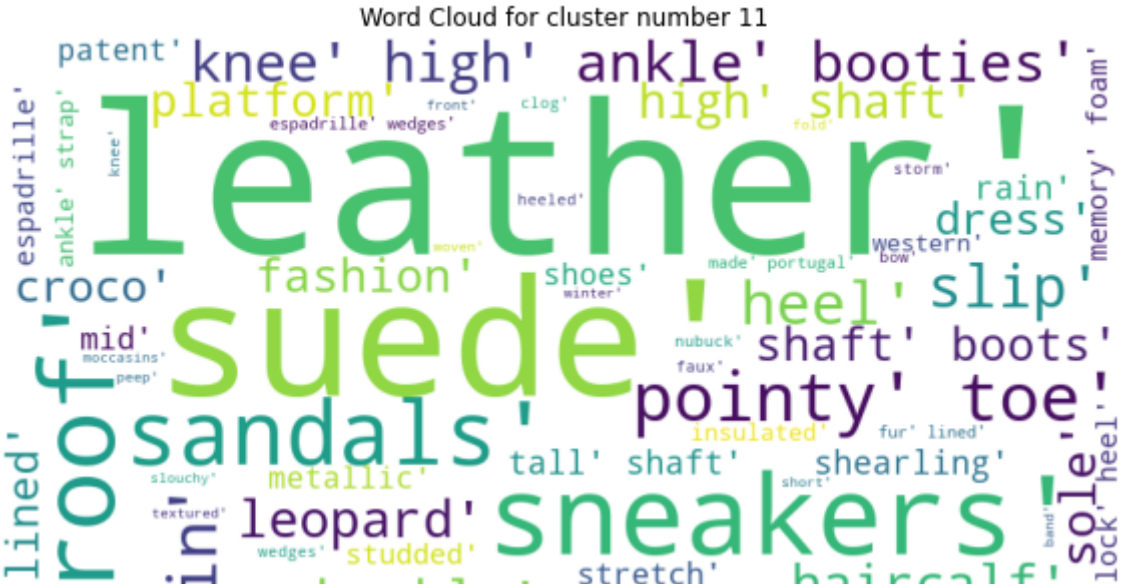
For cluster number 8 word cloud is:



For cluster number 9 word cloud is:

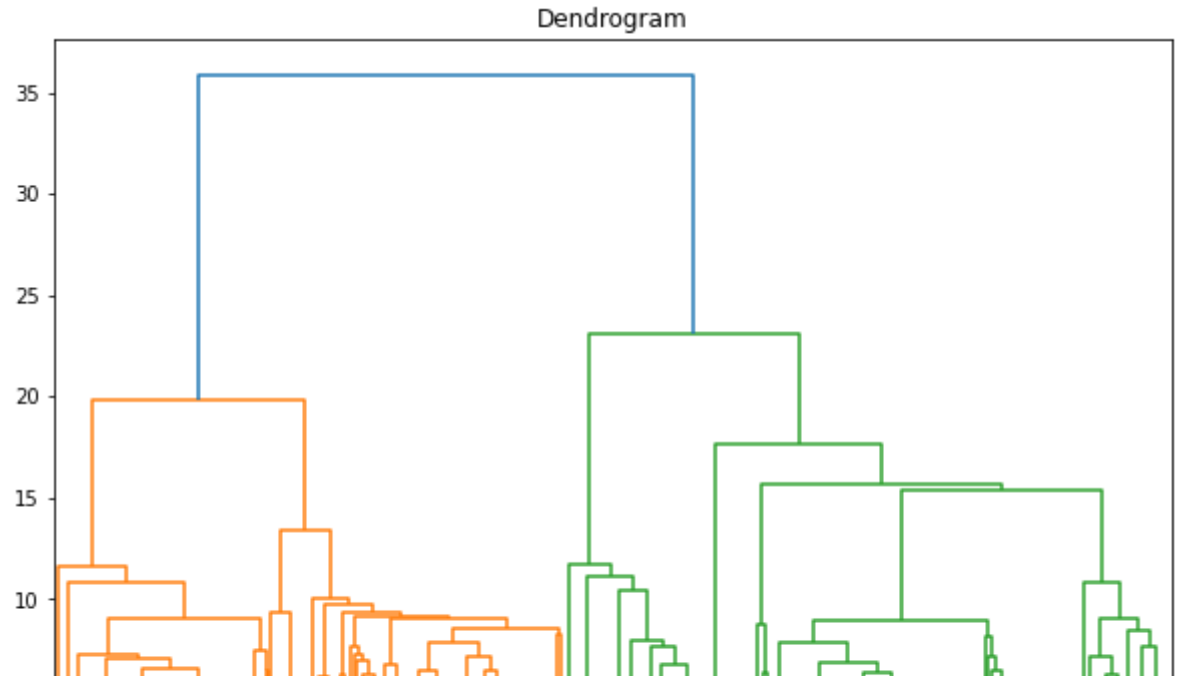
For cluster number 10 word cloud is:

For cluster number 11 word cloud is:



Agglomerative Clustering

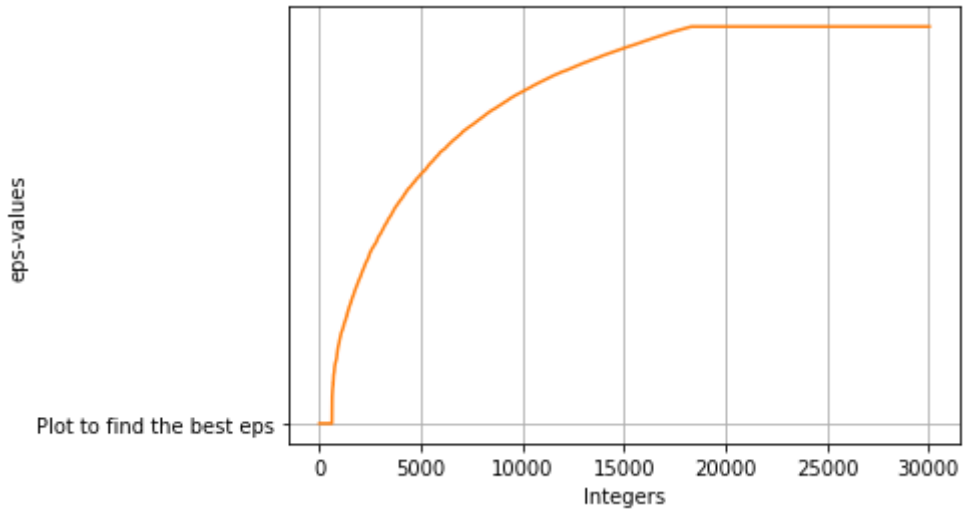
```
import scipy.cluster.hierarchy as cls

algo_title = 'Agglomerative Clustering'
plt.figure(figsize = (10, 7))
plt.title('Dendrogram')
dend = cls.dendrogram(cls.linkage(description_tfidf.toarray(), method = 'ward'))
```



DBSCAN

```
from sklearn.neighbors import NearestNeighbors

neigh = NearestNeighbors(n_neighbors = 2)
nbrs = neigh.fit(description_tfidf)
distances, indices = nbrs.kneighbors(description_tfidf)


distances = np.sort(distances, axis = 0)
distances = distances[:, 1]
plt.plot('Plot to find the best eps')
plt.xlabel('Integers')
plt.ylabel('eps-values')
plt.grid()
plt.plot(distances)
```

```
[<matplotlib.lines.Line2D at 0x7feffd6bd550>]
```




```
description_tfidf.shape
```

```
(30057, 5043)
```

```
from sklearn.cluster import DBSCAN
db = DBSCAN(eps = 15000).fit(description_tfidf.toarray())
```

```
labels = db.labels_
number_of_cluster = len(set(labels)) - (1 if -1 in labels else 0) #Ignoring noise
print('Number of cluster is: {}'.format(number_of_cluster))
```

```
Number of cluster is: 1
```

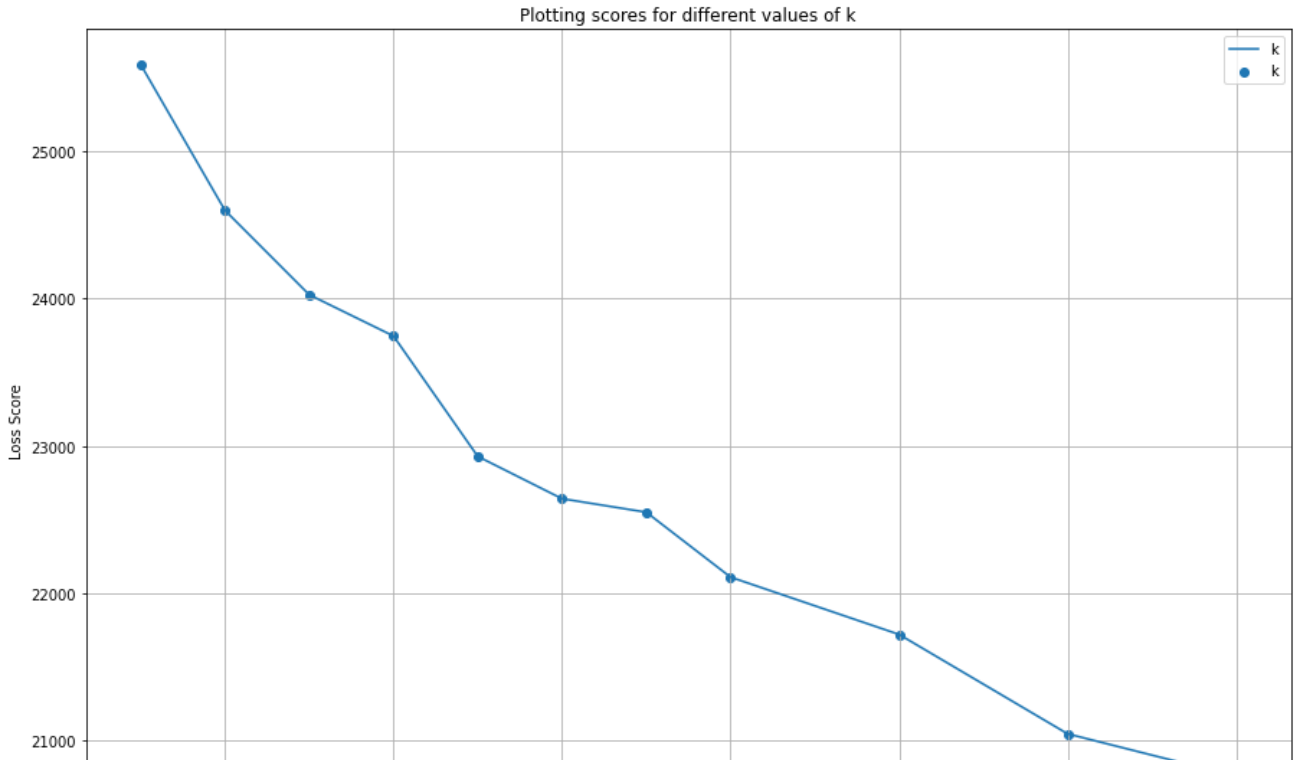
Considering Title Vector

K-Means

```
from sklearn.cluster import KMeans
k = [3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16]
score = []
for i in tqdm(range(len(k))):
    kmeans = KMeans(n_clusters = k[i], random_state = 1).fit(title_tfidf)
    score.append(kmeans.inertia_)
```

```
100%|██████████| 11/11 [00:16<00:00,  1.54s/it]
```

```
plt.figure(figsize = (15, 10))

plt.plot(k, score, label = 'k')
plt.scatter(k, score, label = 'k')

plt.title('Plotting scores for different values of k')
plt.xlabel('Number of clusters(k)')
plt.ylabel('Loss Score')
plt.legend()
plt.grid()
plt.show()
```




```
kmeans = KMeans(n_clusters = 14, random_state = 1).fit(title_tfidf)
kmeans.labels_.shape
```

```
(30057,)
```

```
df = pd.DataFrame(data_df['title'], columns = ['title'])
df['cluster_name'] = kmeans.labels_
df = df.groupby(['cluster_name'])
```

```
from collections import Counter

for key, data in df:
    print('Number of data points in cluster {} are {}'.format(key + 1, data.shape[0]))
    print('Number of words in cluster {} are {}'.format(key + 1, len(' '.join(list(data['t
    counter = Counter(' '.join(list(data['title'])).split())
    most_occur = counter.most_common(10)
    print('Top 10 words for cluster {} are: {} \n\n'.format(key + 1, [most_occur[i][0] for
```

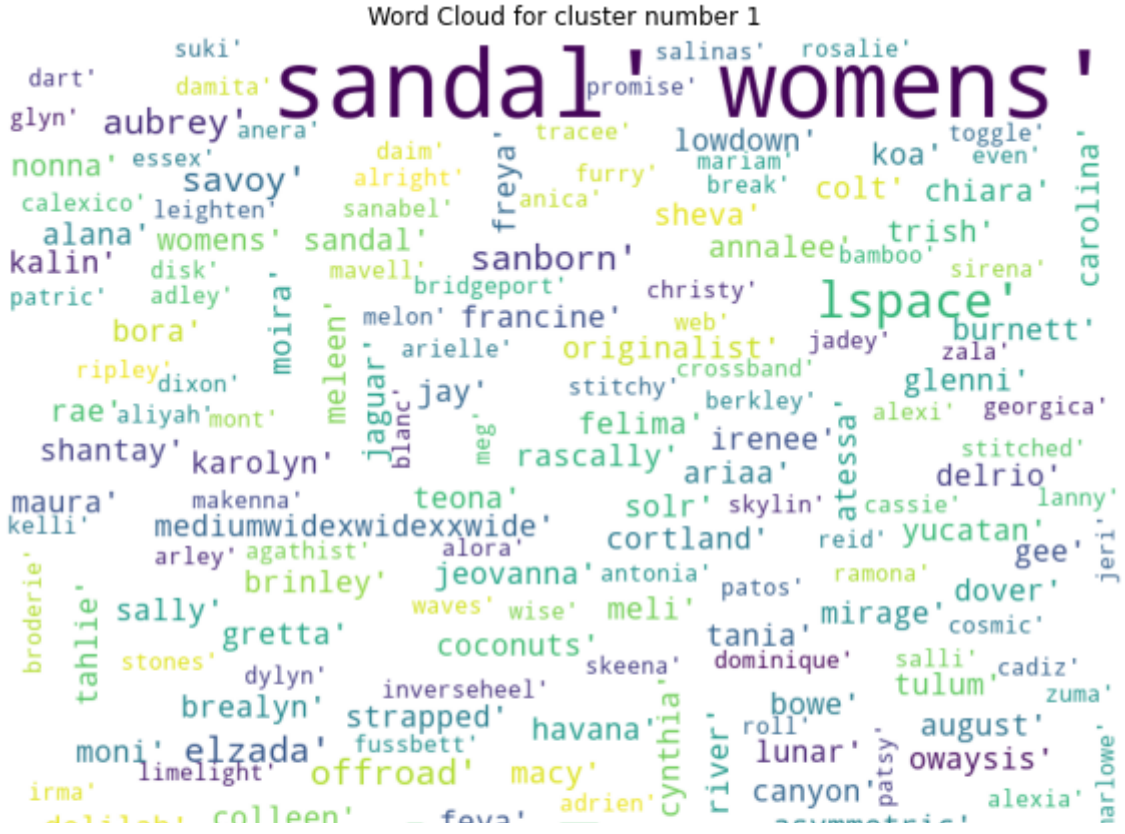
```
Number of data points in cluster 1 are 935
Number of words in cluster 1 are 4164
Top 10 words for cluster 1 are: ['leather', 'sandal', 'boot', 'boots', 'sneaker',


Number of data points in cluster 2 are 1580
Number of words in cluster 2 are 3396
Top 10 words for cluster 2 are: ['sandal', 'womens', 'lspace', 'aubrey', 'elzada',


Number of data points in cluster 3 are 1396
Number of words in cluster 3 are 3406
Top 10 words for cluster 3 are: ['bootie', 'womens', 'wedge', 'western', 'ankle',


Number of data points in cluster 4 are 13614
```

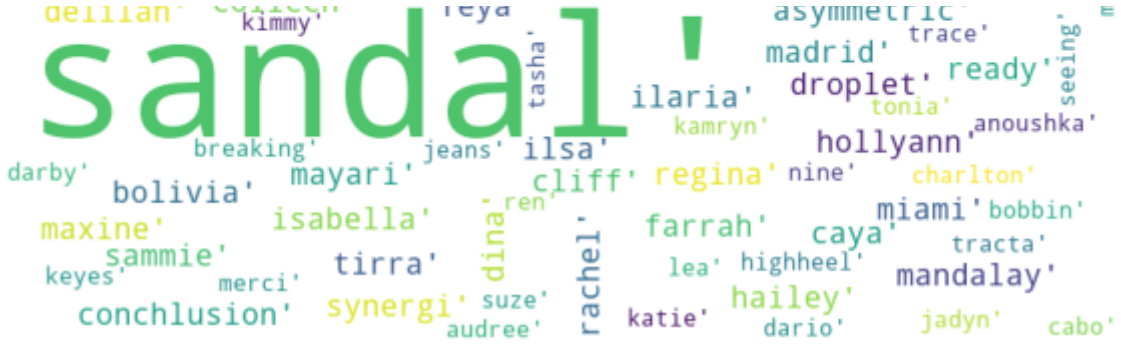
```
Number of words in cluster 4 are 44010
Top 10 words for cluster 4 are: ['sandal', 'womens', 'bootie', 'shoe', 'toe', 'wid


Number of data points in cluster 5 are 745
Number of words in cluster 5 are 2643
Top 10 words for cluster 5 are: ['platform', 'sandal', 'sneaker', 'espadrille', 'w


Number of data points in cluster 6 are 1030
Number of words in cluster 6 are 2940
Top 10 words for cluster 6 are: ['flat', 'ballet', 'womens', 'sandal', 'mediumwide


Number of data points in cluster 7 are 1068
Number of words in cluster 7 are 2681
Top 10 words for cluster 7 are: ['pump', 'wedge', 'womens', 'mediumwide', 'slingba


Number of data points in cluster 8 are 276
Number of words in cluster 8 are 1019
Top 10 words for cluster 8 are: ['classic', 'boot', 'mini', 'short', 'clog', 'wome


Number of data points in cluster 9 are 1766
Number of words in cluster 9 are 6002
Top 10 words for cluster 9 are: ['sneaker', 'womens', 'slipon', 'slip', 'wedge', '


Number of data points in cluster 10 are 2123
Number of words in cluster 10 are 6217
Top 10 words for cluster 10 are: ['mule', 'slipon', 'slide', 'dr', 'sandal', 'scho


Number of data points in cluster 11 are 2644
Number of words in cluster 11 are 10237
Top 10 words for cluster 11 are: ['boot', 'womens', 'wide', 'calf', 'riding', 'wat


Number of data points in cluster 12 are 797
Number of words in cluster 12 are 2185
Top 10 words for cluster 12 are: ['loafer', 'womens', 'penny', 'mediumwide', 'wedg
```

```
for key, data in df:
    print('For cluster number {} word cloud is:'.format(key))

    wordcloud = WordCloud(width = 800, height = 800, background_color = 'white', stopwords
    plt.figure(figsize = (8, 8), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.title('Word Cloud for cluster number {}'.format(key))
    plt.tight_layout(pad = 0)

    plt.show()
```

For cluster number 0 word cloud is:

For cluster number 1 word cloud is:



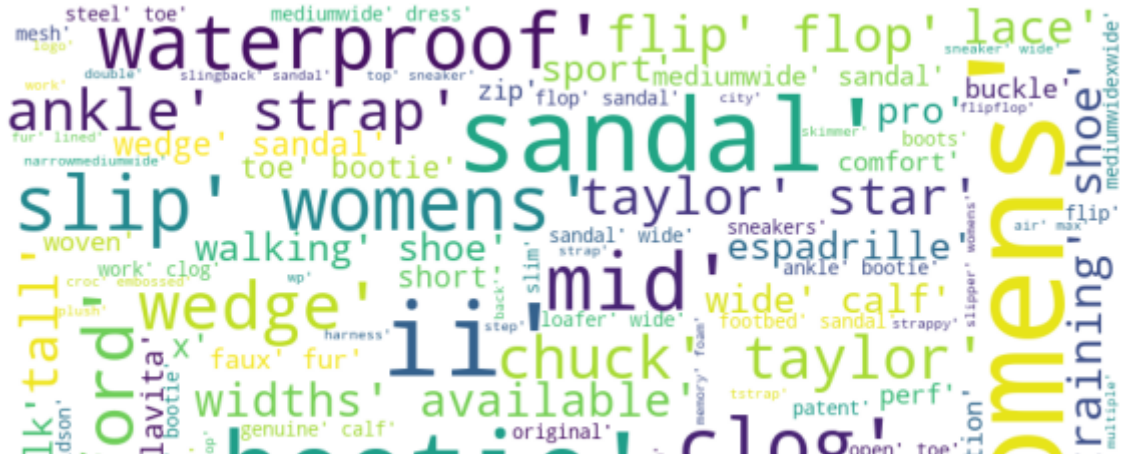


For cluster number 2 word cloud is:

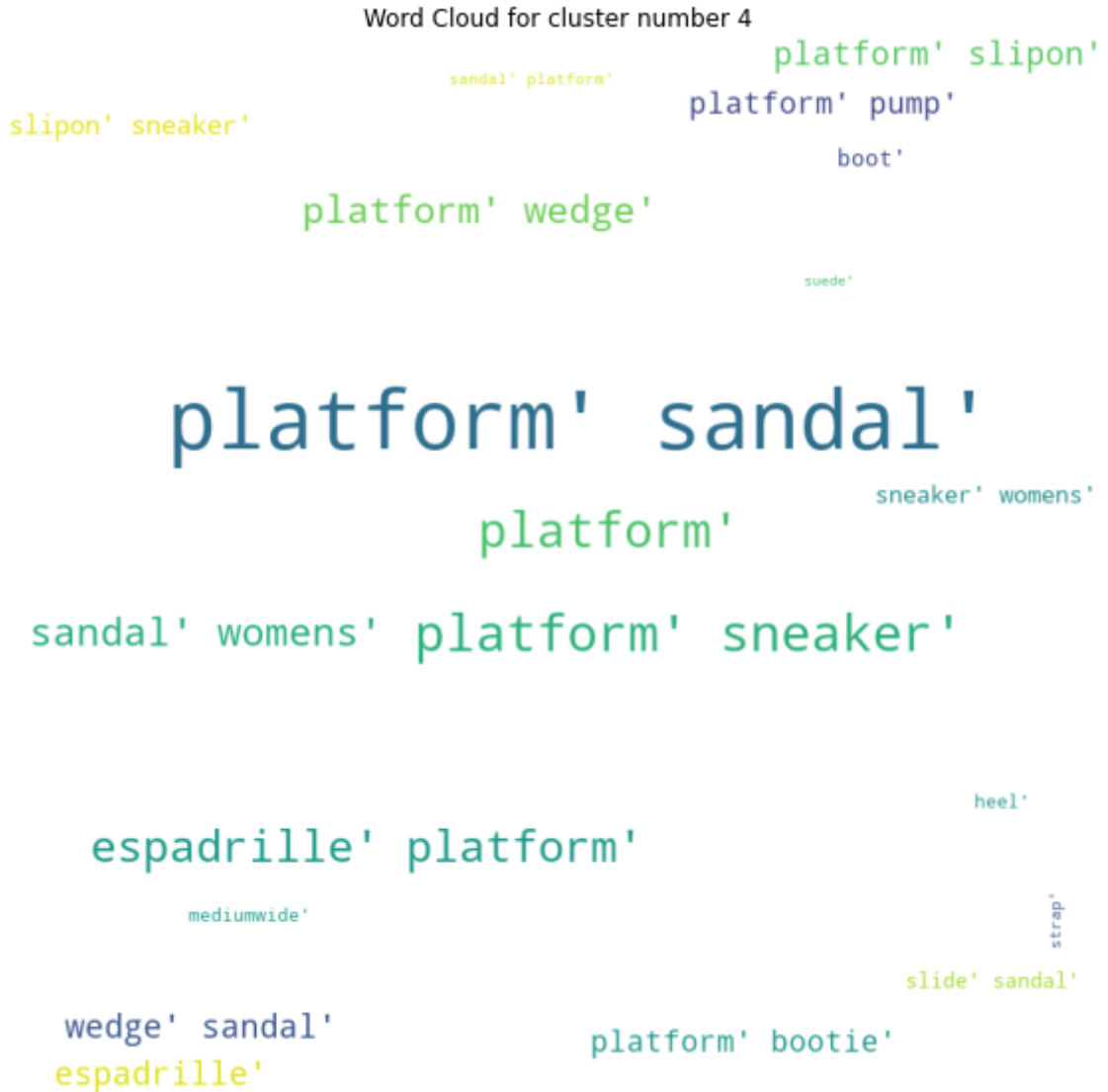
Word Cloud for cluster number 2

For cluster number 3 word cloud is:

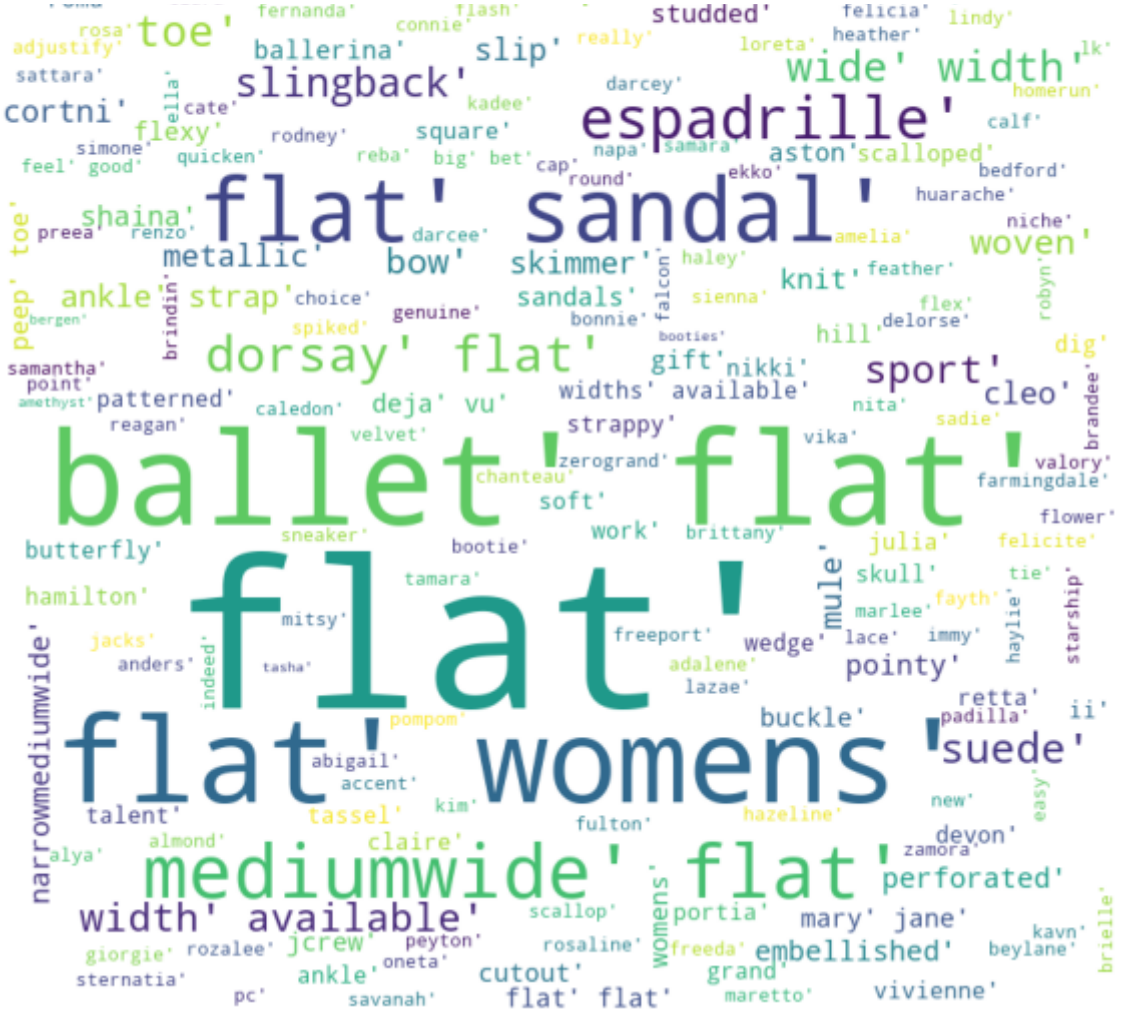
Word Cloud for cluster number 3



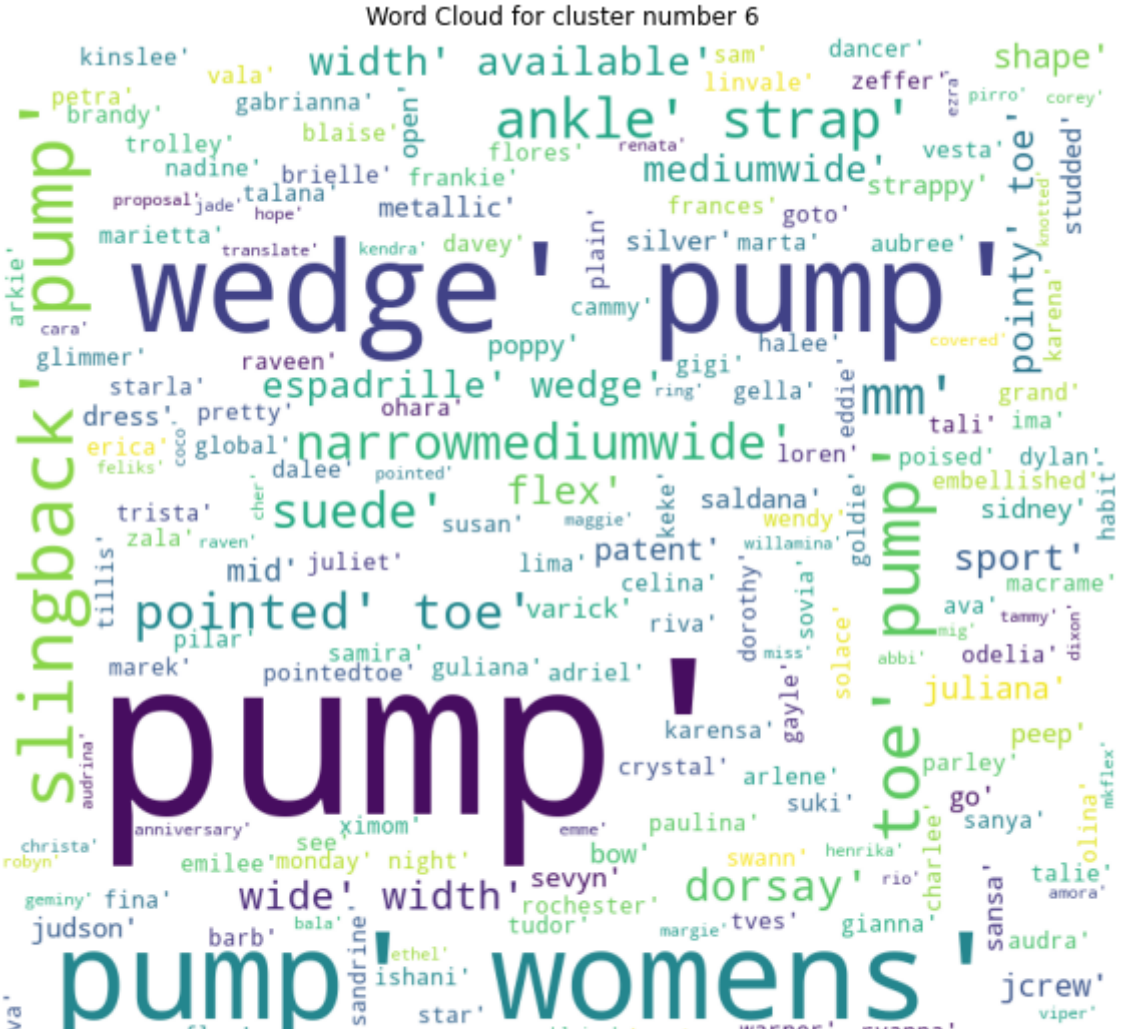
For cluster number 4 word cloud is:



For cluster number 5 word cloud is:



For cluster number 6 word cloud is:



For cluster number 7 word cloud is:

Word Cloud for cluster number 7

For cluster number 8 word cloud is:

Word Cloud for cluster number 8

For cluster number 9 word cloud is:

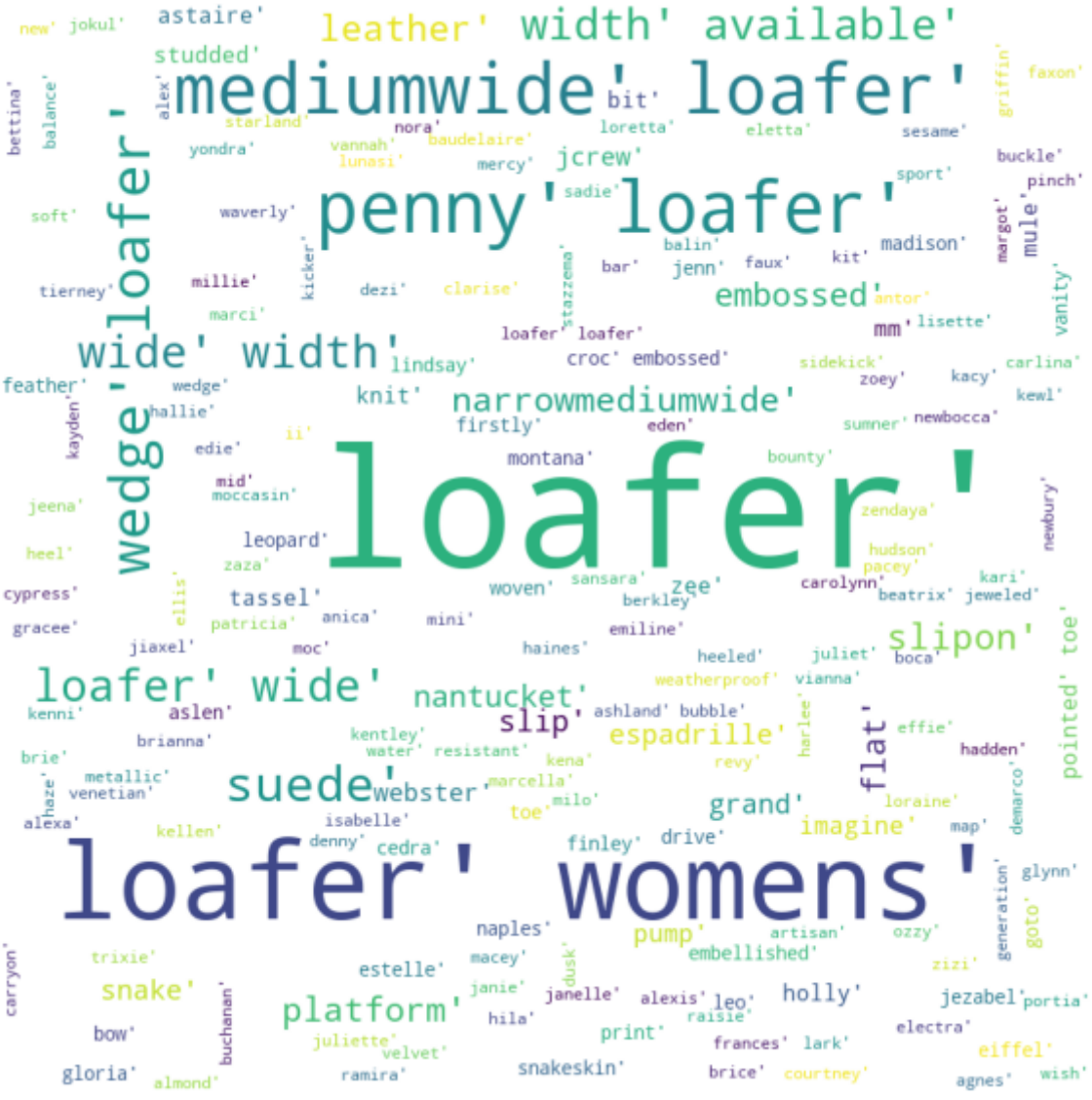
Word Cloud for cluster number 9

For cluster number 10 word cloud is:

Word Cloud for cluster number 10

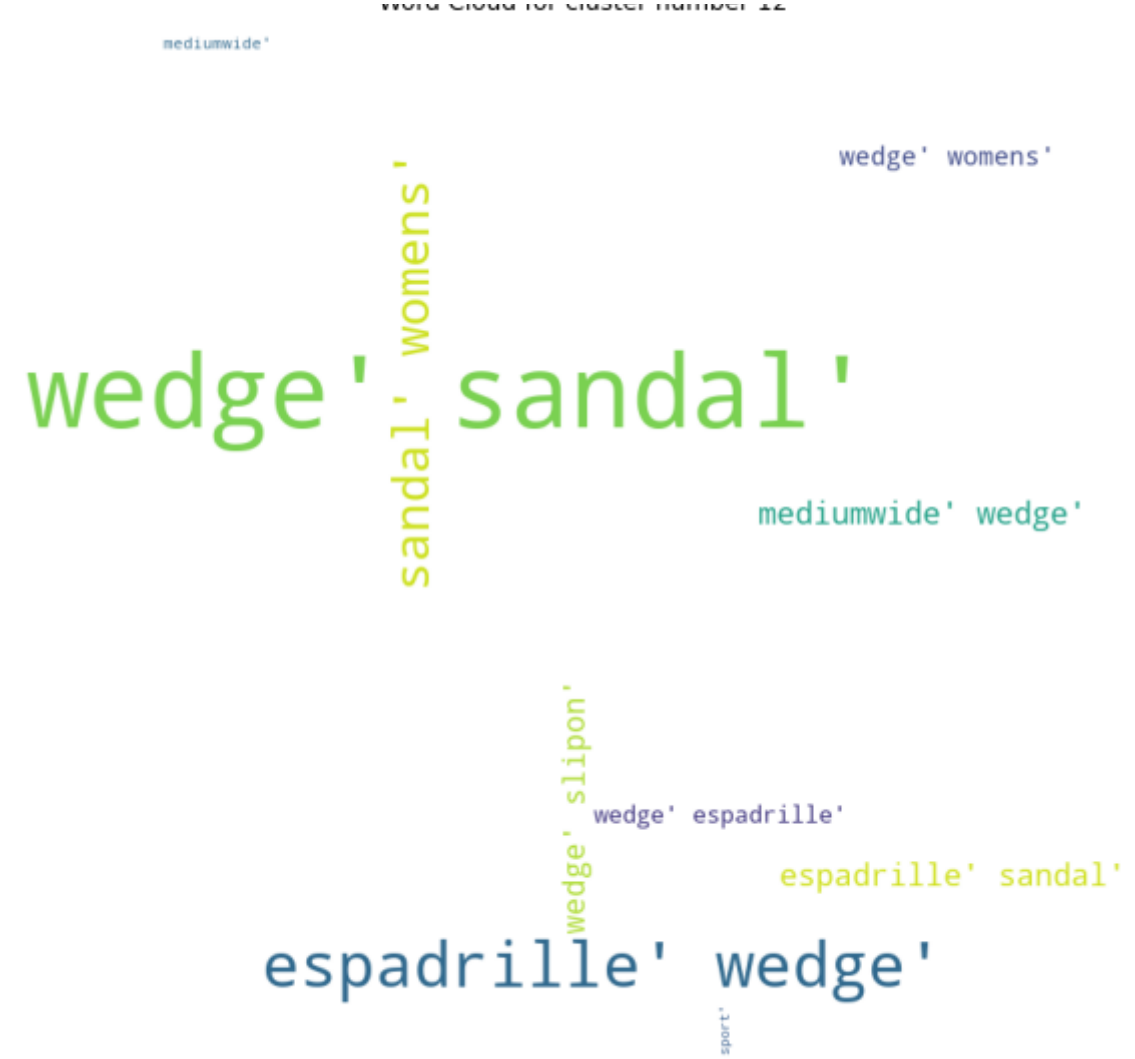
For cluster number 11 word cloud is:

Word Cloud for cluster number 11

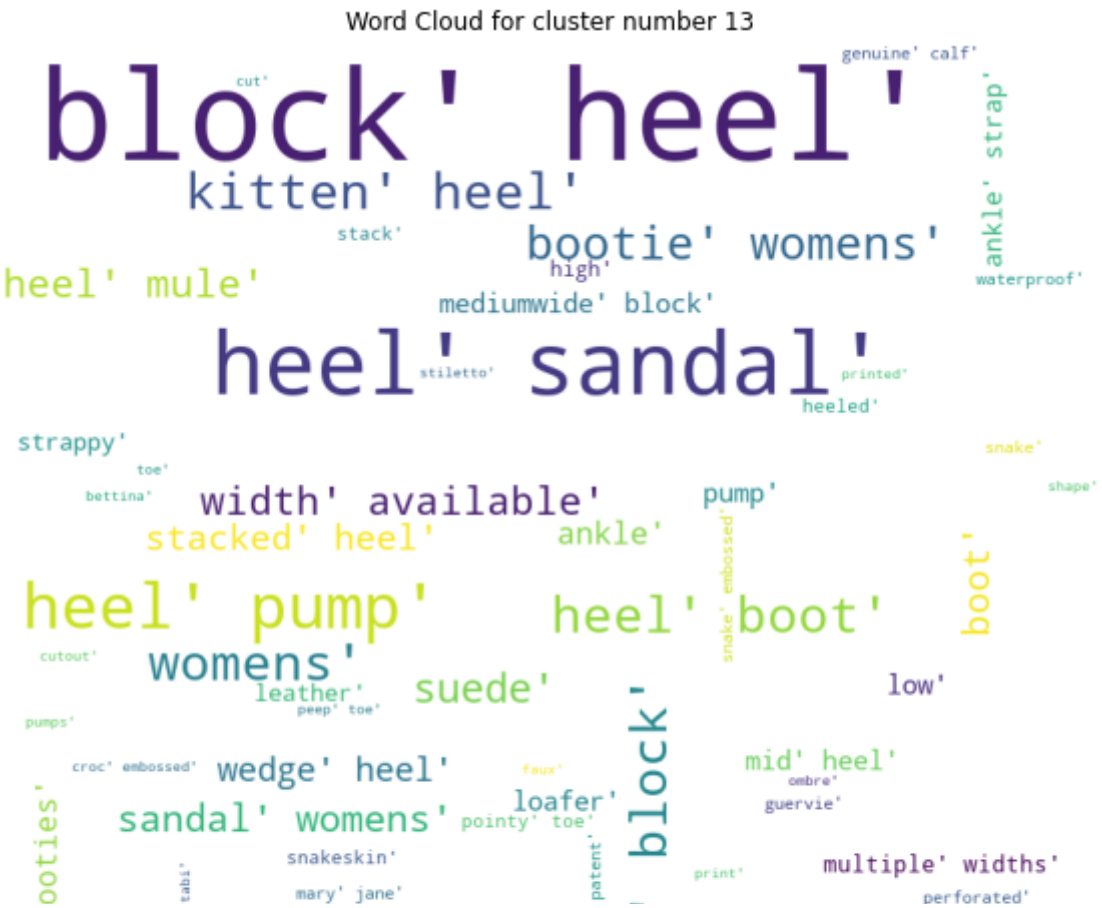


For cluster number 12 word cloud is:

Word Cloud for cluster number 12

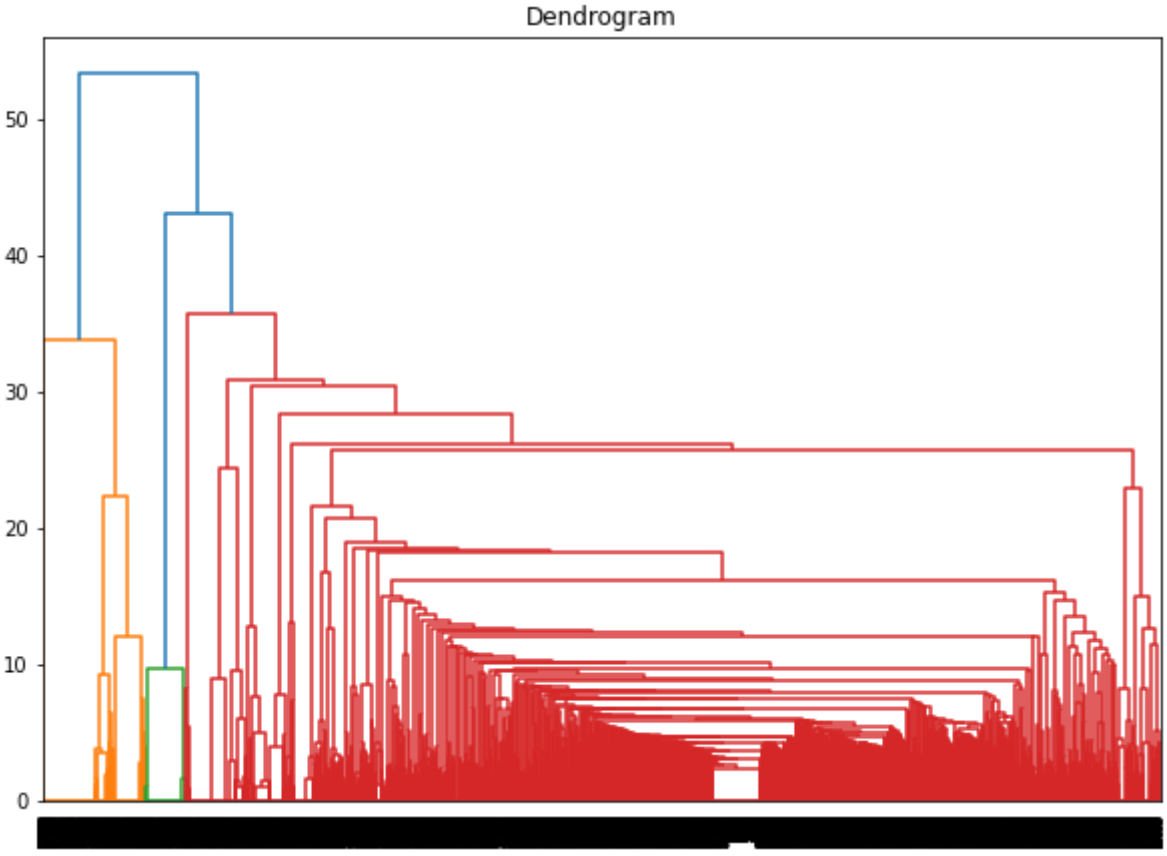


For cluster number 13 word cloud is:



Agglomerative Clustering

```
import scipy.cluster.hierarchy as cls

algo_title = 'Agglomerative Clustering'
plt.figure(figsize = (10, 7))
plt.title('Dendrogram')
dend = cls.dendrogram(cls.linkage(title_tfidf.toarray(), method = 'ward'))
```




```
from sklearn.cluster import AgglomerativeClustering

agg = AgglomerativeClustering(n_clusters = 4, linkage = 'ward')
agg.fit_predict(title_tfidf.toarray())
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
df = pd.DataFrame(data_df['title'], columns = ['title'])
df['cluster_name'] = agg.labels_
df = df.groupby(['cluster_name'])
```

```
from collections import Counter
```

```
for key, data in df:
    print('Number of data points in cluster {} are {}'.format(key + 1, data.shape[0]))
    print('Number of words in cluster {} are {}'.format(key + 1, len(' '.join(list(data['t
    counter = Counter(' '.join(list(data['title'])).split())
    most_occur = counter.most_common(10)
    print('Top 10 words for cluster {} are: {} \n\n'.format(key + 1, [most_occur[i][0] for
```

```
Number of data points in cluster 1 are 2783
Number of words in cluster 1 are 7381
Top 10 words for cluster 1 are: ['sandal', 'wedge', 'espadrille', 'bootie', 'platform


Number of data points in cluster 2 are 25523
Number of words in cluster 2 are 85634
Top 10 words for cluster 2 are: ['womens', 'sandal', 'boot', 'sneaker', 'flat', 'boot


Number of data points in cluster 3 are 1036
Number of words in cluster 3 are 2211
Top 10 words for cluster 3 are: ['bootie', 'womens', 'newbury', 'lottie', 'hollis',


Number of data points in cluster 4 are 715
Number of words in cluster 4 are 1565
Top 10 words for cluster 4 are: ['pump', 'wedge', 'womens', 'sidney', 'zala', 'flax',
```

```
for key, data in df:
    print('For cluster number {} word cloud is:'.format(key))

    wordcloud = WordCloud(width = 800, height = 800, background_color = 'white', stopwords
    plt.figure(figsize = (8, 8), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.title('Word Cloud for cluster number {}'.format(key))
    plt.tight_layout(pad = 0)

    plt.show()
```

```
For cluster number 0 word cloud is:
```

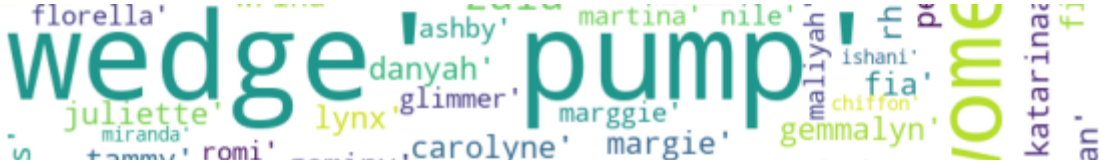

```
For cluster number 1 word cloud is:
```

For cluster number 2 word cloud is:

Word Cloud for cluster number 2

For cluster number 3 word cloud is:

Word Cloud for cluster number 3

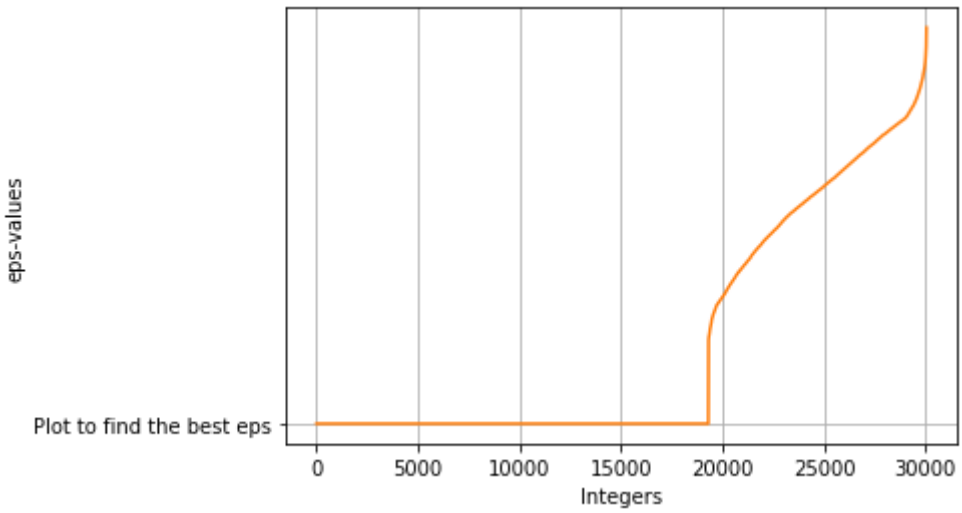


DBSCAN


```
from sklearn.neighbors import NearestNeighbors

neigh = NearestNeighbors(n_neighbors = 2)
nbrs = neigh.fit(title_tfidf)
distances, indices = nbrs.kneighbors(title_tfidf)
```


```
distances = np.sort(distances, axis = 0)
distances = distances[:, 1]
plt.plot('Plot to find the best eps')
plt.xlabel('Integers')
plt.ylabel('eps-values')
plt.grid()
plt.plot(distances)
```

```
[<matplotlib.lines.Line2D at 0x7fef58945d50>]
```




```
from sklearn.cluster import DBSCAN
db = DBSCAN(eps = 19000).fit(title_tfidf.toarray())


labels = db.labels_
number_of_cluster = len(set(labels)) - (1 if -1 in labels else 0) #Ignoring noise
print('Number of cluster is: {}'.format(number_of_cluster))
```

```
Number of cluster is: 1
```

Computing features based on co-occurence matrix

```
#Merging all the text based features into 1 column for generating co-occurence matrix
```

```
data_df['text'] = data_df[['title', 'description']].apply(lambda x: ' '.join(x), axis = 1)


#Verification
print('No of words in 1st row of title, description, text are: {}, {}, {}'.format(len(' '.
```

```
No of words in 1st row of title, description, text are: 3, 16, 19
```

```
vect = TfidfVectorizer(ngram_range = (1, 1), min_df = 10, use_idf = True)
vect.fit(data_df['text'])
```

```
TfidfVectorizer(min_df=10)
```

```
print('Some of the sample words in the corpus: ', vect.get_feature_names()[0:10])
```

```
Some of the sample words in the corpus:  ['aa', 'abbott', 'abigail', 'ability', 'able
```

```
tfidf_text = vect.transform(data_df['text'])
print('Shape of TFIDF vectorizer: {}'.format(tfidf_text.shape))
```

```
Shape of TFIDF vectorizer: (30057, 5731)
```

Taking top features from tfidf

```
idf_score = vect.idf_
feature_names = vect.get_feature_names()
final_score = sorted(list(zip(idf_score, feature_names)))


#selecting top 5000 words
top_5000_words = final_score[: -(5000 + 1) : -1]


#creating a list of top words
top_5000_lst = [top_5000_words[i][1] for i in range(len(top_5000_words))]


print('Top 50 words are: {}'.format(top_5000_lst[0:50]))
```

```
Top 50 words are: ['zola', 'zipup', 'ziplaceup', 'xtra', 'worthy', 'wonderful', 'wonc
```

Computing co-occurence Matrix

```
#checking for co-occurence matrix code on toy data
lst = ['abc', 'pqr', 'def']
lst2 = ['abc def ijk pqr', 'pqr klm opq', 'lmn pqr xyz abc def pqr abc']
window_size = 2
matrix = np.zeros((3, 3))

for row in lst2:
```

```
    for row in lst2:
        words_row = row.split()
        for i in range(len(lst)):
            for index, word in enumerate(words_row):
                if lst[i] == word:
                    for j in range(max(index - window_size, 0), min(index + window_size, len(w
                        if words_row[j] in lst:
                            matrix[i, lst.index(words_row[j])] += 1

matrix
```

```
array([[3., 3., 3.],
       [3., 4., 2.],
       [3., 2., 2.]])
```

This matrix looks good, let us implement it on our data

```
window_size = 5
matrix = np.zeros((5000, 5000))

for row in tqdm(data_df['text'].values):
    words_row = row.split()
    for i in range(len(top_5000_lst)):
        for index, word in enumerate(words_row):
            if top_5000_lst[i] == word:
                for j in range(max(index - window_size, 0), min(index + window_size, len(w
                    if words_row[j] in top_5000_lst:
                        matrix[i, top_5000_lst.index(words_row[j])] += 1
```

```
100%|██████████| 30057/30057 [18:54<00:00, 26.49it/s]
```

```
matrix
```

```
array([[ 10.,   0.,   0., ...,   0.,   0.,   0.],
       [  0.,  10.,   0., ...,   0.,   0.,   0.],
       [  0.,   0.,  10., ...,   0.,   0.,   0.],
       ...,
       [  0.,   0.,   0., ..., 196.,   1.,   0.],
       [  0.,   0.,   0., ...,   1., 194.,   0.],
       [  0.,   0.,   0., ...,   0.,   0., 192.]])
```

```
matrix.shape
```

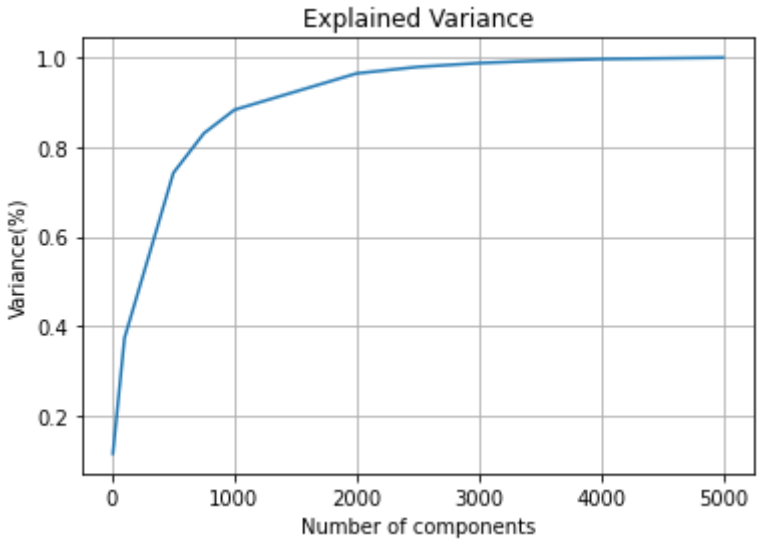
```
(5000, 5000)
```

Using Truncated SVD

```
from sklearn.decomposition import TruncatedSVD

num_com = [5, 100, 500, 750, 1000, 2000, 2500, 3000, 3500, 4000, 4999]
variance_sum = []
```

```
for i in num_com:
    svd = TruncatedSVD(n_components = i)
    svd.fit(matrix)
    variance_sum.append(svd.explained_variance_ratio_.sum())
    print('Number of components: {} and explained variance: {}'.format(i, svd.explained_va

plt.figure()
plt.plot(num_com, variance_sum)
plt.xlabel('Number of components')
plt.ylabel('Variance(%)')
plt.title('Explained Variance')
plt.grid()
plt.show()
```

```
Number of components: 5 and explained variance: 0.11432344661933186
Number of components: 100 and explained variance: 0.372774224778171
Number of components: 500 and explained variance: 0.7421807000785743
Number of components: 750 and explained variance: 0.8307957686964662
Number of components: 1000 and explained variance: 0.8833100835653146
Number of components: 2000 and explained variance: 0.9646731737363985
Number of components: 2500 and explained variance: 0.9791995123781498
Number of components: 3000 and explained variance: 0.9877575609086303
Number of components: 3500 and explained variance: 0.9931409719602768
Number of components: 4000 and explained variance: 0.9966881667706075
Number of components: 4999 and explained variance: 1.0000000000000007
```




```
svd = TruncatedSVD(n_components = 4000)
matrix_svd = svd.fit_transform(matrix)
```

```
matrix_svd.shape
```

```
(5000, 4000)
```

Vectorizing text features using co-occurence matrix

```
vec_title = []
for words in tqdm(data_df['title'].values):
    vector = np.zeros(4000)
```

```
    vector = np.zeros(4000)
    cnt_words = 0
    for word in words.split():
        if word in top_5000_lst:
            i = top_5000_lst.index(word)
            vector += matrix_svd[i]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    vec_title.append(vector)
```

```
100%|██████████| 30057/30057 [00:09<00:00, 3238.07it/s]
```

```
print(len(vec_title))
print(len(vec_title[0]))
```

```
30057
4000
```

```
vec_description = []
for words in tqdm(data_df['description'].values):
    vector = np.zeros(4000)
    cnt_words = 0
    for word in words.split():
        if word in top_5000_lst:
            i = top_5000_lst.index(word)
            vector += matrix_svd[i]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    vec_description.append(vector)
```

```
100%|██████████| 30057/30057 [01:47<00:00, 278.33it/s]
```

```
print(len(vec_description))
print(len(vec_description[0]))
```

```
30057
4000
```

```
vec_description = np.array(vec_description)
vec_title = np.array(vec_title)
```

```
vec_description.shape, vec_title.shape
```

```
((30057, 4000), (30057, 4000))
```

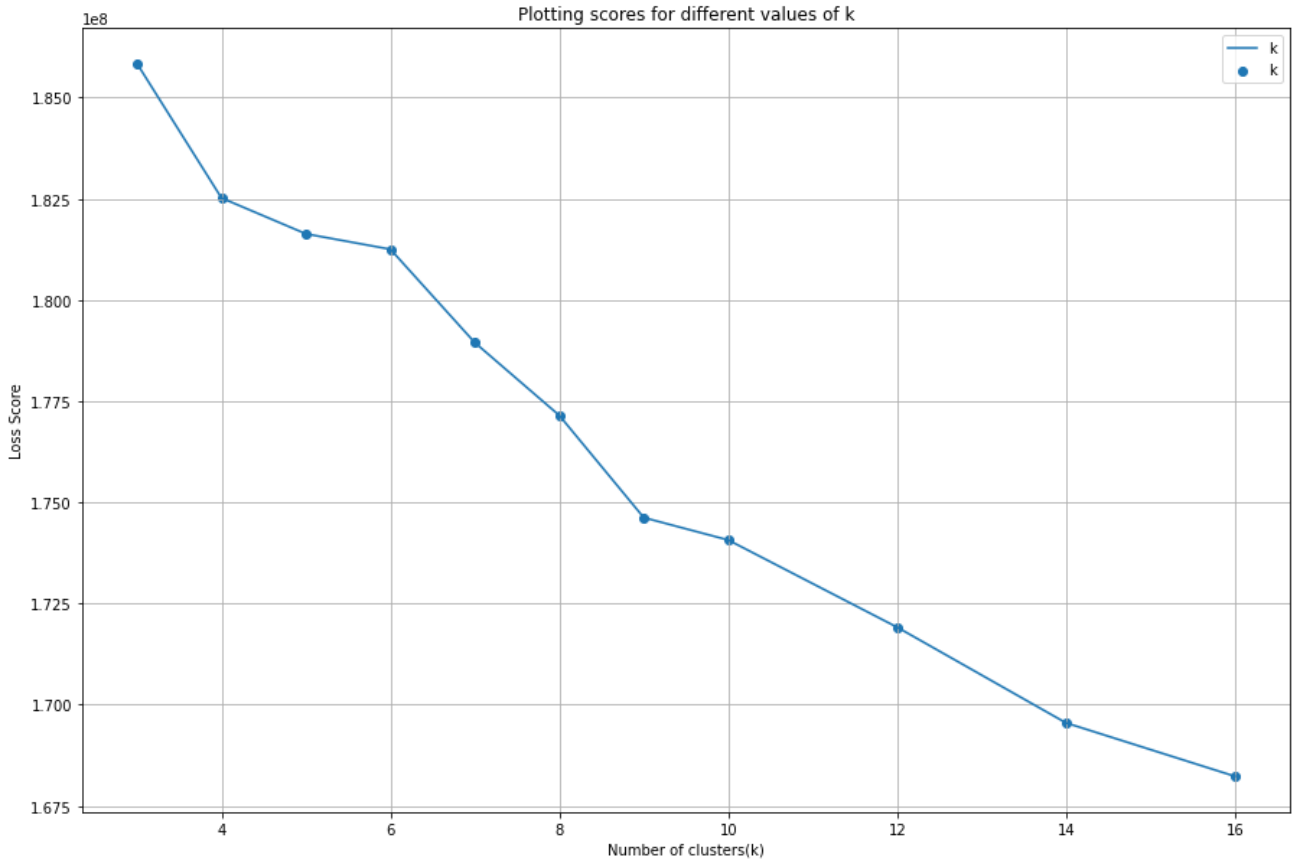
```
final_model_data = np.hstack((vec_description, vec_title))
final_model_data.shape
```

```
(30057, 8000)
```

```
from sklearn.cluster import KMeans
k = [3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16]
score = []
for i in tqdm(range(len(k))):
    kmeans = KMeans(n_clusters = k[i], random_state = 1).fit(final_model_data)
    score.append(kmeans.inertia_)
```

```
 100%|██████████| 11/11 [09:01<00:00, 49.25s/it]
```

```
plt.figure(figsize = (15, 10))

plt.plot(k, score, label = 'k')
plt.scatter(k, score, label = 'k')

plt.title('Plotting scores for different values of k')
plt.xlabel('Number of clusters(k)')
plt.ylabel('Loss Score')
plt.legend()
plt.grid()
plt.show()
```



```
kmeans = KMeans(n_clusters = 14, random_state = 1).fit(final_model_data)
kmeans.labels_.shape
```

```
(30057,)
```

```
df = pd.DataFrame(data_df['title'], columns = ['title'])
df['cluster_name'] = kmeans.labels_
df = df.groupby(['cluster_name'])
```

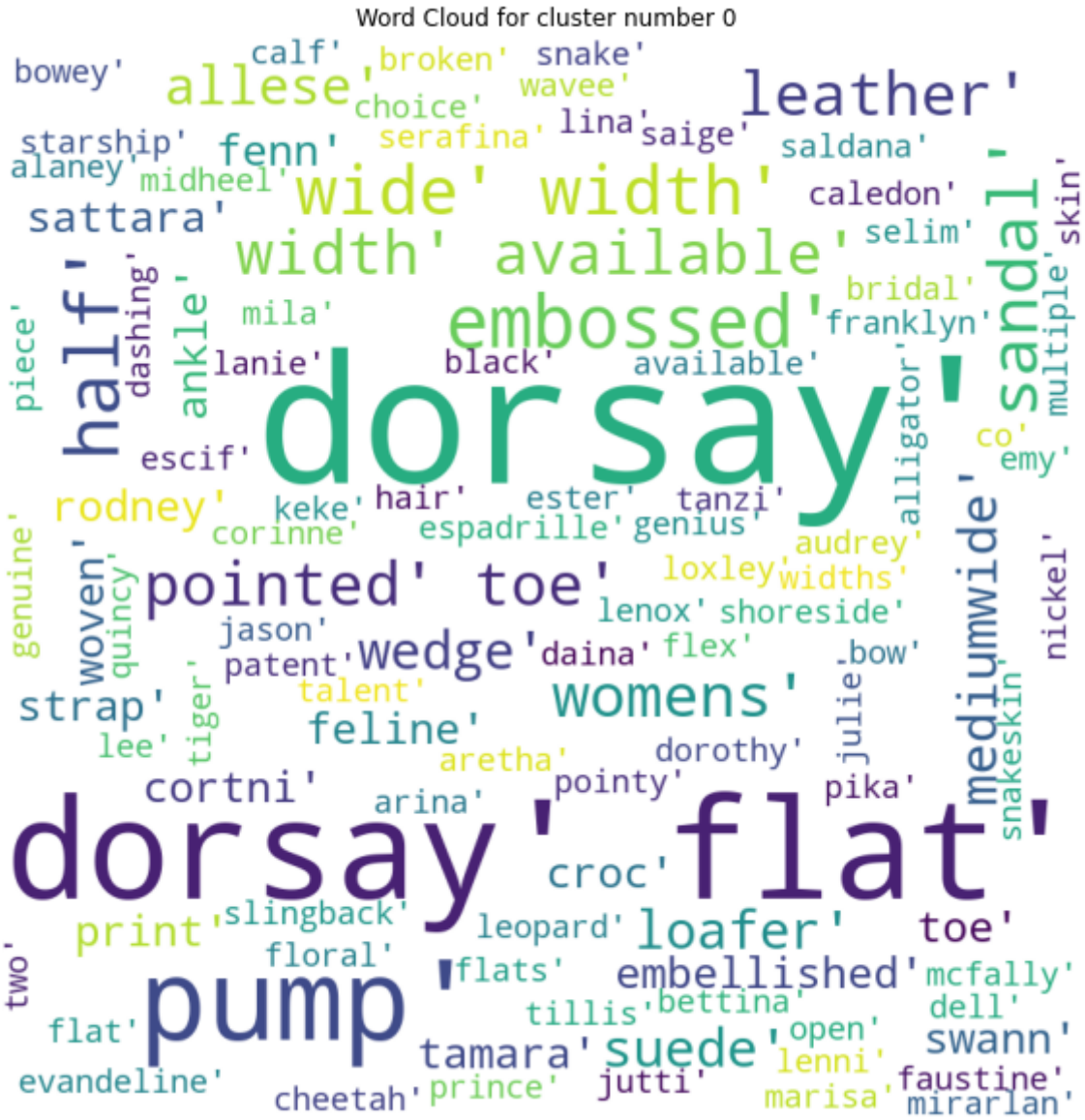
```
from collections import Counter

for key, data in df:
    print('Number of data points in cluster {} are {}'.format(key + 1, data.shape[0]))
    print('Number of words in cluster {} are {}'.format(key + 1, len(' '.join(list(data['t
    counter = Counter(' '.join(list(data['title'])).split())
    most_occur = counter.most_common(10)
    print('Top 10 words for cluster {} are: {} \n\n'.format(i, [most_occur[i][0] for i in
```

```
Number of data points in cluster 1 are 70
Number of words in cluster 1 are 290
Top 10 words for cluster 10 are: ['dorsay', 'flat', 'pump', 'toe', 'embossed', 'ha


Number of data points in cluster 2 are 162
Number of words in cluster 2 are 628
Top 10 words for cluster 10 are: ['boots', 'leather', 'booties', 'comfort', 'suede


Number of data points in cluster 3 are 173
Number of words in cluster 3 are 516
Top 10 words for cluster 10 are: ['bootie', 'sandal', 'boot', 'pump', 'sneaker', '


Number of data points in cluster 4 are 98
Number of words in cluster 4 are 373
Top 10 words for cluster 10 are: ['court', 'sneaker', 'womens', 'vision', 'grand',


Number of data points in cluster 5 are 190
Number of words in cluster 5 are 646
Top 10 words for cluster 10 are: ['dr', 'womens', 'shoe', 'running', 'ii', 'pascal


Number of data points in cluster 6 are 75
Number of words in cluster 6 are 304
Top 10 words for cluster 10 are: ['hiker', 'boot', 'waterproof', 'leather', 'boots


Number of data points in cluster 7 are 138
Number of words in cluster 7 are 307
Top 10 words for cluster 10 are: ['lartiste', 'sandal', 'jcrew', 'pump', 'wedge',


Number of data points in cluster 8 are 107
Number of words in cluster 8 are 311
Top 10 words for cluster 10 are: ['gtx', 'mid', 'terrex', 'arcteryx', 'breeze', 't
```

```
Number of data points in cluster 9 are 159
Number of words in cluster 9 are 951
Top 10 words for cluster 10 are: ['chuck', 'taylor', 'star', 'sneaker', 'womens',


Number of data points in cluster 10 are 226
Number of words in cluster 10 are 975
Top 10 words for cluster 10 are: ['boot', 'womens', 'tall', 'knee', 'wide', 'high'


Number of data points in cluster 11 are 110
Number of words in cluster 11 are 375
Top 10 words for cluster 10 are: ['sandal', 'leather', 'rose', 'bootie', 'tea', 'p


Number of data points in cluster 12 are 40
Number of words in cluster 12 are 163
Top 10 words for cluster 10 are: ['steel', 'toe', 'dr', 'work', 'waterproof', 'boo
```

```
for key, data in df:
    print('For cluster number {} word cloud is:'.format(key))

    wordcloud = WordCloud(width = 800, height = 800, background_color = 'white', stopwords
    plt.figure(figsize = (8, 8), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.title('Word Cloud for cluster number {}'.format(key))
    plt.tight_layout(pad = 0)

    plt.show()
```

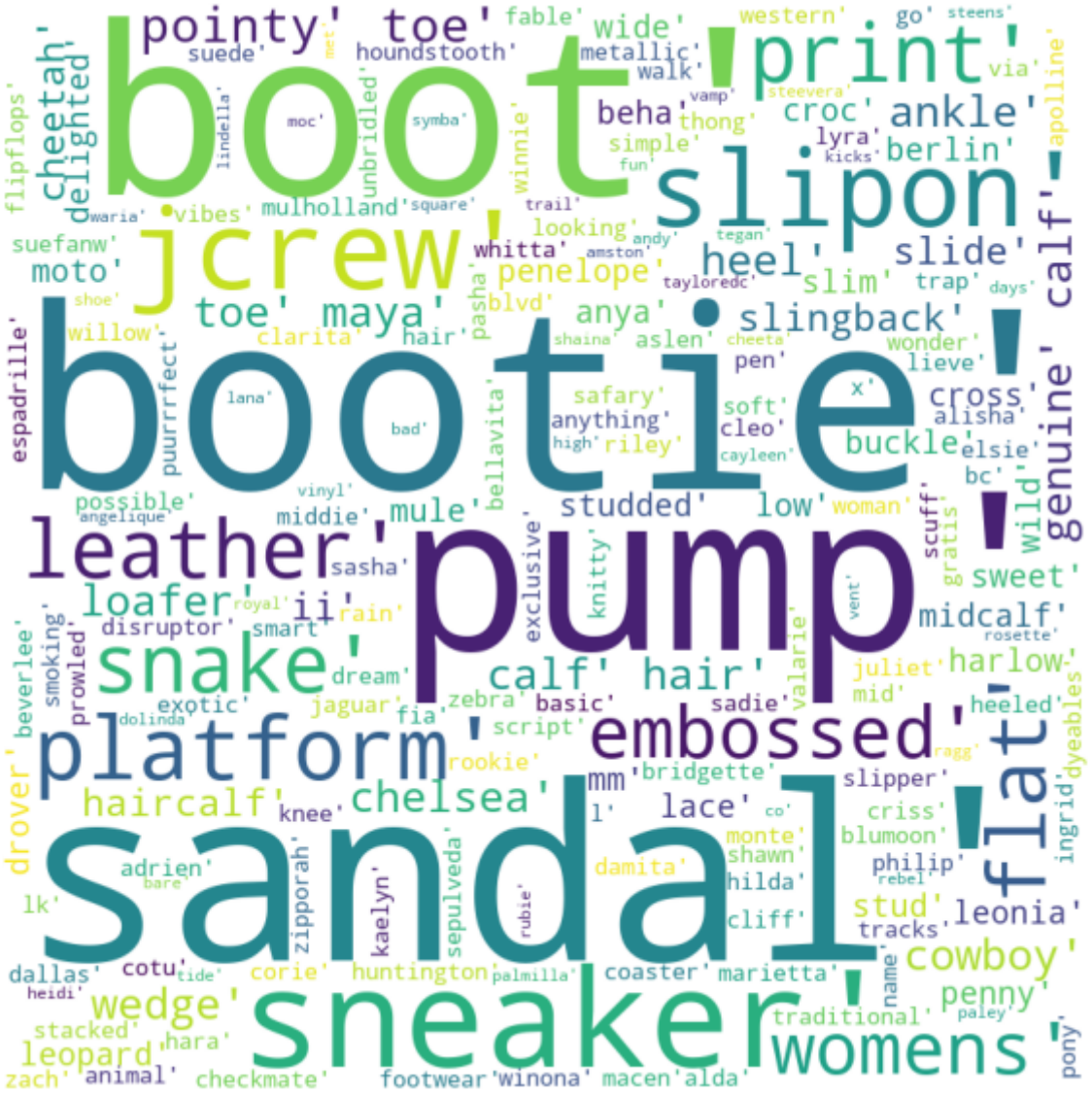
For cluster number 0 word cloud is:



For cluster number 1 word cloud is:

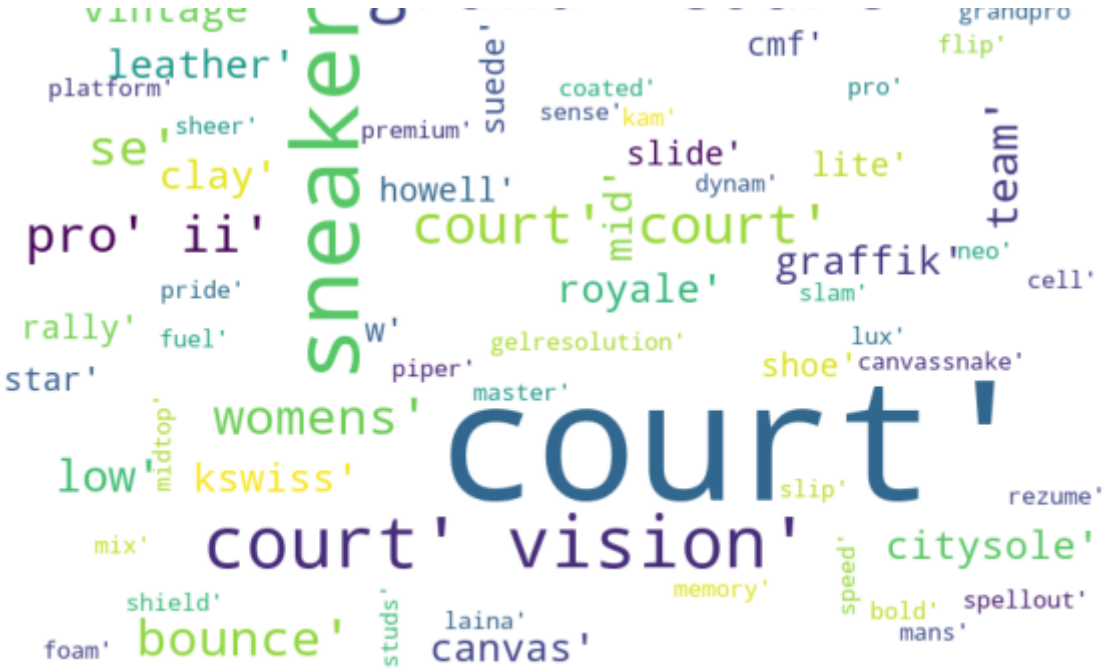
For cluster number 2 word cloud is:

Word Cloud for cluster number 2



For cluster number 3 word cloud is:

Word Cloud for cluster number 3



For cluster number 4 word cloud is:

Word Cloud for cluster number 4

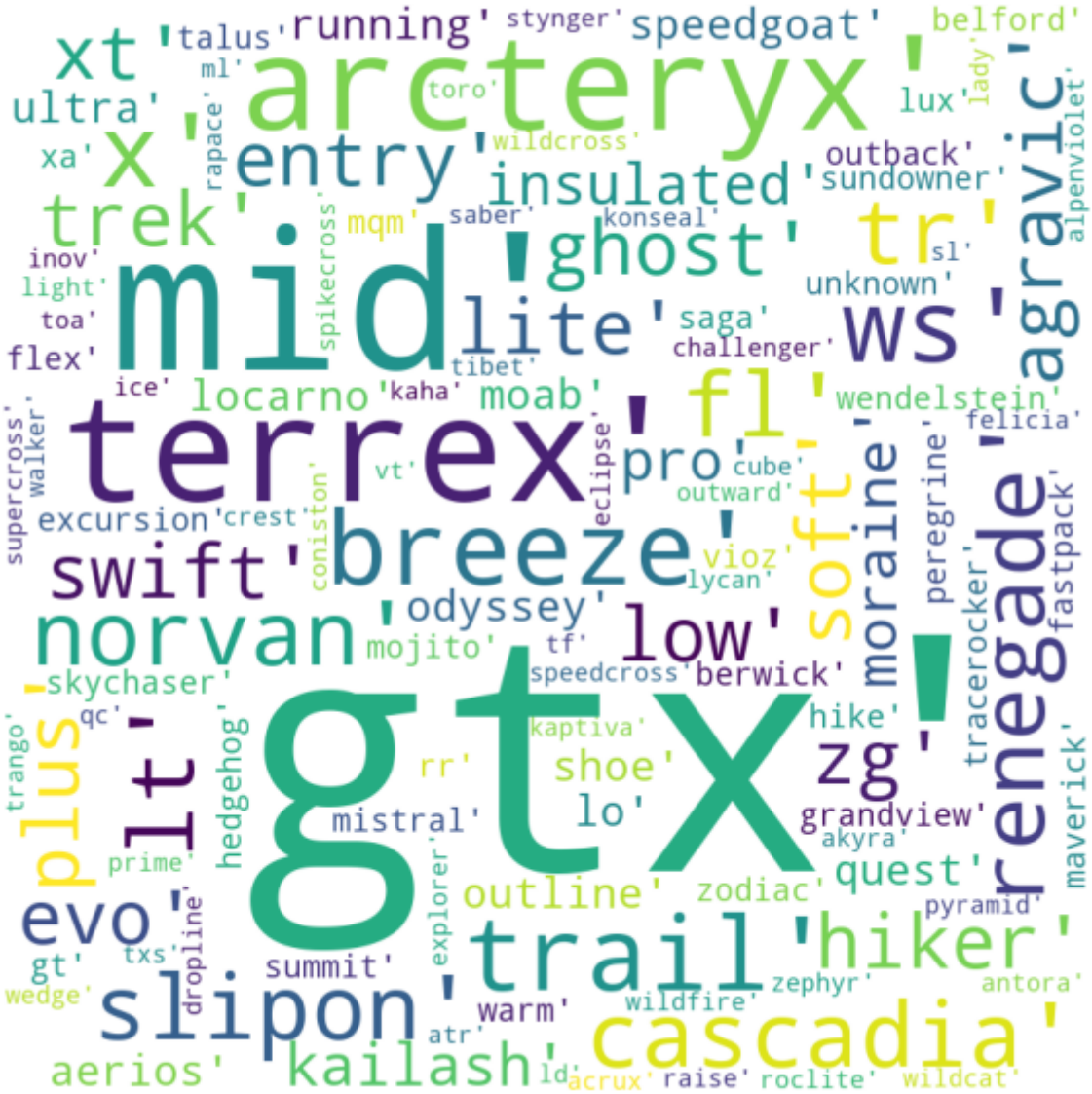
For cluster number 5 word cloud is:

Word Cloud for cluster number 5

For cluster number 6 word cloud is:

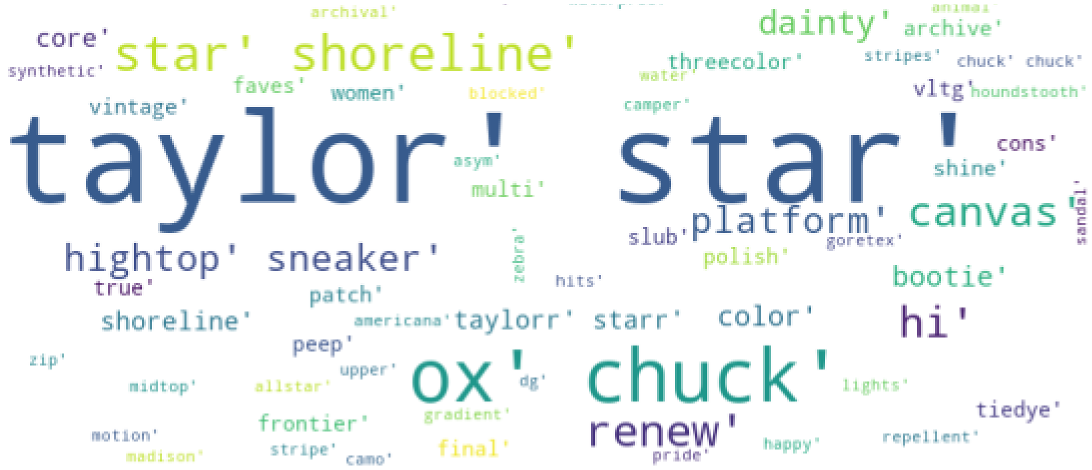
For cluster number 7 word cloud is:

Word Cloud for cluster number 7



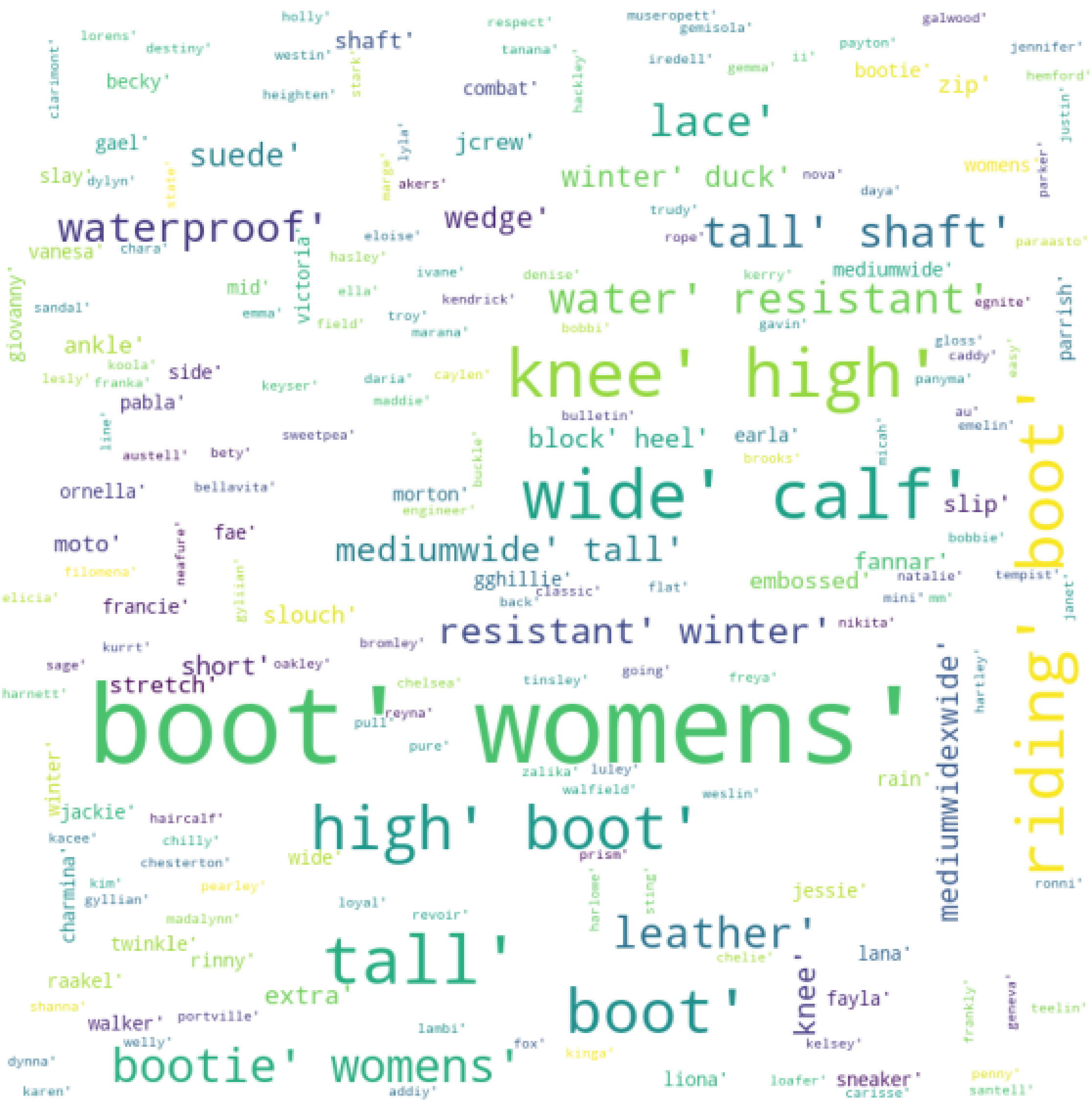
For cluster number 8 word cloud is:

Word Cloud for cluster number 8



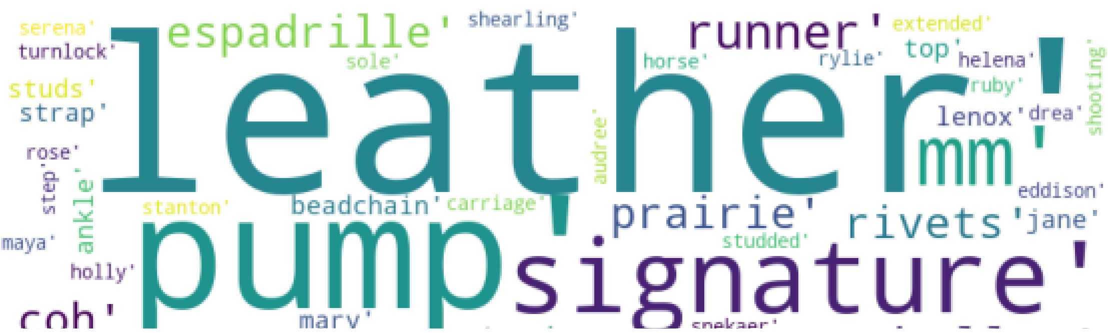
For cluster number 9 word cloud is:

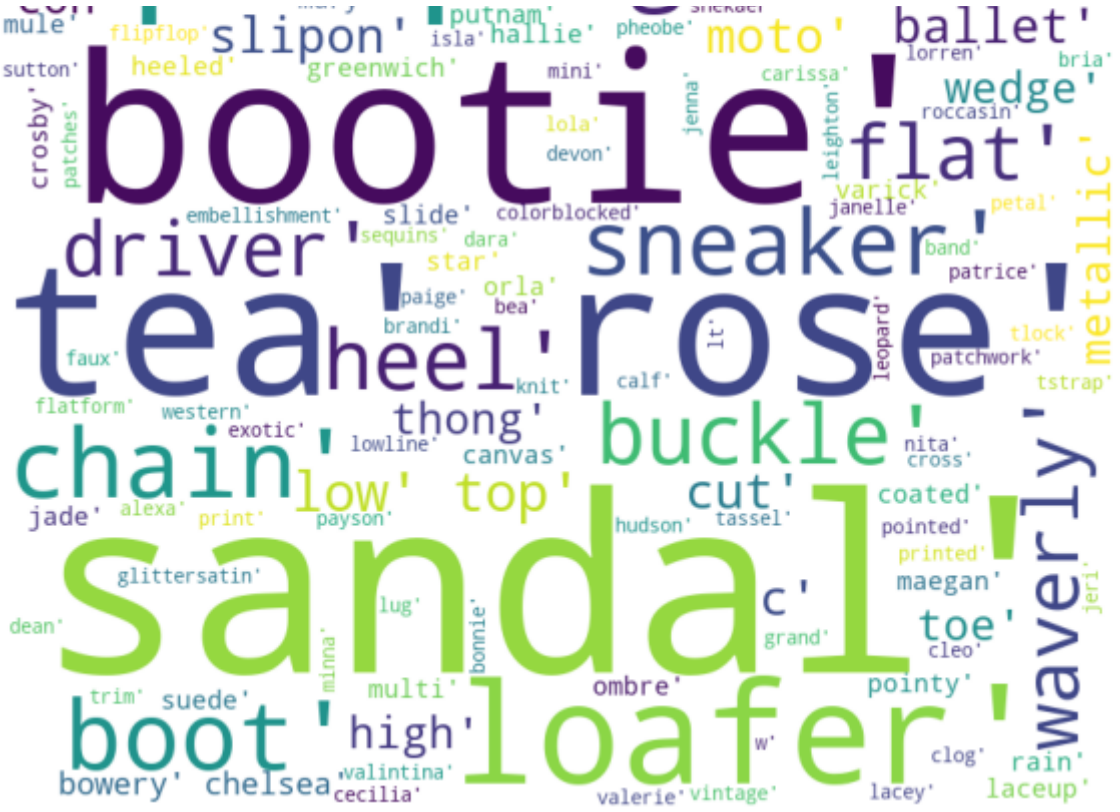
Word Cloud for cluster number 9



For cluster number 10 word cloud is:

Word Cloud for cluster number 10





For cluster number 11 word cloud is:

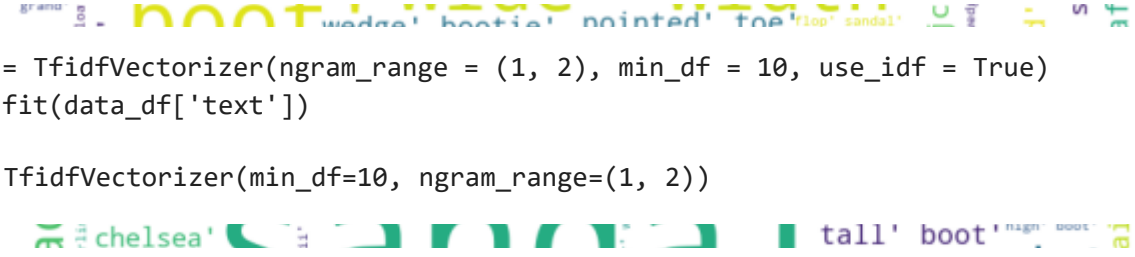
Word Cloud for cluster number 11

For cluster number 12 word cloud is:

Word Cloud for cluster number 12

For cluster number 13 word cloud is:

Using tfidf to vectorize text




```
vect = TfidfVectorizer(ngram_range = (1, 2), min_df = 10, use_idf = True)
vect.fit(data_df['text'])
```

```
TfidfVectorizer(min_df=10, ngram_range=(1, 2))
```

```
print('Some of the sample words in the corpus: ', vect.get_feature_names()[0:10])
```

```
Some of the sample words in the corpus:  ['aa', 'aa please', 'abbott', 'abigail', 'ab
```

```
tfidf_text = vect.transform(data_df['text'])
print('Shape of TFIDF vectorizer: {}'.format(tfidf_text.shape))
```

```
Shape of TFIDF vectorizer: (30057, 18565)
```

Taking top features from tfidf

```
idf_score = vect.idf_
feature_names = vect.get_feature_names()
final_score = sorted(list(zip(idf_score, feature_names)))
```

```
print(final_score[0])
```

```
(1.5237056973582948, 'size')
```

```
len(final_score)
```

```
18565
```

```
#selecting top 15000 words
top_15000_words = final_score[: -(15000 + 1) : -1]
```

```
#creating a list of top words
top_15000_lst = [top_15000_words[i][1] for i in range(len(top_15000_words))]
```

```
print('Top 50 words are: {}'.format(top_15000_lst[0:50]))
```

```
Top 50 words are: ['zola', 'zipup', 'ziplaceup closure', 'ziplaceup', 'youre move',
```

Computing co-occurence Matrix

```
window_size = 5
matrix = np.zeros((15000, 15000))

for row in tqdm(data_df['text'].values):
    words_row = row.split()
    for i in range(len(top_15000_lst)):
        for index, word in enumerate(words_row):
            if top_15000_lst[i] == word:
                for j in range(max(index - window_size, 0), min(index + window_size, len(w
                    if words_row[j] in top_15000_lst:
                        matrix[i, top_15000_lst.index(words_row[j])] += 1
```

```
100%|██████████| 30057/30057 [55:55<00:00,  8.96it/s]
```

```
matrix
```

```
array([[10.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0., 10.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.]])
```

```
matrix.shape
```

```
(15000, 15000)
```

Using Truncated SVD

```
from sklearn.decomposition import TruncatedSVD

num_com = [5, 100, 500, 750, 1000, 2000, 4000, 5000, 10000, 14000, 14999]
variance_sum = []

for i in num_com:
    svd = TruncatedSVD(n_components = i)
    svd.fit(matrix)
    variance_sum.append(svd.explained_variance_ratio_.sum())
    print('Number of components: {} and explained variance: {}'.format(i, svd.explained_va

plt.figure()
plt.plot(num_com, variance_sum)
plt.xlabel('Number of components')
plt.ylabel('Variance(%)')
plt.title('Explained Variance')
plt.grid()
plt.show()
```

```
Number of components: 5 and explained variance: 0.05431833746062892
Number of components: 100 and explained variance: 0.2509190181809771
Number of components: 500 and explained variance: 0.5600237451666363
Number of components: 750 and explained variance: 0.6737921844740556
Number of components: 1000 and explained variance: 0.7536921035121518
Number of components: 2000 and explained variance: 0.9159174827253727
Number of components: 4000 and explained variance: 0.9999999999999869
Number of components: 5000 and explained variance: 0.9999999999999865
Number of components: 10000 and explained variance: 0.9999999999999867
```

```
svd = TruncatedSVD(n_components = 4000)
matrix_svd = svd.fit_transform(matrix)
```

Vectorizing text features using co-occurence matrix

```
vec_title = []
for words in tqdm(data_df['title'].values):
    vector = np.zeros(4000)
    cnt_words = 0
    for word in words.split():
        if word in top_15000_lst:
            i = top_15000_lst.index(word)
            vector += matrix_svd[i]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    vec_title.append(vector)
```

```
100%|██████████| 30057/30057 [00:40<00:00, 734.13it/s]
```

```
print(len(vec_title))
print(len(vec_title[0]))
```

```
30057
4000
```

```
vec_description = []
for words in tqdm(data_df['description'].values):
    vector = np.zeros(4000)
    cnt_words = 0
    for word in words.split():
        if word in top_15000_lst:
            i = top_15000_lst.index(word)
            vector += matrix_svd[i]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    vec_description.append(vector)
```

```
100%|██████████| 30057/30057 [07:56<00:00, 63.09it/s]
```

```
print(len(vec_description))
print(len(vec_description[0]))
```

```
30057
4000
```

```
vec_description = np.array(vec_description)
vec_title = np.array(vec_title)
```

```
vec_description.shape, vec_title.shape
```

```
((30057, 4000), (30057, 4000))
```

```
final_model_data = np.hstack((vec_description, vec_title))
```

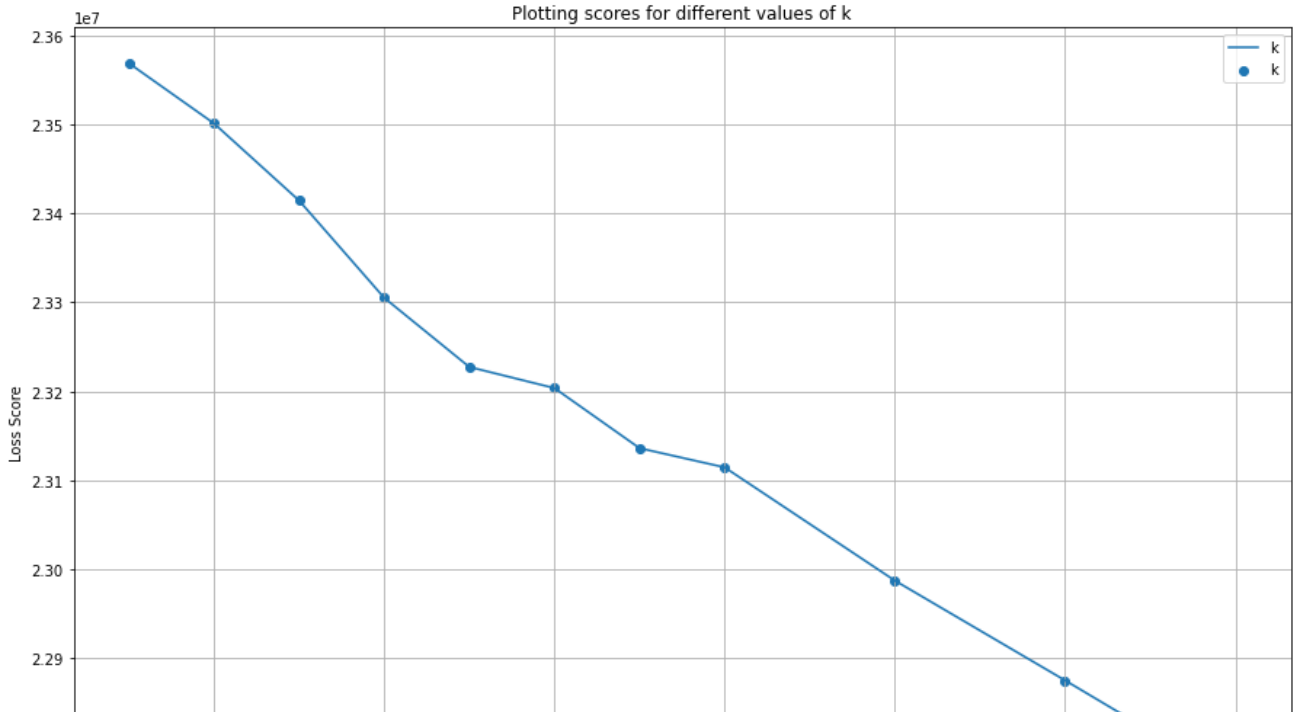
```
final_model_data.shape
```

```
(30057, 8000)
```

```
from sklearn.cluster import KMeans
k = [3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16]
score = []
for i in tqdm(range(len(k))):
    kmeans = KMeans(n_clusters = k[i], random_state = 1).fit(final_model_data)
    score.append(kmeans.inertia_)
```

```
100%|██████████| 11/11 [08:57<00:00, 48.83s/it]
```

```
plt.figure(figsize = (15, 10))

plt.plot(k, score, label = 'k')
plt.scatter(k, score, label = 'k')

plt.title('Plotting scores for different values of k')
plt.xlabel('Number of clusters(k)')
plt.ylabel('Loss Score')
plt.legend()
plt.grid()
plt.show()
```




```
kmeans = KMeans(n_clusters = 7, random_state = 1).fit(final_model_data)
kmeans.labels_.shape
```

```
(30057,)
```

```
df = pd.DataFrame(data_df['title'], columns = ['title'])
df['cluster_name'] = kmeans.labels_
df = df.groupby(['cluster_name'])
```

```
from collections import Counter

for key, data in df:
    print('Number of data points in cluster {} are {}'.format(key + 1, data.shape[0]))
    print('Number of words in cluster {} are {}'.format(key + 1, len(' '.join(list(data['t
    counter = Counter(' '.join(list(data['title'])).split())
    most_occur = counter.most_common(10)
    print('Top 10 words for cluster {} are: {} \n\n'.format(i, [most_occur[i][0] for i in
```

```
Number of data points in cluster 1 are 29746
Number of words in cluster 1 are 95677
Top 10 words for cluster 10 are: ['sandal', 'womens', 'boot', 'sneaker', 'wedge', 'bc


Number of data points in cluster 2 are 36
Number of words in cluster 2 are 137
Top 10 words for cluster 10 are: ['club', 'c', 'sneaker', 'womens', 'memt', 'reedux',


Number of data points in cluster 3 are 42
Number of words in cluster 3 are 132
Top 10 words for cluster 10 are: ['sandal', 'boot', 'sp', 'platform', 'wedge', 'sneak


Number of data points in cluster 4 are 103
Number of words in cluster 4 are 454
Top 10 words for cluster 10 are: ['made', 'italy', 'suede', 'leather', 'flats', 'boot
```

```
Number of data points in cluster 5 are 47
Number of words in cluster 5 are 173
Top 10 words for cluster 10 are: ['wp', 'mid', 'ii', 'boot', 'sugarpine', 'toe', 'hit


Number of data points in cluster 6 are 46
Number of words in cluster 6 are 99
Top 10 words for cluster 10 are: ['circuit', 'short', 'snip', 'bonnie', 'riley', 'dee


Number of data points in cluster 7 are 37
Number of words in cluster 7 are 119
Top 10 words for cluster 10 are: ['joy', 'go', 'walk', 'yoga', 'womens', 'onthego',
```

```
for key, data in df:
    print('For cluster number {} word cloud is:'.format(key))

    wordcloud = WordCloud(width = 800, height = 800, background_color = 'white', stopwords
    plt.figure(figsize = (8, 8), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.title('Word Cloud for cluster number {}'.format(key))
    plt.tight_layout(pad = 0)

    plt.show()
```

For cluster number 0 word cloud is:

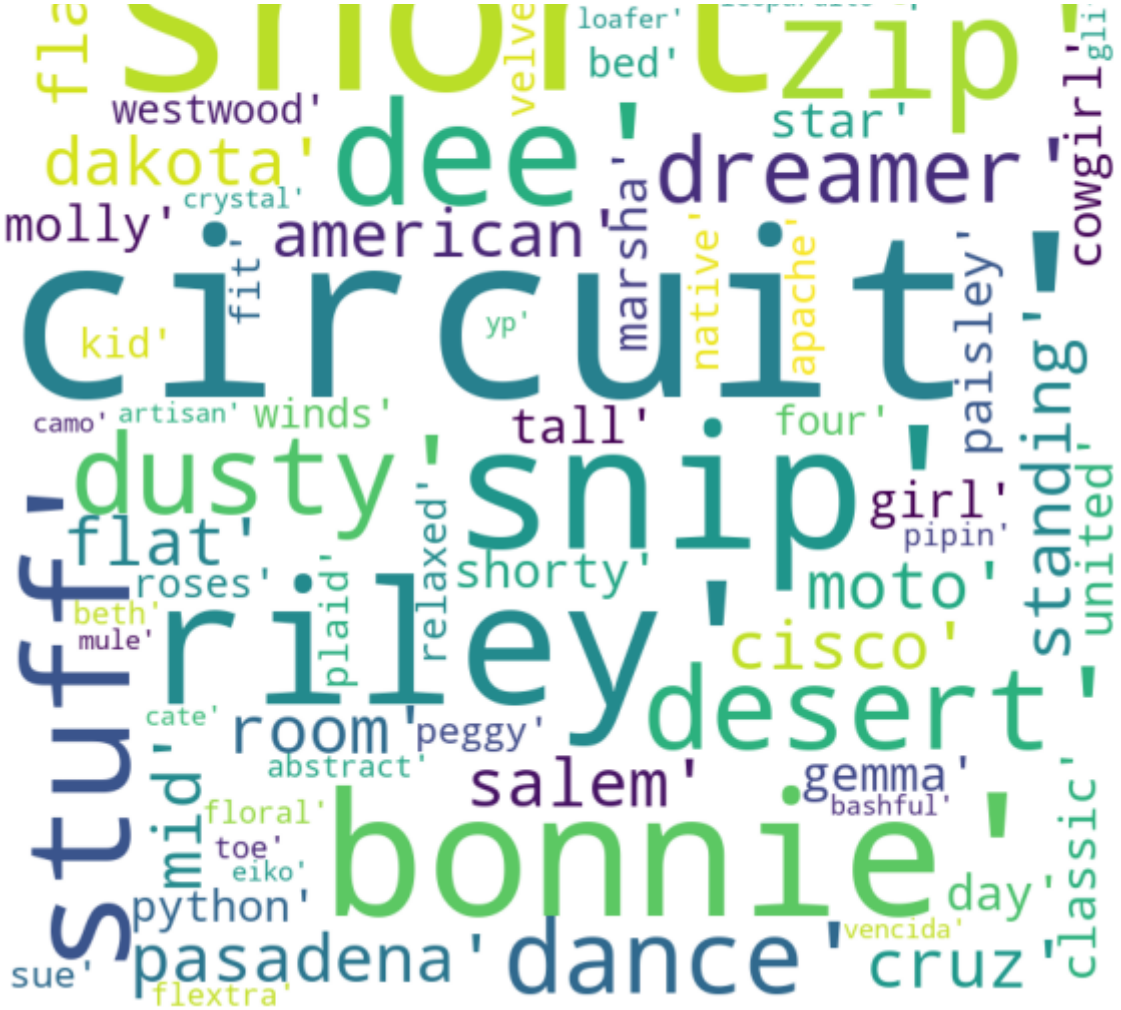
For cluster number 1 word cloud is:

For cluster number 2 word cloud is:

For cluster number 3 word cloud is:

For cluster number 4 word cloud is:

For cluster number 5 word cloud is:



For cluster number 6 word cloud is:

Word Cloud for cluster number 6


```
vect = TfidfVectorizer(ngram_range = (1, 1), min_df = 10, use_idf = True)
vect.fit(data_df['title'])
```

```
TfidfVectorizer(min_df=10)
```

```
tfidf_text = vect.transform(data_df['title'])
print('Shape of TFIDF vectorizer: {}'.format(tfidf_text.shape))
```

```
Shape of TFIDF vectorizer: (30057, 1010)
```

```
idf_score = vect.idf_
feature_names = vect.get_feature_names()
final_score = sorted(list(zip(idf_score, feature_names)))
```

```
len(final_score)
```

```
1010
```

```
final_score[0:50]
```

```
[(2.6914954237631328, 'sandal'),
 (3.238416758139269, 'boot'),
 (3.3439506290892, 'womens'),
 (3.494870288335012, 'bootie'),
 (3.500937041017249, 'sneaker'),
 (3.543196850307132, 'wedge'),
 (3.9463371132383975, 'pump'),
 (4.168056726332419, 'flat'),
 (4.296069776218495, 'slipon'),
 (4.368727421794571, 'leather'),
 (4.410153463448867, 'loafer'),
 (4.51071405919184, 'wide'),
 (4.583452402643184, 'platform'),
 (4.600361018041611, 'women'),
 (4.6815208740565915, 'espadrille'),
 (4.775642856480381, 'heel'),
 (4.8416338106982675, 'toe'),
 (4.8494159511403225, 'waterproof'),
 (4.87794403475486, 'mediumwide'),
 (4.908966930766854, 'mule'),
 (4.908966930766854, 'shoe'),
 (4.912289192958832, 'slide'),
 (4.935859307665943, 'width'),
 (4.982947343764845, 'available'),
 (5.008265151749135, 'suede'),
 (5.104308200769112, 'ankle'),
 (5.158151432789936, 'calf'),
 (5.186200736599835, 'slip'),
 (5.210565175473976, 'ii'),
 (5.219574245416341, 'dr'),
 (5.329469916239559, 'block'),
 (5.381294984104145, 'strap'),
 (5.400087483453513, 'tall'),
 (5.4220061691611585, 'mid'),
 (5.558311488668407, 'clog'),
 (5.610440554103353, 'lace'),
 (5.637560860322547, 'high'),
 (5.647923647358094, 'classic'),
 (5.734935024347723, 'chelsea'),
 (5.769620582335613, 'slipper'),
 (5.781455039982617, 'ride'),
 (5.817822684153491, 'ballet'),
 (5.859845673928339, 'jcrew'),
 (5.959025994017973, 'running'),
 (5.983007958704459, 'slingback'),
 (5.992764133649823, 'emboss'),
 (6.017579302769548, 'star'),
 (6.037884568930293, 'low'),
 (6.063860055333554, 'original'),
 (6.069137112434397, 'print')]
```

BOW Implementation

```
from collections import Counter
from scipy.sparse import csr_matrix
```

```
from scipy.sparse import csr_matrix
from sklearn.preprocessing import normalize


def bow_fit(data):
    unique_words = set()
    for row in data.values:
        for word in row.split():
            if len(word) < 2:
                continue
            unique_words.add(word)
    unique_words = sorted(list(unique_words))
    vocab = {j:i for i,j in enumerate(unique_words)}
    return vocab


def bow_transform(data, vocab):
    rows = []
    columns = []
    values = []
    for row_index, row in enumerate(data):
        word_freq = dict(Counter(row.split()))
        for word, freq in word_freq.items():
            if len(word) < 2:
                continue
            column_index = vocab.get(word, -1)
            if column_index != -1:
                rows.append(row_index)
                columns.append(column_index)
                values.append(freq)
    return csr_matrix((values, (rows, columns)), shape = (len(data), len(vocab)))


vocab = bow_fit(data_df['title'])
len(vocab)
```

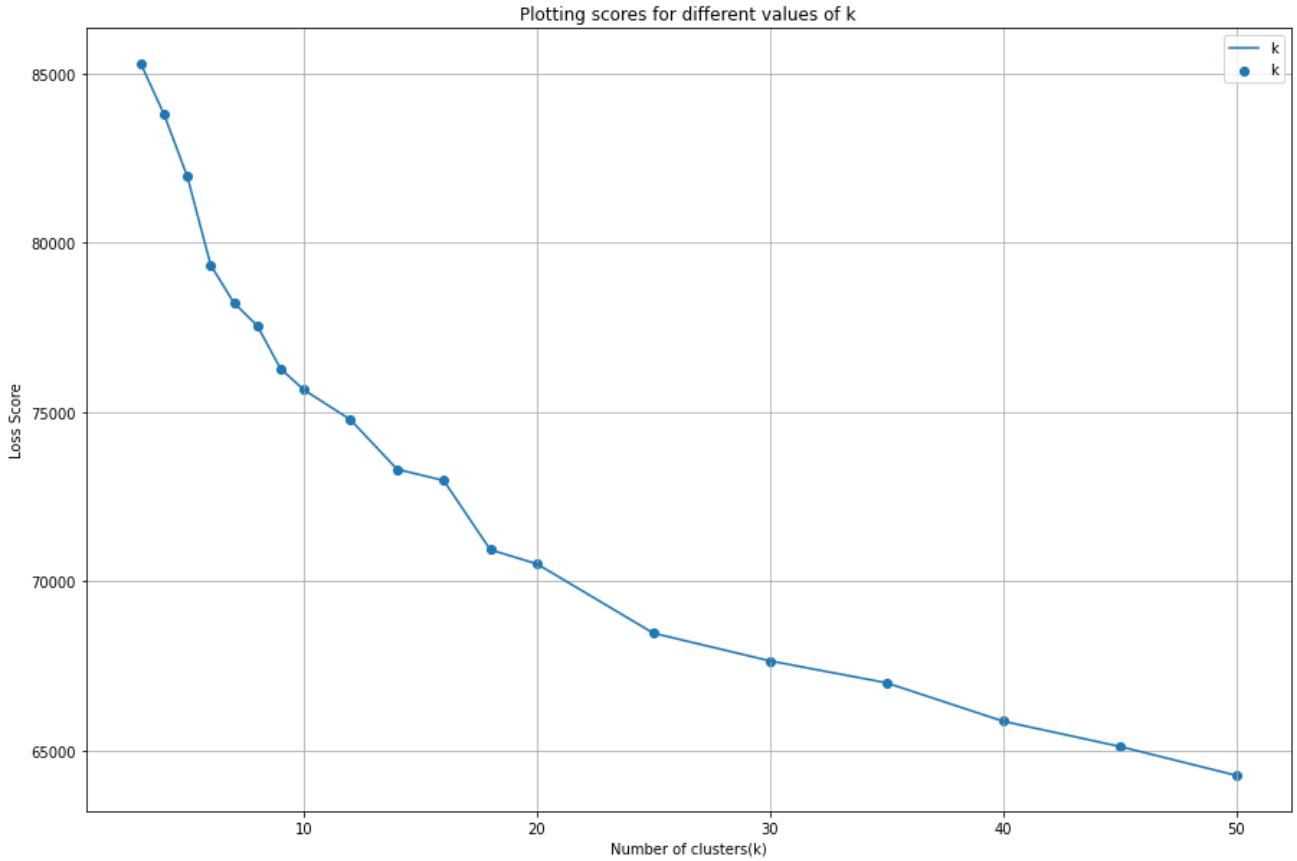
```
13888
```

```
title_bow = bow_transform(data_df['title'], vocab)


from sklearn.cluster import KMeans
k = [3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20, 25, 30, 35, 40, 45, 50]
score = []
for i in tqdm(range(len(k))):
    kmeans = KMeans(n_clusters = k[i], random_state = 1).fit(title_bow)
    score.append(kmeans.inertia_)
```

```
100%|██████████| 19/19 [02:19<00:00,  7.33s/it]
```

```
plt.figure(figsize = (15, 10))

plt.plot(k, score, label = 'k')
plt.scatter(k, score, label = 'k')

plt.title('Plotting scores for different values of k')
```

```
plt.xlabel('Number of clusters(k)')
plt.ylabel('Loss Score')
plt.legend()
plt.grid()
plt.show()
```




```
kmeans = KMeans(n_clusters = 12, random_state = 1).fit(title_bow)
kmeans.labels_.shape
```

```
(30294,)
```

```
df = pd.DataFrame(data_df[['title', 'description']], columns = ['title', 'description'])
df['cluster_name'] = kmeans.labels_
df = df.groupby(['cluster_name'])
```

```
for key, data in df:
    print('Number of data points in cluster {} are {}'.format(key + 1, data.shape[0]))
    print('Number of words in cluster {} are {}'.format(key + 1, len(' '.join(list(data['t
    counter = Counter(' '.join(list(data['title'])).split())
    most_occur = counter.most_common(10)
```

```
print('Top 10 words for cluster {} are: {} \n\n'.format(key + 1, [most_occur[1][0] for
```

```
Number of data points in cluster 1 are 4584
Number of words in cluster 1 are 14057
Top 10 words for cluster 1 are: ['sandal', 'wedge', 'espadrille', 'strap', 'heel',


Number of data points in cluster 2 are 2605
Number of words in cluster 2 are 9574
Top 10 words for cluster 2 are: ['boot', 'waterproof', 'lace', 'ankle', 'tall', 'c


Number of data points in cluster 3 are 2107
Number of words in cluster 3 are 7078
Top 10 words for cluster 3 are: ['sneaker', 'slip', 'platform', 'wedge', 'leather'


Number of data points in cluster 4 are 572
Number of words in cluster 4 are 2026
Top 10 words for cluster 4 are: ['platform', 'sandal', 'espadrille', 'wedge', 'sli


Number of data points in cluster 5 are 13660
Number of words in cluster 5 are 39457
Top 10 words for cluster 5 are: ['flat', 'slip', 'mule', 'shoe', 'leather', 'dr',


Number of data points in cluster 6 are 171
Number of words in cluster 6 are 859
Top 10 words for cluster 6 are: ['top', 'sneaker', 'high', 'low', 'star', 'chuck',


Number of data points in cluster 7 are 998
Number of words in cluster 7 are 5220
Top 10 words for cluster 7 are: ['wide', 'medium', 'sandal', 'slip', 'x', 'narrow'


Number of data points in cluster 8 are 925
Number of words in cluster 8 are 2846
Top 10 words for cluster 8 are: ['loafer', 'available', 'leather', 'wide', 'width'


Number of data points in cluster 9 are 1453
Number of words in cluster 9 are 4651
Top 10 words for cluster 9 are: ['pump', 'wedge', 'toe', 'available', 'heel', 'poi


Number of data points in cluster 10 are 414
Number of words in cluster 10 are 2071
Top 10 words for cluster 10 are: ['wide', 'calf', 'boot', 'riding', 'extra', 'tall


Number of data points in cluster 11 are 2237
Number of words in cluster 11 are 6436
Top 10 words for cluster 11 are: ['bootie', 'wedge', 'heel', 'ankle', 'toe', 'leat


Number of data points in cluster 12 are 568
Number of words in cluster 12 are 1931
Top 10 words for cluster 12 are: ['slide', 'sandal', 'j', 'crew', 'leather', 'espa
```