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

```
Mounted at /content/gdrive
```

```
import pandas as pd
import numpy as np
import re

import matplotlib.pyplot as plt
import seaborn as sns

import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')

from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk import WordPunctTokenizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from textblob import TextBlob
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
sto_word = list(set(stopwords.words('english')))
from nltk.stem import WordNetLemmatizer  # lemmatizer

from wordcloud import WordCloud
from nltk.tokenize import word_tokenize
from nltk.util import ngrams

pd.set_option('mode.chained_assignment', None)

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.cluster import MiniBatchKMeans
import plotly.express as px
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
```

```
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

1.EDA

1.1 WHY COMBINE DATA AND THEN SPLIT

[ ] ↳ 14 cells hidden

1.2 EDA AFTER COMBINING DATA

```
train = pd.read_csv('/content/gdrive/MyDrive/cs1/data/train.csv')
dev = pd.read_csv('/content/gdrive/MyDrive/cs1/data/dev.csv')
test = pd.read_csv('/content/gdrive/MyDrive/cs1/data/test.csv')


def all_data(x,y,z):

  """takes 3 data arguments, train, dev, test
     combine and drop duplicate
     delete null value if any
     return combined data"""

  data = pd.concat([x,y,z], ignore_index=True)
  data.drop_duplicates(inplace=True)
  data.columns = map(str.lower, data.columns)
  data['description'] = data['description'].map(str.lower)
  data.rename(columns = {'ogling/facial expressions/staring' : 'ogling',
  if data.isnull().values.any() == False:
    print(f'shape of combine data : {data.shape}')
  else:
    print(f'deleting nan values')
    data.dropna(axis = 1)
    print(f'shape of combine data : {data.shape}')

  return data
```

```
combined_data = all_data(train, dev, test)
combined_data.head()
#Ogling/Facial Expressions/Staring  Touching /Groping
```

shape of combine data : (9195, 4)

	description	commenting	ogling	grouping
0	was walking along crowded street, holding mums...	0	0	1
1	this incident took place in the evening.i was ...	0	1	0
2	i was waiting for the bus. a man came on a bik...	1	0	0
3	incident happened inside the train	0	0	0

1.2.1 PREPROCESSING DATA

```
lemmatizer = WordNetLemmatizer()
def preprocess(text):

    """performs common expansion of english words, preforms preprocessing

    text = re.sub(r"won\'t", "will not", text)   # decontracting the word
    text = re.sub(r"can\'t", "can not", text)
    text = re.sub(r"n\'t", " not", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'s", " is", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)

    text = re.sub(r'\w+:\s?','',text)
    text = re.sub('[([].*?[\)]', '', text)
    text = re.sub('[<[].*?[\>]', '', text)
    text = re.sub('[{[].*?[\}]', '', text)

    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split()]

    text = re.sub(r'\W', ' ', str(text))
    text = re.sub(r'\s+[a-zA-Z]\s+', ' ', text)
    text = re.sub(r"[^A-Za-z0-9]", " ", text)
```

```
    text = re.sub(r'[^\w\s]','',text)
    text = ' '.join(e for e in text.split() if e.lower() not in set(stopw
    # convert to lower and remove stopwords discard words whose len < 2

    text = re.sub("\s\s+" , " ", text)
    text = text.lower().strip()

    return text



combined_data['description'] = combined_data['description'].map(lambda x
```

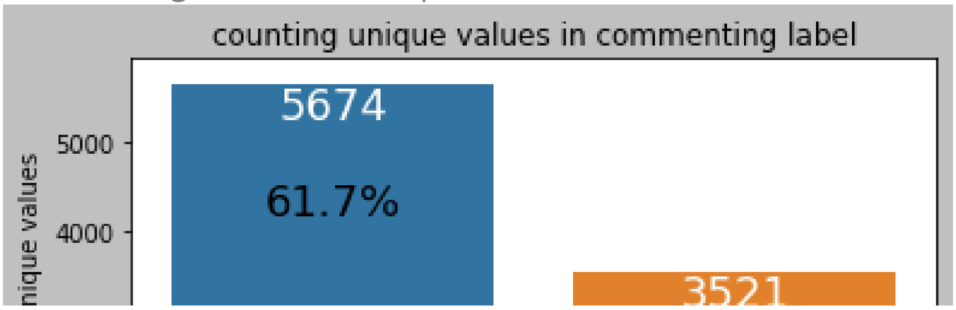
1.2.2 PLOTTING DATA FOR INSIGHTS

1.2.2.1 COUNTING PROPORTION OF UNIQUE VALUE IN EACH LABEL

```
def plot_count_data(data):

  '''take data as input
  outputs each label with their repective counts of 0, 1'''

  targt = data.columns.tolist()[1:]
  for i,lab in enumerate(targt):

    total = data.shape[0]

    fig = plt.figure()
    fig.patch.set_facecolor('silver')

    ax = sns.countplot(x=lab, data=data)
    #ax.set_title("count data")
    for p in ax.patches:
      ax.annotate(f'{p.get_height()}', (p.get_x()+0.4, p.get_height()+1.4
      percentage = '{:.1f}%'.format(100 * p.get_height()/total)
      ax.annotate(percentage, (p.get_x()+0.4, p.get_height()-1500), ha='c

    print(f'\n{lab} ratio 0:1 equals {round(data[lab].value_counts()[0]/d

    plt.title(f'counting unique values in {lab} label')
    plt.ylabel('count in each unique values')
    plt.xlabel(f'{lab} label unique values')
    plt.show()
```

```
    plt.show()

plot_count_data(combined_data)
```

```
commenting ratio 0:1 equals 1.61 : 1
```



OBSERVATION

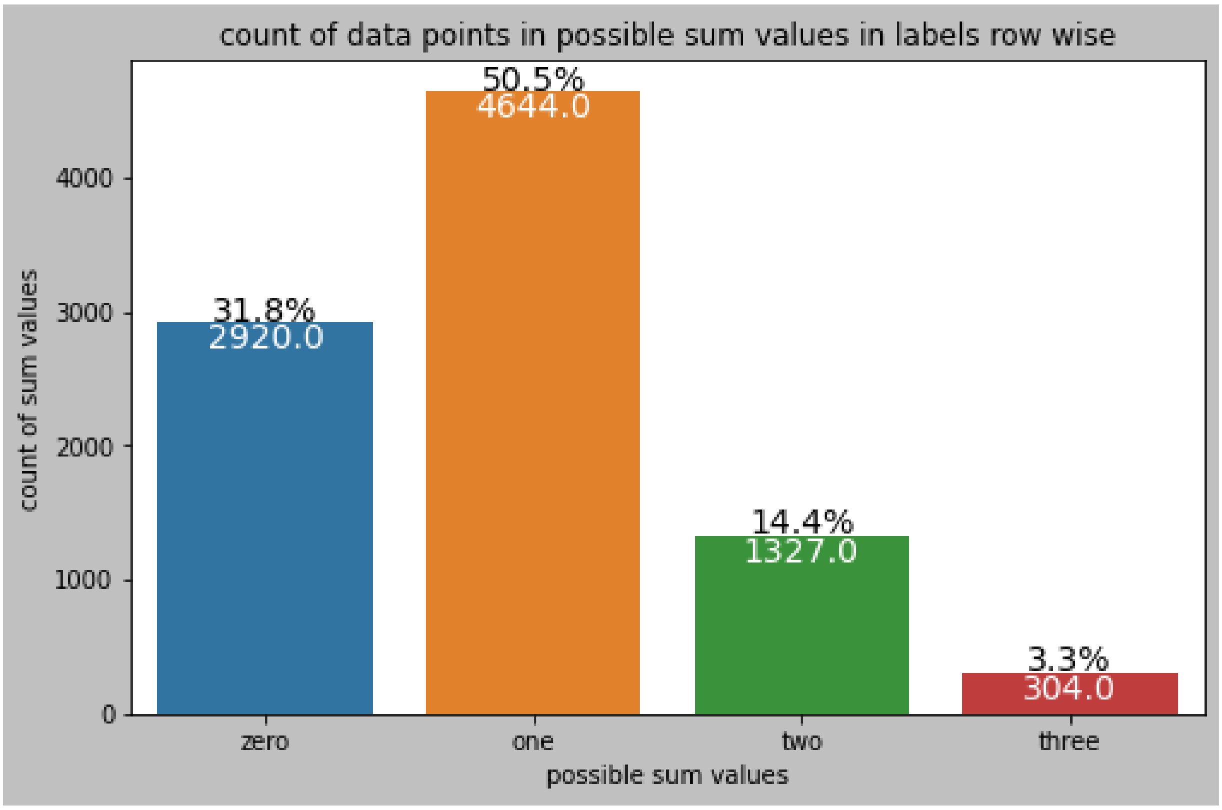
1. in commenting label 5674 have 0 label which compries of 61.7% of total data, and label 1 have 3521 data points which compries of 38.3% of total data.
2. commenting have ratio of 0:1 equals 1.61 : 1 (5674/3521) which we will be using in further case study.
3. in ogling label 7289 have 0 label which compries of 79.3% of total data, and label 1 have 1906 data points which compries of 20.7% of total data.
4. ogling have ratio of 0:1 equals 3.82 : 1 (7289/1906) which we will be using in further case study.
5. in grouping label 6412 have 0 label which compries of 69.7% of total data, and label 1 have 2783 data points which compries of 30.3% of total data.
6. grouping have ratio of 0:1 equals 2.3 : 1 (6412/2783) which we will be using in further case study.

1.2.2.2 COUNTING POSSIBLE SUM OF VALUES AND ITS PROPORTION ROW WISE

```
def row_label_count(frame, column_list, text):

  '''takes data as input
  calculates row wise label count
  returns frequency of items in labels row wise'''


  possible_label = {0:'zero', 1:'one', 2:'two', 3:'three'}
  dic = {}
```

```
for i in range(0,4):
  p0 = ((frame[column_list]).sum(axis=1)).values
  count = (p0 == i).sum()
  dic[i] = count
kd = dict((possible_label[key], value) for key,value in dic.items())

total = frame[text].shape[0]

fig = plt.figure(figsize=(8,5))
fig.patch.set_facecolor('silver')

ax = sns.barplot(x=list(kd.keys()), y=list(kd.values()))
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x()+0.4, p.get_height()+1.4
    percentage = '{:.1f}%'.format(100 * p.get_height()/total)
    ax.annotate(percentage, (p.get_x()+0.4, p.get_height()+10), ha='cer

plt.title('count of data points in possible sum values in labels row wi
plt.xlabel('possible sum values')
plt.ylabel('count of sum values')
```

```
row_label_count(combined_data, ['commenting', 'ogling', 'grouping'], 'des
```

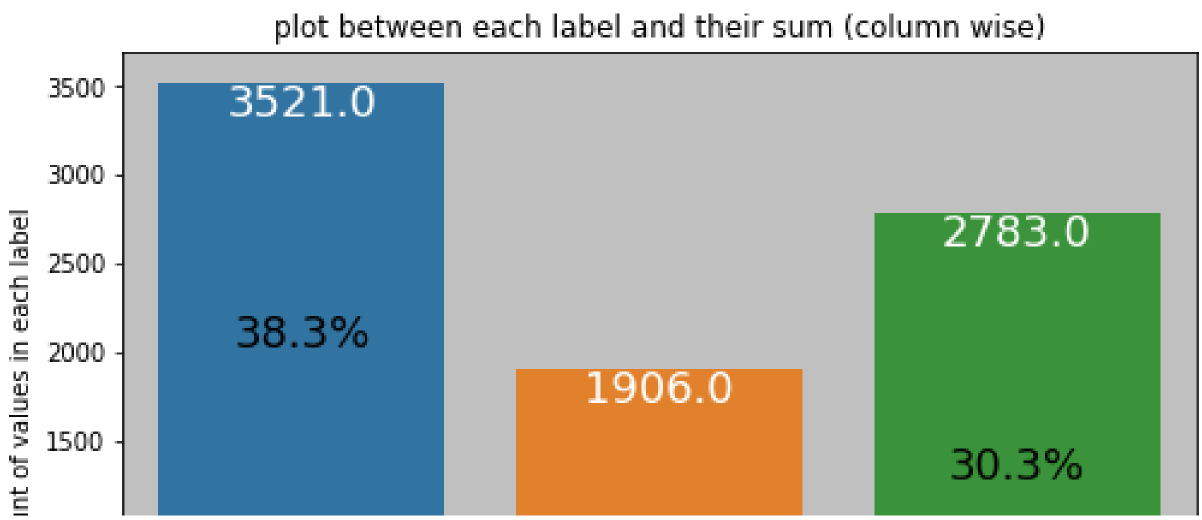


OBSERVATION

1. 4644 points which is roughly 50.5% of total data point, corresponds to any one of category which may be commenting or grouping or ogling.
2. 2920 points which is roughly 31.8% of total data point, have no labels which clearly depicts that the story does not correpond to any sexual harassment activity.
3. 1327 points which is roughly 14.4% of total data point, corresponds to any of two category which may be (commenting and grouping) or (commenting and ogling) or (ogling and grouping).
4. 304 points which is roughly 3.3% of total data point, corresponds to all the three category such as commenting, ogling and grouping.

1.2.2.3 COUNTING POSSIBLE SUM OF VALUES AND ITS PROPORTION COLUMN WISE

```
def zeros_ones(data):
  fig, ax = plt.subplots(figsize=(8, 5))
  ax.patch.set_facecolor('silver')
  total = data.shape[0]
  ax = sns.barplot(x=data.columns[1:].values, y=data.iloc[:,1:].sum(axis=
  for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x()+0.4, p.get_height()), ha=
    percentage = '{:.1f}%'.format(100 * p.get_height()/total)
    ax.annotate(percentage, (p.get_x()+0.4, p.get_height()-1500), ha='cer

  plt.title('plot between each label and their sum (column wise)')
  plt.xlabel('labels')
  plt.ylabel('count of values in each label')


zeros_ones(combined_data)
```



OBSERVATION

1. commenting bar have 3521 data points out of 9195 which means 38.3% of data in commenting have 1 and 61.7% are 0.
2. ogling bar have 1906 data points out of 9195 which means 20.7% of data in ogling have 1 and 79.3% are 0.
3. grouping bar have 2783 data points out of 9195 which means 30.3% of data in grouping have 1 and 69.7% are 0.

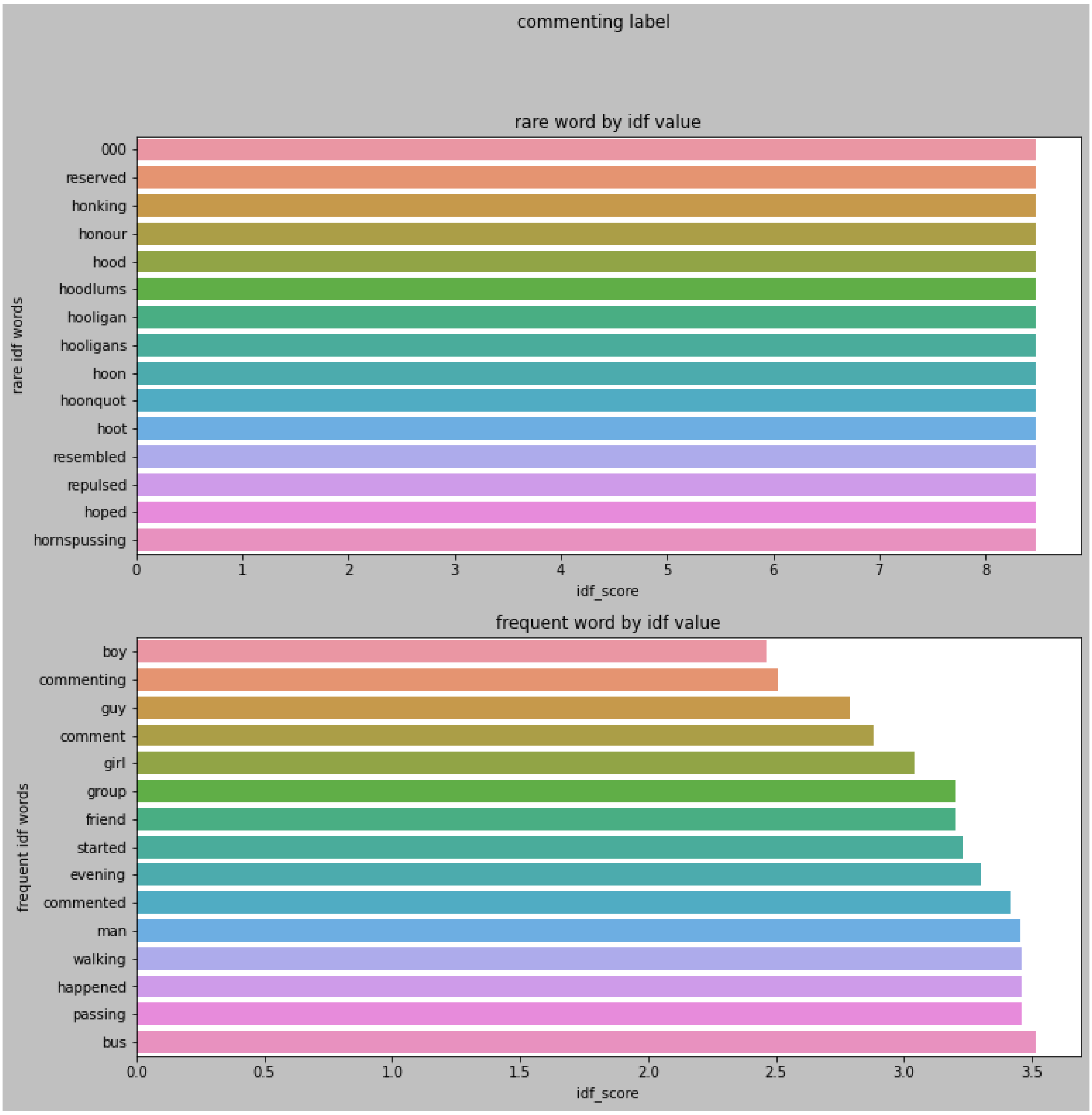
1.2.2.4 VIZUALIZATION OF FREQUENT AND RARE WORDS BASED ON IDF VALUES

```
def vizulaize_idf_rare_feq_word(frame, text, text_col, feat, i):

  '''takes frame : dataframe,
     text        : text column,
     text_col    : target label,
     feature     : int(max. which we want to display)
              i  : ngram range
     returns : frequent, rare words based on idf value
  '''

  tfidf_vect = TfidfVectorizer(ngram_range=(i,i), stop_words=set(stopworc
  idf        = tfidf_vect.fit_transform(frame[text][frame[text_col]==1])
  feat_names = tfidf_vect.get_feature_names()
  idf_value  = tfidf_vect.idf_
  df         = pd.DataFrame(list(zip(feat_names, idf_value)), columns=['v
```

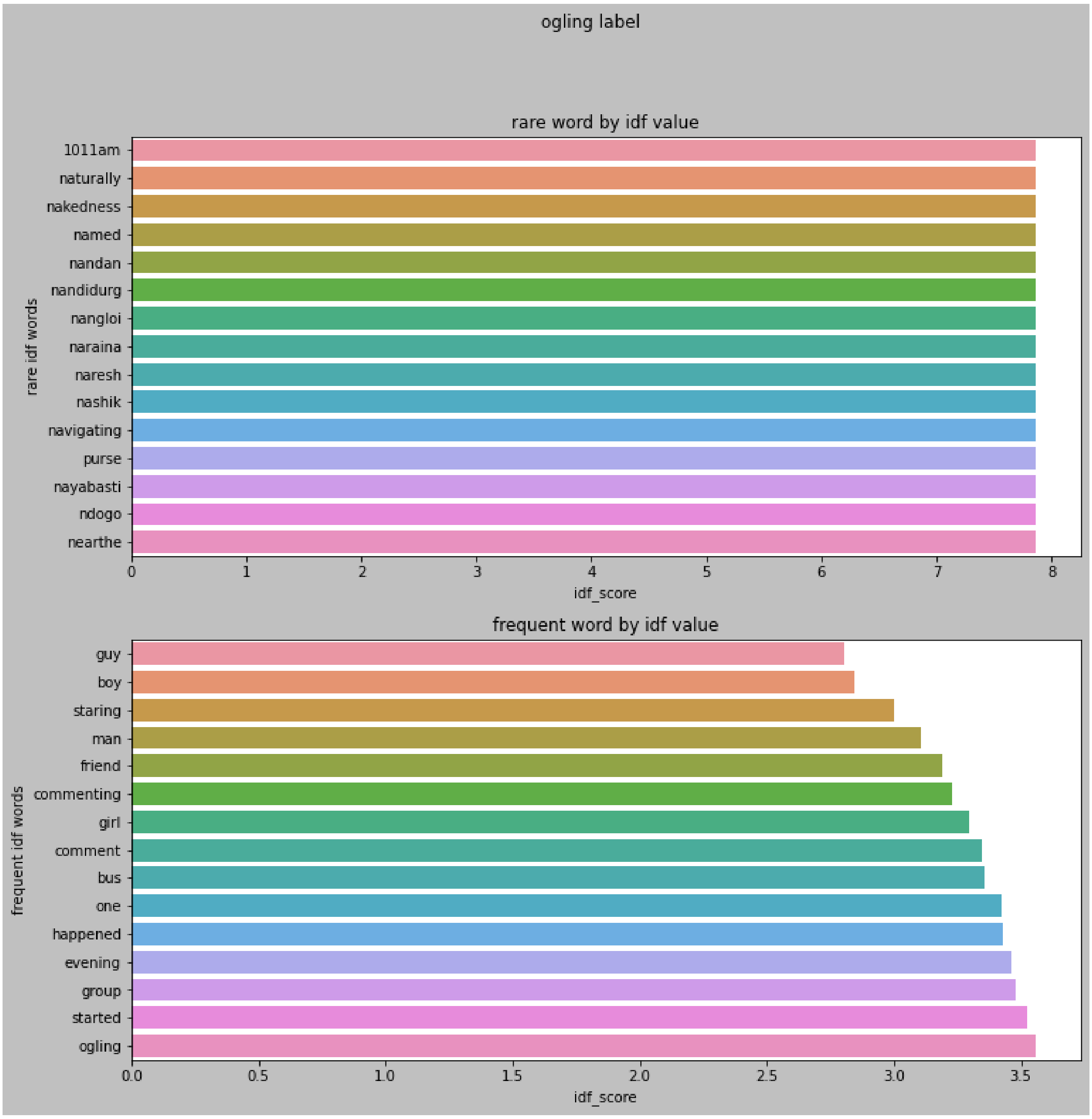
```
    df.sort_values("idf_value", axis = 0, ascending = False, inplace = True
    rare     = df['word'][:feat].tolist()
    rare_idf = df['idf_value'][:feat].tolist()


    fig = plt.figure(figsize=(12,12))
    fig.patch.set_facecolor('silver')

    plt.subplot(211)
    sns.barplot(y = rare, x = rare_idf)
    plt.title('rare word by idf value')
    plt.xlabel('idf_score')
    plt.ylabel('rare idf words')

    df.sort_values("idf_value", axis = 0, ascending = True, inplace = True,
    frequent          = df['word'][:feat].tolist()
    frequent_idf      = df['idf_value'][:feat].tolist()


    plt.subplot(212)
    sns.barplot(y = frequent, x = frequent_idf)
    plt.title('frequent word by idf value')
    plt.xlabel('idf_score')
    plt.ylabel('frequent idf words')

    plt.suptitle(f'{text_col} label')
  plt.show()
```

1.2.2.4.1 VISUALIZING COMMENTING, OGLING, GROUPING LABEL UNIGRAMS BASED ON IDF VALUES

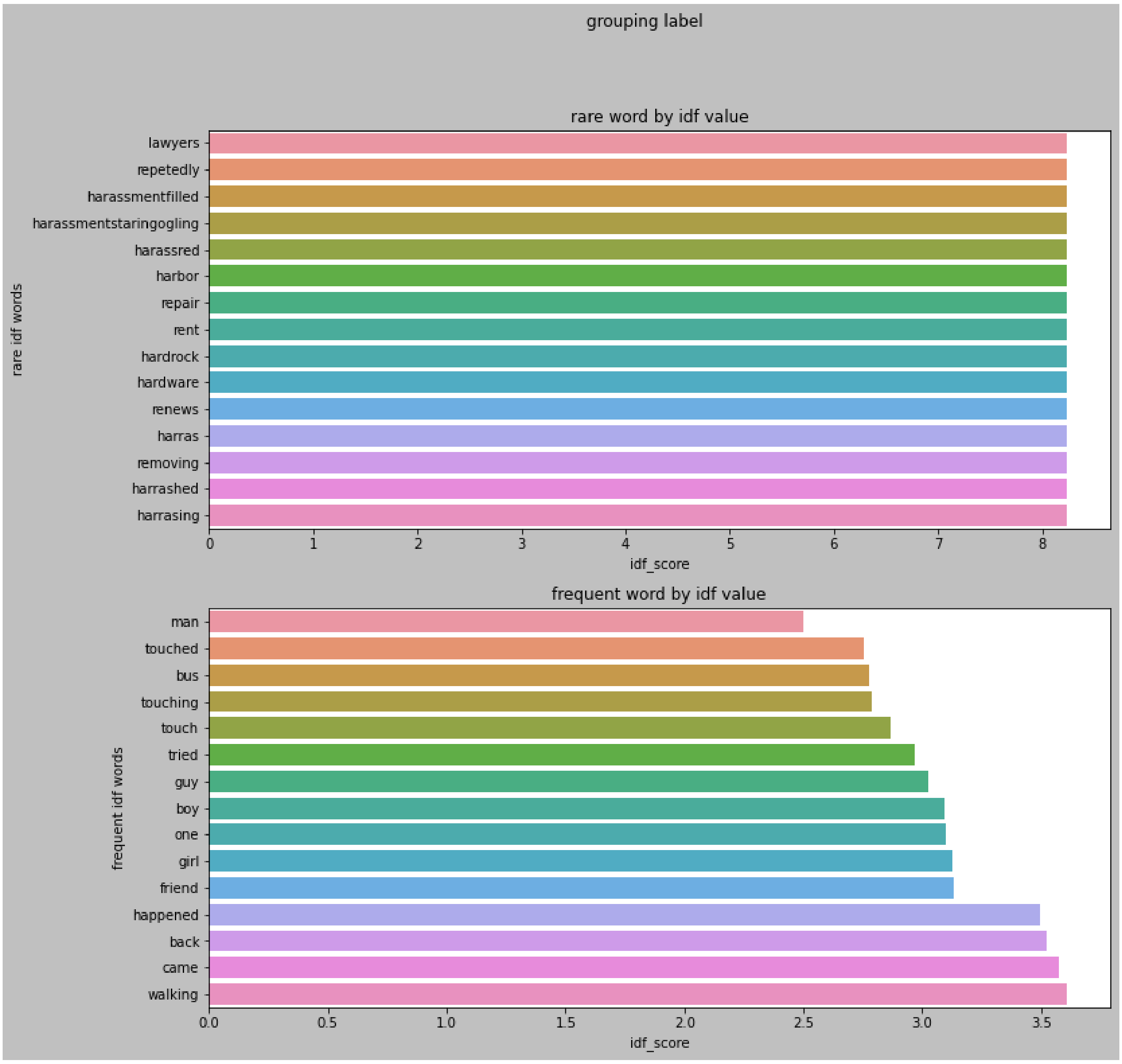
```
vizulaize_idf_rare_feq_word(combined_data, 'description', 'commenting', 1
```




```
vizulaize_idf_rare_feq_word(combined_data, 'description', 'ogling', 15, 1
```

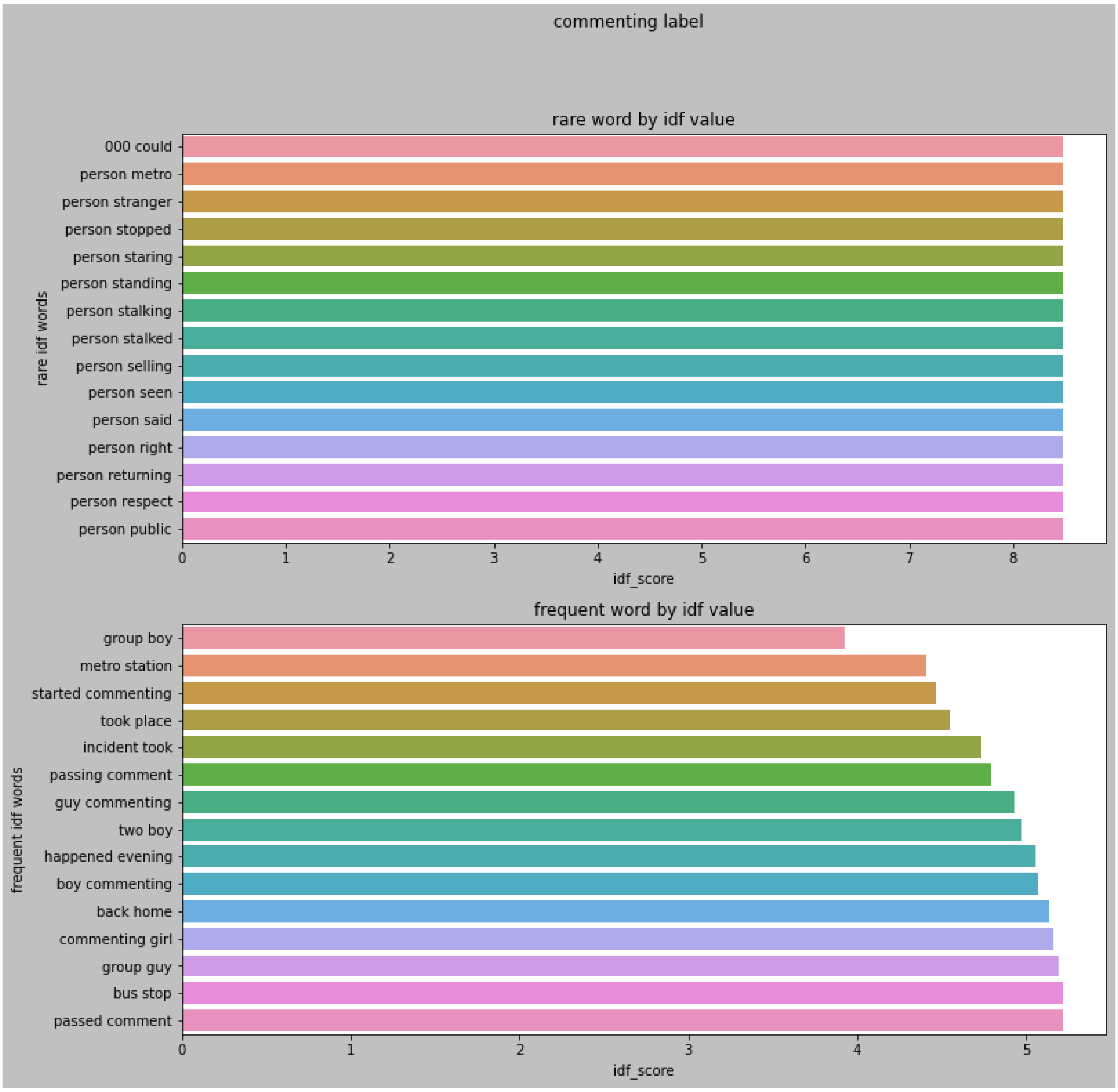



```
vizulaize_idf_rare_feq_word(combined_data, 'description', 'grouping', 15,
```

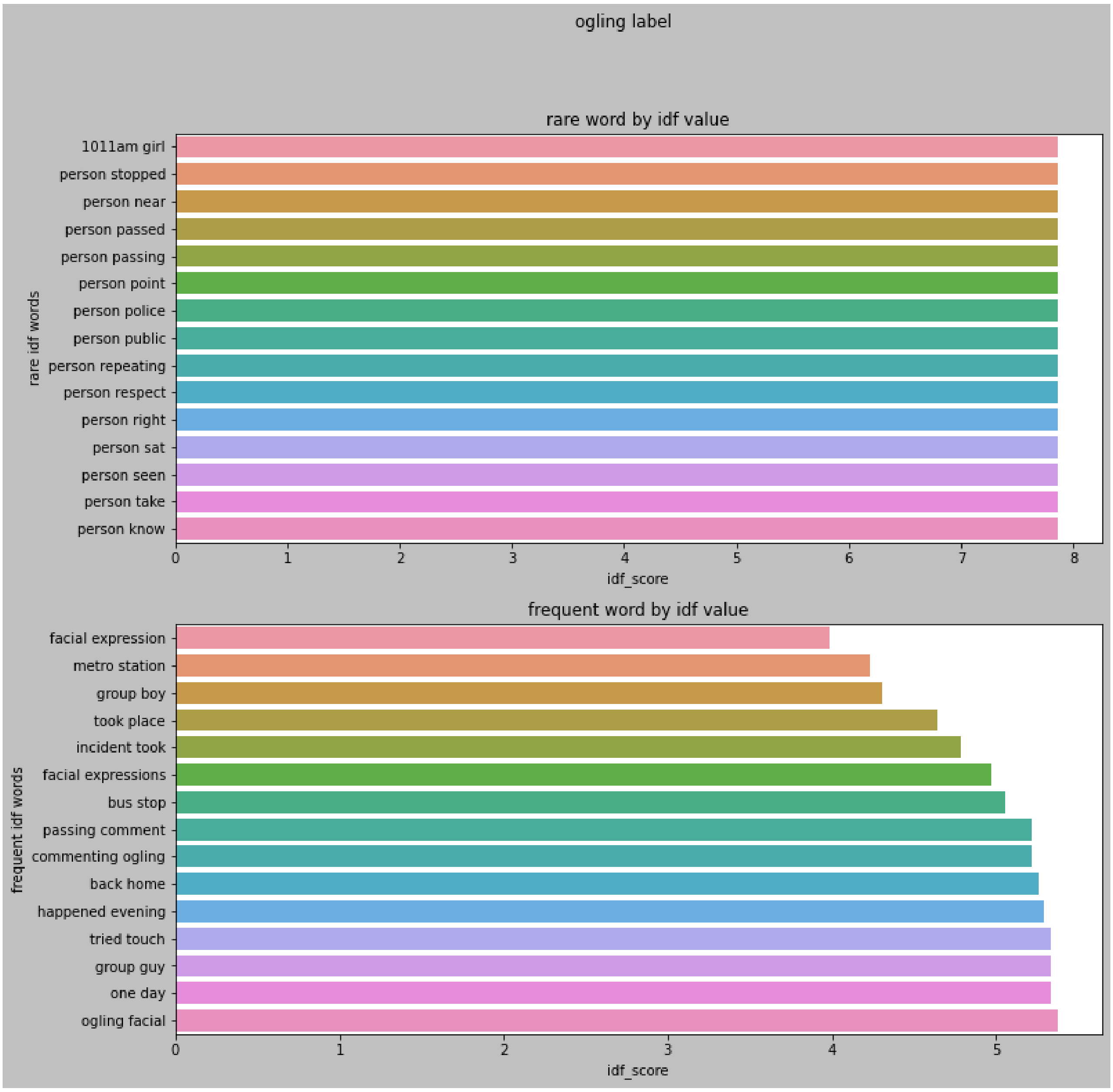


1.2.2.4.2 VISUALIZING COMMENTING, OGLING, GROUPING LABEL BIGRAMS BASED ON IDF VALUES

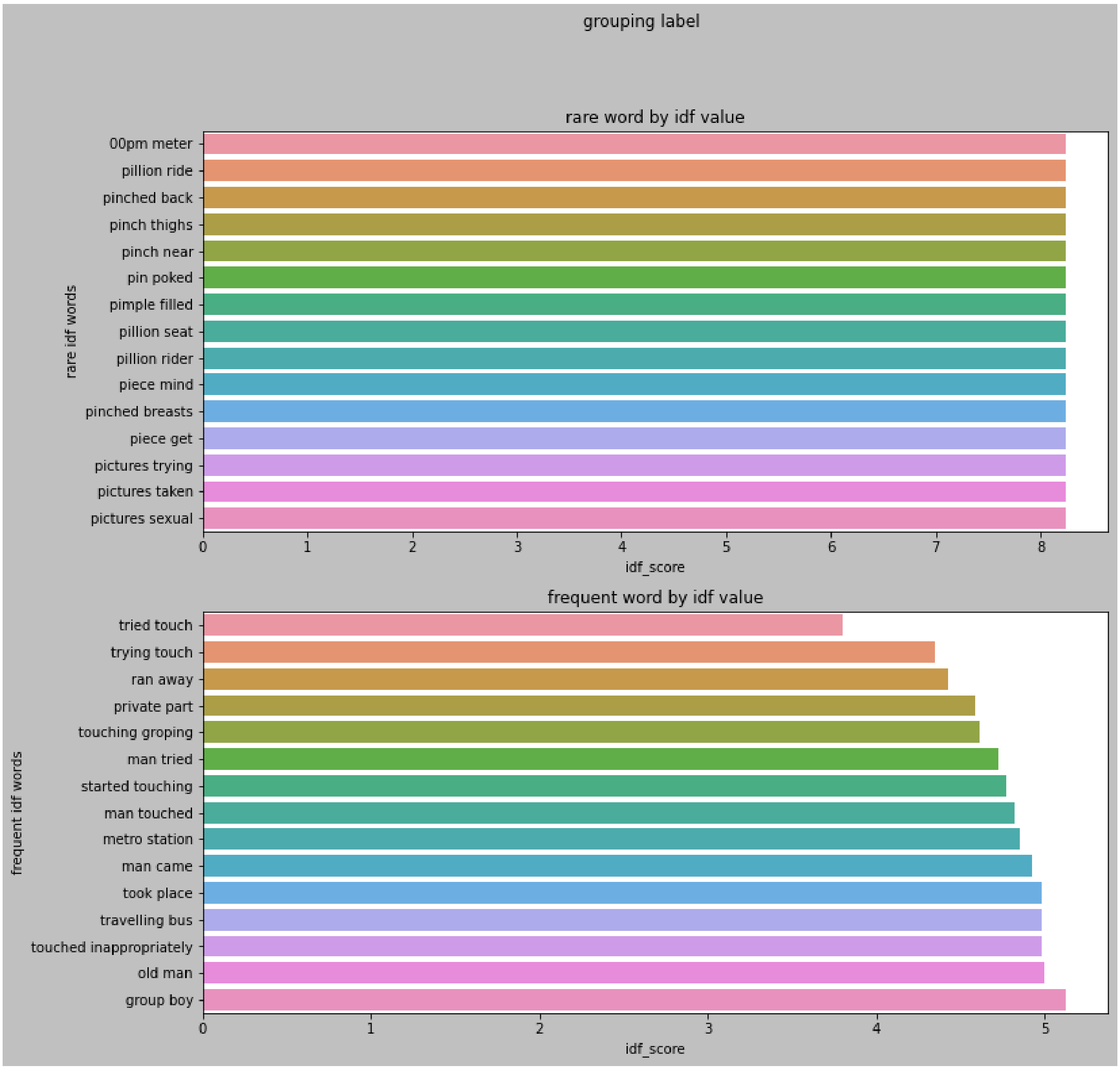
```
vizulaize_idf_rare_feq_word(combined_data, 'description', 'commenting', 1
```




```
vizulaize_idf_rare_feq_word(combined_data, 'description', 'ogling', 15, 2
```

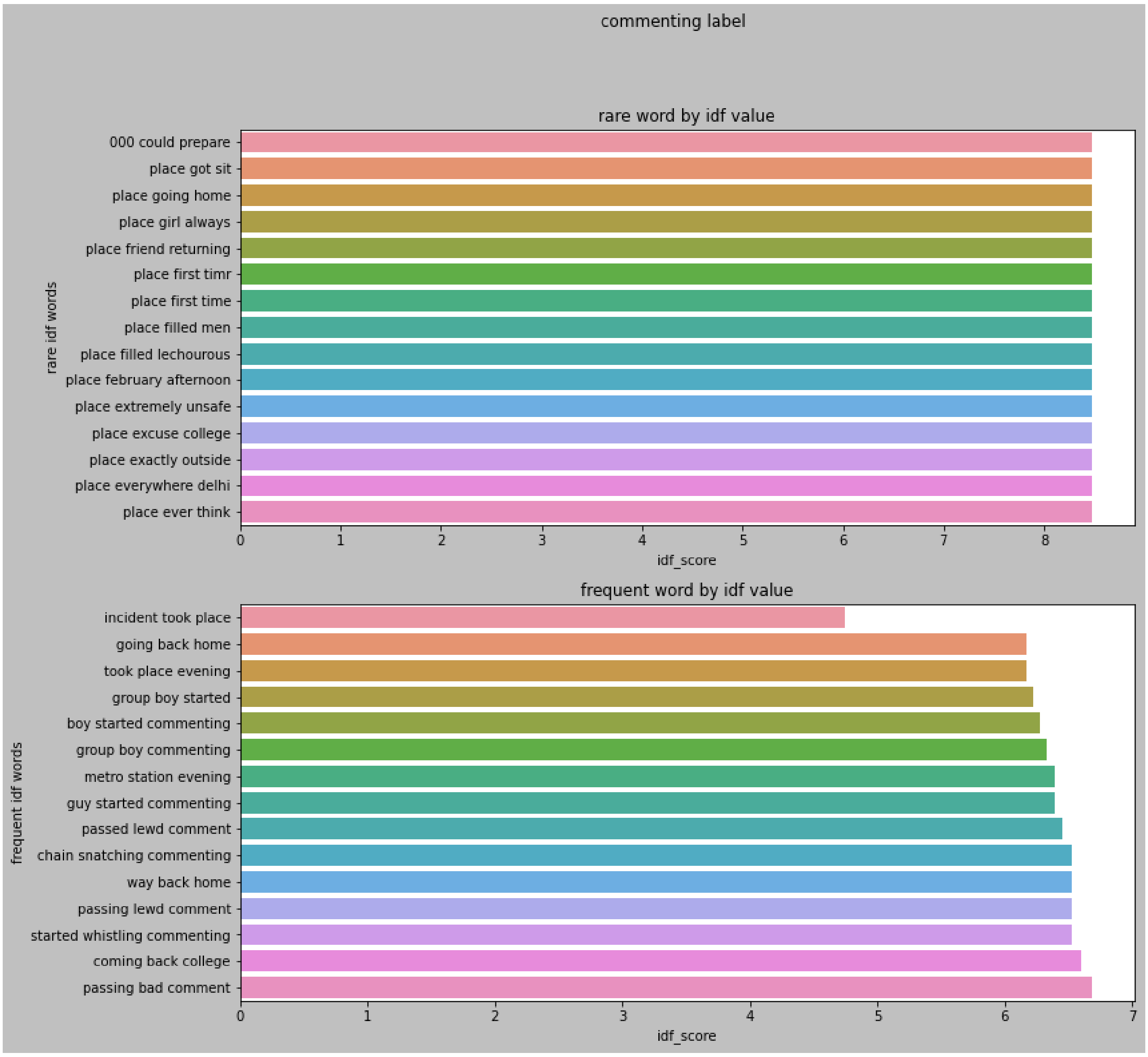



```
vizulaize_idf_rare_feq_word(combined_data, 'description', 'grouping', 15,
```

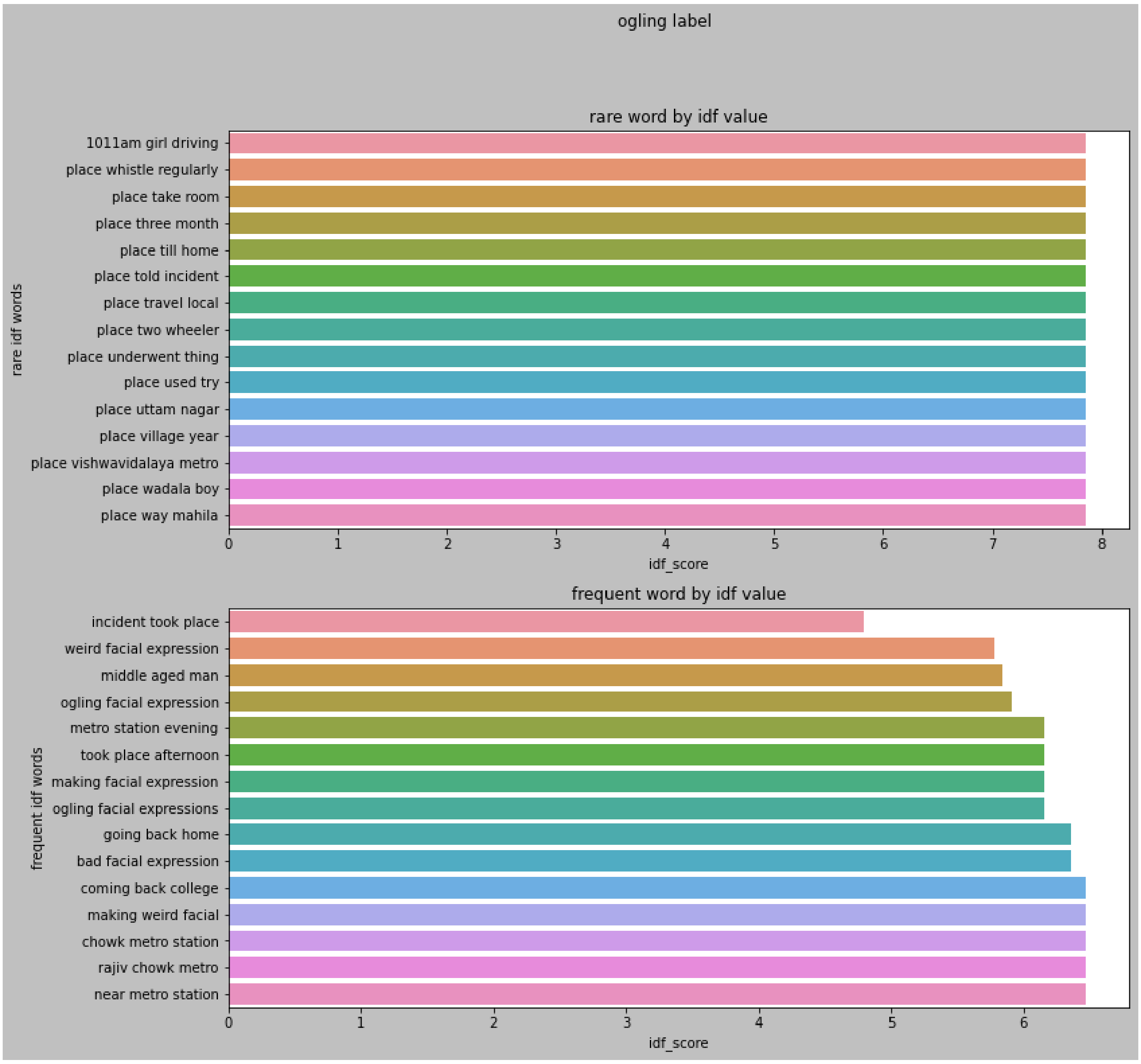


1.2.2.4.3 VISUALIZING COMMENTING, OGLING, GROUPING LABEL TRIGRAMS BASED ON IDF VALUES

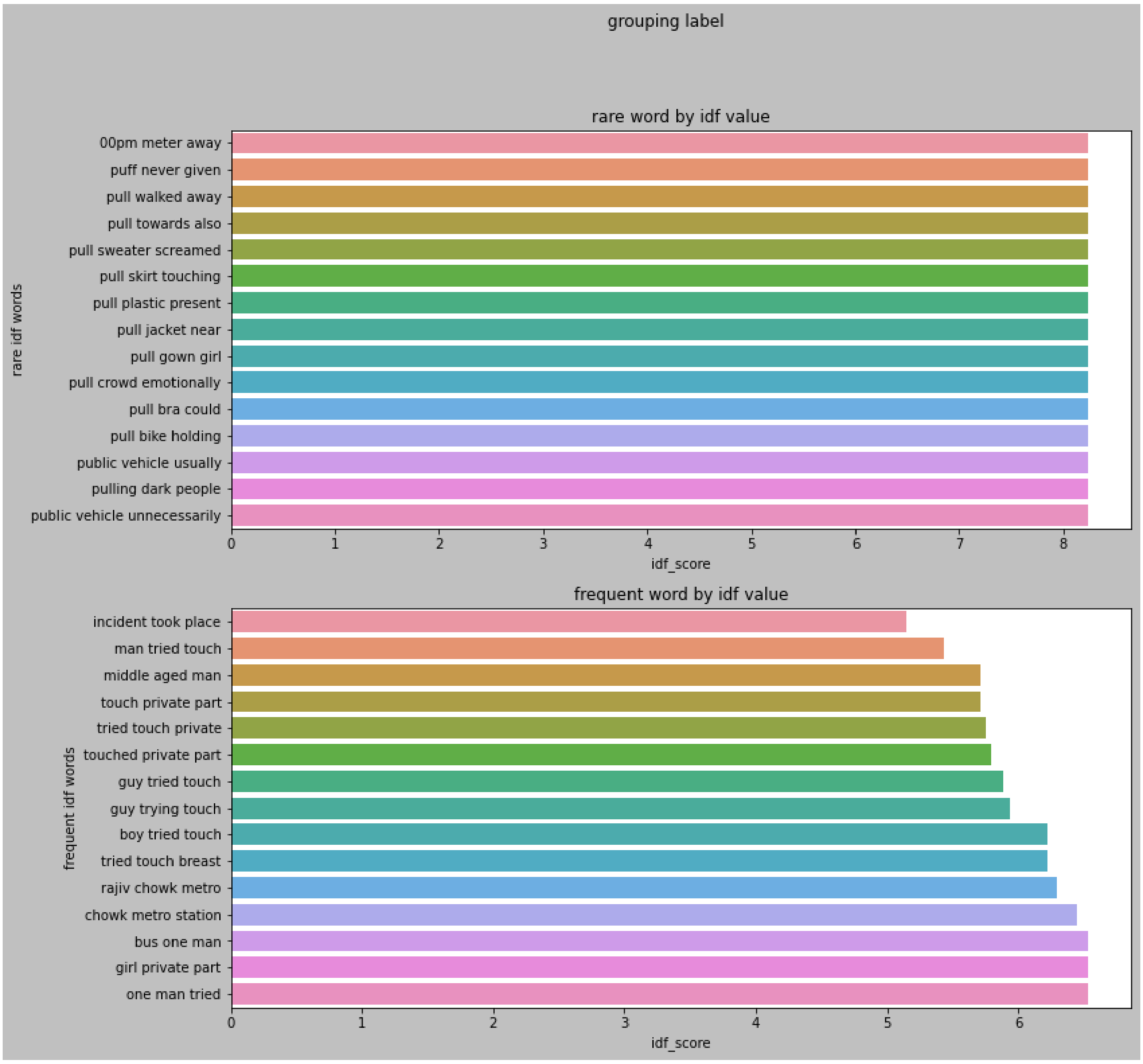
```
vizulaize_idf_rare_feq_word(combined_data, 'description', 'commenting', 1
```




```
vizulaize_idf_rare_feq_word(combined_data, 'description', 'ogling', 15, 1
```

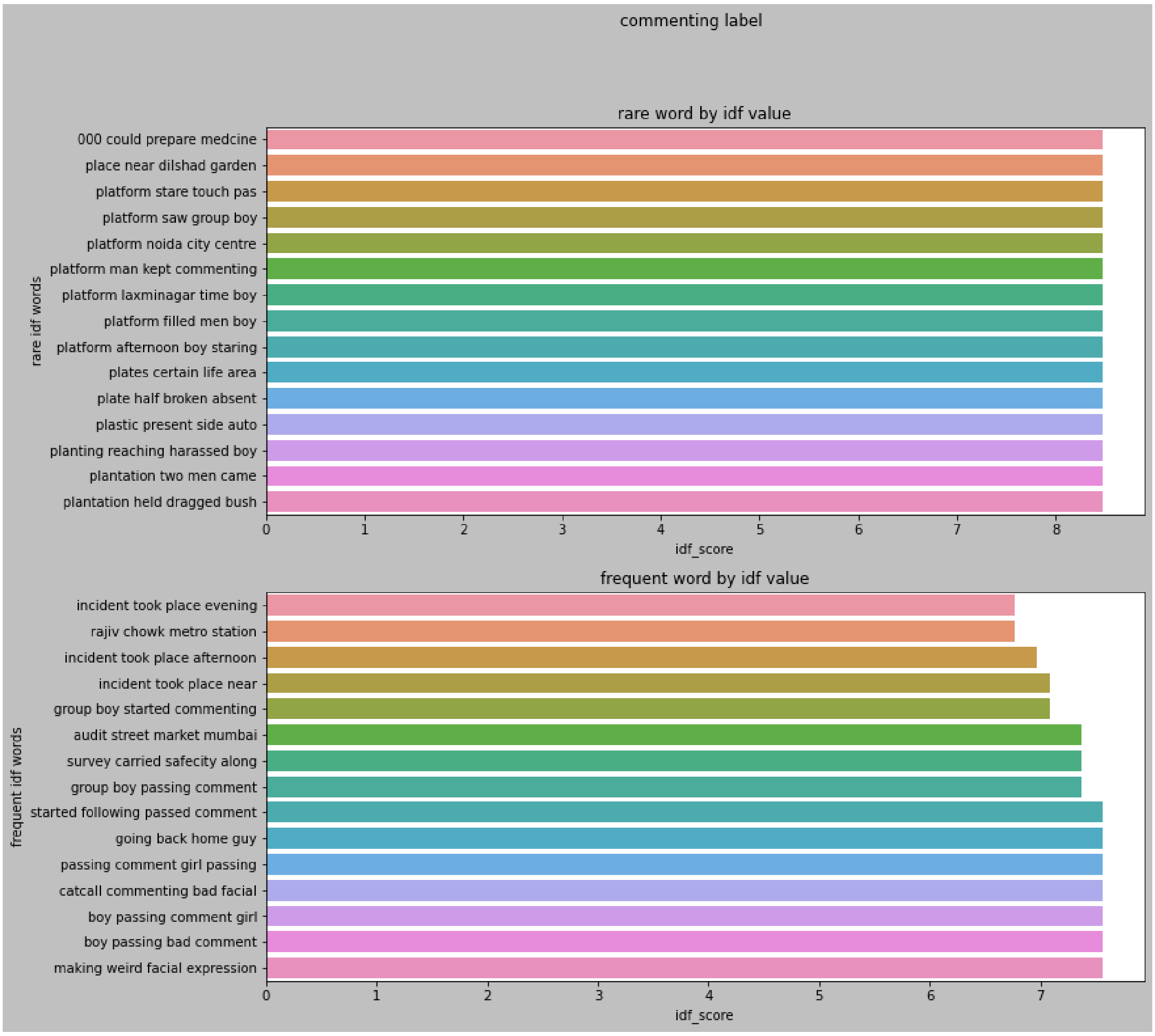



```
vizulaize_idf_rare_feq_word(combined_data, 'description', 'grouping', 15,
```

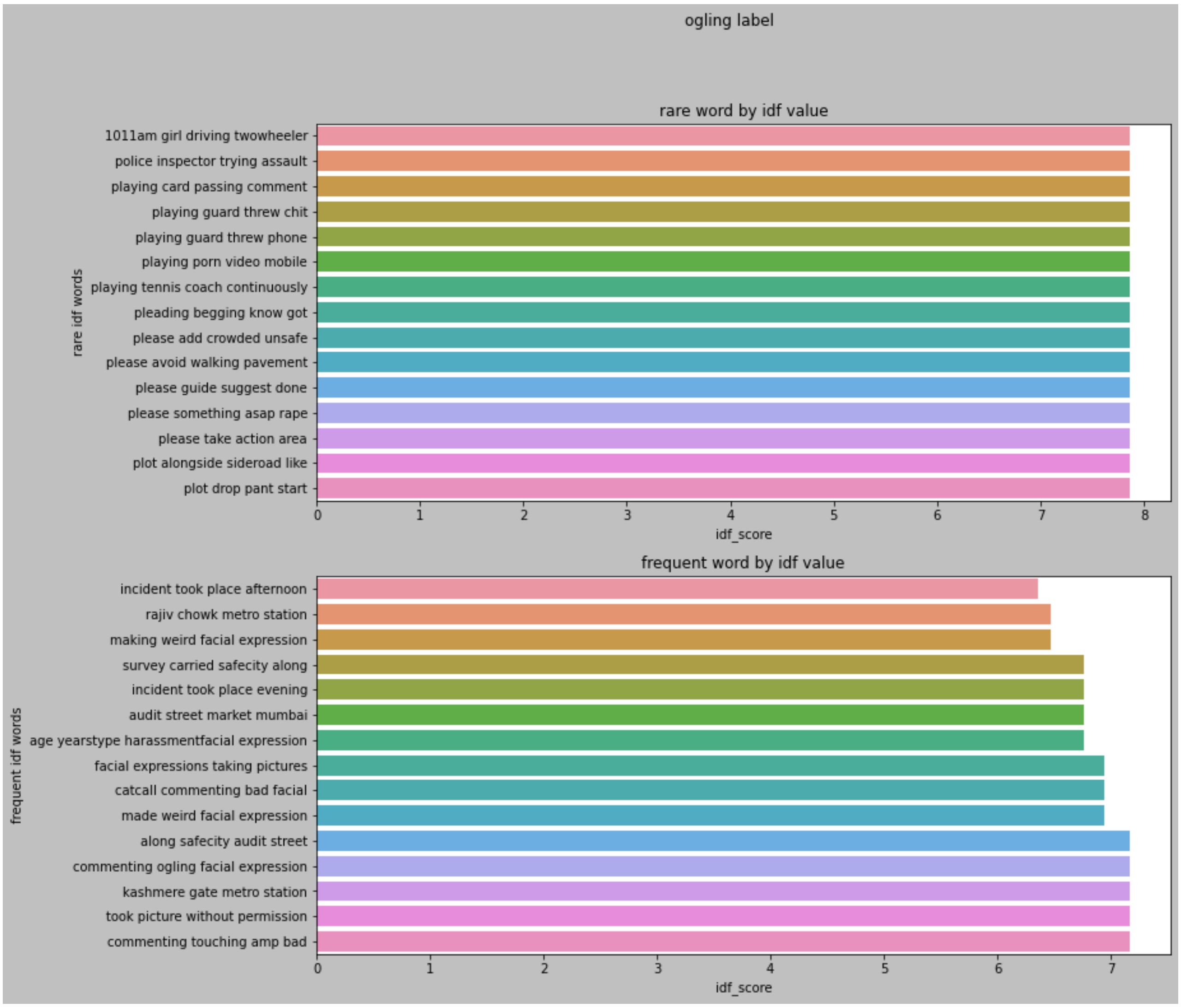


1.2.2.4.4 VISUALIZING COMMENTING, OGLING, GROUPING LABEL FOURGRAMS BASED ON IDF VALUES

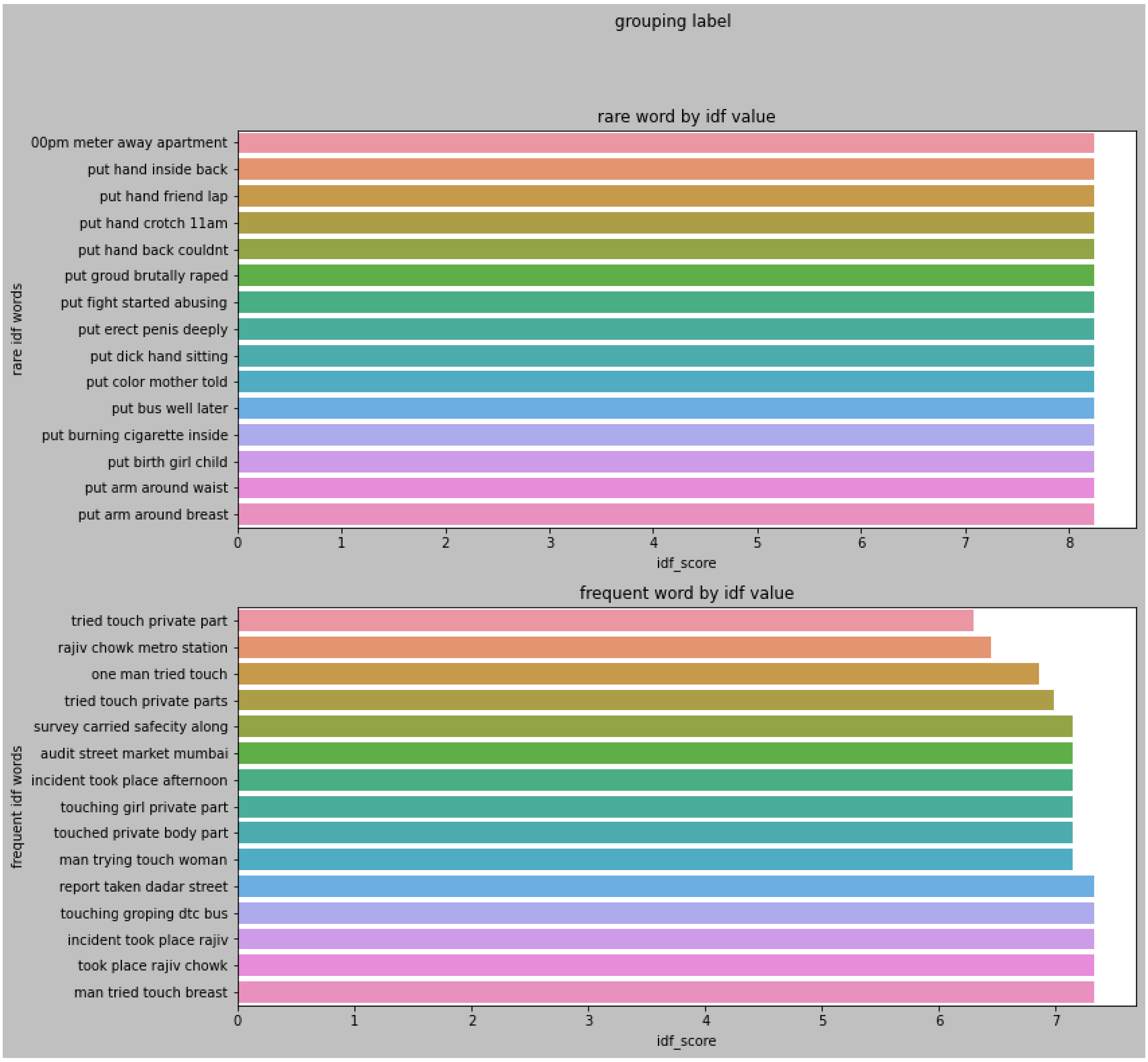
```
vizulaize_idf_rare_feq_word(combined_data, 'description', 'commenting', 1
```




```
vizulaize_idf_rare_feq_word(combined_data, 'description', 'ogling', 15, 4
```




```
vizulaize_idf_rare_feq_word(combined_data, 'description', 'grouping', 15,
```



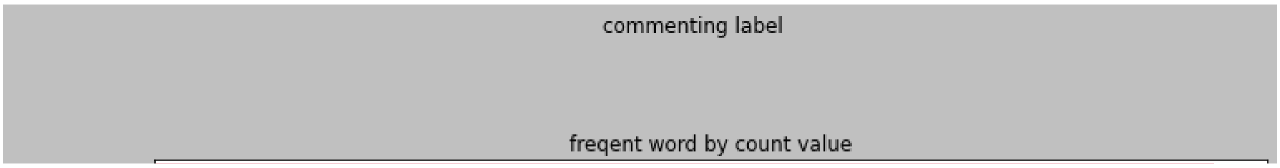
1.2.2.5 VIZUALIZATION OF FREQUENT AND RARE WORDS BASED ON COUNTVECTORIZER

```
def vizulaize_countvec_rare_feq_word(frame, text, text_col, feat, i, stop
  '''takes frame : dataframe,
     text        : text column,
     text_col    : target label,
```

```
        feature : int(max. which we want to display)
        returns : frequent, rare words based on idf value
      '''

      if stop_word == True:
        stop_word  = set(stopwords.words('english'))
        count_vect = CountVectorizer(ngram_range=(i,i), stop_words=stop_word)
      if stop_word == False:
        count_vect = CountVectorizer(ngram_range=(i,i), stop_words=None)

      cnt        = count_vect.fit_transform(frame[text][frame[text_col]==1])
      feat_names = count_vect.get_feature_names()
      count_value= cnt.toarray().sum(axis=0)
      df         = pd.DataFrame(list(zip(feat_names, count_value)), columns=[

      df.sort_values("count", axis = 0, ascending = False, inplace = True, ig
      freq     = df['word'][:feat].tolist()
      freq_cnt = df['count'][:feat].tolist()


      #fig = plt.figure(figsize=(20,5))
      fig = plt.figure(figsize=(12,12))
      fig.patch.set_facecolor('silver')



      plt.subplot(211)
      sns.barplot(y = freq, x = freq_cnt)
      plt.title('freqent word by count value')
      plt.xlabel('count of words')
      plt.ylabel('freq words')

      df.sort_values("count", axis = 0, ascending = True, inplace = True, igr
      rare         = df['word'][:feat].tolist()
      rare_cnt     = df['count'][:feat].tolist()

      plt.subplot(212)
      sns.barplot(y = rare, x = rare_cnt)
      plt.title('rare word by count value')
      plt.xlabel('count of words')
      plt.ylabel('rare words')

      plt.suptitle(f'{text_col} label')
    plt.show()
```

1.2.2.5.1 VIZUALIZATION OF COMMENTING, OGLING, GROUPING LABEL UNIGRAM BASED ON COUNTVECTORIZER WITH STOPWORDS

```
vizulaize_countvec_rare_feq_word(combined_data, 'description', 'commentir
```

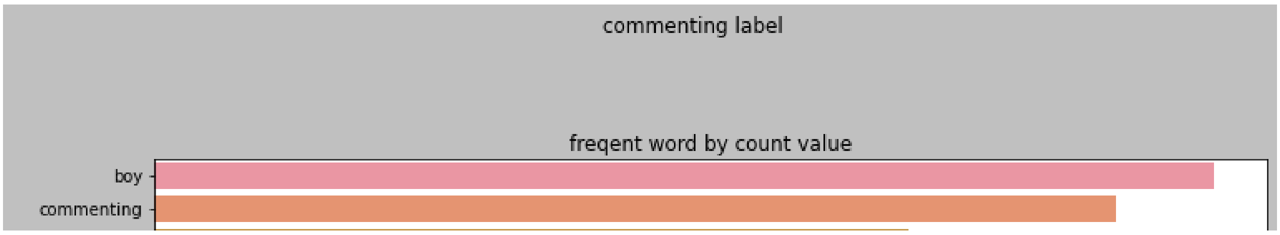



```
vizulaize_countvec_rare_feq_word(combined_data, 'description', 'ogling',
```

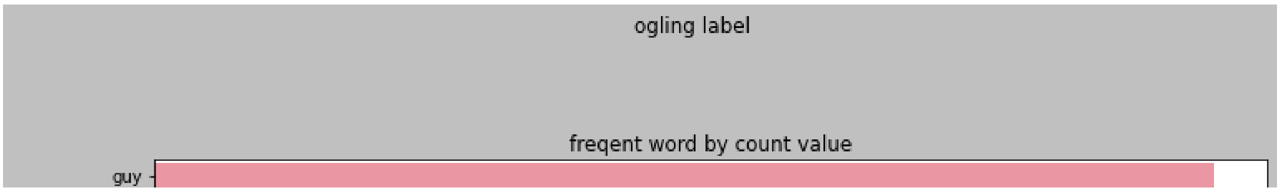

```
vizulaize_countvec_rare_feq_word(combined_data, 'description', 'grouping'
```

1.2.2.5.2 VIZUALIZATION OF COMMENTING, OGLING, GROUPING LABEL UNIGRAM BASED ON COUNTVECTORIZER WITHOUT STOPWORDS

```
vizulaize_countvec_rare_feq_word(combined_data, 'description', 'commentir
```



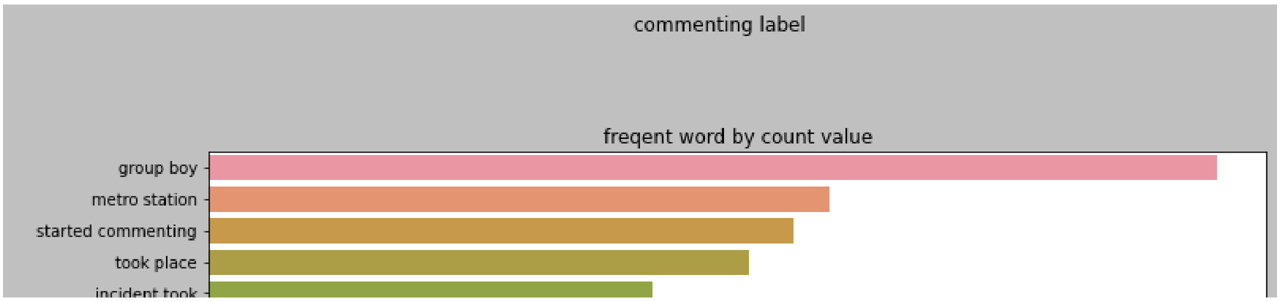

```
vizulaize_countvec_rare_feq_word(combined_data, 'description', 'ogling',
```



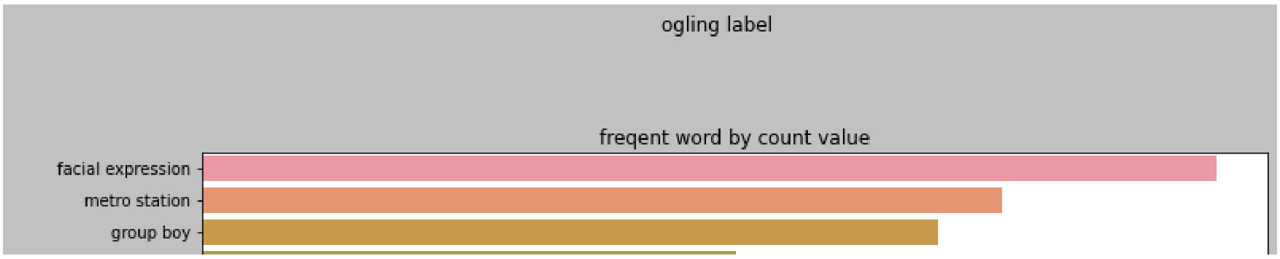

```
vizulaize_countvec_rare_feq_word(combined_data, 'description', 'grouping'
```

1.2.2.5.3 VIZUALIZATION OF COMMENTING, OGLING, GROUPING LABEL BIIGRAM BASED ON COUNTVECTORIZER WITH STOPWORDS

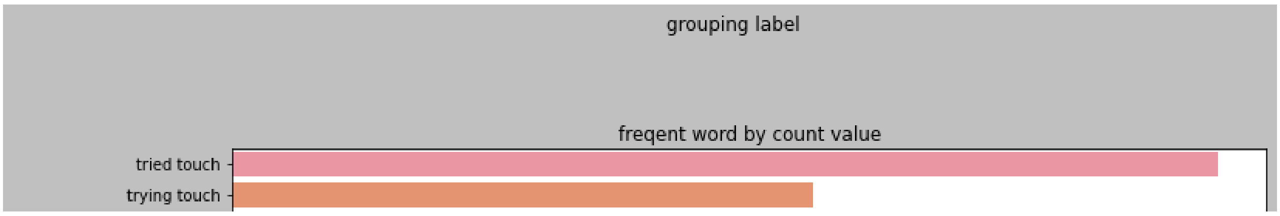
```
vizulaize_countvec_rare_feq_word(combined_data, 'description', 'commentir
```




```
vizulaize_countvec_rare_feq_word(combined_data, 'description', 'ogling',
```

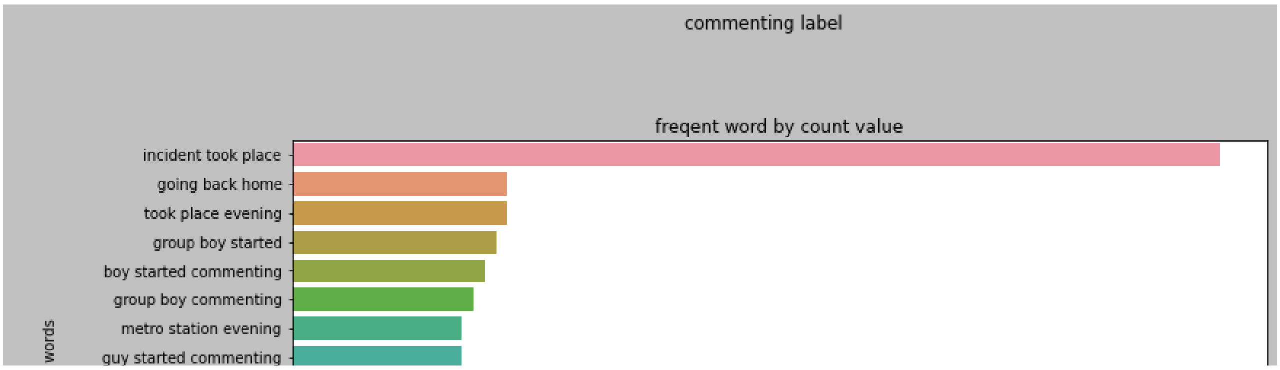



```
vizulaize_countvec_rare_feq_word(combined_data, 'description', 'grouping'
```

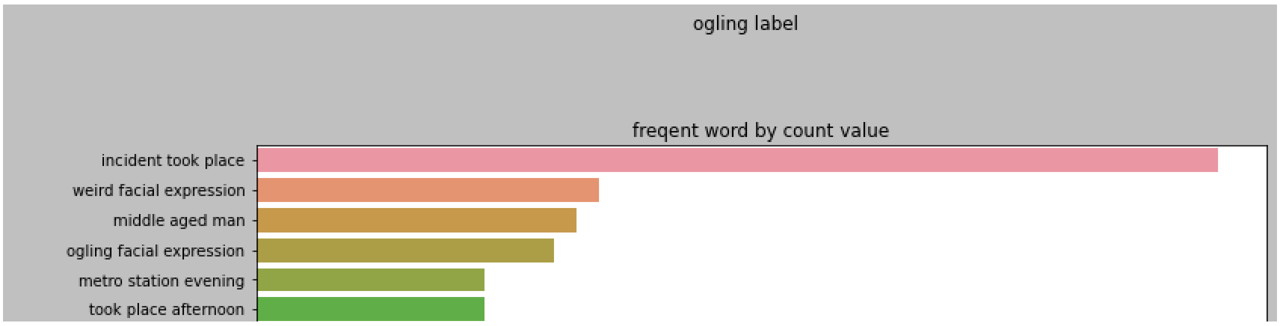


1.2.2.5.4 VIZUALIZATION OF COMMENTING, OGLING, GROUPING LABEL TRIGRAM BASED ON COUNTVECTORIZER WITH STOPWORDS

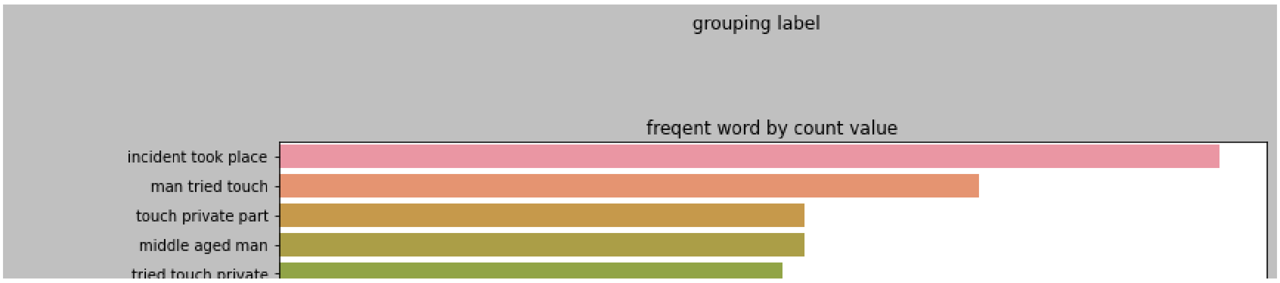
```
vizulaize_countvec_rare_feq_word(combined_data, 'description', 'commentir
```




```
vizulaize_countvec_rare_feq_word(combined_data, 'description', 'ogling',
```

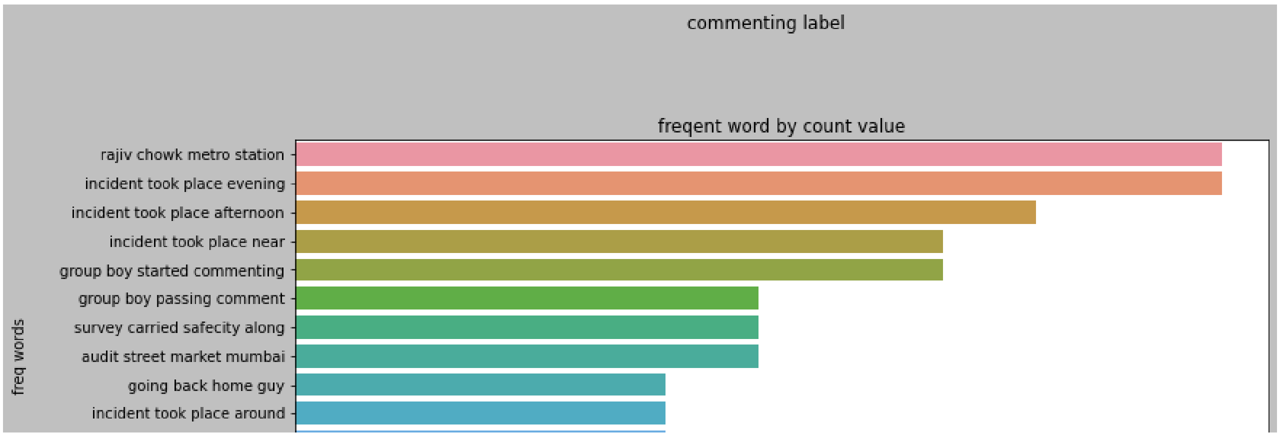



```
vizulaize_countvec_rare_feq_word(combined_data, 'description', 'grouping'
```

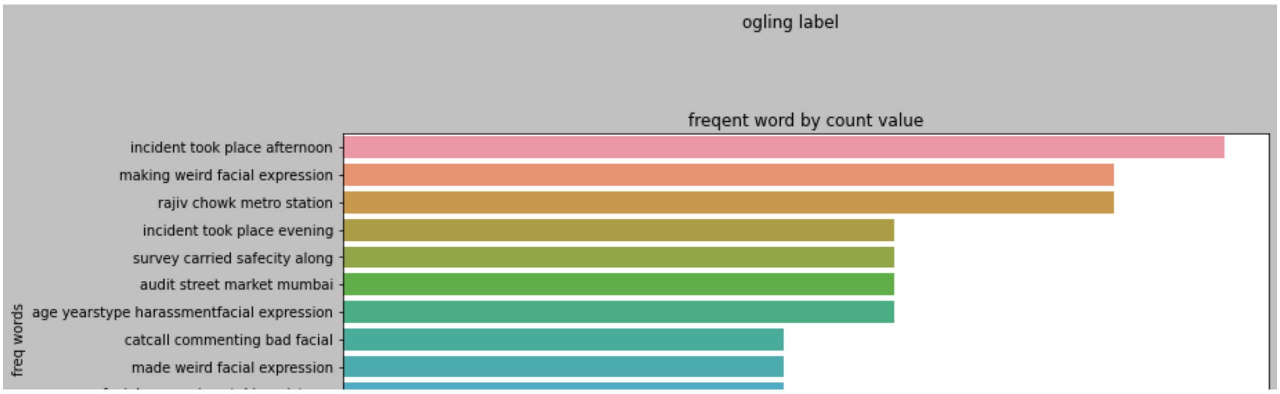


1.2.2.5.5 VIZUALIZATION OF COMMENTING, OGLING, GROUPING LABEL FOURGRAM BASED ON COUNTVECTORIZER WITH STOPWORDS

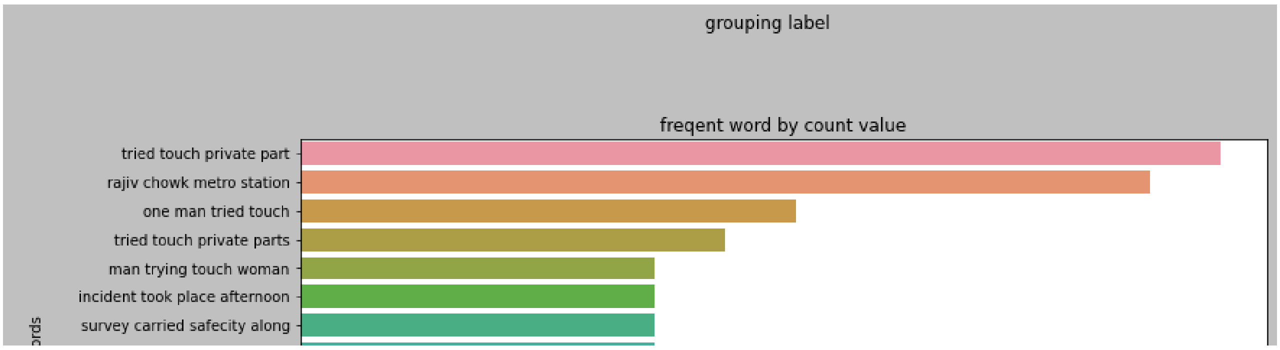
```
vizulaize_countvec_rare_feq_word(combined_data, 'description', 'commentir
```




```
vizulaize_countvec_rare_feq_word(combined_data, 'description', 'ogling',
```



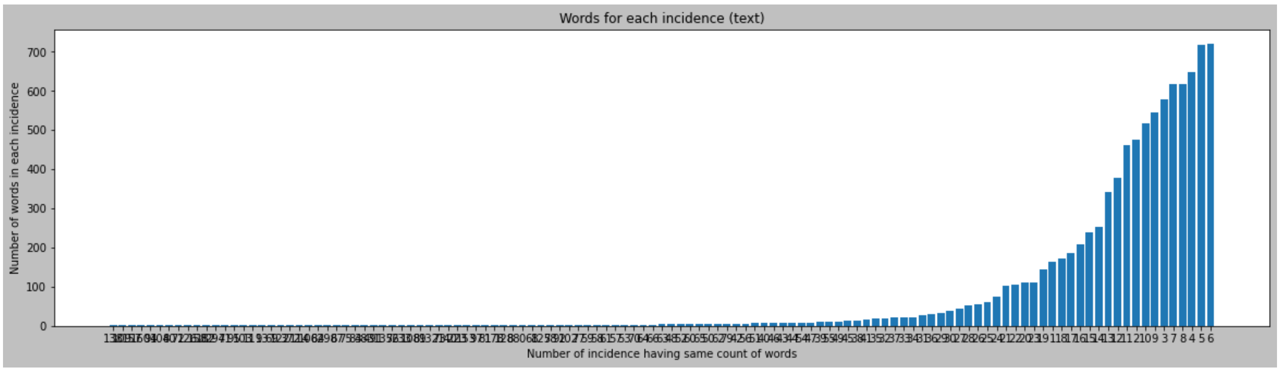

```
vizulaize_countvec_rare_feq_word(combined_data, 'description', 'grouping'
```



1.2.3 PLOTTING DATA FOR INSIGHTS INTO TEXT AND LABEL

1.2.3.1 PLOT OF INCIDENCE HAVING SAME NUMBER OF WORDS UNIVARIATE

```
#How to calculate number of words in a string in DataFrame: https://stack

def words_in_incidence(frame, column):

  '''frame : dataframe,
  column   : text column,
  return   : plot between no.of incidence having same count of words and

  word_count = frame[column].str.split().apply(len).value_counts()
  word_dict  = dict(word_count)
  word_dict  = dict(sorted(word_dict.items(), key=lambda kv: kv[1]))


  ind = np.arange(len(word_dict))
  fig = plt.figure(figsize=(20,5))
  fig.patch.set_facecolor('silver')
  p1 = plt.bar(ind, list(word_dict.values()))

  plt.ylabel('Number of words in each incidence')
  plt.xlabel('Number of incidence having same count of words')
  plt.title('Words for each incidence (text)')
  plt.xticks(ind, list(word_dict.keys()))
  plt.show()
  return word_count



words_in_incidence(combined_data, 'description')
```




```
6       721
5       717
4       649
7       619
8       619
       ...
89        1
132       1
73        1
140       1
223       1
Name: description, Length: 119, dtype: int64
```

OBSERVATION

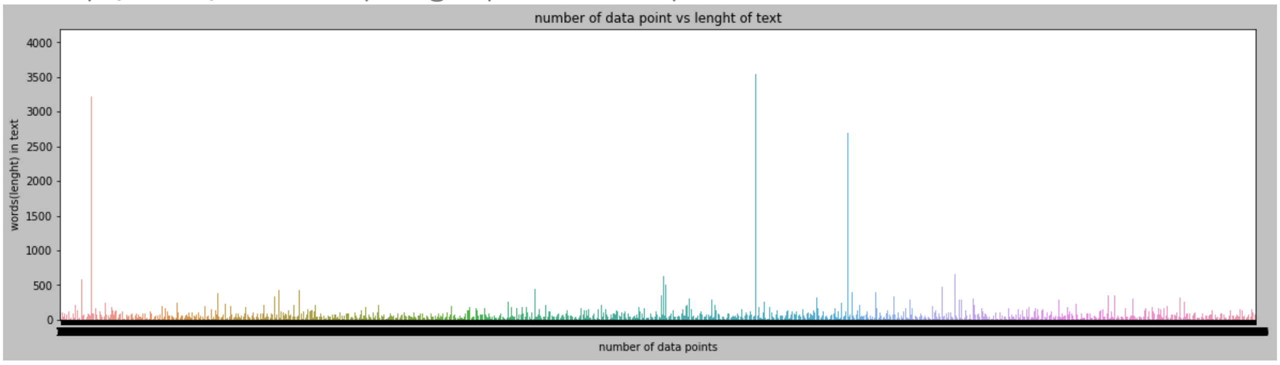
1. 6 (extreme right) incidence having more than 721 words.
2. 4 incidence have 649 words.

1.2.3.2 PLOT OF DATA POINTS AND THEIR LENGTH OF TEXT (WORDS) UNIVARIATE

```
fig = plt.figure(figsize=(20,5))
fig.patch.set_facecolor('silver')
sns.barplot(x=combined_data['description'].index, y= combined_data['descr
plt.title('number of data point vs lenght of text')
plt.xlabel('number of data points')
```

```
plt.ylabel('words(lenght) in text')
```

```
Text(0, 0.5, 'words(lenght) in text')
```




```
print(f"max of words among all data points : {max(combined_data['descript
print(f"min of words among all data points : {min(combined_data['descript
```

```
max of words among all data points : 3992
min of words among all data points : 0
```

1.2.3.3 PLOT OF EACH LABEL POSITIVE (1) OR NEGATIVE (0) CLAIM UNIVARIATE

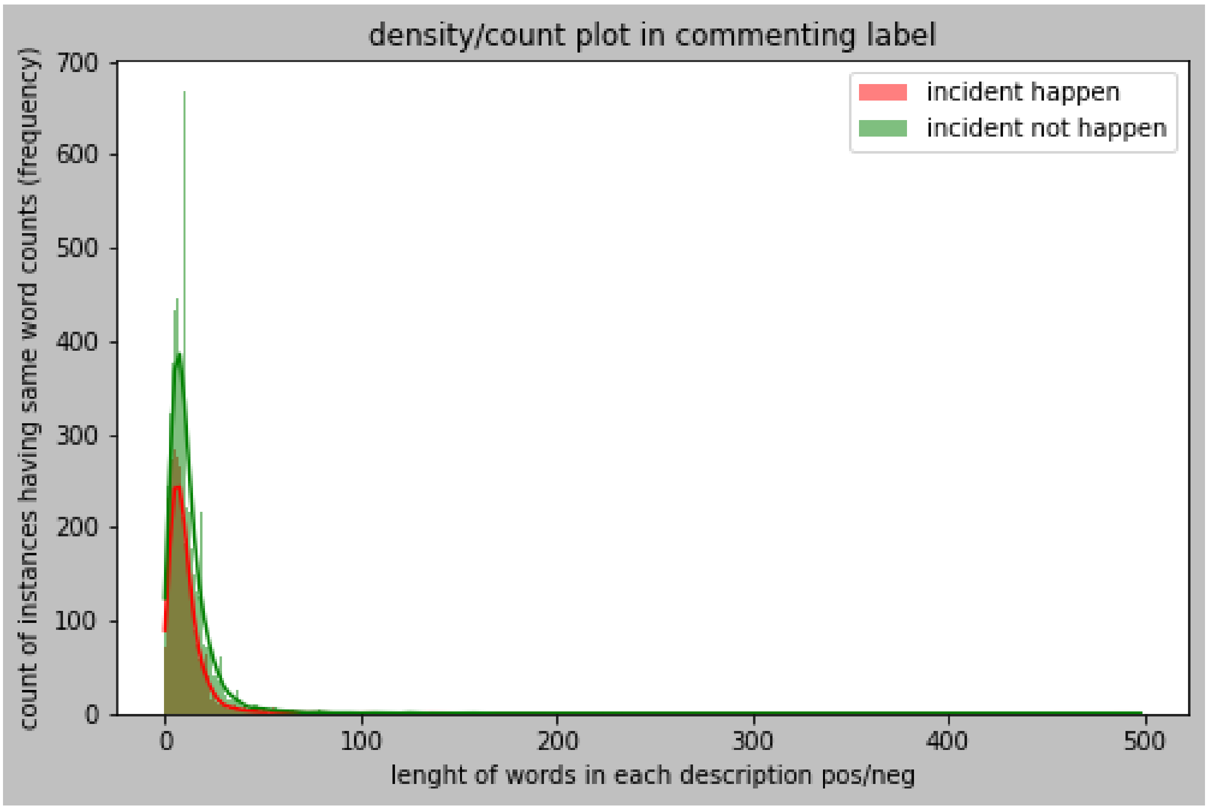
```
commenting_claim_pos = combined_data[combined_data['commenting']==1]['des
#commenting_claim_pos.values

ogling_claim_pos = combined_data[combined_data['ogling']==1]['descriptior
#ogling_claim_pos.values

grouping_claim_pos = combined_data[combined_data['grouping']==1]['descrip
#grouping_claim_pos.values

commenting_claim_neg = combined_data[combined_data['commenting']==0]['des
#commenting_claim_neg.values

ogling_claim_neg = combined_data[combined_data['ogling']==0]['descriptior
```

```
#ogling_claim_neg.values

grouping_claim_neg = combined_data[combined_data['grouping']==0]['descrip
#grouping_claim_neg.values
```

1.2.3.3.1 PLOT OF COMMENTING LABEL POSITIVE (1) OR NEGATIVE (0) CLAIM UNIVARIATE

```
def distribution_plt_pos_neg(posi, negi, label):
  fig = plt.figure(figsize=(8,5))
  fig.patch.set_facecolor('silver')
  sns.histplot(posi.values, kde=True, linewidth=0, label="incident happen
  sns.histplot(negi.values, kde=True, linewidth=0, label="incident not ha
  plt.title(f'density/count plot in {label} label')
  plt.xlabel('lenght of words in each description pos/neg')
  plt.ylabel('count of instances having same word counts (frequency)')  #
  plt.legend()
  plt.show()
```

```
distribution_plt_pos_neg(commenting_claim_pos, commenting_claim_neg, 'com
```



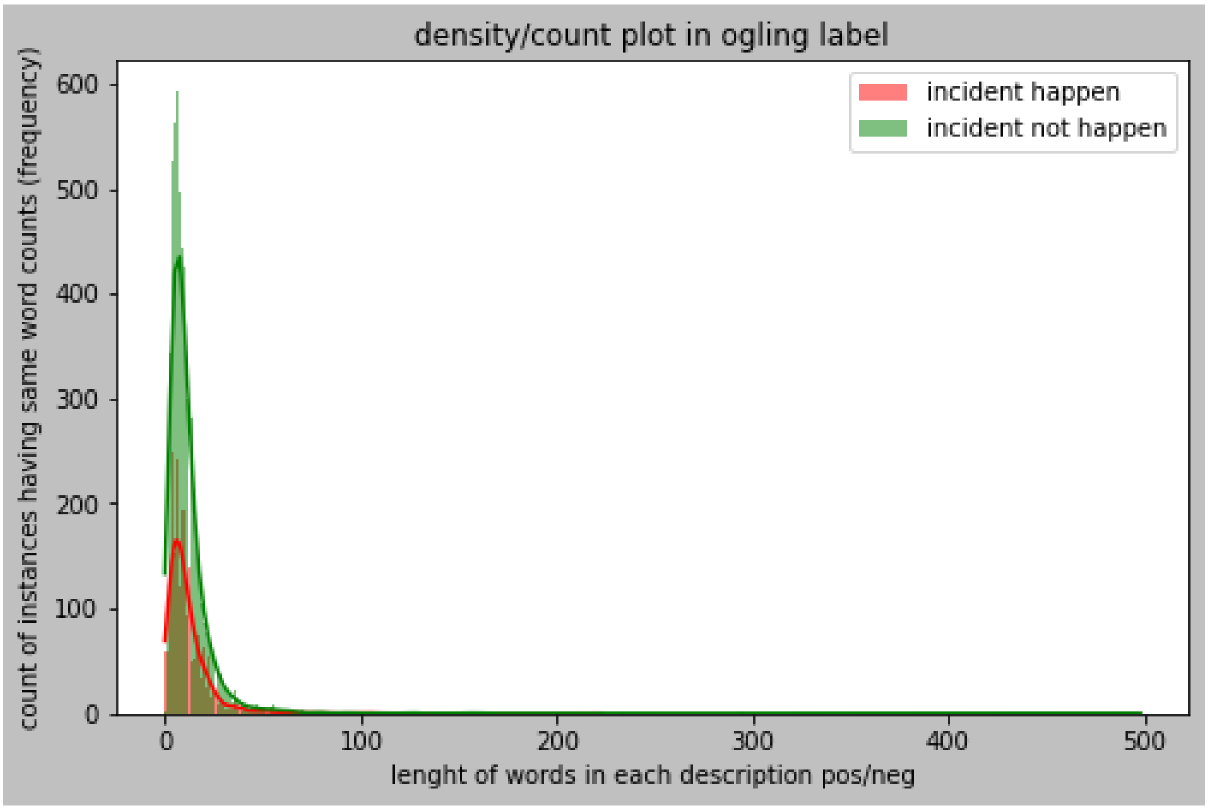

```
print('conclusion from above commenting label graph :')
print('-'*46)
print(f'max lenght of words in each description pos : {max(commenting_cla
```

```
print(f'max lenght of words in each description neg : {max(commenting_cla
print(f'count of instances having same word counts (frequency) pos : {max
print(f'count of instances having same word counts (frequency) neg : {max
print(' ')
print('incidence happen and not happen have very similar distribution wit
```

```
conclusion from above commenting label graph :
------------------------------------------
max lenght of words in each description pos : 491
max lenght of words in each description neg : 498
count of instances having same word counts (frequency) pos : 283
count of instances having same word counts (frequency) neg : 446

incidence happen and not happen have very similar distribution with n
```

1.2.3.3.2 PLOT OF OGLING LABEL POSITIVE (1) OR NEGATIVE (0) CLAIM UNIVARIATE

```
distribution_plt_pos_neg(ogling_claim_pos, ogling_claim_neg, 'ogling')
```



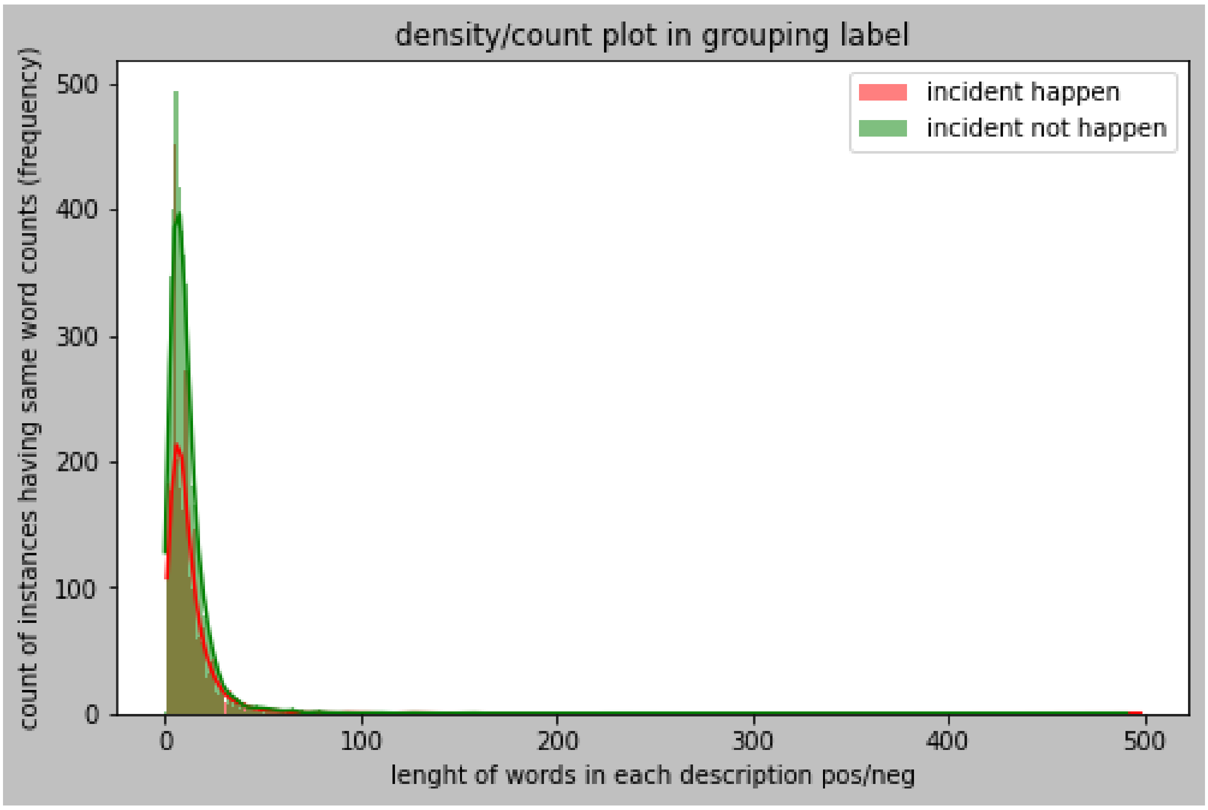

```
print('conclusion from above ogling label graph :')
print('-'*42)
print(f'max lenght of words in each description pos : {max(ogling_claim_p
print(f'max lenght of words in each description neg : {max(ogling_claim_n
print(f'count of instances having same word counts (frequency) pos : {max
```

```
print(f'count of instances having same word counts (frequency) pos : {max
print(f'count of instances having same word counts (frequency) neg : {max
print(' ')
print('incidence happen and not happen have very similar distribution wit
```

```
conclusion from above ogling label graph :
------------------------------------------
max lenght of words in each description pos : 223
max lenght of words in each description neg : 498
count of instances having same word counts (frequency) pos : 153
count of instances having same word counts (frequency) neg : 593

incidence happen and not happen have very similar distribution with n
```

1.2.3.3.3 PLOT OF GROUPING LABEL POSITIVE (1) OR NEGATIVE (0) CLAIM UNIVARIATE

```
distribution_plt_pos_neg(grouping_claim_pos, grouping_claim_neg, 'groupir
```




```
print('conclusion from above grouping label graph :')
print('-'*44)
print(f'max lenght of words in each description pos : {max(grouping_clair
print(f'max lenght of words in each description neg : {max(grouping_clair
print(f'count of instances having same word counts (frequency) pos : {max
print(f'count of instances having same word counts (frequency) neg : {max
```

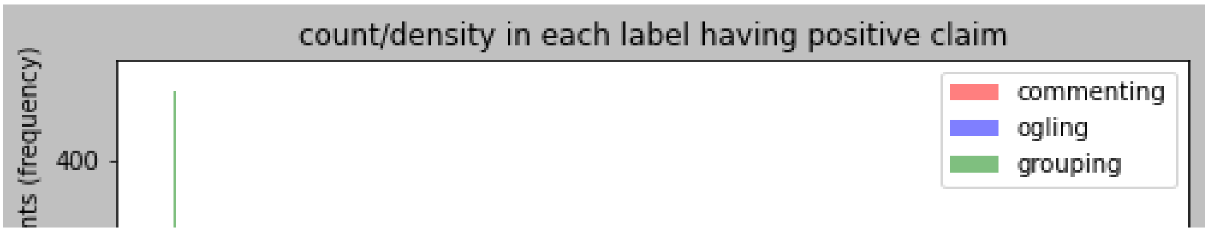
```
print(' ')
print('incidence happen and not happen have very similar distribution wit
```

```
conclusion from above grouping label graph :
--------------------------------------------
max lenght of words in each description pos : 498
max lenght of words in each description neg : 491
count of instances having same word counts (frequency) pos : 228
count of instances having same word counts (frequency) neg : 494

incidence happen and not happen have very similar distribution with n
```

1.2.3.3.4 PLOT OF COMMENTING, OGLING, GROUPING LABEL POSITIVE (1) CLAIM MULTIVARIATE

```
##distribution of pos claim in each label
def claim(x,y,z, lab):
  fig = plt.figure(figsize=(8,5))
  fig.patch.set_facecolor('silver')
  sns.histplot(x.values, kde=True, linewidth=0, label="commenting", col
  sns.histplot(y.values, kde=True, linewidth=0, label="ogling", color='
  sns.histplot(z.values, kde=True, linewidth=0, label="grouping", color
  plt.title(f'count/density in each label having {lab} claim')
  plt.xlabel('lenght of words in each description pos/neg')
  plt.ylabel('count of instances having same word counts (frequency)')
  plt.legend()
  plt.show()
```

```
claim(commenting_claim_pos, ogling_claim_pos, grouping_claim_pos, 'positi
```

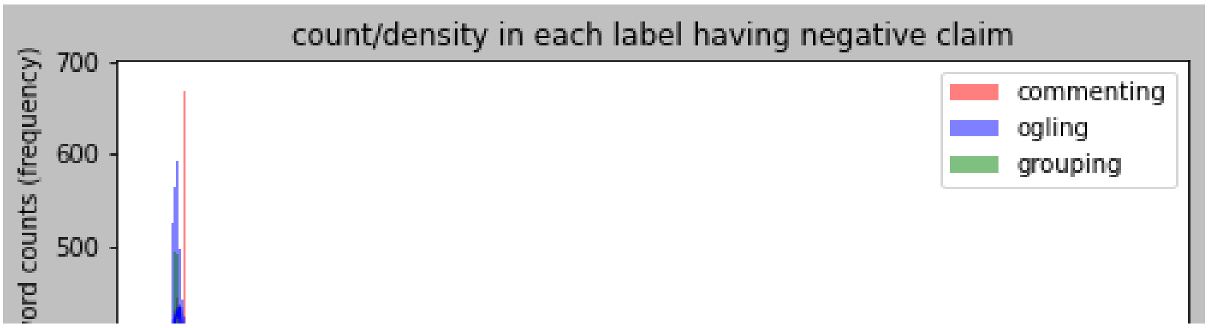



```
print('conclusion from above all three label with positive claim graph :'
print('-'*65)
print(f'commenting max lenght of words in each description pos   : {max(c
print(f'ogling max lenght of words in each description pos       : {max(c
print(f'grouping max lenght of words in each description pos     : {max(g
print(f'commenting count of instances having same word counts (frequency)
print(f'ogling count of instances having same word counts (frequency) pos
print(f'grouping count of instances having same word counts (frequency) p
print(' ')
print('commenting, ogling, grouping with positive claims have very simila
```

```
conclusion from above all three label with positive claim graph :
-----------------------------------------------------------------
commenting max lenght of words in each description pos   : 491
ogling max lenght of words in each description pos       : 223
grouping max lenght of words in each description pos     : 498
commenting count of instances having same word counts (frequency) pos
ogling count of instances having same word counts (frequency) pos
grouping count of instances having same word counts (frequency) pos

commenting, ogling, grouping with positive claims have very similar d
```

1.2.3.3.5 PLOT OF COMMENTING, OGLING, GROUPING LABEL NEGATIVE (0) CLAIM MULTIVARIATE

```
claim(commenting_claim_neg, ogling_claim_neg, grouping_claim_neg, 'negati
```




```
print('conclusion from above all three label with negative claim graph :'
print('-'*65)
print(f'commenting max lenght of words in each description neg : {max(con
print(f'ogling max lenght of words in each description neg     : {max(ogl
print(f'grouping max lenght of words in each description neg   : {max(grc
print(f'commenting count of instances having same word counts (frequency)
print(f'ogling count of instances having same word counts (frequency) neg
print(f'grouping count of instances having same word counts (frequency) r
print(' ')
print('commenting, ogling, grouping with negative claims have very simila
```

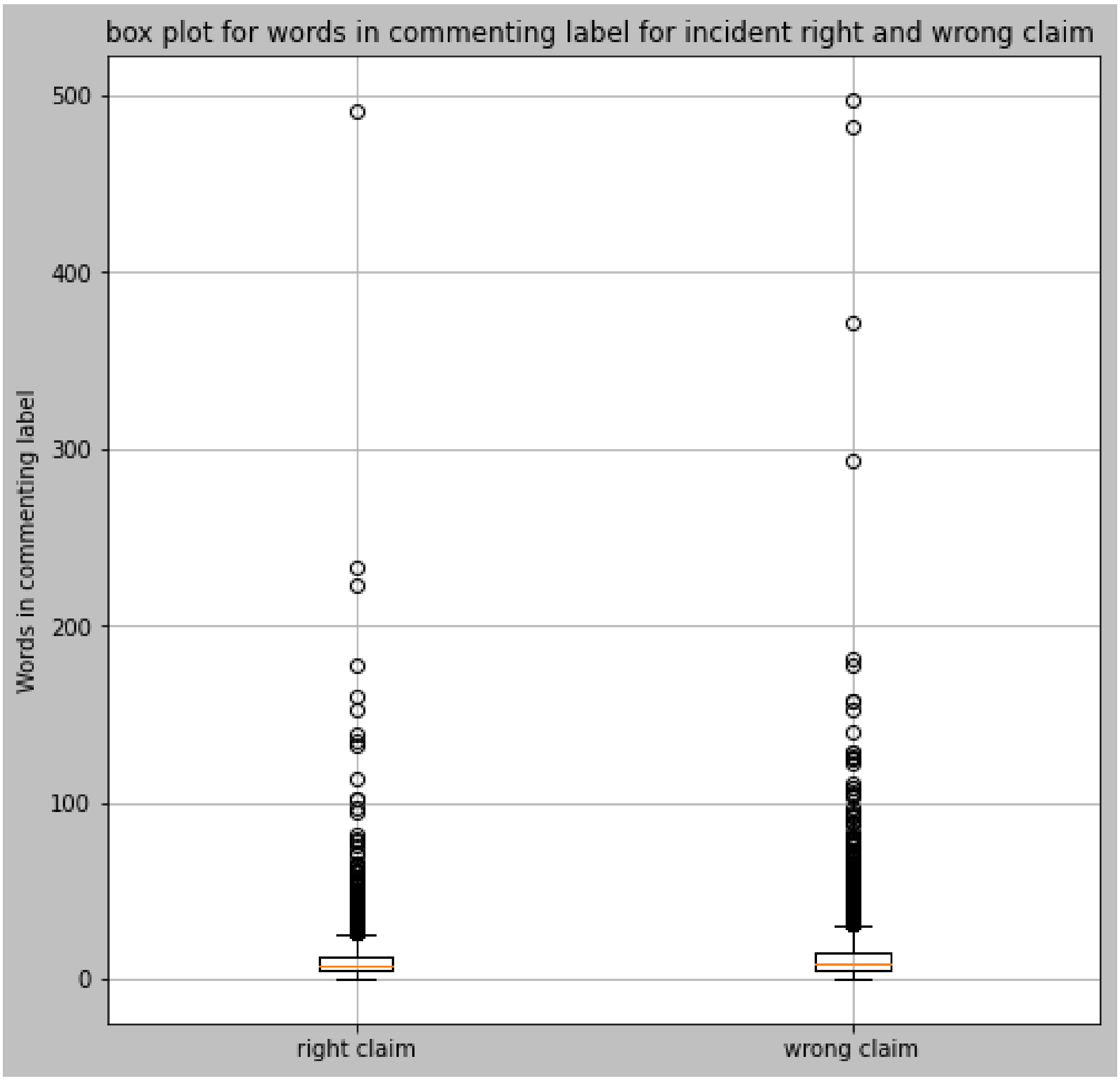
```
conclusion from above all three label with negative claim graph :
-----------------------------------------------------------------
commenting max lenght of words in each description neg : 498
ogling max lenght of words in each description neg     : 498
grouping max lenght of words in each description neg   : 491
commenting count of instances having same word counts (frequency) neg
ogling count of instances having same word counts (frequency) neg
grouping count of instances having same word counts (frequency) neg

commenting, ogling, grouping with negative claims have very similar d
```

1.2.3.4 BOX PLOT

1.2.3.4.1 BOX PLOT FOR LABEL COMMENTING

```
def box_plot(a,b, lab):
  fig = plt.figure(figsize=(8,8))
  fig.patch.set_facecolor('silver')
  plt.boxplot(np.array([a,b], dtype=object))
  plt.title(f'box plot for words in {lab} label for incident right and wr
  plt.xticks([1,2],('right claim','wrong claim'))
  plt.ylabel(f'Words in {lab} label')
  plt.grid()
  plt.show()
```

```
box_plot(commenting_claim_pos, commenting_claim_neg, 'commenting')
```




```
print('conclusion from above plot of right and wrong claim of commenting 
print('-'*75)
print(f'25 percentile of commenting label words of postive claim : {np.pe
print(f'median of commenting label words of postive claim        : {np.me
print(f'75 percentile of commenting label words of postive claim : {np.pe
print(' ')
print(f'commenting label postive claim iqr  : {np.percentile(commenting_c
print(f'commenting label negative claim iqr : {np.percentile(commenting_c
print(' ')
print(f'acc. theory commenting label positive claim outliers words after 
print(f'acc. theory commenting label negative claim outliers words after 
print(' ')
print(f'range of commenting label positive claim words   : {max(commentir
print(f'range of commenting label negative claim words   : {max((commenti
```

```
conclusion from above plot of right and wrong claim of commenting lab
---------------------------------------------------------------------
25 percentile of commenting label words of postive claim : 5.0, and n
median of commenting label words of postive claim        : 8.0, and n
75 percentile of commenting label words of postive claim : 13.0, and

commenting label postive claim iqr  : 8.0
commenting label negative claim iqr : 10.0

acc. theory commenting label positive claim outliers words after  : 1
acc. theory commenting label negative claim outliers words after  : 1

range of commenting label positive claim words   : 491
range of commenting label negative claim words   : 491
```

1.2.3.4.2 BOX PLOT FOR LABEL OGLING

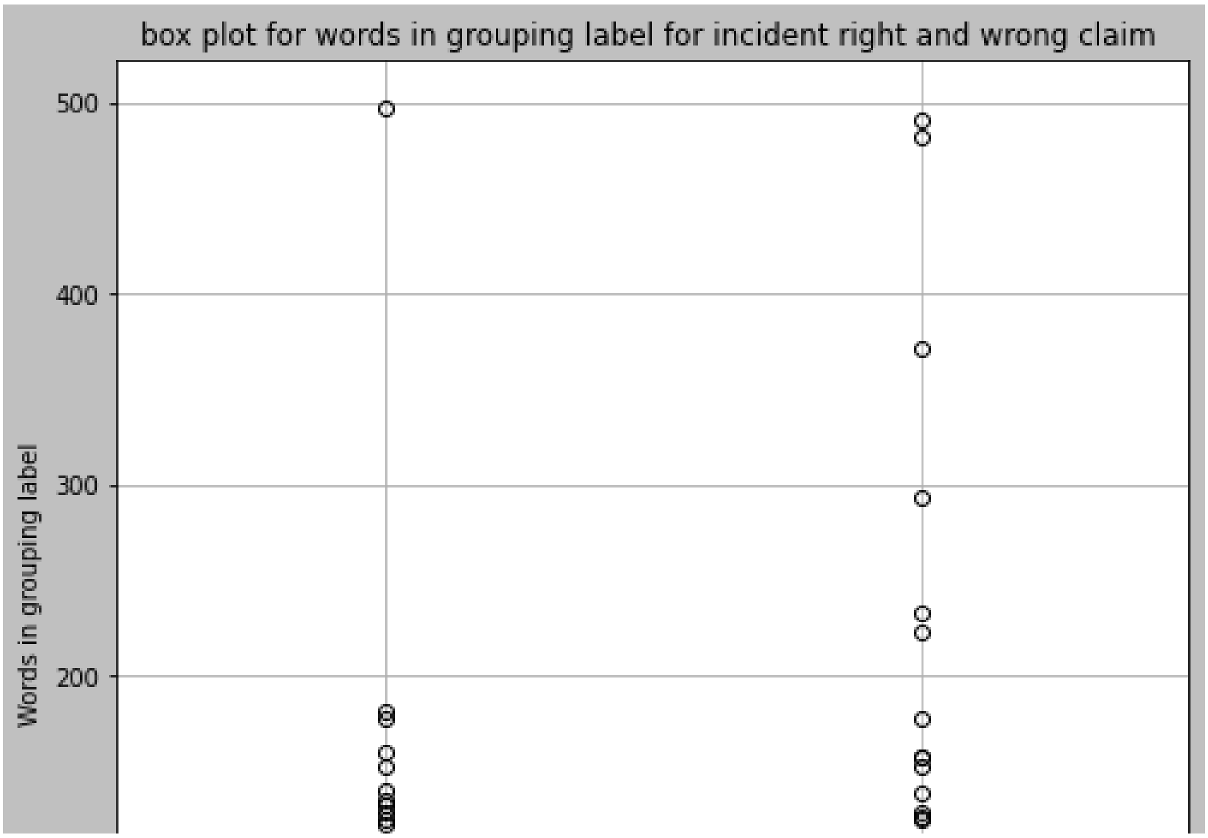
```
box_plot(ogling_claim_pos, ogling_claim_neg, 'ogling')
```


```
print('conclusion from above plot of right and wrong claim of ogling labe
print('-'*75)
print(f'25 percentile of ogling label words of postive claim : {np.percer
print(f'median of ogling label words of postive claim        : {np.mediar
print(f'75 percentile of ogling label words of postive claim : {np.percer
print(' ')
print(f'ogling label postive claim iqr  : {np.percentile(ogling_claim_pos
print(f'ogling label negative claim iqr : {np.percentile(ogling_claim_neg
print(' ')
print(f'acc. theory ogling label positive claim outliers words after  : {
print(f'acc. theory ogling label negative claim outliers words after  : {
print(' ')
print(f'range of ogling label positive claim words   : {max(ogling_claim_
print(f'range of ogling label negative claim words   : {max((ogling_clain
```

```
conclusion from above plot of right and wrong claim of ogling label :
---------------------------------------------------------------------------
25 percentile of ogling label words of postive claim : 5.0, and negat
median of ogling label words of postive claim        : 8.0, and negat
75 percentile of ogling label words of postive claim : 14.0, and nega

ogling label postive claim iqr  : 9.0
ogling label negative claim iqr : 9.0

acc. theory ogling label positive claim outliers words after  : 13.5
acc. theory ogling label negative claim outliers words after  : 13.5

range of ogling label positive claim words   : 223
range of ogling label negative claim words   : 223
```

1.2.3.4.3 BOX PLOT FOR LABEL GROUPING

```
box_plot(grouping_claim_pos, grouping_claim_neg, 'grouping')
```




```
print('conclusion from above plot of right and wrong claim of grouping la
print('-'*75)
print(f'25 percentile of grouping label words of postive claim : {np.perc
print(f'median of grouping label words of postive claim       : {np.medi
print(f'75 percentile of grouping label words of postive claim : {np.perc
print(' ')
print(f'grouping label postive claim iqr  : {np.percentile(grouping_claim
print(f'grouping label negative claim iqr : {np.percentile(grouping_claim
print(' ')
print(f'acc. theory grouping label positive claim outliers words after :
print(f'acc. theory grouping label negative claim outliers words after :
print(' ')
print(f'range of grouping label positive claim words   : {max(grouping_cl
print(f'range of grouping label negative claim words   : {max((grouping_c
```

```
conclusion from above plot of right and wrong claim of grouping label
---------------------------------------------------------------------------
25 percentile of grouping label words of postive claim : 5.0, and neg
median of grouping label words of postive claim        : 9.0, and neg
75 percentile of grouping label words of postive claim : 14.0, and ne

grouping label postive claim iqr  : 9.0
grouping label negative claim iqr : 9.0

acc. theory grouping label positive claim outliers words after : 13.5
acc. theory grouping label negative claim outliers words after : 13.5
```

```
range of grouping label positive claim words   : 497
range of grouping label negative claim words   : 498
```

1.2.3.4.4 INSPECTING MORE ON PERCENTILES

```
def percentile_range(frame,column,a,b,c):
  d_len  =  frame[column].apply(len)
  for i in range(a,b,c):
    print(f'{i} th percentile : {np.percentile(d_len,i)}')
```

```
import math

def percentile_float(frame, column, perc):
  d_len  =  frame[column].apply(len)
  size = len(d_len)
  for i in perc:
    print(f'{i} th percentile {sorted(d_len)[int(math.ceil(int(size * i)
```

```
percentile_range(combined_data, 'description', 0,110,10)
```

```
0 th percentile : 0.0
10 th percentile : 21.0
20 th percentile : 30.0
30 th percentile : 38.0
40 th percentile : 47.0
50 th percentile : 57.0
60 th percentile : 68.0
70 th percentile : 82.0
80 th percentile : 104.20000000000073
90 th percentile : 144.0
100 th percentile : 3992.0
```

```
percentile_range(combined_data, 'description', 90,101,1)
```

```
90 th percentile : 144.0
91 th percentile : 150.54000000000087
92 th percentile : 158.47999999999956
93 th percentile : 168.0
94 th percentile : 178.35999999999876
95 th percentile : 192.0
96 th percentile : 213.23999999999978
```

```
97 th percentile : 242.1800000000003
98 th percentile : 294.0
99 th percentile : 401.0599999999995
100 th percentile : 3992.0
```

```
perc = [99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9]
percentile_float(combined_data, 'description', perc)
```

```
99.1 th percentile 418
99.2 th percentile 439
99.3 th percentile 462
99.4 th percentile 517
99.5 th percentile 554
99.6 th percentile 653
99.7 th percentile 735
99.8 th percentile 880
99.9 th percentile 1223
```

```
perc = [99.91,99.92,99.93,99.94,99.95,99.96,99.97,99.98,99.99]
percentile_float(combined_data, 'description', perc)
```

```
99.91 th percentile 1239
99.92 th percentile 1262
99.93 th percentile 1481
99.94 th percentile 1591
99.95 th percentile 1849
99.96 th percentile 2695
99.97 th percentile 3213
99.98 th percentile 3546
99.99 th percentile 3992
```

```
def top_30(frame,column):

  '''takes frame : dataframe
     column      : text column
     returns     : top 30 len word counts'''

  kl = {}
  d_s =  frame[column].apply(len).values
  for i in range(len(d_s)):
    kl[i] = d_s[i]
  return sorted(kl.items(), key=lambda x: x[1], reverse=True)[:30]

top_30(combined_data, 'description')
```

```
[(3261, 3992),
```

```
 (5347, 3546),
 (239, 3213),
 (6056, 2695),
 (5654, 1849),
 (4124, 1591),
 (8392, 1481),
 (5459, 1262),
 (2616, 1239),
 (1177, 1223),
 (9074, 1109),
 (1087, 1032),
 (3862, 1011),
 (6444, 984),
 (6653, 960),
 (1156, 951),
 (1208, 920),
 (4986, 887),
 (5461, 880),
 (8653, 862),
 (1175, 842),
 (1034, 830),
 (2673, 817),
 (8422, 810),
 (3166, 801),
 (8774, 763),
 (7598, 749),
 (3779, 735),
 (7881, 713),
 (8934, 704)]
```

OBSERVATION

1. if required to select maximum len of words we would select nearly 800 to capture maximum info. and nearly truncate ~25 words which are greater than 800.

1.2.4 PCA ANALYSIS

1.2.4.1 PCA ANALYSIS FOR VARIOUS N COMPONENT VALUE

```
def tfidfvect_for_pca_plot(n, frame, column):

  '''takes frame : dataframe,
        column   : text data
```

```
        column   : text data
        n        : pca components '''

  tfidf_vect = TfidfVectorizer(stop_words=set(stopwords.words('english'))
  idf        = tfidf_vect.fit_transform(frame[column])
  pca        = PCA(n_components=n).fit(idf.todense())
  evr        = pca.explained_variance_ratio_
  datanD     = pca.transform(idf.todense())


  return datanD, idf, evr
```

```
d, i, e = tfidfvect_for_pca_plot(10, combined_data, 'description')
print(f'maximum variance {max(e)}')
```

```
maximum variance 0.011637706784529082
```

```
d, i, e = tfidfvect_for_pca_plot(100, combined_data, 'description')
print(f'maximum variance {max(e)}')
```

```
maximum variance 0.011637715682134446
```

```
d, i, e = tfidfvect_for_pca_plot(200, combined_data, 'description')
print(f'maximum variance {max(e)}')
```

```
maximum variance 0.01163771568213515
```

```
d, i, e = tfidfvect_for_pca_plot(500, combined_data, 'description')
print(f'maximum variance {max(e)}')
```

```
maximum variance 0.011637715682135135
```

```
d, i, e = tfidfvect_for_pca_plot(1000, combined_data, 'description')
print(f'maximum variance {max(e)}')
```

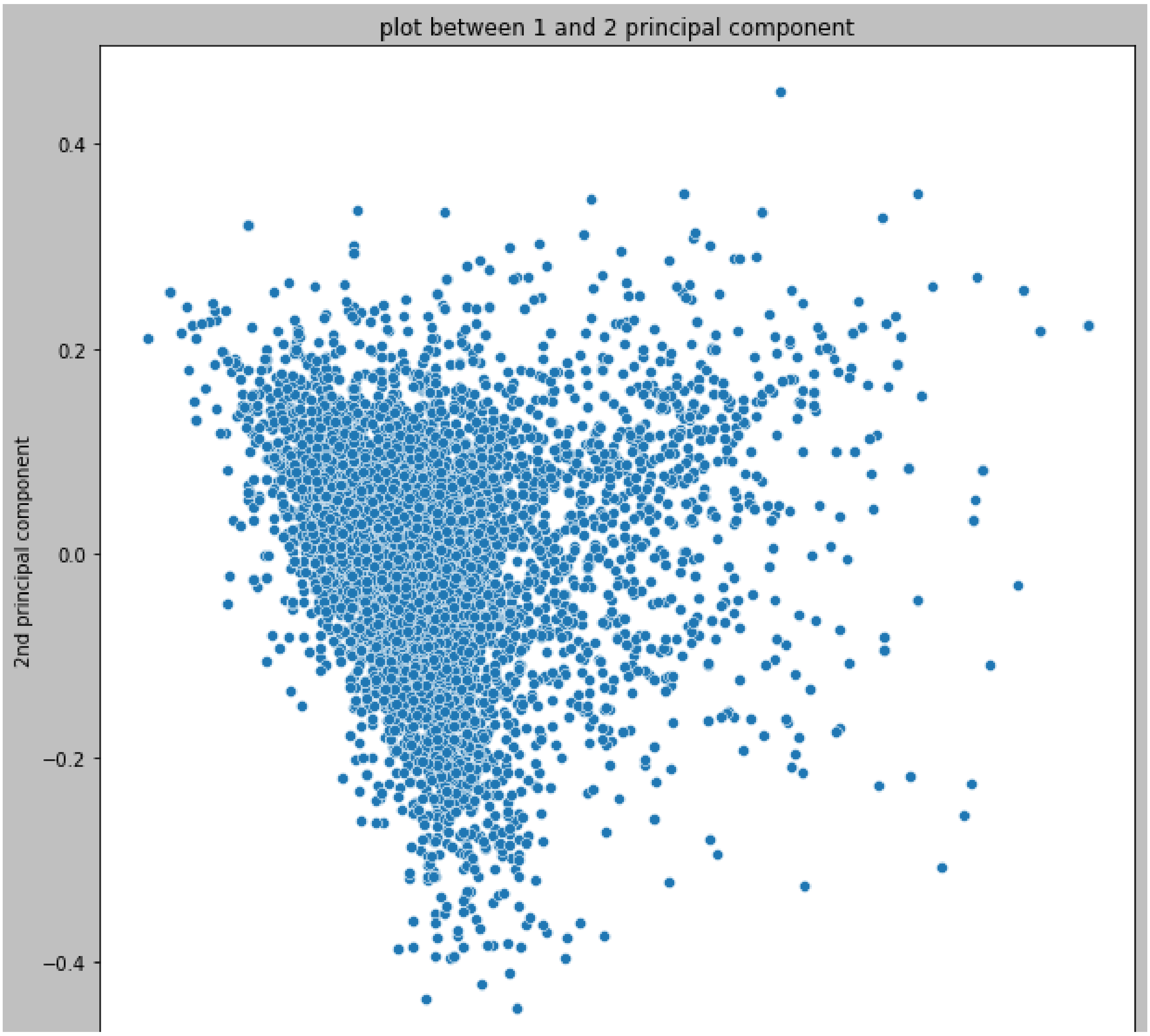
```
maximum variance 0.011637715682135208
```

OBSERVATION

1. nothing much improvement in maximum variance after trying various principal n component values.

1.2.4.2 PLOT FOR PCA ANALYSIS FOR 2 PRINCAIPAL COMPONENT VALUE

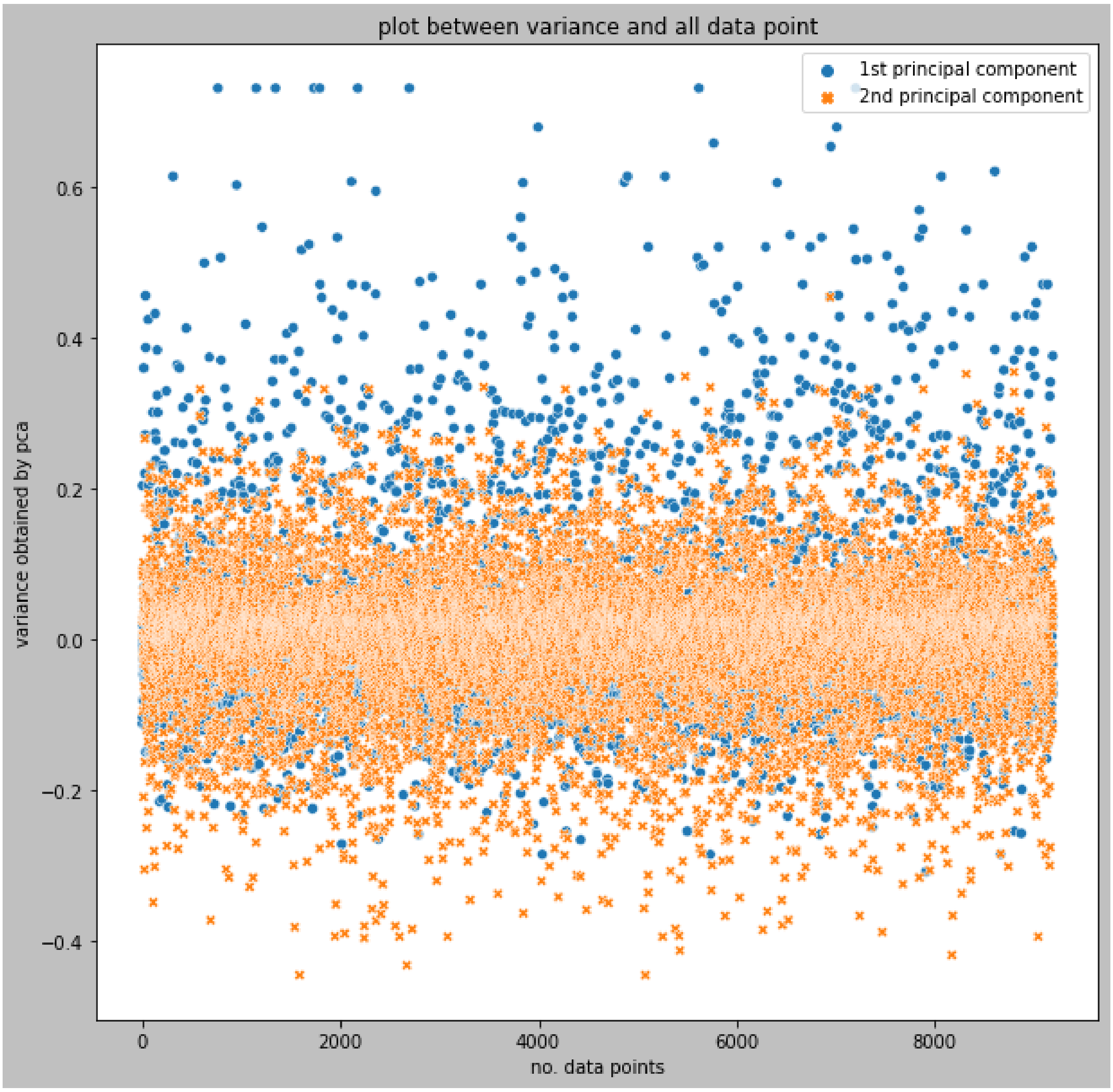
```
def pca_plot(plot, n, frame, column):

  '''frame      : dataframe
     colum      : text data
     plot       : plots to display
     n          : pca components
  '''

  data, _, evr = tfidfvect_for_pca_plot(n, frame, column)
  df = pd.DataFrame(data, columns=['1st principal component', '2nd princi
  X = df['1st principal component']
  Y = df['2nd principal component']

  if plot == 1:
    fig = plt.figure(figsize=(10,10))
    fig.patch.set_facecolor('silver')
    sns.scatterplot(data=df, x='1st principal component', y='2nd principa
    plt.title('plot between 1 and 2 principal component')

  if plot == 2:
    fig = plt.figure(figsize=(10,10))
    fig.patch.set_facecolor('silver')
    sns.scatterplot(data=df)
    plt.title('plot between variance and all data point')
    plt.xlabel('no. data points')
    plt.ylabel('variance obtained by pca')

  plt.show()
  return evr


pca_plot(1, 2, combined_data, 'description')
```



OBSERVATION

1. from above plot maximum variance of 1st principal component lies between [-0.2, 0.4].
2. from above plot maximum variance of 2nd principal component lies between [-0.3, 0.2].
3. variance explained by first component is 1.16%, and by second component is 0.9%.

```
pca_plot(2, 2, combined_data, 'description')
```



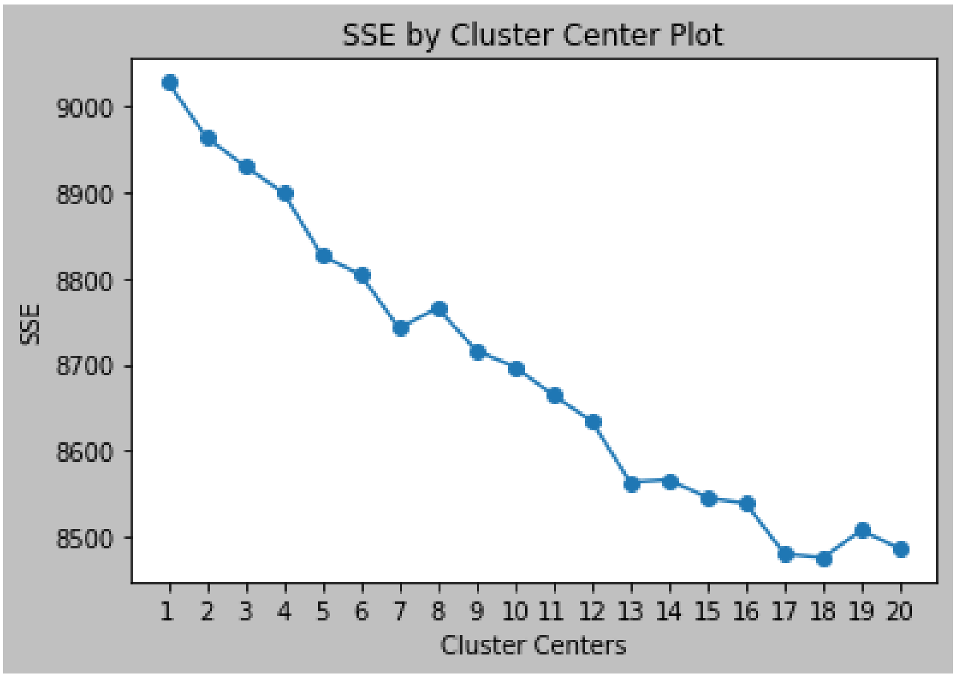

```
array([0.01163748, 0.00976623])
```

OBSERVATION

1. from above plot maximum variance of 1st principal component lies between [-0.2, 0.4].
2. from above plot maximum variance of 2nd principal component lies between [-0.3, 0.2].
3. variance explained by first component is 1.16%, and by second component is 0.9%.

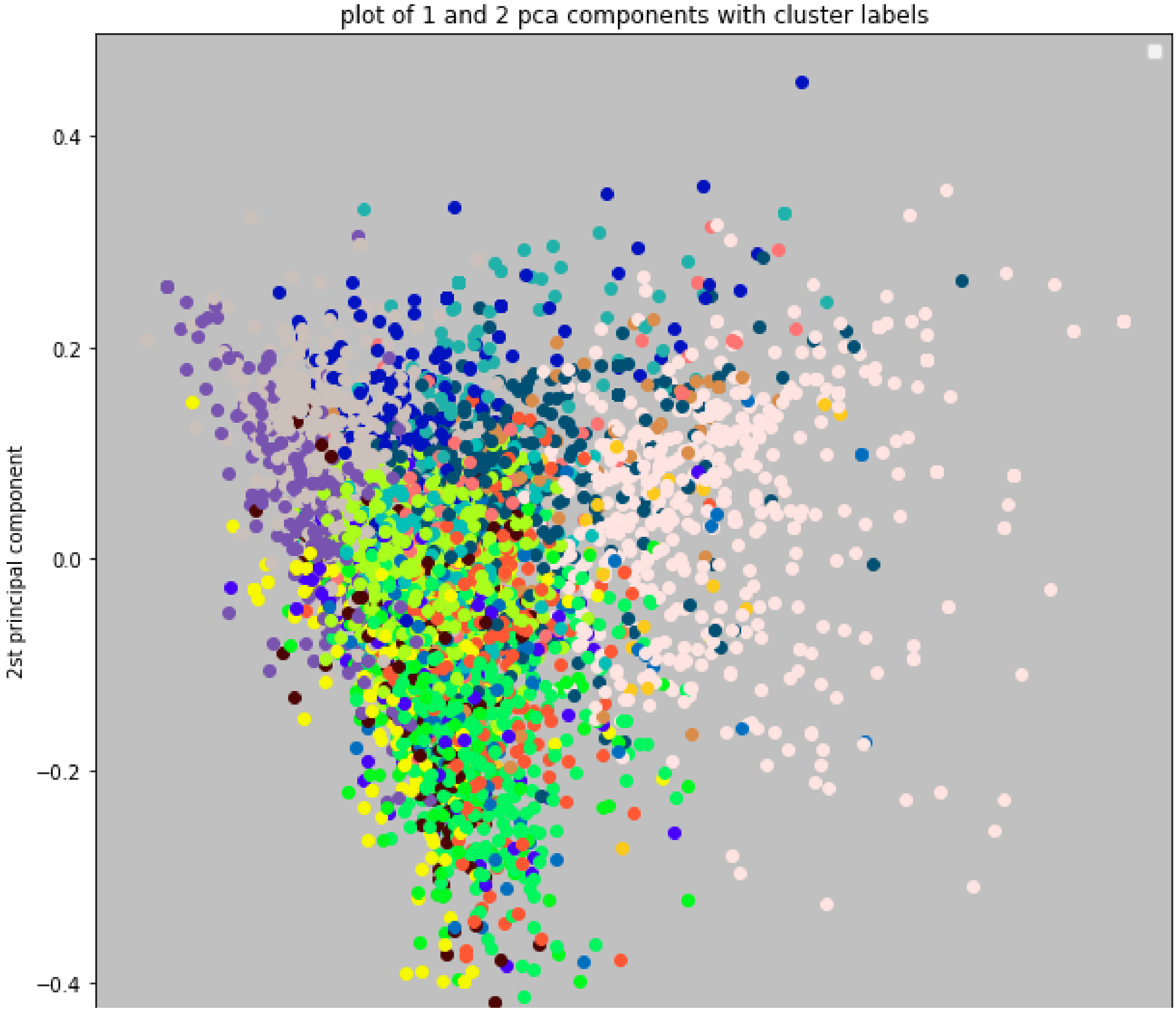
1.2.4.3 PCA ANALYSIS FOR VARIOUS PRINCIPAL COMPONENT VALUE WITH CLUSTERING

```
#https://www.kaggle.com/jbencina/clustering-documents-with-tfidf-and-kmea

def find_optimal_clusters(data, rng):

  '''takes data : idf data,
            rng : max range of cluster we want to check,
        returns : cluster which have minimum sse, graphically'''

  iters = range(1, rng, 1)
  sse = []
  for k in iters:
    clus = MiniBatchKMeans(n_clusters=k, batch_size=180, random_state=42)
    ##interia_ reponsible for sum of squared distance to its NN
    sse.append(clus)

  f, ax = plt.subplots(1, 1)
  f.patch.set_facecolor('silver')
  ax.plot(iters, sse, marker='o')
  ax.set_xlabel('Cluster Centers')
  ax.set_xticks(iters)
  ax.set_xticklabels(iters)
  ax.set_ylabel('SSE')
  ax.set_title('SSE by Cluster Center Plot')
plt.show()



_, idf, _ = tfidfvect_for_pca_plot(2, combined_data, 'description')
find_optimal_clusters(idf, 21)
```



```
def plot_cluster_label(n, frame, column):

  '''takes frame : dataframe,
      column   : text data
      n        : pca components
      return   : labeled tranform pca visulization'''

  data, idf, _ = tfidfvect_for_pca_plot(n, frame, column)
  clusters     = MiniBatchKMeans(n_clusters=18, batch_size=180, random_st

  labels_color_map = {
    0: '#20b2aa', 1: '#ff7373', 2: '#ffe4e1', 3: '#005073', 4: '#4d0404',
    5: '#ccc0ba', 6: '#4700f9', 7: '#f6f900', 8: '#00f91d', 9: '#da8c49',
    13: '#00F65D', 14: '#00BFB6', 15: '#006EBF', 16: '#0011BF', 17: '#785

  fig, ax = plt.subplots(figsize=(10,10))
  ax.patch.set_facecolor('silver')

  for index, instance in enumerate(data):

    pca_comp_1, pca_comp_2 = data[index]
    color = labels_color_map[clusters[index]]
    ax.scatter(pca_comp_1, pca_comp_2, c=color)
    color = labels_color_map[clusters[index]]

  ax.legend()
  plt.title('plot of 1 and 2 pca components with cluster labels')
  plt.xlabel('1st principal component')
  plt.ylabel('2st principal component')
  #ax.legend()
  plt.show()
  return clusters


plot_cluster_label(2, combined_data, 'description')
```

```
No handles with labels found to put in legend.
```



OBSERVATION

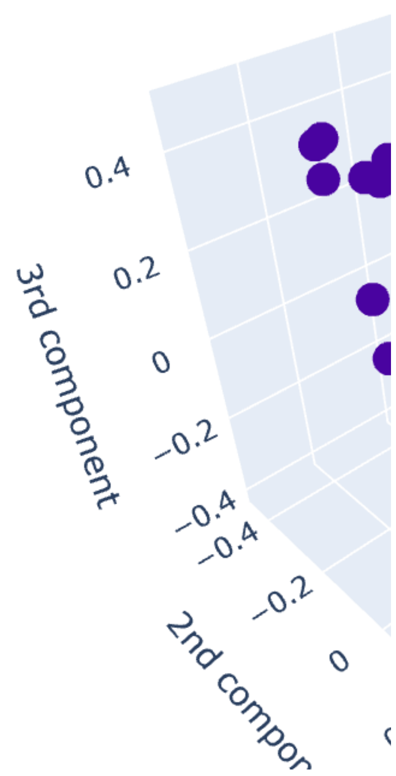
1. it is difficult to use linear classifier for linearly seperating pca transformed data points, it is because pca is projection of datapoints on its high variance axis but from above plot we classify purple,cyan,blue points with some misclassificaton, to check clsssifiaction among points we try for 3d plot, if we want to classify this transformed pca data we can use non linarity based models such as NN.

1.2.4.4 PCA ANALYSIS FOR 3 PRINCIPAL COMPONENT VALUE WITH 3D PLOT

```
def threed_cluster_plot(n, frame, column):
  data, idf, _ = tfidfvect_for_pca_plot(n,frame,column)
  clusters     = MiniBatchKMeans(n_clusters=18, batch_size=180, random_st
  df           = pd.DataFrame(data, columns=['1st component', '2nd compor
```

```
    X = df['1st component']
    Y = df['2nd component']
    Z = df['3rd component']


    fig = px.scatter_3d(df, x= X,y =Y, z=Z, color=clusters)
    fig.update_layout(paper_bgcolor="black")
    fig.update_layout(title_text='3d pca plot')
    fig.show()


threed_cluster_plot(3, combined_data, 'description')
```

3d pca plot



OBSERVATION

1. it is difficult to use linear classifier for linearly seperating pca transformed data points, it is because pca is projection of datapoints on its high variance axis, if we want to classify this transformed pca data we can use non linarity based models such as NN.

1.3 UTILITY FUNCTION

```
def check(text, vocab):

  '''takes text : text column row
          vocab : list of words of all grams
        returns : 1 if word from text present in vocab else 0'''

  w_t = word_tokenize(text)
  for i in w_t:
    if i in vocab:
      return 1
    else:
      return 0


def ngram_check(text, n, vocab):

  '''takes text : text column row
          vocab : list of words of all grams
              n : number of n grams to be considered
        returns : sum/count if word from text present in vocab else 0'''

  n_g = ngrams(word_tokenize(text),n)
  cnt = 0
  try:
    p = [' '.join(i) for i in n_g]
    for k in p:
      if k in vocab:
        cnt +=1
  except:
    pass
  return cnt


def idf_check(x, vocab):
```

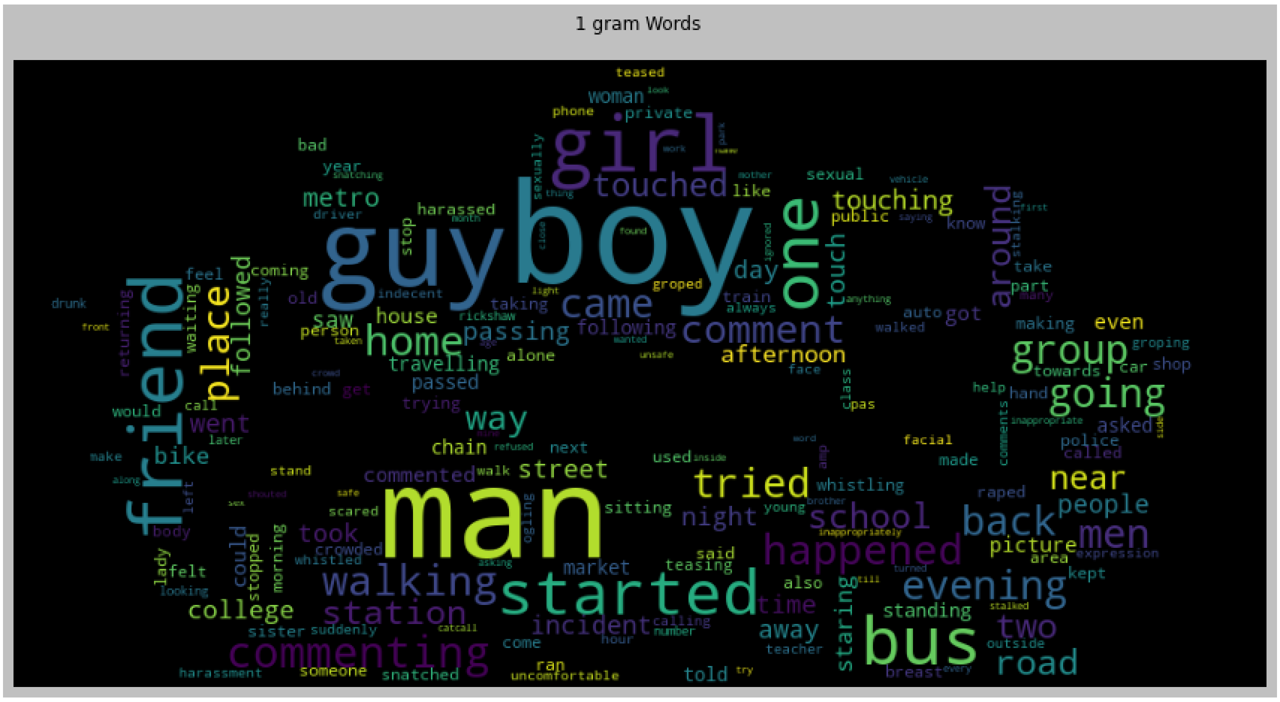
```
    takes x  : text column row
       vocab : list of words of idf
     returns : 1 if word from x present in vocab else 0'''

  w_t = word_tokenize(x)
  for i in w_t:
    if i in vocab:
      return 1
    else:
      return 0
```

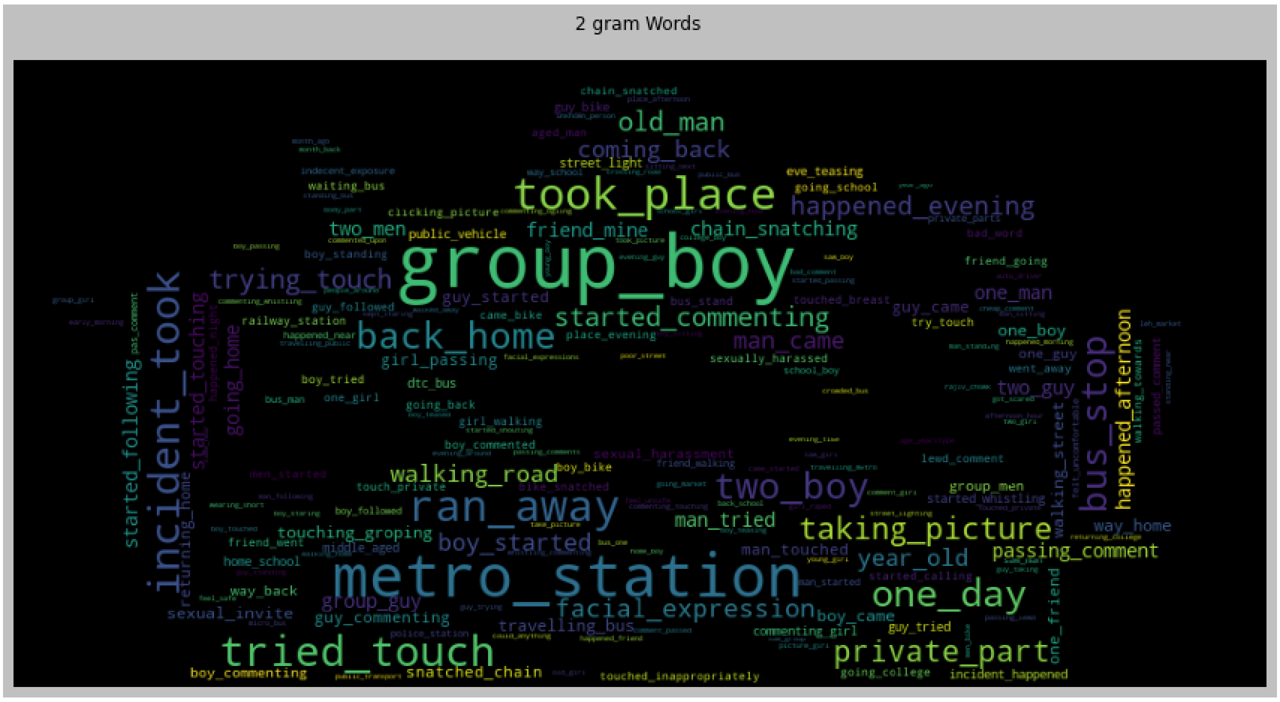
2. FEATURE ENGINEERING AND NGRAM VIZUALIZATION

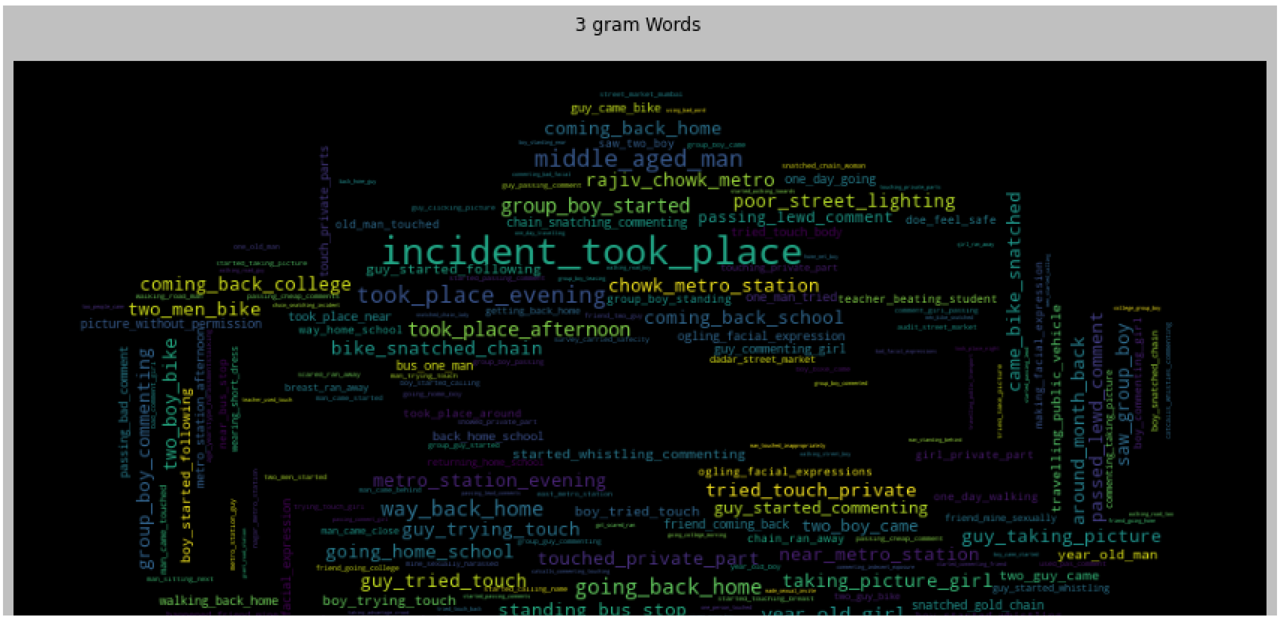
2.1 PLOTTING UNI, BI, TRI, FOUR GRAM WORDCLOUD

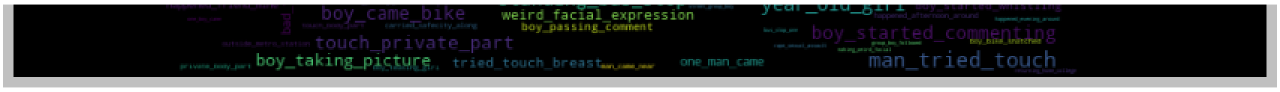
```
#https://stackoverflow.com/questions/49537474/wordcloud-of-bigram-using-p
#https://www.voicesofyouth.org/sites/voy/files/images/2019-02/metoo_0.jpg

from PIL import Image
from wordcloud import ImageColorGenerator

stop_wordss = list(set(stopwords.words('english')))
def n_gram_cloud(data, gram, top_feat):

  '''takes data : text column
           gram : no. of grams we want
       top_feat : no. of feature to consider for construtuing gram
       this calculates 1,2,3,4 grams'''

  uni  = []
  bi   = []
  tri  = []
  four = []

  for i in gram:
    vect = CountVectorizer(ngram_range=(i,i), stop_words=stop_wordss)
    bow  = vect.fit_transform(data)
    word_count = bow.sum(axis=0)
    word_frq   = [(word, word_count[0, k]) for word, k in vect.vocabulary
```

```
    word_frq   = sorted(word_frq, key = lambda x : x[1], reverse=True)
    word_frq   = word_frq[:top_feat]



    if i == 1:
      for j in range(len(word_frq)):
        uni.append(word_frq[j][0])
    elif i == 2:
      for j in range(len(word_frq)):
        bi.append(word_frq[j][0])
    elif i == 3:
      for j in range(len(word_frq)):
        tri.append(word_frq[j][0])
    else:
      for j in range(len(word_frq)):
        four.append(word_frq[j][0])

    n_word = {i[0].replace(' ', '_') : i[1] for i in word_frq}

    fig = plt.figure(figsize=(15,15))
    fig.patch.set_facecolor('silver')

    mt = np.array(Image.open('/content/gdrive/MyDrive/cs1/mt_4.jpg'))

    wordcloud = WordCloud(background_color = 'black', width = 600, height
                          stopwords = stopwords, contour_width=2).generat




    plt.imshow(wordcloud)
    plt.axis('off')
    plt.title(f'{i} gram Words \n')
    plt.show()


    print(' ')

  return uni, bi, tri, four


uni, bi, tri, four = n_gram_cloud(combined_data["description"], [1,2,3,4]
```


1 gram Words
man
boy
guy
girl
friend
started
bus
one
touched
came
comment
home
place
group
going
walking
commenting
happened
evening
school
back
tried
men
station
incident
road
two
near
way
street
people
touching
metro
around
day
staring
college
touch
afternoon
saw
house
passing
night


2 gram Words
group_boy
metro_station
tried_touch
incident_took
took_place
back_home
ran_away
two_boy
taking_picture
one_day
private_part
started_commenting
facial_expression
trying_touch
bus_stop
walking_road
boy_started
old_man
coming_back
happened_evening
chain_snatching
friend_mine
man_came
year_old
passing_comment
happened_afternoon


3 gram Words
incident_took_place
took_place_evening
took_place_afternoon
middle_aged_man
coming_back_home
rajiv_chowk_metro
chowk_metro_station
coming_back_college
two_men_bike
bike_snatched_chain
group_boy_started
poor_street_lighting
passing_lewd_comment
coming_back_school
way_back_home
guy_trying_touch
going_home_school
touched_private_part
guy_tried_touch
going_back_home
taking_picture_girl
tried_touch_private
metro_station_evening
guy_taking_picture
near_metro_station


boy_came_bike
weird_facial_expression
boy_passing_comment
boy_started_commenting
touch_private_part
boy_taking_picture
tried_touch_breast
one_man_came
man_tried_touch

4 gram Words


place_rajiv_chowk_metro
boy_came_bike_snatched
rajiv_chowk_metro_station
taken_dadar_street_market
tried_touch_private_part
incident_took_place_evening
safecity_along_safecity_audit
incident_took_place_near
incident_took_place_rajiv
touching_girl_private_part
two_boy_came_bike
safecity_audit_street_market
making_weird_facial_expression
along_safecity_audit_street
report_taken_dadar_street
two_guy_came_bike
nagar_east_metro_station
incident_took_place_night
tried_touch_private_parts
boy_bike_snatched_chain
incident_took_place_around
came_bike_snatched_chain
incident_took_place_afternoon
guy_taking_picture_girl
going_back_home_school
group_boy_started_commenting
survey_carried_safecity_along
one_man_tried_touch
friend_mine_sexually_harassed
catcall_commenting_bad_facial
audit_street_market_mumbai
group_boy_passing_comment

OBSERVATIONS

1. as we move towards more bigram we are getting the sense of place, time, condition, in which incidence took place.
2. we are getting much more context around harassment incident such as "way_back_home" tri-gram suggest that the incident may be took when person wayback to home, "near_metro_station", "incident_took_place_evening" also give us the context around which sexual harrasment is prevalent.

```
short_vocab = list(set(uni+bi+tri+four))
```

```
def voc_for_present(vo):
  v = [i.split() for i in vo]
  K = [word for lis in v for word in lis]

  return list(set(K))

vocab_present_or_not = voc_for_present(short_vocab)
```

```
def idf_rare_feq_word(frame, text_col, feat):

  '''takes frame : dataframe,
     text column : text column,
     feature : int(max. which we want to display)
     returns : frequent, rare words based on idf value
  '''

  tfidf_vect = TfidfVectorizer(stop_words=set(stopwords.words('english'))
  idf        = tfidf_vect.fit_transform(frame[text_col])
  feat_names = tfidf_vect.get_feature_names()
  idf_value  = tfidf_vect.idf_
  df         = pd.DataFrame(list(zip(feat_names, idf_value)), columns=['w
```

```
  df.sort_values("idf_value", axis = 0, ascending = False, inplace = True

  print('| rare words with idf value |')
  print('-'*29)
  print(df.head(5))
  rare = df['word'][:feat].tolist()

  print('-'*29)
  print('| freq words with idf value |')
  print('-'*29)
  df.sort_values("idf_value", axis = 0, ascending = True, inplace = True,
  print(df.head(5))
  frequent     = df['word'][:feat].tolist()

  return frequent, rare


idf_low, idf_high = idf_rare_feq_word(combined_data, 'description', 1000)
```

```
| rare words with idf value |
-----------------------------
        word  idf_value
0     laoded   9.433377
1    reasons   9.433377
2      hallo   9.433377
3     recess   9.433377
4  halfasleep   9.433377
-----------------------------
| freq words with idf value |
-----------------------------
     word  idf_value
0     boy   2.710747
1     guy   2.851352
2     man   2.883011
3    girl   3.035614
4  friend   3.099209
```

2.2 FINAL DATAFRAME AFTER FE

```
def feature_engineering(data, column):

  '''takes data : dataframe[column]'''

  data['p_or_a'] = data[column].map(lambda x : check(x, vocab_present_or_
  data['4gm']    = data[column].map(lambda x : ngram_check(x,4, short_voc
  data['3gm']    = data[column].map(lambda x : ngram_check(x,3, short_voc
```

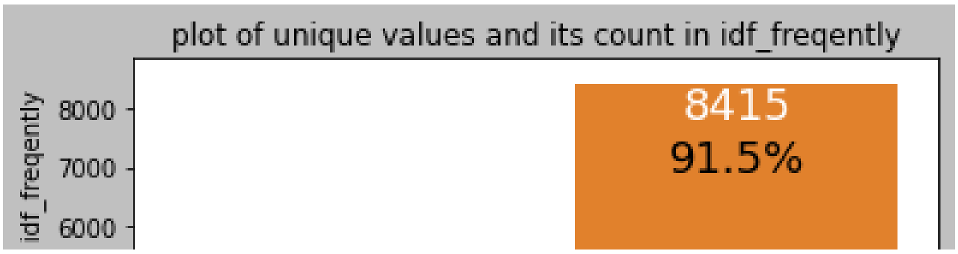
```
    data['3gm']    = data[column].map(lambda x : ngram_check(x,3, short_voc
    data['2gm']    = data[column].map(lambda x : ngram_check(x,2, short_voc
    data['1gm']    = data[column].map(lambda x : ngram_check(x,1, short_voc

    data['idf_freqently'] = data[column].map(lambda x : idf_check(x, idf_lo
    ##for high idf indicates less frequent word, or rarely occuring
    ##for low idf indicates more frequent word , or fequently occuring
    data['idf_rare'] = data[column].map(lambda x : idf_check(x, idf_high))

    data['1_2_3_4gm'] = data['4gm'] + data['3gm'] + data['2gm'] + data['1gm
    data.drop(['4gm','3gm', '2gm', '1gm'], axis=1, inplace=True)

    data['description_len'] = data[column].astype(str).apply(len)
    data['word_count'] = data[column].apply(lambda x: len(str(x).split()))
    data['word_density'] = data['description_len'] / (data['word_count']+1)

    if data.isnull().any().sum() == 0:
      pass
    else:
      data.dropna(inplace=True)

    data[['p_or_a', 'idf_freqently',  'idf_rare']].astype(int)
    data.reset_index(inplace=True, drop=True)

    return data


combined_data_fe = feature_engineering(combined_data, 'description')


combined_data_fe
```

	description	commenting	ogling	grouping	p_or_a	idf_freqe
0	walking along crowded street holding mum hand ...	0	0	1	1.0	
1	incident took place evening metro two guy star...	0	1	0	1.0	
2	incident happened inside	0	0	0	1.0	

2.3.1 VIZUALIZING IDF_FREQUENTLY COLUMN OF FINAL DATAFRAME AFTER FE

```
def after_fe_vizualization(frame, column):
  fig = plt.figure()
  fig.patch.set_facecolor('silver')
  total = frame[column].count()
  ax = sns.countplot(x=frame[column], data=frame)
  ax.set_title("count data")
  for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x()+0.4, p.get_height()+1.4),
    percentage = '{:.1f}%'.format(100 * p.get_height()/total)
    ax.annotate(percentage, (p.get_x()+0.4, p.get_height()-1500), ha='cer

  plt.title(f'plot of unique values and its count in {column}')
  plt.xlabel(f'unique values in {column}')
  plt.ylabel(f'count of unique values in {column}')
```

```
after_fe_vizualization(combined_data_fe, 'idf_freqently')
```

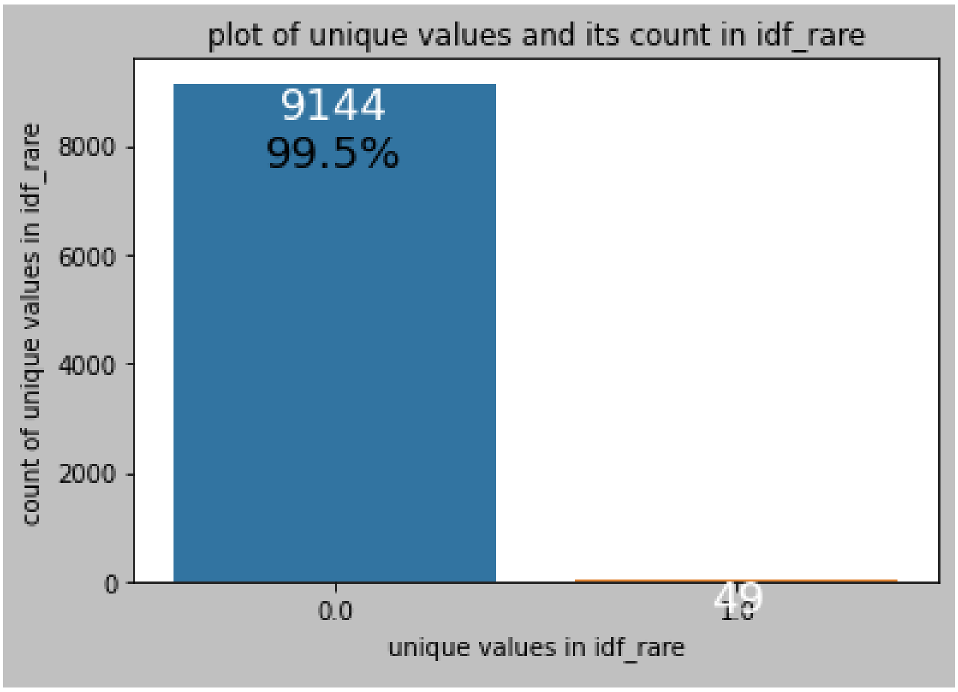


OBSERVATION

1. in final data idf_frequently have 91.5% of 1 of all the datapoints, which clearly indicated we have correctly taken high frequecy words (low_idf score) for frequently occuring words.

2.3.2 VIZUALIZING IDF_RARE COLUMN OF FINAL DATAFRAME AFTER FE

```
after_fe_vizualization(combined_data_fe, 'idf_rare')
```



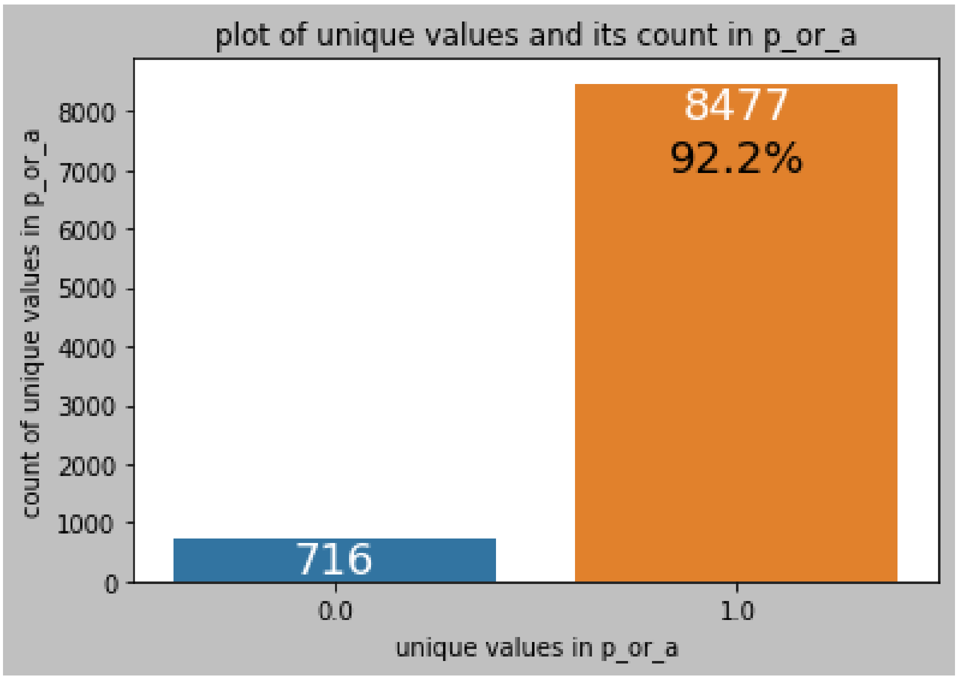
OBSERVATION

1. in final data idf_rare have 99.5% of 0 of all the datapoints, which clearly indicated we have correctly taken low frequency words (high_idf score) for rarely occuring words.

2.3.3 VIZUALIZING P_OR_A COLUMN OF FINAL DATAFRAME AFTER FE

WORDS FROM VOCAB OF UNI, BI, TRI, FOUR GRAM PRESENT (P) OR NOT (A)

```
after_fe_vizualization(combined_data_fe, 'p_or_a')
```



OBSERVATION

1. in final data p_or_a(present or absent) have 92.2% of 1 of all the datapoints, which is quite good as its corpus contains 4000 words from uni, bi, tri, four grams 1000 from each, it makes good proportion in overall data.

```
import pickle
#pickle.dump((combined_data_fe), open('/content/gdrive/MyDrive/cs1/combir
#combined_data_fe = pickle.load(open('/content/gdrive/MyDrive/cs1/combine
```

3. METRIC

1. from section 1.2.2 we can see there is lot of imbalance in label data, so i would consider micro f1 score as primary metric, as it captures tp, fn, fp, of data so tells the details of each label in dataset, apart from that i will also use exact match ratio, hamming loss, micro-precision and micro-recall as secondary metric.

REFERENCES

1. geeksforgeeks
2. barplot percentage
3. ngram
4. it is groping actully but i have used grouping.

✓ 0s completed at 03:46