Final-Code-part-1

December 20, 2022

Ref from [1]

[1]:
```
import pandas as pd
import swifter
import warnings
warnings.filterwarnings('ignore')

df=pd.read_csv('../Project/Reviews.csv')

df=df.drop(['Id', 'ProfileName', 'Time'], axis=1)
df=df.dropna()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 568427 entries, 0 to 568453
Data columns (total 7 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   ProductId               568427 non-null  object
 1   UserId                  568427 non-null  object
 2   HelpfulnessNumerator    568427 non-null  int64
 3   HelpfulnessDenominator  568427 non-null  int64
 4   Score                   568427 non-null  int64
 5   Summary                 568427 non-null  object
 6   Text                    568427 non-null  object
dtypes: int64(3), object(4)
memory usage: 34.7+ MB
```

[2]:
```
df.head()
```

[2]:

	ProductId	UserId	HelpfulnessNumerator	HelpfulnessDenominator	Score	Summary	\
0	B001E4KFG0	A3SGXH7AUHU8GW	1	1			
1	B00813GRG4	A1D87F6ZCVE5NK	0	0			
2	B000LQOCH0	ABXLMWJIXXAIN	1	1			
3	B000UA0QIQ	A395BORC6FGVXV	3	3			
4	B006K2ZZ7K	A1UQRSCLF8GW1T	0	0			

```
0      5  Good Quality Dog Food
1      1      Not as Advertised
2      4  "Delight" says it all
3      2         Cough Medicine
4      5            Great taffy

                                                Text
0  I have bought several of the Vitality canned d…
1  Product arrived labeled as Jumbo Salted Peanut…
2  This is a confection that has been around a fe…
3  If you are looking for the secret ingredient i…
4  Great taffy at a great price.  There was a wid…
```

[3]:
```
textColumns = df.select_dtypes(exclude="number").columns
print(textColumns)
```

```
Index(['ProductId', 'UserId', 'Summary', 'Text'], dtype='object')
```

[4]:
```
visualization_df = df.copy()
```

For understanding the data better we have added few features as text length, word count, and count of non-alphanumerics in each feature

[5]:
```
for column in textColumns:
    visualization_df[column + '-textLen'] = visualization_df[column].map(str).
 ↪apply(len)
    visualization_df[column + '-wordCount'] = [len(str(row).split()) for row in␣
 ↪visualization_df[column].tolist()]
    visualization_df[column + '-alphaNumerics'] = visualization_df[column].str.
 ↪findall(r'[^a-zA-Z0-9 ]').str.len()
```

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

[6]:
```
     ProductId          UserId  HelpfulnessNumerator  HelpfulnessDenominator  \
0  B001E4KFG0  A3SGXH7AUHU8GW                     1                       1
1  B00813GRG4  A1D87F6ZCVE5NK                     0                       0
2  B000LQOCH0   ABXLMWJIXXAIN                     1                       1
3  B000UA0QIQ  A395BORC6FGVXV                     3                       3
4  B006K2ZZ7K  A1UQRSCLF8GW1T                     0                       0

   Score                Summary  \
0      5  Good Quality Dog Food
1      1      Not as Advertised
2      4  "Delight" says it all
3      2         Cough Medicine
4      5            Great taffy
```

```
                                                Text  ProductId-textLen  \
0  I have bought several of the Vitality canned d…                  10
1  Product arrived labeled as Jumbo Salted Peanut…                  10
2  This is a confection that has been around a fe…                  10
3  If you are looking for the secret ingredient i…                  10
4  Great taffy at a great price.  There was a wid…                  10

   ProductId-wordCount  ProductId-alphaNumerics  UserId-textLen  \
0                    1                        0              14
1                    1                        0              14
2                    1                        0              13
3                    1                        0              14
4                    1                        0              14

   UserId-wordCount  UserId-alphaNumerics  Summary-textLen  Summary-wordCount  \
0                 1                     0               21                  4
1                 1                     0               17                  3
2                 1                     0               21                  4
3                 1                     0               14                  2
4                 1                     0               11                  2

   Summary-alphaNumerics  Text-textLen  Text-wordCount  Text-alphaNumerics
0                      0           263              48                   3
1                      0           190              31                   7
2                      2           509              94                  18
3                      0           219              41                   5
4                      0           140              27                   5
```

0.0.1 Continuous Features Report

Continuous features report includes: 1. Count 2. Miss Percentage 3. Cardinality - num of distinct values for a feature 4. Min 5. 1st quartile 6. Mean 7. Median 8. 3rd quartile 9. Max 10. Standard Deviation

Pandas provides a function for generating data quality reports however it doesn't include all the statistics. [1]

[7]:
```
import pandas as pd
import warnings

def build_continuous_features_report(data_df):

    """Build tabular report for continuous features"""

    stats = {
        "Count": len,
        "Card.": lambda df: df.nunique(),
        "Min": lambda df: df.min(),
```

```
        "1st Qrt.": lambda df: df.quantile(0.25),
        "Mean": lambda df: df.mean(),
        "Median": lambda df: df.median(),
        "3rd Qrt": lambda df: df.quantile(0.75),
        "Max": lambda df: df.max(),
        "Std. Dev.": lambda df: df.std(),
    }

    contin_feat_names = data_df.select_dtypes("number").columns
    continuous_data_df = data_df[contin_feat_names]

    report_df = pd.DataFrame(index=contin_feat_names, columns=stats.keys())

    for stat_name, fn in stats.items():
        # NOTE: ignore warnings for empty features
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=RuntimeWarning)
            report_df[stat_name] = fn(continuous_data_df)

    return report_df


print(build_continuous_features_report(visualization_df))
```

```
                          Count  Card.  Min  1st Qrt.        Mean  Median  \
HelpfulnessNumerator     568427    231    0       0.0    1.743855     0.0
HelpfulnessDenominator   568427    234    0       0.0    2.227859     1.0
Score                    568427      5    1       4.0    4.183299     5.0
ProductId-textLen        568427      1   10      10.0   10.000000    10.0
ProductId-wordCount      568427      1    1       1.0    1.000000     1.0
ProductId-alphaNumerics  568427      1    0       0.0    0.000000     0.0
UserId-textLen           568427     10   10      13.0   13.740822    14.0
UserId-wordCount         568427      1    1       1.0    1.000000     1.0
UserId-alphaNumerics     568427      2    0       0.0    0.002505     0.0
Summary-textLen          568427    128    1      13.0   23.446858    20.0
Summary-wordCount        568427     32    1       2.0    4.113297     4.0
Summary-alphaNumerics    568427     49    0       0.0    0.902338     0.0
Text-textLen             568427   3828   12     179.0  436.236197   302.0
Text-wordCount           568427    998    3      33.0   80.266458    56.0
Text-alphaNumerics       568427    457    0       5.0   17.436862    10.0

                         3rd Qrt    Max   Std. Dev.
HelpfulnessNumerator         2.0    866    7.636692
HelpfulnessDenominator       2.0    923    8.288679
Score                        5.0      5    1.310385
ProductId-textLen           10.0     10    0.000000
ProductId-wordCount          1.0      1    0.000000
```

```
ProductId-alphaNumerics      0.0      0    0.000000
UserId-textLen              14.0     21    0.479508
UserId-wordCount             1.0      1    0.000000
UserId-alphaNumerics         0.0      2    0.070739
Summary-textLen             30.0    128   14.028064
Summary-wordCount            5.0     42    2.597313
Summary-alphaNumerics        1.0     72    1.461635
Text-textLen               527.0  21409  445.345607
Text-wordCount              98.0   3432   79.456485
Text-alphaNumerics          20.0   2035   24.855878
```

0.0.2 Categorical Features Report

Categorical features report includes: 1. Cardinality 2. Frequency of mode 3. 2nd mode - the second most frequent value 4. Proportion of mode in the dataset 5. Proportion of 2nd mode in the dataset 6. Frequency of 2nd mode 7. % missing values 8. Mode - the most frequent value

[1]

[8]:
```python
import pandas as pd
import warnings

def build_categorical_features_report(data_df):

    def _mode(df):
        return df.apply(lambda ft: ft.mode().to_list()).T

    def _mode_freq(df):
        return df.apply(lambda ft: ft.value_counts()[ft.mode()].sum())

    def _second_mode(df):
        return df.apply(lambda ft: ft[~ft.isin(ft.mode())].mode().to_list()).T

    def _second_mode_freq(df):
        return df.apply(
            lambda ft: ft[~ft.isin(ft.mode())]
            .value_counts()[ft[~ft.isin(ft.mode())].mode()]
            .sum()
        )

    def _length(df):
        return df.apply(lambda ft: ft.map(str).apply(len))

    stats = {
        "Count": len,
        "Card.": lambda df: df.nunique(),
        "Mode": _mode,
        "Mode Freq": _mode_freq,
```

```
        "Mode %": lambda df: _mode_freq(df) / len(df) * 100,
        "2nd Mode": _second_mode,
        "2nd Mode Freq": _second_mode_freq,
        "2nd Mode %": lambda df: _second_mode_freq(df) / len(df) * 100,
    }

    cat_feat_names = data_df.select_dtypes(exclude="number").columns
    continuous_data_df = data_df[cat_feat_names]

    report_df = pd.DataFrame(index=cat_feat_names, columns=stats.keys())

    for stat_name, fn in stats.items():
        # NOTE: ignore warnings for empty features
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=RuntimeWarning)
            report_df[stat_name] = fn(continuous_data_df)

    return report_df


print(build_categorical_features_report(visualization_df))
```

```
            Count   Card.                                                Mode  \
ProductId  568427   74258                                          B007JFMH8M
UserId     568427  256056                                       A3OXHLG6DIBRW8
Summary    568427  295742                                           Delicious!
Text       568427  393576  This review will make me sound really stupid, …

           Mode Freq    Mode %  \
ProductId        913  0.160619
UserId           448  0.078814
Summary         2462  0.433125
Text             199  0.035009

                                                  2nd Mode  2nd Mode Freq  \
ProductId   [B0026RQTGE, B002QWHJOU, B002QWP89S, B002QWP8H0]           2528
UserId                                        [A1YUL9PCJR3JTY]            421
Summary                                            [Delicious]           2316
Text       [Diamond Almonds<br />Almonds are a good sourc…            126

           2nd Mode %
ProductId    0.444736
UserId       0.074064
Summary      0.407440
Text         0.022166
```

0.0.3 Feature Visualization

[9]:
```
#[6]https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.select_dtypes.
 ↪html
cont_feat = visualization_df.select_dtypes("number").columns
print(cont_feat)
```

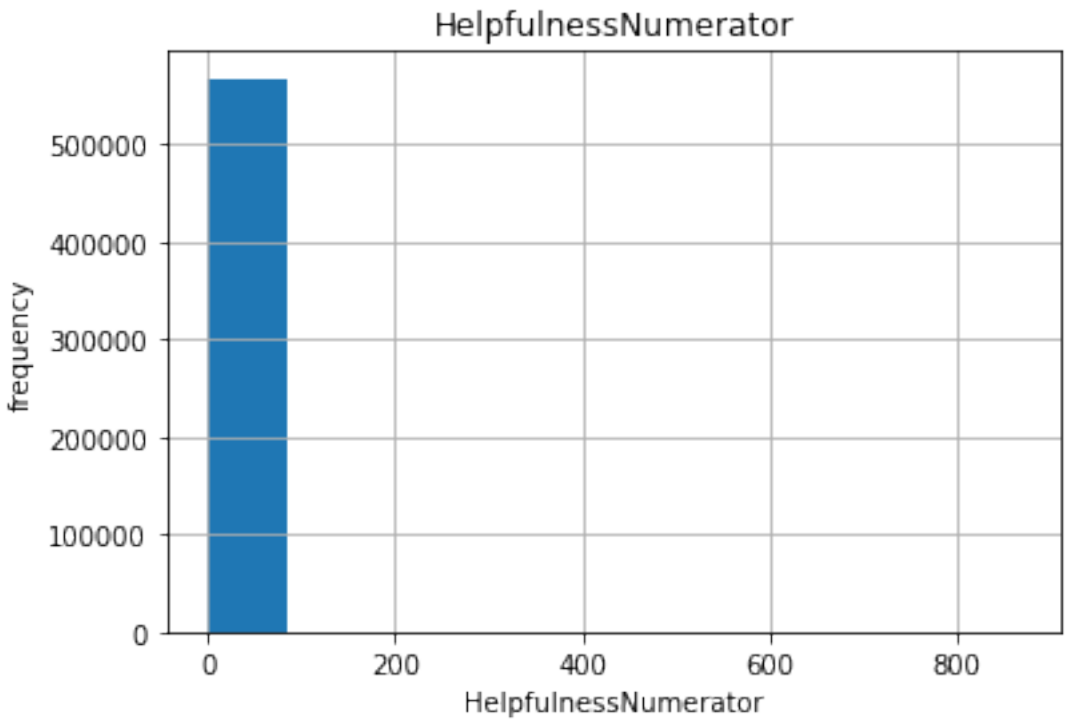
```
Index(['HelpfulnessNumerator', 'HelpfulnessDenominator', 'Score',
       'ProductId-textLen', 'ProductId-wordCount', 'ProductId-alphaNumerics',
       'UserId-textLen', 'UserId-wordCount', 'UserId-alphaNumerics',
       'Summary-textLen', 'Summary-wordCount', 'Summary-alphaNumerics',
       'Text-textLen', 'Text-wordCount', 'Text-alphaNumerics'],
      dtype='object')
```

Creating histograms for for the above features.[5]

[139]:
```
from matplotlib import pyplot as plt
```

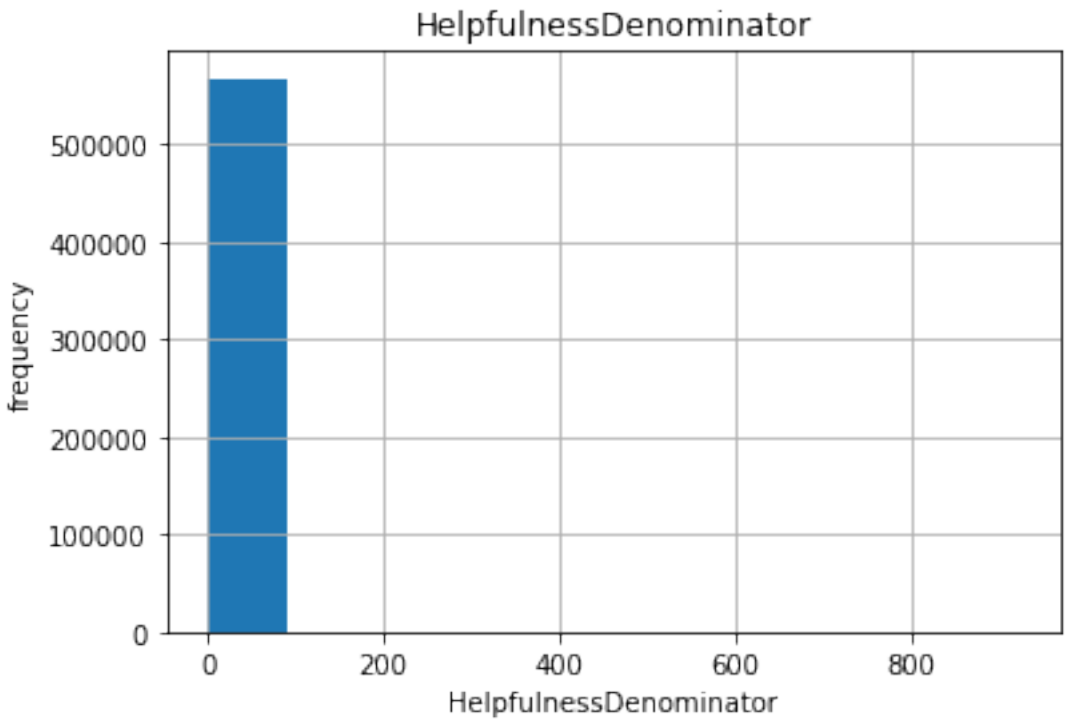
[10]:
```
i = -1
#latitude graph.
from matplotlib import pyplot as plt

i = i+1
#specifying the title, x-axis and y-axis name.
# Ref from [1]
visualization_df.hist(column=[cont_feat[i]])
plt.xlabel(cont_feat[i])
plt.ylabel("frequency")
```

[10]:
```
Text(0, 0.5, 'frequency')
```



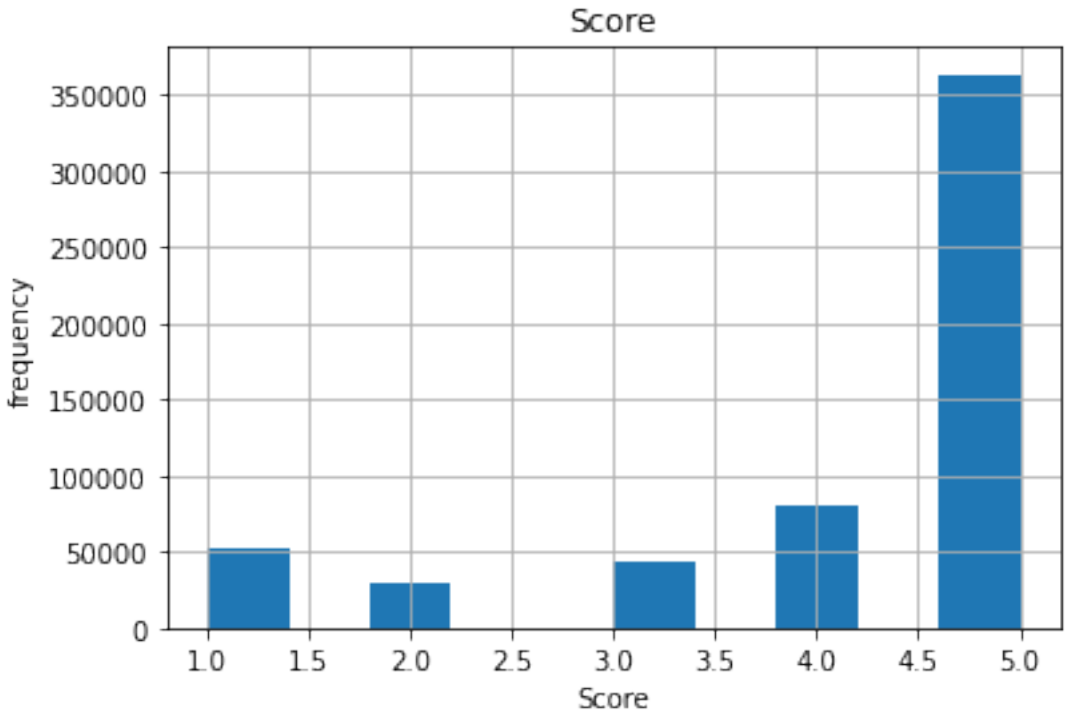
[11]:
```
i = i+1
#specifying the title, x-axis and y-axis name.
# Ref from [1]
visualization_df.hist(column=[cont_feat[i]])
plt.xlabel(cont_feat[i])
plt.ylabel("frequency")
```

[11]: Text(0, 0.5, 'frequency')



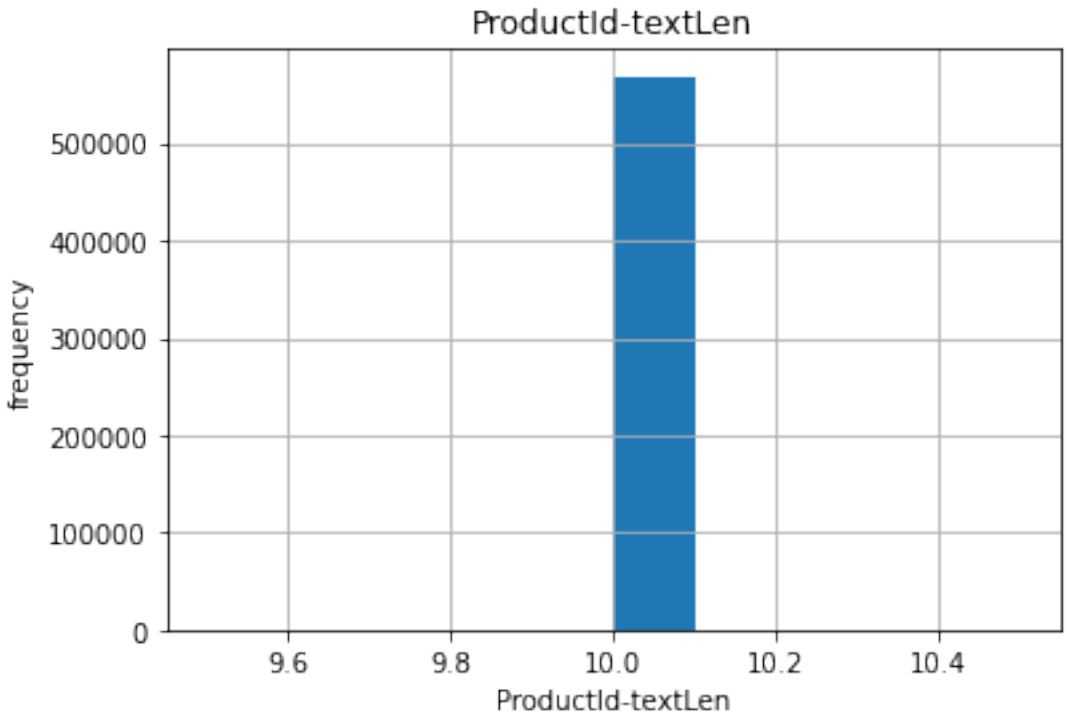
[12]:
```
i = i+1
#specifying the title, x-axis and y-axis name.
# Ref from [1]
visualization_df.hist(column=[cont_feat[i]])
plt.xlabel(cont_feat[i])
plt.ylabel("frequency")
```

[12]: Text(0, 0.5, 'frequency')



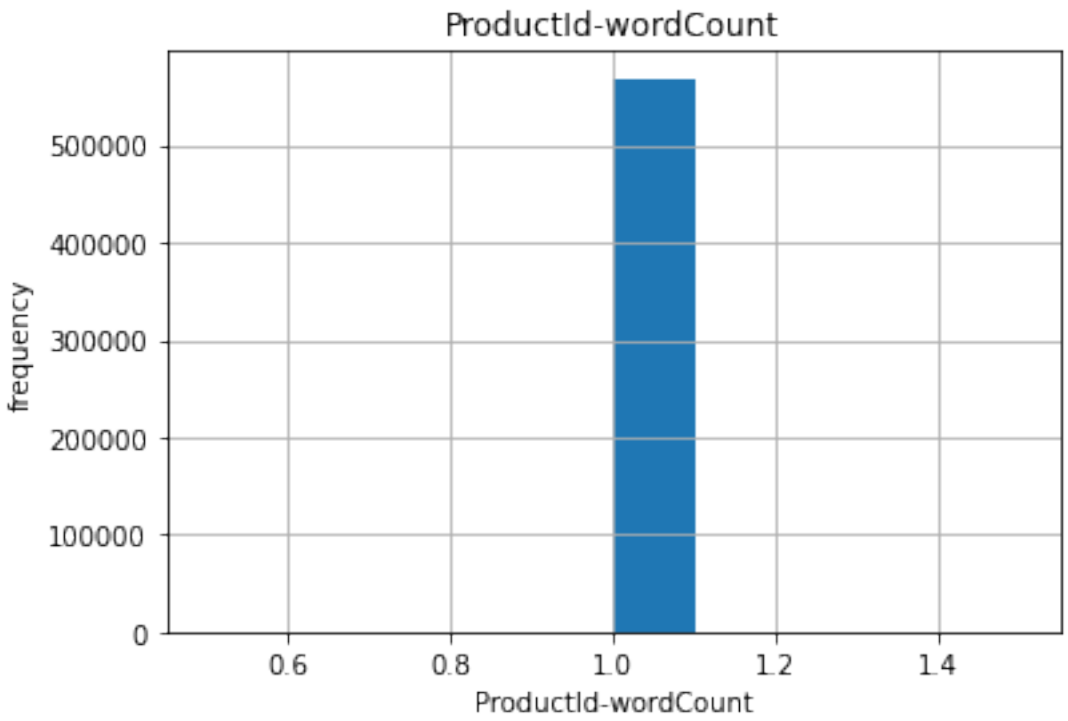
[13]:
```
i = i+1
#specifying the title, x-axis and y-axis name.
# Ref from [1]
visualization_df.hist(column=[cont_feat[i]])
plt.xlabel(cont_feat[i])
plt.ylabel("frequency")
```

[13]: Text(0, 0.5, 'frequency')



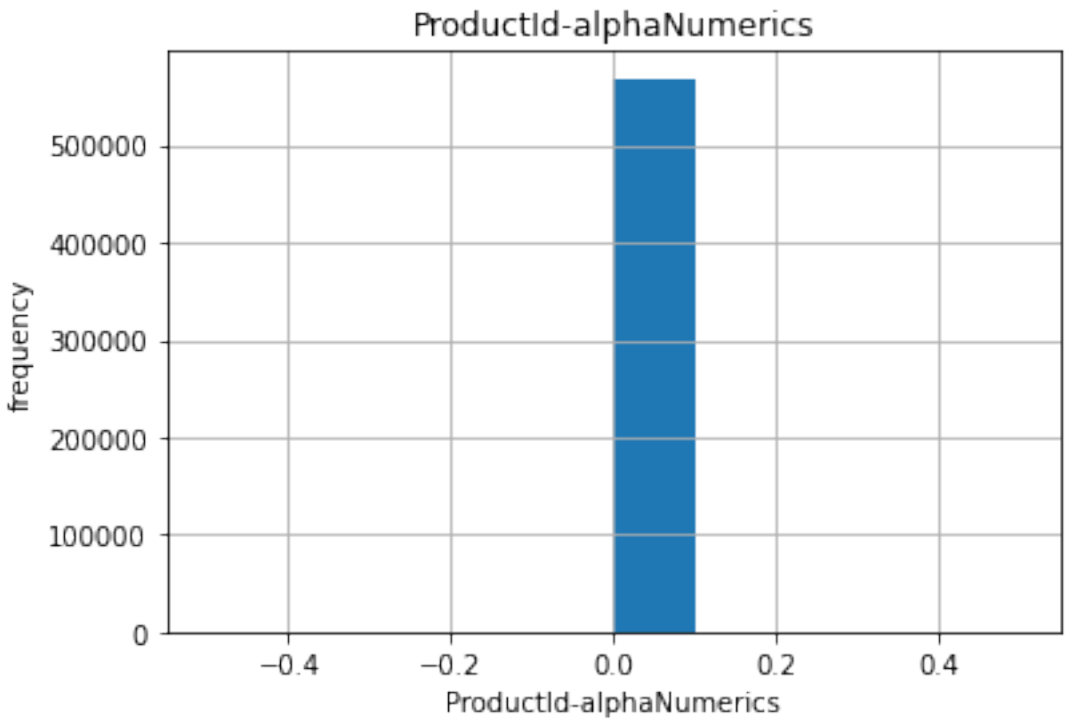
[14]:
```
i = i+1
#specifying the title, x-axis and y-axis name.
# Ref from [1]
visualization_df.hist(column=[cont_feat[i]])
plt.xlabel(cont_feat[i])
plt.ylabel("frequency")
```

[14]: Text(0, 0.5, 'frequency')



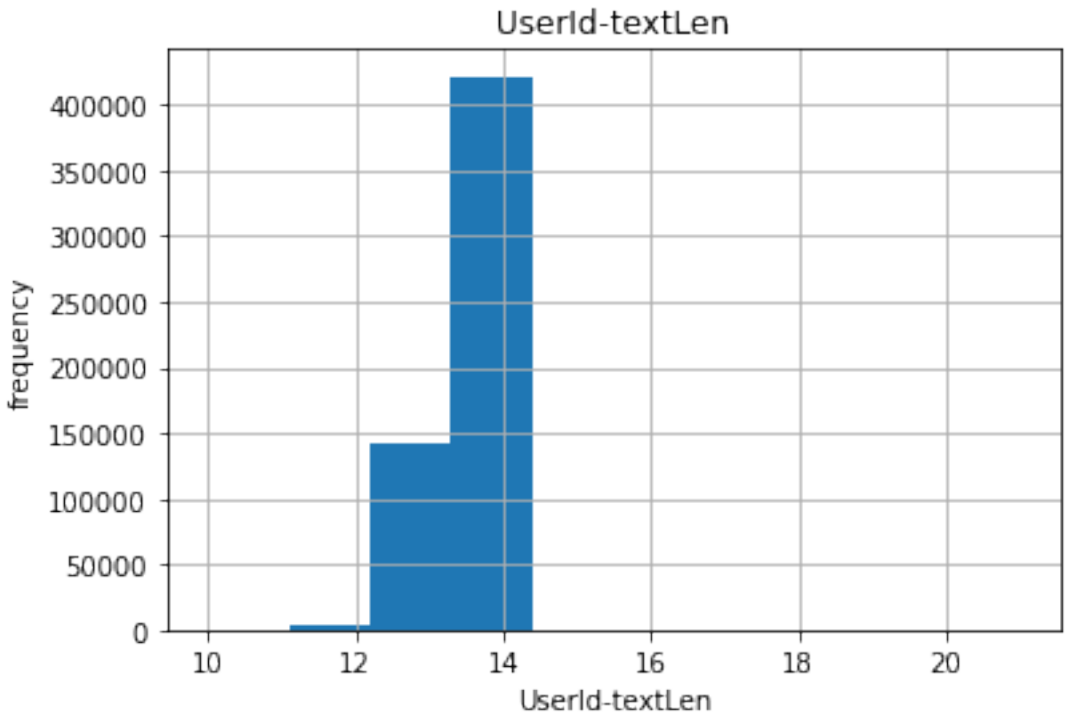
[15]:
```
i = i+1
#specifying the title, x-axis and y-axis name.
# Ref from [1]
visualization_df.hist(column=[cont_feat[i]])
plt.xlabel(cont_feat[i])
plt.ylabel("frequency")
```

[15]: Text(0, 0.5, 'frequency')



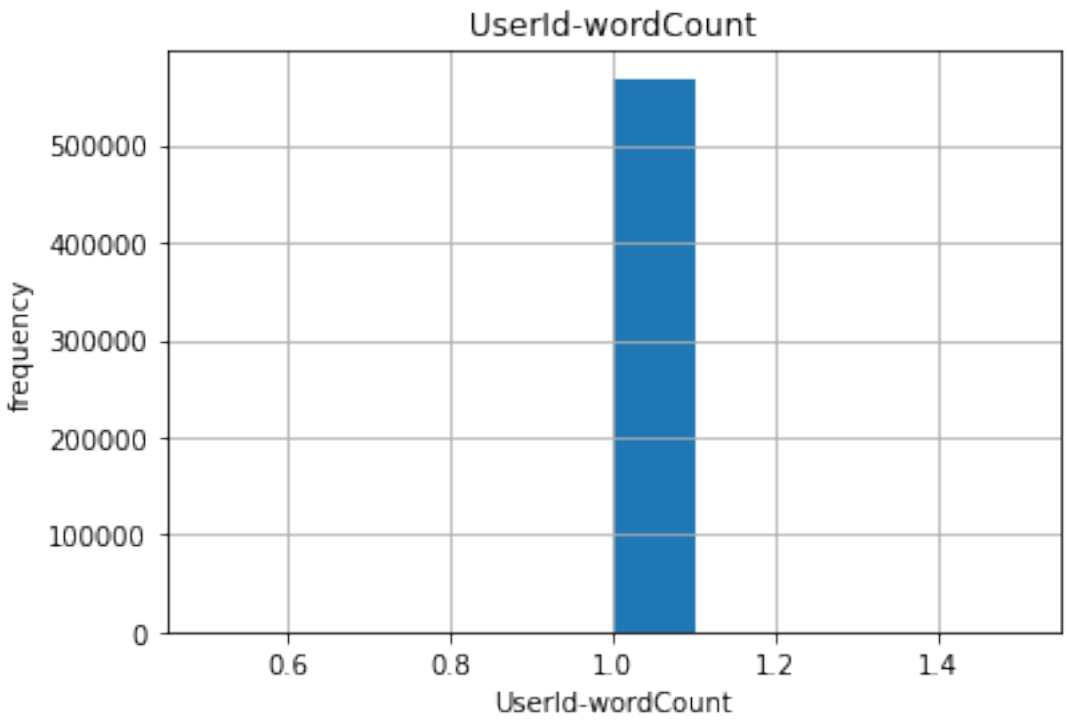
[16]:
```
i = i+1
#specifying the title, x-axis and y-axis name.
# Ref from [1]
visualization_df.hist(column=[cont_feat[i]])
plt.xlabel(cont_feat[i])
plt.ylabel("frequency")
```

[16]: Text(0, 0.5, 'frequency')



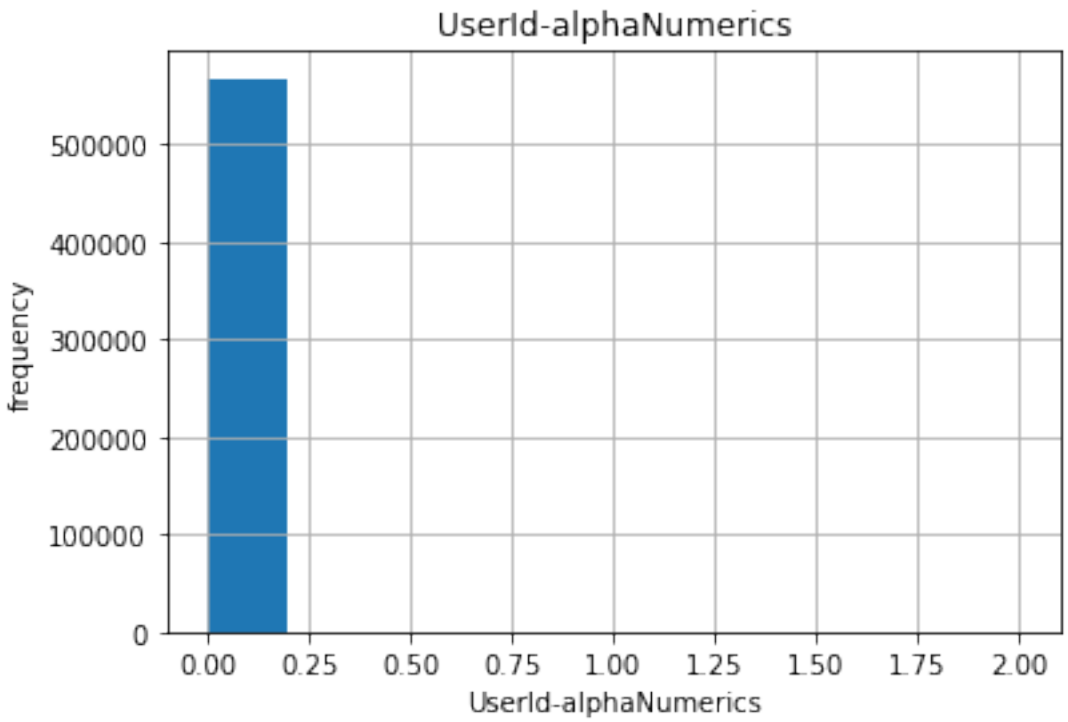
[17]:
```
i = i+1
#specifying the title, x-axis and y-axis name.
# Ref from [1]
visualization_df.hist(column=[cont_feat[i]])
plt.xlabel(cont_feat[i])
plt.ylabel("frequency")
```

[17]: Text(0, 0.5, 'frequency')



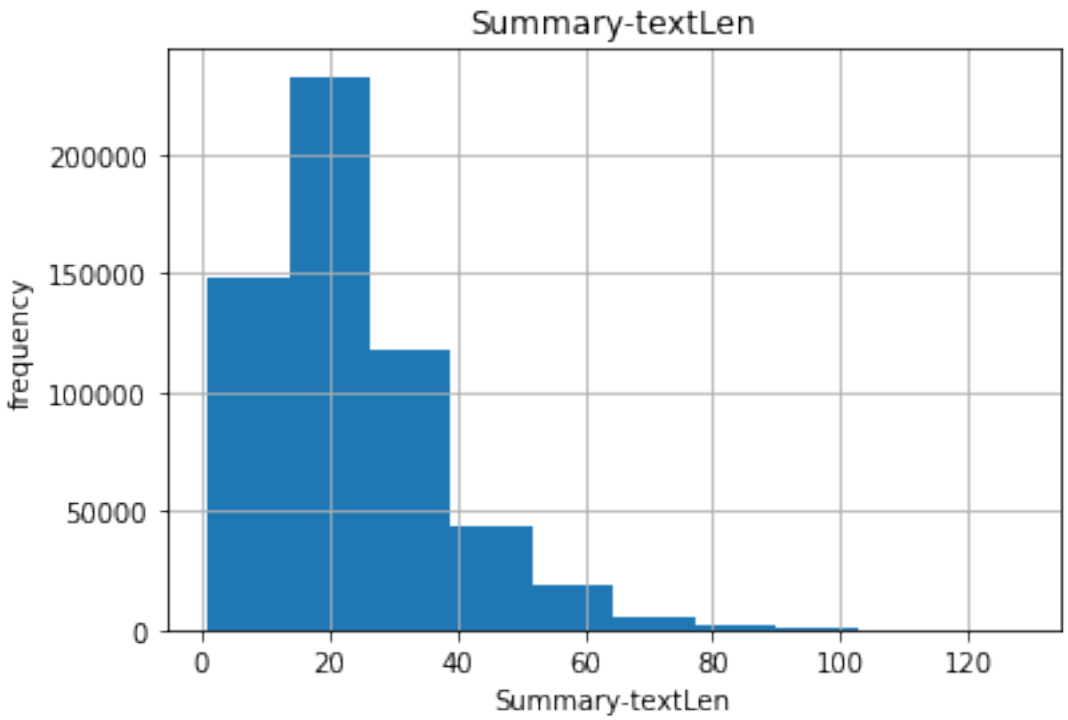
[18]:
```
i = i+1
#specifying the title, x-axis and y-axis name.
# Ref from [1]
visualization_df.hist(column=[cont_feat[i]])
plt.xlabel(cont_feat[i])
plt.ylabel("frequency")
```

[18]: Text(0, 0.5, 'frequency')



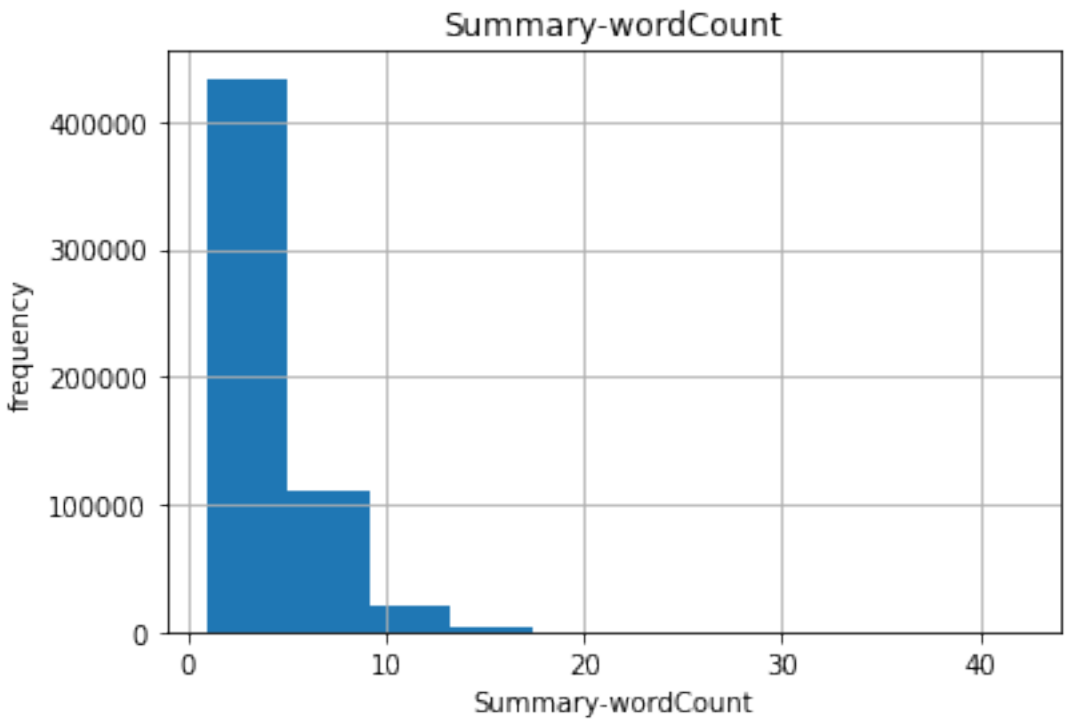
[19]:
```
i = i+1
#specifying the title, x-axis and y-axis name.
# Ref from [1]
visualization_df.hist(column=[cont_feat[i]])
plt.xlabel(cont_feat[i])
plt.ylabel("frequency")
```

[19]: Text(0, 0.5, 'frequency')



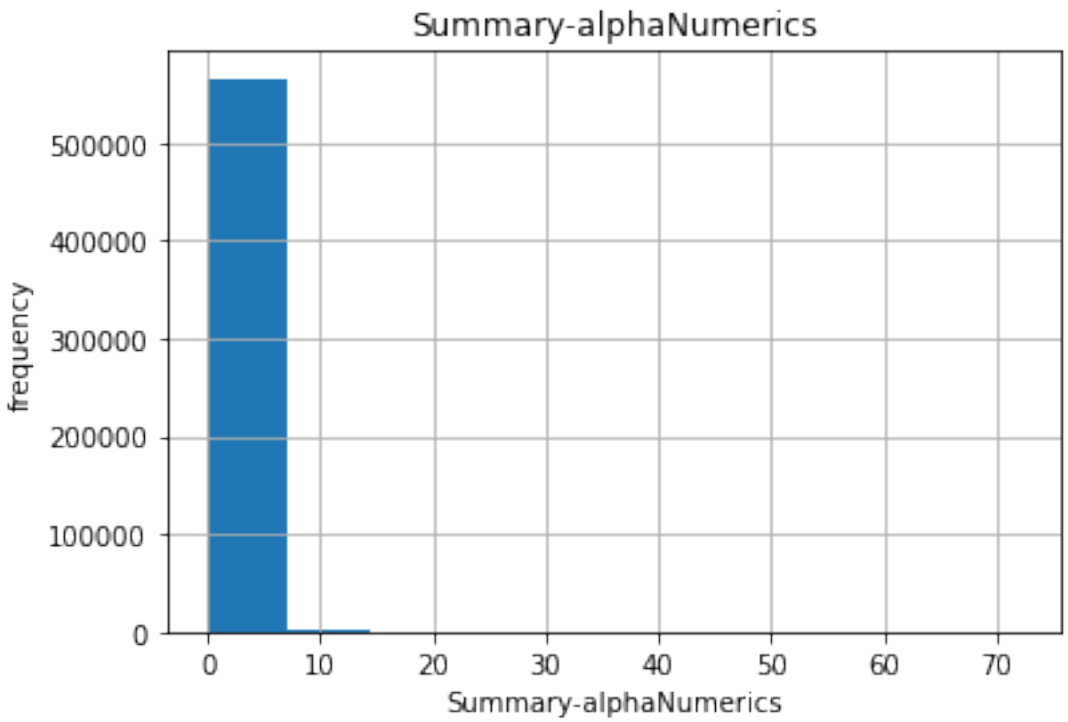
[20]:
```
i = i+1
#specifying the title, x-axis and y-axis name.
# Ref from [1]
visualization_df.hist(column=[cont_feat[i]])
plt.xlabel(cont_feat[i])
plt.ylabel("frequency")
```

[20]: Text(0, 0.5, 'frequency')



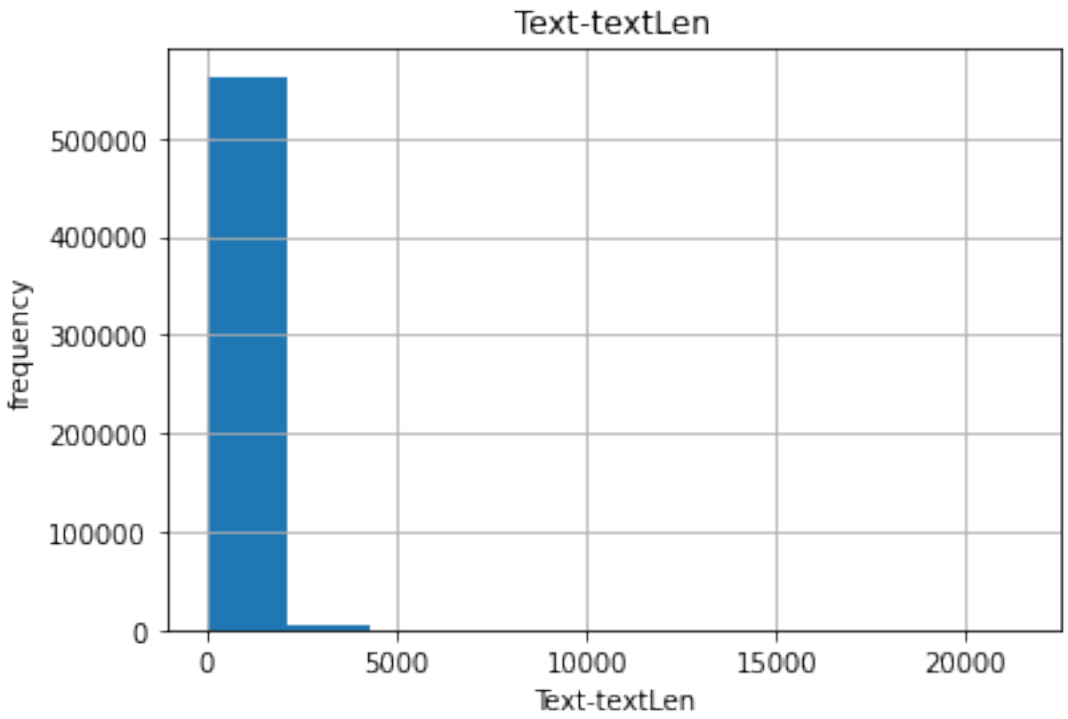
[21]:
```
i = i+1
#specifying the title, x-axis and y-axis name.
# Ref from [1]
visualization_df.hist(column=[cont_feat[i]])
plt.xlabel(cont_feat[i])
plt.ylabel("frequency")
```

[21]: Text(0, 0.5, 'frequency')



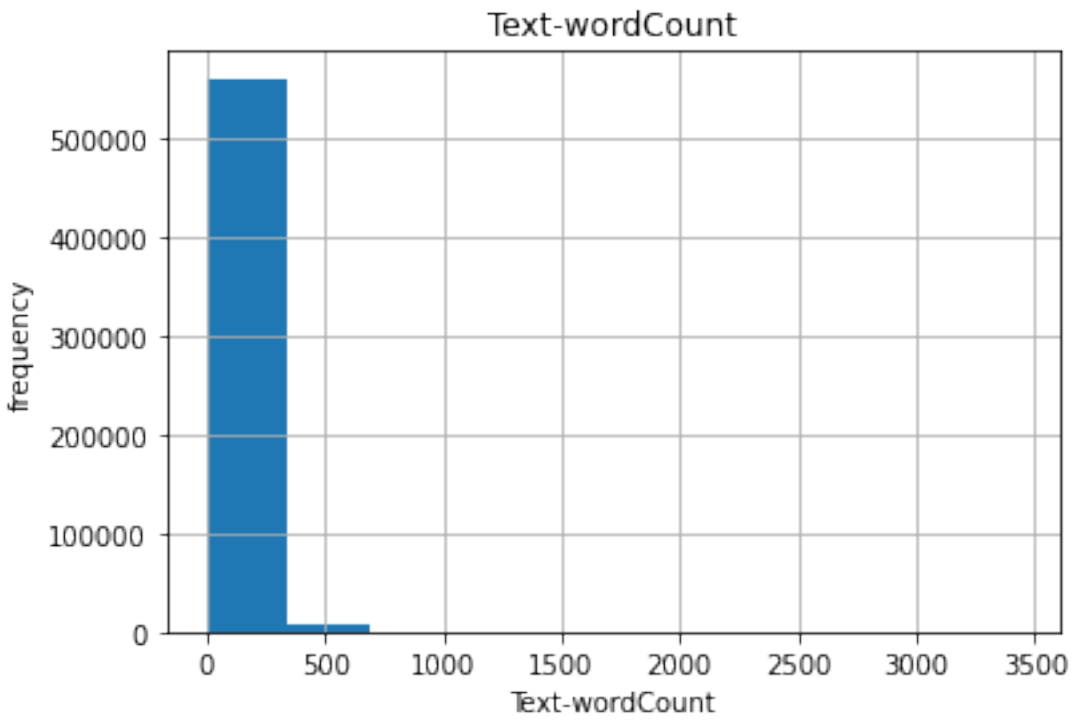
[22]:
```
i = i+1
#specifying the title, x-axis and y-axis name.
# Ref from [1]
visualization_df.hist(column=[cont_feat[i]])
plt.xlabel(cont_feat[i])
plt.ylabel("frequency")
```

[22]: Text(0, 0.5, 'frequency')



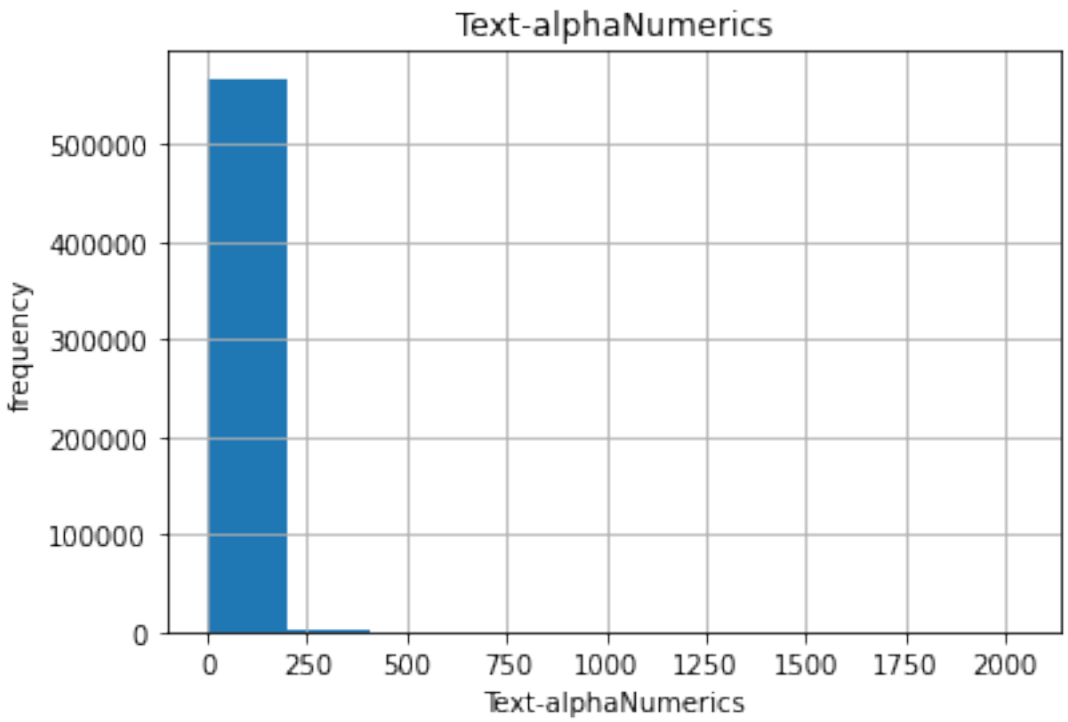
[23]:
```
i = i+1
#specifying the title, x-axis and y-axis name.
# Ref from [1]
visualization_df.hist(column=[cont_feat[i]])
plt.xlabel(cont_feat[i])
plt.ylabel("frequency")
```

[23]: Text(0, 0.5, 'frequency')



[24]:
```
i = i+1
#specifying the title, x-axis and y-axis name.
# Ref from [1]
visualization_df.hist(column=[cont_feat[i]])
plt.xlabel(cont_feat[i])
plt.ylabel("frequency")
```

[24]: Text(0, 0.5, 'frequency')



[25]:
```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

nltk.download('stopwords')

stop_words=set(stopwords.words('english'))

def Len_Not_stopwords(doc):
    doc = word_tokenize(doc) #making a list of cell in dataset
    doc = [w for w in doc if not w in stop_words] #removing the stop words
    return len(doc)
```

```
[nltk_data] Error loading stopwords: <urlopen error [SSL:
[nltk_data]     CERTIFICATE_VERIFY_FAILED] certificate verify failed:
[nltk_data]     unable to get local issuer certificate (_ssl.c:997)>
```

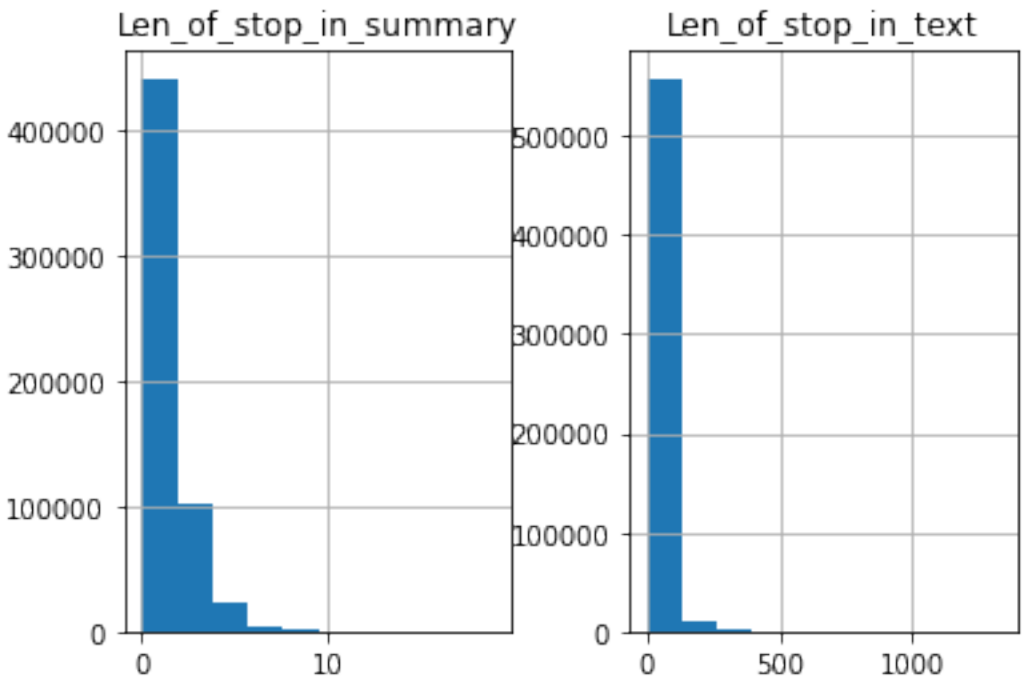
[26]:
```
# Ref from [2] and [3]
visualization_df['Summary']=visualization_df['Summary'].swifter.apply(lambda l:␣
 ↪l.strip())
visualization_df['Text']=visualization_df['Text'].swifter.apply(lambda l: l.
 ↪strip())
```

```
visualization_df['Summary_Stop_len']=visualization_df['Summary'].swifter.
 ↪apply(lambda l: Len_Not_stopwords(l))
visualization_df['Text_Stop_len']=visualization_df['Text'].swifter.apply(lambda␣
 ↪l: Len_Not_stopwords(l))
visualization_df['Summary_len']=visualization_df['Summary'].swifter.
 ↪apply(lambda l: len(word_tokenize(l)))
visualization_df['Text_len']=visualization_df['Text'].swifter.apply(lambda l:␣
 ↪len(word_tokenize(l)))
visualization_df['Len_of_stop_in_summary']=visualization_df['Summary_len']-visualization_df['S
visualization_df['Len_of_stop_in_text']=visualization_df['Text_len']-visualization_df['Text_St
df_stop=visualization_df[['Len_of_stop_in_summary','Len_of_stop_in_text']]
display(df_stop.describe(),df_stop.hist())
```

```
Pandas Apply:   0%|          | 0/568427 [00:00<?, ?it/s]

Pandas Apply:   0%|          | 0/568427 [00:00<?, ?it/s]

Pandas Apply:   0%|          | 0/568427 [00:00<?, ?it/s]

Pandas Apply:   0%|          | 0/568427 [00:00<?, ?it/s]

Pandas Apply:   0%|          | 0/568427 [00:00<?, ?it/s]

Pandas Apply:   0%|          | 0/568427 [00:00<?, ?it/s]
```

	Len_of_stop_in_summary	Len_of_stop_in_text
count	568427.000000	568427.000000
mean	0.866289	33.611333
std	1.283965	33.524029
min	0.000000	0.000000
25%	0.000000	14.000000
50%	0.000000	24.000000
75%	1.000000	42.000000
max	19.000000	1327.000000

```
array([[<AxesSubplot:title={'center':'Len_of_stop_in_summary'}>,
        <AxesSubplot:title={'center':'Len_of_stop_in_text'}>]],
      dtype=object)
```



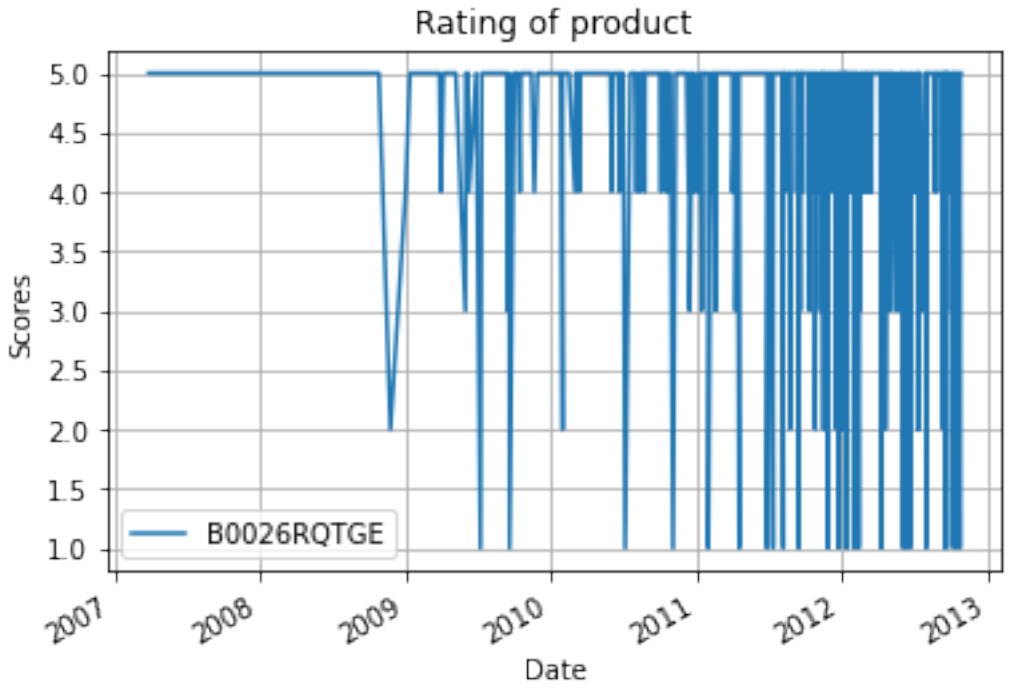
[27]:
```
sort = visualization_df.groupby(['ProductId']).size()
sort = pd.DataFrame(sort)
sort.sort_values([0],ascending=False)
```

[27]:
```
                0
ProductId
B007JFMH8M  913
B0026RQTGE  632
B002QWHJOU  632
B002QWP89S  632
B002QWP8H0  632
…           …
B000YPQC08    1
B003YU5T6I    1
B000YPQC44    1
B000YPQE6U    1
B009WVB40S    1

[74258 rows x 1 columns]
```

[101]:
```
df_vis=pd.read_csv('../Project/Reviews.csv')

df_vis=df_vis.drop(['Id', 'ProfileName'], axis=1)
```

```
def get_graph(pid):
    req=df_vis.loc[df_vis['ProductId']==pid]
    req=req[['Time','Score']]
    req=req.reset_index()
    req=req.drop(['index'], axis=1)
    req['date'] = pd.to_datetime(req['Time'],unit='s')
    req['Month']=req['date'].dt.month
    req['Year']=req['date'].dt.year
    req=req.sort_values(['date'])
    req=req.drop_duplicates()
    X = list(req['date'])
    Y = {pid:list(req['Score'])}

    graph = pd.DataFrame(Y,X)
    return graph.plot(kind='line', grid=True, title='Rating of product',␣
 ↪xlabel='Date', ylabel='Scores')

get_graph(pid='B0026RQTGE')
```

[101]: <AxesSubplot:title={'center':'Rating of product'}, xlabel='Date', ylabel='Scores'>



0.1 Pre-Processing

[29]:
```
import nltk
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import swifter
import ssl

try:
    _create_unverified_https_context = ssl._create_unverified_context
except AttributeError:
    pass
else:
    ssl._create_default_https_context = _create_unverified_https_context

#import spacy
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

nltk.download('stopwords')

stop_words=set(stopwords.words('english'))
#nlp = spacy.load("en_core_web_sm")

# ref from [6]
def pre_processing(doc):
    doc = word_tokenize(doc.lower()) #making a list of cell in dataset
    doc = [w for w in doc if not w in stop_words] #removing the stop words
    doc = [re.sub(r'[^a-zA-Z ]+', '', item) for item in doc] #removing all␣
 ↪characters other than character
    doc = [lemmatizer.lemmatize(token) for token in doc] #lemmatization
    doc = ' '.join(doc)
    doc = doc.strip() #removing empty space
    return doc
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/jaswanth106/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[30]:
```
#df['Summary_pre']=df['Summary'].swifter.apply(lambda l: pre_processing(l))
df['Text_pre']=df['Text'].swifter.apply(lambda l: pre_processing(l))
df
```

```
Pandas Apply:   0%|          | 0/568427 [00:00<?, ?it/s]
```

[30]:

	ProductId	UserId	HelpfulnessNumerator	\
0	B001E4KFG0	A3SGXH7AUHU8GW	1	
1	B00813GRG4	A1D87F6ZCVE5NK	0	
2	B000LQOCH0	ABXLMWJIXXAIN	1	
3	B000UA0QIQ	A395BORC6FGVXV	3	
4	B006K2ZZ7K	A1UQRSCLF8GW1T	0	
…	…	…	…	
568449	B001EO7N10	A28KG5XORO54AY	0	
568450	B003S1WTCU	A3I8AFVPEE8KI5	0	
568451	B004I613EE	A121AA1GQV751Z	2	
568452	B004I613EE	A3IBEVCTXKNOH	1	
568453	B001LR2CU2	A3LGQPJCZVL9UC	0	

	HelpfulnessDenominator	Score	Summary	\
0	1	5	Good Quality Dog Food	
1	0	1	Not as Advertised	
2	1	4	"Delight" says it all	
3	3	2	Cough Medicine	
4	0	5	Great taffy	
…	…	…	…	
568449	0	5	Will not do without	
568450	0	2	disappointed	
568451	2	5	Perfect for our maltipoo	
568452	1	5	Favorite Training and reward treat	
568453	0	5	Great Honey	

	Text	\
0	I have bought several of the Vitality canned d…	
1	Product arrived labeled as Jumbo Salted Peanut…	
2	This is a confection that has been around a fe…	
3	If you are looking for the secret ingredient i…	
4	Great taffy at a great price. There was a wid…	
…	…	
568449	Great for sesame chicken..this is a good if no…	
568450	I'm disappointed with the flavor. The chocolat…	
568451	These stars are small, so you can give 10-15 o…	
568452	These are the BEST treats for training and rew…	
568453	I am very satisfied ,product is as advertised,…	

	Text_pre
0	bought several vitality canned dog food produc…
1	product arrived labeled jumbo salted peanut p…
2	confection around century light pillowy citr…
3	looking secret ingredient robitussin believe f…
4	great taffy great price wide assortment yummy…
…	…
568449	great sesame chicken good better resturants e…

```
568450  m disappointed flavor  chocolate note especial…
568451  star small  give  one training session  tried …
568452  best treat training rewarding dog good groomin…
568453  satisfied  product advertised  use cereal  raw…

[568427 rows x 8 columns]
```

[31]:
```
df.Score.value_counts()
```

[31]:
```
5    363122
4     80655
1     52268
3     42638
2     29744
Name: Score, dtype: int64
```

[32]:
```
count_5, count_4, count_1, count_3, count_2 = df.Score.value_counts()
avg=int((count_5 + count_4 + count_1 + count_3 + count_2)/5)
```

[33]:
```
df_class_1 = df[df['Score']==1]
df_class_2 = df[df['Score']==2]
df_class_3 = df[df['Score']==3]
df_class_4 = df[df['Score']==4]
df_class_5 = df[df['Score']==5]
```

[34]:
```
class_1 = df_class_1.sample(avg, replace=True)
class_2 = df_class_2.sample(avg, replace=True)
class_3 = df_class_3.sample(avg, replace=True)
class_4 = df_class_4.sample(avg, replace=True)
class_5 = df_class_5.sample(avg)
```

[35]:
```
df_balanced= pd.concat([class_1,class_2,class_3,class_4,class_5], axis=0)
df_balanced
```

[35]:
```
          ProductId          UserId  HelpfulnessNumerator  \
425093  B000WNLC66  A1LD2S14L3AC1G                     0
380879  B0029NM6NU  A1ZORN2GEZBNV7                     0
560024  B0030GNQMU  A17Z0PXGEGVYLY                     0
243087  B005K4Q4KG  A2BZN96W6QP6XH                     0
88745   B000F4BCS0  A20DNQ7AZD7XF4                     0
…              …              …                   …
402227  B0008IUQE4  A2G11GPHNCN130                     0
91399   B005HF23NU  A2XFHRKK86ZCFE                     0
403931  B003FDG4K4  A1EM5PUUCWSXUQ                     0
168555  B0001ES9F8  A37B8WTZ6GINOB                     0
171172  7310172001  A14YBOI1XSJ0UP                     0
```

```
        HelpfulnessDenominator  Score                        Summary  \
425093                       0      1  Most disgusting product ever!
380879                       3      1         Crazy people buy this!
560024                       0      1                      Overpriced
243087                       2      1             not quality at all
88745                        0      1     Packing issues - Open bags
…                          …    …                            …
402227                       0      5                Excellent sauce
91399                        0      5   Best chip I have ever eaten!
403931                       0      5                Awesome Product!
168555                       0      5              Senseo Coffee Pods
171172                       0      5            Really great treats!

                                                    Text  \
425093  When this arrived there were no whole dog trea…
380879  I bought this because it was on sale, and I wa…
560024  I have written to this company to try to get a…
243087  this product taste stale and is full of artifi…
88745   Half of the bags in case did not get sealed at…
…                                                     …
402227  Despite the name, it's great for those who req…
91399   I live in southwest missouri now and recently …
403931  This product is the best meal replacement I ha…
168555  I love my senseo pod maker and the price you o…
171172  I use these as a training treat when I am work…

                                                Text_pre
425093  arrived whole dog treat bag  br   bag filled l…
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…
…                                                     …
402227  despite name  s great require bit heat every d…
91399   live southwest missouri recently visited daugh…
403931  product best meal replacement ever found  actu…
168555  love senseo pod maker price offer coffee pod c…
171172  use training treat working two dog  dog gettin…

[568425 rows x 8 columns]
```

[36]:
```
df_balanced.Score.value_counts()
```

[36]:
```
1    113685
2    113685
3    113685
4    113685
```

```
5    113685
Name: Score, dtype: int64
```

1 Model Preparation

[37]:
```
from sklearn.model_selection import train_test_split

x=df_balanced['Text_pre']
y=df_balanced['Score']

X_train, X_test, y_train, y_test = train_test_split(x, y)
```

[38]:
```
y_test.value_counts()
```

[38]:
```
1    28577
2    28467
4    28373
5    28370
3    28320
Name: Score, dtype: int64
```

[39]:
```
y_train.value_counts()
```

[39]:
```
3    85365
5    85315
4    85312
2    85218
1    85108
Name: Score, dtype: int64
```

1.1 Model Selection

1.1.1 Text Only

[85]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Ref from [4]
le = LabelEncoder()
x=df_balanced['Text_pre']
y= le.fit_transform(df_balanced['Score'])

X_train, X_test, y_train, y_test = train_test_split(x, y)

results = []
names = []
```

```
scoring = 'accuracy'
```

[87]:
```
#KNN ref from [8]
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
from sklearn.feature_extraction.text import TfidfVectorizer

# Ref from [5] for pipelining and vectorizing
KNN_Text = Pipeline([('vectorizer_tfidf',TfidfVectorizer()),('KNN',␣
 ↪KNeighborsClassifier())])


kfold = model_selection.KFold(n_splits=5, random_state=5, shuffle=True)
cv_results = model_selection.cross_val_score(KNN_Text, x, y, cv=kfold,␣
 ↪scoring=scoring)
results.append(cv_results)
names.append("KNN")
msg = "%s: %f (%f)" % ("KNN", cv_results.mean(), cv_results.std())
print(msg)
```

```
KNN: 0.708169 (0.035347)
```

[88]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier

# Ref from [5] for pipelining and vectorizing
RFC_Text = Pipeline([('vectorizer_tfidf',TfidfVectorizer()),('RandomForest',␣
 ↪RandomForestClassifier())])

kfold = model_selection.KFold(n_splits=5, random_state=5, shuffle=True)
cv_results = model_selection.cross_val_score(RFC_Text, x, y, cv=kfold,␣
 ↪scoring=scoring)
results.append(cv_results)
names.append("RandomForest")
msg = "%s: %f (%f)" % ("RandomForest", cv_results.mean(), cv_results.std())
print(msg)
```

```
RandomForest: 0.932220 (0.000454)
```

[86]:
```
#XG Boost
import xgboost as xgb

# Ref from [5] for pipelining and vectorizing
XGB_Text = Pipeline([('vectorizer_tfidf',TfidfVectorizer()),('XGBoost', xgb.
 ↪XGBClassifier())])
```

```
kfold = model_selection.KFold(n_splits=5, random_state=5, shuffle=True)
cv_results = model_selection.cross_val_score(XGB_Text, x, y, cv=kfold,␣
 ↪scoring=scoring)
results.append(cv_results)
names.append("XGBoost")
msg = "%s: %f (%f)" % ("XGBoost", cv_results.mean(), cv_results.std())
print(msg)
```

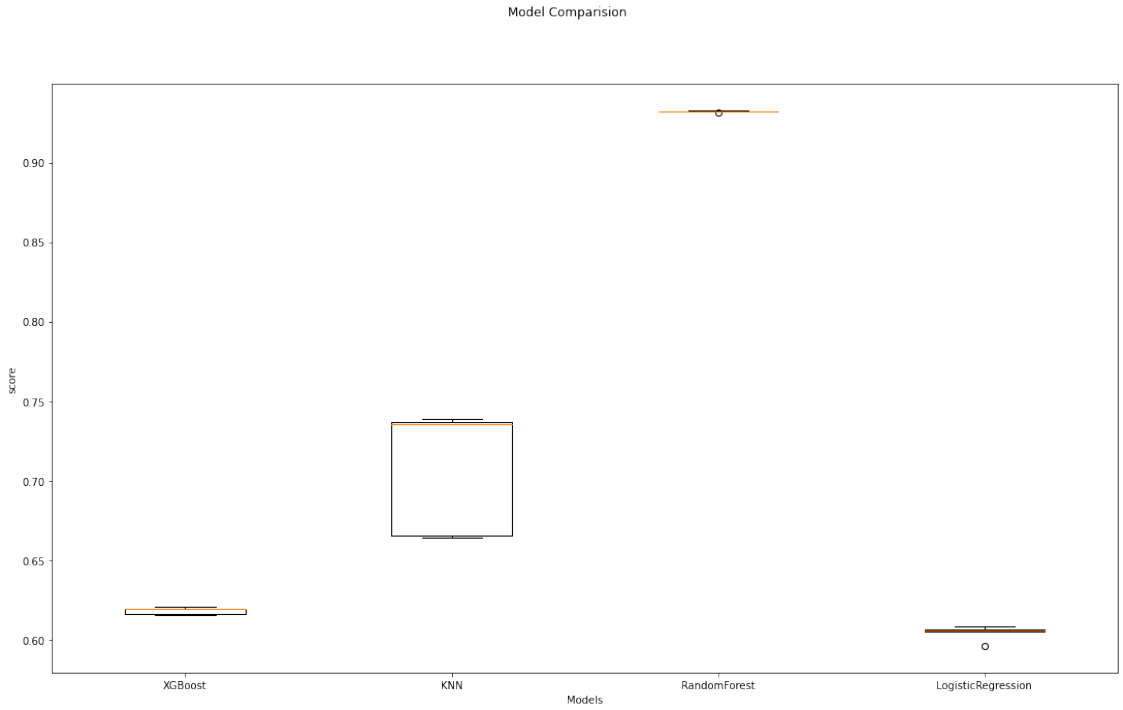
```
XGBoost: 0.618110 (0.002017)
```

[95]:
```
# Logistic Regression
from sklearn.linear_model import LogisticRegression

# Ref from [5] for pipelining and vectorizing
LR_Text =␣
 ↪Pipeline([('vectorizer_tfidf',TfidfVectorizer()),('LogisticRegression',␣
 ↪LogisticRegression())])

kfold = model_selection.KFold(n_splits=5, random_state=5, shuffle=True)
cv_results = model_selection.cross_val_score(LR_Text, x, y, cv=kfold,␣
 ↪scoring=scoring)
results.append(cv_results)
names.append("LogisticRegression")
msg = "%s: %f (%f)" % ("LogisticRegression", cv_results.mean(), cv_results.
 ↪std())
print(msg)
```

```
LogisticRegression: 0.604361 (0.004264)
```

[151]:
```
from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
import xgboost as xgb
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer

# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Model Comparision')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
fig = plt.gcf()
fig.set_size_inches(18.5, 10.5)
plt.xlabel('Models')
```

```
plt.ylabel('score')
plt.show()
```



[100]:
```
print("                    XGBoost                KNN                 ␣
↪Randomforest              LogisticRegression")
print("mean score                0.618                0.708                0.
↪932                       0.604")
print("standard deviation        0.002                0.035                0.
↪0004                      0.004")
```

```
                          XGBoost                KNN
Randomforest              LogisticRegression
mean score                0.618                0.708                0.932
0.604
standard deviation        0.002                0.035                0.0004
0.004
```

1.2 Experiments

1.2.1 Summary only

[55]:
```
# creating new df for experiments
exp_df = df_balanced.copy()

# summary should be preprocessed before using it.
```

```
exp_df['Summary_preprocessed']=exp_df['Summary'].swifter.apply(lambda l:␣
 ↪pre_processing(l))
exp_df.head()
```

```
Pandas Apply:   0%|          | 0/568425 [00:00<?, ?it/s]
```

[55]:

	ProductId	UserId	HelpfulnessNumerator	HelpfulnessDenominator	Score	Summary	Text	Text_pre
425093	B000WNLC66	A1LD2S14L3AC1G	0	0	1	Most disgusting product ever!	When this arrived there were no whole dog trea…	arrived whole dog treat bag br bag filled l…
380879	B0029NM6NU	A1ZORN2GEZBNV7	0	3	1	Crazy people buy this!	I bought this because it was on sale, and I wa…	
560024	B0030GNQMU	A17ZOPXGEGVYLY	0	0	1	Overpriced	I have written to this company to try to get a…	
243087	B005K4Q4KG	A2BZN96W6QP6XH	0	2	1	not quality at all	this product taste stale and is full of artifi…	
88745	B000F4BCS0	A20DNQ7AZD7XF4	0	0	1	Packing issues - Open bags	Half of the bags in case did not get sealed at…	
…	…	…	…	…	…	…	…	
402227	B0008IUQE4	A2G11GPHNCN13O	0	0	5	Excellent sauce	Despite the name, it's great for those who req…	
91399	B005HF23NU	A2XFHRKK86ZCFE	0	0	5	Best chip I have ever eaten!	I live in southwest missouri now and recently …	
403931	B003FDG4K4	A1EM5PUUCWSXUQ	0	0	5	Awesome Product!	This product is the best meal replacement I ha…	
168555	B0001ES9F8	A37B8WTZ6GINOB	0	0	5	Senseo Coffee Pods	I love my senseo pod maker and the price you o…	
171172	7310172001	A14YBOI1XSJOUP	0	0	5	Really great treats!	I use these as a training treat when I am work…	

```
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…
…                                                     …
402227  despite name  s great require bit heat every d…
91399   live southwest missouri recently visited daugh…
403931  product best meal replacement ever found  actu…
168555  love senseo pod maker price offer coffee pod c…
171172  use training treat working two dog  dog gettin…

           Summary_preprocessed
425093  disgusting product ever
380879         crazy people buy
560024               overpriced
243087                  quality
88745   packing issue  open bag
…                          …
402227          excellent sauce
91399      best chip ever eaten
403931           awesome product
168555        senseo coffee pod
171172       really great treat

[568425 rows x 9 columns]
```

[56]:
```
feature = exp_df['Summary_preprocessed']
target = exp_df['Score']
```

[57]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
x= feature
y= le.fit_transform(target)

X_train_summary, X_test_summary, y_train_summary, y_test_summary =␣
 ↪train_test_split(x, y)
```

[58]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier

# Ref from [5] for pipelining and vectorizing
RFC_Text = Pipeline([('vectorizer_tfidf',TfidfVectorizer()),('RandomForest',␣
 ↪RandomForestClassifier())])

RFC_Text.fit(X_train_summary, y_train_summary)
```

```
y_pred_summary = RFC_Text.predict(X_test_summary)


print(classification_report(y_test_summary, y_pred_summary))
```

```
              precision    recall  f1-score   support

           0       0.83      0.82      0.83     28281
           1       0.74      0.84      0.79     28269
           2       0.83      0.79      0.81     28566
           3       0.76      0.71      0.73     28634
           4       0.75      0.74      0.74     28357

    accuracy                           0.78    142107
   macro avg       0.78      0.78      0.78    142107
weighted avg       0.78      0.78      0.78    142107
```

1.2.2 text + summary

[62]:
```
# summary and text added to a new column.
exp_df['Text_Summary']=exp_df['Text_pre'] + ' ' + exp_df['Summary_preprocessed']
exp_df.head()
```

[62]:

	ProductId	UserId	HelpfulnessNumerator	HelpfulnessDenominator	Score	Summary	Text	Text_pre
425093	B000WNLC66	A1LD2S14L3AC1G	0	0	1	Most disgusting product ever!	When this arrived there were no whole dog trea…	arrived whole dog treat bag br bag filled l…
380879	B0029NM6NU	A1Z0RN2GEZBNV7	0	3	1	Crazy people buy this!	I bought this because it was on sale, and I wa…	
560024	B0030GNQMU	A17Z0PXGEGVYLY	0	0	1	Overpriced	I have written to this company to try to get a…	
243087	B005K4Q4KG	A2BZN96W6QP6XH	0	2	1	not quality at all	this product taste stale and is full of artifi…	
88745	B000F4BCS0	A20DNQ7AZD7XF4	0	0	1	Packing issues - Open bags	Half of the bags in case did not get sealed at…	

```
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…

          Summary_preprocessed  \
425093  disgusting product ever
380879        crazy people buy
560024              overpriced
243087                 quality
88745   packing issue  open bag

                                            Text_Summary
425093  arrived whole dog treat bag  br   bag filled l…
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…
```

[63]:
```
feature = exp_df['Text_Summary']
target = exp_df['Score']
```

[64]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
x= feature
y= le.fit_transform(target)

X_train_text_summary, X_test_text_summary, y_train_text_summary,␣
 ↪y_test_text_summary = train_test_split(x, y)
```

[65]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier

# Ref from [5] for pipelining and vectorizing
RFC_Text = Pipeline([('vectorizer_tfidf',TfidfVectorizer()),('RandomForest',␣
 ↪RandomForestClassifier())])

RFC_Text.fit(X_train_text_summary, y_train_text_summary)
y_pred_text_summary = RFC_Text.predict(X_test_text_summary)


print(classification_report(y_test_text_summary, y_pred_text_summary))
```

```
              precision    recall  f1-score   support
```

```
              0       0.95      0.97      0.96     28542
              1       0.99      0.97      0.98     28393
              2       0.97      0.94      0.95     28375
              3       0.92      0.84      0.88     28475
              4       0.83      0.94      0.88     28322

       accuracy                           0.93    142107
      macro avg       0.93      0.93      0.93    142107
   weighted avg       0.93      0.93      0.93    142107
```

1.2.3 Text with user id

[66]:
```
exp_df['text_user']= exp_df['Text_pre'] + " User " + str(exp_df['UserId'])
exp_df.head()
```

[66]:
```
         ProductId          UserId  HelpfulnessNumerator  \
425093  B000WNLC66  A1LD2S14L3AC1G                     0
380879  B0029NM6NU  A1ZORN2GEZBNV7                     0
560024  B0030GNQMU  A17ZOPXGEGVYLY                     0
243087  B005K4Q4KG  A2BZN96W6QP6XH                     0
88745   B000F4BCS0  A20DNQ7AZD7XF4                     0

        HelpfulnessDenominator  Score                        Summary  \
425093                       0      1  Most disgusting product ever!
380879                       3      1          Crazy people buy this!
560024                       0      1                      Overpriced
243087                       2      1              not quality at all
88745                        0      1     Packing issues - Open bags

                                                    Text  \
425093  When this arrived there were no whole dog trea…
380879  I bought this because it was on sale, and I wa…
560024  I have written to this company to try to get a…
243087  this product taste stale and is full of artifi…
88745   Half of the bags in case did not get sealed at…

                                                Text_pre  \
425093  arrived whole dog treat bag  br   bag filled l…
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…

           Summary_preprocessed  \
425093  disgusting product ever
380879         crazy people buy
```

```
560024               overpriced
243087                  quality
88745   packing issue  open bag

                                                 Text_Summary  \
425093  arrived whole dog treat bag  br   bag filled l…
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…

                                                    text_user
425093  arrived whole dog treat bag  br   bag filled l…
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…
```

[67]:
```
feature = exp_df['text_user']
target = exp_df['Score']
```

[68]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
x= feature
y= le.fit_transform(target)

X_train_user, X_test_user, y_train_user, y_test_user = train_test_split(x, y)
```

[69]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier

# Ref from [5] for pipelining and vectorizing
RFC_Text = Pipeline([('vectorizer_tfidf',TfidfVectorizer()),('RandomForest',␣
 ↪RandomForestClassifier())])

RFC_Text.fit(X_train_user, y_train_user)
y_pred_user = RFC_Text.predict(X_test_user)


print(classification_report(y_test_user, y_pred_user))
```

```
              precision    recall  f1-score   support

           0       0.95      0.95      0.95     28439
           1       0.99      0.97      0.98     28343
```

```
              2       0.97      0.93      0.95     28476
              3       0.92      0.83      0.87     28609
              4       0.79      0.93      0.86     28240

       accuracy                           0.92    142107
      macro avg       0.93      0.92      0.92    142107
   weighted avg       0.93      0.92      0.92    142107
```

1.2.4 text with helpfulness demnominator

[70]:
```
exp_df['experiment_likes']= exp_df['Text_pre'] + " Like" +␣
 ↪str(exp_df['HelpfulnessNumerator'])
exp_df.head()
```

[70]:
```
         ProductId          UserId  HelpfulnessNumerator  \
425093  B000WNLC66  A1LD2S14L3AC1G                     0
380879  B0029NM6NU  A1ZORN2GEZBNV7                     0
560024  B0030GNQMU  A17ZOPXGEGVYLY                     0
243087  B005K4Q4KG  A2BZN96W6QP6XH                     0
88745   B000F4BCS0  A20DNQ7AZD7XF4                     0

        HelpfulnessDenominator  Score                       Summary  \
425093                       0      1  Most disgusting product ever!
380879                       3      1         Crazy people buy this!
560024                       0      1                     Overpriced
243087                       2      1             not quality at all
88745                        0      1     Packing issues - Open bags

                                                    Text  \
425093  When this arrived there were no whole dog trea…
380879  I bought this because it was on sale, and I wa…
560024  I have written to this company to try to get a…
243087  this product taste stale and is full of artifi…
88745   Half of the bags in case did not get sealed at…

                                                Text_pre  \
425093  arrived whole dog treat bag  br   bag filled l…
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…

           Summary_preprocessed  \
425093  disgusting product ever
380879        crazy people buy
560024              overpriced
```

```
243087                  quality
88745   packing issue  open bag

                                                 Text_Summary  \
425093  arrived whole dog treat bag  br   bag filled l…
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…

                                                    text_user  \
425093  arrived whole dog treat bag  br   bag filled l…
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…

                                          experiment_likes
425093  arrived whole dog treat bag  br   bag filled l…
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…
```

[71]:
```
feature = exp_df['experiment_likes']
target = exp_df['Score']
```

[72]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
x= feature
y= le.fit_transform(target)

X_train_likes, X_test_likes, y_train_likes, y_test_likes = train_test_split(x,␣
 ↪y)
```

[73]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier

# Ref from [5] for pipelining and vectorizing
RFC_Text = Pipeline([('vectorizer_tfidf',TfidfVectorizer()),('RandomForest',␣
 ↪RandomForestClassifier())])

RFC_Text.fit(X_train_likes, y_train_likes)
y_pred_likes = RFC_Text.predict(X_test_likes)
```

```
print(classification_report(y_test_likes, y_pred_likes))
```

```
              precision    recall  f1-score   support

           0       0.96      0.95      0.95     28466
           1       0.99      0.97      0.98     28294
           2       0.97      0.93      0.95     28437
           3       0.92      0.84      0.88     28441
           4       0.80      0.93      0.86     28469

    accuracy                           0.92    142107
   macro avg       0.93      0.92      0.93    142107
weighted avg       0.93      0.92      0.93    142107
```

1.3 Using only nouns

[74]:
```
# removing all words other than nouns from tags feature ref from [7]

def pos(line):
    tempCol = ''
    posTag = nltk.pos_tag(line.split(' '))
    for (word, pos) in posTag:
        if(pos.startswith("NN")):
            tempCol += ' ' + word.strip()
    return tempCol

exp_df['TextNN'] = exp_df['Text'].apply(lambda x: pos(x))
exp_df['TextNN'] = exp_df['Text'].swifter.apply(lambda l: pre_processing(l))
exp_df.head()
```

```
Pandas Apply:   0%|          | 0/568425 [00:00<?, ?it/s]
```

[74]:

	ProductId	UserId	HelpfulnessNumerator	HelpfulnessDenominator	Score	Summary
425093	B000WNLC66	A1LD2S14L3AC1G	0	0	1	Most disgusting product ever!
380879	B0029NM6NU	A1Z0RN2GEZBNV7	0	3	1	Crazy people buy this!
560024	B0030GNQMU	A17Z0PXGEGVYLY	0	0	1	Overpriced
243087	B005K4Q4KG	A2BZN96W6QP6XH	0	2	1	not quality at all
88745	B000F4BCS0	A20DNQ7AZD7XF4	0	0	1	Packing issues - Open bags

```
                                                        Text  \
425093  When this arrived there were no whole dog trea…
380879  I bought this because it was on sale, and I wa…
560024  I have written to this company to try to get a…
243087  this product taste stale and is full of artifi…
88745   Half of the bags in case did not get sealed at…

                                                   Text_pre  \
425093  arrived whole dog treat bag  br   bag filled l…
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…

          Summary_preprocessed  \
425093  disgusting product ever
380879        crazy people buy
560024              overpriced
243087                 quality
88745   packing issue  open bag

                                               Text_Summary  \
425093  arrived whole dog treat bag  br   bag filled l…
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…

                                                 text_user  \
425093  arrived whole dog treat bag  br   bag filled l…
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…

                                           experiment_likes  \
425093  arrived whole dog treat bag  br   bag filled l…
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…

                                                     TextNN
425093  arrived whole dog treat bag  br   bag filled l…
380879  bought sale  helping motherinlaw save money  f…
560024  written company try get answer overcharged cof…
```

```
243087  product taste stale full artificial ingredient…
88745   half bag case get sealed bottom bag spilled in…
```

[75]:
```
feature = exp_df['TextNN']
target = exp_df['Score']
```

[76]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
x= feature
y= le.fit_transform(target)

X_train_nouns, X_test_nouns, y_train_nouns, y_test_nouns = train_test_split(x,␣
 ↪y)
```

[77]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier

# Ref from [5] for pipelining and vectorizing
RFC_Text = Pipeline([('vectorizer_tfidf',TfidfVectorizer()),('RandomForest',␣
 ↪RandomForestClassifier())])

RFC_Text.fit(X_train_nouns, y_train_nouns)
y_pred_nouns = RFC_Text.predict(X_test_nouns)

print(classification_report(y_test_nouns, y_pred_nouns))
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.95     28308
           1       0.99      0.97      0.98     28576
           2       0.98      0.93      0.95     28551
           3       0.93      0.84      0.88     28278
           4       0.81      0.93      0.87     28394

    accuracy                           0.93    142107
   macro avg       0.93      0.93      0.93    142107
weighted avg       0.93      0.93      0.93    142107
```

1.4 Experiment Results

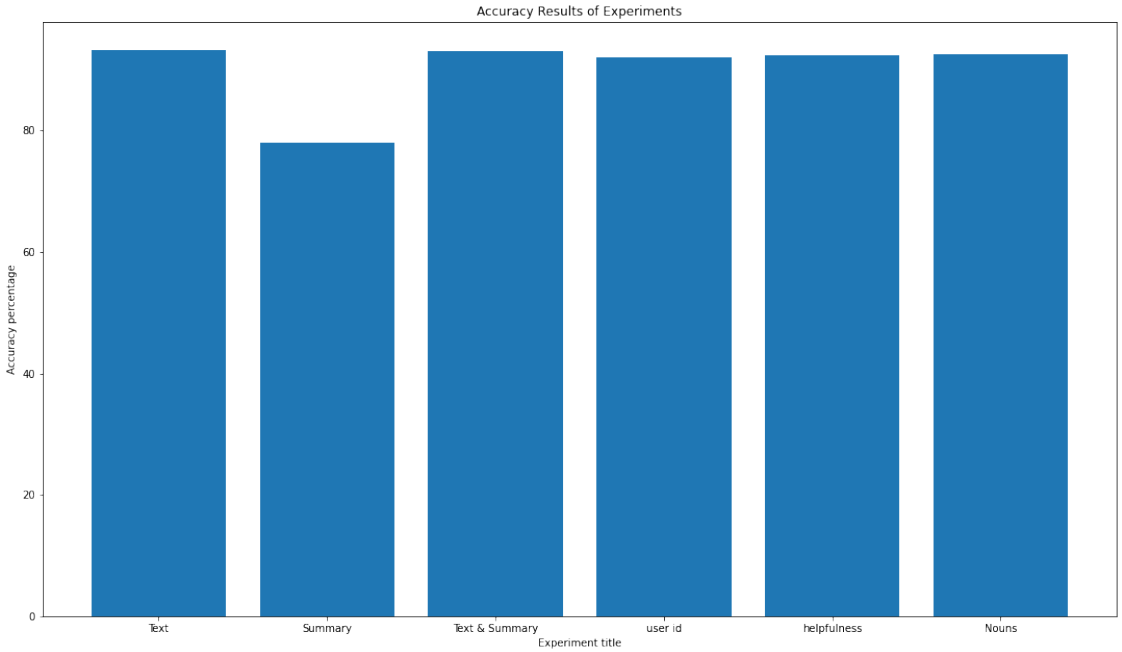
[102]:
```
from sklearn.metrics import accuracy_score

print('Accuracy Scores using text:')
print("93.22")
```

```
print()
print('Accuracy Scores using summary:')
print(str(round(accuracy_score(y_test_summary, y_pred_summary) * 100, 2)))
print()
print('Accuracy Scores using summary and text:')
print(str(round(accuracy_score(y_test_text_summary, y_pred_text_summary) * 100,␣
 ↪2)))
print()
print('Accuracy Scores using user id:')
print(str(round(accuracy_score(y_test_user, y_pred_user) * 100, 2)))
print()
print('Accuracy Scores using text with helpfullness numerator:')
print(str(round(accuracy_score(y_test_likes, y_pred_likes) * 100, 2)))
print()
print('Accuracy Scores using text with only nouns:')
print(str(round(accuracy_score(y_test_nouns, y_pred_nouns) * 100, 2)))
```

```
Accuracy Scores using text:
92.63

Accuracy Scores using summary:
77.92

Accuracy Scores using summary and text:
93.07

Accuracy Scores using user id:
92.09

Accuracy Scores using text with helpfullness numerator:
92.43

Accuracy Scores using text with only nouns:
92.62
```

[105]:
```
accuracy_experiments = []
accuracy_experiments.append(93.22)
accuracy_experiments.append(round(accuracy_score(y_test_summary,␣
 ↪y_pred_summary) * 100, 2))
accuracy_experiments.append(round(accuracy_score(y_test_text_summary,␣
 ↪y_pred_text_summary) * 100, 2))
accuracy_experiments.append(round(accuracy_score(y_test_user, y_pred_user) *␣
 ↪100, 2))
accuracy_experiments.append(round(accuracy_score(y_test_likes, y_pred_likes) *␣
 ↪100, 2))
accuracy_experiments.append(round(accuracy_score(y_test_nouns, y_pred_nouns) *␣
 ↪100, 2))
```

[148]:
```
import numpy as np
import matplotlib.pyplot as plt

# create dataset
experiments_titles = ['Text', 'Summary', 'Text & Summary', 'user id',␣
 ↪'helpfulness', 'Nouns']

# Create bars and choose color
plt.bar(experiments_titles, accuracy_experiments)

# Add title and axis names
plt.title('Accuracy Results of Experiments')
plt.xlabel('Experiment title')
plt.ylabel('Accuracy percentage')

fig = plt.gcf()
fig.set_size_inches(18.5, 10.5)

# Create names on the x axis
plt.xticks(experiments_titles)

# Show graph
plt.show()
```



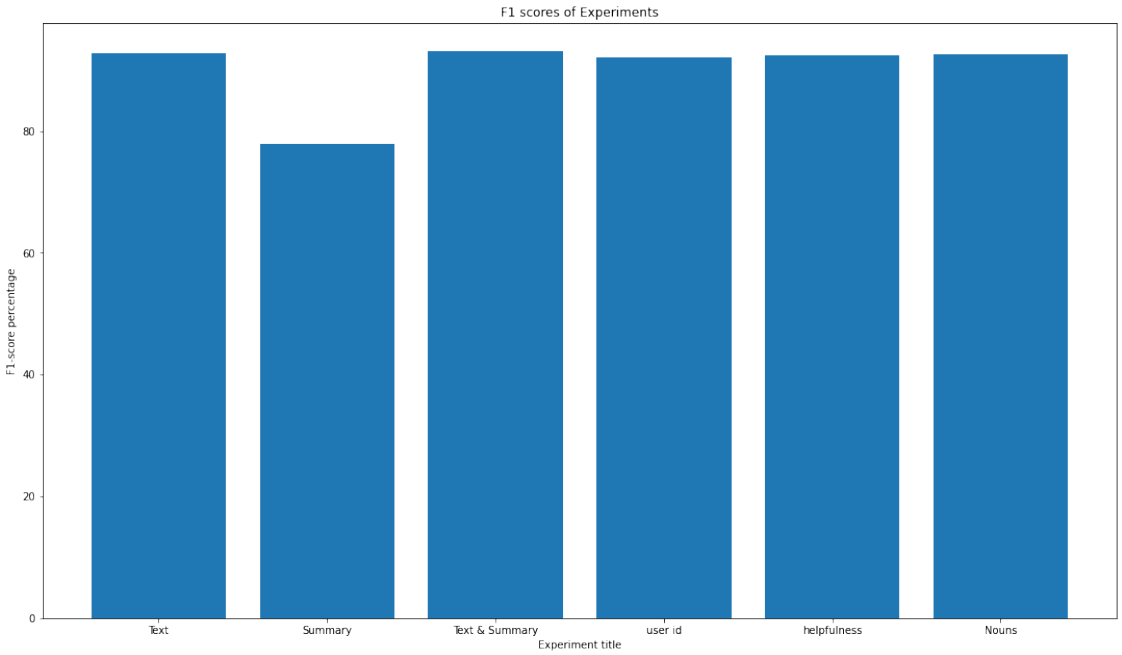
[103]:
```
from sklearn.metrics import f1_score

print('f1 Scores using text:')
print("92.69")
print()
print('f1 Scores using summary:')
print(str(round(f1_score(y_test_summary, y_pred_summary, pos_label='positive',␣
 ↪average='micro') * 100, 2)))
print()
print('f1 Scores using summary and text:')
print(str(round(f1_score(y_test_text_summary, y_pred_text_summary,␣
 ↪pos_label='positive', average='micro') * 100, 2)))
print()
print('f1 Scores using user id:')
print(str(round(f1_score(y_test_user, y_pred_user, pos_label='positive',␣
 ↪average='micro') * 100, 2)))
print()
print('f1 Scores using text with helpfullness numerator:')
print(str(round(f1_score(y_test_likes, y_pred_likes, pos_label='positive',␣
 ↪average='micro') * 100, 2)))
print()
print('f1 Scores using text with only nouns:')
print(str(round(f1_score(y_test_nouns, y_pred_nouns, pos_label='positive',␣
 ↪average='micro') * 100, 2)))
```

```
f1 Scores using text:
92.69

f1 Scores using summary:
77.92

f1 Scores using summary and text:
93.07

f1 Scores using user id:
92.09

f1 Scores using text with helpfullness numerator:
92.43

f1 Scores using text with only nouns:
92.62
```

[110]:
```
f1_experiments = []
f1_experiments.append(92.69)
f1_experiments.append(round(f1_score(y_test_summary, y_pred_summary,␣
 ↪pos_label='positive', average='micro') * 100, 2))
```

```
f1_experiments.append(round(f1_score(y_test_text_summary, y_pred_text_summary,␣
 ↪pos_label='positive', average='micro') * 100, 2))
f1_experiments.append(round(f1_score(y_test_user, y_pred_user,␣
 ↪pos_label='positive', average='micro') * 100, 2))
f1_experiments.append(round(f1_score(y_test_likes, y_pred_likes,␣
 ↪pos_label='positive', average='micro') * 100, 2))
f1_experiments.append(round(f1_score(y_test_nouns, y_pred_nouns,␣
 ↪pos_label='positive', average='micro') * 100, 2))
```

[149]:
```
import numpy as np
import matplotlib.pyplot as plt

# create dataset
experiments_titles = ['Text', 'Summary', 'Text & Summary', 'user id',␣
 ↪'helpfulness', 'Nouns']

# Create bars and choose color
plt.bar(experiments_titles, f1_experiments)

# Add title and axis names
plt.title('F1 scores of Experiments')
plt.xlabel('Experiment title')
plt.ylabel('F1-score percentage')

fig = plt.gcf()
fig.set_size_inches(18.5, 10.5)

# Create names on the x axis
plt.xticks(experiments_titles)

# Show graph
plt.show()
```



[1]:
```
print("                          XGBoost                KNN                 ␣
↪Randomforest               LogisticRegression")
print("mean score                0.618                  0.708                  0.
↪932                        0.604")
print("F1Score                   0.620                  0.700                  0.
↪930                        0.600")#F1 score has been refered from␣
↪notebook-part2 file
print("standard deviation        0.002                  0.035                  0.
↪0004                       0.004")
```

```
                          XGBoost                KNN
Randomforest               LogisticRegression
mean score                0.618                  0.708                  0.932
0.604
F1Score                   0.620                  0.700                  0.930
0.600
standard deviation        0.002                  0.035                  0.0004
0.004
```

1.4.1 References

[1] - https://dal.brightspace.com/d2l/le/content/232269/viewContent/3256445/View

[2] - https://pandas.pydata.org/docs/reference/general_functions.html

[3] - https://miamioh.instructure.com/courses/38817/pages/data-cleaning

[4] - https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/

[5] - https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[6] - https://www.nltk.org/_modules/nltk/stem/wordnet.html

[7] - https://www.geeksforgeeks.org/bag-of-words-bow-model-in-nlp/

[8] - https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html