```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of
```

```
/kaggle/input/aid-escalating-internet-coverage/sample_submission.csv
/kaggle/input/aid-escalating-internet-coverage/train.csv
/kaggle/input/aid-escalating-internet-coverage/test.csv
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/5
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/3
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/4
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/9
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/7
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/54
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/10
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/10
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/49
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/59
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/14
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/5
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/1
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/5
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/24
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/9
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/89
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/3
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/10
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/64
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/1
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/8
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/6
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/9
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/17
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/18
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/1
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/7
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/6
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/40
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/10
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/3
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/3
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/10
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/84
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/20
```

```
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/1
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/5
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/9
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/1
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/3
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/9
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/5
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/5
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/6
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/1
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/1
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/2
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/4
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/4
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/7
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/6
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/4
/kaggle/input/aid-escalating-internet-coverage/page_information/page_information/7
```

EDA

```
from sklearn.preprocessing import RobustScaler
import seaborn as sns
import matplotlib.pyplot as plt
```

description of dataset

link: URL of the webpage to be classified

page_description: Description of the webpage

alchemy_category: Alchemy category (per the publicly available Alchemy API found at www.alchemyapi.com)

alchemy_category_score: Alchemy category score (per the publicly available Alchemy API found at www.alchemyapi.com)

avg_link_size: Average number of words in a webpage

common_word_link_ratio_1: # of links sharing at least 1 word with 1 other links / # of links

common_word_link_ratio_2: # of links sharing at least 1 word with 2 other links / # of links

common_word_link_ratio_3: # of links sharing at least 1 word with 3 other links / # of links

common_word_link_ratio_4: # of links sharing at least 1 word with 4 other links / # of links

compression_ratio: Measure of redundancy computed by finding the compression achieved on this web page via gzip

embed_ratio: Count of tags or simply the number of usages.

frame_based: Binary indication of whether a webpage has frameset markup

frame_tag_ratio: Ratio of frameset markups over total markups

has_domain_link: Binary indication of whether the webpage contains in URL with a domain

html_ratio: Ratio of tags vs text on the page

image_ratio: Ratio of tags vs text in the page

is_news: This is true(1) if this webpage is news

lengthy_link_domain: This is true (1) if the webpage's text contains more than 30 alpha-numeric characters

link_word_score: Percentage of words on the webpage that are also in the hyperlink text

news_front_page: True (1) if StumbleUpon's news classifier determines that this webpage is front-page news

non_markup_alphanumeric_characters: Number of alpha-numeric characters in webpage's text

count_of_links: Number of markups

number_of_words_in_url: Number of words in URL

parametrized_link_ratio: A link is parametrized if its URL contains parameters or has an attached onClick event

spelling_mistakes_ratio: Ratio of words not found in the wiki (considered to be a spelling mistake)

label: The label value of 0 represents that the webpage is not "ad-worthy", and a label value of 1 represents that the webpage is "ad-worthy". This is available only for train.csv.

```
# replacing missing values with nan

missing_values = ['?','--','-','??','.','unknown']
train_data=pd.read_csv("../input/aid-escalating-internet-coverage/train.csv",na_values=mis
test_data=pd.read_csv("../input/aid-escalating-internet-coverage/test.csv",na_values=missi



## merging train and test csv for applying preprocessing to combine dataset
# data = pd.concat(frames)
data=train_data.copy()
data2=test_data.copy()
# data.shape
train=data.drop(labels=["label"],axis=1)
print(train.shape)
test=data2
print(test.shape)
ultdf=[train,test]
finaldf=pd.concat(ultdf).reset_index(drop=True)
print(finaldf.shape)
```

```
(4437, 26)
(2958, 26)
(7395, 26)
```

finaldf is the combined entire dataset with all '?' replaced with NaN

Checking the page description for a link ID

```
finaldf.loc[finaldf["link_id"]==10109]["page_description"]
```

```
298    {"title":"BBC NEWS Health Depression link to p...
Name: page_description, dtype: object
```

```
#columns / features with na values
finaldf.columns[finaldf.isna().any()].tolist()
```

```
['alchemy_category', 'alchemy_category_score', 'is_news', 'news_front_page']
```

```
finaldf["is_news"].value_counts()
#is_news is only having null or 1
```

```
1.0    4552
Name: is_news, dtype: int64
```

```
finaldf["frame_based"].value_counts()
#frame_based has only 0
```

```
0    7395
Name: frame_based, dtype: int64
```

```
#dropping frame_based
finaldf.drop(labels=["frame_based"],axis=1,inplace=True)
```

```
for c in finaldf.columns:
    print(finaldf[c].unique())
```

```
   59  78  79  71  74  63  50  60  53  83  87  75  67  61  70  90  64  73
   80  93  95  97  81 100  94  99  89  98  96]
 [ 0.  1. nan]
 [ 1236  3887   780 ... 25467  3380   356]
 [  98  230   75  192  407   57  142  137  116  134  110  333  266    5
   389  101  133  184  130   62   84  317  394   89  106  124  102  128
    99  255   30  176  576  516  197  150  143  243  159  288   16  664
    97  277  153  264   74   80  131  380  588   94   93  425  285   46
    37  194   20   71  830  119   31  183   43  338  471  114  391   88
   175   53 1112  321  140  238   95  165  111  206   77  258  680   35
   236  319  127   58  103  306  191  377   25  744  392   96  793   18
    91  225   61   70  227  444  281  684    8  357  162  122  208  295
   270  195  117  339   81  249  300  151  224  528  196   73  216  218
   250  149  482   33   66    9  602    3  233  240  125   60  155   72
    26  186  767   11  323  178   67  345  212  205   82  169  305  136
    29  138  132    1  199  246  120  158   22  105  200  279  113  463
```

```
  86   76  115  146  633  404  568  296  126  135  172  865  232  294
 367  193  174  123  203   32  273  121   13  402  152  214   10  109
 326  410  185  156 1042  283   55   69   24  330  179  388  359   65
  83  163  313    7  354  198   87   42  545  177   27  202  187  370
  92   19  293   51  141  248  470   48  228  271   44  100  220  309
   2   54   63  112  139   41 1050  211  396  217  678  173  324    4
  79  526  310  108  496  260  265  337  213  403  171  157  104  269
  52  427   59  358   85  518   45  438  262  435  182  147  442  325
  34   28  107  257  416  426   12  280  154  348  161  472   90  316
 541  190  180  189   68  145  234   14  256  554  247   39  263  381
 160   50  201   15  428  537  129  361  371  994  507   17  421  512
 681  456  166   36 1114   47  118  188  360  382  460  298  167  502
 490  461  219  253   78  261  353  291   38  278  488  207  327  572
 362  181  297  332  292  303  500   64  210  366  204  168  221  620
 351  368  477  244  441  373  424  665  144  315  259  647  774 1027
 311   21  990  267  349  459  483  440  405  363  384  449  777  510
2371  387  709  401  598  274  497  322  148  515  950  501  334  164
 514  226   49  451  301  268  245  552  286  254   23  209  239  476
 750 1256  893  668  543  452  242  567  627  383  688  726  350  766
 369  536  561  241  329  312  639  346  289  231  429  849  170    6
 222  547  390  713  251  320  386 1168  287  275   56  400  690  229
  40  432  235  499  430  433  409  755  290  787  498  439  379  276
 385  629  604  768  789  607  826  314 1766  395  331  252  689  644
 356  489  419  453  475  519  550  458  653  445  437  411  508  679
 484  455  687  578  364  215  570  648  378  542  881  577  479  343
 864  340  237  336  398  223  494 1612  480  418  399  584  328  318
 638  302  723  551  654  447  308  304 1137  624  355  454  415  806
 406  945  517  524  880  272  670  640  465  921  511  731  673  282
 646  344  436  677  871  534  852  412  374  413  443  365  299  623
 619  749  431  529  481 1761  284  376  671  527  375  522  492  840
 521  491  417 1502  599  663  631  798  585  530  506  474  560  397
 838  786  600  694  307 1507  686  509  446  504  473  556 1082  468
 493  809  667  935  549  587  450 4997  645  342  592  523  845  341
 347  608  335  548  613 1158  597 4990  566  603  485  707 1057  676
 352  535  544  761  888  984  505  503 1111  618  612  559  420 1125
 724  414  747  513 1054  831  813  486  642  820 1546  594  692  601
 372  752  745  821 3283  875  635  408 1033 1006  782  609  532  847
 822  538  448  628  657  696 1064 1237  422  894  571 2406  712  989
 661  801]
[ 8  6 11  0  1  5  2  7  4  9 14  3 10 12 13 19 16 15 22 21 20 17 18]
[0.06122449 0.33043478 0.16       ... 0.47979798 0.00761421 0.01345291]
[0.07612457 0.13074205 0.07647059 ... 0.08349901 0.12086514 0.12888199]
```

```
finaldf.nunique()
```

```
link                            7395
link_id                         7395
page_description                7394
alchemy_category                  12
alchemy_category_score          4805
avg_link_size                   5710
common_word_link_ratio_1        4476
common_word_link_ratio_2        4038
common_word_link_ratio_3        3266
common_word_link_ratio_4        2695
compression_ratio               6453
embed_ratio                      366
frame_tag_ratio                 5911
has_domain_link                    2
```

```
html_ratio                          7376
image_ratio                         5418
is_news                                1
lengthy_link_domain                    2
link_word_score                      101
news_front_page                        2
non_markup_alphanumeric_characters  5301
count_of_links                       702
number_of_words_in_url                23
parametrized_link_ratio             3922
spelling_mistakes_ratio             4219
dtype: int64
```

checking for duplicate rows

```
print(finaldf.duplicated().sum())
```

```
0
```

checking numerical features

```
finaldf.describe()
```

	link_id	alchemy_category_score	avg_link_size	common_word_link_ratio_1
count	7395.000000	5053.000000	7395.000000	7395.000000
mean	5305.704665	0.603334	2.761823	0.468230
std	3048.384114	0.212864	8.619793	0.203133
min	1.000000	0.070833	0.000000	0.000000
25%	2688.500000	0.452424	1.602062	0.340370
50%	5304.000000	0.625616	2.088235	0.481481
75%	7946.500000	0.780851	2.627451	0.616604
max	10566.000000	0.999426	363.000000	1.000000

8 rows × 22 columns

```
finaldf['has_domain_link'].value_counts()
```

```
0    7238
1     157
Name: has_domain_link, dtype: int64
```

```
finaldf['html_ratio'].describe()
```

```
count    7395.000000
mean        0.233778
std         0.052487
min         0.045564
25%         0.201061
50%         0.230564
75%         0.260770
max         0.716883
Name: html_ratio, dtype: float64
```

```
finaldf['has_domain_link'].value_counts()
```

```
0    7238
1     157
Name: has_domain_link, dtype: int64
```

'has_domain_link' is highly imbalanced

imbalance is very apparent we can test the model with and without this feature..

```
finaldf['lengthy_link_domain'].value_counts()
```

```
1    4883
0    2512
Name: lengthy_link_domain, dtype: int64
```

Checking the correlation among the features

```
finaldf.corr()
```

	link_id	alchemy_category_score	avg_link_size
link_id	1.000000	0.007229	-0.011162
alchemy_category_score	0.007229	1.000000	-0.014264
avg_link_size	-0.011162	-0.014264	1.000000
common_word_link_ratio_1	0.002856	0.118382	0.120467
common_word_link_ratio_2	0.008407	0.075179	0.161769
common_word_link_ratio_3	0.005285	0.043958	0.174554
common_word_link_ratio_4	0.009573	0.025410	0.134527
compression_ratio	-0.007343	-0.154228	-0.003578
embed_ratio	0.013340	0.125943	0.005254
frame_tag_ratio	0.010065	-0.069614	-0.049270
has_domain_link	-0.005802	0.027408	-0.002046
html_ratio	0.016989	-0.015281	0.018974
image_ratio	-0.000590	-0.044715	-0.003002
is_news	NaN	NaN	NaN

common_word_link_ratio_1 ,common_word_link_ratio_2 common_word_link_ratio_3 are highly correlated and also, common_word_link_ratio_3 are highly correlated with common_word_link_ratio_4 and embedded ratio and commpressed ratio are highly correlated

```
finaldf.drop(labels=["embed_ratio", "common_word_link_ratio_2", "common_word_link_ratio_3"
```

number of words in url	-0.017342	0.183589	-0.033890

```
finaldf.describe()
```

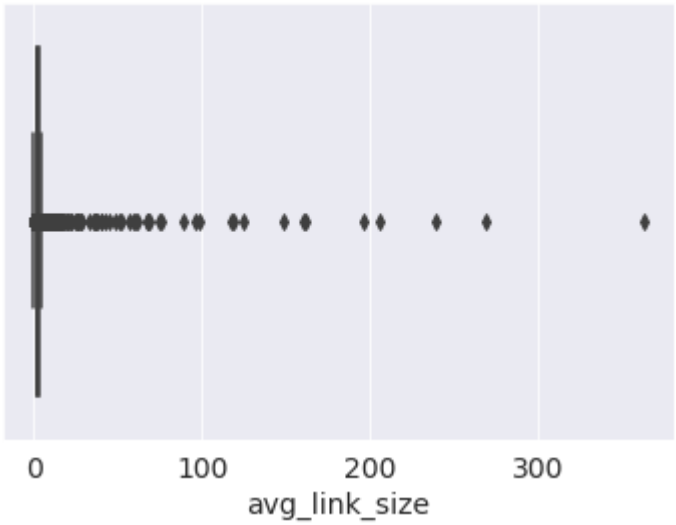
	link_id	alchemy_category_score	avg_link_size	common_word_link_ratio_1
count	7395.000000	5053.000000	7395.000000	7395.000000
mean	5305.704665	0.603334	2.761823	0.468230
std	3048.384114	0.212864	8.619793	0.203133
min	1.000000	0.070833	0.000000	0.000000
25%	2688.500000	0.452424	1.602062	0.340370
50%	5304.000000	0.625616	2.088235	0.481481
75%	7946.500000	0.780851	2.627451	0.616604
max	10566.000000	0.999426	363.000000	1.000000

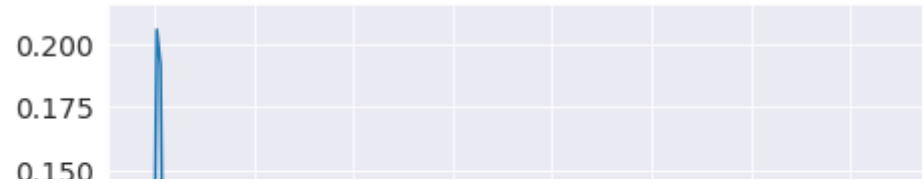
outlier detection and removal

Functions for OutlierPlot and IQR Method of Outlier Detection

```
def outlierPlot(p):
    sns.boxplot(x=p)
    plt.figure(figsize=(16,5))
    plt.subplot(1,2,1)
    sns.distplot(p)

#function for getting upper and lower limit
def limits(columnName):
    percentile25 = columnName.quantile(0.25)
    percentile75 = columnName.quantile(0.75)
    iqr = percentile75 - percentile25
    upper_limit = percentile75 + 1.5 * iqr
    lower_limit = percentile25 - 1.5 * iqr
    return upper_limit,lower_limit

#function caps i.e outlier removal
def capping(df,upper_limit,lower_limit,column_name):
    new_df_cap = df.copy()
    new_df_cap[column_name] = np.where(
    new_df_cap[column_name] > upper_limit,
    upper_limit,
    np.where(
        new_df_cap[column_name] < lower_limit,
        lower_limit,
        new_df_cap[column_name]
        )
    )
    return new_df_cap[column_name]



outlierPlot(finaldf['avg_link_size'])
```

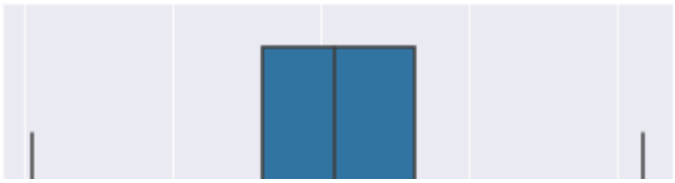
```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```





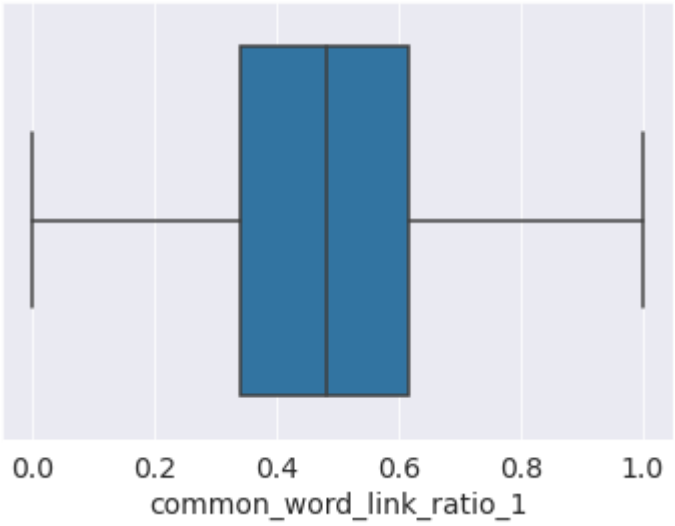

```
upper_limit,lower_limit = limits(finaldf['avg_link_size'])
finaldf['avg_link_size'] = capping(finaldf,upper_limit,lower_limit,'avg_link_size')
outlierPlot(finaldf['avg_link_size'])
```

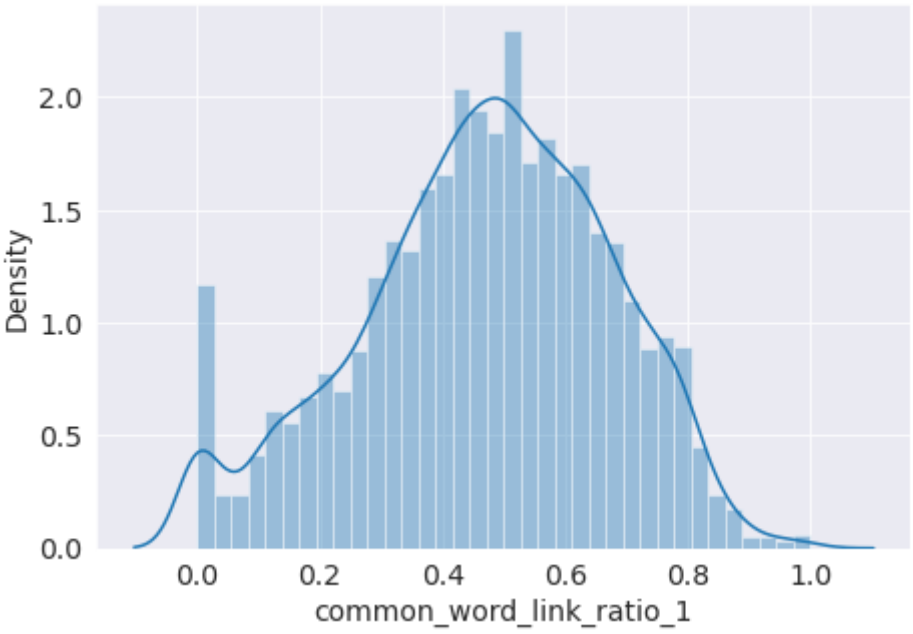
```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```



```
outlierPlot(finaldf['common_word_link_ratio_1'])
#no outliers found
```

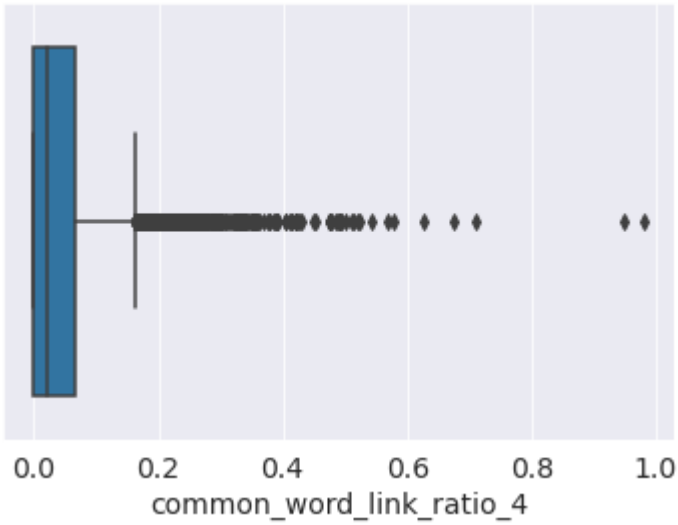
```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```

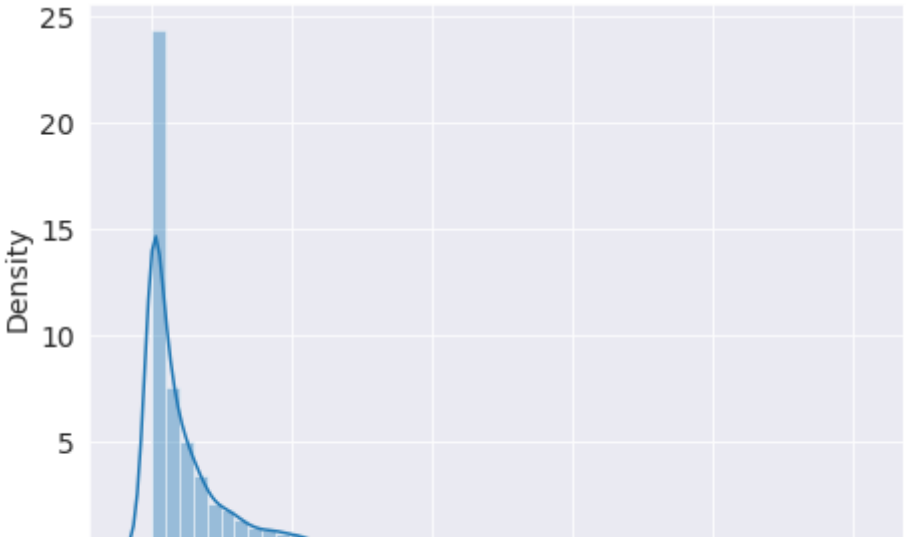





```
outlierPlot(finaldf['common_word_link_ratio_4'])
```

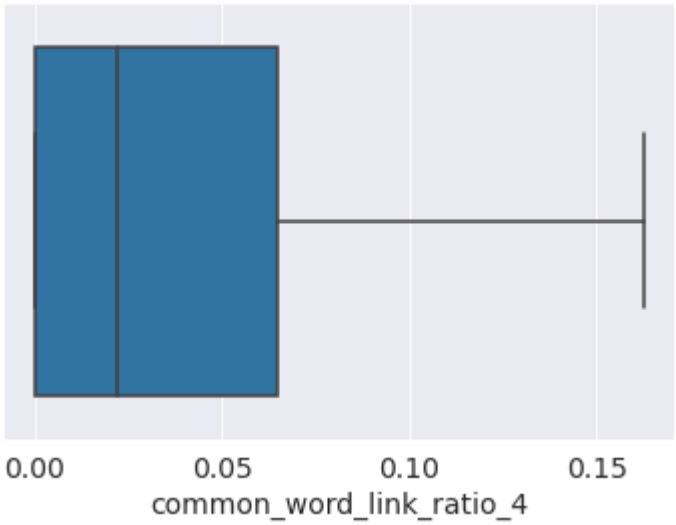
```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```





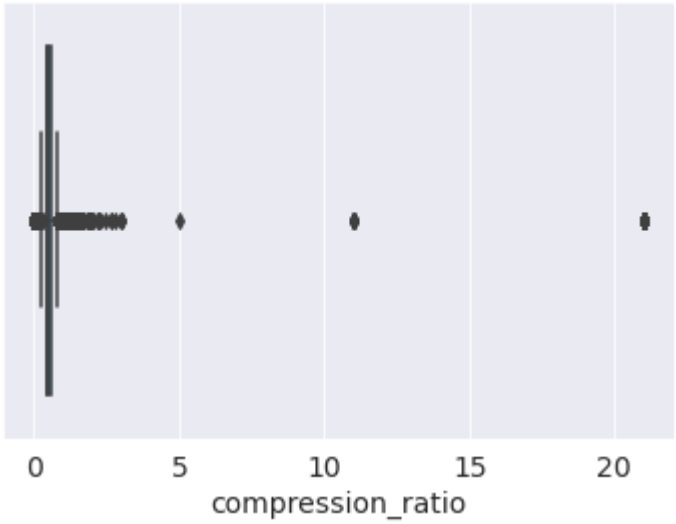

```
upper_limit,lower_limit = limits(finaldf['common_word_link_ratio_4'])
finaldf['common_word_link_ratio_4'] = capping(finaldf,upper_limit,lower_limit,'common_word
outlierPlot(finaldf['common_word_link_ratio_4'])
```

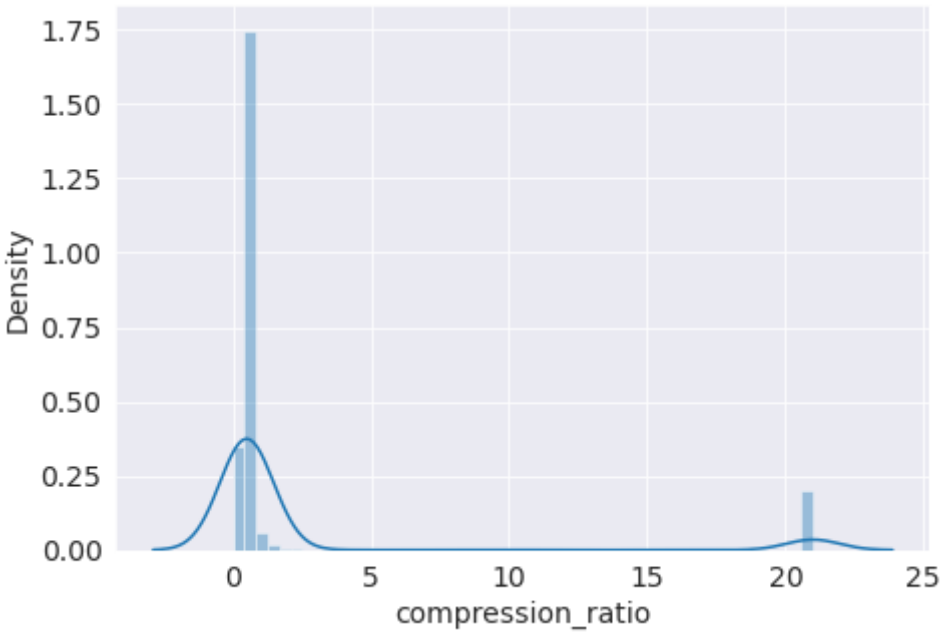
```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```




```
outlierPlot(finaldf['compression_ratio'])
```

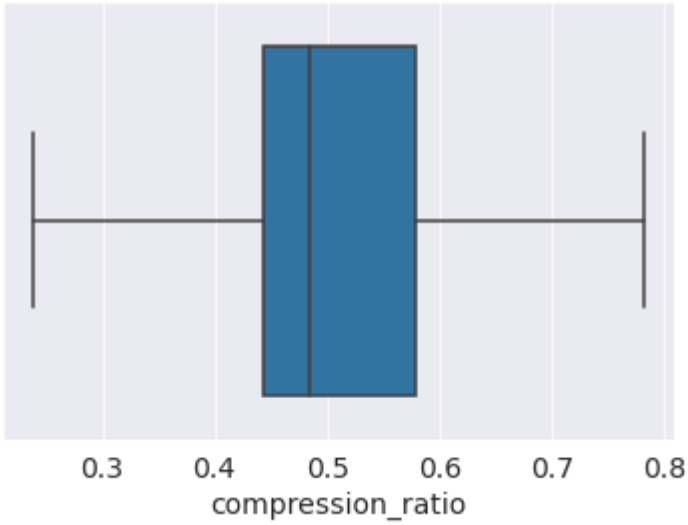
```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```

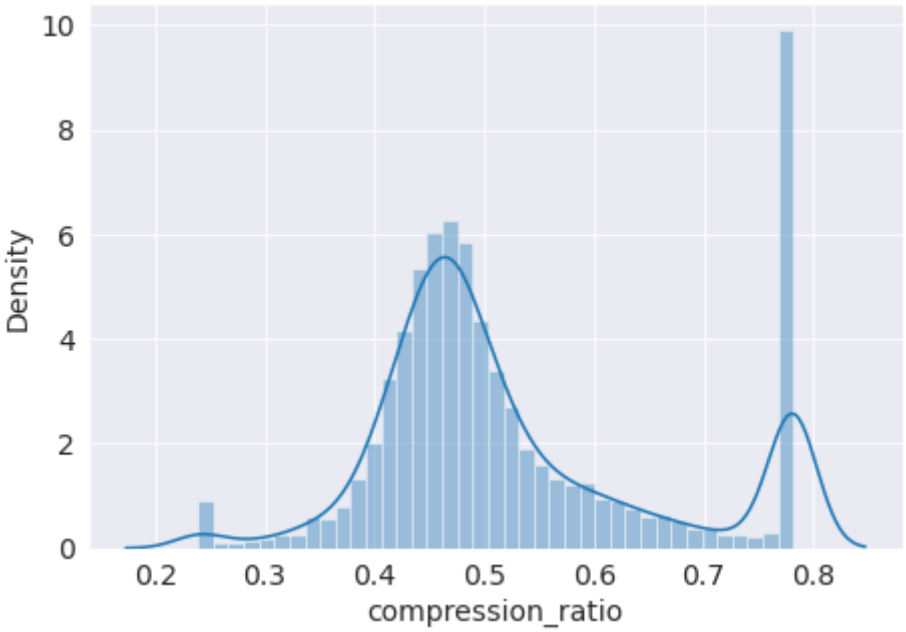




```
upper_limit,lower_limit = limits(finaldf['compression_ratio'])
finaldf['compression_ratio'] = capping(finaldf,upper_limit,lower_limit,'compression_ratio'
outlierPlot(finaldf['compression_ratio'])
```

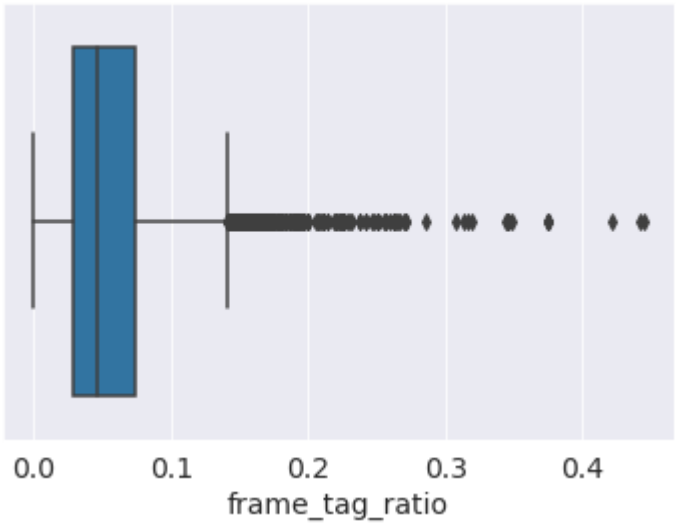
```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```

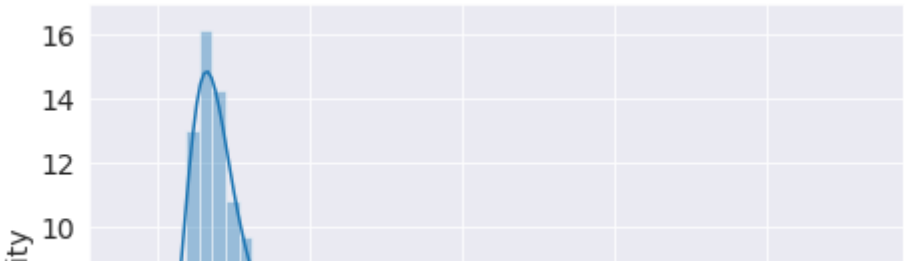





```
outlierPlot(finaldf['frame_tag_ratio'])
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```



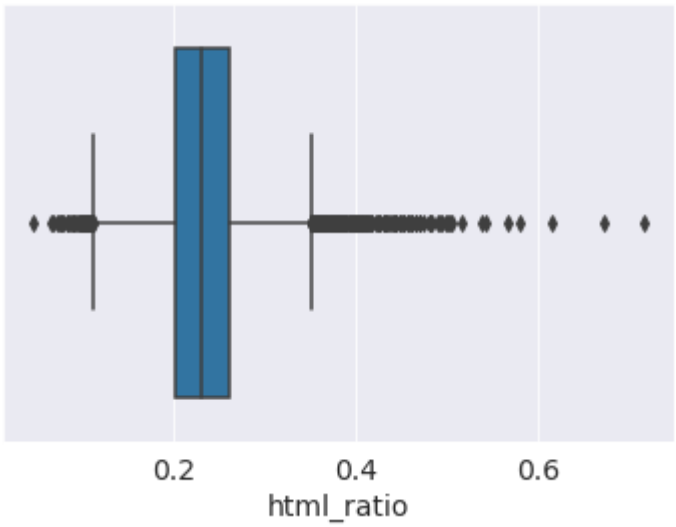


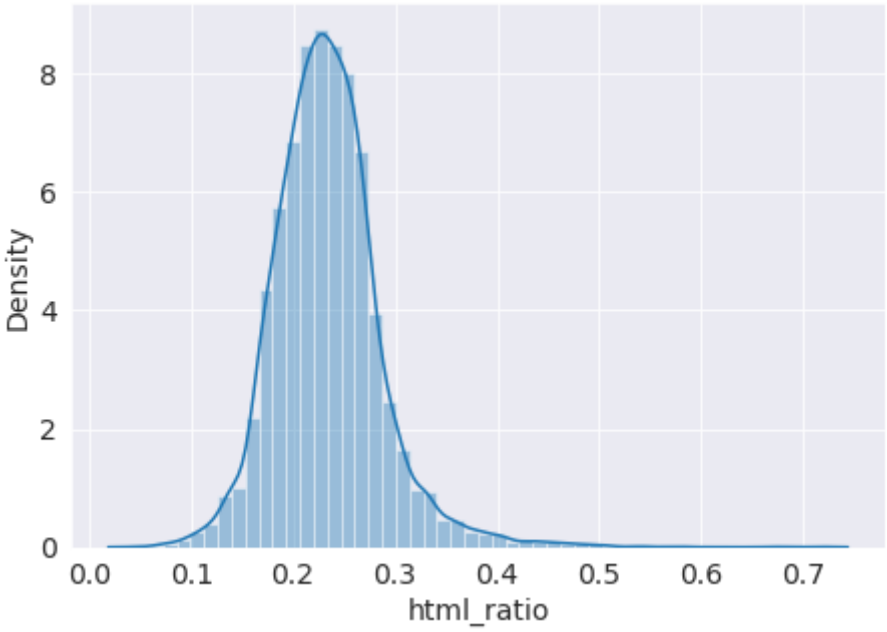

```
upper_limit,lower_limit = limits(finaldf['frame_tag_ratio'])
finaldf['frame_tag_ratio'] = capping(finaldf,upper_limit,lower_limit,'frame_tag_ratio')
outlierPlot(finaldf['frame_tag_ratio'])
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```

```
outlierPlot(finaldf['html_ratio'])
```

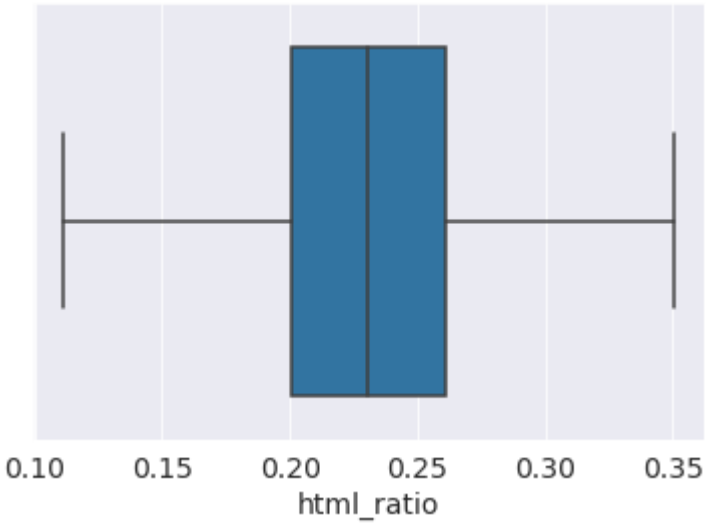
```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```

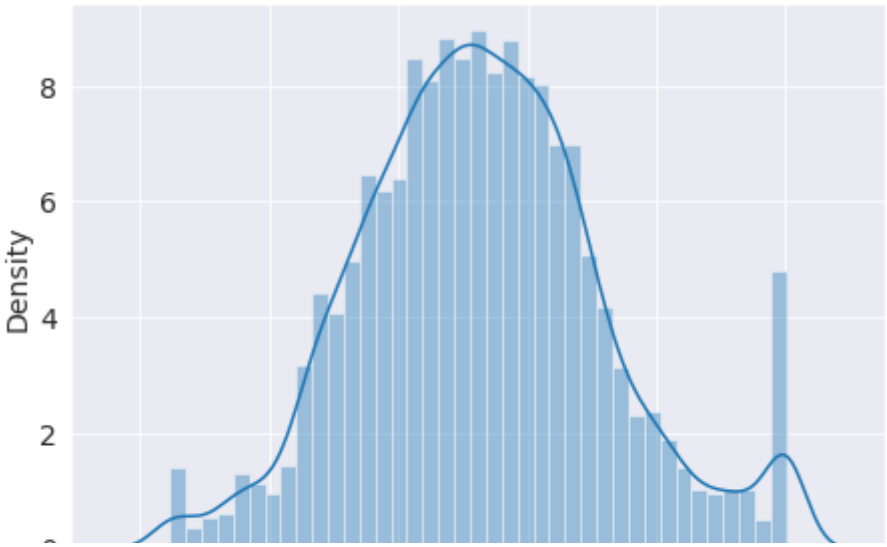





```
upper_limit,lower_limit = limits(finaldf['html_ratio'])
finaldf['html_ratio'] = capping(finaldf,upper_limit,lower_limit,'html_ratio')
outlierPlot(finaldf['html_ratio'])
```

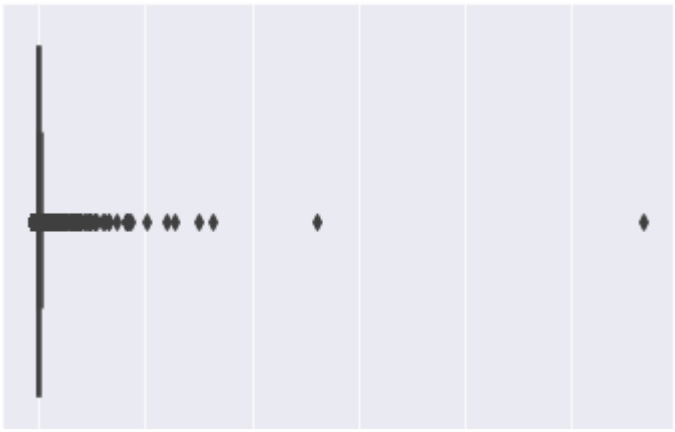
```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```





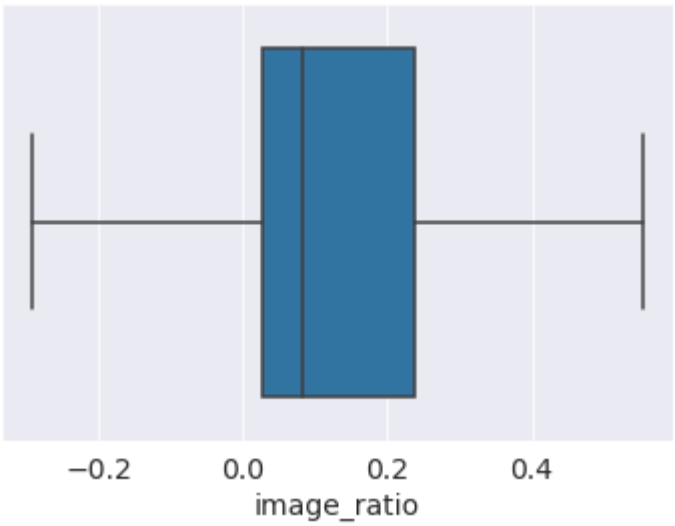

```
outlierPlot(finaldf['image_ratio'])
```

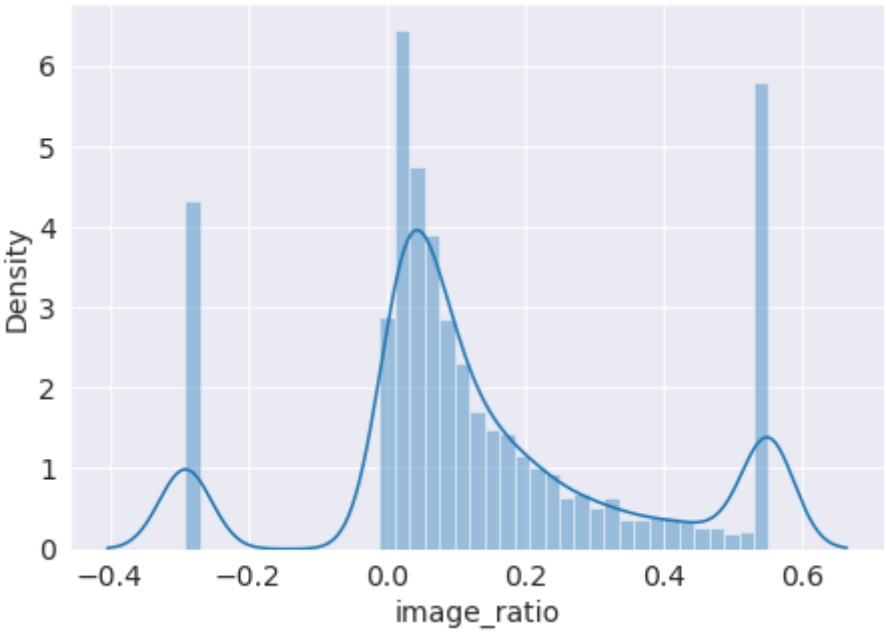
```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```



```
upper_limit,lower_limit = limits(finaldf['image_ratio'])
finaldf['image_ratio'] = capping(finaldf,upper_limit,lower_limit,'image_ratio')
outlierPlot(finaldf['image_ratio'])
```

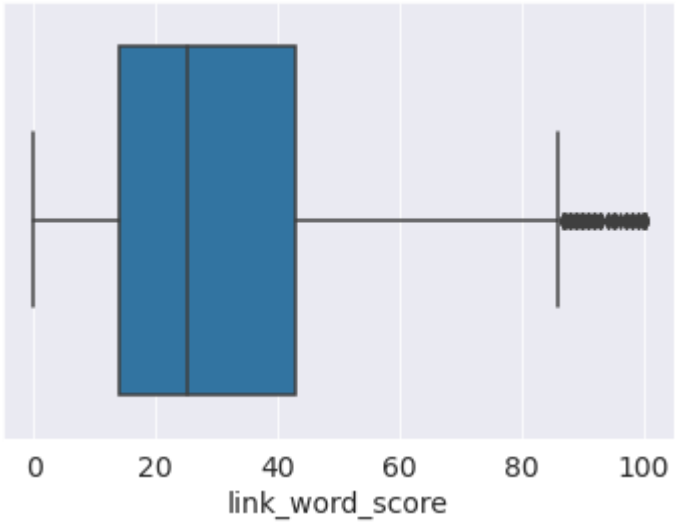
```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```

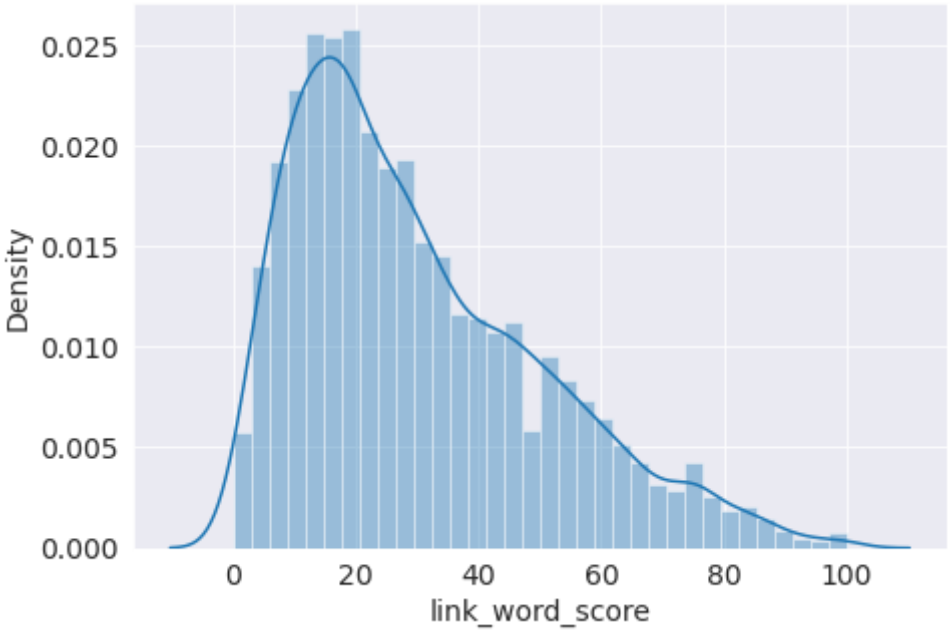




```
outlierPlot(finaldf['link_word_score'])
```

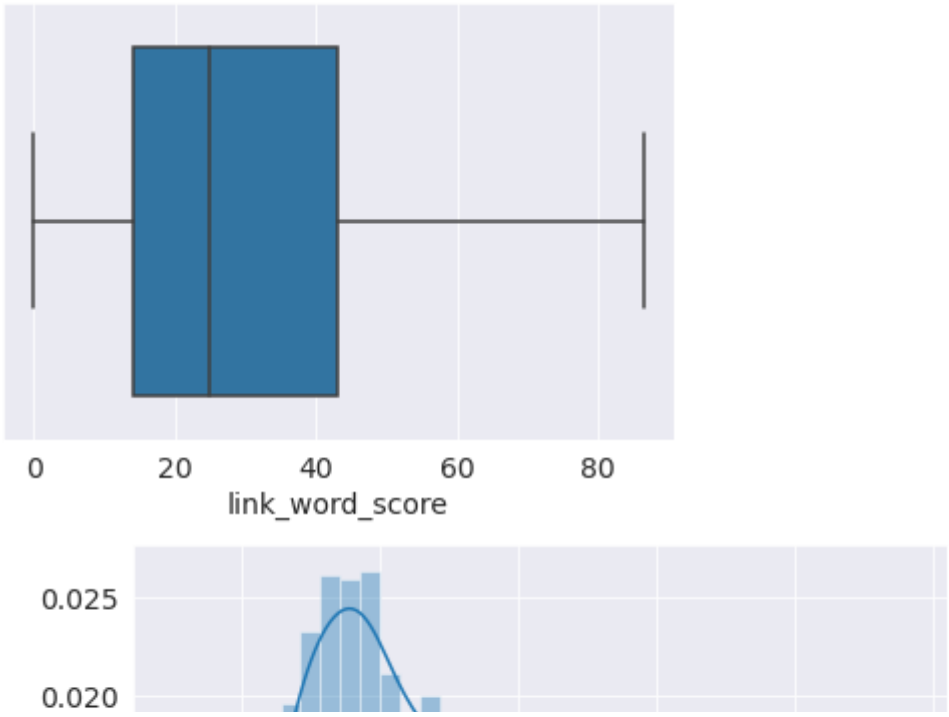
```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```






```
upper_limit,lower_limit = limits(finaldf['link_word_score'])
finaldf['link_word_score'] = capping(finaldf,upper_limit,lower_limit,'link_word_score')
outlierPlot(finaldf['link_word_score'])
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```

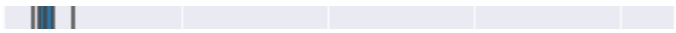



```
outlierPlot(finaldf['non_markup_alphanumeric_characters'])
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```



```
df=finaldf
```



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7395 entries, 0 to 7394
Data columns (total 22 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   link                                  7395 non-null   object 
 1   link_id                               7395 non-null   int64  
 2   page_description                      7395 non-null   object 
 3   alchemy_category                      5047 non-null   object 
 4   alchemy_category_score                5053 non-null   float64
 5   avg_link_size                         7395 non-null   float64
 6   common_word_link_ratio_1              7395 non-null   float64
 7   common_word_link_ratio_4              7395 non-null   float64
 8   compression_ratio                     7395 non-null   float64
 9   frame_tag_ratio                       7395 non-null   float64
 10  has_domain_link                       7395 non-null   int64  
 11  html_ratio                            7395 non-null   float64
 12  image_ratio                           7395 non-null   float64
 13  is_news                               4552 non-null   float64
 14  lengthy_link_domain                   7395 non-null   int64  
 15  link_word_score                       7395 non-null   float64
 16  news_front_page                       6147 non-null   float64
 17  non_markup_alphanumeric_characters    7395 non-null   int64  
 18  count_of_links                        7395 non-null   int64  
 19  number_of_words_in_url                7395 non-null   int64  
 20  parametrized_link_ratio               7395 non-null   float64
 21  spelling_mistakes_ratio               7395 non-null   float64
dtypes: float64(13), int64(6), object(3)
memory usage: 1.2+ MB
```

```
for i in df.columns:
    print(i)
```

```
link
link_id
page_description
alchemy_category
alchemy_category_score
avg_link_size
common_word_link_ratio_1
common_word_link_ratio_4
compression_ratio
frame_tag_ratio
has_domain_link
html_ratio
image_ratio
is_news
lengthy_link_domain
link_word_score
```

```
      news_front_page
      non_markup_alphanumeric_characters
      count_of_links
      number_of_words_in_url
      parametrized_link_ratio
      spelling_mistakes_ratio


vec=["avg_link_size",
"common_word_link_ratio_1",
"common_word_link_ratio_4",
"compression_ratio",
"frame_tag_ratio",
"has_domain_link",
"html_ratio",
"image_ratio",
"is_news",
"lengthy_link_domain",
"link_word_score",
"news_front_page",
"non_markup_alphanumeric_characters",
"count_of_links",
"number_of_words_in_url",
"parametrized_link_ratio",
"spelling_mistakes_ratio"]
```

Here we used Robust Scallar. This Scaler removes the median and scales the data according to the quantile range.

```
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()
for feature in vec:
        scaler.fit(df[[feature]])
        df[feature] = scaler.transform(df[[feature]])


df.describe().T
```

	count	mean	std	min	
link_id	7395.0	5305.704665	3048.384114	1.000000	2688.5
alchemy_category_score	5053.0	0.603334	0.212864	0.070833	0.4
avg_link_size	7395.0	0.085805	0.846056	-1.974135	-0.4
common_word_link_ratio_1	7395.0	-0.047970	0.735367	-1.743019	-0.5
common_word_link_ratio_4	7395.0	0.319489	0.789160	-0.341540	-0.3
compression_ratio	7395.0	0.303308	0.958575	-1.802802	-0.3
frame_tag_ratio	7395.0	0.192394	0.761468	-1.018200	-0.3
has_domain_link	7395.0	0.021231	0.144162	0.000000	0.0
html_ratio	7395.0	0.031428	0.789611	-1.994120	-0.4
image_ratio	7395.0	0.258491	1.060416	-1.771112	-0.2

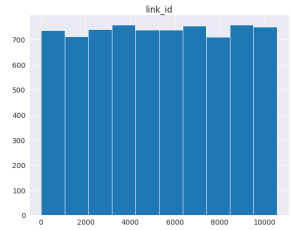
```
#graph here

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

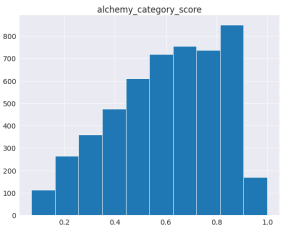
news_front_page	6147.0	0.047828	0.213420	0.000000	0.0

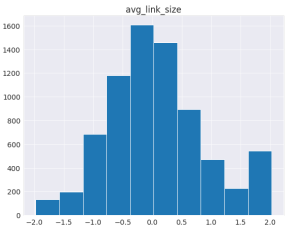
```
sns.set_style("darkgrid")
plt.rcParams['font.size'] = 14
plt.rcParams['figure.facecolor'] = '#00000000'
```

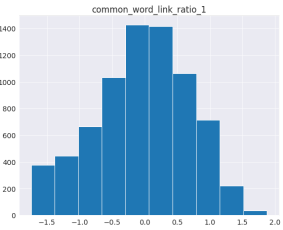
number of words in url	7395.0	-0.009838	0.808278	-1.250000	-0.5

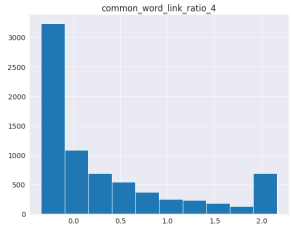
```
df.hist(figsize=(45,45));
```

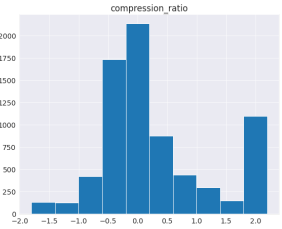


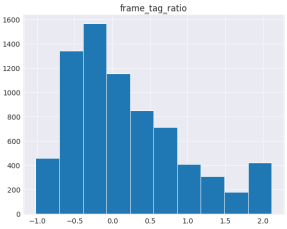


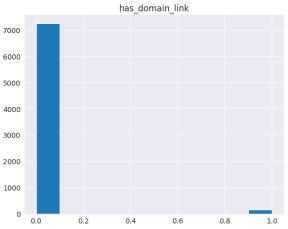


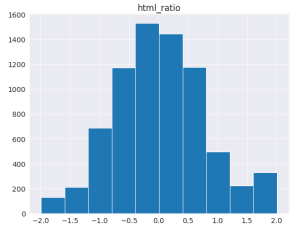


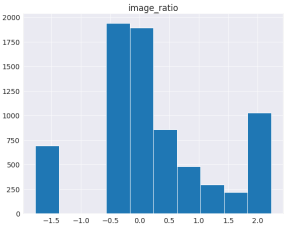


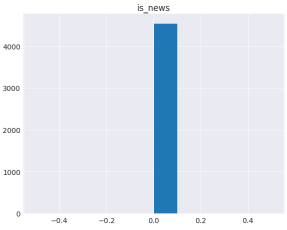


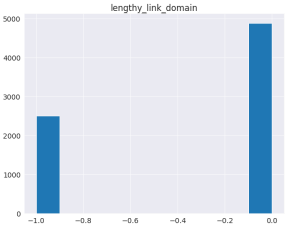


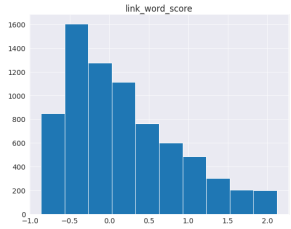


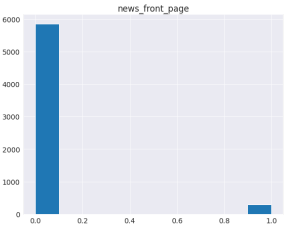


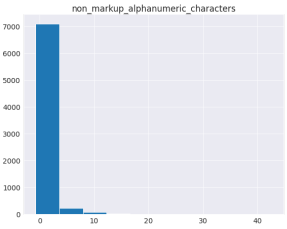


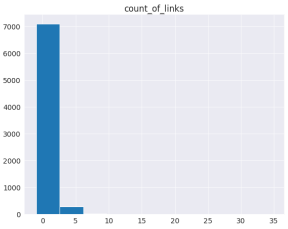


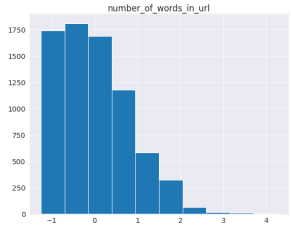


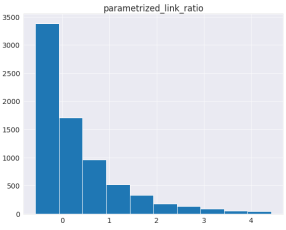


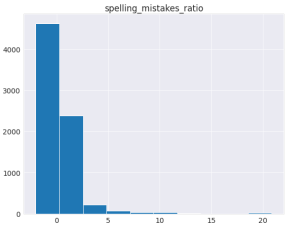













```
df.isnull().any()
```

```
link                                  False
link_id                               False
page_description                      False
alchemy_category                       True
alchemy_category_score                 True
avg_link_size                         False
common_word_link_ratio_1              False
common_word_link_ratio_4              False
compression_ratio                     False
frame_tag_ratio                       False
has_domain_link                       False
html_ratio                            False
image_ratio                           False
is_news                                True
lengthy_link_domain                   False
link_word_score                       False
news_front_page                        True
non_markup_alphanumeric_characters    False
count_of_links                        False
number_of_words_in_url                False
parametrized_link_ratio               False
spelling_mistakes_ratio               False
dtype: bool
```

Exploring Alchemy_category and Score features..

```
#no of datavalues for each alchemy catergory
countplt, ax = plt.subplots(figsize = (10,7))

ax.tick_params(axis='x', rotation=90)
sns.countplot(df.alchemy_category)
for rect in ax.patches:
    ax.text (rect.get_x() + rect.get_width()  / 2,rect.get_height()+ 0.75,rect.get_height(
countplt;
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
```


```
from wordcloud import WordCloud
```

This Function plots Word Cloud for each title


```
def create_wordcloud(words,title):
    wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).gener
    plt.figure(figsize=(10, 7))
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis('off')
    plt.title(title, fontsize=13)
    plt.show()
```

```
df["alchemy_category"].value_counts()
```

```
recreation            1229
arts_entertainment     941
business               880
health                 506
sports                 380
culture_politics       343
computer_internet      296
science_technology     289
gaming                  76
religion                72
law_crime               31
weather                  4
Name: alchemy_category, dtype: int64
```

```
list_labels=["recreation","arts_entertainment","business","health","sports","culture_polit
# list_labels=data["alchemy_category"].unique()
print(len(list_labels))
```

```
12
```

```
from nltk.corpus import stopwords
import nltk
```

```
nltk.download('omw-1.4')
nltk.download('punkt')
```

```
[nltk_data] Downloading package omw-1.4 to /usr/share/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
import json
import re
```

```
df.shape
```

```
(7395, 22)
```

JSON Encoding for page_description column

```
k=0
for i in range(df.shape[0]):
    stopword=["http",":","//",".","-","/","www"]
    t=""
    y=json.loads(df["page_description"][i])
    if("url" not in y.keys()):
#         a=re.split("/",df["link"][i])
        print(df["link"][i])
        if("title" in y.keys()):
            print(y["title"],k)
            k=k+1
print(k)
```

```
How to Make Fried Green Tomatoes 30
http://www.funnyordie.com/slideshows/d2c16682d5/the-best-pictures-front-the-london
The Best Pictures front the London Olympics So Far from Look What I Found the best
http://www.youtube.com/watch?v=IYkdPxRwPVA
Ronaldinho bicycle kick v Villareal - incl buildup + Eng com 32
http://www.youtube.com/watch?v=ksofdpVhEns
Well...Simply better than Ronaldinho 33
http://www.youtube.com/watch?v=wFB_vHVFM_8
Meat Salad - Epic Meal Time 34
http://www.youtube.com/watch?v=QbvPRNzaxaY
How to Make Candy Sushi 35
http://www.youtube.com/watch?v=dj5inlxMgj0
Medicine not Myth: Subway Poles 36
http://www.youtube.com/watch?v=bg7pDvUhfes
Trampoline wall action with Oli Lemieux 37
http://www.youtube.com/watch?v=8d8EymQPiqk
Anthony Bourdain in Quebec 38
http://www.youtube.com/watch?v=eKtsMfmQ_0s
Pizza Dough Recipes 39
http://www.youtube.com/watch?v=mXqB-FeTWS0
Hutchison Effect - Beyond Invention discovery 40
http://www.youtube.com/watch?v=eU3mxGiQ76o
Mental Health Hotline 41
http://www.youtube.com/watch?v=ZYffV7qhvTc
84 Egg Sandwich - Epic Meal Time 42
```

```
http://www.youtube.com/watch?v=AP27HxEzdK8
Fast Proof (209): Jacques Pépin: More Fast Food My Way 43
http://www.youtube.com/watch?v=x4StzNTpNls
This cat is the most inefficient drinker 44
http://www.youtube.com/watch?v=ippMPPu6gh4
Speed Air Man--David Belle 45
http://www.youtube.com/watch?v=g6I8Fc3kw60
Epic Chicken Burger Combo - Epic Meal Time 46
http://www.youtube.com/watch?v=7SVrhTmlx7c
What Pots and Pans Every Well Stocked Kitchen Needs 47
http://www.youtube.com/watch?v=tpKINd8NPwo
Quitting your job with style 48
http://www.youtube.com/watch?v=GbH5gAh0qPQ
Slap Chop Rap (broadcast version) by Porterhousemedia.com 49
http://www.youtube.com/watch?v=BlTCkNkfmRY
Food Wishes Recipes - Inside-Out Grilled Cheese Sandwich - Ultimate Cheese Sandwic
http://www.youtube.com/watch?v=v0l7_yvclRc
Autsch das tut weh!!! 51
http://www.youtube.com/watch?v=TXGpmxCicpI
Spread The Love - Support Therapy Dogs 52
http://www.youtube.com/watch?v=Sk_CGtA4HIY
Cookalong Live | How To Make Bolognaise | Gordon Ramsay on Channel 4 53
http://www.youtube.com/watch?v=5jb2DNJPgk4
How A Safe Works 54
http://www.youtube.com/watch?v=-Yv9jcIXn_I
Awesome ice skating warm-up routine 55
http://www.youtube.com/watch?v=cPsY2NfPJtw
Lunartic_Hubless Wheel Prototype.mov 56
http://www.youtube.com/watch?v=BESzeg9T6mk
A Little France in Your Toast-Food Network 57
http://www.youtube.com/watch?v=EtVUW-eAg40
Madden NFL 13 - Madden Forever: Black and Yellow 58
59
```

There are 59 rows with no url in page_description

```
df["collection"]=''


k=0
for i in range(df.shape[0]):
    t=""
    y=json.loads(df["page_description"][i])
    if("url" in y.keys()):
        t=t+y["url"]
    else:
        k=k+1
    if(y["body"]!=None):
        t=t+y["body"]
    elif("title" in y.keys()):
        if(y["title"]!=None):
            t=t+y["title"]
        else:
            df.drop(df.index[i],axis=0)
```

```
    df["collection"][i]=t
print("without url : ",k)
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:17: SettingWithCopyWarni
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  app.launch_new_instance()
without url :  60
```

```
df['collection'][0]
```

```
"cbc ca stevenandchris 2012 11 peggy ks sexy mood boosting cupcakes htmlIf you re
ready to give your libido a boost with a sweet treat then you re going to want to
try Peggy K s sexy mood boosting cupcakes Wet ingredients 1 tablespoon ground chia
mixed with 1 4 cup water1 1 2 ripe banana2 tablespoons coconut oil½ cup walnut
butter 2 teaspoon s vanilla extract1 cup almond milk ½ cup coconut sugar Dry
ingredients 1 cup brown rice flour½ cup cooked quinoa 3 tablespoon maca powder1 3
cup cocoa powder1 teaspoon baking powder1 teaspoon non aluminum baking soda½
teaspoon saltchopped walnuts Raspberry frosting 1 cups Raw cashews1 4 cup maple
syrup 1 cups raspberries Almond milk For the Cupcakes Preheat the oven to 350 F
Place ground chia in a small bowl Add warm water and mix with a fork Set aside for
gel to form Mash banana in large bowl then add oil nut butter vanilla and milk and
stir to mix Add the rest of the ingredients and mix until well incorporated Grease
rubber mini muffin tins with coconut oil Drop in batter and bake 25 30 minutes until
toothpick comes out clean For the frosting Add all ingredients to high power blender
and blend until smooth Add a splash of Almond milk just to blend the ingredients but
frosting should be thick Transfer to piping bag and pipe onto cupcakes whether the
subject is home decor, health, beauty, cooking, relationships, finance or
entertaining, steven and chris want to help you add some fabulous to your life! if
you're ready to give your libido a boost with a sweet treat, then you're going to
want to try peggy k's sexy mood-boosting cupcakes! wet ingredients: 1 tablespoon
ground chia; mixed with 1/4 cup water 1 1/2 ripe banana..."
```

trying word2vec for word embedding in 'page_description' column

```
import nltk.data
```

```
from bs4 import BeautifulSoup
```

Function for text cleansing

```
def prepro_page_desc (description, remove_stopwords=False, no_empty_lists=False):

    words = BeautifulSoup(description).get_text()

    words = re.sub("[^a-zA-Z\d]", " ", words)

    words = words.lower().split()
```

```
    if remove_stopwords:
        stops = set(stopwords.words("english"))
        words = [w for w in words if not w in stops]

    return words


tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
```

This function to split a description into parsed sentences

```
def description_to_sentences(description, tokenizer, remove_stopwords=False):

    raw_sentences = tokenizer.tokenize(description.strip())


    sentences = []
    for raw_sentence in raw_sentences:

        if len(raw_sentence) > 0:
            sentences.append(prepro_page_desc(raw_sentence, remove_stopwords))


    return sentences



sentences = []
for desc in df['collection']:
   sentences += description_to_sentences(desc, tokenizer)
```

```
/opt/conda/lib/python3.7/site-packages/bs4/__init__.py:439: MarkupResemblesLocatorWar
  MarkupResemblesLocatorWarning
/opt/conda/lib/python3.7/site-packages/bs4/__init__.py:408: MarkupResemblesLocatorWar
  MarkupResemblesLocatorWarning
```

```
sentences[0]
```

Show hidden output

```
from gensim.models import Word2Vec, Phrases


# Set values for various parameters
num_features = 300
min_word_count = 1
num_workers = 4
context = 10
downsampling = 1e-3
```

```
w2v_model = Word2Vec(sentences, window=context, workers=num_workers,
                     vector_size=num_features,
                     min_count=min_word_count, sample=downsampling, sg=1)
model_name = "w2v"
w2v_model.save(model_name)
```

```
w2v_model = Word2Vec.load('w2v')
```

```
processed_data = df.copy(deep=True)
processed_data['tokenizedDescription'] = processed_data['collection'].apply(lambda x: prep
```

```
/opt/conda/lib/python3.7/site-packages/bs4/__init__.py:439: MarkupResemblesLocatorWar
  MarkupResemblesLocatorWarning
```

```
def makeFeatureVec(words, model, num_features):

    featureVec = np.zeros((num_features,),dtype="float32")


    nwords = 0.


    index2word_set = set(model.wv.index_to_key)

    if len(words) == 0:
        words = ['unknown']

    for word in words:
        if word in index2word_set:
            nwords = nwords + 1.
            featureVec = np.add(featureVec, model.wv[word])


    featureVec = np.divide(featureVec, nwords)

    return featureVec


def getAvgFeatureVecs(descriptions, model, num_features):
    descriptionFeatureVecs = np.zeros((len(descriptions),num_features),dtype="float32")

    for i, description in enumerate(descriptions):

        descriptionFeatureVecs[i] = makeFeatureVec(description, model, num_features)

    return descriptionFeatureVecs


vectorized_data = pd.DataFrame(getAvgFeatureVecs(processed_data['tokenizedDescription'], w
```

```
vectorized_data.shape
```

```
     (7395, 300)
```

```
df3=vectorized_data
```

```
df=pd.concat([df, df3], axis=1)
```

```
df.shape
```

```
     (7395, 323)
```

this is the vectorized column we have created via word2vec

since alchemy category and score prediction was dependent on tfifdf vectors, we used these vector instead

stopwords removal,tokenizing and use of regular expression for text purification

We applied TFIDF and later applied Word2Vec and found better results using the later.

Therefore, the tfidf implimetation is commented below.

```
# from nltk.corpus import stopwords
# from nltk.tokenize import word_tokenize
# from nltk.stem.porter import PorterStemmer
# from nltk.stem import WordNetLemmatizer
# ps=PorterStemmer()
# wordnet=WordNetLemmatizer()
```

```
# for i in range(df.shape[0]):
#     text = re.sub(r'\[[0-9]*\]',' ',df["collection"][i])
#     text = re.sub(r'\s+',' ',text)
#     text = text.lower()
#     text = re.sub(r'\d',' ',text)
#     text = re.sub(r'\s+',' ',text)
#     text = text.lower().replace('\n',' ').replace('\r','').strip()
#     text = re.sub(' +', ' ', text)
#     text = re.sub(r'[^\w\s]','',text)
```

```
#     stop_words = set(stopwords.words('english'))
#     word_tokens = word_tokenize(text)
# #     filtered_sentence = [ps.stem(w) for w in word_tokens if not w in stop_words]
#     filtered_sentence = [wordnet.lemmatize(w) for w in word_tokens if not w in stop_word
# #     filtered_sentence = []
# #     for w in word_tokens:
# #         if w not in stop_words:
# #             filtered_sentence.append(w)

#     text = " ".join(filtered_sentence)



#     df["collection"][i]=text
```

```
k=0
for i in range(df.shape[0]):
    y=json.loads(df["page_description"][i])
    if("title" not in y.keys()):
        k=k+1
print(k)
```

```
1
```

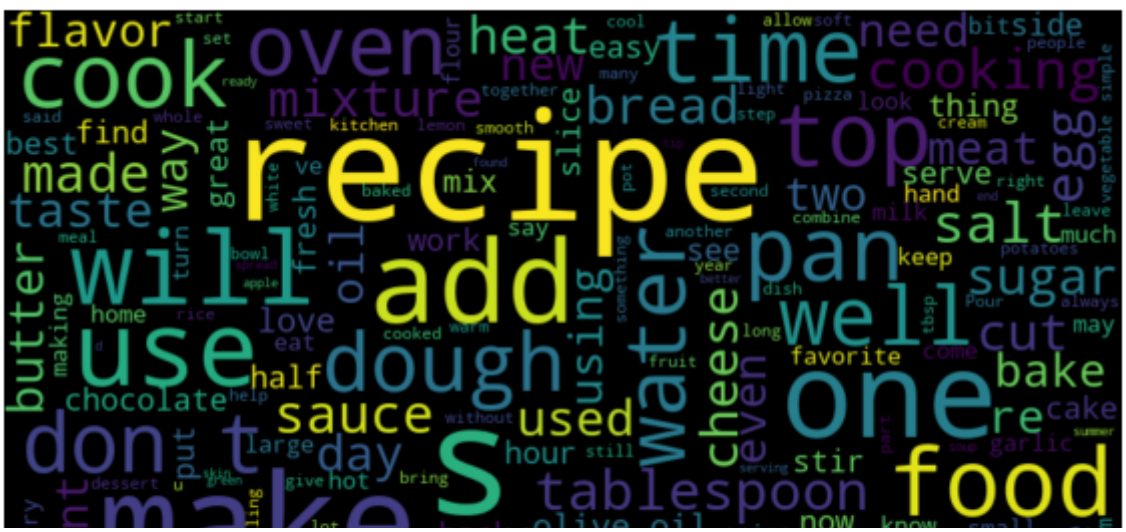
plotting word cloud for various alchemy category

```
for i in range (0,len(list_labels)):
    subset=df[df.alchemy_category==list_labels[i]]
    '''[word for word in sentences[i] if word not in stopwords.words('english')]'''
    text=[word for word in subset.collection.values if word not in stopwords.words('englis
    words =" ".join(text)
    create_wordcloud(words,list_labels[i])
```

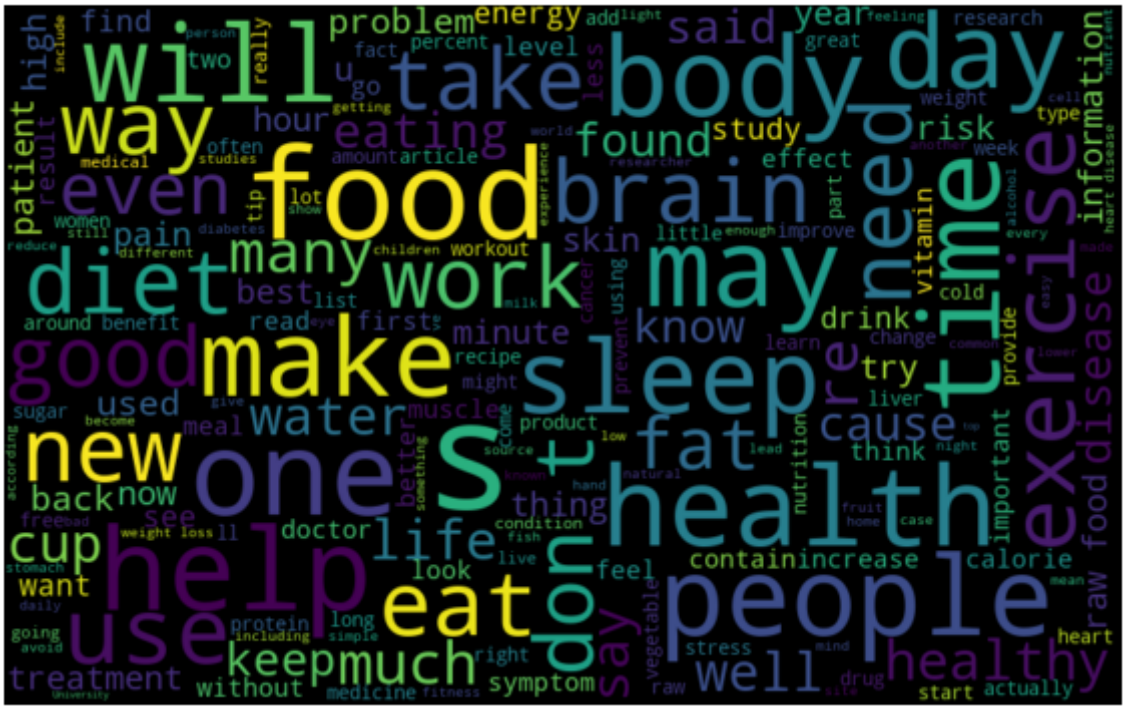
recreation

arts_entertainment

business

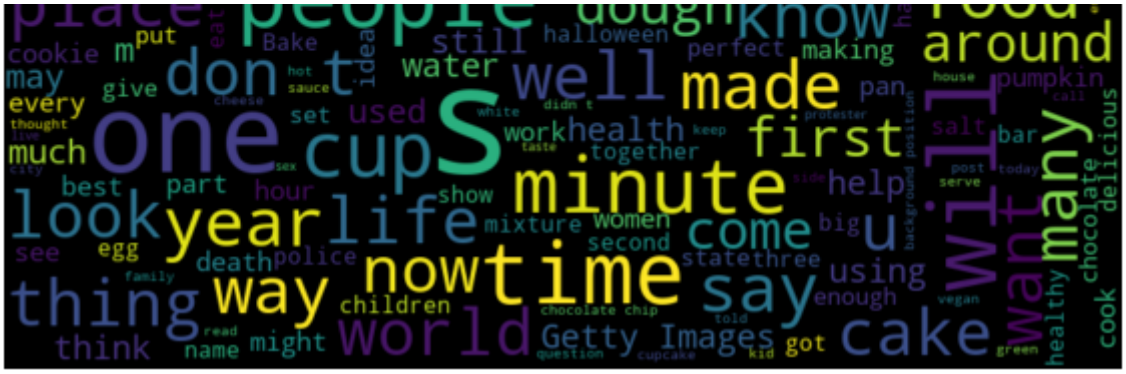


health

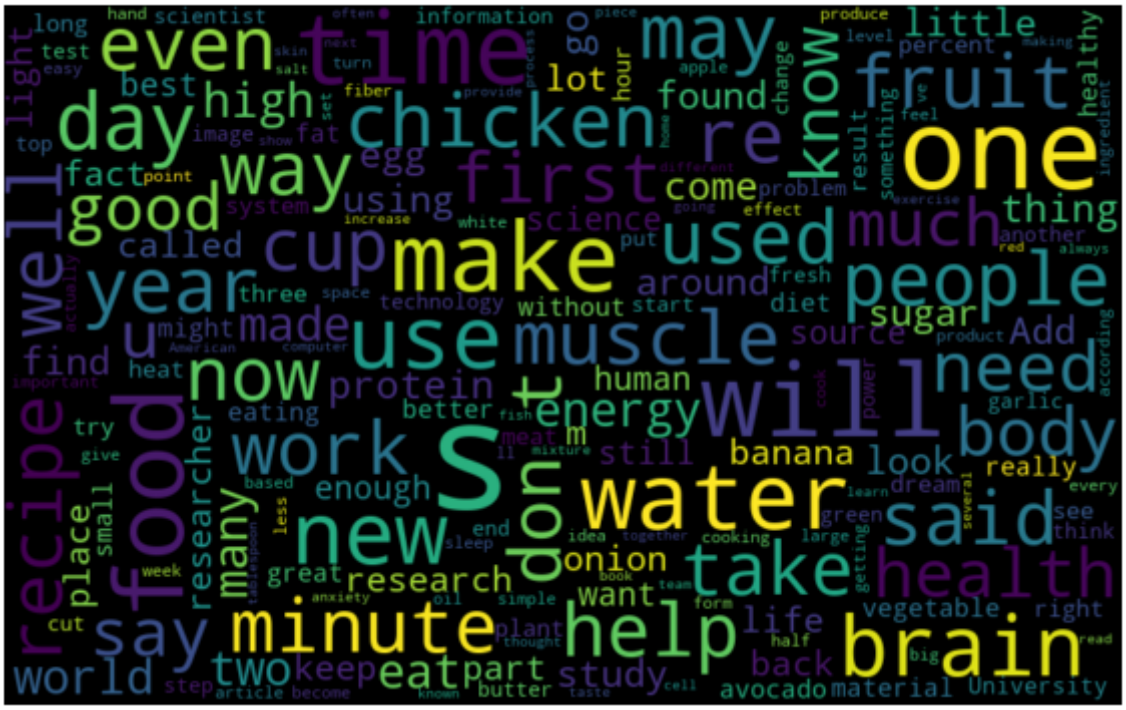


sports

culture_politics

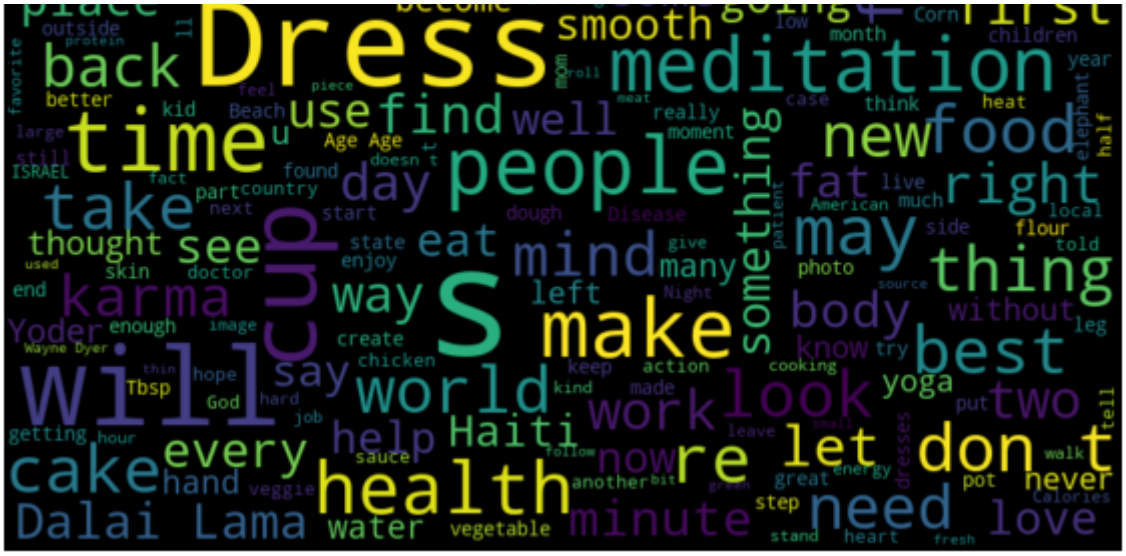


science_technology



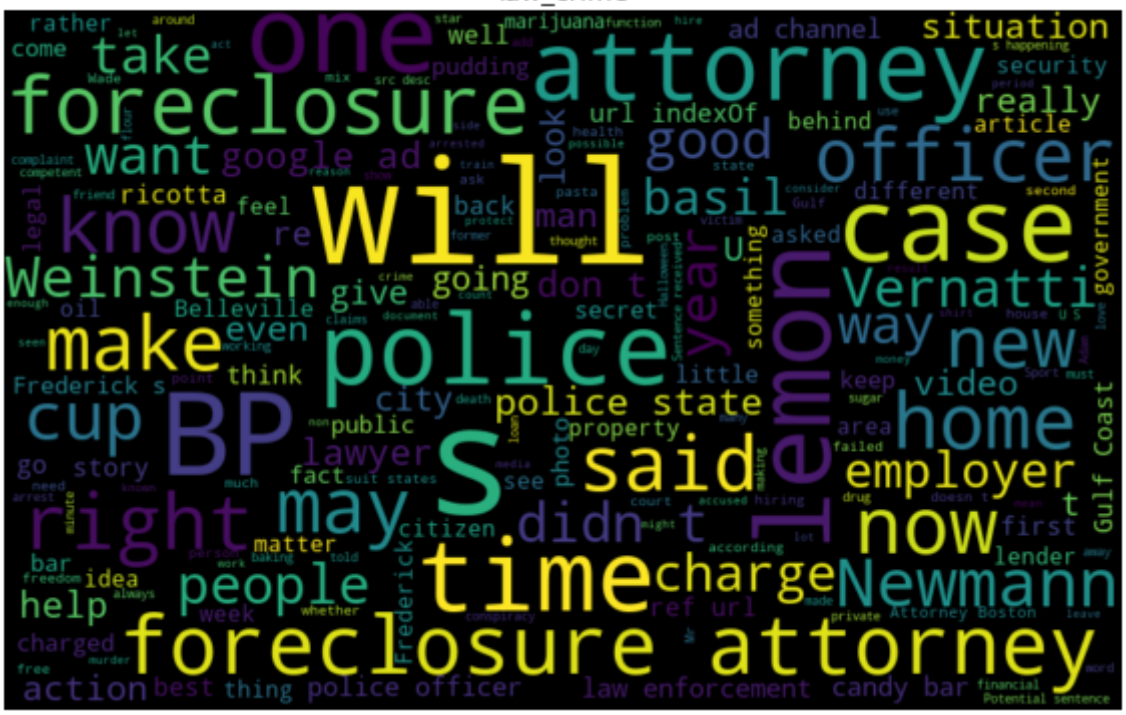
computer_internet

religion



gaming

law_crime



```
df["alchemy_category"].unique()
```

```
array(['arts_entertainment', 'recreation', 'business', 'sports', nan,
       'culture_politics', 'computer_internet', 'religion', 'health',
       'science_technology', 'gaming', 'law_crime', 'weather'],
      dtype=object)
```

creating vectors via tfidf

```
# from sklearn.feature_extraction.text import TfidfVectorizer
```

```
# X_train=df["collection"]
# df["vec"]="";
```

```
# ngram_range = (1,2)
# min_df = 10
# max_df = 1.
# max_features = None
```

Parameterizing TFIDF for optimal results

```
# tfidf = TfidfVectorizer(encoding='utf-8',
#                         ngram_range=ngram_range,
#                         stop_words=None,
#                         lowercase=False,
#                         max_df=max_df,
#                         min_df=min_df,
#                         max_features=max_features,
#                         norm='l2',
#                         sublinear_tf=True)

# features_train = tfidf.fit_transform(X_train).toarray()
# print(features_train)
# feature_name=tfidf.get_feature_names()
# print(feature_name)


# len(feature_name)


# df3=pd.DataFrame(features_train)


# df=pd.concat([df, df3], axis=1)


df.shape
```

```
(7395, 323)
```

```
for i in df.columns:
    print(i)
```

```
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
```

```
164
165
166
```

```
df.describe()
```

	link_id	alchemy_category_score	avg_link_size	common_word_link_ratio_1
count	7395.000000	5053.000000	7395.000000	7395.000000
mean	5305.704665	0.603334	0.085805	-0.047970
std	3048.384114	0.212864	0.846056	0.735367
min	1.000000	0.070833	-1.974135	-1.743019
25%	2688.500000	0.452424	-0.474135	-0.510841
50%	5304.000000	0.625616	0.000000	0.000000
75%	7946.500000	0.780851	0.525865	0.489159
max	10566.000000	0.999426	2.025865	1.877098

8 rows × 319 columns

droping link , linkid, page description as the information the give is being sustain in vectors of tfidf that was made from page description

dropping is_news and news_front as they contain only single boolean value or null

```
# column=["link","link_id","page_description","vec","collection","news_front_page","is_new
column=["link","link_id","page_description","collection","news_front_page","is_news"]

df.drop(column,axis=1,inplace=True)


p=df.dropna(axis=0)
```

model for prediction of null values in alchemy category and alchmey scores

```
preprocessing_train=p.iloc[:,17:]
preprocessing_test=p.iloc[:,0:1]


preprocessing_train
```

	0	1	2	3	4	5	6	7
0	-0.068508	0.148376	0.128258	0.118385	-0.165992	-0.338777	0.145037	0.589548
1	-0.017838	0.149078	0.187062	0.170628	-0.093950	-0.366698	0.126942	0.456239
2	-0.028942	0.163552	0.138368	0.180540	-0.091550	-0.234489	0.046797	0.524915
3	-0.066669	0.229343	0.208808	0.228460	-0.059012	-0.288841	0.097990	0.522283
4	-0.026426	0.210528	0.094497	0.131334	-0.052167	-0.292249	0.101627	0.577637
...	...	...	...	...	...	...	...	...
7384	-0.050816	0.195688	0.043598	0.138692	0.001136	-0.262186	0.047521	0.438514
7385	-0.036273	0.158852	0.126808	0.152189	-0.163522	-0.297087	0.141951	0.539328
7387	-0.052628	0.167797	0.101629	0.135696	-0.140195	-0.316152	0.141002	0.541762
7391	-0.033996	0.186613	0.106214	0.192878	-0.084731	-0.142452	0.025540	0.472347
7394	0.023547	0.196497	0.143480	0.262296	0.024761	-0.271987	0.112844	0.568172

```
preprocessing_test
```

	alchemy_category
0	arts_entertainment
1	recreation
2	business
3	arts_entertainment
4	sports
...	...
7384	arts_entertainment
7385	recreation
7387	arts_entertainment
7391	health
7394	business

5047 rows × 1 columns

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
preprocessing_test['alchemy_category']= label_encoder.fit_transform(preprocessing_test['al
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:5: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
    See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
      """
```

```
x=preprocessing_test.index
```

Here we used decision tree model for multi-class classifcation

```
from sklearn.tree import DecisionTreeClassifier
clf_pruned = DecisionTreeClassifier(criterion = "gini", random_state = 100,max_depth=3, mi
clf_pruned.fit(preprocessing_train, preprocessing_test)
```

```
DecisionTreeClassifier(max_depth=3, min_samples_leaf=5, random_state=100)
```

```
df.iloc[:,17:]
```

	0	1	2	3	4	5	6	7
0	-0.068508	0.148376	0.128258	0.118385	-0.165992	-0.338777	0.145037	0.589548
1	-0.017838	0.149078	0.187062	0.170628	-0.093950	-0.366698	0.126942	0.456239
2	-0.028942	0.163552	0.138368	0.180540	-0.091550	-0.234489	0.046797	0.524915
3	-0.066669	0.229343	0.208808	0.228460	-0.059012	-0.288841	0.097990	0.522283
4	-0.026426	0.210528	0.094497	0.131334	-0.052167	-0.292249	0.101627	0.577637
...	...	...	...	...	...	...	...	...
7390	0.017042	0.170433	0.107001	0.185401	-0.114041	-0.273779	0.072669	0.490160
7391	-0.033996	0.186613	0.106214	0.192878	-0.084731	-0.142452	0.025540	0.472347
7392	-0.025067	0.173424	0.134872	0.141732	-0.156548	-0.325155	0.052547	0.471125
7393	-0.015723	0.170996	0.080832	0.169244	-0.127769	-0.199208	0.077124	0.459254
7394	0.023547	0.196497	0.143480	0.262296	0.024761	-0.271987	0.112844	0.568172

7395 rows × 300 columns

```
y_pred=clf_pruned.predict(df.iloc[:,17:])
tr=clf_pruned.predict_proba(df.iloc[:,17:])
```

```
nr,nc=df.shape
nr
```

```
7395
```

```
tr.shape
```

```
(7395, 12)
```

```
tempdf = pd.DataFrame(data = y_pred,columns = ["alchemy_category"])
```

```
df["alchemy_category"]
```

```
0       arts_entertainment
1                recreation
2                  business
3       arts_entertainment
4                    sports
                  ...
7390                    NaN
7391                 health
7392                    NaN
7393                    NaN
7394                business
Name: alchemy_category, Length: 7395, dtype: object
```

we have predicted the category from the entirety of the data set and replaced just those values which were nell in orignal dataset

```
k=0
for i in range(nr):
    if df["alchemy_category"][i]  is np.nan:
        df["alchemy_category"][i]=tempdf["alchemy_category"][i]
    else:
        df["alchemy_category"][i]=preprocessing_test["alchemy_category"][i]
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:6: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us

/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:4: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  after removing the cwd from sys.path.
```

```
data2["alchemy_category_score"]
```

```
0       0.365831
1       0.876315
2            NaN
3            NaN
4       0.747449
          ...
2953         NaN
2954    0.605566
2955         NaN
2956         NaN
```

```
2957    0.196317
Name: alchemy_category_score, Length: 2958, dtype: float64
```

```
df["alchemy_category"].unique()
```

```
array([0, 7, 1, 10, 2, 3, 8, 5, 9, 4, 6, 11], dtype=object)
```

```
df.head()
```

	alchemy_category	alchemy_category_score	avg_link_size	common_word_link_ratio_1
0	0	0.471752	-0.353974	-0.043781
1	7	0.885088	-1.210372	-1.255090
2	1	0.716379	0.512097	0.235978
3	0	0.562999	-0.637758	-0.404330
4	10	0.893246	-0.299303	0.178222

5 rows × 317 columns

we were first predicting alchemy category score using Linear Regression, but accuracy was not optimal and thus we decided to replace null with category-wise mean of each null entry wrt to its category.

```
# d_X_train=df.dropna(axis=0)
# d_Y_train=d_X_train
# d_X_train=d_X_train.iloc[:,np.r_[0,20:21741]]
# d_Y_train=d_Y_train.iloc[:,1:2]
# from sklearn import linear_model
# regr = linear_model.LinearRegression()
# regr.fit(d_X_train, d_Y_train)
```

```
# y_score_pred=regr.predict(df.iloc[:,np.r_[0,20:21741]])
```

```
# y_score_pred
```

```
# tempdf123 = pd.DataFrame(data = y_score_pred,columns = ["alchemy_category_score"])
```

```
# tempdf123["alchemy_category_score"][7390]
```

```
nr
```

```
7395
```

```
# for i in range(nr):
#     if df["alchemy_category_score"][i]  is np.NaN:
#         df["alchemy_category_score"][i]=tempdf123["alchemy_category_score"][i]
#     else:
#         df["alchemy_category_score"][i]=df["alchemy_category_score"][i]
```

```
x=df.groupby('alchemy_category')['alchemy_category_score'].mean()
```

```
x
```

```
alchemy_category
0     0.635313
1     0.610034
2     0.699808
3     0.577337
4     0.505969
5     0.705927
6     0.499999
7     0.527689
8     0.426596
9     0.549531
10    0.666918
11    0.596973
Name: alchemy_category_score, dtype: float64
```

```
k=0
for i in range(nr):
    if pd.isna(df["alchemy_category_score"][i]):
        df["alchemy_category_score"][i]=x[df["alchemy_category"][i]]
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:4: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  after removing the cwd from sys.path.
```

```
df["alchemy_category_score"]
```

```
0       0.471752
1       0.885088
2       0.716379
3       0.562999
4       0.893246
          ...
7390    0.699808
7391    0.605566
7392    0.635313
7393    0.705927
7394    0.196317
Name: alchemy_category_score, Length: 7395, dtype: float64
```

```
type(df["alchemy_category_score"][7393])
```

```
numpy.float64
```

```
type(df["alchemy_category"][67])
```

```
numpy.int64
```

```
df["alchemy_category_score"]
```

```
0       0.471752
1       0.885088
2       0.716379
3       0.562999
4       0.893246
          ...
7390    0.699808
7391    0.605566
7392    0.635313
7393    0.705927
7394    0.196317
Name: alchemy_category_score, Length: 7395, dtype: float64
```

```
print(data.shape)
df.head()
```

```
(4437, 27)
```

	alchemy_category	alchemy_category_score	avg_link_size	common_word_link_ratio_1
0	0	0.471752	-0.353974	-0.043781
1	7	0.885088	-1.210372	-1.255090
2	1	0.716379	0.512097	0.235978
3	0	0.562999	-0.637758	-0.404330
4	10	0.893246	-0.299303	0.178222

5 rows × 317 columns

Till now alchemy_category was label encoded , but now we have connverted it to one hot encoding.

[ ] ↳ *1 cell hidden*

At this point we have the data fully preprocessed with no na values and categorical data.

```
df["alchemy_category_11"]
```

```
0       0
1       0
2       0
3       0
4       0
       ..
7390    0
7391    0
7392    0
7393    0
7394    0
Name: alchemy_category_11, Length: 7395, dtype: uint8
```

```
finaltrain=df.iloc[:4437,:];
```

df is split into orignal train and test dataset

```
finaltrain["label"]=data["label"]
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  """Entry point for launching an IPython kernel.
```

```
finaltrain.head()
```

	alchemy_category_score	avg_link_size	common_word_link_ratio_1	common_word_link
0	0.471752	-0.353974	-0.043781	
1	0.885088	-1.210372	-1.255090	
2	0.716379	0.512097	0.235978	
3	0.562999	-0.637758	-0.404330	
4	0.893246	-0.299303	0.178222	

5 rows × 329 columns

At this stage we have finaltrain fully ready for model training

applying models and making predictions

```
from sklearn.model_selection import train_test_split
TX=finaltrain.drop('label',axis=1)
Ty=finaltrain["label"]
TX_train, TX_test, Ty_train, Ty_test = train_test_split(TX, Ty, test_size=0.33, random_sta
```

random forest and its hyperparameter tuning

```
from sklearn.ensemble import RandomForestClassifier
model  = RandomForestClassifier(random_state=1)
model.fit(TX_train, Ty_train)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
RandomForestClassifier(random_state=1)
```

```
from sklearn.model_selection import RandomizedSearchCV

# Number of trees in random forest
n_estimators = [int(x) for x in range(200,2000,200)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800], 'max_features':
```

```
rf = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter =
# Fit the random search model
rf_random.fit(TX_train, Ty_train)
```

Show hidden output

```
rfbest=rf_random.best_params_
rfbest
```

```
{'n_estimators': 600,
 'min_samples_split': 10,
 'min_samples_leaf': 4,
 'max_features': 'sqrt',
 'max_depth': 50,
 'bootstrap': True}
```

```
from sklearn.ensemble import RandomForestClassifier
model  = RandomForestClassifier(max_depth= 50, min_samples_leaf= 4, min_samples_split= 10,
model.fit(finaltrain.drop(["label"],axis=1), finaltrain["label"])
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
RandomForestClassifier(max_depth=50, max_features='sqrt', min_samples_leaf=4,
                       min_samples_split=10, n_estimators=600)
```

```
data2.shape
```

```
(2958, 26)
```

```
from sklearn.metrics import roc_auc_score
y_pred = model.predict_proba(TX_test)[:,1]
roc_auc_score(Ty_test,y_pred)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
0.9925492024997669
```

This is the actual test data with no label column provided

```
finaltest=df.iloc[4437:,:]
finaltest.shape
```

```
(2958, 328)
```

```
model_predictions = model.predict_proba(finaltest)[:,1]
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
```

```
sample_sum=pd.read_csv("../input/aid-escalating-internet-coverage/sample_submission.csv")
sample_sum["label"]=model_predictions
sample_sum.to_csv("./sumrandomforest.csv",index=False)
```

```
sample_sum.head()
```

	link_id	label
0	4049	0.910542
1	3692	0.170304
2	9739	0.193699
3	1548	0.610561
4	5574	0.982950

logistic regression and its hyper parameter tuning

```
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression


model2 = LogisticRegression()
print(model2.get_params())
model2.fit(TX_train, Ty_train)
model2_predictions = model2.predict(TX_test)
print('Accuracy: ', accuracy_score(Ty_test, model2_predictions))
print(classification_report(Ty_test, model2_predictions))
```

```
{'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_sca
Accuracy:  0.8006825938566553
              precision    recall  f1-score   support

           0       0.78      0.82      0.80       710
           1       0.82      0.78      0.80       755

    accuracy                           0.80      1465
   macro avg       0.80      0.80      0.80      1465
weighted avg       0.80      0.80      0.80      1465

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:818: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
```

```
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.1,0.001,1,100],
```

```
    # Create a list of options for the regularization penalty
    'penalty' : ['l1', 'l2'] }
clf = GridSearchCV(model2, param_grid, cv = 3, verbose = 1)

bestF = clf.fit(TX_train, Ty_train)
bestF.best_params_
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:818: Conv

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:818: Conv
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:818: Conv
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:818: Conv
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
```

```
lrbest=bestF.best_params_
lrbest
```

```
{'n_neighbors': 7, 'p': 1}
```

```
model2f = LogisticRegression(C=0.1,penalty='l2')
print(model2f.get_params())
model2.fit(TX_train, Ty_train)
model2_predictions = model2.predict(TX_test)
print('Accuracy: ', accuracy_score(Ty_test, model2_predictions))
print(classification_report(Ty_test, model2_predictions))
```

```
{'C': 0.1, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_sca
Accuracy:  0.8006825938566553
              precision    recall  f1-score   support

           0       0.78      0.82      0.80       710
           1       0.82      0.78      0.80       755

    accuracy                           0.80      1465
   macro avg       0.80      0.80      0.80      1465
weighted avg       0.80      0.80      0.80      1465

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:818: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
```

```
y_pred = model2.predict_proba(TX_test)[:,1]
roc_auc_score(Ty_test,y_pred)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
0.8563025837142058
```

```
model2f = LogisticRegression(C=0.1,penalty='l2')
print(model2f.get_params())
model2f.fit(finaltrain.drop(["label"],axis=1), finaltrain["label"])
model_predictions2 = model2f.predict_proba(finaltest)[:,1]
```

```
{'C': 0.1, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_sca
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnin
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnin
  FutureWarning,
```

```
sample_sum=pd.read_csv("../input/aid-escalating-internet-coverage/sample_submission.csv")
sample_sum["label"]=model_predictions2
sample_sum.to_csv("./sumlogisticreggression.csv",index=False)
```

KNN and its hyperparameter tuning

```
from sklearn.neighbors import KNeighborsClassifier
model3 = KNeighborsClassifier()
model3.fit(TX_train, Ty_train)
model3_predictions = model3.predict(TX_test)
print('Accuracy: ', accuracy_score(Ty_test, model3_predictions))
print(classification_report(Ty_test, model3_predictions))
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnin
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnin
  FutureWarning,
Accuracy:  0.7030716723549488
              precision    recall  f1-score   support

           0       0.70      0.67      0.69       710
           1       0.70      0.73      0.72       755

    accuracy                           0.70      1465
   macro avg       0.70      0.70      0.70      1465
weighted avg       0.70      0.70      0.70      1465
```

```
from sklearn.model_selection import GridSearchCV
params_KNN = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7],
              'p': [1, 2, 5]}


gridF = GridSearchCV(model3, params_KNN, cv = 5, verbose = 1,
                     n_jobs = -1)
bestF = gridF.fit(TX_train, Ty_train)
```

Show hidden output

```
bestF.best_params_
```

```
{'n_neighbors': 7, 'p': 1}
```

```
modelknn = KNeighborsClassifier(n_neighbors=7, p=1)
modelknn.fit(finaltrain.drop(["label"],axis=1), finaltrain["label"])
knnpredictions = modelknn.predict_proba(finaltest)[:,1]
sample_sum["label"]=knnpredictions
sample_sum.to_csv("./sumknn.csv",index=False)
sample_sum.head(30)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
```

	link_id	label
0	4049	1.000000
1	3692	0.285714

Decision Tree and its hyperparameter tuning

3	1548	0.714286

```
from sklearn.tree import DecisionTreeClassifier
model4 = DecisionTreeClassifier(random_state=1)
model4.fit(TX_train, Ty_train)
model_predictions = model4.predict(TX_test)
print('Accuracy: ', accuracy_score(Ty_test, model_predictions))
print(classification_report(Ty_test, model_predictions))
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
Accuracy:  0.7037542662116041
              precision    recall  f1-score   support

           0       0.70      0.69      0.69       710
           1       0.71      0.72      0.71       755

    accuracy                           0.70      1465
   macro avg       0.70      0.70      0.70      1465
weighted avg       0.70      0.70      0.70      1465

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
```

17	6948	0.857143

```
import numpy as np
from scipy.stats import randint
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV

min_samples_leaf = randint(1,9)

criterion = ['gini', 'entropy']
max_depth = [3,None]



hyperF = dict(max_depth=max_depth,
                      criterion=criterion,
                      min_samples_leaf=min_samples_leaf)

model4  = DecisionTreeClassifier(random_state=1)
gridF4 = RandomizedSearchCV(model, hyperF, cv = 3, verbose = 1)
bestF4 = gridF4.fit(TX_train, Ty_train)
```

```
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
```

```
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
```

```
dtbest=bestF4.best_params_
dtbest
```

```
{'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 8}
```

```
model4f = DecisionTreeClassifier(criterion= 'entropy', max_depth= None,min_samples_leaf=8)
model4f.fit(finaltrain.drop(["label"],axis=1), finaltrain["label"])
model_predictions4 = model4f.predict_proba(finaltest)[:,1]
sample_sum["label"]=model_predictions4
sample_sum.to_csv("./sumdecisiontree.csv",index=False)
sample_sum.head(30)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
```

	link_id	label
0	4049	1.000000
1	3692	0.000000
2	9739	0.466667
3	1548	1.000000

XGBoost and its hyperparameter tuning

```
import xgboost
clf=xgboost.XGBClassifier()
# clf = xgb.XGBClassifier()
clf.fit(TX_train, Ty_train)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...)
```

```
params={
 "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]

}


classifier=xgboost.XGBClassifier()
random_search=RandomizedSearchCV(classifier,param_distributions=params,n_iter=5,scoring='r
random_search.fit(TX_train,Ty_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
RandomizedSearchCV(cv=5,
                   estimator=XGBClassifier(base_score=None, booster=None,
                                           callbacks=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None,
                                           early_stopping_rounds=None,
                                           enable_categorical=False,
```

```
                                        eval_metric=None, gamma=None,
                                        gpu_id=None, grow_policy=None,
                                        importance_type=None,
                                        interaction_constraints=None,
                                        learning_rate=None, max_bin=None,...
                                        n_estimators=100, n_jobs=None,
                                        num_parallel_tree=None,
                                        predictor=None, random_state=None,
                                        reg_alpha=None, reg_lambda=None, ...),
                   n_iter=5, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.3, 0.4, 0.5,
                                                             0.7],
                                        'gamma': [0.0, 0.1, 0.2, 0.3, 0.4],
                                        'learning_rate': [0.05, 0.1, 0.15, 0.2,
                                                          0.25, 0.3],
                                        'max_depth': [3, 4, 5, 6, 8, 10, 12,
                                                      15],
                                        'min_child_weight': [1, 3, 5, 7]},
                   scoring='roc_auc', verbose=3)
```

```
print(random_search.best_params_)
```

```
{'min_child_weight': 7, 'max_depth': 5, 'learning_rate': 0.05, 'gamma': 0.1, 'colsamp
```

```
print(random_search.best_estimator_)
ttx=random_search.best_estimator_
```

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.7,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0.1, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.05, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=5, max_leaves=0, min_child_weight=7,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...)
```

```
clf=xgboost.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.3, gamma=0.2, learning_rate=0.05,
       max_delta_step=0, max_depth=10, min_child_weight=1,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)
clf.fit(finaltrain.drop(["label"],axis=1), finaltrain["label"])
```

```
[12:39:05] WARNING: ../src/learner.cc:627:
Parameters: { "silent" } might not be used.

  This could be a false alarm, with some parameters getting used by language bindings
  then being mistakenly passed down to XGBoost core, or some parameter actually being
  but getting flagged wrongly here. Please open an issue if you find any such cases.
```

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.3,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0.2, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.05, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=10, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=1, nthread=1, num_parallel_tree=1, predictor='auto',
              random_state=0, reg_alpha=0, ...)
```

```
model_predictions5 = clf.predict_proba(finaltest)[:,1]
sample_sum["label"]=model_predictions5
sample_sum.to_csv("./sumxgboost.csv",index=False)
sample_sum.head()
```

	link_id	label
0	4049	0.979695
1	3692	0.041104
2	9739	0.144254
3	1548	0.858568
4	5574	0.991684

SVM and its hyperparameter tuning

```
from sklearn.svm import SVC
model = SVC()
model.fit(TX_train, Ty_train)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
SVC()
```

```
param_grid = {'C':[0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

# fitting the model for grid search
grid.fit(TX_train, Ty_train)
grid.best_estimator_
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
```

```
  FutureWarning,
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.514 total time=   1.5s
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.514 total time=   1.5s
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.515 total time=   1.5s
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.513 total time=   1.5s
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.513 total time=   1.5s
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
[CV 1/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.741 total time=   1.3s
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
[CV 2/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.719 total time=   1.3s
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
[CV 3/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.747 total time=   1.3s
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
[CV 4/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.756 total time=   1.3s
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
[CV 5/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.766 total time=   1.2s
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
  FutureWarning,
[CV 1/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.756 total time=   1.4s
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWar
```

```
print(grid.best_params_)
```

```
{'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
```

```
print(grid.best_estimator_)
```

```
SVC(C=1000, gamma=0.0001)
```

```
model6f = SVC(C=1000,gamma=0.0001,kernel='rbf',probability=True)
model6f.fit(finaltrain.drop(["label"],axis=1), finaltrain["label"])
model_predictions6 = model6f.predict_proba(finaltest)[:,1]
sample_sum["label"]=model_predictions6
sample_sum.to_csv("./sumsvc.csv",index=False)
sample_sum.head()
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnir
  FutureWarning,
```

	link_id	label
0	4049	0.756989
1	3692	0.154919
2	9739	0.290335
3	1548	0.708370
4	5574	0.882137

Voting Classifier using LogisticRegression SVC Xgboost RandomForestClassifier and we are using tuned parameters from previous models.

This gave us the best results !!!

```
estimator = []
estimator.append(('LR',
                  LogisticRegression(C=0.1,penalty='l2')))
estimator.append(('SVC', SVC(C=1000,gamma =0.0001,kernel='rbf',probability = True)))
estimator.append(('XGB',xgboost.XGBClassifier(base_score=0.5, booster='gbtree',
                                              colsample_bylevel= 1,
                                              colsample_bytree=0.3, gamma=0.2,
                                              learning_rate= 0.05, max_delta_step=0,
                                              max_depth=10, min_child_weight=1,
                                              n_estimators=100, n_jobs=1,
                                              reg_lambda=1)))
estimator.append(('rfc', RandomForestClassifier(max_depth=30 , min_samples_leaf=4, min_sam
```

```
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
```

```
vot_hard = VotingClassifier(estimators = estimator, voting ='soft')
```

```
vot_hard.fit(finaltrain.drop(["label"],axis=1), finaltrain["label"])
y_pred = vot_hard.predict_proba(finaltest)[:,1]
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnin
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnin
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnin
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnin
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnin
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarnin
  FutureWarning,
```

```
y_pred
```

```
array([0.8377404 , 0.13295562, 0.20129892, ..., 0.16157927, 0.22413078,
       0.19646293])
```

```
sample_sum["label"]=y_pred
sample_sum.to_csv("./sumvoting.csv",index=False)
sample_sum.head()
```

	link_id	label
0	4049	0.837740
1	3692	0.132956
2	9739	0.201299
3	1548	0.698581
4	5574	0.933676