```
import numpy as np
import pandas as pd
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
onj = SentimentIntensityAnalyzer()
from textblob import TextBlob
import re
import chardet
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
%pip install scikit-learn
```

```
%pip install chardet
```

```
%pip install textblob
```

```
%pip install vaderSentiment
```

```
%pip install numpy
%pip install pandas
```

```
subreddit = [
    "gameofthrones",
    "aww",
    "gaming",
    "news",
    "politics",
    "dankmemes",
    "relationship_advice",
    "nba",
    "worldnews",
    "AskReddit",
    "AmItheAsshole",
    "SquaredCircle",
    "The_Donald",
    "leagueoflegends",
    "hockey",
    "videos",
    "teenagers",
    "gonewild",
    "movies",
    "funny",
    "pics",
    "marvelstudios",
    "memes",
    "soccer",
    "freefolk",
    "MortalKombat",
    "todayilearned",
    "apexlegends",
    "asoiaf",
    "Market76",
    "Animemes",
    "FortNiteBR",
    "nfl",
    "trashy",
    "unpopularopinion",
    "ChapoTrapHouse",
    "RoastMe",
    "Showerthoughts",
    "wallstreetbets",
    "Pikab",
]
```

```
subreddit_dict = {subreddit[i]:i for i in range(len(subreddit))}
```

Sentence

```
def detect_encoding(file_path):
    with open(file_path, 'rb') as file:
        result = chardet.detect(file.read())
    return result['encoding']

file_path = 'pre-processed-data.csv'
detected_encoding = detect_encoding(file_path)

df = pd.read_csv(file_path, encoding=detected_encoding)
```

```
df1 = df.copy()
```

```
df = df1.drop(columns=['score'])
```

```
df
```

	0	body
0	0	submission ha automatically removed post title...
1	1	dont squeeze massive hand mean giant
2	2	pretty known wa paid product placement hamilto...
3	3	know law currently correct willfully ignorant ...
4	4	yes difference gentle suppression hard suppres...
...	...	...
999996	39	–∏ –ª–∞ –∏ –Ω–µ—Ç –µ—Å–ª–∏ –∂–æ–ø–∞ –±—É–¥–µ—Ç...
999997	39	–∏—Å—Ö–æ–¥—è –∏–∑ —ç—Ç–æ–≥–æ —è —Ç–æ–≥–¥–∞ 3 —...
999998	39	—á–µ–ª–æ–≤–µ–∫ –∏–∑ –ª–∏–≥–∏ –ª–≥–±—Ç –ø–æ–¥–∏...
999999	39	—Å –ø–∏–∫–∞–±—É —É—à–µ–ª ,–Ω–µ –¥–ª—è —Ç–æ–≥–æ...
1000000	39	—è –¥—É–º–∞—é —Ç—É—Ç –≤—Å–µ–º –ø–æ—Ö—É–π

1000001 rows × 2 columns

df -->train and test below --?train data

```
from sklearn.model_selection import train_test_split

y = df['0']
X = df['body']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Train set shape:", X_train.shape, y_train.shape)
print("Test set shape:", X_test.shape, y_test.shape)
```

```
Train set shape: (800000,) (800000,)
Test set shape: (200001,) (200001,)
```

```
df_train = pd.concat([y_train,X_train], axis=1)
```

```
df_train.head()
```

	0	body
382311	17	love
241519	19	guys better massage known fact
719221	27	updated ps4
784387	26	bad asphalt ha little value 2019 making road r...
259727	3	repost r /hypocrite karma dinner

```
df_train.iloc[0]['body']
```

```
'love'
```

```
df_data = {'Subjectivity':[],'Polarity':[],'Neg':[],'Pos':[],'Compound':[],'Complexity':[],'Class':[]}
```

```
df_data
```

```
{'Subjectivity': [],
 'Polarity': [],
 'Neg': [],
 'Pos': [],
 'Compound': [],
 'Complexity': [],
 'Class': []}
```

```
def count_syllables(word):
    word = word.lower().strip()
    vowel_sounds = re.findall(r'[aeiouy]+', word)
    syllables = len(vowel_sounds)
    if word.endswith('e'):
        syllables -= 1
    if word.endswith('y') and not re.match(r'[aeiouy]+y$', word):
        syllables += 1
    return syllables

def flesch_kincaid_grade_level(text):
    words = text.split()
    sentences = re.split(r'[.?!]+', text.strip())

    avg_syllables = sum(count_syllables(word) for word in words) / len(words)
    avg_words_per_sentence = len(words) / len(sentences)

    fkgl = 0.39 * avg_words_per_sentence + 11.8 * avg_syllables - 15.59
    return round(fkgl, 2)
```

```
df_train.shape[0]
```

```
800000
```

```
df_train.iloc[0]
```

```
0         17
body    love
Name: 382311, dtype: object
```

```
for i in range(df_train.shape[0]):
    text = str(df_train.iloc[i]['body'])
    blob = TextBlob(text)
    df_data['Subjectivity'].append(blob.subjectivity)
    df_data['Polarity'].append(blob.sentiment.polarity)
    polarity_scores = onj.polarity_scores(text)
    df_data['Neg'].append(polarity_scores['neg'])
    df_data['Pos'].append(polarity_scores['pos'])
    df_data['Compound'].append(polarity_scores['compound'])
    fkgl = flesch_kincaid_grade_level(text)
    df_data['Complexity'].append(fkgl)
    df_data['Class'].append(df_train.iloc[i][0])
```

```
/var/folders/mw/60fg6l4s04b_w2zqvbz38b0c0000gn/T/ipykernel_82751/1005280258.py:12: FutureWarning: Series.__getitem__ tre
  df_data['Class'].append(df_train.iloc[i][0])
```

```
len(df_data['Subjectivity'])
```

```
800000
```

```
len(df_data['Class'])
```

```
800000
```

```
df_data
```

```
{'Subjectivity': [0.6,
  0.5,
  0.0,
  0.5833333333333333,
  0.0,
  0.39999999999999997,
  0.7999999999999999,
  0.0,
  0.5,
  0.55,
  0.25,
  0.0,
  1.0,
  0.0,
  0.0,
  0.0,
  0.75,
  0.8,
  0.3,
  0.8035714285714286,
  1.0,
  0.1,
  0.0,
  0.875,
  0.6,
  0.4,
  0.3333333333333333,
  0.6875,
  0.6000000000000001,
  0.65,
  0.4125,
  0.8,
  0.0,
  0.6772727272727272,
  0.2,
  0.0,
  0.0,
  0.0,
  1.0,
  0.0,
  0.7,
  0.0,
  0.3666666666666667,
  0.0,
  0.0,
  0.3831018518518519,
  0.0,
  0.0,
  0.0,
  0.0,
  0.7,
  0.475,
  0.0,
  0.4388888888888889,
  0.8,
  0.40952380952380957,
  0.0,
  0.5839285714285715
```

```
df_data_train = pd.DataFrame(df_data)
```

```
df_data_train.head()
```

	Subjectivity	Polarity	Neg	Pos	Compound	Complexity	Class
0	0.600000	0.50000	0.000	1.00	0.6369	-3.40	17
1	0.500000	0.50000	0.000	0.42	0.4404	2.88	19
2	0.000000	0.00000	0.000	0.00	0.0000	2.89	27
3	0.583333	-0.44375	0.233	0.30	0.0018	1.29	26
4	0.000000	0.00000	0.000	0.00	0.0000	7.60	3

```
df_data_train.to_csv('Mid_data_train.csv',index = False)
```

Training

```
data = pd.read_csv("Mid_data_train.csv")
```

```
features = data.iloc[:,:6]
labels = data.iloc[:,-1]
```

```
class_medians = features.groupby(labels).mean()
```

```
def return_params(text):
    blob = TextBlob(text)
    polarity_scores = onj.polarity_scores(text)
    fkgl = flesch_kincaid_grade_level(text)
    return [blob.subjectivity,blob.sentiment.polarity,polarity_scores['neg'],polarity_scores['pos'],polarity_scores['compour
```

```
def cosine_similarity(data_point1, data_point2):
    dot_product = np.dot(data_point1, data_point2)
    norm1 = np.linalg.norm(data_point1)
    norm2 = np.linalg.norm(data_point2)
    similarity = dot_product / (norm1 * norm2)
    return similarity
```

```
test set--?below
```

```
df_test = pd.concat([y_test,X_test], axis=1)
```

```
df_test
```

	0	body
624589	32	pff example ferrell listed 35 said *ferrell ra...
79954	35	started work 99 boomer younger asshole vigor b...
567130	29	combination following aae pump action shotgun ...
500891	8	compare pharma marketing budget r &amp;d, comp...
55399	8	wasn 't illegitimate election
...	...	...
639297	31	kevin *thoroughly* disliked cousin
311939	24	way saw wa wa waiting climb drogon "bran" like...
324459	21	friend mentioned idea black widow movie end cr...
390499	24	actually 15 probably 14 filming scene **bella ...
566853	2	agree completely better yes fun fairly connect...

200001 rows × 2 columns

```
test_data_point = []
for i in range(df_test.shape[0]):
    test_text = str(df_test.iloc[i]['body'])
    test_data_point.append(return_params(test_text))
```

```
# test_text = str(input("Enter the post whose subreddit you want to find: "))
# test_data_point = return_params(test_text)
```

```
class_medians
```

	Subjectivity	Polarity	Neg	Pos	Compound	Complexity
Class						
0	0.424170	0.070801	0.168653	0.191561	-0.006223	11.399015
1	0.397665	0.127278	0.091891	0.285472	0.229866	5.993868
2	0.400138	0.039763	0.125518	0.231560	0.157051	7.706284
3	0.423485	0.015515	0.204087	0.179719	-0.084070	11.973943
4	0.398060	0.028849	0.162510	0.184647	0.024188	12.290783
5	0.254376	0.026262	0.103331	0.173840	0.148960	6.557916
6	0.483256	0.082796	0.166887	0.275321	0.219649	12.085724
7	0.385596	0.027481	0.152500	0.211290	0.078084	5.478457
8	0.408799	0.037790	0.164516	0.186189	0.020916	13.682676
9	0.393976	0.049671	0.144552	0.204560	0.097077	9.393399
10	0.463969	0.048080	0.164463	0.215945	0.102395	12.579035
11	0.403419	0.074508	0.126961	0.220147	0.145090	8.000376
12	0.375551	0.022862	0.175161	0.183553	-0.007990	9.174007
13	0.416404	0.051511	0.136799	0.237002	0.178537	8.623096
14	0.405896	0.023025	0.155834	0.218639	0.077512	6.809493
15	0.429654	0.067772	0.146339	0.220097	0.096148	10.371355
16	0.346533	0.054153	0.130297	0.223912	0.094840	5.267911
17	0.459629	0.231103	0.086646	0.405931	0.315713	3.372760
18	0.433727	0.080265	0.140611	0.222059	0.134130	10.219436
19	0.358288	0.056598	0.132166	0.207414	0.085253	7.036787

```
len(test_data_point)
```

```
200001
```

```
list_sim = []
for j in range(len(test_data_point)):
    similarity = {}
    for i in range(class_medians.shape[0]):
        similarity[i] = cosine_similarity(test_data_point[j],list(class_medians.iloc[i]))
    list_sim.append(similarity)
```

```
/var/folders/mw/60fg6l4s04b_w2zqvbz38b0c0000gn/T/ipykernel_82751/1848760392.py:5: RuntimeWarning: invalid value encounte
  similarity = dot_product / (norm1 * norm2)
```

```
list_sim
```

```
[{0: 0.9945848426909374,
  1: 0.9899895007532586,
  2: 0.9925434243530014,
  3: 0.9951104386596538,
  4: 0.9941171739023635,
  5: 0.9919994519140725,
  6: 0.9926167730479974,
  7: 0.9932586442083119,
  8: 0.9940587855846826,
  9: 0.9935647040526497,
  10: 0.9936378807484314,
  11: 0.9928481613228304,
  12: 0.994709058112989,
  13: 0.9924803800988937,
  14: 0.9936549435166017,
  15: 0.9936677353342029,
  16: 0.9927694044256955,
  17: 0.9729587787852,
  18: 0.9932738190536943,
  19: 0.9935304505263797,
  20: 0.993415061023056,
  21: 0.9930970213074654,
  22: 0.9924921926175797,
  23: 0.9931700503767211,
  24: 0.9951157173867651,
  25: 0.9932172594809789,
  26: 0.9937413853964228,
  27: 0.9936769849473186,
  28: 0.9946563662497602,
  29: 0.9894401508382136,
  30: 0.993376325091424,
  31: 0.9930518993719555,
  32: 0.9922989703652496,
  33: 0.9947311079433256,
  34: 0.9938438066400119,
  35: 0.9942995496879817,
  36: 0.9935436149483418,
  37: 0.9928170327223272,
  38: 0.9931672407075919,
  39: -0.9921699213527637},
 {0: 0.9985317278746587,
  1: 0.9939631887899824,
  2: 0.9966718341049048,
  3: 0.9988831550663766,
  4: 0.9985013336881873,
  5: 0.9969620295288867,
  6: 0.9973028757510375,
  7: 0.9960016640842969,
  8: 0.9985638305520937,
  9: 0.9977764179313103,
  10: 0.9980657742654627,
  11: 0.9969201767521966,
  12: 0.9984375440016351,
  13: 0.9968254173102962,
  14: 0.9969061476213036,
  15: 0.9978517979309816,
  16: 0.9958667674181021,
```

```
top_10_elements = []
for i in range(len(list_sim)):
    top_10_elements.append(sorted(list_sim[i].items(), key=lambda x: x[1], reverse=True)[:10])
```

```
top_10_elements
```

```
[[(24, 0.9951157173867651),
  (3, 0.9951104386596538),
  (33, 0.9947311079433256),
  (12, 0.994709058112989),
  (28, 0.9946563662497602),
  (0, 0.9945848426909374),
  (35, 0.9942995496879817),
  (4, 0.9941171739023635),
  (8, 0.9940587855846826),
  (34, 0.9938438066400119)],
 [(3, 0.9988831550663766),
  (28, 0.9986335010136566),
  (8, 0.9985638305520937),
  (0, 0.9985317278746587),
  (4, 0.9985013336881873),
  (24, 0.9984679287428577),
  (12, 0.9984375440016351),
  (35, 0.9984221002775703),
  (34, 0.9982048406675993),
  (33, 0.9980934443320753)],
 [(3, 0.999063645208299),
  (8, 0.998954761262041),
  (28, 0.9988836744883081),
  (4, 0.9988578353300387),
  (0, 0.9987668551265354),
  (35, 0.9986957713554503),
  (12, 0.9985956457246898),
  (34, 0.9985613915605996),
  (24, 0.9985243276176995),
  (10, 0.9984447916384009)],
 [(3, 0.9994706662285131),
  (8, 0.9993515697370671),
  (28, 0.9993210563212066),
  (4, 0.9992929382715475),
  (0, 0.9992084360759734),
  (35, 0.9991718060145904),
  (12, 0.999109197110829),
  (34, 0.9990669722519603),
  (24, 0.999039419366928),
  (10, 0.9989721175814271)],
 [(8, 0.9993843956533232),
  (4, 0.999271389989514),
  (28, 0.9991202812233839),
  (3, 0.9990924375830259),
  (34, 0.9990876533235994),
  (10, 0.9990478761130076),
  (35, 0.9990463483978559),
  (0, 0.9990389604138162),
  (26, 0.9990039305098588),
  (37, 0.998808797043725)],
 [(9, 0.9999488067283877),
  (23, 0.9999483843587426),
  (30, 0.9999465140514324),
  (5, 0.9999456074039642),
  (6, 0.9999406053914097),
  (37, 0.9999372708510282),
  (18, 0.9999284689002179),
  (15, 0.9999259302899934)
```

Word parameters

Ishaan's code

```
token_dict={}

with open("pre-processed-data_`.csv", "r", encoding="utf8", errors="ignore") as f:

    k=f.readline()

    while(True):

        k=f.readline()

        if len(k)==0:
            break

        _,x,k=k.split(",",2)

        k=re.sub('[\d|\_]', '', k)

        token_dict[x]=k
```

```
tfidf = TfidfVectorizer(sublinear_tf=True, stop_words='english')
tfs = tfidf.fit_transform(token_dict.values())

feature_names = tfidf.get_feature_names_out()

dense=tfs.todense()
denselist = dense.tolist()
df = pd.DataFrame(denselist, columns=feature_names, index= list(token_dict.keys()))
```

```
subreddit_lists = {i:subreddit[i] for i in range(len(subreddit))}
```

```
df.head()
```

	aa	aaa	aaaa	aaaaa	aaaaaa	aaaaaaa	aaaaaaaa	aaaaaaaaa	aaaaaaaaaa	aaaaaaaaaaa	...	*but*	*di*
gameofthrones	0.007886	0.002547	0.000000	0.00000	0.000000	0.000000	0.0	0.0	0.00616	0.0	...	0.0	0.
aww	0.003819	0.000000	0.000000	0.00513	0.006479	0.006479	0.0	0.0	0.00000	0.0	...	0.0	0.
gaming	0.006342	0.013756	0.004896	0.00000	0.000000	0.000000	0.0	0.0	0.00000	0.0	...	0.0	0.
news	0.007556	0.002270	0.000000	0.00000	0.000000	0.000000	0.0	0.0	0.00000	0.0	...	0.0	0.
politics	0.003848	0.004966	0.000000	0.00000	0.000000	0.000000	0.0	0.0	0.00000	0.0	...	0.0	0.

```
s = pd.Series(df.loc[subreddit])
```

```
    aa        0.007633
    aaa       0.000000
    aaaa      0.000000
    aaaaa     0.000000
    aaaaaa    0.000000
                 ...
    the      0.011703
    wolf     0.011703
    Jリ        0.000000
    JJ       0.000000
    you      0.000000
    Name: freefolk, Length: 239095, dtype: float64
```

```
inputsubs
```

```
    ['freefolk',
     'news',
     'trashy',
     'The_Donald',
     'asoiaf',
     'gameofthrones',
     'ChapoTrapHouse',
     'politics',
     'worldnews',
     'unpopularopinion']
```

```
subreddit
```

```
'Pikab'
```

```
df
```

	aa	aaa	aaaa	aaaaa	aaaaaa	aaaaaaa	aaaaaaaa	aaaaaaaaa	aaaaaaaaaa	aaaaaaaaaaa	...
gameofthrones	0.007886	0.002547	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.006160	0.000000	...

```
list_indices = df.index
```

```
list_indices
```

```
Index(['gameofthrones', 'aww', 'gaming', 'news', 'politics', 'dankmemes',
       'relationship_advice', 'nba', 'worldnews', 'AskReddit', 'AmItheAsshole',
       'SquaredCircle', 'The_Donald', 'leagueoflegends', 'hockey', 'videos',
       'teenagers', 'gonewild', 'movies', 'funny', 'pics', 'marvelstudios',
       'memes', 'soccer', 'freefolk', 'MortalKombat', 'todayilearned',
       'apexlegends', 'asoiaf', 'Market76', 'Animemes', 'FortNiteBR', 'nfl',
       'trashy', 'unpopularopinion', 'ChapoTrapHouse', 'RoastMe',
       'Showerthoughts', 'wallstreetbets', 'Pikabu'],
      dtype='object')
```

```
list_indices = {i:i for i in list_indices}
```

```
list_indices
```

```
{'gameofthrones': 'gameofthrones',
 'aww': 'aww',
 'gaming': 'gaming',
 'news': 'news',
 'politics': 'politics',
 'dankmemes': 'dankmemes',
 'relationship_advice': 'relationship_advice',
 'nba': 'nba',
 'worldnews': 'worldnews',
 'AskReddit': 'AskReddit',
 'AmItheAsshole': 'AmItheAsshole',
 'SquaredCircle': 'SquaredCircle',
 'The_Donald': 'The_Donald',
 'leagueoflegends': 'leagueoflegends',
 'hockey': 'hockey',
 'videos': 'videos',
 'teenagers': 'teenagers',
 'gonewild': 'gonewild',
 'movies': 'movies',
 'funny': 'funny',
 'pics': 'pics',
 'marvelstudios': 'marvelstudios',
 'memes': 'memes',
 'soccer': 'soccer',
 'freefolk': 'freefolk',
 'MortalKombat': 'MortalKombat',
 'todayilearned': 'todayilearned',
 'apexlegends': 'apexlegends',
 'asoiaf': 'asoiaf',
 'Market76': 'Market76',
 'Animemes': 'Animemes',
 'FortNiteBR': 'FortNiteBR',
 'nfl': 'nfl',
 'trashy': 'trashy',
 'unpopularopinion': 'unpopularopinion',
 'ChapoTrapHouse': 'ChapoTrapHouse',
 'RoastMe': 'RoastMe',
 'Showerthoughts': 'Showerthoughts',
 'wallstreetbets': 'wallstreetbets',
 'Pikabu': 'Pikabu'}
```

```
list_indices['Pikabu'] = 'Pikab'
```

```
list_indices
```

```
{'gameofthrones': 'gameofthrones',
 'aww': 'aww',
 'gaming': 'gaming',
 'news': 'news',
 'politics': 'politics',
 'dankmemes': 'dankmemes',
 'relationship_advice': 'relationship_advice',
 'nba': 'nba',
 'worldnews': 'worldnews',
 'AskReddit': 'AskReddit',
 'AmItheAsshole': 'AmItheAsshole',
 'SquaredCircle': 'SquaredCircle',
 'The_Donald': 'The_Donald',
 'leagueoflegends': 'leagueoflegends',
 'hockey': 'hockey',
 'videos': 'videos',
 'teenagers': 'teenagers',
 'gonewild': 'gonewild',
 'movies': 'movies',
 'funny': 'funny',
 'pics': 'pics',
 'marvelstudios': 'marvelstudios',
 'memes': 'memes',
 'soccer': 'soccer',
 'freefolk': 'freefolk',
 'MortalKombat': 'MortalKombat',
 'todayilearned': 'todayilearned',
 'apexlegends': 'apexlegends',
 'asoiaf': 'asoiaf',
 'Market76': 'Market76',
 'Animemes': 'Animemes',
 'FortNiteBR': 'FortNiteBR',
 'nfl': 'nfl',
 'trashy': 'trashy',
 'unpopularopinion': 'unpopularopinion',
 'ChapoTrapHouse': 'ChapoTrapHouse',
 'RoastMe': 'RoastMe',
 'Showerthoughts': 'Showerthoughts',
 'wallstreetbets': 'wallstreetbets',
 'Pikabu': 'Pikab'}
```

```
df = df.rename(index=list_indices)
```

```
df.index[39]
```

```
'Pikab'
```

```
list_impwords = []
for k in range(len(top_10_elements)):
    inputsubs =[subreddit_lists[top_10_elements[k][i][0]] for i in range(len(top_10_elements[k]))]
    # print(inputsubs)
    impwords={}
    for subreddit in inputsubs:
        s = pd.Series(df.loc[subreddit])
        # print(s)
        impwords[subreddit]=s[s > 0.0001].sort_values(ascending=False)[:100].keys().tolist()
    list_impwords.append(impwords)
```

```
def count_matching_words(words, text):
  count = 0
  for word in text.lower().split():
    if word in words:
      count += 1
  return count
```

```
list_most_prob_sub = []

for impwords in list_impwords:
    count = 0
    most_prob_sub = 0
    for i in impwords:
        if count<count_matching_words(impwords[i],test_text):
            count = count_matching_words(impwords[i],test_text)
            most_prob_sub = i
    list_most_prob_sub.append(most_prob_sub)
```

```
len(list_most_prob_sub)
```

```
200001
```

```
top_10_elements[1][0][0]
```

```
3
```

```
list(y_test)
```

```
[32,
 35,
 29,
 8,
 8,
 21,
 13,
 11,
 33,
 19,
 25,
 6,
      26,
      25,
      12,
      33,
      24,
      17,
      36,
      37,
      35,
      20,
      3,
      32,
      23,
      29,
      0,
      27,
      4,
      1,
      30,
      3,
      25,
      19,
      6,
      10,
      8,
      9,
      28,
      15,
      22,
      31,
      8,
      29,
      4,
      35,
      23,
      15,
      6,
      26,
      31,
      31,
      7,
      18,
      30,
      32,
      3,
      7
```

```
total = len(y_test)
positives = 0
for i in range(total):
    print(i)
    if top_10_elements[i][0][0] == list(y_test)[i]:
        positives+=1

accuracy = positives/total
print("accuracy of the model is ",accuracy*100,"%")
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
```

```
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
```

```
positives

    12477
```

```
total

    200001
```

You dont need to add the next part, it is for further extensions

```
import pandas as pd
from textblob import TextBlob
import re


def analyze_text(text):
    """
    Analyzes a text and returns a dictionary containing values for 6 dimensions:
        - joy
        - anger
        - complexity (average number of syllables per word)
    """
    # Create a TextBlob object
    blob = TextBlob(text)

    # Calculate emotion scores (range: 0-1)
    joy = blob.sentiment.polarity
    anger = blob.sentiment.subjectivity

    # Calculate word complexity (average syllables per word)
    syllables = sum(count_syllables(word) for word in text.split())
    word_count = len(text.split())
    complexity = syllables / word_count if word_count else 0


    # Return dictionary with analysis results
    return {
        "joy": joy,
        "anger": anger,
        "complexity": complexity,
    }


def count_syllables(word):
    # Remove punctuation and convert to lowercase
    word = word.lower().strip()

    # Count vowel sounds
    vowel_sounds = re.findall(r"[aeiouy]+", word)

    # Count syllables (assume consonant sounds between vowel sounds)
    syllables = len(vowel_sounds)

    # Special cases for silent "e" and "y"
    if word.endswith("e"):
        syllables -= 1
    if word.endswith("y") and not re.match(r"[aeiouy]+y$", word):
        syllables += 1

    return syllables


def flesch_kincaid_grade_level(text):

    words = text.split()
    sentences = re.split(r"[.?!]+", text.strip())

    avg_syllables = sum(count_syllables(word) for word in words) / len(words)
    avg_words_per_sentence = len(words) / len(sentences)

    fkgl = 0.39 * avg_words_per_sentence + 11.8 * avg_syllables - 15.59

    return round(fkgl, 2)

text = "Supercalifragilisticexpialidocious"
analysis_data = analyze_text(text)

print(analysis_data)
```