```
### Wikipedia abstracts on json file format to classify people by their profession
```

The input for training is a file wiki-train.json, which contains Wikipedia abstracts in the following form: {"title": "George_Washington", "summary": "George Washington was one of the ..." "occupations": ["yago:politician"]}

The input for testing is a file wiki-test.json, which contains Wikipedia abstracts of the same shape without the occupations: {"title": "Douglas_Adams", "summary": "Douglas Noel Adams was ..."}

The **training** dataset has the labels
The **development** dataset has the labels
The **testing** dataset does not have the labels

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import warnings
```

```
warnings.filterwarnings('ignore')
```

```
np.version.version
```

```
    '1.23.5'
```

```
### !mkdir ~/.kaggle
```

```
####!cp /kaggle.json ~/.kaggle/
```

```
###!kaggle datasets download -d angevalli/wikipedia-abstracts
```

```
###! unzip /content/wikipedia-abstracts.zip
```

```
###! pip install bokeh==2.4.0
```

```
%pylab inline
import pandas as pd
import numpy as np
import os
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.manifold import TSNE
from pprint import pprint
from gensim.models import Phrases, LdaModel
from gensim.corpora import Dictionary
import nltk
from nltk.stem import WordNetLemmatizer, SnowballStemmer
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import brown
from nltk import FreqDist
from collections import OrderedDict
from bokeh.plotting import figure, show, output_notebook, save
from bokeh.models import HoverTool, value, LabelSet, Legend, ColumnDataSource
```

```
    Populating the interactive namespace from numpy and matplotlib
```

```
wiki_train = pd.read_json("/content/wiki-train.json/new_wiki-train.json", lines=True)
```

```
wiki_test = pd.read_json("/content/wiki-test.json/new_wiki-test.json", lines=True)
```

```
print(wiki_train.shape, wiki_test.shape)
```

```
    (266938, 3) (201406, 2)
```

```
print(wiki_train.columns, wiki_test.columns)
```

```
    Index(['title', 'summary', 'occupations'], dtype='object') Index(['title', 'summary'], dtype='object')
```

## Text Pre-Processing

```
import pandas, numpy, string, textblob
import pickle
from sklearn import model_selection, preprocessing, linear_model, naive_bayes, metrics, svm, decomposition, ensemble
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import xgboost
from keras import layers, models, optimizers
from keras.preprocessing import text, sequence
import matplotlib.pyplot as plt
```

```
###! pip install unidecode
```

```
import re, unidecode
from bs4 import BeautifulSoup
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Needed only once
# import nltk
# nltk.download('stopwords')
# nltk.download('punkt')
# nltk.download('wordnet')

def remove_html_tags(text):
    soup = BeautifulSoup(text, "html.parser")
    stripped_text = soup.get_text(separator=" ")
    return stripped_text
def remove_accented_chars(text):
    text = unidecode.unidecode(text)
    return text
def remove_numbers(text):
    result = re.sub(r'\d+', '', text)
    return result
def remove_slash_with_space(text):
    return text.replace('\\', " ")
def remove_punctuation(text):
    translator = str.maketrans('', '', string.punctuation)
    return text.translate(translator)
def text_lowercase(text):
    return text.lower()
def remove_whitespace(text):
    return  " ".join(text.split())
```

```
def remove_stopwords(text):
    stop_words = set(stopwords.words("english"))
    word_tokens = word_tokenize(text)
    filtered_text = [word for word in word_tokens if word not in stop_words]
    return ' '.join(filtered_text)
def stem_words(text):
    stemmer = PorterStemmer()
    word_tokens = word_tokenize(text)
    stems = [stemmer.stem(word) for word in word_tokens]
    return ' '.join(stems)
def lemmatize_words(text):
    lemmatizer = WordNetLemmatizer()
    word_tokens = word_tokenize(text)
    # provide context i.e. part-of-speech
    lemmas = [lemmatizer.lemmatize(word, pos ='v') for word in word_tokens]
    return ' '.join(lemmas)
```

```
wiki_train.columns
```

```
     Index(['title', 'summary', 'occupations'], dtype='object')
```

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

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

```
# Perform preprocessing
def perform_preprocessing(text):
    text = remove_html_tags(text)
    text = remove_accented_chars(text)
    text = remove_numbers(text)
    text = remove_stopwords(text)
    text = text_lowercase(text)
    text = remove_slash_with_space(text)
    text = remove_punctuation(text)
    # text = stem_words(text)
    text = lemmatize_words(text)
    text = remove_whitespace(text)
    return text
```

```
wiki_train['summary'] = wiki_train['summary'].apply(perform_preprocessing)
```

```
wiki_test['summary'] = wiki_test['summary'].apply(perform_preprocessing)
```

```
##wiki_train['title'] = wiki_train['title'].apply(perform_preprocessing)
```

```
###wiki_test['title'] = wiki_test['title'].apply(perform_preprocessing)
```

```
wiki_train.isnull().sum()
```

```
     title          0
     summary        0
     occupations    0
     dtype: int64
```

```
wiki_train['occupations']=wiki_train['occupations'].apply(str)
```

```
def clean_html(html):

    # parse html content
    soup = BeautifulSoup(html, "html.parser")

    for data in soup(['style', 'script', 'code', 'a']):
        # Remove tags
        data.decompose()

    # return data by retrieving the tag content
    return ' '.join(soup.stripped_strings)
```

```
#wiki_train['occupations'] = wiki_train['occupations'].apply(lambda x: clean_html(x))
```

```
import spacy
```

```
# Load spacy
nlp = spacy.load('en_core_web_sm')

def clean_string(text, stem="None"):

    final_string = ""

    # Make lower
    text = text.lower()

    # Remove line breaks
    text = re.sub(r'\n', '', text)
    text = re.sub(r'yago:', '', text)

    # Remove puncuation
    translator = str.maketrans('', '', string.punctuation)
    text = text.translate(translator)

    # Remove stop words
    text = text.split()
    useless_words = nltk.corpus.stopwords.words("english")
    useless_words = useless_words + ['hi', 'im']

    text_filtered = [word for word in text if not word in useless_words]

    # Remove numbers
    text_filtered = [re.sub(r'\w*\d\w*', '', w) for w in text_filtered]

    # Stem or Lemmatize
    if stem == 'Stem':
        stemmer = PorterStemmer()
        text_stemmed = [stemmer.stem(y) for y in text_filtered]
    elif stem == 'Lem':
        lem = WordNetLemmatizer()
        text_stemmed = [lem.lemmatize(y) for y in text_filtered]
    elif stem == 'Spacy':
        text_filtered = nlp(' '.join(text_filtered))
        text_stemmed = [y.lemma_ for y in text_filtered]
    else:
        text_stemmed = text_filtered

    final_string = ' '.join(text_stemmed)
    return final_string
```

```
wiki_train['occupations'] = wiki_train['occupations'].apply(lambda x: clean_string(x, stem='Stem'))
```

```
mask = wiki_train['occupations'].map(wiki_train['occupations'].value_counts()) < 100
wiki_train['occupations'] =  wiki_train['occupations'].mask(mask, 'other')
```

```
wiki_train['occupations'].value_counts()
```

```
politician                           54845
footballplay                         49957
actor                                13495
writer                               12993
painter                              11610
                                     ...
poet compos                            110
universityteach lawyer                 107
journalist lawyer                      106
writer journalist historian            105
actor singer filmactor musician        102
Name: occupations, Length: 89, dtype: int64
```

```
wiki_train['occupations'].unique()
```

```
array(['politician', 'writer poet',
       'actor filmactor filmdirector screenwrit', 'actor filmactor',
       'politician historian', 'politician militarypersonnel',
       'universityteach historian', 'other', 'footballplay',
       'filmactor filmdirector screenwrit', 'universityteach compos',
       'actor filmactor filmdirector', 'historian', 'writer',
       'compos musician', 'compos', 'writer historian', 'painter',
       'politician writer poet', 'singer compos musician',
       'universityteach', 'writer journalist',
       'actor filmdirector screenwrit', 'businessperson',
       'singer filmactor', 'singer', 'politician lawyer',
       'singer musician', 'actor singer filmactor',
       'writer journalist poet', 'actor singer', 'filmactor',
       'militarypersonnel', 'politician businessperson', 'actor',
       'universityteach physician', 'writer compos',
       'actor filmactor screenwrit', 'politician poet', 'singer compos',
       'journalist', 'musician', 'physician', 'writer screenwrit',
       'politician writer', 'painter universityteach',
       'politician universityteach', 'actor singer musician',
       'writer filmdirector screenwrit',
       'actor singer filmactor musician', 'filmdirector screenwrit',
       'poet', 'politician actor', 'politician journalist',
       'writer universityteach', 'lawyer', 'politician writer journalist',
       'politician universityteach lawyer', 'writer physician',
       'politician physician', 'writer actor', 'journalist historian',
       'universityteach lawyer', 'filmdirector', 'screenwrit',
       'writer businessperson', 'writer painter',
       'writer journalist screenwrit', 'writer journalist historian',
       'filmactor screenwrit', 'writer universityteach historian',
       'actor filmdirector', 'filmactor filmdirector',
       'writer painter poet', 'writer actor filmactor', 'actor musician',
       'actor screenwrit', 'painter poet', 'actor singer compos',
       'poet compos', 'writer universityteach poet', 'research',
       'journalist poet', 'actor journalist', 'journalist screenwrit',
       'politician journalist lawyer', 'journalist lawyer',
       'actor compos', 'writer lawyer'], dtype=object)
```

## Word Cloud

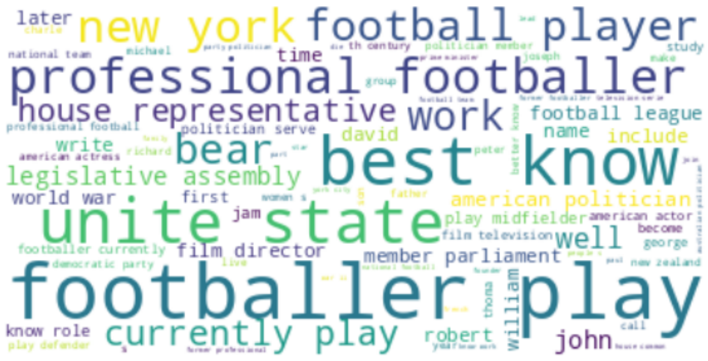
```
import wordcloud
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from nltk.corpus import stopwords
```

```
wiki_train.columns
```

```
Index(['title', 'summary', 'occupations'], dtype='object')
```

```
texts1 = " ".join(summary_values for summary_values in wiki_train.summary)
```

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
stopwords = set(STOPWORDS)
stopwords = stopwords.union(["ha", "thi", "now", "onli", "im", "becaus", "wa", "will", "even", "go", "realli", "didnt", "abl"])
wordcl = WordCloud(stopwords = stopwords, background_color='white', max_font_size = 50, max_words = 5000).generate(texts1)
plt.figure(figsize=(7, 5))
plt.imshow(wordcl, interpolation='bilinear')
plt.axis('off')
plt.show()
```



```
wiki_train.columns
```

```
Index(['title', 'summary', 'occupations', 'summary_len', 'summary_count'], dtype='object')
```

```
wiki_train['summary_len'] = wiki_train['summary'].astype(str).apply(len)
wiki_train['summary_count'] = wiki_train['summary'].apply(lambda x: len(str(x).split()))
```

```
wiki_train[['summary_len', 'summary_count']].hist(bins=20, figsize=(13, 3), color='red')
```

```
    array([[<Axes: title={'center': 'summary_len'}>,
            <Axes: title={'center': 'summary_count'}>]], dtype=object)
```

```
wiki_train.columns
```

```
    Index(['title', 'summary', 'occupations', 'summary_len', 'summary_count'], dtype='object')
```

```
wiki_train.to_csv("wikipaedia_train.csv")
```

**Feature Engineering**

Text files are actually series of words (ordered). In order to run machine learning algorithms we need to convert the text files into numerical feature vectors.

We will implement the following different ideas in order to obtain relevant features from our dataset.

CountVectors – I have used scikit-learn library's CountVectorizer module to vectorize sentences. It generates vocabulary for all unique words of sentence. From this count of words, a feature vector is created. This essentially is the Bag of Words BOW model.

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
wiki_train.head(2)
```

| | title | summary | occupations | summary_len | summary_count |
|---|---|---|---|---|---|
| 0 | George_Washington | george washington one found father unite state... | 42 | 309 | 43 |
| 1 | Pierre_Corneille | pierre corneille french tragedian he generally... | 84 | 129 | 16 |

```
c = wiki_train["occupations"].astype('category')

d = dict(enumerate(c.cat.categories))
print (d)
```

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

```
wiki_train["occupations"] = wiki_train["occupations"].astype('category').cat.codes
```

```
wiki_train.head(3)
```

| | title | summary | occupations | summary_len | summary_count |
|---|---|---|---|---|---|
| 0 | George_Washington | george washington one found father unite state... | 42 | 309 | 43 |
| 1 | Pierre_Corneille | pierre corneille french tragedian he generally... | 84 | 129 | 16 |
| 2 | Andrei_Tarkovsky | andrei arsenyevich tarkovsky russian filmmaker... | 4 | 366 | 47 |

```
wiki_test['summary_len'] = wiki_test['summary'].astype(str).apply(len)
wiki_test['summary_count'] = wiki_test['summary'].apply(lambda x: len(str(x).split()))
```

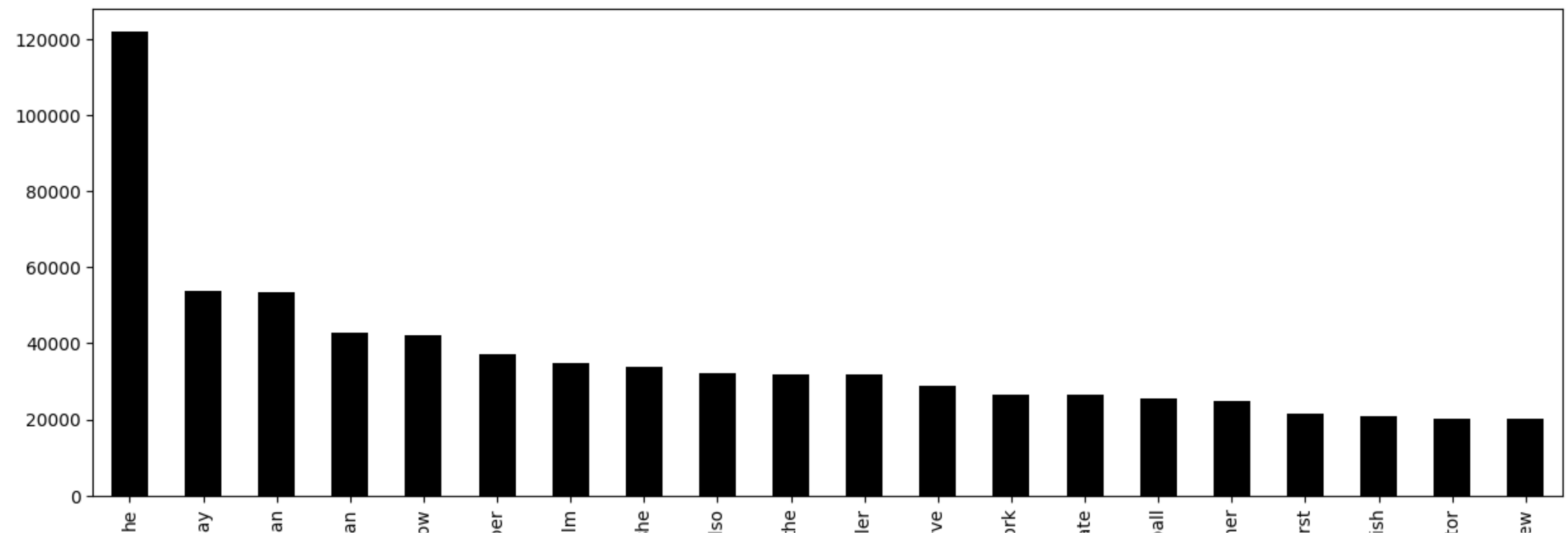
```
wiki_test["occupations"] = 0
```

## Top - N -Words

```
def get_top_n_words(corpus, n=None):
    vec=CountVectorizer().fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
common_words = get_top_n_words(wiki_train['summary'], 20)
wiki_train1 = pd.DataFrame(common_words, columns = ['summary', 'count'])
wiki_train1.head()
```

| | summary | count |
|---|---|---|
| 0 | he | 121941 |
| 1 | play | 53750 |
| 2 | american | 53445 |
| 3 | politician | 42914 |
| 4 | know | 42233 |

```
wiki_train1.groupby('summary').sum()['count'].sort_values(ascending=False).plot(kind='bar',color='black',figsize = (15, 5))
xlabel = 'Top Words'
ylabel = 'Count'
title = 'BarChart represent the Top Words Frequency'
plt.show()
```

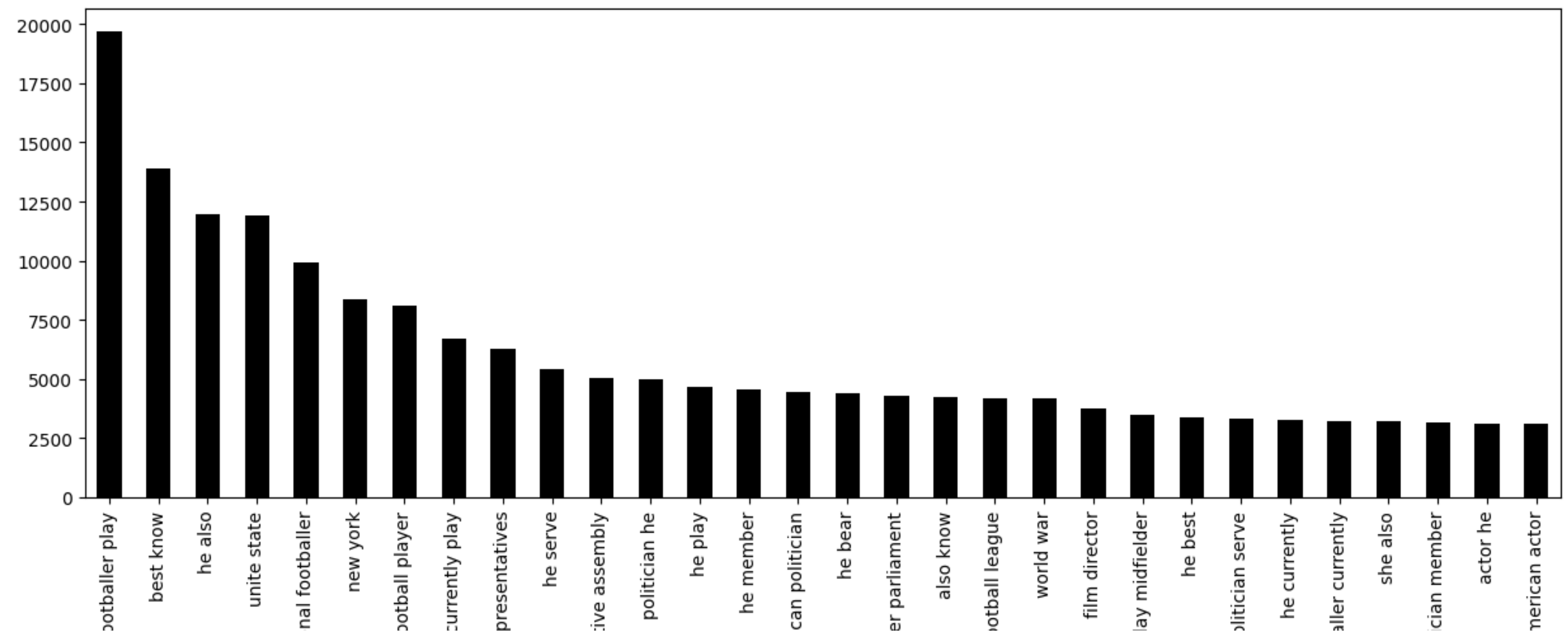


## Bi-Gram Frequency Of Words

```
def get_top_n_bigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(2,2)).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
common_words2 = get_top_n_bigram(wiki_train['summary'], 30)
wiki_train2 = pd.DataFrame(common_words2, columns=['summary', "Count"])
wiki_train2.head()
```

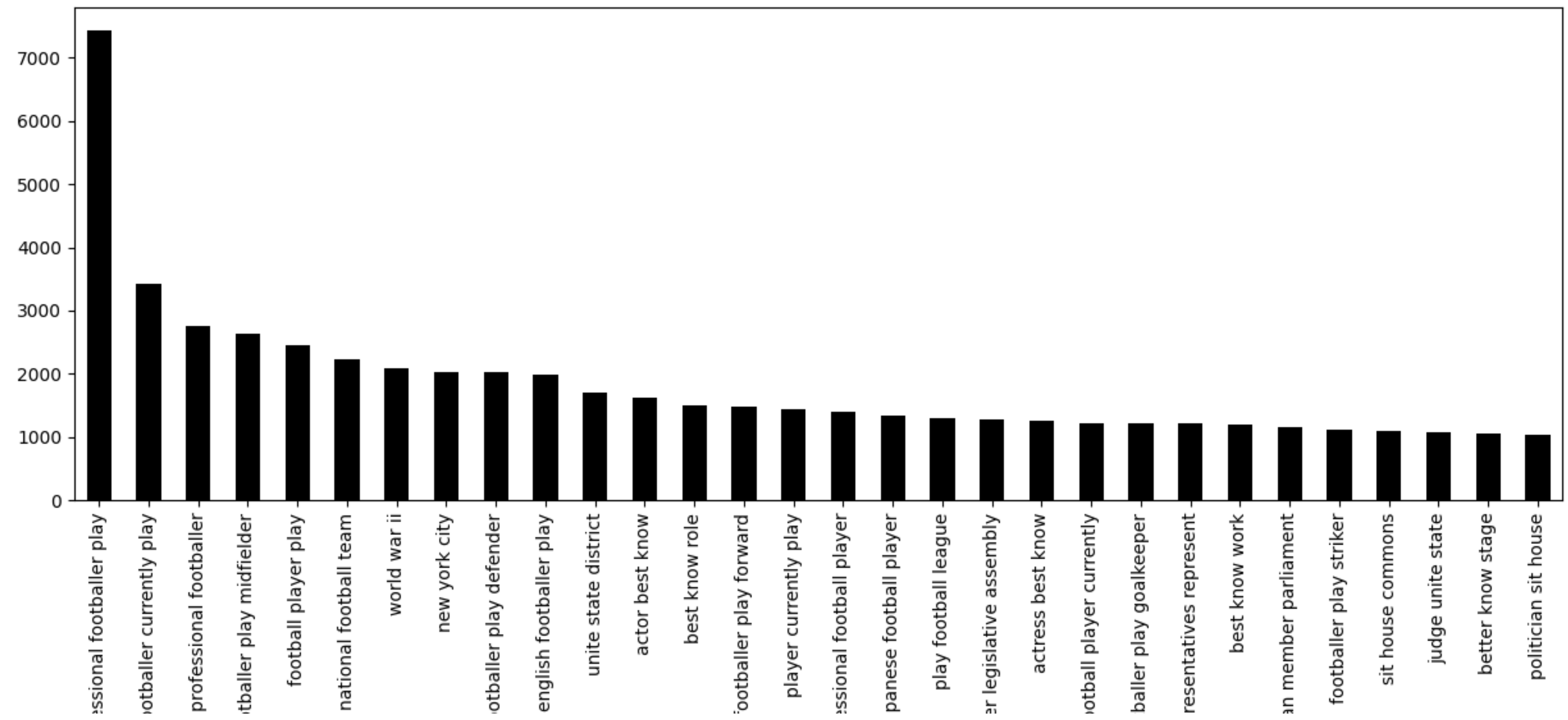
|   | summary | Count |
|---|---|---|
| 0 | footballer play | 19671 |
| 1 | best know | 13896 |
| 2 | he also | 11993 |
| 3 | unite state | 11937 |
| 4 | professional footballer | 9935 |

```
wiki_train2.groupby('summary').sum()['Count'].sort_values(ascending=False).plot(kind='bar',figsize=(15,5), color='black')
xlabel = "Bigram Words"
ylabel = "Count"
title = "Bar chart of Bigrams Frequency"
plt.show()
```



## Tri-gram Frequency Of Words

```
def get_top_n_trigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(3, 3), stop_words='english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
common_words5 = get_top_n_trigram(wiki_train['summary'], 30)
wiki_train4 = pd.DataFrame(common_words5, columns = ['summary' , 'Count'])
wiki_train4.groupby('summary').sum()['Count'].sort_values(ascending=False).plot(kind='bar',figsize=(15,5), color='black')
xlabel = "Trigram Words"
ylabel = "Count"
title = "Bar chart of Trigrams Frequency"
plt.show()
```



```
wiki_train.columns
```

```
Index(['title', 'summary', 'occupations', 'summary_len', 'summary_count'], dtype='object')
```

```
wiki_train.head(3)
```

|   | title | summary | occupations | summary_len | summary_count |
|---|---|---|---|---|---|
| 0 | George_Washington | george washington one found father unite state... | 42 | 309 | 43 |
| 1 | Pierre_Corneille | pierre corneille french tragedian he generally... | 84 | 129 | 16 |
| 2 | Andrei_Tarkovsky | andrei arsenyevich tarkovsky russian filmmaker... | 4 | 366 | 47 |

```
wiki_test.columns
```

```
Index(['title', 'summary', 'summary_len', 'summary_count', 'occupations'], dtype='object')
```

```
wiki_test.head(3)
```

|   | title | summary | summary_len | summary_count | occupations |
|---|---|---|---|---|---|
| 0 | Abou_Ouattara | ben qadir abou ouattara burkinabe internationa... | 88 | 12 | 0 |
| 1 | Jorge_Pereira | jorge javier moreira pereira portuguese profes... | 92 | 11 | 0 |
| 2 | Emma_Sheridan_Fry | emma sheridan fry american actor playwright te... | 313 | 46 | 0 |

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

```
tfidf = TfidfVectorizer(max_features = 500,
                        ngram_range = (1,3),
                              stop_words = "english")
X_train_title = tfidf.fit_transform(wiki_train["title"].tolist())
```

```
tfidf = TfidfVectorizer(max_features = 500,
                              ngram_range = (1,3),
                              stop_words = "english")
X_test_title = tfidf.fit_transform(wiki_test["title"].tolist())
```

```
tfidf = TfidfVectorizer(max_features = 500,
                              ngram_range = (1,3),
                              stop_words = "english")
X_train_summary = tfidf.fit_transform(wiki_train["summary"].tolist())
```

```
tfidf = TfidfVectorizer(max_features = 500,
                              ngram_range = (1,3),
                              stop_words = "english")
X_test_summary = tfidf.fit_transform(wiki_test["summary"].tolist())
```

```
import scipy
```

```
wiki_train.columns
```

```
     Index(['title', 'summary', 'occupations', 'summary_len', 'summary_count'], dtype='object')
```

```
wiki_train.head(2)
```

| | title | summary | occupations | summary_len | summary_count |
|---|---|---|---|---|---|
| **0** | George_Washington | george washington one found father unite state... | 42 | 309 | 43 |
| **1** | Pierre_Corneille | pierre corneille french tragedian he generally... | 84 | 129 | 16 |

```
X_train = scipy.sparse.hstack((X_train_title,
                         X_train_summary,
                         wiki_train[["occupations", "summary_len", "summary_count"]].to_numpy())).tocsr()
```

```
X_test = scipy.sparse.hstack((X_test_title,
                         X_test_summary,
                         wiki_test[["occupations", "summary_len", "summary_count"]].to_numpy())).tocsr()
```

```
Y_train = wiki_train['occupations']
```

```
Y_test = wiki_test['occupations']
```

## Random Forest Classifier

```
##### Create Model Model ######
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, recall_score, classification_report, cohen_kappa_score
from sklearn import metrics

# Baseline Random forest based Model
rfc = RandomForestClassifier()
rfcg = rfc.fit(X_train,Y_train) # fit on training data
```

```
####### Prediction ##########
predictions = rfcg.predict(X_test)
print('Baseline: Accuracy: ', round(accuracy_score(Y_test, predictions)*100, 2))
print('\n Classification Report:\n', classification_report(Y_test,predictions))
```

```
     Baseline: Accuracy:  78.9

      Classification Report:
                   precision    recall  f1-score   support

               0       1.00      0.79      0.88    201406
               1       0.00      0.00      0.00         0
               2       0.00      0.00      0.00         0
              11       0.00      0.00      0.00         0
              12       0.00      0.00      0.00         0
              16       0.00      0.00      0.00         0
              17       0.00      0.00      0.00         0
              25       0.00      0.00      0.00         0
              26       0.00      0.00      0.00         0
              27       0.00      0.00      0.00         0
              32       0.00      0.00      0.00         0
              33       0.00      0.00      0.00         0
              34       0.00      0.00      0.00         0
              35       0.00      0.00      0.00         0
              36       0.00      0.00      0.00         0
              39       0.00      0.00      0.00         0
              40       0.00      0.00      0.00         0
              42       0.00      0.00      0.00         0
              59       0.00      0.00      0.00         0

        accuracy                           0.79    201406
       macro avg       0.05      0.04      0.05    201406
    weighted avg       1.00      0.79      0.88    201406
```

## XGBoost Classifier

```
###! pip install xgboost
```

```
import xgboost as xgb
import pandas as pd
import numpy as np
```

```
from xgboost import XGBClassifier
```

```
xgb_clf2 = XGBClassifier(n_estimators=20, learning_rate=0.5, max_features=2, max_depth=2, random_state=0)
xgb_clf2.fit(X_train, Y_train)
predictions_xgb = xgb_clf2.predict(X_test)
```

```
print('Baseline: Accuracy: ', round(accuracy_score(Y_test, predictions_xgb)*100, 2))
print('\n Classification Report:\n', classification_report(Y_test,predictions_xgb))
```

```
     Baseline: Accuracy:  83.89

      Classification Report:
                   precision    recall  f1-score   support

               0       1.00      0.84      0.91    201406
              36       0.00      0.00      0.00         0
              42       0.00      0.00      0.00         0

        accuracy                           0.84    201406
       macro avg       0.33      0.28      0.30    201406
     weighted avg       1.00      0.84      0.91    201406
```

```
predictions_xgb = pd.DataFrame(predictions_xgb)
```

```
predictions = pd.DataFrame(predictions)
```

```
predictions_xgb.rename(columns = {0 : "Predict"}, inplace=True)
```

```
predictions.rename(columns = {0:"Predict"}, inplace=True)
```

```
predictions.shape
```

```
     (201406, 1)
```

```
predictions.columns
```

```
     Index(['Predict'], dtype='object')
```

```
predictions.value_counts()
```

```
     Predict
     0          158916
     25          16507
     42          10670
     17           6346
     16           3917
     36           2027
     27            946
     32            886
     35            499
     33            433
     26            145
     11             66
     34             15
     59             13
     40              9
     1               6
     39              3
     12              1
     2               1
     dtype: int64
```

```
output = {

0: 'actor', 1: 'actor compos', 2: 'actor filmactor', 3: 'actor filmactor filmdirector',
4: 'actor filmactor filmdirector screenwrit', 5: 'actor filmactor screenwrit',
6: 'actor filmdirector', 7: 'actor filmdirector screenwrit', 8: 'actor journalist',
9: 'actor musician', 10: 'actor screenwrit', 11: 'actor singer', 12: 'actor singer compos',
13: 'actor singer filmactor', 14: 'actor singer filmactor musician', 15: 'actor singer musician',
16: 'businessperson', 17: 'compos', 18: 'compos musician', 19: 'filmactor', 20: 'filmactor filmdirector',
21: 'filmactor filmdirector screenwrit', 22: 'filmactor screenwrit', 23: 'filmdirector', 24: 'filmdirector screenwrit',
25: 'footballplay', 26: 'historian', 27: 'journalist', 28: 'journalist historian', 29: 'journalist lawyer',
30: 'journalist poet', 31: 'journalist screenwrit', 32: 'lawyer', 33: 'militarypersonnel', 34: 'musician',
35: 'other', 36: 'painter', 37: 'painter poet', 38: 'painter universityteach', 39: 'physician', 40: 'poet',
41: 'poet compos', 42: 'politician', 43: 'politician actor', 44: 'politician businessperson', 45: 'politician historian',
46: 'politician journalist', 47: 'politician journalist lawyer', 48: 'politician lawyer', 49: 'politician militarypersonnel',
50: 'politician physician', 51: 'politician poet', 52: 'politician universityteach', 53: 'politician universityteach lawyer',
54: 'politician writer', 55: 'politician writer journalist', 56: 'politician writer poet', 57: 'research', 58: 'screenwrit', 59: 'singer',
60: 'singer compos', 61: 'singer compos musician', 62: 'singer filmactor', 63: 'singer musician', 64: 'universityteach',
65: 'universityteach compos', 66: 'universityteach historian', 67: 'universityteach lawyer', 68: 'universityteach physician',
69: 'writer', 70: 'writer actor', 71: 'writer actor filmactor', 72: 'writer businessperson', 73: 'writer compos',
74: 'writer filmdirector screenwrit', 75: 'writer historian', 76: 'writer journalist', 77: 'writer journalist historian',
78: 'writer journalist poet', 79: 'writer journalist screenwrit', 80: 'writer lawyer', 81: 'writer painter', 82: 'writer painter poet',
83: 'writer physician', 84: 'writer poet', 85: 'writer screenwrit', 86: 'writer universityteach', 87: 'writer universityteach historian',
88: 'writer universityteach poet'

}
```

```
predictions = predictions['Predict'].map(output)
```

```
predictions.value_counts()
```

```
     actor                  158916
     footballplay            16507
     politician              10670
     compos                   6346
     businessperson           3917
     painter                  2027
     journalist                946
     lawyer                    886
     other                     499
     militarypersonnel         433
     historian                 145
     actor singer               66
     musician                   15
     singer                     13
     poet                        9
     actor compos                6
     physician                   3
     actor filmactor             1
     actor singer compos         1
     Name: Predict, dtype: int64
```

```
predictions.shape
```

```
     (201406,)
```

```
test_data = pd.read_json("/content/wiki-test.json/new_wiki-test.json", lines=True)
```

```
test_data.shape
```

```
     (201406, 2)
```

```
result = pd.concat([test_data, predictions], axis=1)
```

```
result.columns
```

```
     Index(['title', 'summary', 'Predict'], dtype='object')
```

```
result["Predict"].value_counts()
```

```
     actor                  158916
     footballplay            16507
     politician              10670
     compos                   6346
     businessperson           3917
     painter                  2027
     journalist                946
     lawyer                    886
     other                     499
     militarypersonnel         433
     historian                 145
```

```
actor singer               66
musician                   15
singer                     13
poet                        9
actor compos                6
physician                   3
actor filmactor             1
actor singer compos         1
Name: Predict, dtype: int64
```

```
result["Predict"].unique()
```

```
array(['actor', 'footballplay', 'politician', 'lawyer', 'painter',
       'compos', 'militarypersonnel', 'businessperson', 'journalist',
       'historian', 'other', 'actor singer', 'singer', 'musician',
       'actor compos', 'poet', 'actor filmactor', 'physician',
       'actor singer compos'], dtype=object)
```