In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk

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

In [2]:
```
title = pd.read_csv('title.csv')
print(title.shape)
title.head()
```

```
(10000, 3)
```

Out[2]:

	original_grade	emp_title	default
0	C	global config engineer	0
1	C	warehouse office clerk	0
2	D	assembly	0
3	A	customer service	0
4	C	security supervisor	0

In [3]:
```
title.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   original_grade  10000 non-null  object
 1   emp_title       9167 non-null   object
 2   default         10000 non-null  int64 
dtypes: int64(1), object(2)
memory usage: 234.5+ KB
```

In [4]:
```
title.rename(columns= {'original_grade':'grade'}, inplace=True)
```

In [5]:
```
round(100 * title.emp_title.value_counts(normalize= True, dropna= False),2).head()
```

Out[5]:
```
NaN        8.33
manager    2.18
owner      2.04
teacher    2.01
driver     1.23
Name: emp_title, dtype: float64
```

In [6]:
```
title.dropna(inplace= True)
round(100 * title.emp_title.value_counts(normalize= True, dropna= False),2).head()
```

Out[6]:
```
manager    2.38
owner      2.23
teacher    2.19
driver     1.34
sales      1.06
Name: emp_title, dtype: float64
```

In [7]:
```
title.emp_title.nunique()
```

Out[7]:
```
4741
```

Removing accented characters

to make sure that the characters are converted and standardized into ASCII characters.

In [8]:
```
import unicodedata

def remove_accented_chars(text):
    text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
    return text
```

In [9]:
```
title['emp_title'] = title['emp_title'].apply(lambda x: str(x))
```

In [10]:
```
title.emp_title = title.emp_title.apply(lambda x: remove_accented_chars(x))
```

In [11]:
```
title.emp_title.shape
```

Out[11]:
```
(9167,)
```

In [12]:
```
title.emp_title.nunique()
```

Out[12]:
```
4741
```

Removing Special Characters

In [13]:
```
import re

def remove_special_characters(text, remove_digits=False):
    pattern = r'[^a-zA-z0-9\s]' if not remove_digits else r'[^a-zA-z\s]'
    text = re.sub(pattern, '', text)
    return text
```

In [14]:
```
title.emp_title = title.emp_title.apply(lambda x: remove_special_characters(x, remove_digits= True))
```

In [15]:
```
title.emp_title.shape
```

Out[15]:
```
(9167,)
```

In [16]:
```
title.emp_title.nunique()
```

Out[16]:
```
4723
```

Remove stopwords

In [17]:
```
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
```

In [18]:
```
type(stop_words)
```

Out[18]:
```
list
```

In [19]:
```
from nltk.tokenize import word_tokenize

def remove_stopwords(text, is_lower_case=False):
    tokens = word_tokenize(text)
    #tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in stop_words]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in stop_words]
    filtered_text = ' '.join(filtered_tokens)

    return filtered_text
```

In [20]:
```
title.emp_title = title.emp_title.apply(lambda x: remove_stopwords(x))
```

In [21]:
```
title.emp_title.shape
```

Out[21]:
```
(9167,)
```

In [22]:
```
title.emp_title.nunique()
```

Out[22]:
```
4322
```

spacy-Noun Chunks

In [23]:
```
import spacy

nlp = spacy.load("en_core_web_sm")
title['doc'] = title.emp_title.apply(lambda x: nlp(x))
```

In [24]:
```
title.head()
```

Out[24]:

	grade	emp_title	default	doc
0	C	global config engineer	0	(global, config, engineer)
1	C	warehouse office clerk	0	(warehouse, office, clerk)
2	D	assembly	0	(assembly)
3	A	customer service	0	(customer, service)
4	C	security supervisor	0	(security, supervisor)

In [25]:
```
type(title.doc[0])
```

Out[25]:
```
spacy.tokens.doc.Doc
```

In [26]:
```
def get_noun_root(doc):
    root_lst=[]
    try:
        for chunk in doc.noun_chunks:
            root_lst.append(chunk.root.text)
    except:
        pass
    return root_lst
```

In [27]:
```
title['noun_root'] = title.doc.apply(lambda x: get_noun_root(x))
```

In [28]:
```
title.head()
```

Out[28]:

	grade	emp_title	default	doc	noun_root
0	C	global config engineer	0	(global, config, engineer)	[engineer]
1	C	warehouse office clerk	0	(warehouse, office, clerk)	[clerk]
2	D	assembly	0	(assembly)	[assembly]
3	A	customer service	0	(customer, service)	[service]
4	C	security supervisor	0	(security, supervisor)	[supervisor]

In [29]:
```
title.emp_title.nunique()
```

Out[29]:
```
4322
```

number of noun root for each emp_title

In [30]:
```
title['num_roots']= [len(root) for root in title['noun_root']]
```

In [31]:
```
title.head()
```

Out[31]:

	grade	emp_title	default	doc	noun_root	num_roots
0	C	global config engineer	0	(global, config, engineer)	[engineer]	1
1	C	warehouse office clerk	0	(warehouse, office, clerk)	[clerk]	1
2	D	assembly	0	(assembly)	[assembly]	1
3	A	customer service	0	(customer, service)	[service]	1
4	C	security supervisor	0	(security, supervisor)	[supervisor]	1

In [32]:
```
title.num_roots.value_counts()
```

Out[32]:
```
1    8546
0     581
2      40
Name: num_roots, dtype: int64
```

In [33]:
```
title = title[title.num_roots != 0]
```

In [34]:
```
len(title)
```

Out[34]:
```
8586
```

In [35]:
```
title[title.num_roots == 2].head()
```

Out[35]:

	grade	emp_title	default	doc	noun_root	num_roots
50	E	hes advisor	0	(he, s, advisor)	[he, advisor]	2
760	B	case managet loan	0	(case, managet, loan)	[case, loan]	2
837	B	sales chat banker	0	(sales, chat, banker)	[sales, banker]	2
923	C	right way agent	0	(right, way, agent)	[way, agent]	2
1115	B	teaching assistant iii	0	(teaching, assistant, iii)	[assistant, iii]	2

In [36]:
```
title[title.num_roots == 2]['noun_root'].apply(pd.Series)
```

Out[36]:

	0	1
50	he	advisor
760	case	loan
837	sales	banker
923	way	agent
1115	assistant	iii
1144	analyst	iii
1280	underwriters	assistant
1458	specialist	iii
1577	server	manager
1585	writer	iii
1611	programs	ii
1766	manager	assistant
2140	president	affairs
2168	officer	iii
2507	veterans	technician
2768	aide	technician
2814	it	d
2965	planning	analysis
3380	provider	iii
3502	directoroperator	_
3644	counselo	iii
3714	director	av
4007	eeo	officer
4209	executive	counsel
4496	driver	dc
5123	head	teacher
5763	team	production
5935	representative	iii
7017	manager	nicu
7153	paraeducator	iii
7296	director	educators
7512	teachers	assistant
7626	part	sorter
7781	administrator	iii
7970	analyst	iii
8050	servicesales	associate
8150	analyst	iii
8290	president	founder
8540	vice	development
9765	analyst	iv

In [37]:
```
root = round(100 * title.loc[title.num_roots == 1, 'noun_root'].value_counts(normalize= True, dropna= F
alse),2).head(11)
root
```

Out[37]:
```
[manager]       14.84
[driver]         3.41
[engineer]       3.02
[teacher]        2.93
[owner]          2.88
[supervisor]     2.81
[specialist]     2.66
[technician]     2.38
[analyst]        2.33
[director]       2.22
[officer]        2.22
Name: noun_root, dtype: float64
```

---

split into train and test

In [38]:
```
df = pd.concat([title.noun_root.apply(pd.Series)[0] , title['grade']], axis= 1)
df.columns = ['noun_root', 'grade']
df.head()
```

Out[38]:

	noun_root	grade
0	engineer	C
1	clerk	C
2	assembly	D
3	service	A
4	supervisor	C

In [39]:
```
y = df['grade'].replace({'A':0, 'B': 1, 'C':2, 'D': 3, 'E':4, 'F':5, 'G':6})
print(y.shape)
y.head()
```

```
(8586,)
```

Out[39]:
```
0    2
1    2
2    3
3    0
4    2
Name: grade, dtype: int64
```

In [40]:
```
X = df['noun_root']
print(X.shape)
X.head()
```

```
(8586,)
```

Out[40]:
```
0      engineer
1         clerk
2      assembly
3       service
4    supervisor
Name: noun_root, dtype: object
```

In [41]:
```
from sklearn.model_selection import train_test_split
X_Train, X_Test, y_Train, y_Test = train_test_split(X, y, test_size=0.30, random_state= 42)
```

In [42]:
```
y_Train.value_counts()
```

Out[42]:
```
1    1827
2    1583
0    1506
3     852
4     192
5      42
6       8
Name: grade, dtype: int64
```

In [43]:
```
y_Test.value_counts()
```

Out[43]:
```
1    783
2    682
0    644
3    361
4     94
5     10
6      2
Name: grade, dtype: int64
```

In [44]:
```
X_Train.shape, y_Train.shape
```

Out[44]:
```
((6010,), (6010,))
```

TFIDF, bag-of-words

In [45]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(use_idf=True) #tfidf = TfidfVectorizer()


tfidf.fit(X_Train)

X_Train_tfidf = tfidf.transform(X_Train)
X_Test_tfidf = tfidf.transform(X_Test)
```

tfidf.get_feature_names()

In [46]:
```
df_idf = pd.DataFrame(tfidf.idf_, index= tfidf.get_feature_names(), columns=["idf_weights"])

# sort ascending
df_idf.sort_values(by=['idf_weights'])
```

Out[46]:

	idf_weights
manager	2.928266
driver	4.408042
teacher	4.470238
supervisor	4.519563
engineer	4.565548
...	...
hygienst	9.008199
icare	9.008199
information	9.008199
ownergroomer	9.008199
youthdevelopment	9.008199

777 rows × 1 columns

The lower the IDF value of a word, the less unique.

Model Evaluation

In [58]:
```
from sklearn.metrics import confusion_matrix

def conf_mat(y_true, y_pred, set_name):
    print('Confusion Matrix on ', set_name)
    cnf_table = pd.DataFrame(confusion_matrix(y_true, y_pred), index = ['A','B', 'C','D','E','F','G'],
                columns=['predicted_A', 'predicted_B', 'predicted_C', 'predicted_D', 'predicted_E','pre
dicted_F'\
                          ,'predicted_G'])
    return cnf_table
```

MultinomialNB

In [61]:
```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB

from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline

X_Train, X_Test, y_Train, y_Test = train_test_split(X, y, test_size=0.30, random_state= 42)
title_clf = Pipeline([('vect', CountVectorizer()),
                      ('tfidf', TfidfTransformer()),
                      ('smote', SMOTE(random_state= 42)),
                      ('clf', MultinomialNB())])
```

In [70]:
```
title_clf.fit(X_Train, y_Train)
y_pred_test = title_clf.predict(X_Test)
y_pred_train = title_clf.predict(X_Train)
```

In [71]:
```
from sklearn.metrics import accuracy_score

print('Accuracy on train set: ', round(100 * accuracy_score(y_Train, y_pred_train),2))
conf_mat(y_Train, y_pred_train, 'Train')
```

```
Accuracy on train set:  26.59
Confusion Matrix on  Train
```

Out[71]:

	predicted_A	predicted_B	predicted_C	predicted_D	predicted_E	predicted_F	predicted_G
A	445	95	151	132	360	171	152
B	212	399	154	181	500	224	157
C	195	105	396	138	394	209	146
D	106	68	70	212	222	105	69
E	14	3	13	7	115	24	16
F	4	0	1	2	7	23	5
G	0	0	0	0	0	0	8

In [72]:
```
print('Accuracy on test set: ', round(100 * accuracy_score(y_Test, y_pred_test),2))
conf_mat(y_Test, y_pred_test, 'Test')
```

```
Accuracy on test set:  14.48
Confusion Matrix on  Test
```

Out[72]:

	predicted_A	predicted_B	predicted_C	predicted_D	predicted_E	predicted_F	predicted_G
A	144	58	77	60	166	76	63
B	173	73	96	84	192	102	63
C	140	71	90	60	188	73	60
D	74	33	34	42	94	50	34
E	19	11	13	12	24	10	5
F	3	1	1	1	3	0	1
G	1	0	0	0	0	1	0

SVC

In [75]:
```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()

# this steps generates word counts for the words in your docs
X_train_counts = count_vect.fit_transform(X_Train)
X_test_counts = count_vect.transform(X_Test)



from sklearn.feature_extraction.text import TfidfTransformer

tf_transformer = TfidfTransformer(use_idf=False).fit(X_train_counts)
X_Train_tf = tf_transformer.transform(X_train_counts)
X_Test_tf = tf_transformer.transform(X_test_counts)


from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state= 42)
X_Train_tf_smote , y_Train_smote = smote.fit_resample(X_Train_tf, y_Train)
```

In [76]:
```
from sklearn.svm import SVC
from skopt import BayesSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import RandomizedSearchCV

params = dict()
params['C'] = (1e-6, 100.0, 'log-uniform')
params['gamma'] = (1e-6, 100.0, 'log-uniform')
params['degree'] = (1,5)
params['kernel'] = ['linear', 'poly', 'rbf', 'sigmoid']

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define the search
search = RandomizedSearchCV(SVC(), params, n_jobs=-1, cv=cv)
search.fit(X_Train_tf_smote, y_Train_smote)
```

Out[76]:
```
RandomizedSearchCV(cv=RepeatedStratifiedKFold(n_repeats=3, n_splits=10, random_state=1),
                   estimator=SVC(), n_jobs=-1,
                   param_distributions={'C': (1e-06, 100.0, 'log-uniform'),
                                        'degree': (1, 5),
                                        'gamma': (1e-06, 100.0, 'log-uniform'),
                                        'kernel': ['linear', 'poly', 'rbf',
                                                   'sigmoid']})
```

In [77]:
```
search.best_params_
```

Out[77]:
```
{'kernel': 'rbf', 'gamma': 100.0, 'degree': 5, 'C': 100.0}
```

In [84]:
```
y_pred_train = search.predict(X_Train_tf_smote)
y_pred_test = search.predict(X_Test_tf)

print('Accuracy on train set: ', round(100 * accuracy_score(y_Train_smote, y_pred_train),2))
cm = conf_mat(y_Train_smote, y_pred_train, 'train set')
cm
```

```
Accuracy on train set:  57.57
Confusion Matrix on  train set
```

Out[84]:

	predicted_A	predicted_B	predicted_C	predicted_D	predicted_E	predicted_F	predicted_G
A	810	246	136	468	143	22	2
B	384	609	95	525	192	18	4
C	420	275	436	491	190	13	2
D	314	186	106	1042	167	7	5
E	110	83	45	372	1204	13	0
F	29	57	3	168	125	1445	0
G	5	3	0	0	3	0	1816

In [83]:
```
print('Accuracy on test set: ', round(100 * accuracy_score(y_Test, y_pred_test),2))
cm = conf_mat(y_Test, y_pred_test, 'test set')
cm
```

```
Accuracy on test set:  22.48
Confusion Matrix on  test set
```

Out[83]:

	predicted_A	predicted_B	predicted_C	predicted_D	predicted_E	predicted_F	predicted_G
A	183	154	56	186	58	4	3
B	196	199	74	243	65	5	1
C	172	166	72	197	68	6	1
D	80	91	29	123	31	5	2
E	18	31	11	32	2	0	0
F	2	2	1	4	1	0	0
G	1	0	0	0	1	0	0