In [3]:
```
import pandas as pd
import re
import string
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import numpy as np
```

In [4]:
```
data= pd.read_csv('/Users/spring/Desktop/BA Data/BA_data.csv')
```

In [3]:
```
data.columns
```

Out[3]:
```
Index(['Unnamed: 0', 'Job Title', 'Salary Estimate', 'Job Descript
ion',
       'Rating', 'Company Name', 'Location', 'Headquarters', 'Size
', 'Founded',
       'Type of ownership', 'Industry', 'Sector', 'Revenue', 'Comp
etitors',
       'Easy Apply', 'job_description_cleaned', 'job_description_f
ormatted',
       'index'],
      dtype='object')
```

In [4]:
```
data.shape
```

Out[4]:
```
(6162, 19)
```

In [5]:
```
df=data.copy()
```

In [6]:
```
df.drop(columns = 'Unnamed: 0', inplace= True)
df.drop(columns = 'job_description_cleaned' , inplace= True)
df.drop(columns = 'job_description_formatted' , inplace= True)
```

In [7]:
```
df.shape
```

Out[7]:
```
(6162, 16)
```

In [ ]:
```

```

In [ ]:
```
# Assign Column 'Industry' to its occupation code (2022 NAICS)
```

In [9]:
```
df.Industry.unique()
```

```
       'Beauty & Personal Accessories Stores',
       'Automotive Parts & Accessories Stores', 'Chemical Manufact
uring',
       'Department, Clothing, & Shoe Stores', 'K-12 Education',
       'Transportation Management', 'Miscellaneous Manufacturing',
       'Logistics & Supply Chain', 'Catering & Food Service Contra
ctors',
       'Health Care Products Manufacturing', 'Drug & Health Stores
',
       'Wholesale', 'Membership Organizations', 'Oil & Gas Service
s',
       'Sporting Goods Stores', 'Gas Stations', 'Truck Rental & Le
asing',
       'Express Delivery Services',
       'Cable, Internet & Telephone Providers', 'Metals Brokers',
       'Grocery Stores & Supermarkets', 'Pet & Pet Supplies Stores
',
       'Education Training Services', 'Financial Analytics & Resea
rch',
```

In [6]:
```
# According to 2022 NAICS (North American Industry Classification S


df['Industry']=[x.lower() for x in data['Industry']]

# 62-- Health care and Social assistance
df.loc[df['Industry'].str.contains("health care"), 'Industry'] = '6
df.loc[df['Industry'].str.contains("social assistance"), 'Industry'
df.loc[df['Industry'].str.contains("preschool & child care"), 'Indu


# 51-- Information: Web search portal and Internet publishing
df.loc[df['Industry'].str.contains("internet"), 'Industry'] = '51'
df.loc[df['Industry'].str.contains("it services"), 'Industry'] = '5
df.loc[df['Industry'].str.contains("motion picture production & dis
df.loc[df['Industry'].str.contains("tv broadcast & cable networks")
df.loc[df['Industry'].str.contains("biotech & pharmaceuticals"), 'I
df.loc[df['Industry'].str.contains("publish"), 'Industry'] = '51'
df.loc[df['Industry'].str.contains("telecommunications services"),
df.loc[df['Industry'].str.contains("news outlet"), 'Industry'] = '5


# 61-- Educational Services: Sports and Recreation
df.loc[df['Industry'].str.contains("sports & recreation"), 'Industr
df.loc[df['Industry'].str.contains("colleges"), 'Industry'] = '61'
df.loc[df['Industry'].str.contains("universities"), 'Industry'] = '
df.loc[df['Industry'].str.contains("education training services"),
df.loc[df['Industry'].str.contains("k-12 education"), 'Industry'] =

# 52-- Finance and Insurance: Banking
df.loc[df['Industry'].str.contains("insurance"), 'Industry'] = '52'
df.loc[df['Industry'].str.contains("bank"), 'Industry'] = '52'
df.loc[df['Industry'].str.contains("venture capital"), 'Industry']
df.loc[df['Industry'].str.contains("brokerage"), 'Industry'] = '52'
```

```
df.loc[df['Industry'].str.contains("lend"), 'Industry'] = '52
df.loc[df['Industry'].str.contains("financial transaction processin
df.loc[df['Industry'].str.contains("stock exchanges"), 'Industry']


# 54-- Professional, Scientific, and Technical Services
df.loc[df['Industry'].str.contains("research"), 'Industry'] = '54'
df.loc[df['Industry'].str.contains("advertising"), 'Industry'] = '5
df.loc[df['Industry'].str.contains("marketing"), 'Industry'] = '54'
df.loc[df['Industry'].str.contains("consult"), 'Industry'] = '54'
df.loc[df['Industry'].str.contains("engineering"), 'Industry'] = '5
df.loc[df['Industry'].str.contains("architectural"), 'Industry'] =
df.loc[df['Industry'].str.contains("enterprise software"), 'Industr
df.loc[df['Industry'].str.contains("accounting"), 'Industry'] = '54
df.loc[df['Industry'].str.contains("video games"), 'Industry'] = '5
df.loc[df['Industry'].str.contains("logistics & supply chain"), 'In


# 23--Construction
df.loc[df['Industry'].str.contains("building"), 'Industry'] = '23'
df.loc[df['Industry'].str.contains("construction"), 'Industry'] = '


# 72-- Accommodation and Food Services
df.loc[df['Industry'].str.contains("restaurant"), 'Industry'] = '72
df.loc[df['Industry'].str.contains("food"), 'Industry'] = '72'
df.loc[df['Industry'].str.contains("hotels, motels, & resorts"), 'I



# 42-- Wholesale Trade: Computer and Computer Peripheral Equipment
df.loc[df['Industry'].str.contains("computer hardware & software"),
df.loc[df['Industry'].str.contains("wholesale"), 'Industry'] = '42'
df.loc[df['Industry'].str.contains("metals brokers"), 'Industry'] =


# 56-- Administrative and Support and Waste Management and Remediat

df.loc[df['Industry'].str.contains("staffing"), 'Industry'] = '56'
df.loc[df['Industry'].str.contains("outsourcing"), 'Industry'] = '5
df.loc[df['Industry'].str.contains("security services"), 'Industry'
df.loc[df['Industry'].str.contains("travel agencies"), 'Industry']

# 53- Real Estate and Rental and Leasing
df.loc[df['Industry'].str.contains("real estate"), 'Industry'] = '5
df.loc[df['Industry'].str.contains("truck rental & leasing"), 'Indu
df.loc[df['Industry'].str.contains("rental"), 'Industry'] = '53'
df.loc[df['Industry'].str.contains("consumer electronics & applianc
df.loc[df['Industry'].str.contains("self-storage services"), 'Indus


# 92-- Public Administration: Regulation and Administration of Comm
df.loc[df['Industry'].str.contains("federal agencies"), 'Industry']
df.loc[df['Industry'].str.contains("utilities"), 'Industry'] = '92'



# 71-- Arts, Entertainment, and Recreation
```

```
df.loc[df['Industry'].str.contains("gambling"), 'Industry'] = '71'
df.loc[df['Industry'].str.contains("recreation"), 'Industry'] = '71
df.loc[df['Industry'].str.contains("museums, zoos & amusement parks


# 81-- Other Services (except Public Administration)

df.loc[df['Industry'].str.contains("repair & maintenance"), 'Indust
df.loc[df['Industry'].str.contains("health, beauty, & fitness"), 'I
df.loc[df['Industry'].str.contains("health fundraising organization
df.loc[df['Industry'].str.contains("grantmaking foundations"), 'Ind
df.loc[df['Industry'].str.contains("membership organizations"), 'In
df.loc[df['Industry'].str.contains("religious organizations"), 'Ind


# 44-- car dealer and Cosmetics, Beauty Supplies, cloth store, phar
df.loc[df['Industry'].str.contains("vehicle dealers"), 'Industry']
df.loc[df['Industry'].str.contains("beauty & personal accessories s
df.loc[df['Industry'].str.contains("automotive parts & accessories
df.loc[df['Industry'].str.contains("department, clothing, & shoe st
df.loc[df['Industry'].str.contains("gas stations"), 'Industry'] = '
df.loc[df['Industry'].str.contains("drug & health stores"), 'Indust
df.loc[df['Industry'].str.contains("grocery stores"), 'Industry'] =
df.loc[df['Industry'].str.contains("supermarkets"), 'Industry'] = '
df.loc[df['Industry'].str.contains("convenience stores & truck stop
df.loc[df['Industry'].str.contains("home furniture & housewares sto
df.loc[df['Industry'].str.contains("home centers & hardware stores"


# 45-- Retail trade: sport goods stores and pet

df.loc[df['Industry'].str.contains("retail stores"), 'Industry'] =
df.loc[df['Industry'].str.contains("sporting goods stores"), 'Indus
df.loc[df['Industry'].str.contains("pet & pet supplies stores"), 'I
df.loc[df['Industry'].str.contains("general merchandise & superstor


# 33--manufacturing
df.loc[df['Industry'].str.contains("manufacturing"), 'Industry'] =
df.loc[df['Industry'].str.contains("aerospace & defense"), 'Industr
df.loc[df['Industry'].str.contains("audiovisual"), 'Industry'] = '3
df.loc[df['Industry'].str.contains("radio"), 'Industry'] = '33'

# 92-- Federal, State, and Local Government,
df.loc[df['Industry'].str.contains("government"), 'Industry'] = '92
df.loc[df['Industry'].str.contains("regional agencies"), 'Industry'



# 21-- Mining, Quarrying, and Oil and Gas Extraction
df.loc[df['Industry'].str.contains("oil & gas services"), 'Industry
df.loc[df['Industry'].str.contains("oil & gas exploration & product


# 48-- Transportation
df.loc[df['Industry'].str.contains("transportation management"), 'I
```

```python
df.loc[df['Industry'].str.contains("express delivery services"), 'I
df.loc[df['Industry'].str.contains("shipping"), 'Industry'] = '48'
df.loc[df['Industry'].str.contains("cruise ships"), 'Industry'] = '
df.loc[df['Industry'].str.contains("trucking"), 'Industry'] = '48'


# 11--Agriculture, Forestry, Fishing and Hunting
df.loc[df['Industry'].str.contains("farm support services"), 'Indus

# industry-- energy and legal not have efficient info and is meanig
# so this industry is not considered and assign it as -1.

df['Industry'].replace(to_replace="energy", value='-1', inplace=Tru
df['Industry'].replace(to_replace="legal", value='-1', inplace=True
```

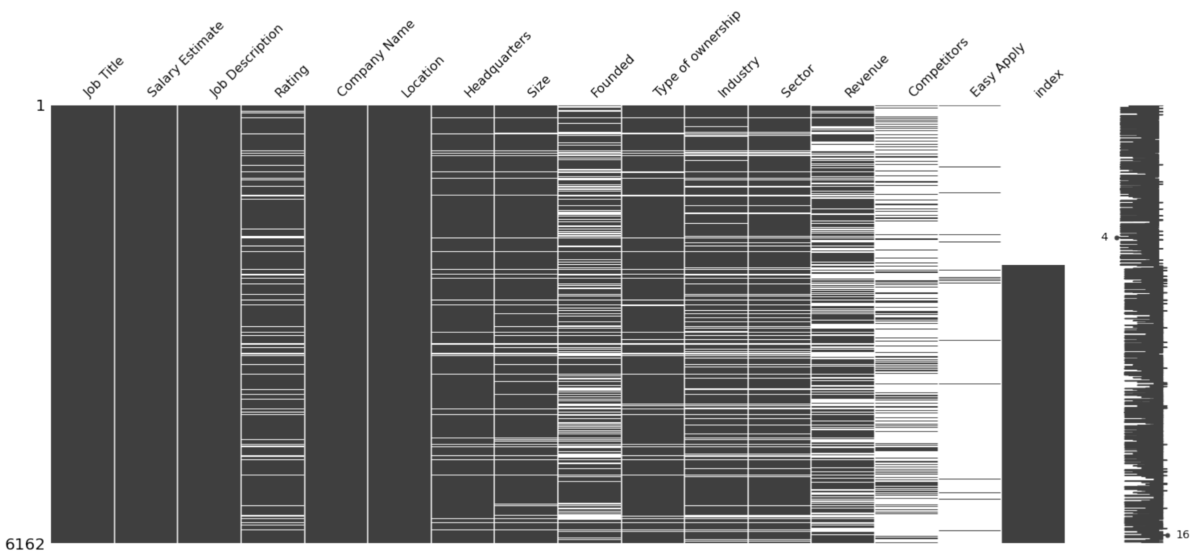
In [11]:
```python
df.Industry.unique()
```

Out[11]:
```
array(['62', '51', '61', '52', '54', '-1', '23', '72', '42', '56',
'53',
       '92', '81', '71', '45', '33', '44', '48', '21'], dtype=obje
ct)
```

In [7]:
```python
# -1 is replaced by nan value
df.replace([-1, '-1', 'Unknown', 'Unknown / Non-Applicable'], np.na
```

In [13]:
```python
# check missing value distribution
import missingno as msno
msno.matrix(df)
```

Out[13]: <AxesSubplot:>



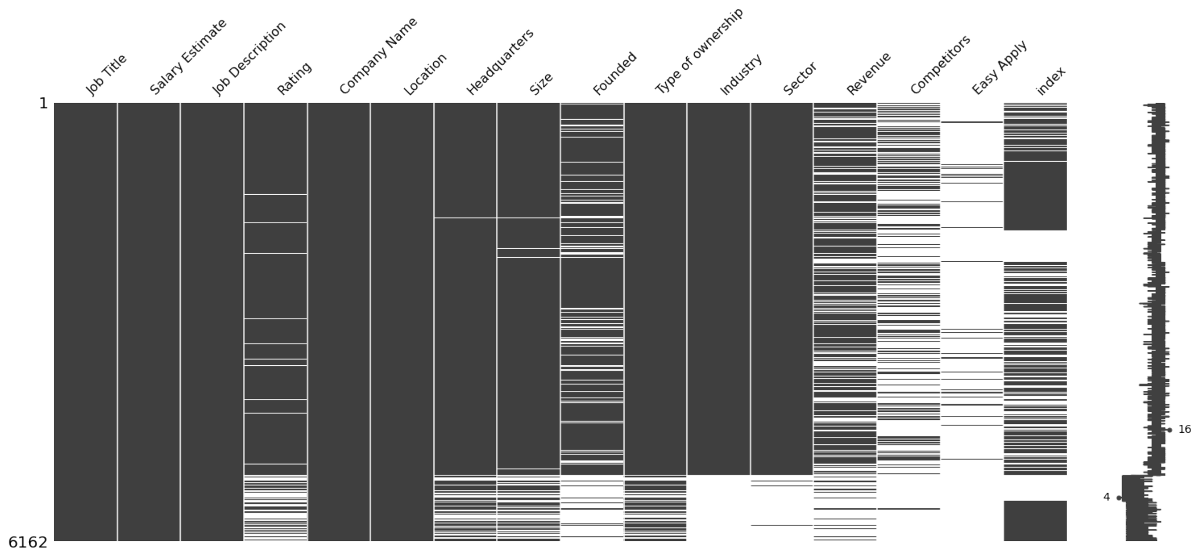
In [ ]:

```
'''
Data is (MCAR) Missing completely at Random.
Missing value exist in these columns:
-- Rating, Headquarters, size,founded, type of ownership, industry,


Among them, only industry is considered.

'''
```

In [14]:

```
msno.matrix(df.sort_values('Industry'))
```

Out[14]: <AxesSubplot:>



In [ ]:

```

'''
This implies MCAR. Deleting instances is likely a good idea.

'''
```

In [15]:

```
# only not null value in column Industry are kept.

df = df[df['Industry'].notna()]
```

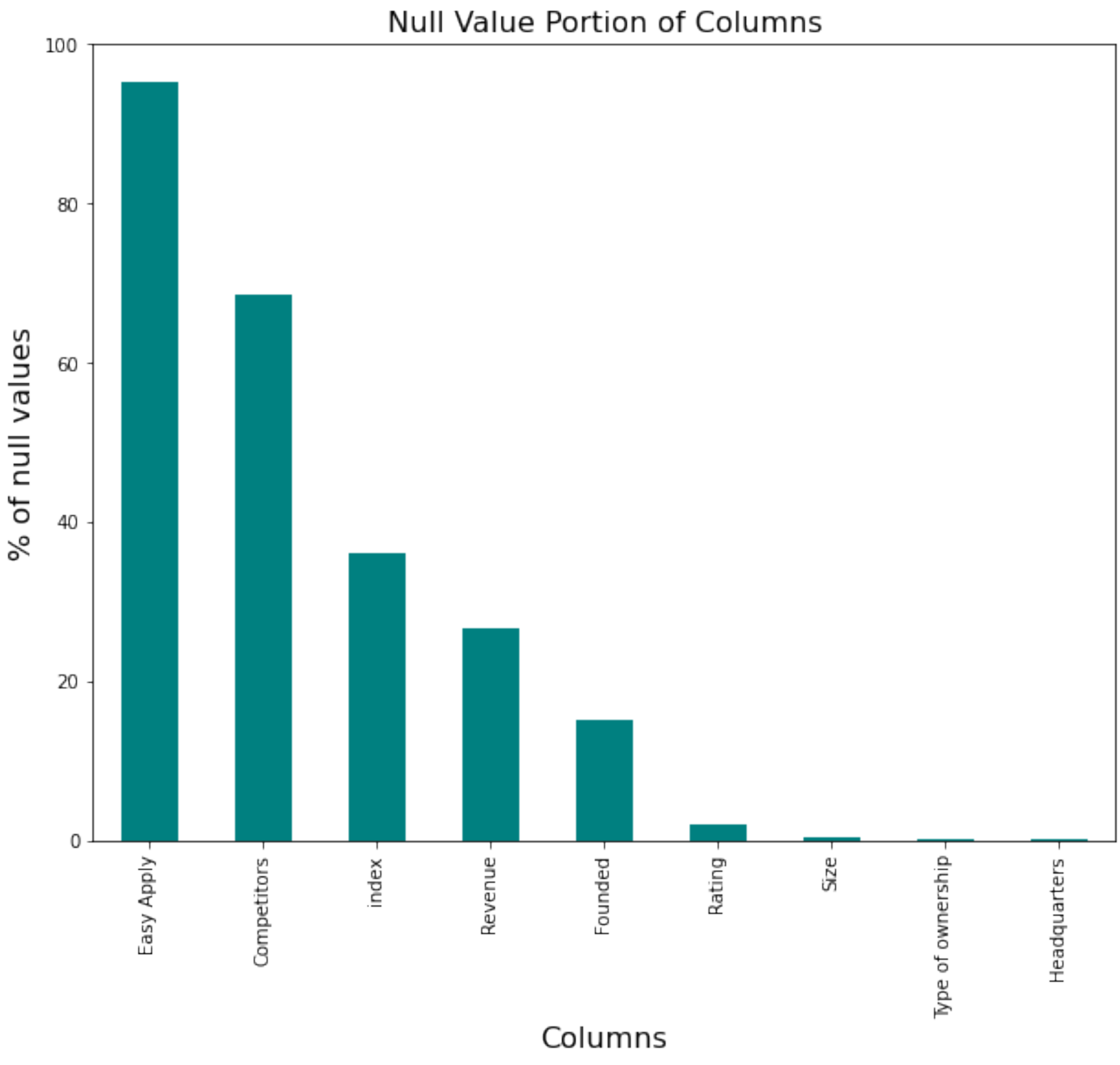
In [16]:

```
df.shape
```

Out[16]: (5234, 16)

In [17]:
```
# calculate % of null value in each column and sort them in descend

null_portion = df.isnull().sum().sort_values(ascending=False)/len(d

# make a plot
null_portion[ null_portion>0.1].plot(kind='bar', figsize=(10,8), co
plt.xlabel("Columns",fontsize=16)
plt.ylabel("% of null values",fontsize=16);
plt.title("Null Value Portion of Columns", fontsize=16)
```

Out[17]: Text(0.5, 1.0, 'Null Value Portion of Columns')



In [40]:
```
# calculate % of null value in each row and sort them in descending
row_null_portion = df.isnull().sum(axis=1).sort_values(ascending=Fa
row_null_portion
```

Out[40]:
```
893     0.133741
391     0.133741
402     0.133741
4192    0.114635
4006    0.114635
          ...
3337    0.000000
5146    0.000000
5419    0.000000
2454    0.000000
3133    0.000000
Length: 5234, dtype: float64
```

In [ ]:
```
'''
each row is kept due to their low row_null portion.

'''
```

In [18]:
```
# Check the frequency counts of the values of output feature (Indus
df.Industry.value_counts()
```

Out[18]:
```
51    1522
54     943
56     643
52     580
62     409
42     394
61     158
33     152
92     123
44      72
81      55
72      42
53      35
45      30
23      28
21      24
48      18
71       6
Name: Industry, dtype: int64
```

In [8]:
```
df.rename(columns= {df.columns[10]: 'Industry_code'}, inplace = Tru
```

In [9]:
```
# change column name from job title to position

df.rename(columns={ df.columns[0]: 'position'}, inplace = True)
```

In [10]:
```
df.head(2)
```

Out[10]:

	position	Salary Estimate	Job Description	Rating	Company Name	Location	Headquarters
0	Data Analyst, Center on Immigration and Justic...	$37K-66K (Glassdoor est.)	Are you eager to roll up your sleeves and harn...	3.2	Vera Institute of Justice\n3.2	New York, NY	New York, NY
1	Quality Data Analyst	$37K-66K (Glassdoor est.)	Overview\n\nProvides analytical and technical ...	3.8	Visiting Nurse Service of New York\n3.8	New York, NY	New York, NY

In [22]:
```
df['position']
```

Out[22]:
```
0       Data Analyst, Center on Immigration and Justic...
1                                    Quality Data Analyst
2       Senior Data Analyst, Insights & Analytics Team...
3                                            Data Analyst
4                                  Reporting Data Analyst
                              ...
6157                                    AWS Data Engineer
6158                                  Data Analyst â Junior
6159                       Security Analytics Data Engineer
6160                       Security Analytics Data Engineer
6161    Patient Safety Physician or Safety Scientist -...
Name: position, Length: 5234, dtype: object
```

In [11]:
```
# Assign relevant job positions to certain broad job title
# using keywords to assign detailed job titles to certain broad job

df['position']=[x.upper() for x in df['position']]
df['description']=[x.upper() for x in df['Job Description']]

df.loc[df['position'].str.contains("SCIENTIST"), 'position'] = 'Dat

df.loc[df['position'].str.contains('ENGINEER'),'position']='Machine
df.loc[df['position'].str.contains('PRINCIPAL STATISTICAL PROGRAMME
df.loc[df['position'].str.contains('PROGRAMMER'),'position']='Machi
df.loc[df['position'].str.contains('MODELER'),'position']='Machine
df.loc[df['position'].str.contains('MACHINE LEARNING'),'position']=

df.loc[df['position'].str.contains('ANALYST'), 'position'] = 'Data
df.loc[df['position'].str.contains('STATISTICIAN'), 'position'] = '
df.loc[df['position'].str.contains('COMPUTATIONAL BIOLOGIST'), 'pos


df.loc[df['position'].str.contains('MANAGER'),'position']='Data Sci
df.loc[df['position'].str.contains('CONSULTANT'),'position']='Data
df.loc[df['position'].str.contains('DATA SCIENCE'),'position']='Dat
df.loc[df['position'].str.contains('DIRECTOR'),'position']='Data Sc
df.loc[df['position'].str.contains('MANAGEMENT'),'position']='Data
df.loc[df['position'].str.contains('CONSULTING'),'position']='Data


df.loc[df['position'].str.contains('ARCHITECT'),'position']='Data a

df.loc[df['position'].str.contains('RESEARCH'),'position']='Reseach
```

In [12]:
```
df.position=df[(df.position == 'Data Scientist') | (df.position ==

df.position=['Others' if x is np.nan else x for x in df.position]
```

In [25]:
```
df.position.value_counts()
```

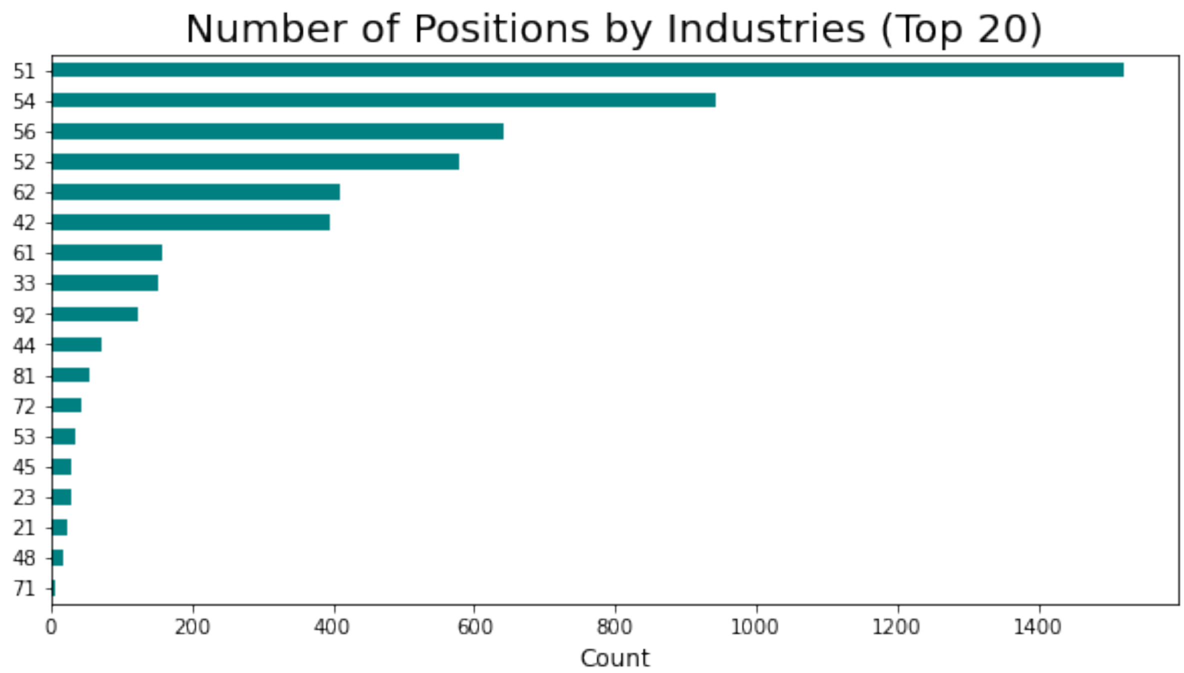
Out[25]:
```
Data Analyst                 2536
Data Scientist               1556
Machine Learning Engineer     842
Data Science Manager          230
Reseacher                      34
Data architect                 20
Others                         16
Name: position, dtype: int64
```

In [26]:
```
# Amount of vacany position for each job position
position=df.groupby(['position'])['Industry_code'].count()
position=position.reset_index(name='Industry_code')
position=position.sort_values(['Industry_code'],ascending=False)

print('Amount of new created roles :', '\n\n', position)
```

```
Amount of new created roles :

                    position  Industry_code
0              Data Analyst           2536
2            Data Scientist           1556
4  Machine Learning Engineer            842
1      Data Science Manager            230
6                 Reseacher             34
3            Data architect             20
5                    Others             16
```

5.1.1 Vacant Positions by Industry

In [27]:
```
# Amount of vacancy Positions by industry

Industry = df.groupby(['Industry_code']).count().sort_values('posit
Industry['position'].plot(kind='barh',figsize = (10,5),color='teal'
plt.xlabel('Count', size = 12)
plt.ylabel('')
plt.yticks(size = 10)
plt.xticks(size = 10)
plt.title('Number of Positions by Industries (Top 20)', size = 20)
plt.show()
```



In [ ]:

In [24]:
```
df_model= df_clean.copy()
```

In [26]:
```
df_model.description_clean = df_model.description_clean.astype(str)
```

In [27]:
```
# Focus on classifiying job description to right job position :
# explore the difference of asymmetry information between job descr

from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

X= df_model.description_clean
Y= df_model.position

X=[re.sub(r"[^a-zA-Z0-9]+", ' ', k) for k in X]
X=[re.sub("[0-9]+",' ',k) for k in X]

#applying stemmer

ps = PorterStemmer()
X = [ps.stem(k) for k in X]

#Note: I have not removed stop words because there are important ke
tfidf=TfidfVectorizer()
# one-hot encoding
label_enc= LabelEncoder()

X= tfidf.fit_transform(X)
Y= label_enc.fit_transform(Y)

x_train,x_test,y_train,y_test= train_test_split(X,Y,stratify=Y,test_

# divide train dataset into train and validation subsets.
x_train_1,x_val_1,y_train_1,y_val_1= train_test_split(x_train, y_tr
```

In [ ]:
```
x_test,y_test

x_train_1, y_train_1

x_val_1,, y_val_1
```

4.1.1 Support Vector Machine (SVM) with RBF Kernels

In [139]:

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,confusion_matrix,classif
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_validate

# first algorithm SVM: SVM classification

svm= SVC(kernel='rbf')
svm.fit(x_train,y_train)

svm_y=svm.predict(x_test)

print('Accuracy of SVM :', accuracy_score(y_test,svm_y))
print ('Confusion Matrix of SVM : ', '\n\n', confusion_matrix(y_tes
print(1(y_test,svm_y))
```

```
Accuracy of SVM : 0.8690932311621967
Confusion Matrix of SVM :

 [[735   1  13   0  12   0]
 [ 26  18  25   0   0   0]
 [ 37   0 422   0   8   0]
 [  2   0   0   1   3   0]
 [ 43   0  30   0 180   0]
 [  4   0   1   0   0   5]]
              precision    recall  f1-score   support

           0       0.87      0.97      0.91       761
           1       0.95      0.26      0.41        69
           2       0.86      0.90      0.88       467
           3       1.00      0.17      0.29         6
           4       0.89      0.71      0.79       253
           5       1.00      0.50      0.67        10

    accuracy                           0.87      1566
   macro avg       0.93      0.58      0.66      1566
weighted avg       0.87      0.87      0.86      1566
```

In [ ]:

```
'''
not all recall score is high, which means a part of job description
The reason could be high similarity of two job description or the a




'''
```

In [135]:

```
#crossfold Validation of 7 folds for SVM
cross_val_SVM= cross_validate(svm, x_train, y=y_train,cv=7, return_
```

In [136]:
```
sorted(cross_val_SVM.keys())
```

Out[136]:
```
['fit_time', 'score_time', 'test_score', 'train_score']
```

In [137]:
```
print ('SVM Train fit score is : ', '\n\n', cross_val_SVM ['train_s
print ('SVM TEST score is : ', '\n\n', cross_val_SVM ['test_score']
```

```
SVM Train fit score is :

 [0.96549521 0.96741214 0.96805112 0.97092652 0.96932907 0.966145
 0.96933887]
SVM TEST score is :

 [0.88697318 0.85249042 0.89463602 0.88314176 0.86015326 0.8733205
4
 0.84261036]
```

4.1.3 Multinomial Naive Bayes (MNB)

In [244]:
```
# Naive Bayes classification
NB= MultinomialNB()
NB.fit(x_train,y_train)
NB_y= NB.predict(x_test)

print('Accuracy of Naive Bayes :', accuracy_score(y_test,NB_y))
print ('Confusion Matrix of Naive Bayes : ', '\n\n', confusion_matr
print(classification_report(y_test,NB_y))
```

```
Accuracy of Naive Bayes : 0.5830140485312899
Confusion Matrix of Naive Bayes :

 [[755   0   6   0   0   0]
 [ 68   0   1   0   0   0]
 [312   0 155   0   0   0]
 [  6   0   0   0   0   0]
 [243   0   7   0   3   0]
 [ 10   0   0   0   0   0]]
              precision    recall  f1-score   support

           0       0.54      0.99      0.70       761
           1       0.00      0.00      0.00        69
           2       0.92      0.33      0.49       467
           3       0.00      0.00      0.00         6
           4       1.00      0.01      0.02       253
           5       0.00      0.00      0.00        10

    accuracy                           0.58      1566
   macro avg       0.41      0.22      0.20      1566
weighted avg       0.70      0.58      0.49      1566


/Users/spring/opt/anaconda3/lib/python3.8/site-packages/sklearn/me
trics/_classification.py:1318: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/spring/opt/anaconda3/lib/python3.8/site-packages/sklearn/me
trics/_classification.py:1318: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/spring/opt/anaconda3/lib/python3.8/site-packages/sklearn/me
trics/_classification.py:1318: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [142]:
```
#crossfold Validation of 7 folds for NB
cross_val_NB=sklearn.model_selection.cross_validate(NB, x_train, y=

print ('NB Train fit score is : ', '\n\n', cross_val_NB ['train_sco
print ('NB TEST score is : ', '\n\n', cross_val_NB ['test_score'])
```

```
NB Train fit score is :

 [0.59456869 0.60543131 0.59808307 0.5942492  0.5971246  0.5988502
1
 0.59565634]
NB TEST score is :

 [0.59003831 0.56704981 0.58045977 0.55555556 0.56704981 0.5604606
5
 0.58541267]
```

4.1.2 SGD Classifier

In [148]:
```
# 3rd Classifier SGDC
# SGD classification

from sklearn.linear_model import SGDClassifier
sgd= SGDClassifier()
sgd.fit(x_train,y_train)
sgd_y=sgd.predict(x_test)

print('Accuracy of SGD :', accuracy_score(y_test,sgd_y))
print ('Confusion Matrix of SGD : ', '\n\n', confusion_matrix(y_tes
print(classification_report(y_test,sgd_y))
```

```
Accuracy of SGD : 0.8920817369093231
Confusion Matrix of SGD :

 [[726   3  18   0  14   0]
 [ 18  29  22   0   0   0]
 [ 25   1 431   0  10   0]
 [  1   0   0   2   3   0]
 [ 25   2  22   0 204   0]
 [  4   0   1   0   0   5]]
              precision    recall  f1-score   support

           0       0.91      0.95      0.93       761
           1       0.83      0.42      0.56        69
           2       0.87      0.92      0.90       467
           3       1.00      0.33      0.50         6
           4       0.88      0.81      0.84       253
           5       1.00      0.50      0.67        10

    accuracy                           0.89      1566
   macro avg       0.92      0.66      0.73      1566
weighted avg       0.89      0.89      0.89      1566
```

In [150]:
```
sorted(cross_val_SGD.keys())
```
Out[150]:
```
['fit_time', 'score_time', 'test_score']
```

In [151]:
```
#crossfold Validation of 7 folds for SGD
cross_val_SGD=sklearn.model_selection.cross_validate(sgd, x_train,

print ('SGD Train fit score is : ', '\n\n', cross_val_SGD ['train_s
print ('SGD TEST score is : ', '\n\n', cross_val_SGD ['test_score']
```
```
SGD Train fit score is :

 [0.9942492  0.99361022 0.99392971 0.99520767 0.99520767 0.9942510
4
 0.99329288]
SGD TEST score is :

 [0.91954023 0.88122605 0.90613027 0.89463602 0.89463602 0.8982725
5
 0.85796545]
```

In [153]:
```
#Classifier: XGBOOST classification
from sklearn.ensemble import GradientBoostingClassifier

xgboost= GradientBoostingClassifier(n_estimators=90)
xgboost.fit(x_train,y_train)
xgboost_y=xgboost.predict(x_test)

print('Accuracy of XGBOOST :', accuracy_score(y_test,xgboost_y))
print ('Confusion Matrix of XGBOOST : ', '\n\n', confusion_matrix(y_
print(classification_report(y_test,xgboost_y))
```
```
Accuracy of XGBOOST : 0.8729246487867177
Confusion Matrix of XGBOOST :

 [[728   4  12   1  15   1]
 [ 23  23  22   0   1   0]
 [ 33   3 413   1  16   1]
 [  1   0   1   1   3   0]
 [ 37   3  14   0 197   2]
 [  4   0   1   0   0   5]]
              precision    recall  f1-score   support

           0       0.88      0.96      0.92       761
           1       0.70      0.33      0.45        69
           2       0.89      0.88      0.89       467
           3       0.33      0.17      0.22         6
           4       0.85      0.78      0.81       253
           5       0.56      0.50      0.53        10

    accuracy                           0.87      1566
   macro avg       0.70      0.60      0.64      1566
weighted avg       0.87      0.87      0.87      1566
```

In [154]:
```
cross_val_xgboost=sklearn.model_selection.cross_validate(xgboost, x

print ('XGBOOST Train fit score is : ', '\n\n', cross_val_xgboost [
print ('XGBOOST TEST score is : ', '\n\n', cross_val_xgboost ['test
```

```
XGBOOST Train fit score is :

 [0.98146965 0.98626198 0.9798722  0.98242812 0.98370607 0.9789204
7
 0.97892047]
XGBOOST TEST score is :

 [0.90229885 0.89463602 0.89655172 0.85823755 0.8697318  0.8790786
9
 0.86564299]
```

In [155]:
```
# Inverse Transform of label Encoder
print (label_enc.inverse_transform([0,1,2,3,4]))
```

```
['Data Analyst' 'Data Science Manager' 'Data Scientist' 'Data arch
itect'
 'Machine Learning Engineer']
```

4.1.4 Gradient Boosting Classifier (GB) and AdaBoost Classifier (AB)

In [29]:
```
# model

# 1.  Adaptive Boosting classifier (AdaBoostClassifier from sklearn

from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {"base_estimator__criterion" : ["gini", "entropy"],
              "base_estimator__splitter" :   ["best", "random"],
              "n_estimators": [1, 10, 20],
              'learning_rate': [1.0, 2.0, 3.0],
             }


tree = DecisionTreeClassifier(random_state = 42, class_weight = "ba

ada = AdaBoostClassifier(base_estimator =tree)


grid_search_Ada = GridSearchCV(ada, param_grid=param_grid)

grid_search_Ada.fit(x_train,y_train)

print('{}, {}'.format(grid_search_Ada.best_params_,grid_search_Ada.
```

```
{'base_estimator__criterion': 'gini', 'base_estimator__splitter':
'random', 'learning_rate': 2.0, 'n_estimators': 20}, 0.86355589768
52227
```

In [30]:
```
from sklearn.metrics import precision_score, recall_score
```

In [32]:
```
# average= average
# and find their average weighted by support (the number of true in
# This alters ‘macro’ to account for label imbalance;
# it can result in an F-score that is not between precision and rec

adaBoost_pred= grid_search_Ada.predict(x_test)
results = {}
results['ada boost:'] = 'Acc: {:.2f}, Prec: {:.2f}, Recall: {:.2f}'


print(results)
```

```
{'ada boost:': 'Acc: 0.89, Prec: 0.86, Recall: 0.65'}
```

In [ ]:
```
x_test,y_test

x_train_1, y_train_1

x_val_1, y_val_1
```

4.4. Deep NN

In [33]:
```
# Neural network- x: decsiption; y: position

from keras.models import Sequential
from keras.layers import Dense
from keras.utils.vis_utils import plot_model
from tensorflow.keras import initializers

# reduce the variance in the results

from numpy.random import seed
np.random.seed(1)

from tensorflow import random
random.set_seed(2)

import random as rn
rn.seed(1234)
```

In [34]:
```
x_train_1.shape
```

Out[34]: (3011, 37002)

In [42]:
```

# could compare resuts that Relu function could be set as prelu and
# PReLU is similar to Relu

model = Sequential([
    Dense(5, input_shape = [37002], activation="PReLU"),
    Dense(11, activation="relu"),
    Dense(18, activation="softmax") #using softmax for last layer b
])
```

In [44]:
```
model.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 5)                 185020

 dense_4 (Dense)             (None, 11)                66

 dense_5 (Dense)             (None, 18)                216

=================================================================
Total params: 185,302
Trainable params: 185,302
Non-trainable params: 0
_________________________________________________________________
```

In [45]:
```
#compile model
model.compile(loss="sparse_categorical_crossentropy",
            optimizer="sgd" , #stochastic gradient descent
            metrics=["sparse_categorical_accuracy"])
```

In [46]:
```
x_train,x_test,y_train,y_test
```

Out[46]:
```
(<4302x37002 sparse matrix of type '<class 'numpy.float64'>'
        with 1042966 stored elements in Compressed Sparse Row form
at>,
 <1844x37002 sparse matrix of type '<class 'numpy.float64'>'
        with 441635 stored elements in Compressed Sparse Row forma
t>,
 array([2, 2, 2, ..., 4, 4, 4]),
 array([1, 0, 0, ..., 0, 2, 4]))
```

In [47]:
```
x_train_1.sort_indices()
x_val_1.sort_indices()
x_test.sort_indices()
```

In [48]:
```
#fit model
history = model.fit(x_train_1, y_train_1, validation_data= (x_val_1
```

```
61/61 [==============================] - 1s 10ms/step - loss: 0.13
01 - sparse_categorical_accuracy: 0.9754 - val_loss: 0.4755 - val_
sparse_categorical_accuracy: 0.8730
Epoch 296/400
61/61 [==============================] - 1s 8ms/step - loss: 0.129
4 - sparse_categorical_accuracy: 0.9774 - val_loss: 0.4760 - val_s
parse_categorical_accuracy: 0.8737
Epoch 297/400
61/61 [==============================] - 1s 10ms/step - loss: 0.12
84 - sparse_categorical_accuracy: 0.9777 - val_loss: 0.4859 - val_
sparse_categorical_accuracy: 0.8699
Epoch 298/400
61/61 [==============================] - 0s 6ms/step - loss: 0.128
1 - sparse_categorical_accuracy: 0.9771 - val_loss: 0.4765 - val_s
parse_categorical_accuracy: 0.8737
Epoch 299/400
61/61 [==============================] - 0s 6ms/step - loss: 0.127
4 - sparse_categorical_accuracy: 0.9777 - val_loss: 0.4814 - val_s
parse_categorical_accuracy: 0.8706
Epoch 300/400
```

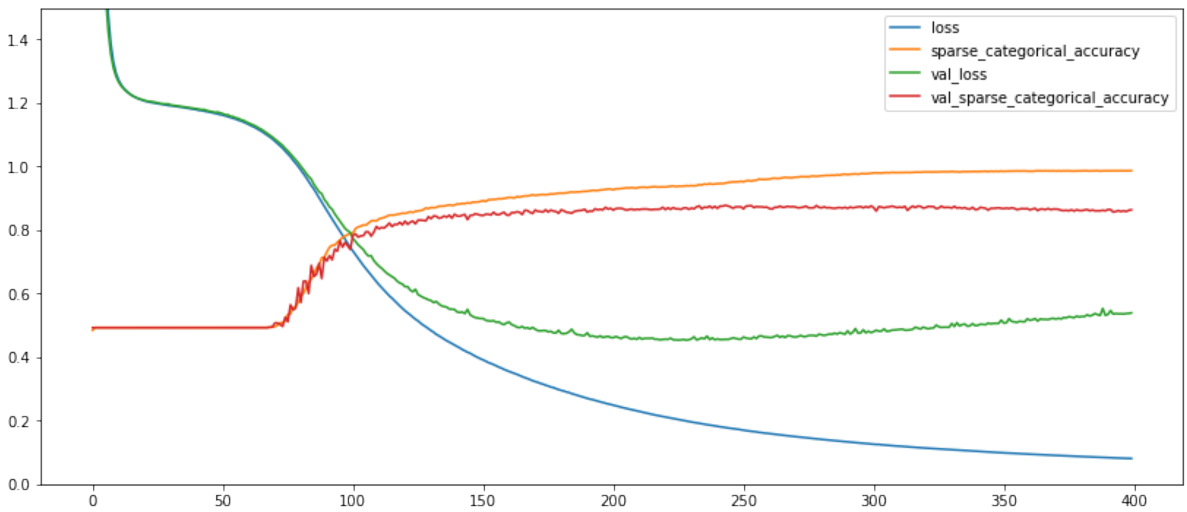
In [ ]:
```
x_test,y_test
```

In [49]:
```
#-- make predictions: use sum function for mutiple output lables.

test_predict = model.predict(x_test).sum(axis=1)

from sklearn.metrics import mean_absolute_error

mae_1 = mean_absolute_error(y_test, test_predict)

print('Test Error: {}'.format(mae_1))
```

```
58/58 [==============================] - 0s 4ms/step
Test Error: 1.3101952341981151
```

In [50]:
```
# Complete the code and plot your model

import matplotlib.pyplot as plt

pd.DataFrame(history.history).plot(figsize=(14, 6))

plt.gca().set_ylim(0, 1.5)

# is your model overfitting or underfitting?
# A model is overfitting when: loss on train set decreasing; loss o

# if there is a overfitting/underfitting problem, you might need to
```

Out[50]:
```
(0.0, 1.5)
```



4. Simple RNN

In [ ]:
```
# Simple RNN
```

In [ ]:
```
x_nn_train, y_nn_train

x_nn_valid, y_nn_valid
```

In [919]:

```
# Detect hardware, return appropriate distribution strategy
try:
    # TPU detection. No parameters necessary if TPU_NAME environmen
    # set: this is always the case on Kaggle.
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Running on TPU ', tpu.master())
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
else:
    # Default distribution strategy in Tensorflow. Works on CPU and
    strategy = tf.distribute.get_strategy()

print("REPLICAS: ", strategy.num_replicas_in_sync)
```

```
REPLICAS:  1
```

In [985]:

```
from keras.preprocessing import sequence, text
from keras.layers import GlobalMaxPooling1D, Conv1D, MaxPooling1D,
from keras.layers.embeddings import Embedding
from keras.layers.recurrent import LSTM, GRU,SimpleRNN


X_NN = df_model.description_clean
Y_NN = df_model.position

x_nn_train, x_nn_valid, y_nn_train, y_nn_valid = train_test_split(X
                                                  stratify=Y_NN,
                                                  random_state=42,
                                                  test_size=0.2, sh
```

In [986]:

```
# using keras tokenizer here

token = text.Tokenizer(num_words=None)
max_len = 1500

token.fit_on_texts(list(x_nn_train) + list(x_nn_valid))
x_nn_train_seq = token.texts_to_sequences(x_nn_train)
x_nn_valid_seq = token.texts_to_sequences(x_nn_valid)

#zero pad the sequences
x_nn_train_pad = sequence.pad_sequences(x_nn_train_seq, maxlen=max_
x_nn_valid_pad = sequence.pad_sequences(x_nn_valid_seq, maxlen=max_

word_index = token.word_index
```

In [988]:

```
# change datatype from pandas series to numpy adarray
y_nn_train= y_nn_train.to_numpy()
y_nn_valid= y_nn_valid.to_numpy()
```

In [989]:
```
#Initialzie an OneHotEncoder object
ohe = OneHotEncoder(handle_unknown='value')

# nominal feature (position) is encoded using one-hot encoding
# apply the encoding on training and validation set
y_nn_train_ohe = ohe.fit_transform(y_nn_train)
y_nn_valid_ohe = ohe.fit_transform(y_nn_valid)
```

In [ ]:

In [990]:
```
%%time
with strategy.scope():

# A simpleRNN without any pretrained embeddings and one dense layer
  # loss: categorical_crossentropy - categorical;
    # loss: sparse_categorical_crossentropy loss- int encoder is re
    # thus use categorical_crossentropy because y_nn_train_ohe data
    model_nn = Sequential()
    model_nn.add(Embedding(len(word_index) + 1,
                    300,
                    input_length=max_len))
    model_nn.add(SimpleRNN(100))
    model_nn.add(Dense(6, activation='softmax'))
    model_nn.compile(loss='categorical_crossentropy', optimizer='ad

model_nn.summary()
```

```
Model: "sequential_146"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_10 (Embedding)    (None, 1500, 300)         10184700

 simple_rnn_8 (SimpleRNN)    (None, 100)               40100

 dense_214 (Dense)           (None, 6)                 606

=================================================================
Total params: 10,225,406
Trainable params: 10,225,406
Non-trainable params: 0
_________________________________________________________________
CPU times: user 326 ms, sys: 135 ms, total: 461 ms
Wall time: 461 ms
```

In [ ]:

In [991]:
```
type(x_nn_train_pad)
```

Out[991]: numpy.ndarray

In [992]:
```
type(y_nn_train_ohe)
```

Out[992]: pandas.core.frame.DataFrame

In [993]:
```
y_nn_train
```

Out[993]:
```
array(['Machine Learning Engineer', 'Data Scientist', 'Data Analys
t', ...,
       'Data Analyst', 'Machine Learning Engineer',
       'Machine Learning Engineer'], dtype=object)
```

In [970]:
```
model_nn.fit(x_nn_train_pad, y_nn_train_ohe, epochs=5, batch_size=6
# Multiplying by Strategy to run on TPU's
```

```
Epoch 1/5
66/66 [==============================] - 114s 2s/step - loss: 1.19
07 - accuracy: 0.5163
Epoch 2/5
66/66 [==============================] - 119s 2s/step - loss: 0.77
95 - accuracy: 0.7245
Epoch 3/5
66/66 [==============================] - 139s 2s/step - loss: 0.63
68 - accuracy: 0.7856
Epoch 4/5
66/66 [==============================] - 110s 2s/step - loss: 0.38
39 - accuracy: 0.8757
Epoch 5/5
66/66 [==============================] - 92s 1s/step - loss: 0.294
5 - accuracy: 0.9001
```

Out[970]:
```
<keras.callbacks.History at 0x7fb285855a00>
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
# RNN: LSTM
x_train,x_test,y_train,y_test
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [732]:

```
pip install imageai --upgrade
```

```
WARNING: Ignoring invalid distribution -cipy (/Users/spring/opt/an
aconda3/lib/python3.8/site-packages)
WARNING: Ignoring invalid distribution -cipy (/Users/spring/opt/an
aconda3/lib/python3.8/site-packages)
Collecting imageai
  Downloading imageai-2.1.6-py3-none-any.whl (160 kB)
     |████████████████████████████████| 160 kB 930 kB/s eta 0:00:0
1
Collecting opencv-python
  Downloading opencv_python-4.6.0.66-cp36-abi3-macosx_10_15_x86_64
.whl (46.4 MB)
     |████████████████████████████████| 46.4 MB 12.0 MB/s eta 0:00
:01   |███                             | 4.1 MB 4.9 MB/s eta 0:00:
09
Collecting keras==2.4.3
  Downloading Keras-2.4.3-py2.py3-none-any.whl (36 kB)
Collecting pillow==7.0.0
  Downloading Pillow-7.0.0-cp38-cp38-macosx_10_9_x86_64.whl (2.1 M
B)
     |████████████████████████████████| 2.1 MB 22.5 MB/s eta 0:00:
```

In [733]:
```
#  adjust model complexity
#  Self-normalizing neural model could replace Batch normalization

import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import BatchNormalization

model2 = keras.models.Sequential([
    BatchNormalization( input_shape = [34125]), #  all layers in Ke
    keras.layers.Dense(10, activation="elu",kernel_initializer=kera
    BatchNormalization(),
    keras.layers.Dense(8, activation="elu",kernel_initializer=keras
    BatchNormalization(),
    keras.layers.Dense(6, activation="elu",kernel_initializer=keras
    BatchNormalization(),
    keras.layers.Dense(18, activation="softmax")
])
```

In [734]:
```

model2.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["sparse_categorical_accuracy"])

history_2 = model2.fit(x_train, y_train, epochs=400,
                    validation_data=(x_test, y_test),verbose = 0, b

pd.DataFrame(history_2.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent c
all last)
Input In [734], in <cell line: 5>()
      1 model2.compile(loss="sparse_categorical_crossentropy",
      2               optimizer="sgd",
      3               metrics=["sparse_categorical_accuracy"])
----> 5 history_2 = model2.fit(x_train, y_train, epochs=400,
      6                     validation_data=(x_test, y_test),verbo
se = 0, batch_size = 50)
      8 pd.DataFrame(history_2.history).plot(figsize=(8, 5))
      9 plt.grid(True)

File ~/opt/anaconda3/lib/python3.8/site-packages/keras/utils/trace
back_utils.py:67, in filter_traceback.<locals>.error_handler(*args
, **kwargs)
     65 except Exception as e:  # pylint: disable=broad-except
     66   filtered_tb = _process_traceback_frames(e.__traceback__)
---> 67   raise e.with_traceback(filtered_tb) from None
     68 finally:
     69   del filtered_tb
```

```
File ~/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python
/framework/func_graph.py:1147, in func_graph_from_py_func.<locals>
.autograph_handler(*args, **kwargs)
   1145 except Exception as e:  # pylint:disable=broad-except
   1146   if hasattr(e, "ag_error_metadata"):
-> 1147     raise e.ag_error_metadata.to_exception(e)
   1148   else:
   1149     raise

TypeError: in user code:

    File "/Users/spring/opt/anaconda3/lib/python3.8/site-packages/
keras/engine/training.py", line 1021, in train_function  *
        return step_function(self, iterator)
    File "/Users/spring/opt/anaconda3/lib/python3.8/site-packages/
keras/engine/training.py", line 1010, in step_function  **

    File "/Users/spring/opt/anaconda3/lib/python3.8/site-packages/
keras/engine/training.py", line 1000, in run_step  **

    File "/Users/spring/opt/anaconda3/lib/python3.8/site-packages/
keras/engine/training.py", line 859, in train_step

    File "/Users/spring/opt/anaconda3/lib/python3.8/site-packages/
keras/utils/traceback_utils.py", line 67, in error_handler


    TypeError: Exception encountered when calling layer "batch_nor
malization_4" (type BatchNormalization).

    Failed to convert elements of SparseTensor(indices=Tensor("Des
erializeSparse:0", shape=(None, 2), dtype=int64), values=Tensor("D
eserializeSparse:1", shape=(None,), dtype=float32), dense_shape=Te
nsor("stack:0", shape=(2,), dtype=int64)) to Tensor. Consider cast
ing elements to a supported type. See https://www.tensorflow.org/a
pi_docs/python/tf/dtypes
(https://www.tensorflow.org/api_docs/python/tf/dtypes) for support
ed TF dtypes.

    Call arguments received:
      • inputs=<tensorflow.python.framework.sparse_tensor.SparseTe
nsor object at 0x7fb261732e20>
      • training=True
```

In [ ]:
```
# save_fig("keras_learning_curves_graph")
plt.show()
model2.evaluate(x_test, y_test)
```

In [ ]:

In [ ]:

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
x_train,y_train,x_test, y_test
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
# Expand input data:
# x-- job position and job description
# y-- industry_code
```

In [236]:

```
df_model.columns
```

Out[236]:

```
Index(['position', 'Rating', 'Headquarters', 'Size', 'Founded',
       'Type of ownership', 'Industry_code', 'Sector', 'Revenue',
       'Competitors', 'Job_state', 'description_clean', 'skill_set
',
       'min_salary_$K', 'max_salary_$K', 'avg_salary_$K',
       'Headquarters_State'],
      dtype='object')
```

In [ ]:

```

X= df_model[['description_clean', 'position']]

Y= df_model.Industry_code

X=[re.sub(r"[^a-zA-Z0-9]+", ' ', k) for k in X]
X=[re.sub("[0-9]+",' ',k) for k in X]

#applying stemmer

ps = PorterStemmer()
X = [ps.stem(k) for k in X]

#Note: I have not removed stop words because there are important ke
tfidf=TfidfVectorizer()
label_enc=LabelEncoder()

X= tfidf.fit_transform(X)
Y= label_enc.fit_transform(Y)

x_train,x_test,y_train,y_test=train_test_split(X,Y,stratify=Y,test_
```

In [ ]:

```

```

In [ ]:

```
# NER with regexptagger
```

In [173]:
```
# Named Entity Recognition (two most likely trees could belong to o
# and they're considered as continuous and put into chunk)
# tag consecutive NNPs as one NE

from nltk import ne_chunk, pos_tag, word_tokenize
from nltk.tree import Tree

def get_continuous_chunks(text):
    chunked = ne_chunk(pos_tag(word_tokenize(text)))
    prev = None
    continuous_chunk = []
    current_chunk = []

    for i in chunked:
        if type(i) == Tree:
            current_chunk.append(" ".join([token for token, pos in
        elif current_chunk:
            named_entity = " ".join(current_chunk)
            if named_entity not in continuous_chunk:
                continuous_chunk.append(named_entity)
                current_chunk = []
        else:
            continue
    if current_chunk:
        named_entity = " ".join(current_chunk)
        if named_entity not in continuous_chunk:
            continuous_chunk.append(named_entity)
            current_chunk = []
    return continuous_chunk
```

In [ ]:

In [243]:
```
# Apply get_continuous_chunks and add as one new column
df_model['NER_set'] = df['Job Description'].apply(get_continuous_ch
```

In [ ]:
```
# use this new column as input to classify
```

In [246]:
```
df_model['NER_set']
```

Out[246]:
```
0       [Data Analyst, Veras Center, Justice, CIJ, Ver...
1       [Overview, Quality Management, Quality, HEDIS,...
2       [Senior Data, Insights, Analytics, Customer Op...
3       [Agile, Digital Strategy, Technology, Creative...
4       [ABOUT FANDUEL, FanDuel Group, FanDuel, FanDue...
                               ...
6157    [Us Tachyon, Digital, Tachyon Technologies, Ro...
6158    [Job, Interpret, Develop, Acquire, Identify, F...
6159    [Job DescriptionThe, Python, Splunk, ELK, Elas...
6160    [Security Analytics Data Engineer, MSSQL, SSRS...
6161    [UCB, Europe, US, UCB At UCB, Patient Safety, ...
Name: NER_set, Length: 5218, dtype: object
```

In [ ]:

In [186]:
```python
import nltk
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /Users/spring/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /Users/spring/nltk_data..
.
[nltk_data]   Unzipping corpora/words.zip.
```

Out[186]: True

In [232]:
```python
txt_3= df['Job Description']
type(txt_3)
```

Out[232]: pandas.core.series.Series

In [234]:
```python
# change format from series to string
txt_3= txt_3.to_string()
type(txt_3)
```

Out[234]: str

In [235]:
```python
# all chunks put together
get_continuous_chunks(txt_3)
```

Out[235]:
```
['Job',
 'Data Analyst',
 'Veras Center',
 'Justice',
 'CIJ',
 'Vera Institute',
 'Veras Center Justice',
 'CIJ CIJ',
 'Data Analyst Centers',
 'CIJ AWS',
 'Caspio',
 'SQL',
 'R',
 'Python',
 'Data Analyst Justice',
 'CIJ Unaccompanied Childrens Program',
 'UCP',
 'Legal Orientation Program',
 'Custodians',
```

In [248]:
```python
# Apply as new column
df_model['NER_set'] = df_clean_2['description_clean'].apply(get_con
```

In [ ]:
```python

```

In [ ]:
```
# new function that nned to explore

def conditions(tree_node):
    return tree_node.height() == 2

    def coninuous_entities(self, input_text, file_handle):
      from nltk import ne_chunk, pos_tag, word_tokenize
      from nltk.tree import Tree

      # Note: Currently, the chunker categorizes only 2 'NNP' toget
      docs = input_text.split('\n')
      for input_text in docs:
          chunked_data = ne_chunk(pos_tag(word_tokenize(input_text)
          child_data = [subtree for subtree in chunked_data.subtree

          named_entities = []
          for child in child_data:
              if type(child) == Tree:
                  named_entities.append(" ".join([token for token,

          # Dump all entities to file for now, we will see how to g
          if file_handle is not None:
              file_handle.write('\n'.join(named_entities) + '\n')
      return named_entities
```

In [310]:
```
df_clean['description_clean']
```

Out[310]:
```
0       are,you,eager,to,roll,up,your,sleeve,and,harne...
1       overview,provides,analytical,and,technical,sup...
2       were,looking,for,a,senior,data,analyst,who,ha,...
3       requisition,remoteyes,we,collaborate,we,create...
4       about,fanduel,group,fanduel,group,is,a,worldcl...
                              ...
6157    about,u,tachyon,technology,is,a,digital,transf...
6158    job,description,interpret,data,analyze,result,...
6159    job,descriptionthe,security,analytics,data,eng...
6160    the,security,analytics,data,engineer,will,inte...
6161    help,u,transform,patient,life,at,ucb,we,put,ou...
Name: description_clean, Length: 5218, dtype: object
```

In [313]:
```
df['Job Description']
```

Out[313]:
```
0       Are you eager to roll up your sleeves and harn...
1       Overview\n\nProvides analytical and technical ...
2       We’re looking for a Senior Data Analyst who ha...
3       Requisition NumberRR-0001939\nRemote:Yes\nWe c...
4       ABOUT FANDUEL GROUP\n\nFanDuel Group is a worl...
                              ...
6157    About Us\n\nTachyon Technologies is a Digital ...
6158    Job description\nInterpret data, analyze resul...
6159    Job DescriptionThe Security Analytics Data Eng...
6160    The Security Analytics Data Engineer will inte...
6161    Help us transform patients' lives.\nAt UCB, we...
Name: Job Description, Length: 5234, dtype: object
```

In [ ]:

```
# use Spacy to find top 100 entity and relative skills
```

In [314]:

```
import en_core_web_sm
import spacy

nlp = en_core_web_sm.load()

# list to store extracted skill keywords
skill_list = []

# feed the entire corpus into batches of 100 samples at a time
for i in range(0,len(df), 100):
    # for the last batch
    if i+np.mod(2253,100)==len(df):
        # combine job descriptions of 100 samples into a single str
        text = " ".join(des for des in df['Job Description'][i:len(
    else :
        text = " ".join(des for des in df['Job Description'][i:i+10

    # process raw text with the nlp object that holds all informati
    #features and relationships
    doc = nlp(text)

    # loop over the named entities
    for entity in set(doc.ents):
        # select entities with label 'ORG'
        if entity.label_ == 'ORG':
            # add to the list
            skill_list.append(entity.text)
```

In [340]:
```python
from collections import Counter

# count how many times each entity appears in the list
word_count = Counter(skill_list)

# print the top 100 named entities
word_count.most_common(100)
```

Out[340]:
```
[('SQL', 2930),
 ('SAS', 797),
 ('AI', 555),
 ('Hadoop', 553),
 ('Data Analyst', 541),
 ('IBM', 515),
 ('Bachelor', 453),
 ('BI', 431),
 ('ML', 411),
 ('TX', 403),
 ('Data Science', 394),
 ('Big Data', 364),
 ('ETL', 299),
 ('NLP', 261),
 ('Data Scientist', 255),
 ('PowerPoint', 247),
 ('Computer Science', 242),
 ('GSK', 238),
 ('SQL Server', 230),
 ('Microsoft Office', 208)
```

In [ ]:
```python
# based on (Spacy Skill extraction) skill_set, extract skill as new
```

In [ ]:
```python

```

In [354]:
```python


skill_label = ['ORG']

# make a list of actual skills extracted from the corpus (100)
skill_set = ['SQL','SQL Server','SSIS','SSRS', 'ETL', 'SQL\n','ELT'
             'SAS', 'Hadoop','AI',
             'BI','Business Intelligence',
             'Big Data','ETL','NLP','PowerPoint',
             'Microsoft Office', 'Microsoft','Microsoft Excel',
             'ML','Machine Learning','SVM',
             'SPSS', 'Oracle','Power BI','TensorFlow', 'JavaScript'
             'Matlab','MATLAB',
             'UAT','IoT','MIS'

            ]
def extract_skills(text):
    doc = nlp(text)
    results = [(ent.text) for ent in doc.ents if ent.label_ in skil
    return results

df_clean_2['spacy_skill_set'] = df['Job Description'].apply(extract
```

In [355]:
```
df_clean_2['spacy_skill_set']
```
Out[355]:
```
0                                                [SQL]
1                             [SQL, SAS, PowerPoint]
2                                     [BI, SQL, SQL]
3                                                [SQL]
4                                         [SQL, SQL]
                             ...
6157                                                []
6158               [SQL, JavaScript, ETL, SPSS, SAS]
6159    [Big Data, Oracle, Big Data, Oracle, SQL, SSRS...
6160    [Big Data, Oracle, Big Data, Oracle, SQL, SSRS...
6161                                                []
Name: spacy_skill_set, Length: 5218, dtype: object
```
In [ ]:
```

```
In [ ]:
```
# add match skills to new column
```
In [346]:
```
df_clean_2['skill_set']= df_clean['skill_set']
```
In [ ]:
```

```
In [370]:
```
spac= df_clean_2['spacy_skill_set'].copy()
spac.astype(str)
```
Out[370]:
```
0                                              ['SQL']
1                       ['SQL', 'SAS', 'PowerPoint']
2                                 ['BI', 'SQL', 'SQL']
3                                              ['SQL']
4                                       ['SQL', 'SQL']
                             ...
6157                                                []
6158          ['SQL', 'JavaScript', 'ETL', 'SPSS', 'SAS']
6159    ['Big Data', 'Oracle', 'Big Data', 'Oracle', '...
6160    ['Big Data', 'Oracle', 'Big Data', 'Oracle', '...
6161                                                []
Name: spacy_skill_set, Length: 5218, dtype: object
```
In [348]:
```
# change column name: skill_set to regex_skill_set

df_clean_2.rename(columns={'skill_set': 'regex_skill_set'}, inplace
```

In [374]:
```python
# skill_set using SpaCy
df_clean_2.head(5)
```

Out[374]:

	description_clean	regex_skill_set	spacy_skill_set
0	are,you,eager,to,roll,up,your,sleeve,and,harne...	Python, SQL	[SQL]
1	overview,provides,analytical,and,technical,sup...	SQL	[SQL, SAS, PowerPoint]
2	were,looking,for,a,senior,data,analyst,who,ha,...	SQL, Tableau, Visualization	[BI, SQL, SQL]
3	requisition,remoteyes,we,collaborate,we,create...	SQL, Tableau, Agile, Power BI, Visualization	[SQL]
4	about,fanduel,group,fanduel,group,is,a,worldcl...	Python, SQL, Visualization	[SQL, SQL]

In [383]:
```python
#  skill_set using regex
df_clean.head(2)
```

Out[383]:

	position	Rating	Headquarters	Size	Founded	Type of ownership	Industry_code	Sector
0	Data Analyst	3.2	New York, NY	201 to 500 employees	1961.0	Nonprofit Organization	62	Non-Profit
1	Data Analyst	3.8	New York, NY	10000+ employees	1893.0	Nonprofit Organization	62	Health Care

In [384]:
```python
df_clean_2['position']= df_clean['position']
df_clean_2['Industry_code']=df_clean['Industry_code']
```

In [385]:
```python
df_clean_2.head(3)
```

Out[385]:

	description_clean	regex_skill_set	spacy_skill_set	position	Ind
0	are,you,eager,to,roll,up,your,sleeve,and,harne...	Python, SQL	[SQL]	Data Analyst	
1	overview,provides,analytical,and,technical,sup...	SQL	[SQL, SAS, PowerPoint]	Data Analyst	
2	were,looking,for,a,senior,data,analyst,who,ha,...	SQL, Tableau, Visualization	[BI, SQL, SQL]	Data Analyst	

In [421]:
```python

X_2 = df_clean_2[['description_clean', 'regex_skill_set','position'
```

In [522]:
```python
Y_2 = df_clean_2 [['Industry_code']]
```

In [595]:
```python
x_train_2_ohe .shape
```

Out[595]: (3652, 258)

In [525]:
```

Y_2.shape
```
Out[525]: (5218, 1)

In [563]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [682]:
```
# Input: description_clean, regex_skill_set, position (next time wi
# Output: Industry_code

X_2 = df_clean_2[['description_clean', 'regex_skill_set','position'

Y_2 = df_clean_2 [['Industry_code']]

# divide input as train and test dataset

x_train_2,x_test_2,y_train_2,y_test_2= train_test_split(X_2,Y_2, st
```

In [684]:
```
x_train_2.shape
```
Out[684]: (3652, 3)

In [685]:
```
y_train_2.shape
```
Out[685]: (3652, 1)

In [686]:
```
# divide train dataset into train and validatio subsets.
x_train_3,x_val_2,y_train_3,y_val_2= train_test_split(x_train_2, y_
```

In [687]:
```
x_train_3.shape
```
Out[687]: (2556, 3)

In [ ]:
```
# train, validation and test datasets name
x_train_3,x_val_2, x_test_2
y_train_3,y_val_2, y_test_2
```

In [688]:
```
# deal with categorical data 'regex_skill_set' and 'position'(exclu

from category_encoders.one_hot import OneHotEncoder

#Initialzie an OneHotEncoder object
ohe = OneHotEncoder(handle_unknown='value')

#Learn and apply the encoding on training set
x_train_3_ohe = ohe.fit_transform(x_train_3[['regex_skill_set','pos
```

In [689]:
```
x_train_3_ohe.head(3)
```

Out[689]:

	regex_skill_set_1	regex_skill_set_2	regex_skill_set_3	regex_skill_set_4	regex_skill_set
3391	1	0	0	0	
1520	1	0	0	0	
961	1	0	0	0	

3 rows × 216 columns

In [700]:
```
#Apply the encoding on train and testing set

x_test_2_ohe = ohe.transform(x_test_2[['regex_skill_set','position'
x_val_2_ohe = ohe.transform(x_val_2[['regex_skill_set','position']]
```

In [ ]:
```
x_test_2_ohe
y_test_2_enc
```

In [699]:
```
# integer encodeuse:  label encoder to encode industry_code

label_encoder = LabelEncoder()

y_train_3_enc = label_encoder.fit_transform(y_train_3)
y_val_2_enc= label_encoder.fit_transform(y_val_2)
y_test_2_enc= label_encoder.fit_transform(y_test_2)
```

In [ ]:

In [692]:
```
y_train_2_enc
```

Out[692]:
```
array([13, 11, 11, ..., 11, 10, 11])
```

In [ ]:
```
# train, validation and test datasets name
x_train_3_ohe,x_val_2_ohe, x_test_2
y_train_3_enc,y_val_2_enc, y_test_2_enc
```

In [ ]:

In [ ]:
```
!!!!!! How to connect ohe-hot encoded two columns with job descript
```

In [570]:
```
whether or not do smote !!!!!

y_train_2.value_counts()
```

Out[570]:
```
Industry_code
51               1062
54                659
56                448
52                404
62                286
42                276
61                110
33                105
92                 86
44                 50
81                 38
72                 29
53                 24
45                 21
23                 20
21                 17
48                 13
71                  4
dtype: int64
```

In [849]:
```



from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import AlphaDropout
from keras.utils.vis_utils import plot_model

#this code is to randomness fix the randomness in the process
#so we can get reproducible results (not 100%, but to reduce the va
#but it is totally up to whether to set this up
from numpy.random import seed
np.random.seed(1)

from tensorflow import random
random.set_seed(2)

import random as rn
rn.seed(1234)
```

In [ ]:
```
# train, validation and test datasets name
x_train_3_ohe,x_val_2, x_test_2
y_train_3_enc,y_val_2, y_test_2_enc
```

In [694]:
```
x_train_3_ohe.shape
```

Out[694]:
```
(2556, 216)
```

In [856]:
```
initializer = tf.keras.initializers.HeNormal()


model_2 = Sequential([
    Dropout(.2, input_shape = [216]),  # dropout 后面一定要跟 input_s
    Dense(20,  activation="relu"),
    Dense(21, activation="relu"),
    Dense(3, kernel_initializer=initializer),
    AlphaDropout(.3),
    Dense(18, activation="softmax") #using softmax for last layer b
])
```

In [857]:
```
model_2.summary()
```

```
Model: "sequential_127"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dropout_10 (Dropout)        (None, 216)               0

 dense_196 (Dense)           (None, 20)                4340

 dense_197 (Dense)           (None, 21)                441

 dense_198 (Dense)           (None, 3)                 66

 alpha_dropout_8 (AlphaDropo  (None, 3)                0
 ut)

 dense_199 (Dense)           (None, 18)                72

=================================================================
Total params: 4,919
Trainable params: 4,919
Non-trainable params: 0
_________________________________________________________________
```

In [875]:
```
#compile model
model_2.compile(loss="sparse_categorical_crossentropy",
            optimizer="sgd" , #stochastic gradient descent
            metrics=["sparse_categorical_accuracy"])
```

In [870]:
```
x_train_3_ohe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2556 entries, 3391 to 312
Columns: 216 entries, regex_skill_set_1 to position_6
dtypes: int64(216)
memory usage: 4.2 MB
```

In [873]:

```
# change datatype to float

x_train_3_ohe = x_train_3_ohe.astype(float)
```

In [ ]:

```

```

In [876]:

```
#fit model
history = model_2.fit(x_train_3_ohe, y_train_3_enc, validation_data
```

```
Epoch 1/400
52/52 [==============================] - 1s 8ms/step - loss: 2.917
8 - sparse_categorical_accuracy: 0.0485 - val_loss: 2.8245 - val_s
parse_categorical_accuracy: 0.2719
Epoch 2/400
52/52 [==============================] - 0s 4ms/step - loss: 2.855
4 - sparse_categorical_accuracy: 0.0646 - val_loss: 2.7641 - val_s
parse_categorical_accuracy: 0.2911
Epoch 3/400
52/52 [==============================] - 0s 3ms/step - loss: 2.788
9 - sparse_categorical_accuracy: 0.0782 - val_loss: 2.7069 - val_s
parse_categorical_accuracy: 0.2911
Epoch 4/400
52/52 [==============================] - 0s 3ms/step - loss: 2.744
3 - sparse_categorical_accuracy: 0.1827 - val_loss: 2.6512 - val_s
parse_categorical_accuracy: 0.2911
Epoch 5/400
52/52 [==============================] - 0s 3ms/step - loss: 2.686
7 - sparse_categorical_accuracy: 0.2829 - val_loss: 2.5980 - val_s
parse_categorical_accuracy: 0.2911
```

In [878]:

```
#-- make predictions: use sum function for mutiple output lables.

test_predict_2 = model_2.predict(x_val_2_ohe).sum(axis=1)
```

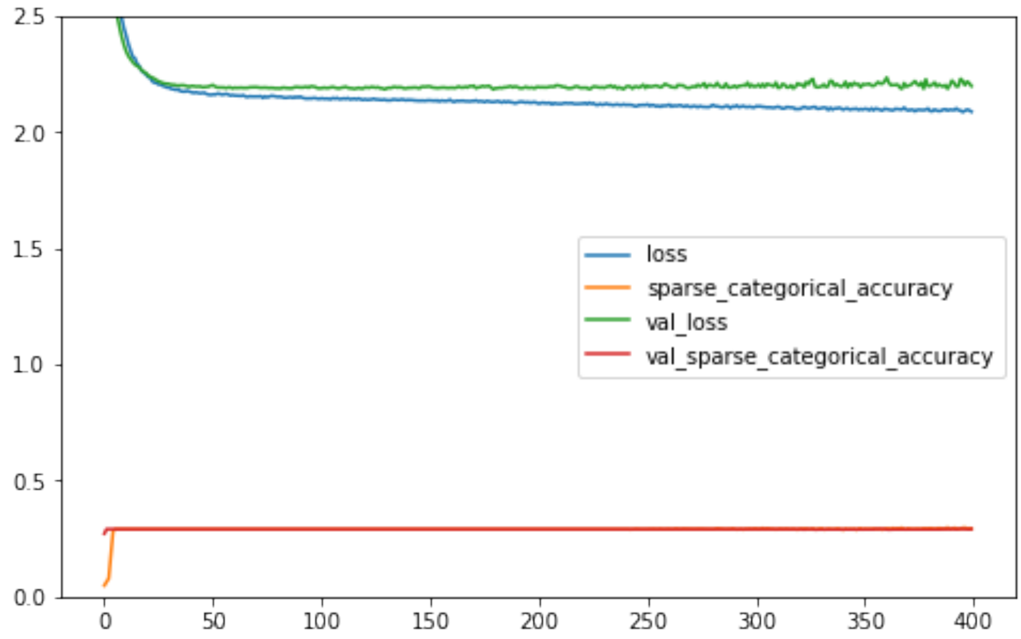
In [879]:

```
from sklearn.metrics import mean_absolute_error

mae = mean_absolute_error(y_val_2_enc, test_predict_2)
print('Test Error: {}'.format(mae))
```

```
Test Error: 7.614963513275567
```

In [880]:
```
#Complete the code and plot your model
import matplotlib.pyplot as plt
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.gca().set_ylim(0, 2.5)

#is your model overfitting or underfitting?
#A model is overfitting when:
#loss on train set decreasing
#loss on valid set stable/increasing

# if there is a overfitting/underfitting problem, you might need to
```

Out[880]: (0.0, 2.5)



In [ ]:
```
'''
performance keep stable

and train and validation loss decresing

neural network seems underfit


increase complexity: increase neurons or increase layers

'''
```

In [ ]:

In [ ]:
```
x_train_3_ohe, y_train_3_enc
x_val_2_ohe, y_val_2_enc
```

In [881]:
```
x_train_3_ohe.shape
```

Out[881]: (2556, 216)

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [281]:
```
regex_map
```

Out[281]:
```
{'\\WR\\W+\\s*': 'R',
 '(?i)\\WPython\\W': 'Python',
 '(?i)\\WHadoop\\W?': 'Hadoop',
 '(?i)SQL\\w*': 'SQL',
 '(?i)\\WTableau\\W?': 'Tableau',
 '(?i)\\WTensorFlow\\W?': 'TensorFlow',
 '(?i)\\WAgile\\W?': 'Agile',
 '(?i)\\WPower\\s?BI\\W?': 'Power BI',
 '(?i)\\WSSAS\\W?': 'SSaS',
 '(?i)\\WAlgorithms?\\W?': 'Algorithm',
 '(?i)Java\\w*': 'Java',
 '(?i)\\WVisualization\\W?': 'Visualization'}
```

In [ ]:
```
df_regex = pd.DataFrame({'skills': ['R','Python','Hadoop','SQL','Ta
                'Regex': ['\WR\W+\s*','(?i)\WPython\W','(?i)\WHadoop\W
                 "(?i)\WTensorFlow\W?","(?i)\WAgile\W?","(?i)\WPower\s
                "(?i)\WSSAS\W?","(?i)\WAlgorithms?\W?",'(?i)Java\w*','

df_clean_2 = pd.DataFrame(df_clean['description_clean'])

regex_map = dict(zip(df_regex.Regex, df_regex.skills))

def skill_collection(row):
    matches = []
    for reg in regex_map:
        if re.search(reg, row):
            matches.append(regex_map[reg])
    return ', '.join(matches)

df_clean_2['skill_set'] = df_clean_2['description_clean'].apply(ski
```

In [ ]:

In [ ]:
```
#
```

In [ ]:
```
use feature importance to filter columns
```

In [ ]:
```
df_model
```

In [ ]:

In [ ]:

In [ ]:

In [337]:
```
# data set used is df_clean
y= df_clean.Industry
X = df_clean.drop(columns = 'Industry')
```

In [340]:
```
y .shape
```

Out[340]: (10020,)

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [319]:
```
df_model= df_clean.copy()
```

In [327]:
```
df_model['position']= pd.DataFrame(df_model['position'])
```

In [335]:
```
pd.Series(df_model['position']).to_frame()
```

Out[335]:

	position
0	Data Analyst
1	Data Analyst
2	Data Analyst
3	Data Analyst
4	Data Analyst
...	...
10015	Machine Learning Engineer
10016	Machine Learning Engineer
10017	Machine Learning Engineer
10018	Machine Learning Engineer
10019	Data Scientist

10020 rows × 1 columns

In [ ]:
```
# 1. column 'position' is nominal, so use one-hot encoding
#
```

In [ ]:
```
from category_encoders.one_hot import OneHotEncoder

#Initialzie an OneHotEncoder object
ohe = OneHotEncoder(handle_unknown='value')

#Learn and apply the encoding on training set
d = ohe.fit_transform(df_model['position'])
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [315]:
```
# data set used is df_clean
y= df_clean.Industry
X = df_clean.drop(columns = 'Industry')
```

In [ ]:

In [ ]:

In [190]:
```
X= df[['Job Description', 'position']]
```

In [317]:
```
# Split data into train and test sets as well as for validation and

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
```

In [ ]:

In [318]:
```
# Setup the Bagging classifier (BaggingClassifier from sklearn)

from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

#Set Bagging parameters
baggging_params = {
    'base_estimator': DecisionTreeClassifier(max_depth = 10),
    'n_estimators': 100,
    'max_samples': 1.0,
    'max_features': 1.0,
    'bootstrap': True,
    'bootstrap_features': False,
    'n_jobs': -1,
    'random_state' : 42
}

#Initiate a BaggingClassifier object using the parameters set above
bagging = BaggingClassifier(**baggging_params)

#Fit the BaggingClassifier object on the resampled training data
bagging.fit(X_train, y_train)
```

```
---------------------------------------------------------------------------
_RemoteTraceback                          Traceback (most recent call last)
_RemoteTraceback:
"""
Traceback (most recent call last):
  File "/Users/spring/opt/anaconda3/lib/python3.8/site-packages/joblib/externals/loky/process_executor.py", line 436, in _process_worker
    r = call_item()
  File "/Users/spring/opt/anaconda3/lib/python3.8/site-packages/joblib/externals/loky/process_executor.py", line 288, in __call__
    return self.fn(*self.args, **self.kwargs)
  File "/Users/spring/opt/anaconda3/lib/python3.8/site-packages/joblib/_parallel_backends.py", line 595, in __call__
    return self.func(*args, **kwargs)
  File "/Users/spring/opt/anaconda3/lib/python3.8/site-packages/joblib/parallel.py", line 262, in __call__
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```