In [1]:
```
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 [2]:
```
data= pd.read_csv('/Users/spring/Desktop/BA Data/BA_data.csv')
```

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

Out[3]:
```
Index(['Job Title', 'Salary Estimate', 'Job Description', 'Rating'
,
       'Company Name', 'Location', 'Headquarters', 'Size', 'Founde
d',
       'Type of ownership', 'Industry', 'Sector', 'Revenue', 'Comp
etitors',
       'Easy Apply', 'index'],
      dtype='object')
```

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

Out[4]:
```
(6162, 16)
```

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

3.Data Collection and Preprocessing: Industry code- Table 5

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

In [6]:
```
df.Industry.unique()
```

Out[6]:
```
array(['Social Assistance', 'Health Care Services & Hospitals',
       'Internet', 'IT Services', 'Sports & Recreation',
       'Investment Banking & Asset Management', 'Insurance Carrier
s',
       'Venture Capital & Private Equity', 'Research & Development
', '-1',
       'Advertising & Marketing', 'Brokerage Services',
       'Building & Personnel Services', 'Consulting',
       'Casual Restaurants', 'Enterprise Software & Network Soluti
ons',
       'Lending', 'Banks & Credit Unions', 'Computer Hardware & So
ftware',
       'Staffing & Outsourcing',
       'Motion Picture Production & Distribution', 'Legal', 'Real
Estate',
       'Federal Agencies', 'TV Broadcast & Cable Networks', 'Accou
nting',
       'Food & Beverage Stores', 'Health, Beauty, & Fitness',
       'Biotech & Pharmaceuticals', 'Insurance Agencies & Brokerag
es'
```

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

df.loc[df['Industry'].str.contains("government"), 'Industry'] = '92
df.loc[df['Industry'].str.contains("regional agencies"), 'Industry'
```

```
# 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'



# 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
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
```

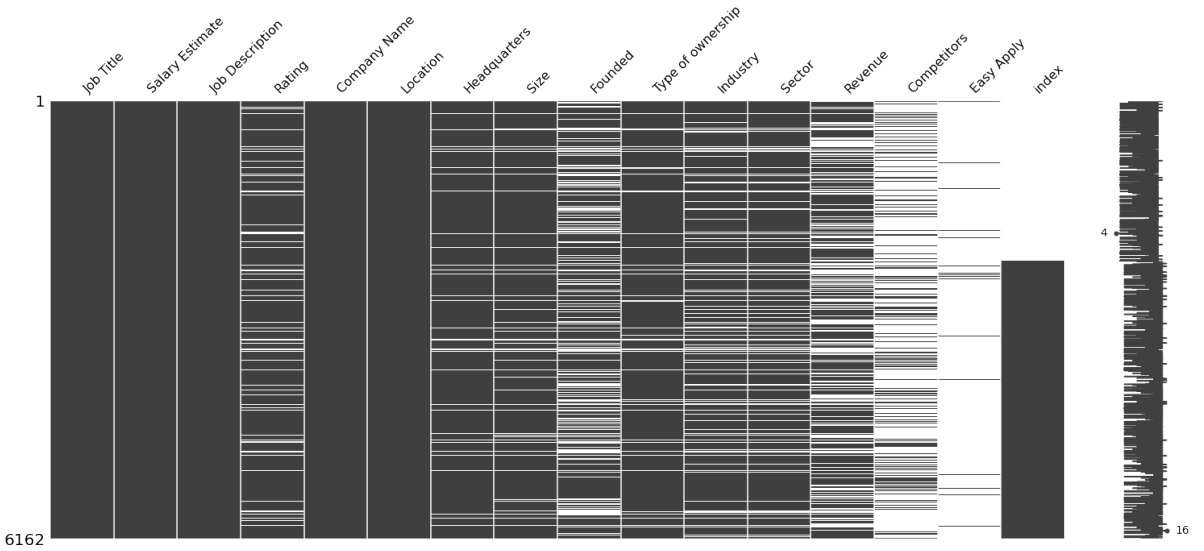
In [7]:
```
df.Industry.unique()
```

Out[7]:
```
array(['62', '51', '61', '52', '54', '-1', '23', '72', '42', '56',
'53',
       '92', '81', '71', '45', '33', '44', '48', '21', '11'], dtyp
e=object)
```

In [8]:
```
# -1 is replaced by nan value
df.replace([-1, '-1', 'Unknown', 'Unknown / Non-Applicable'], np.na
```

In [9]:
```
# check missing value distribution
import missingno as msno
msno.matrix(df)
```

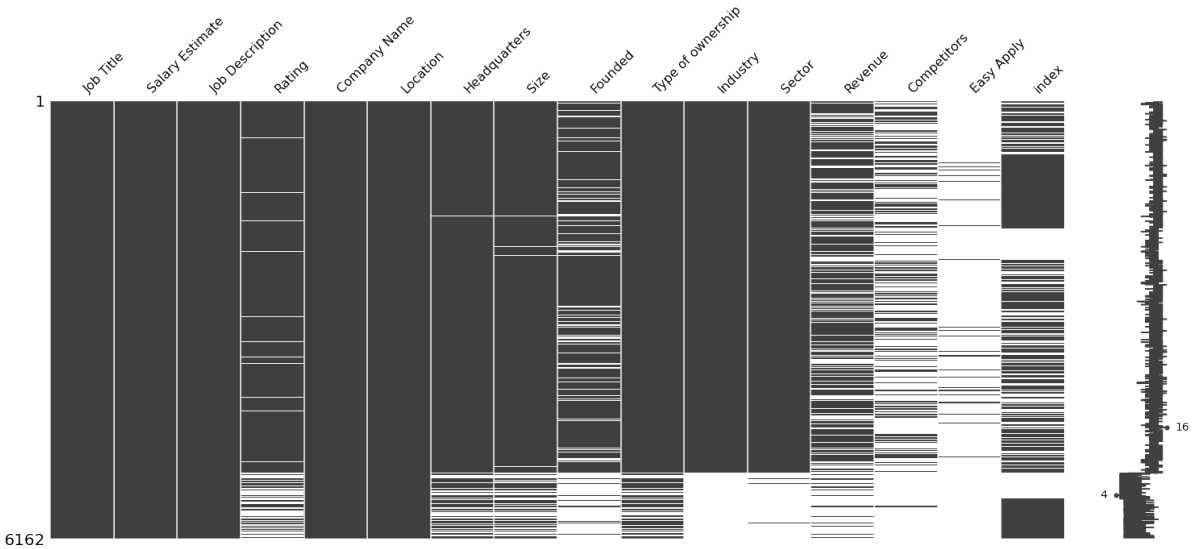
Out[9]:
```
<AxesSubplot:>
```



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 [10]:
```
msno.matrix(df.sort_values('Industry'))
```

Out[10]: <AxesSubplot:>



In [ ]:
```
'''
This implies MCAR. Deleting instances is likely a good idea.
'''
```

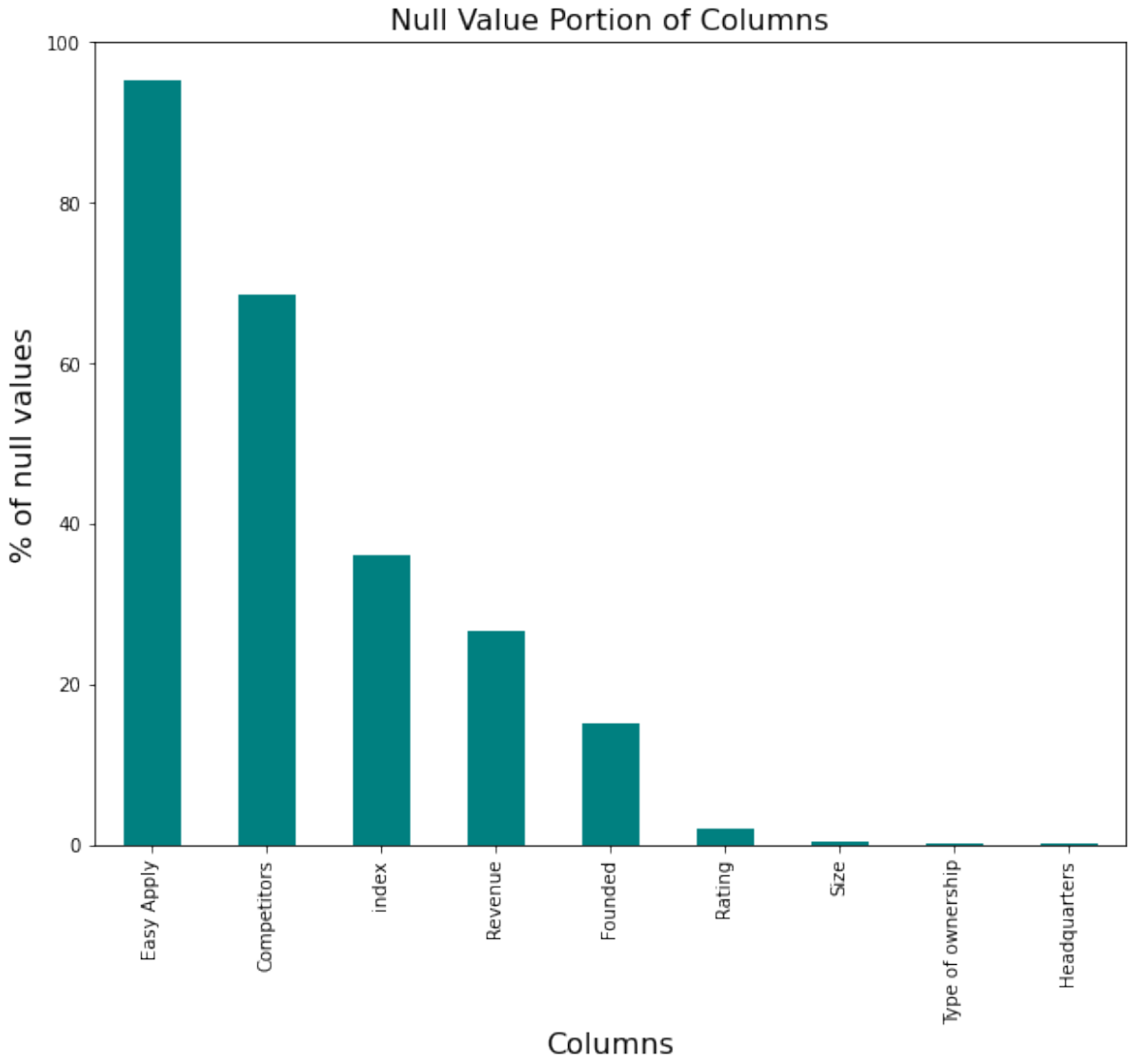
In [11]:
```
# only not null value in column Industry are kept.

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

In [12]:
```
df.shape
```

Out[12]: (5234, 16)

In [13]:
```
# 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[13]:
```
Text(0.5, 1.0, 'Null Value Portion of Columns')
```



In [15]:
```
# 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[15]:
```
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 [16]:
```
# Check the frequency counts of the values of output feature (Indus
df.Industry.value_counts()
```

Out[16]:
```
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 [17]:
```
df.head(2)
```

Out[17]:

	Job Title	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

3.Data Collection and Preprocessing: Table 1: Different expressions of job position titles

In [14]:
```
# change column name from job title to position
# and Industry to Industry_code

df.rename(columns={ df.columns[0]: 'position'}, inplace = True)
df.rename(columns= {df.columns[10]: 'Industry_code'}, inplace = Tru
```

In [15]:
```
df.head(2)
```

Out[15]:

	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 [20]:
```
df['position']
```

Out[20]:
```
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 [16]:
```
# 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 [17]:
```
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 [18]:
```
df.position.value_counts()
```

Out[18]:
```
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 [19]:
```
# 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
```

In [*]:
```
# 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 10)', size = 20)
plt.show()
```

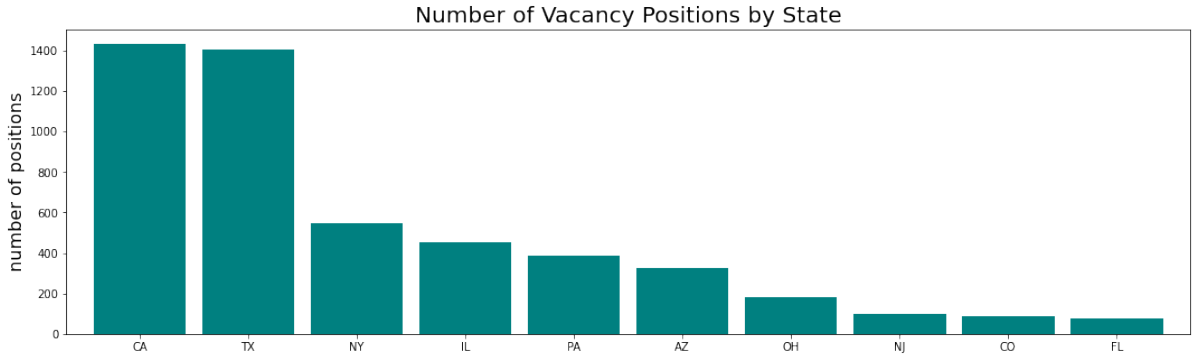
5.1.2 Vacant Positions by State

In [22]:
```
# Column 'Job_state'
# extract state only from columns 'Location' that have mixed info (

df['Job_state'] = df['Location'].apply(lambda x:x.split(",")[-1].st
```

In [23]:
```
df['Job_state'].head(2)
```

Out[23]:
```
0    NY
1    NY
Name: Job_state, dtype: object
```

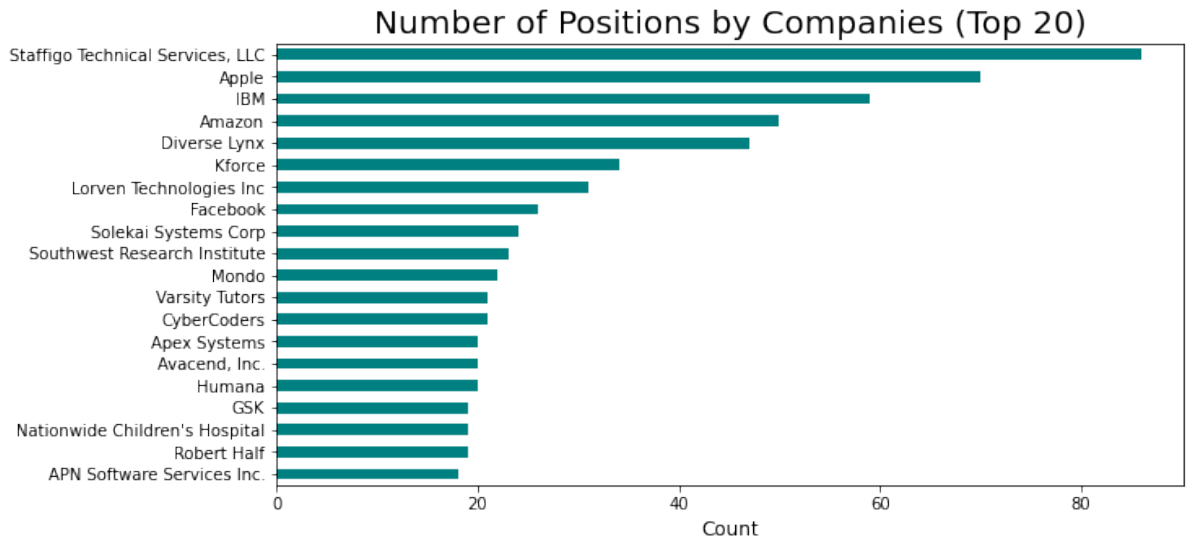
In [24]:
```
# Vacancy Positions by States

state = df.groupby('Job_state').count().sort_values('position',asce

state['position'].plot(kind = 'bar',figsize = (18,5) ,width = 0.85,
plt.xlabel('')
plt.ylabel('number of positions',size = 16)
plt.title('Number of Vacancy Positions by State', size = 20)
plt.yticks(size = 10)
plt.xticks(size = 10, rotation = 720)
plt.show()
```



5.1.3 Job Vacancy overview via the company.

In [25]:
```
# clean column 'Company Name' and  extract only the company name
df['Company Name'] = df['Company Name'].apply(lambda x:x.split("\n"
```

In [26]:
```
# Vacancy positions by Company
company = df.groupby(['Company Name']).count().sort_values('positio

company['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 Companies (Top 20)', size = 20)
plt.show()
```



In [ ]:
```

```

In [27]:
```
# For column 'job description', clean text

# Create a function to clean text data
def clean_text(text):
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\w*\d\w*', '', text)
    text = re.sub('[‘’“”…]', '', text)
    text = re.sub('\n',' ',text)
    return text
```

In [28]:
```
# Clean the text data and remove the job title 'Others'

clean = lambda x :clean_text(x)
df['description_clean'] = pd.DataFrame(df['Job Description'].apply(

df_clean = df[df['position'] != 'Others'].copy()
```

In [29]:
```
# rows about others are deleted
df_clean.shape
```

Out[29]: (5218, 19)

In [30]:
```
#  Tokenization and lowercase text

def tokenization(text):
    text = re.split('\W+', text)
    return text

df_clean['description_clean'] = df_clean['description_clean'].apply
```

In [31]:
```
df_clean['description_clean'].head(3)
```

Out[31]:
```
0    [are, you, eager, to, roll, up, your, sleeves,...
1    [overview, provides, analytical, and, technica...
2    [were, looking, for, a, senior, data, analyst,...
Name: description_clean, dtype: object
```

In [32]:
```
# Lemmentize the text data to improve analysis
import nltk
lemmatizers = nltk.WordNetLemmatizer()

def lemmatizer(text):
    text = [lemmatizers.lemmatize(word) for word in text]
    return text


df_clean['description_clean']= df_clean['description_clean'].apply(
```

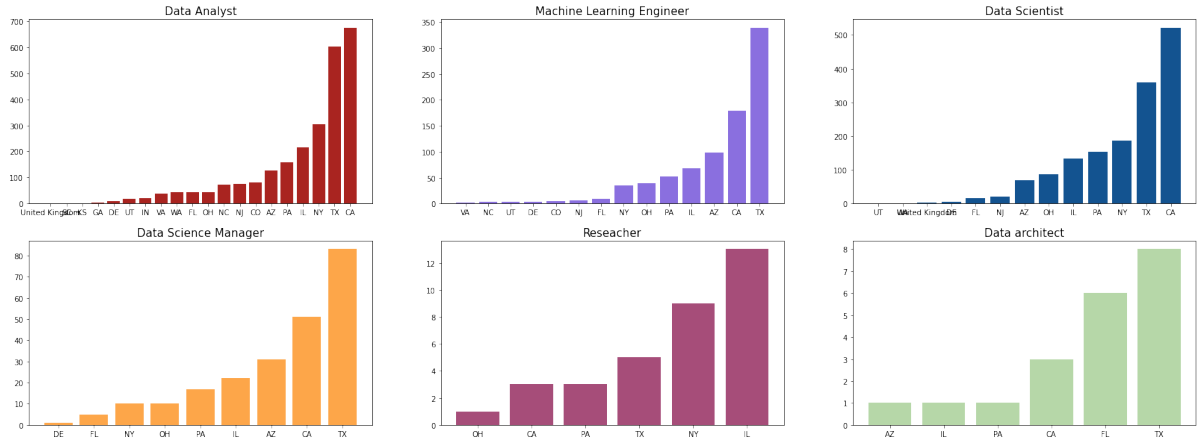
In [33]:
```
df_clean.head(2)
```

Out[33]:

	position	Salary Estimate	Job Description	Rating	Company Name	Location	Headquarters	
0	Data Analyst	$37K-66K (Glassdoor est.)	Are you eager to roll up your sleeves and harn...	3.2	Vera Institute of Justice	New York, NY	New York, NY	20 em
1	Data Analyst	$37K-66K (Glassdoor est.)	Overview\n\nProvides analytical and technical ...	3.8	Visiting Nurse Service of New York	New York, NY	New York, NY	em

5.1.4 Trend of job position: Number of vacancy positions (6 job positions) by State and Position.

In [34]:
```
# Position by State and Position: amount of vacancy positions (6 jo

import matplotlib.pyplot as plt

i = 1
color = ['#A92420','#8A6FDF','#135390','#FDA649','#a64d79','#b6d7a8
fig = plt.figure(figsize=(28,10))

for position in df_clean.position.unique():
    x =df_clean[df_clean['position']== str(position)].groupby(df_cl
    plt.subplot(2, 3, i)
    i=i+ 1
    plt.bar(x.index,x['Industry_code'], color = color[i-2])
    plt.xlabel('')
    plt.xticks(size = 10)
    plt.title(str(position), size = 15)
plt.show()
```



In [ ]:
```
```

In [35]:
```
# convert list to string
df_clean['description_clean'] = [','.join(map(str, text)) for text
```

In [36]:
```
# new column: Extract skill from column 'description_clean' (job des

df_regex = pd.DataFrame({'skills': ['R','Python','Hadoop','SQL','Tab
             '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_skill = 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_skill['regex_skill_set'] = df_skill['description_clean'].apply(sk
```

In [37]:
```
df_skill.head(2)
```

Out[37]:

	description_clean	regex_skill_set
0	are,you,eager,to,roll,up,your,sleeve,and,harne...	Python, SQL
1	overview,provides,analytical,and,technical,sup...	SQL

In [37]:
```
df_clean['regex_skill_set']= df_skill['regex_skill_set']
```

In [38]:
```
df_clean.head(2)
```

Out[38]:

	position	Salary Estimate	Job Description	Rating	Company Name	Location	Headquarters	
0	Data Analyst	$37K-66K (Glassdoor est.)	Are you eager to roll up your sleeves and harn...	3.2	Vera Institute of Justice	New York, NY	New York, NY	20 em
1	Data Analyst	$37K-66K (Glassdoor est.)	Overview\n\nProvides analytical and technical ...	3.8	Visiting Nurse Service of New York	New York, NY	New York, NY	em

In [39]:
```
df_clean.shape
```

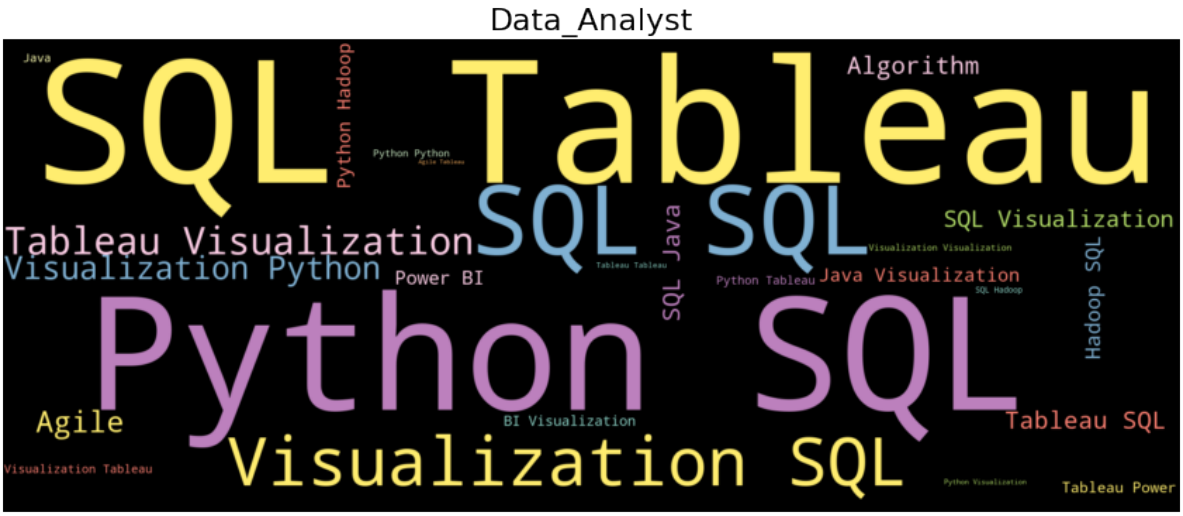
Out[39]:
```
(5218, 20)
```

5.3 Word cloud of 6 job positions

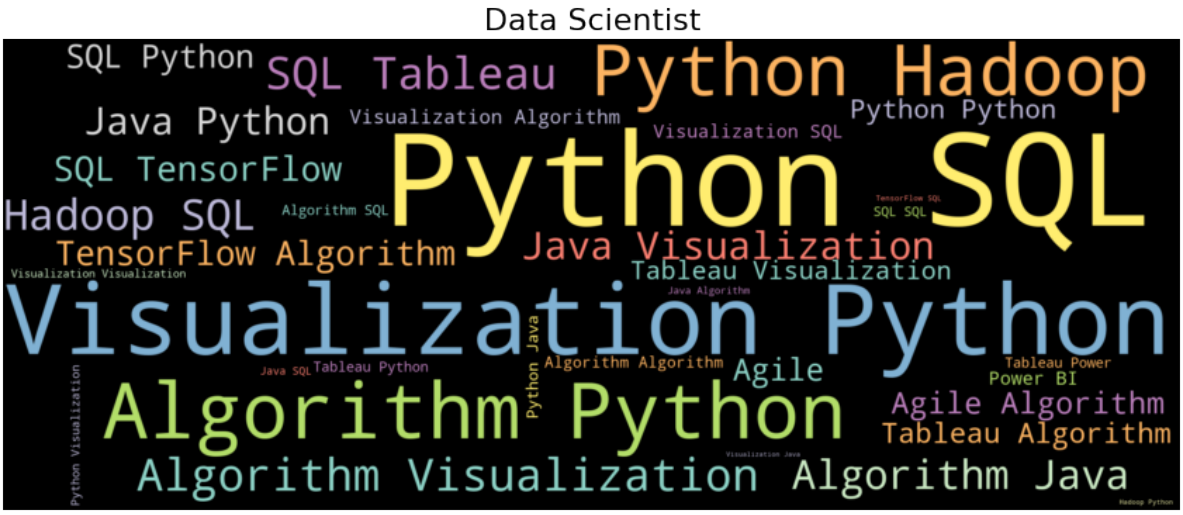
In [41]:
```
df_clean.position.value_counts()
```

Out[41]:
```
Data Analyst                  2536
Data Scientist                1556
Machine Learning Engineer      842
Data Science Manager           230
Reseacher                       34
Data architect                  20
Name: position, dtype: int64
```

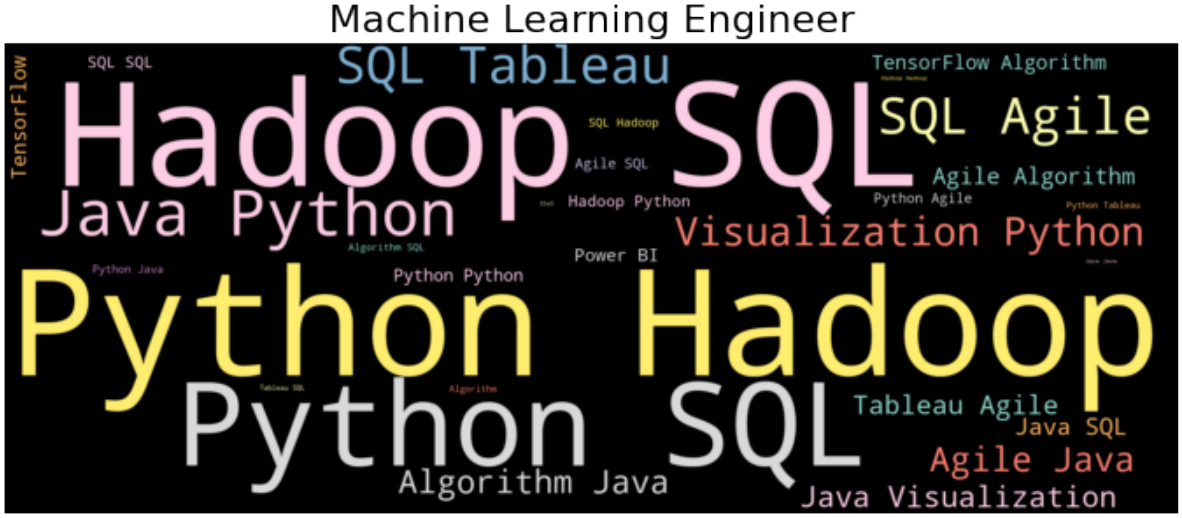
In [40]:
```
from wordcloud  import WordCloud, STOPWORDS
import matplotlib.pyplot as plt

# choose position= 'Data Analyst'
df_Data_Analyst= df_clean[df_clean.position=='Data Analyst']

# add stopwords
from sklearn.feature_extraction import text
extra_stopword = ['data','experience','work','team','will','skill',
stop_words = text.ENGLISH_STOP_WORDS.union(extra_stopword)

text = ",".join(review for review in df_Data_Analyst['regex_skill_s
wordcloud = WordCloud(stopwords=stop_words, width = 2000, height =
plt.figure(figsize=(15,10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.figure(1,figsize=(16,14))
plt.title('Data_Analyst',fontsize=22)
plt.show()
```



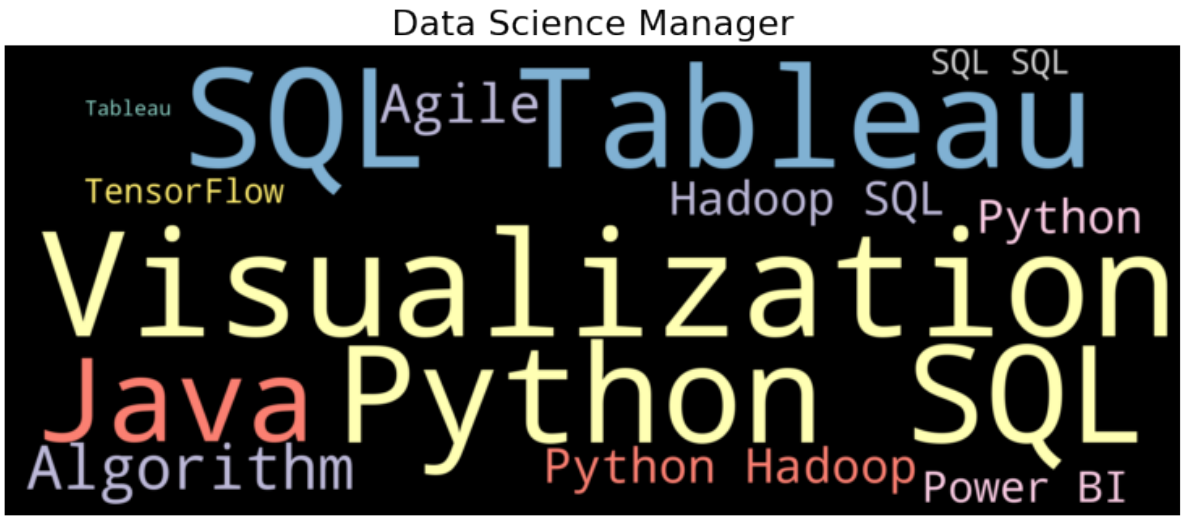
In [41]:
```
# choose position= 'Data Scientist'
df_Data_Scientist= df_clean[df_clean.position=='Data Scientist']

text = ",".join(review for review in df_Data_Scientist['regex_skill
wordcloud = WordCloud(stopwords=stop_words, width = 2000, height =
plt.figure(figsize=(15,10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.figure(1,figsize=(16,14))
plt.title('Data Scientist',fontsize=22)
plt.show()
```



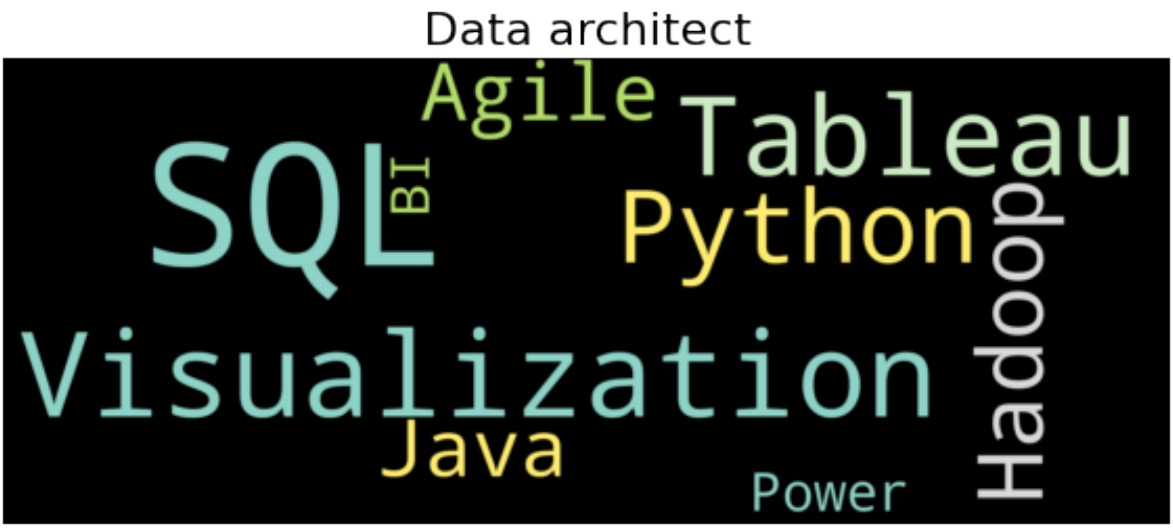
In [42]:
```
# choose position= 'Machine Learning Engineer'
df_Machine_Learning_Engineer= df_clean[df_clean.position=='Machine

text = ",".join(review for review in df_Machine_Learning_Engineer['
wordcloud = WordCloud(stopwords=stop_words, width = 2000, height =
plt.figure(figsize=(12,10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.figure(1,figsize=(16,14))
plt.title('Machine Learning Engineer',fontsize=22)
plt.show()
```



In [43]:
```
choose position= 'Data Science Manager'
f_Data_Science_Manager= df_clean[df_clean.position=='Data Science Ma

ext = ",".join(review for review in df_Data_Science_Manager['regex_s
ordcloud = WordCloud(stopwords=stop_words, width = 2000, height = 80
lt.figure(figsize=(13,6))
lt.imshow(wordcloud, interpolation='bilinear')
lt.axis("off")
lt.figure(1,figsize=(16,14))
lt.title('Data Science Manager',fontsize=22)
lt.show()
```



In [44]:
```
# 'Data architect'

df_architect= df_clean[df_clean.position=='Data architect']

text = ",".join(review for review in df_architect['regex_skill_set'
wordcloud = WordCloud(stopwords=stop_words, width = 2000, height =
plt.figure(figsize=(10,5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.figure(1,figsize=(7,5))
plt.title('Data architect',fontsize=22)
plt.show()
```



In [45]:
```
choose position= 'Reseacher'
_Reseacher= df_clean[df_clean.position=='Reseacher']

xt = ",".join(review for review in df_Reseacher['regex_skill_set'] )
rdcloud = WordCloud(stopwords=stop_words, width = 2000, height = 800
t.figure(figsize=(10,5))
t.imshow(wordcloud, interpolation='bilinear')
t.axis("off")
t.figure(1,figsize=(6,5))
t.title('Reseacher',fontsize=22)
t.show()
```

5.2 Experience and Education Requirement

In [47]:
```
# Print out the first 3 examples of matches

text = df_clean['Job Description'].values

limit = 0
for t in text:
    for sentence in t.split('\n'):
        if 'EXPERIENCE' in sentence:
            year = re.findall("\d{1,2}\+? year", sentence)
            if len(year)==1:
                print(year[0])
                print(sentence)
                print("*"*20)
                limit +=1
    if limit >= 5:
        break
```

```
4 year
OverviewCBO Data AnalystLocation: Pearland Administrative OfficeDe
partment: Health Care FinanceJob Type: Full TimeCOMPANY PROFILEKel
sey-Seybold Clinic. Changing the way health cares.Kelsey-Seybold C
linic is Houston's premier multispecialty group practice, founded
in 1949 by Dr. Mavis Kelsey. With 19 clinic locations and more tha
n 400 physicians, Kelsey-Seybold provides medical care in 55 medic
al specialties and is home to a nationally accredited Breast Diagn
ostic Center, Endoscopy Center, Infusion Center and Cancer Center.
Our mission is to provide our team members with exceptional opport
unities for professional and personal growth.JOB SUMMARYThe Consul
ting Analytics team serves as the go-to resource for providing rep
orting, support, and analysis to external departments with the ult
imate goal of identifying and supporting initiatives to further th
e Kelsey Triple Aim - value, quality, and the patient experience.
The team develops in-depth expertise in enterprise data assets and
in mining patient and healthcare provider data to deliver actionab
le insights to supported business units.The Central Business Offic
e (CBO) analyst will serve as a primary resource for providing rep
```

In [48]:
```
# Compile the year value found into a list

exp_year = []

for t in text:
    for sentence in t.split('\n'):
        if 'EXPERIENCE' in sentence:
            year = re.findall("\d{1,2}\+? year", sentence)
            if len(year)==1:
                print(year[0])
                print(sentence)
                print("*"*20)
                num = year[0].split(' ')
                exp_year.append(num[0])
```

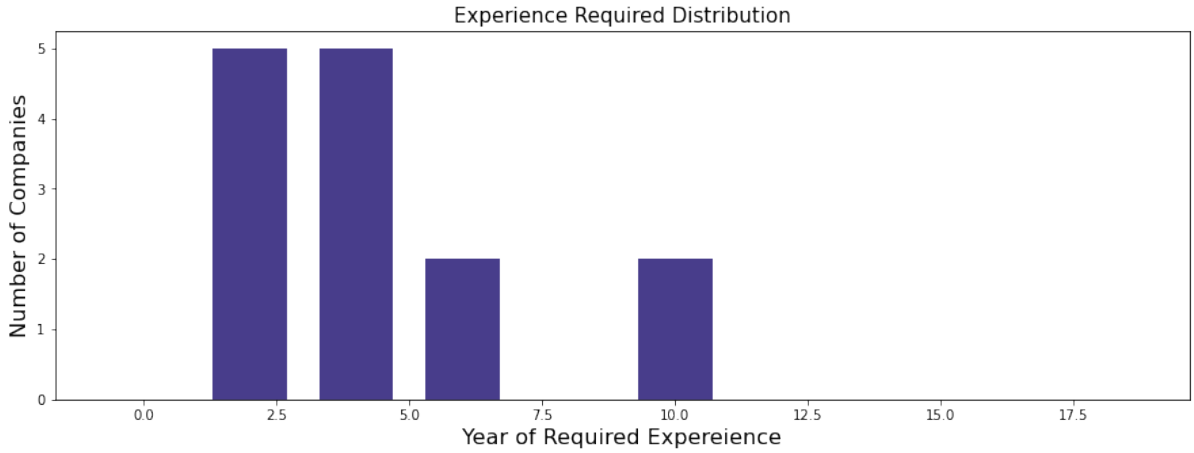
(https://www.dol.gov/ofccp/regs/compliance/posters/pdf/eeopost.pdf) If you are an individual with a disability and require a reasonable accommodation to complete any part of the application process, or are limited in the ability or unable to access or use this online application process and need an alternative method for applying , you may email "" for assistance. This email address is for accommodation requests only and cannot be used to inquire about the application process or status. For Pay Transparency Non Discrimination provision, please copy and paste the following link: Pay Transparency Nondiscrimination Provision; https://www.dol.gov/ofccp/pdf/pay-transp_%20English_formattedESQA508c.pdf (https://www.dol.gov/ofccp/pdf/pay-transp_%20English_formattedESQA508c.pdf) We maintain an Affirmative Action Plan for the purpose of proactively seeking employment and advancement for qualified protected veterans and individuals with disabilities. Upon request, we will schedule time to make our Affirmative Action Plan accessible. If you are interested, please submit a written request with the email subject line: 2020 Request to View Affirmative Action Plan to the Compliance Administrator at "" This email box is not for resumes or follow up on job applications

In [49]:
```
exp_year
```

Out[49]:
```
['4', '3+', '5+', '6', '4', '3', '10+', '10+', '5+', '6', '4', '3', '3', '2']
```

In [50]:
```
# Remove the '+' sign after year value
for n,i in enumerate(exp_year):
    if "+" in i:
        exp_year[n] = re.sub(r'\+','',i)
exp_year = [int(item) for item in exp_year]
```

In [51]:
```
plt.figure(figsize = (15,5))
plt.hist(exp_year,bins = list(range(0,21,2)), align = 'left', color
plt.title('Experience Required Distribution', size = 15)
plt.ylabel('Number of Companies', size = 16)
plt.xlabel('Year of Required Expereience', size = 16)
plt.show()
print(f'The average year of experience required is {round(np.mean(e
```



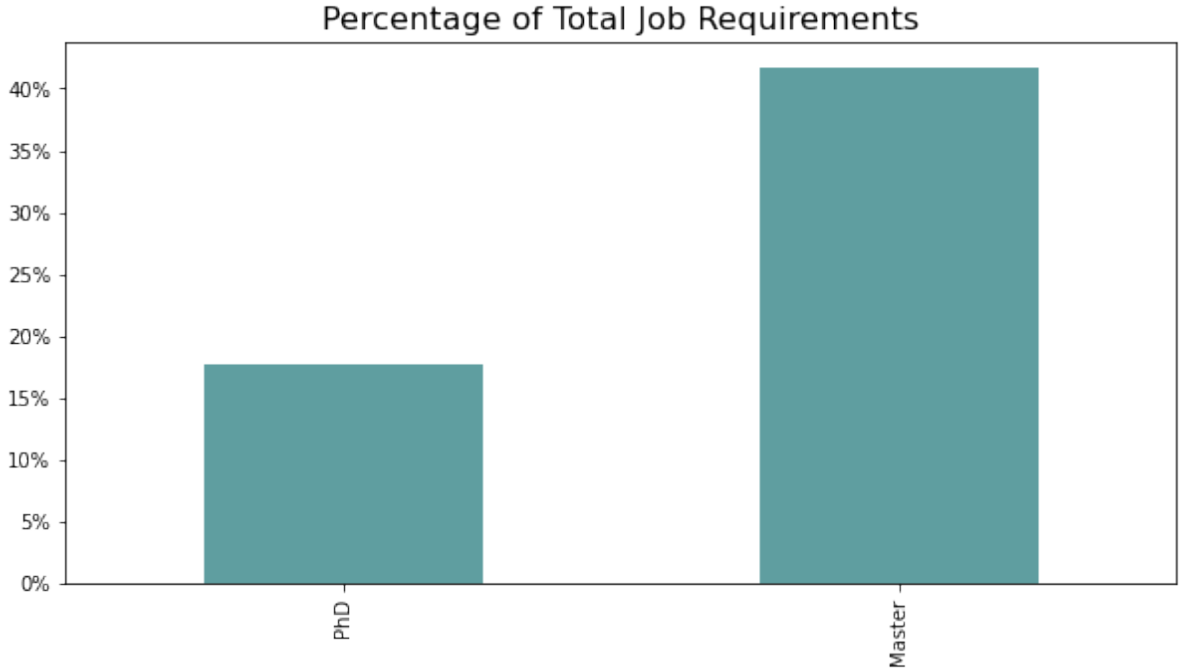
The average year of experience required is 4.86 years

In [52]:
```
# extract education info from 'Job Description'

# Define regex pattern and seach for PhD
pattern = re.compile('(?i)\WPh.?D\W')
pattern2 = re.compile('(?i)\WDoctorate\W')
count = 0
for t in text:
    if pattern.search(t):
        count +=1
    elif pattern2.search(t):
        count +=1
degree = {"PhD": count}

# Define regex pattern and seach for Master
pattern = re.compile("(?i)\WMasters?'?s?\W")
pattern2 = re.compile('(?i)\WM.?S\W')
count = 0
for t in text:
    if pattern.search(t):
        count +=1
    elif pattern2.search(t):
        count +=1
degree.update({"Master":count})
```

In [53]:
```
degree = pd.DataFrame.from_dict(degree,orient='index',
                                columns=[ 'count'])
degree['ptg'] = degree['count']/len(text)
```

In [54]:
```
ax =degree['ptg'].plot(kind = "bar", figsize =(10,5), color= 'cadet
ax.set_title('Percentage of Total Job Requirements' , size = 16)
ax.set_xticklabels(degree.index)
ax.set_yticklabels(['{:.0%}'.format(x) for x in ax.get_yticks()])
plt.show()
```

```
/var/folders/jf/p_35y64n6m75xk9k6br1fsfw0000gn/T/ipykernel_20057/5
66847158.py:4: UserWarning: FixedFormatter should only be used tog
ether with FixedLocator
  ax.set_yticklabels(['{:.0%}'.format(x) for x in ax.get_yticks()]
)
```



In [55]:
```
df_clean['Easy Apply'].isnull().sum()
```

Out[55]:
```
4977
```

In [56]:
```
# clean column 'Salary Estimate'
df_clean['Salary Estimate'][:3]
```

Out[56]:
```
0    $37K-$66K (Glassdoor est.)
1    $37K-$66K (Glassdoor est.)
2    $37K-$66K (Glassdoor est.)
Name: Salary Estimate, dtype: object
```

In [57]:
```
# delete bracket and its content
import re
df_clean['Salary Estimate'] = df_clean['Salary Estimate'].str.repla
```

```
/var/folders/jf/p_35y64n6m75xk9k6br1fsfw0000gn/T/ipykernel_20057/3
177852856.py:3: FutureWarning: The default value of regex will cha
nge from True to False in a future version.
  df_clean['Salary Estimate'] = df_clean['Salary Estimate'].str.re
place(r"\(.*\)","")
```

In [58]:
```
# the minimum salary offered
df_clean['min_salary_$K'] = df_clean['Salary Estimate'].apply(lambda
                                                              if not pd.

# the maximum salary offered
df_clean['max_salary_$K'] = df_clean['Salary Estimate'].apply(lambda
                                                              if not pd.

# the average salary offered
df_clean['avg_salary_$K'] = (df_clean['min_salary_$K'] + df_clean['m

# Delete original column 'Salary Estimate' and create 3 new columns
df_clean.drop(columns = 'Salary Estimate', inplace=True)
```

In [58]:
```
df_clean.head(3)
```

Out[58]:

	position	Salary Estimate	Job Description	Rating	Company Name	Location	Headquarters
0	Data Analyst	$37K-66K	Are you eager to roll up your sleeves and harn...	3.2	Vera Institute of Justice	New York, NY	New York, NY
1	Data Analyst	$37K-66K	Overview\n\nProvides analytical and technical ...	3.8	Visiting Nurse Service of New York	New York, NY	New York, NY
2	Data Analyst	$37K-66K	We're looking for a Senior Data Analyst who ha...	3.4	Squarespace	New York, NY	New York, NY

In [ ]:
```
```

In [ ]:
```
```

5.1.2 Vacant Positions by State

In [59]:
```
df_clean.Headquarters.value_counts()
```

Out[59]:
```
New York, NY          466
Chicago, IL           205
San Diego, CA         177
Austin, TX            124
Los Angeles, CA       122
                     ...
San Clemente, CA        1
Santa Barbara, CA       1
Renton, WA              1
Jenkintown, PA          1
Omaha, NE               1
Name: Headquarters, Length: 598, dtype: int64
```

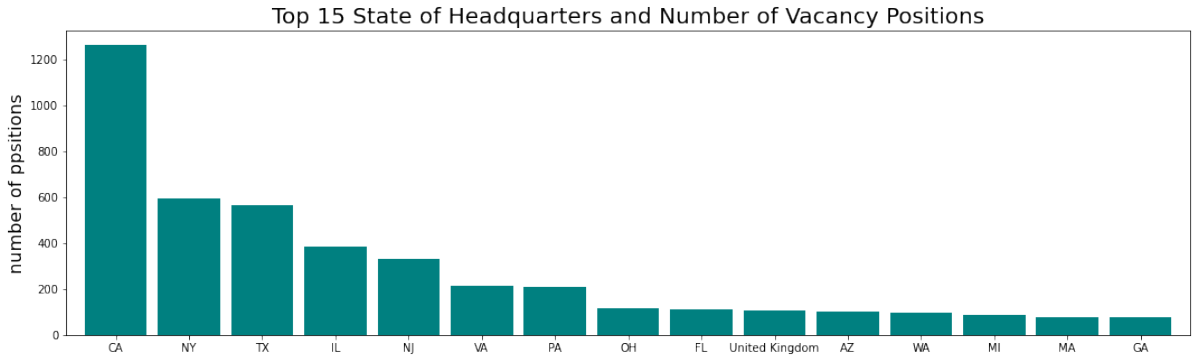
In [60]:
```
df_clean['Headquarters'] = df_clean['Headquarters'].astype(str)
```

In [61]:
```
# Positions by States from Headquarters

df_clean['Headquarters_State'] = df_clean['Headquarters'].apply(lam
```

In [62]:
```
df_clean['Headquarters_State']
```

Out[62]:
```
0            NY
1            NY
2            NY
3            VA
4            NY
          ...
6157         TX
6158         IL
6159         TX
6160         NE
6161    Belgium
Name: Headquarters_State, Length: 5218, dtype: object
```

In [63]:
```
# Positions by States from Headquarters

df_clean['Headquarters_State'] = df_clean['Headquarters'].apply(lam

Headquarters_state = df_clean.groupby('Headquarters_State').count()

Headquarters_state['position'].plot(kind = 'bar',figsize = (18,5) ,
plt.xlabel('')
plt.ylabel('number of ppsitions',size = 16)
plt.title('Top 15 State of Headquarters and Number of Vacancy Posit
plt.yticks(size = 10)
plt.xticks(size = 10, rotation = 720)
plt.show()
```



In [64]:
```
df_clean.head(2)
```

Out[64]:

	position	Salary Estimate	Job Description	Rating	Company Name	Location	Headquarters	
0	Data Analyst	$37K-66K	Are you eager to roll up your sleeves and harn...	3.2	Vera Institute of Justice	New York, NY	New York, NY	20 em
1	Data Analyst	$37K-66K	Overview\n\nProvides analytical and technical ...	3.8	Visiting Nurse Service of New York	New York, NY	New York, NY	em

2 rows × 21 columns

In [ ]:
```
```

In [65]:
```
df_model= df_clean.copy()
```

In [ ]:
```
```

In [ ]:
```
#  New Column (Job backrgound): getting skill collection list
```

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]:
```
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),
```

In [ ]:
```
```

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

In [71]:
```
pip install NLP-python
```

```
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 NLP-python
  Downloading NLP_python-1.1.0-py3-none-any.whl (3.0 kB)
Requirement already satisfied: numpy in /Users/spring/opt/anaconda
3/lib/python3.8/site-packages (from NLP-python) (1.23.0)
Requirement already satisfied: scikit-learn in /Users/spring/opt/a
naconda3/lib/python3.8/site-packages (from NLP-python) (1.0.2)
Requirement already satisfied: pandas in /Users/spring/opt/anacond
a3/lib/python3.8/site-packages (from NLP-python) (1.4.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /Users/sp
ring/opt/anaconda3/lib/python3.8/site-packages (from pandas->NLP-p
ython) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /Users/spring/opt/a
naconda3/lib/python3.8/site-packages (from pandas->NLP-python) (20
22.1)
Requirement already satisfied: six>=1.5 in /Users/spring/opt/anaco
```

In [81]:
```
import en_core_web_sm
import spacy

nlp = en_core_web_sm.load()

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

In [82]:
```
df_clean['spacy_skill_set']
```

Out[82]:
```
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 [ ]:
```
# New Column (Job backrgound) add match skills to new column
```

In [ ]:
```
Above are 3 information extraction methods: Regex, NER and SpaCy. A
    NER is important information extraction method.
```

In [ ]:
```
# new column job_backgroud based on description_clean that delete s
df_model['job_backgroud']
```

In [82]:
```
df_model['job_backgroud'] = df_model.apply(lambda row: row.descript
```

In [ ]:

4.2 RCNN

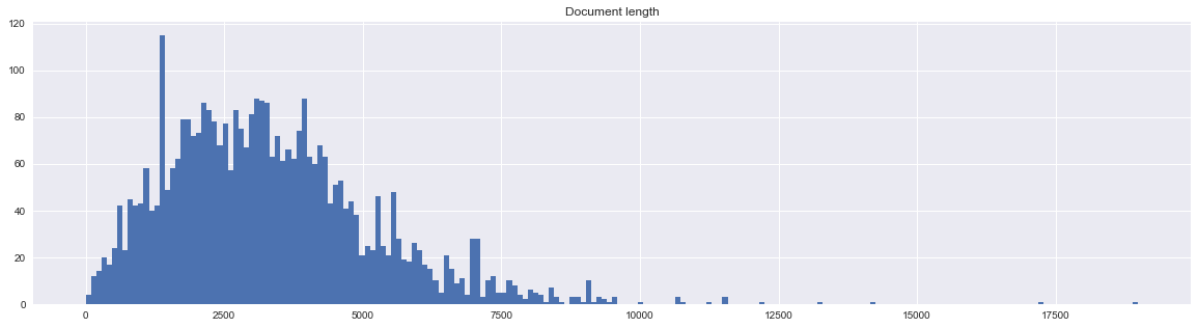
In [ ]:
```
# CNN

"""
       Arguments:
           embedding_layer    : If not defined with pre-trained emb
           num_words          : Maximal amount of words in the voca
           embedding_dim      : Dimension of word representation (d
           max_seq_length     : Max length of word sequence (defaul
           filter_sizes       : An array of filter sizes per channe
           feature_maps       : Defines the feature maps per channe
           use_char           : If True, char-based model will be a
           char_embedding_dim : Dimension of char representation (d
           char_max_length    : Max length of char sequence (defaul
           alphabet_size      : Amount of differnent chars used for
           hidden_units       : Hidden units per convolution channe
           dropout_rate       : If defined, dropout will be added a
           nb_classes         : Number of classes which can be pred
       """
```

4.4 RCNN with job backgroud: full text infomration exclude skill-relevant information

In [83]:
```
# Dataset:

X_5= df_model.job_backgroud
Y_5= df_model.position


x_train_5, x_test_5, y_train_5, y_test_5= train_test_split(X_5,Y_5,


# WORD-level
MAX_NUM_WORDS  = 15000
EMBEDDING_DIM  = 300
MAX_SEQ_LENGTH = 200
USE_GLOVE      = True
KERNEL_SIZES   = [3,4,5]
FEATURE_MAPS   = [200,200,200]

# CHAR-level
USE_CHAR           = False
ALPHABET           = " abcdefghijklmnopqrstuvwxyz0123456789-,;.!?:'
ALPHABET_SIZE      = len(ALPHABET)
CHAR_EMBEDDING_DIM = 300
CHAR_MAX_LENGTH    = 2500
CHAR_KERNEL_SIZES  = [5,10,20]
CHAR_FEATURE_MAPS  = [200,200,200]

# GENERAL
DROPOUT_RATE = 0.5
HIDDEN_UNITS = 250
NB_CLASSES   = 6

# LEARNING
BATCH_SIZE = 100
NB_EPOCHS  = 10
RUNS       = 5
VAL_SIZE   = 0.2
```

In [ ]:
```
# Data: train and test

X_5= df_model.job_backgroud.apply(clean_doc)
Y_5= df_model.position.apply(clean_doc)


x_train_5, x_test_5, y_train_5, y_test_5= train_test_split(X_5,Y_5,


print(f"Train samples: {len(x_train_5)}")
print(f"Test samples:  {len(x_test_5)}")
```

In [84]:
```
# Data preparation for word-based model

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
plt.style.use('seaborn')

tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=MAX_NUM
tokenizer.fit_on_texts(x_train_5)

sequences  = tokenizer.texts_to_sequences(x_train_5)
word_index = tokenizer.word_index
result     = [len(x) for x in x_train_5]

# Plot histogram
plt.figure(figsize=(20,5))
plt.title("Document length")
plt.hist(result, 200, density=False, range=(0,np.max(result)))
plt.show()

print("Text informations:")
print(f" - max length:   {np.max(result)}")
print(f" - min length:   {np.min(result)}")
print(f" - mean length:  {np.mean(result)}")
print(f" - limit length: {MAX_SEQ_LENGTH}")

# Padding all sequences to same length of `MAX_SEQ_LENGTH`
word_data = tf.keras.preprocessing.sequence.pad_sequences(
    sequences,
    maxlen  = MAX_SEQ_LENGTH,
    padding = 'post'
)
```




```
Text informations:
 - max length:   18989
 - min length:   5
 - mean length:  3329.103231106243
 - limit length: 200
```

In [786]:
```
# Data preparation for char-based model


if USE_CHAR:
    char2idx_dict = {}
    idx2char_dict = {}

    for idx, char in enumerate(ALPHABET):
        char2idx_dict[char] = idx + 1

    idx2char_dict = dict([(i+1, char) for i, char in enumerate(char

    # Get informations about char sequence length
    result = [len(x) for x in x_train_5]
    plt.figure(figsize=(20,5))
    plt.title("Char length")
    plt.hist(result, 200, density=False, range=(0,np.max(result)))
    plt.show()

    print("Text informations:")
    print(f" - max:   {np.max(result)}")
    print(f" - min:   {np.min(result)}")
    print(f" - mean:  {np.mean(result)}")
    print(f" - limit: {CHAR_MAX_LENGTH}")

    char_data = char_vectorizer(x_train_5, CHAR_MAX_LENGTH, char2id
```

In [85]:
```
# Y datatype convert to integer

label_enc=LabelEncoder()

y_train_5= label_enc.fit_transform(y_train_5)
y_test_5= label_enc.transform(y_test_5)
```

In [471]:
```
y_train_5
```

Out[471]:
```
array([2, 2, 4, ..., 0, 0, 0])
```

In [ ]:
```
# CNN parameter

from sklearn.model_selection import train_test_split
from cnn_model import CNN
from Utils import create_glove_embeddings

histories = []

for i in range(RUNS):
    print(f"Running iteration {i+1}/{RUNS}")
    random_state = np.random.randint(1000)

    _X_train, _X_val, _y_train, _y_val = train_test_split(
        word_data,                                  # concluded from
        tf.keras.utils.to_categorical(y_train_5),
        test_size    = VAL_SIZE,
        random_state = random_state
    )
```

```
)

if USE_CHAR:
    _X_train_c, _X_val_c, _, _ = train_test_split(
        char_data,
        tf.keras.utils.to_categorical(y_train_5),
        test_size    = VAL_SIZE,
        random_state = random_state
    )

    _X_train = [_X_train, _X_train_c]
    _X_val   = [_X_val,   _X_val_c]


emb_layer = None

if USE_GLOVE:
    emb_layer = create_glove_embeddings(
        embedding_dim  = EMBEDDING_DIM,
        max_num_words  = MAX_NUM_WORDS,
        max_seq_length = MAX_SEQ_LENGTH,
        tokenizer      = tokenizer
    )

model_CNN= CNN(
    embedding_layer    = emb_layer,
    num_words          = MAX_NUM_WORDS,
    embedding_dim      = EMBEDDING_DIM,
    kernel_sizes       = KERNEL_SIZES,
    feature_maps       = FEATURE_MAPS,
    max_seq_length     = MAX_SEQ_LENGTH,
    use_char           = USE_CHAR,
    char_embedding_dim = CHAR_EMBEDDING_DIM,
    char_max_length    = CHAR_MAX_LENGTH,
    alphabet_size      = ALPHABET_SIZE,
    char_kernel_sizes  = CHAR_KERNEL_SIZES,
    char_feature_maps  = CHAR_FEATURE_MAPS,
    dropout_rate       = DROPOUT_RATE,
    hidden_units       = HIDDEN_UNITS,
    nb_classes         = NB_CLASSES
).build_model()

model_CNN.compile(
    loss      = "categorical_crossentropy",
    optimizer = tf.optimizers.Adam(),
    metrics   = ["accuracy"]
)

history_CNN = model_CNN.fit(
    _X_train, _y_train,
    epochs          = NB_EPOCHS,
    batch_size      = BATCH_SIZE,
    validation_data = (_X_val, _y_val),
    callbacks       = [tf.keras.callbacks.ModelCheckpoint(
        filepath       = f"model-{i+1}.h5",
        monitor        = "val_loss",
        verbose        = 1,
        save_best_only = True,
```

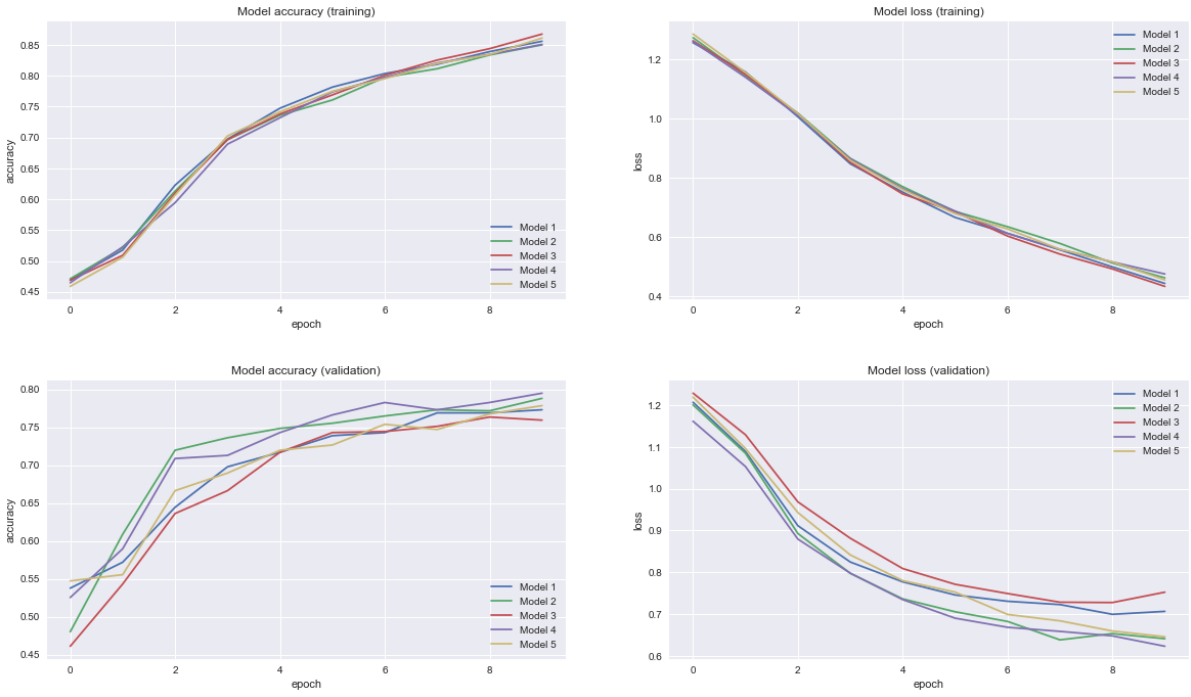
```
            mode            = "min"
        )]
    )

    histories.append(history_CNN.history)
```

In [ ]:

In [789]:
```
# Evaluation

def get_avg(histories, his_key):
    tmp = []
    for history in histories:
        tmp.append(history[his_key][np.argmin(history['val_loss'])]
    return np.mean(tmp)

print(f"Training: \t{get_avg(histories,'loss')} loss / {get_avg(his
print(f"Validation: \t{get_avg(histories,'val_loss')} loss / {get_a
```

```
Training:       0.5007279992103577 loss / 0.8412187457084656 acc
Validation:     0.6669504046440125 loss / 0.7756497979164123 acc
```

In [790]:
```
from Utils import plot_acc_loss

plot_acc_loss('training', histories, 'accuracy', 'loss')
plot_acc_loss('validation', histories, 'val_accuracy', 'val_loss')
```



In [ ]:

In [791]:
```
# Test

sequences_test = tokenizer.texts_to_sequences(x_test_5)
X_test_word    = tf.keras.preprocessing.sequence.pad_sequences(
    sequences_test,
    maxlen  = MAX_SEQ_LENGTH,
    padding = 'post'
)

if USE_CHAR:
    X_test_word = [X_test_word, char_vectorizer(X_test, CHAR_MAX_LE
else:
    X_test_word = X_test_word
```

In [792]:
```
import cnn_model

test_loss = []
test_accs = []

for i in range(0, RUNS):
    cnn_ = tf.keras.models.load_model(f"model-{i+1}.h5")
    score = cnn_.evaluate(X_test_word, tf.keras.utils.to_categorica
    test_loss.append(score[0])
    test_accs.append(score[1])
    print(f"Running test with model {i+1}: {score[0]} loss / {score

print(f"\nAverage loss / accuracy on testset: {np.mean(test_loss)}
print(f"Standard deviation: (+-{np.std(test_loss)}) loss / (+-{np.s
```

```
Running test with model 1: 0.6759737133979797 loss / 0.78160917758
94165 acc
Running test with model 2: 0.6850547790527344 loss / 0.76564496755
59998 acc
Running test with model 3: 0.6506010293960571 loss / 0.78799492120
7428 acc
Running test with model 4: 0.6627236008644104 loss / 0.79310345649
71924 acc
Running test with model 5: 0.6901544332504272 loss / 0.78671777248
38257 acc

Average loss / accuracy on testset: 0.6729015111923218 loss / 0.78
30140590667725 acc
Standard deviation: (+-0.014530691742728462) loss / (+-0.009423178
371694488) acc
```

In [ ]:

In [ ]:
```
"""
Result: use CNN model
input: df_model job backgroud that job description minus skill info
output: job position


"""
```

In [ ]:
```

```

4.4 RCNN with Full Info

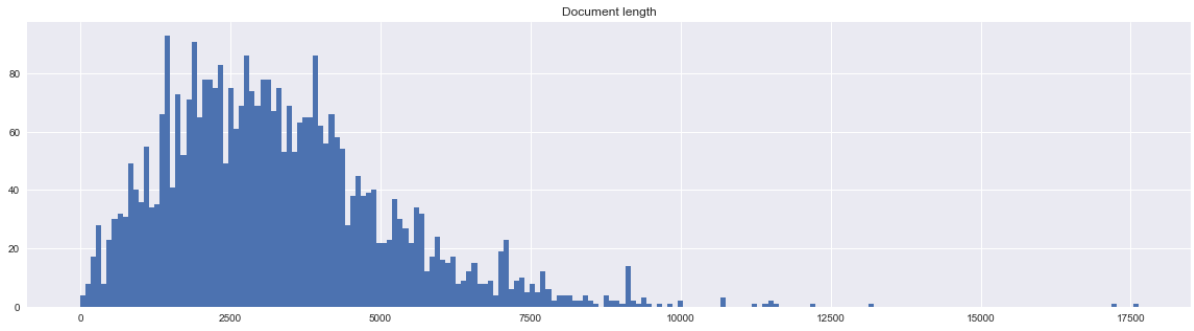
In [73]:
```
#  Dataset  use full job description

from sklearn.model_selection import train_test_split

X_6= df_model.description_clean
Y_6=  df_model.position

x_train_6, x_test_6, y_train_6, y_test_6= train_test_split(X_6,Y_6,
```

In [74]:
```

# WORD-level
MAX_NUM_WORDS  = 15000
EMBEDDING_DIM  = 300
MAX_SEQ_LENGTH = 200
USE_GLOVE      = True
KERNEL_SIZES   = [3,4,5]
FEATURE_MAPS   = [200,200,200]

# CHAR-level
USE_CHAR           = False
ALPHABET           = " abcdefghijklmnopqrstuvwxyz0123456789-,;.!?:'
ALPHABET_SIZE      = len(ALPHABET)
CHAR_EMBEDDING_DIM = 300
CHAR_MAX_LENGTH    = 2500
CHAR_KERNEL_SIZES  = [5,10,20]
CHAR_FEATURE_MAPS  = [200,200,200]

# GENERAL
DROPOUT_RATE = 0.5
HIDDEN_UNITS = 250
NB_CLASSES   = 6

# LEARNING
BATCH_SIZE = 100
NB_EPOCHS  = 10
RUNS       = 5
VAL_SIZE   = 0.2
```

In [75]:
```
# Data: train and test

print(f"Train samples: {len(x_train_6)}")
print(f"Test samples:  {len(x_test_6)}")
```

```
Train samples: 3652
Test samples:  1566
```

In [89]:
```
# Data preparation for word-based model

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
plt.style.use('seaborn')

tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=MAX_NUM
tokenizer.fit_on_texts(x_train_6)

sequences  = tokenizer.texts_to_sequences(x_train_6)
word_index = tokenizer.word_index
result     = [len(x) for x in x_train_6]

# Plot histogram
plt.figure(figsize=(20,5))
plt.title("Document length")
plt.hist(result, 200, density=False, range=(0,np.max(result)))
plt.show()

print("Text informations:")
print(f" - max length:   {np.max(result)}")
print(f" - min length:   {np.min(result)}")
print(f" - mean length:  {np.mean(result)}")
print(f" - limit length: {MAX_SEQ_LENGTH}")

# Padding all sequences to same length of `MAX_SEQ_LENGTH`
word_data = tf.keras.preprocessing.sequence.pad_sequences(
    sequences,
    maxlen  = MAX_SEQ_LENGTH,
    padding = 'post'
)
```




```
Text informations:
 - max length:   17624
 - min length:   41
 - mean length:  3294.764512595838
 - limit length: 200
```

In [90]:
```
# Data preparation for char-based model


if USE_CHAR:
    char2idx_dict = {}
    idx2char_dict = {}

    for idx, char in enumerate(ALPHABET):
        char2idx_dict[char] = idx + 1

    idx2char_dict = dict([(i+1, char) for i, char in enumerate(char

    # Get informations about char sequence length
    result = [len(x) for x in x_train_6]
    plt.figure(figsize=(20,5))
    plt.title("Char length")
    plt.hist(result, 200, density=False, range=(0,np.max(result)))
    plt.show()

    print("Text informations:")
    print(f" - max:   {np.max(result)}")
    print(f" - min:   {np.min(result)}")
    print(f" - mean:  {np.mean(result)}")
    print(f" - limit: {CHAR_MAX_LENGTH}")

    char_data = char_vectorizer(x_train_6, CHAR_MAX_LENGTH, char2id
```

In [91]:
```
# Y datatype convert to integer

from sklearn.preprocessing import LabelEncoder

label_enc=LabelEncoder()

y_train_6= label_enc.fit_transform(y_train_6)
y_test_6= label_enc.transform(y_test_6)
```

In [92]:
```
from cnn_model import CNN
from Utils import create_glove_embeddings

histories_2 = []

for i in range(RUNS):
    print(f"Running iteration {i+1}/{RUNS}")
    random_state = np.random.randint(1000)

    _X_train, _X_val, _y_train, _y_val = train_test_split(
        word_data,                                  # concluded from
        tf.keras.utils.to_categorical(y_train_6),
        test_size    = VAL_SIZE,
        random_state = random_state
    )

    if USE_CHAR:
        _X_train_c, _X_val_c, _, _ = train_test_split(
            char_data,
            tf.keras.utils.to_categorical(y_train_6),
```

```
        test_size    = VAL_SIZE,
        random_state = random_state
    )

    _X_train = [_X_train, _X_train_c]
    _X_val   = [_X_val,   _X_val_c]


emb_layer = None

if USE_GLOVE:
    emb_layer = create_glove_embeddings(
        embedding_dim  = EMBEDDING_DIM,
        max_num_words  = MAX_NUM_WORDS,
        max_seq_length = MAX_SEQ_LENGTH,
        tokenizer      = tokenizer
    )

model_CNN= CNN(
    embedding_layer    = emb_layer,
    num_words          = MAX_NUM_WORDS,
    embedding_dim      = EMBEDDING_DIM,
    kernel_sizes       = KERNEL_SIZES,
    feature_maps       = FEATURE_MAPS,
    max_seq_length     = MAX_SEQ_LENGTH,
    use_char           = USE_CHAR,
    char_embedding_dim = CHAR_EMBEDDING_DIM,
    char_max_length    = CHAR_MAX_LENGTH,
    alphabet_size      = ALPHABET_SIZE,
    char_kernel_sizes  = CHAR_KERNEL_SIZES,
    char_feature_maps  = CHAR_FEATURE_MAPS,
    dropout_rate       = DROPOUT_RATE,
    hidden_units       = HIDDEN_UNITS,
    nb_classes         = NB_CLASSES
).build_model()

model_CNN.compile(
    loss      = "categorical_crossentropy",
    optimizer = tf.optimizers.Adam(),
    metrics   = ["accuracy"]
)

history_CNN = model_CNN.fit(
    _X_train, _y_train,
    epochs          = NB_EPOCHS,
    batch_size      = BATCH_SIZE,
    validation_data = (_X_val, _y_val),
    callbacks       = [tf.keras.callbacks.ModelCheckpoint(
        filepath       = f"model-{i+1}.h5",
        monitor        = "val_loss",
        verbose        = 1,
        save_best_only = True,
        mode           = "min"
    )]
)

histories_2.append(history_CNN.history)
```

```
del-2.h5
30/30 [==============================] - 63s 2s/step - loss: 1.282
4 - accuracy: 0.4574 - val_loss: 1.1642 - val_accuracy: 0.4952
Epoch 2/10
30/30 [==============================] - ETA: 0s - loss: 1.1640 -
accuracy: 0.5046
Epoch 2: val_loss improved from 1.16420 to 1.05636, saving model t
o model-2.h5
30/30 [==============================] - 60s 2s/step - loss: 1.164
0 - accuracy: 0.5046 - val_loss: 1.0564 - val_accuracy: 0.5650
Epoch 3/10
30/30 [==============================] - ETA: 0s - loss: 1.0256 -
accuracy: 0.5984
Epoch 3: val_loss improved from 1.05636 to 0.90578, saving model t
o model-2.h5
30/30 [==============================] - 59s 2s/step - loss: 1.025
6 - accuracy: 0.5984 - val_loss: 0.9058 - val_accuracy: 0.6977
Epoch 4/10
30/30 [==============================] - ETA: 0s - loss: 0.8595 -
accuracy: 0.6946
```

In [93]:

```
# Evaluation

def get_avg(histories, his_key):
    tmp = []
    for history in histories:
        tmp.append(history[his_key][np.argmin(history['val_loss'])]
    return np.mean(tmp)

print(f"Training: \t{get_avg(histories_2,'loss')} loss / {get_avg(h
print(f"Validation: \t{get_avg(histories_2,'val_loss')} loss / {get
```

```
Training:       0.44365962147712706 loss / 0.8596371054649353 acc
Validation:     0.6250372409820557 loss / 0.8060191512107849 acc
```

In [95]:

```
# Test

sequences_test = tokenizer.texts_to_sequences(x_test_6)
X_test_word    = tf.keras.preprocessing.sequence.pad_sequences(
    sequences_test,
    maxlen  = MAX_SEQ_LENGTH,
    padding = 'post'
)

if USE_CHAR:
    X_test_word = [X_test_word, char_vectorizer(X_test, CHAR_MAX_LE
else:
    X_test_word = X_test_word
```

In [96]:
```
import cnn_model

test_loss = []
test_accs = []

for i in range(0, RUNS):
    cnn_ = tf.keras.models.load_model(f"model-{i+1}.h5")
    score = cnn_.evaluate(X_test_word, tf.keras.utils.to_categorica
    test_loss.append(score[0])
    test_accs.append(score[1])
    print(f"Running test with model {i+1}: {score[0]} loss / {score

print(f"\nAverage loss / accuracy on testset: {np.mean(test_loss)}
print(f"Standard deviation: (+-{np.std(test_loss)}) loss / (+-{np.s
```

```
Running test with model 1: 0.6919768452644348 loss / 0.77330780029
29688 acc
Running test with model 2: 0.6992615461349487 loss / 0.77650064229
96521 acc
Running test with model 3: 0.6728969216346741 loss / 0.77458494901
6571 acc
Running test with model 4: 0.6790328621864319 loss / 0.78352493047
71423 acc
Running test with model 5: 0.6795976161956787 loss / 0.78735631704
33044 acc

Average loss / accuracy on testset: 0.6845531582832336 loss / 0.77
90549278259278 acc
Standard deviation: (+-0.009616098177290536) loss / (+-0.005448459
353361795) acc
```

In [ ]:

In [ ]:

5.4: Table 4- Top important and discriminate words of each job position class

In [ ]:
```
''' six sub datasets:

Data Scientist, Data Analyst, Machine Learning Engineer, Data Scien

convert each sub dataset to binary classification

'''
```

In [615]:
```
_Data_Scientist= df_clean.copy()
_Data_Scientist['position'].replace(to_replace="Data Scientist", va
_Data_Scientist['position']= df_Data_Scientist['position'].replace(
```

In [620]:
```
df_Data_Scientist['position'].value_counts()
```

Out[620]:
```
0    3662
1    1556
Name: position, dtype: int64
```

In [625]:
```
X_7= df_Data_Scientist[['description_clean','spacy_skill_set']]
Y_7= df_Data_Scientist['position']
```

In [636]:
```
# randomly select subset of 0 to make a balanced dataset of 0 and 1

class_count_0, class_count_1 = df_Data_Scientist['position'].value_

class_0 = df_Data_Scientist[df_Data_Scientist['position'] == 0]
class_1 = df_Data_Scientist[df_Data_Scientist['position'] == 1]

# print the shape of the class
print('class 0:', class_0.shape)
print('class 1:', class_1.shape)
```

```
class 0: (3662, 24)
class 1: (1556, 24)
```

In [644]:
```
class_0_unsample = class_0.sample(class_count_1)

data_unsample = pd.concat([class_0_unsample, class_1], axis=0)

print("total class of 1 and 0:")
print(data_unsample['position'].value_counts())
```

```
total class of 1 and 0:
0    1556
1    1556
Name: position, dtype: int64
```

In [769]:

```
# Data Scientist
# SPACY-- class_0_unsample.description_clean

import en_core_web_sm
import spacy

nlp = en_core_web_sm.load()

# list to store extracted skill keywords
skill_list_Data_Scientist = []

# feed the entire corpus into batches of 100 samples at a time
for i in range(0,len(df_Data_Scientist), 100):
    # for the last batch
    if i+np.mod(2253,100)==len(class_0_unsample):
        # combine job descriptions of 100 samples into a single str
        text = " ".join(des for des in df_Data_Scientist['Job Descr
    else :
        text = " ".join(des for des in df_Data_Scientist['Job Descr

    # 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_Data_Scientist.append(entity.text)
```

In [770]:
```
from collections import Counter

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

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

Out[770]:
```
[('SQL', 2925),
 ('SAS', 797),
 ('AI', 553),
 ('Hadoop', 549),
 ('Data Analyst', 541),
 ('IBM', 515),
 ('Bachelor', 452),
 ('BI', 430),
 ('ML', 411),
 ('TX', 402),
 ('Data Science', 391),
 ('Big Data', 362),
 ('ETL', 299),
 ('NLP', 261),
 ('Data Scientist', 254),
 ('PowerPoint', 247),
 ('Computer Science', 242),
 ('GSK', 238),
 ('SQL Server', 229),
```

In [ ]:
```

```

In [ ]:
```
all_class = {'Data Scientist': 0, 'Data Analyst': 1, 'Machine Learn
            'Data Science Manager': 3, 'Data architect': 4, 'Reseach

current_class = 'Data Scientist'
```

In [ ]:
```

```

In [97]:
```
# Data Analyst: 1.

df_Data_Analyst= df_clean.copy()
df_Data_Analyst['position'].replace(to_replace="Data Analyst", valu
df_Data_Analyst['position']= df_Data_Analyst['position'].replace(to
```

In [99]:
```
df_Data_Analyst_1= df_Data_Analyst[df_Data_Analyst['position']== 1]
```

In [102]:

```
# SPACY-- df_Data_Analyst_1.description_clean

import en_core_web_sm
import spacy

nlp = en_core_web_sm.load()

# list to store extracted skill keywords
skill_list_Data_Analyst = []

# feed the entire corpus into batches of 100 samples at a time
for i in range(0,len(df_Data_Analyst_1), 100):
    # for the last batch
    if i+np.mod(2536,100)==len(df_Data_Analyst_1):
        # combine job descriptions of 100 samples into a single str
        text = " ".join(des for des in df_Data_Analyst_1['Job Descr
    else :
        text = " ".join(des for des in df_Data_Analyst_1['Job Descr

    # 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_Data_Analyst.append(entity.text)
```

In [103]:
```
from collections import Counter

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

# print the top 100 named entities
word_count.most_common(20)
```

Out[103]:
```
[('SQL', 1742),
 ('SAS', 532),
 ('Data Analyst', 532),
 ('BI', 300),
 ('Bachelor', 245),
 ('Microsoft Office', 174),
 ('PowerPoint', 173),
 ('The Data Analyst', 169),
 ('ETL', 164),
 ('TX', 156),
 ('SQL Server', 150),
 ('Finance', 132),
 ('Microsoft Excel', 130),
 ('Business Objects', 130),
 ('Power BI', 126),
 ('SPSS', 124),
 ('Looker', 114),
 ('Business Intelligence', 113),
 ('Healthcare', 107),
 ('Microsoft', 106)]
```

In [ ]:
```

```

In [130]:
```
# Machine Learning Engineer: 2.

df_ml_Engineer= df_clean.copy()
df_ml_Engineer['position'].replace(to_replace="Machine Learning Eng
df_ml_Engineer['position']= df_ml_Engineer['position'].replace(to_r
```

In [131]:
```
df_ml_Engineer_1= df_ml_Engineer[df_ml_Engineer['position']== 1]
```

In [132]:

```
# SPACY-- df_ml_Engineer_1.description_clean

import en_core_web_sm
import spacy

nlp = en_core_web_sm.load()

# list to store extracted skill keywords
skill_list_ml_Engineer = []

# feed the entire corpus into batches of 100 samples at a time
for i in range(0,len(df_ml_Engineer_1), 100):
    # for the last batch
    if i+np.mod(842,100)==len(df_ml_Engineer_1):
        # combine job descriptions of 100 samples into a single str
        text = " ".join(des for des in df_ml_Engineer_1['Job Descri
    else :
        text = " ".join(des for des in df_ml_Engineer_1['Job Descri

    # 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_ml_Engineer.append(entity.text)
```

In [108]:
```
# count how many times each entity appears in the list
word_count = Counter(skill_list_ml_Engineer)

# print the top 100 named entities
word_count.most_common(20)
```

Out[108]:
```
[('SQL', 540),
 ('Hadoop', 302),
 ('Big Data', 198),
 ('Data Engineer', 122),
 ('AI', 110),
 ('ETL', 100),
 ('ML', 96),
 ('Computer Science', 93),
 ('BI', 88),
 ('Bachelor', 75),
 ('TX', 74),
 ('Data Engineering', 72),
 ('DevOps', 65),
 ('Data Science', 57),
 ('Apple', 56),
 ('ELT', 50),
 ('Data', 49),
 ('The Data Engineer', 47),
 ('SQL Server', 46),
 ('SSIS', 44)]
```

In [ ]:
```
```

In [109]:
```
# Data Science Manager

df_Manager= df_clean.copy()
df_Manager['position'].replace(to_replace="Data Science Manager", v
df_Manager['position']= df_Manager['position'].replace(to_replace=r
```

In [111]:
```
df_Manager_1= df_Manager[df_Manager['position']== 1]
```

In [113]:

```
# SPACY-- df_Data_Analyst_1.description_clean

import en_core_web_sm
import spacy

nlp = en_core_web_sm.load()

# list to store extracted skill keywords
skill_list_Manager = []

# feed the entire corpus into batches of 50 samples at a time
for i in range(0,len(df_Manager_1), 50):
    # for the last batch
    if i+np.mod(230,50)==len(df_Manager_1):
        # combine job descriptions of 50 samples into a single stri
        text = " ".join(des for des in df_Manager_1['Job Descriptio
    else :
        text = " ".join(des for des in df_Manager_1['Job Descriptio

    # 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_Manager.append(entity.text)
```

In [114]:
```
# count how many times each entity appears in the list
word_count = Counter(skill_list_Manager)

# print the top 100 named entities
word_count.most_common(20)
```

Out[114]:
```
[('Data Science', 111),
 ('SQL', 97),
 ('IBM', 73),
 ('Varsity Tutors', 42),
 ('SAS', 38),
 ('AI', 37),
 ('SEI Consultants', 36),
 ('Wells Fargo', 31),
 ('Microsoft', 28),
 ('SSRS', 24),
 ('TX', 24),
 ('Honey', 22),
 ('Big Data', 21),
 ('Hadoop', 21),
 ('Bachelor', 19),
 ('IBM Services', 18),
 ('SSIS', 18),
 ('NCS', 18),
 ('BI', 17),
 ('NLP', 16)]
```

In [ ]:
```
```

In [115]:
```
 Data architect: 4,        'Reseacher'

f_architect= df_clean.copy()
f_architect['position'].replace(to_replace="Data architect", value=
f_architect['position']= df_architect['position'].replace(to_replace
```

In [116]:
```
df_architect_1= df_architect[df_architect['position']== 1]
```

In [118]:

```
# SPACY-- df_Data_Analyst_1.description_clean

import en_core_web_sm
import spacy

nlp = en_core_web_sm.load()

# list to store extracted skill keywords
skill_list_architect = []

# feed the entire corpus into batches of 5 samples at a time
for i in range(0,len(df_architect_1), 5):
    # for the last batch
    if i+np.mod(20,5)==len(df_architect_1):
        # combine job descriptions of 5 samples into a single strin
        text = " ".join(des for des in df_architect_1['Job Descript
    else :
        text = " ".join(des for des in df_architect_1['Job Descript

    # 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_architect.append(entity.text)
```

In [119]:
```
# count how many times each entity appears in the list
word_count = Counter(skill_list_architect)

# print the top 100 named entities
word_count.most_common(20)
```

Out[119]:
```
[('SQL', 35),
 ('SSIS', 24),
 ('the Business Analysts', 23),
 ('Business Systems Management', 12),
 ('Hadoop', 12),
 ('Project Management, Budget and Human Resource Management, Busin
ess Intelligence / Data',
  12),
 ('ERWin', 12),
 ('Informatica', 12),
 ('PL-SQL', 12),
 ('Blackbaud', 10),
 ('Master Data Management', 9),
 ('ELT', 9),
 ('Data Governance', 9),
 ('On-Board', 8),
 ('Data Architect', 8),
 ('Facilitate', 8),
 ('DBMS', 8),
 ('Database Management Systems', 8),
 ('Computer Science, Information Systems', 7),
 ('EIM', 6)]
```

In [ ]:
```

```

In [120]:
```
#'Reseacher'

df_Reseacher= df_clean.copy()
df_Reseacher['position'].replace(to_replace="Reseacher", value= 1,
df_Reseacher['position']= df_Reseacher['position'].replace(to_repla
```

In [121]:
```
df_Reseacher_1= df_Reseacher[df_Reseacher['position']== 1]
```

In [124]:

```
# SPACY-- df_Data_Analyst_1.description_clean

import en_core_web_sm
import spacy

nlp = en_core_web_sm.load()

# list to store extracted skill keywords
skill_list_Reseacher = []

# feed the entire corpus into batches of 5 samples at a time
for i in range(0,len(df_Reseacher_1), 5):
    # for the last batch
    if i+np.mod(34,5)==len(df_Reseacher_1):
        # combine job descriptions of 5 samples into a single strin
        text = " ".join(des for des in df_Reseacher_1['Job Descript
    else :
        text = " ".join(des for des in df_Reseacher_1['Job Descript

    # 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_Reseacher.append(entity.text)
```

In [125]:
```
# count how many times each entity appears in the list
word_count = Counter(skill_list_Reseacher)

# print the top 100 named entities
word_count.most_common(20)
```

Out[125]:
```
[('DRW', 53),
 ('SIG', 40),
 ('Quantitative Researcher - Concentration Risk Analytics', 36),
 ('Citi', 36),
 ('Franklin', 21),
 ('Wholesale Credit', 21),
 ('the California Privacy Notice', 19),
 ('Bachelor', 16),
 ('Quantitative Researcher', 15),
 ('https://drw.com/privacy-notice.California', 14),
 ('Franklin University', 14),
 ('Social Studies', 14),
 ('Economics', 14),
 ('Citigroup Inc.', 12),
 ('Corporate Risk Reporting', 12),
 ('the Quantitative Risk', 12),
 ('EEO', 12),
 ('the Concentration Risk Analytics', 12),
 ('Macro/Cross-Asset Trading', 12),
 ('Enterprise Concentration Risk Management', 12)]
```

In [ ]: