data-collection-notebook

Data Scientists Salary Estimator Project

(Source: September 2020 Glassdoor Estimate)

- Created a tool to estimate data scientists salaries with MAE ~$16K to help future data scientists estimate salary based on job location, company size, company rating, job title, seniority etc.
- Scraped 1000 jobs posted on Glassdoor.com
- Engineered features from 1000 job descriptions to quantify the value of having hottest data science skills including python, excel, aws, spark, tensorflow.
- Optimized Linear, Lasso, Ridge and Random Forest Regressors using GridsearchCV

Resources Used

Python Version: 3.8

Packages: pandas, numpy, sklearn, matplotlib, seaborn, selenium, pickle

Scraper Github: https://github.com/arapfaik/scraping-glassdoor-selenium (https://github.com/arapfaik/scraping-glassdoor-selenium)

Scraper Article: https://towardsdatascience.com/selenium-tutorial-scraping-glassdoor-com-in-10-minutes-3d0915c6d905 (https://towardsdatascience.com/selenium-tutorial-scraping-glassdoor-com-in-10-minutes-3d0915c6d905)

Data Collection

Glassdoor Scraper Author: Ömer Sakarya

Glassdoor Scraper Github: https://github.com/arapfaik/scraping-glassdoor-selenium (https://github.com/arapfaik/scraping-glassdoor-selenium)

Glassdoor Scraper Article: https://towardsdatascience.com/selenium-tutorial-scraping-glassdoor-com-in-10-minutes-3d0915c6d905 (https://towardsdatascience.com/selenium-tutorial-scraping-glassdoor-com-in-10-minutes-3d0915c6d905)

Import Packages

In [4]:
```
import glassdoor_scraping as gs
import pandas as pd
```

Data Explanation

I used the glassdoor scraper to scrape 1000 jobs posted on Glassdoor.com. Each entry contains job information including:

- Job title
- Salary Estimate (provided by Glassdoor estimate)
- Job Description
- Rating (Company rating)
- Company
- Location
- Company Headquarters
- Company Size
- Company Founded Date
- Type of Ownership
- Industry
- Sector
- Revenue
- Competitors

In [5]:
```
path = "C:/Users/51973/Desktop/projects/Data-Scientist-Salary-Project/chromedr
iver"
```

In [6]:
```
df_rawdata = gs.get_jobs("data scientist", 1000, False, path, 15)
```

```
x out worked
Progress: 0/1000
Progress: 1/1000
Progress: 2/1000
Progress: 3/1000
Progress: 4/1000
Progress: 5/1000
Progress: 6/1000
Progress: 7/1000
Progress: 8/1000
Progress: 9/1000
Progress: 10/1000
Progress: 11/1000
Progress: 12/1000
Progress: 13/1000
Progress: 14/1000
Progress: 15/1000
Progress: 16/1000
Progress: 17/1000
Progress: 18/1000
Progress: 19/1000
Progress: 20/1000
Progress: 21/1000
Progress: 22/1000
Progress: 23/1000
Progress: 24/1000
Progress: 25/1000
Progress: 26/1000
Progress: 27/1000
Progress: 28/1000
Progress: 29/1000
 x out failed
Progress: 30/1000
Progress: 31/1000
Progress: 32/1000
Progress: 33/1000
Progress: 34/1000
Progress: 35/1000
Progress: 36/1000
Progress: 37/1000
Progress: 38/1000
Progress: 39/1000
Progress: 40/1000
Progress: 41/1000
Progress: 42/1000
Progress: 43/1000
Progress: 44/1000
Progress: 45/1000
Progress: 46/1000
Progress: 47/1000
Progress: 48/1000
Progress: 49/1000
Progress: 50/1000
Progress: 51/1000
Progress: 52/1000
Progress: 53/1000
Progress: 54/1000
```

```
Progress: 55/1000
Progress: 56/1000
Progress: 57/1000
Progress: 58/1000
Progress: 59/1000
Progress: 60/1000
Progress: 61/1000
 x out failed
Progress: 62/1000
Progress: 63/1000
Progress: 64/1000
Progress: 65/1000
Progress: 66/1000
Progress: 67/1000
Progress: 68/1000
Progress: 69/1000
Progress: 70/1000
Progress: 71/1000
Progress: 72/1000
Progress: 73/1000
Progress: 74/1000
Progress: 75/1000
Progress: 76/1000
Progress: 77/1000
Progress: 78/1000
Progress: 79/1000
Progress: 80/1000
Progress: 81/1000
Progress: 82/1000
Progress: 83/1000
Progress: 84/1000
Progress: 85/1000
Progress: 86/1000
Progress: 87/1000
Progress: 88/1000
Progress: 89/1000
Progress: 90/1000
Progress: 91/1000
Progress: 92/1000
Progress: 93/1000
 x out failed
Progress: 94/1000
Progress: 95/1000
Progress: 96/1000
Progress: 97/1000
Progress: 98/1000
Progress: 99/1000
Progress: 100/1000
Progress: 101/1000
Progress: 102/1000
Progress: 103/1000
Progress: 104/1000
Progress: 105/1000
Progress: 106/1000
Progress: 107/1000
Progress: 108/1000
Progress: 109/1000
```

```
Progress: 938/1000
Progress: 939/1000
Progress: 940/1000
Progress: 941/1000
Progress: 942/1000
Progress: 943/1000
Progress: 944/1000
Progress: 945/1000
Progress: 946/1000
Progress: 947/1000
Progress: 948/1000
Progress: 949/1000
Progress: 950/1000
Progress: 951/1000
 x out failed
Progress: 952/1000
Progress: 953/1000
Progress: 954/1000
Progress: 955/1000
Progress: 956/1000
Progress: 957/1000
Progress: 958/1000
Progress: 959/1000
Progress: 960/1000
Progress: 961/1000
Progress: 962/1000
Progress: 963/1000
Progress: 964/1000
Progress: 965/1000
Progress: 966/1000
Progress: 967/1000
Progress: 968/1000
Progress: 969/1000
Progress: 970/1000
Progress: 971/1000
Progress: 972/1000
Progress: 973/1000
Progress: 974/1000
Progress: 975/1000
Progress: 976/1000
Progress: 977/1000
Progress: 978/1000
Progress: 979/1000
Progress: 980/1000
Progress: 981/1000
 x out failed
Progress: 982/1000
Progress: 983/1000
Progress: 984/1000
Progress: 985/1000
Progress: 986/1000
Progress: 987/1000
Progress: 988/1000
Progress: 989/1000
Progress: 990/1000
Progress: 991/1000
Progress: 992/1000
```

```
Progress: 993/1000
Progress: 994/1000
Progress: 995/1000
Progress: 996/1000
Progress: 997/1000
Progress: 998/1000
Progress: 999/1000
Progress: 1000/1000
```

Export to a CSV file

In [10]:
```
df_rawdata.to_csv("data-scientist-salary-data.csv", index= False)
```

data-cleaning-notebook

Data Wrangling

I cleanned and reorganized the raw data collected from Glassdoor.com for data science purpose. To prepare for model building, I list the data wrangling plan below:

- Salary parsing

  The "Salary Estimate" was retrived as object (e.g. $78K-$133K (Glassdoor est.)). I removed "$", "K", "-" and "(Glassdoor est.)" and only left numerical values. The salary information will be represented by "max_salary", "min_salary" and "avg_salary".
- Company name parsing

  The "Company" was collected as "*company name company rating*" format (e.g. Amazon 4.0). I removed the rating element from the column which makes the Company name text-only.
- Location parsing

  The "Location" column was collcted as "*city name, state abbreviation*" format (e.g. Chicago, IL). For data science purpose, I decided to only use state information and removed the city information. Including the city information in the models would potentially decrease the efficiency. Using only state information is much more reasonable and effective.
- Age of Company

  The "Founded" column has information about when the company was founded. Instead of using the specific year, I transformed that information into the age of the company.
- Job description parsing

  The "Job description" column contains text information. However, it was very long. Since the goal of this project is to estimate salary, I only extracted useful information from the column. According to *"14 most used data science tools for 2019" (https://data-flair.training/blogs/data-science-tools/ (https://data-flair.training/blogs/data-science-tools/))*, python, r studio, spark, aws, excel, sas, matlab, tableau, tensorflow are widely used by data scientists. Thus, I am interested in how many companies would include those tools in their job description pages and what the correlation between having experiences with these tools and potentially earning a higher salary.

Import packages

In [3]:
```
import pandas as pd
import numpy as np
```

In [4]:
```python
df = pd.read_csv("data-scientist-salary-data.csv")
# Check NULL values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Job Title          1000 non-null   object 
 1   Salary Estimate    1000 non-null   object 
 2   Job Description    1000 non-null   object 
 3   Rating             1000 non-null   float64
 4   Company Name       1000 non-null   object 
 5   Location           1000 non-null   object 
 6   Headquarters       1000 non-null   int64  
 7   Size               1000 non-null   object 
 8   Founded            1000 non-null   int64  
 9   Type of ownership  1000 non-null   object 
 10  Industry           1000 non-null   object 
 11  Sector             1000 non-null   object 
 12  Revenue            1000 non-null   object 
 13  Competitors        1000 non-null   int64  
dtypes: float64(1), int64(3), object(10)
memory usage: 109.5+ KB
```

Salary parsing

The "Salary Estimate" was retrived as object (e.g. $78K -$133K (Glassdoor est.)). I removed "$", "K", "-" and " (Glassdoor est.)" and only left numerical values. The salary information will be represented by "max_salary", "min_salary" and "avg_salary".

In [5]:
```python
# remove "$", "K", "-" and "(Glassdoor est.)" and only left numerical values.
salary = df["Salary Estimate"].apply(lambda x: x.split("(")[0])
salary_minusdollorandk = salary.apply(lambda x: x.replace("K", "").replace("$"
, ""))
```

In [6]:
```python
# create "min_salary" and "max_salary"
df["min_salary"] = salary_minusdollorandk.apply(lambda x: x.split("-")[0])
df["max_salary"] = salary_minusdollorandk.apply(lambda x: x.split("-")[1])
```

In [9]:
```python
# convert into int64 format
df = df.astype({
    "min_salary":"int64",
    "max_salary":"int64"
})
```

In [10]:
```python
# create "avg_salary"
df["avg_salary"] = (df["min_salary"] + df["max_salary"])/2
```

Company name parsing

The "Company" was collected as "company name company rating" format (e.g. Amazon 4.0). I removed the rating element from the column which makes the Company name text-only.

In [11]:
```
# remove rating from company name column
df["company_text"] = df.apply(lambda x: x["Company Name"] if x["Rating"] < 0 e
lse x["Company Name"][:-4], axis= 1)
```

In [12]:
```
df["company_text"].value_counts()
```

Out[12]:
```
Amazon                    22
AstraZeneca               17
MITRE                     14
Pfizer                    11
Ascension                 10
                          ..
Nextdoor                   1
Chan Zuckerberg Biohub     1
Ecolab                     1
Entefy                     1
OneTrust                   1
Name: company_text, Length: 488, dtype: int64
```

Location parsing

The "Location" column was collcted as "city name, state abbreviation" format (e.g. Chicago, IL). For data science purpose, I decided to only use state information and removed the city information. Including the city information in the models would potentially decrease the efficiency. Using only state information is much more reasonable and effective.

In [14]:
```
# remove city info from the column
df["job_state"] = df["Location"].apply(lambda x: x.split(",")[1].strip() if
"," in x else x)
df["job_state"].value_counts()
# Some location cells are extracted in different formats (e.g. Virginia, Unite
d States...)
```

Out[14]:
```
CA               254
VA               130
NY                89
MA                83
MD                53
DC                45
IL                39
Remote            33
TX                30
United States     27
WA                24
FL                24
NJ                23
PA                16
MO                12
NC                12
CT                10
OH                10
Virginia           8
WI                 7
OR                 6
CO                 6
MI                 6
UT                 5
SC                 5
GA                 5
KS                 4
Massachusetts      4
AL                 4
ID                 3
IN                 3
Utah               3
TN                 3
DE                 3
Ohio               1
HI                 1
New Jersey         1
NM                 1
AZ                 1
MN                 1
AR                 1
Maryland           1
California         1
WY                 1
KY                 1
Name: job_state, dtype: int64
```

In [15]:
```
# replace those values with standard state abbreviation
df["job_state"].replace({
    "United States":"US",
    "Virginia":"VA",
    "Massachusetts":"MA",
    "Utah":"UT",
    "New Jersey":"NJ",
    "Maryland":"MD",
    "Ohio":"OH",
    "California":"CA"
}, inplace= True)
```

In [17]:
```
# check
df["job_state"].value_counts()
```

Out[17]:
```
CA        255
VA        138
NY         89
MA         87
MD         54
DC         45
IL         39
Remote     33
TX         30
US         27
FL         24
NJ         24
WA         24
PA         16
NC         12
MO         12
OH         11
CT         10
UT          8
WI          7
CO          6
OR          6
MI          6
GA          5
SC          5
AL          4
KS          4
IN          3
ID          3
TN          3
DE          3
AZ          1
WY          1
AR          1
MN          1
NM          1
HI          1
KY          1
Name: job_state, dtype: int64
```

In [18]:
```
# The "Headquarters" column is "-1" for all jobs. I guess Glassdoor.com made s
ome changes and the scraper did not gather the information. So, I decided to d
rop the column
df.drop("Headquarters", axis= 1, inplace= True)
```

Age of Company

The "Founded" column has information about when the company was founded. Instead of using the specific year, I transformed that information into the age of the company.

In [20]:
```
df["age"] = df["Founded"].apply(lambda x: x if x < 0 else 2020 - x)
```

In [22]:
```
# -1 means missing value, and we can see 143 companies did not report such inf
ormation
df["age"].value_counts()
```

Out[22]:
```
-1     143
 8      39
 26     39
 9      36
 10     34
      ...
 80      1
 83      1
 0       1
 61      1
 63      1
Name: age, Length: 110, dtype: int64
```

Job description parsing

The "Job description" column contains text information. However, it was very long. Since the goal of this project is to estimate salary, I only extracted useful information from the column. According to "14 most used data science tools for 2019" (https://data-flair.training/blogs/data-science-tools/ (https://data-flair.training/blogs/data-science-tools/)), python, r studio, spark, aws, excel, sas, matlab, tableau, tensorflow are widely used by data scientists. Thus, I am interested in how many companies would include those tools in their job description pages and what the correlation between having experiences with these tools and potentially earning a higher salary.

In [24]:
```
# create dummy variables which indicate whether a certain tool appeared in the
job description

df["python_y/n"] = df["Job Description"].apply(lambda x: 1 if "python" in x.lo
wer() else 0)
df["r_y/n"] = df["Job Description"].apply(lambda x: 1 if "r studio" in x.lower
() or "r-studio" in x.lower() else 0)
df["spark_y/n"] = df["Job Description"].apply(lambda x: 1 if "spark" in x.lowe
r() else 0)
df["aws_y/n"] = df["Job Description"].apply(lambda x: 1 if "aws" in x.lower()
else 0)
df["excel_y/n"] = df["Job Description"].apply(lambda x: 1 if "excel" in x.lowe
r() else 0)
df["sas_y/n"] = df["Job Description"].apply(lambda x: 1 if "sas" in x.lower()
else 0)
df["matlab_y/n"] = df["Job Description"].apply(lambda x: 1 if "matlab" in x.lo
wer() else 0)
df["tableau_y/n"] = df["Job Description"].apply(lambda x: 1 if "tableau" in x.
lower() else 0)
df["tensorflow_y/n"] = df["Job Description"].apply(lambda x: 1 if "tensorflow"
in x.lower() else 0)
```

Export the cleaned csv

In [17]:
```
df.to_csv("data-scientist-salary-cleaned.csv", index= False)
```

exploratory data analysis

Exploratory Data Analysis

I decided to dive deep into the data before building models. My EDA has information about:

Import packages

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud, STOPWORDS
```

In [2]:
```
df = pd.read_csv("data-scientist-salary-cleaned.csv")
```

Simplify job title

I found the job title is not very informative. Thus, I decided to parse it into job title and seniority, which I think are highly relevant to data scientists salary.

In [4]:
```
# Two functions for parsing the job title: title_simplifier and seniority

def title_simplifier(title):
    if "data scientist" in title.lower() or "scientist" in title.lower():
        return "data scientist"
    elif "data engineer" in title.lower():
        return "data engineer"
    elif "analyst" in title.lower():
        return "data analyst"
    elif "machine learning" in title.lower():
        return "machine learning engineer"
    elif "manager" in title.lower():
        return "manager"
    elif "director" in title.lower():
        return "director"
    else:
        return "na"

def seniority(title):
    if "sr" in title.lower() or "senior" in title.lower() or "lead" in title.l
ower() or "principal" in title.lower() or "sr." in title.lower():
        return "senior"
    elif "jr" in title.lower() or "jr." in title.lower() or "junior" in title.
lower() or "associate" in title.lower():
        return "junior"
    else:
        return "na"
```

In [5]:
```
# create simplified job title column
df["job_simplified"] = df["Job Title"].apply(title_simplifier)
```

In [8]:
```
# 667 out of 1000 jobs are data scientist related and 136 out of 1000 are data
analyst related.
df["job_simplified"].value_counts(ascending= False)
```

Out[8]:
```
data scientist               667
data analyst                 136
data engineer                 71
na                            67
machine learning engineer     46
manager                       11
director                       2
Name: job_simplified, dtype: int64
```

In [9]:
```
# create seniority column
df["seniority"] = df["Job Title"].apply(seniority)
```

In [11]:
```
# 814 out of 1000 jobs did not specify seniority
df["seniority"].value_counts(ascending= False)
```

Out[11]:
```
na        814
senior    176
junior     10
Name: seniority, dtype: int64
```

Job description length

The length of job description may be an interesting thing to look at. I personally assume that the longer the description the higher the salary. I would like to see the correlation between these two so I make the "description length" column.

In [12]:
```
df["description_length"] = df["Job Description"].apply(lambda x: len(x))
```

Analysis

I made some graphs and tables to see if I can find anything interesting!

In [10]:
```
df.describe()
```

Out[10]:

	Rating	Founded	Competitors	min_salary	max_salary	avg_salary	a
count	1000.000000	1000.000000	1000.0	1000.000000	1000.000000	1000.000000	1000.000
mean	3.655000	1698.493000	-1.0	86.142000	135.506000	110.824000	32.361
std	1.134541	695.722682	0.0	24.416265	32.365313	27.533818	42.336
min	-1.000000	-1.000000	-1.0	38.000000	74.000000	56.000000	-1.000
25%	3.400000	1939.000000	-1.0	65.000000	112.000000	88.500000	7.000
50%	3.900000	1994.000000	-1.0	86.000000	128.000000	107.500000	18.000
75%	4.200000	2009.000000	-1.0	105.000000	160.000000	130.500000	44.000
max	5.000000	2020.000000	-1.0	135.000000	215.000000	175.000000	236.000

In [11]:
```
df.columns
```

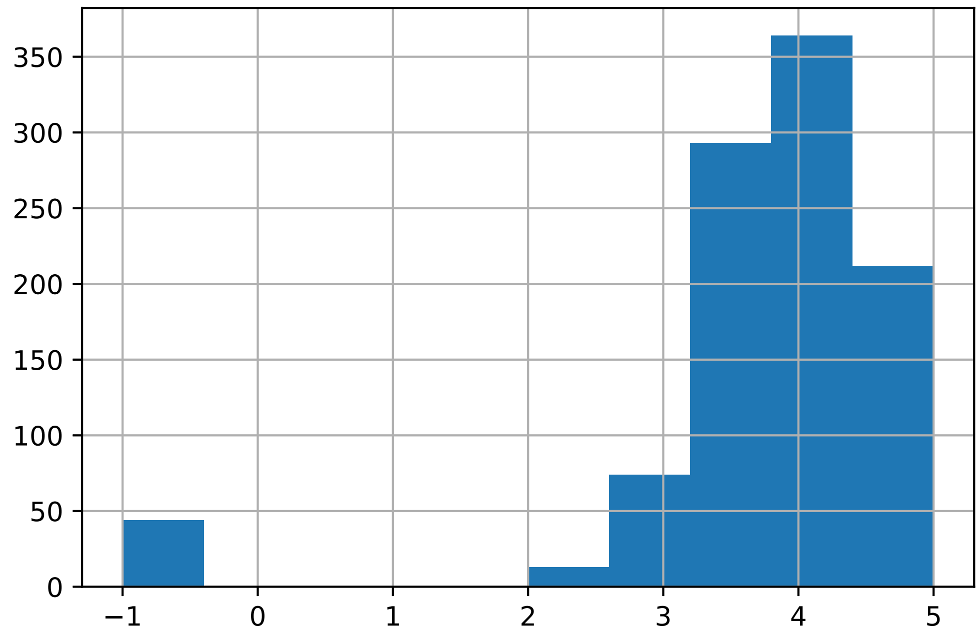
Out[11]:
```
Index(['Job Title', 'Salary Estimate', 'Job Description', 'Rating',
       'Company Name', 'Location', 'Size', 'Founded', 'Type of ownership',
       'Industry', 'Sector', 'Revenue', 'Competitors', 'min_salary',
       'max_salary', 'avg_salary', 'company_text', 'job_state', 'age',
       'python_y/n', 'r_y/n', 'spark_y/n', 'aws_y/n', 'excel_y/n', 'sas_y/n',
       'matlab_y/n', 'tableau_y/n', 'tensorflow_y/n', 'job_simplified',
       'seniority', 'description_length'],
      dtype='object')
```

Distribution

I made some histograms to visulize the distribution of some solumns.

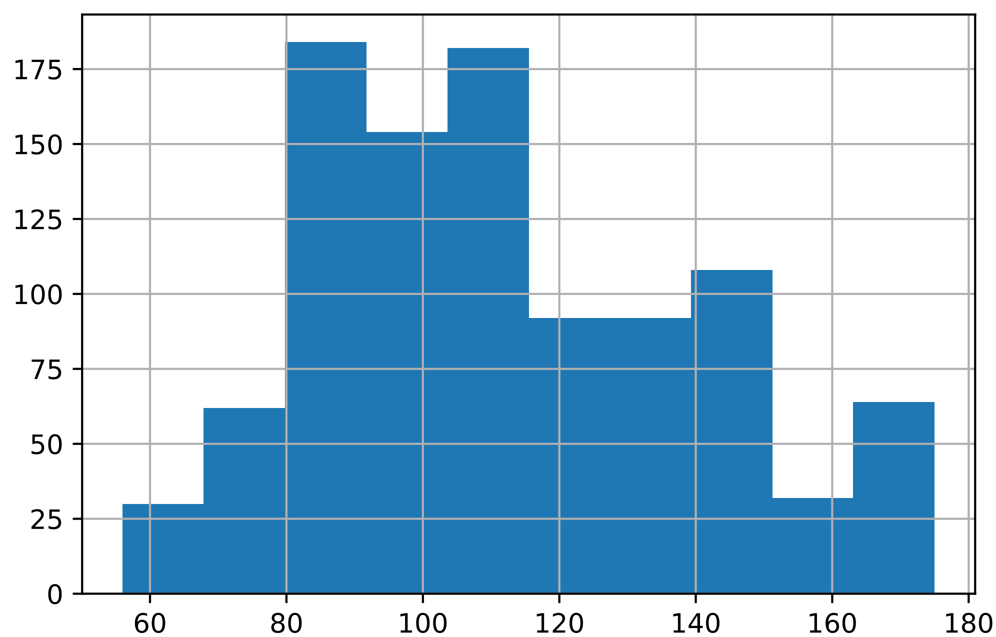
In [17]:
```
# it approximates normal
df.Rating.hist()
```

Out[17]: <AxesSubplot:>



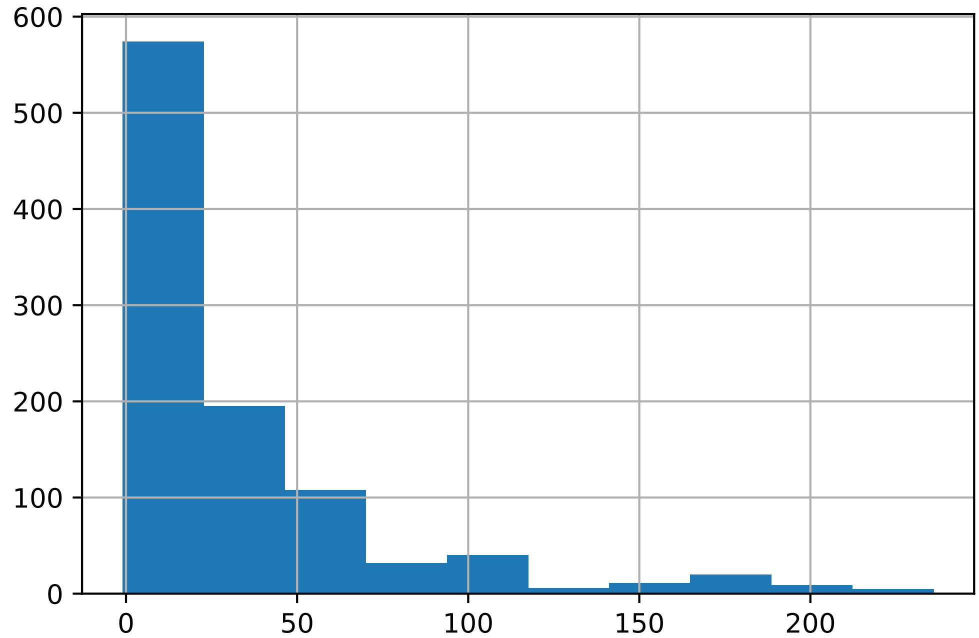
In [14]:
```
# it approximates normal
df.avg_salary.hist()
```

Out[14]: <AxesSubplot:>



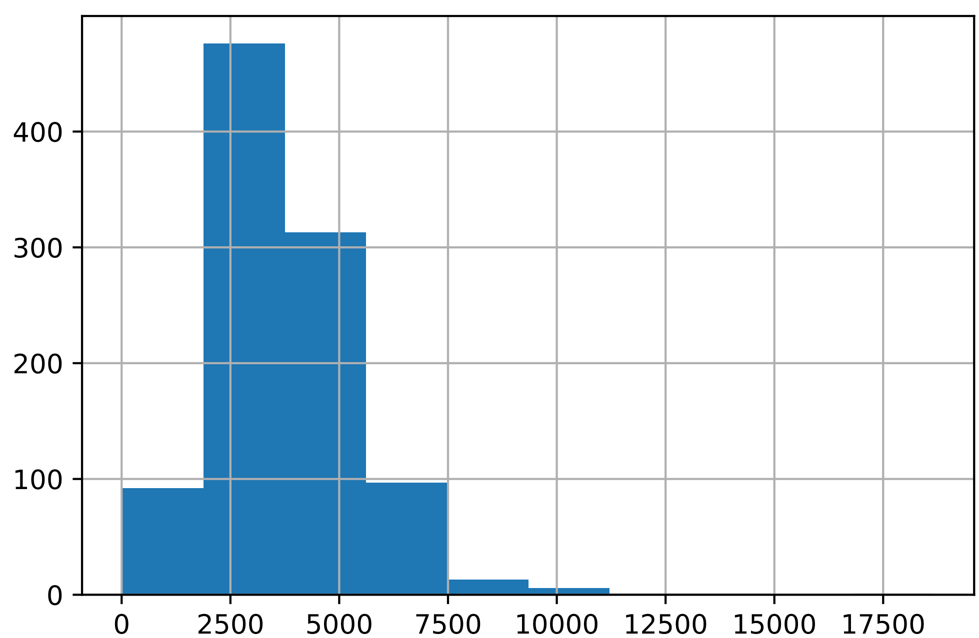
In [15]:
```
# it is skewed. As we can see, younger companies are hiring aggressively.
df.age.hist()
```

Out[15]: <AxesSubplot:>



In [16]:
```
# it approximates normal
df.description_length.hist()
```

Out[16]: <AxesSubplot:>



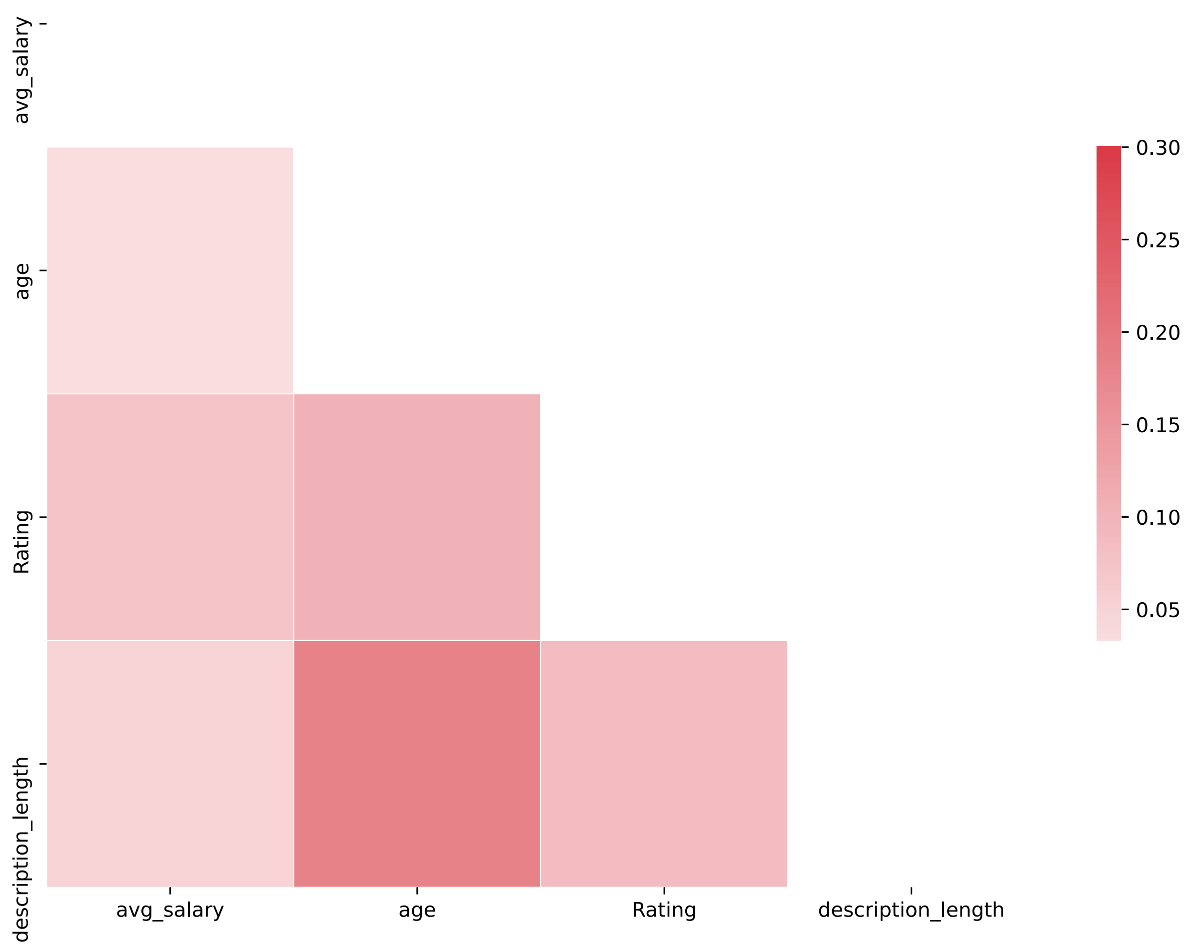
Correlation between age/rating/description_length and average salary

Would there be any correlation between salary and age of the company/rating/description length? I made a correlation graph to visualize the relationships.

In [18]:
```
corr = df[["avg_salary", "age", "Rating", "description_length"]].corr()
```

In [19]:
```
mask = np.triu(np.ones_like(corr, dtype=np.bool))
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

Out[19]: <AxesSubplot:>



Surprisingly, the correlations are weak. Among age, rating, and description_length, age of the company seems to be correlated with salary. So, we may assume that established companies usually offer higher salaries for data scientists.

Histograms about some categorical data

Let's see some interesting graphs that can answer following questions:

- Which states have a high demand for data scientists?
- Company of which size/industry/type of ownership hire more data scientists?
- How many jobs require/prefer applicants to have python/r/spark/aws/excel/sas/matlab/tableau/tensorflow skills?
- What are the demands for data scientists, data analysts and others?

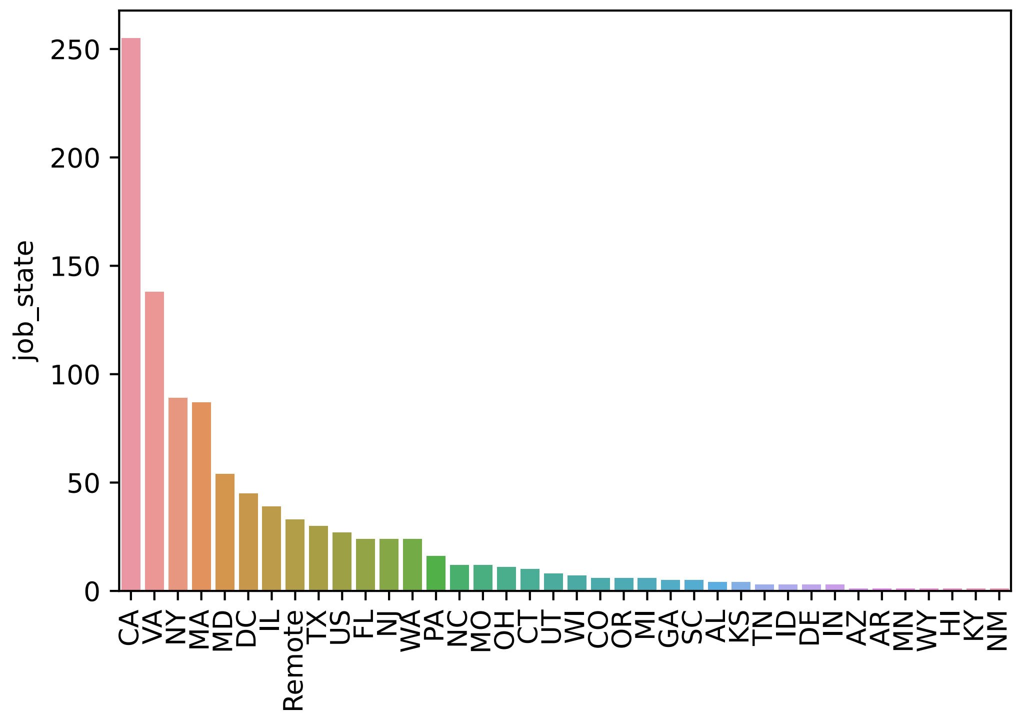
In [19]:
```
df.columns
```

Out[19]:
```
Index(['Job Title', 'Salary Estimate', 'Job Description', 'Rating',
       'Company Name', 'Location', 'Size', 'Founded', 'Type of ownership',
       'Industry', 'Sector', 'Revenue', 'Competitors', 'min_salary',
       'max_salary', 'avg_salary', 'company_text', 'job_state', 'age',
       'python_y/n', 'r_y/n', 'spark_y/n', 'aws_y/n', 'excel_y/n', 'sas_y/n',
       'matlab_y/n', 'tableau_y/n', 'tensorflow_y/n', 'job_simplified',
       'seniority', 'description_length'],
      dtype='object')
```

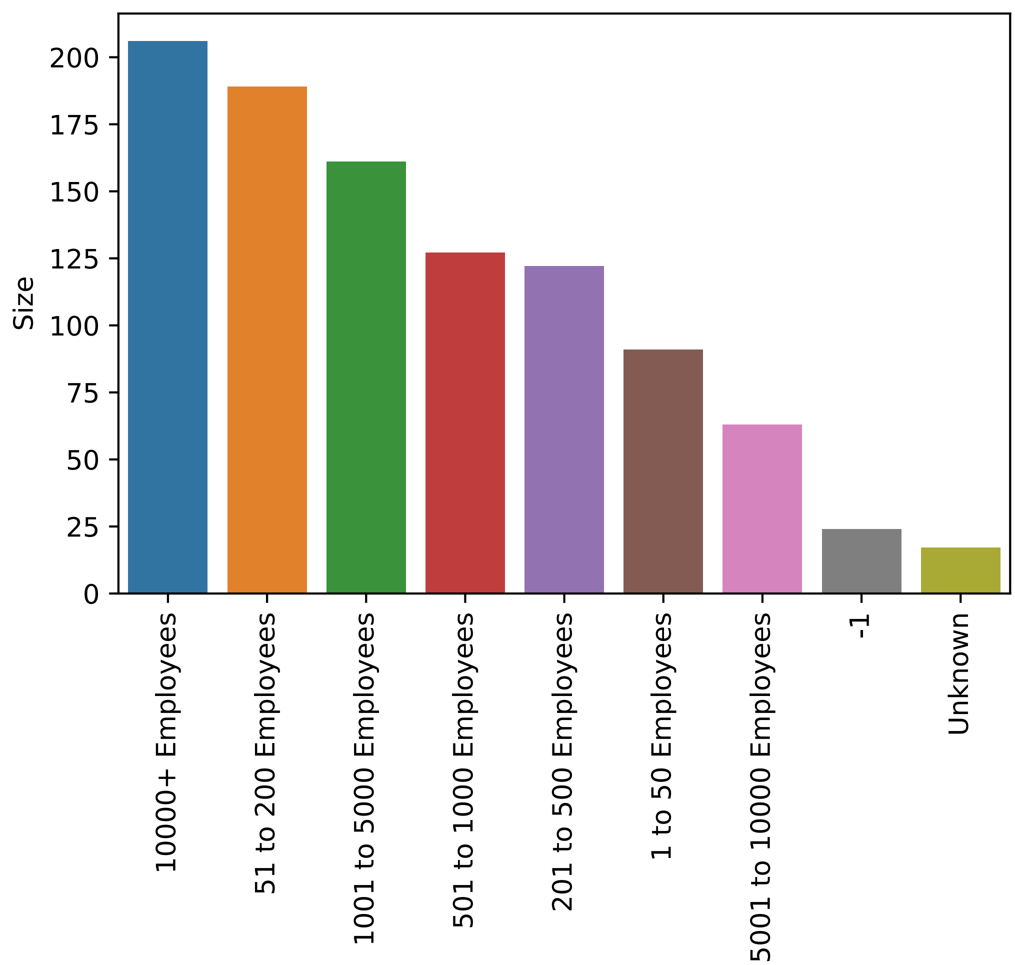
In [20]:
```
df_cat = df[['job_state', 'Size', 'python_y/n', 'r_y/n', 'spark_y/n', 'aws_y/n', 'excel_y/n', 'sas_y/n', 'matlab_y/n', 'tableau_y/n', 'tensorflow_y/n', 'job_simplified', 'seniority', 'Type of ownership', 'Sector', 'Revenue']]
```

In [21]:
```
for column in df_cat.columns:
    cat_num = df_cat[column].value_counts()
    print("graph for {}: total = {}".format(column, len(cat_num)))
    chart = sns.barplot(x= cat_num.index, y= cat_num)
    chart.set_xticklabels(chart.get_xticklabels(), rotation= 90)
    plt.show()
```

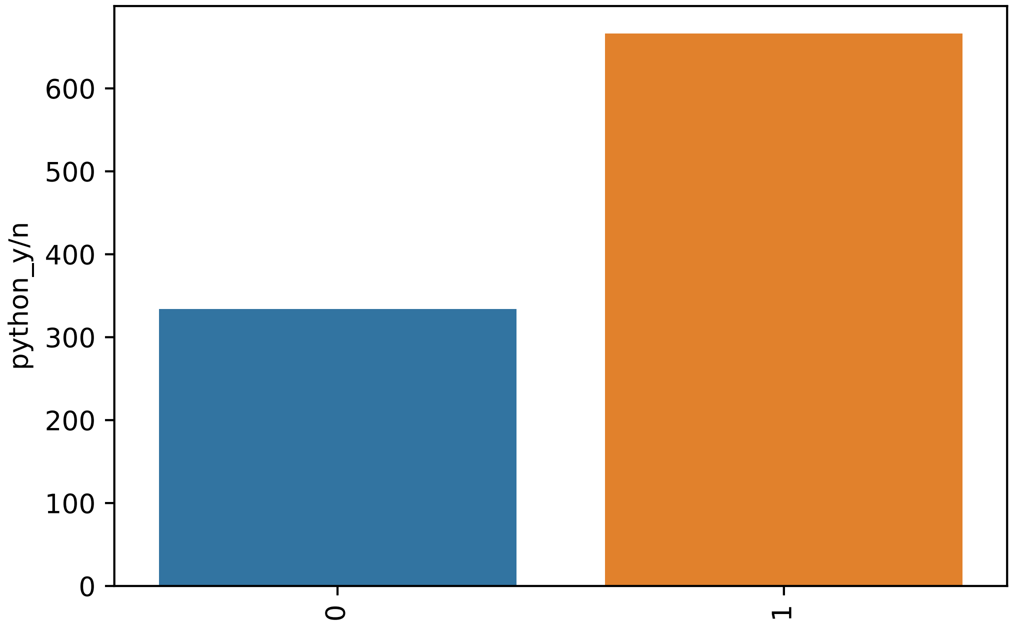
```
graph for job_state: total = 38
```



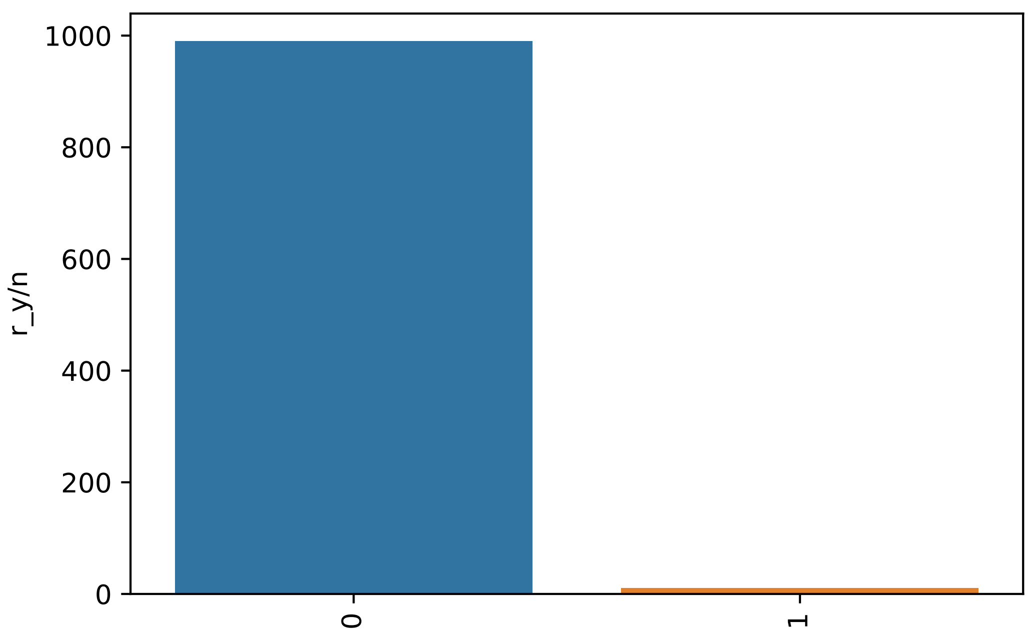

```
graph for Size: total = 9
```



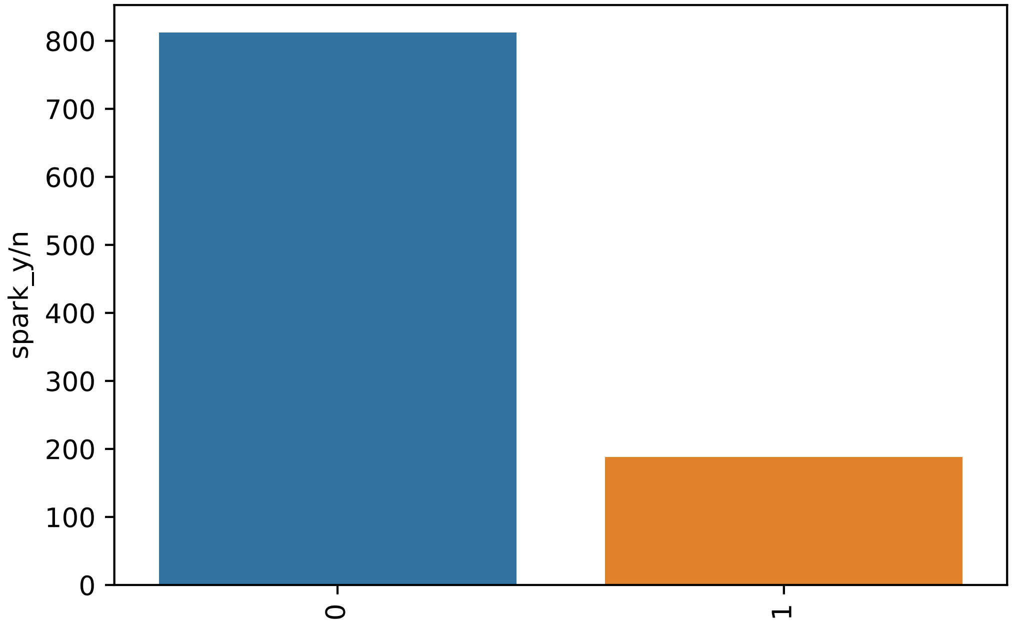
```
graph for python_y/n: total = 2
```



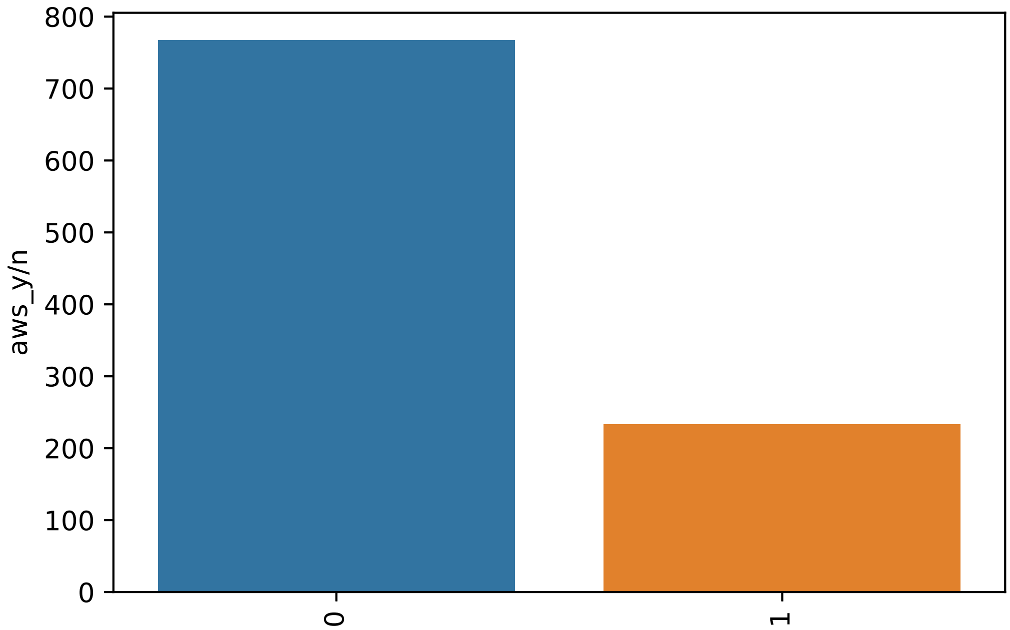

```
graph for r_y/n: total = 2
```



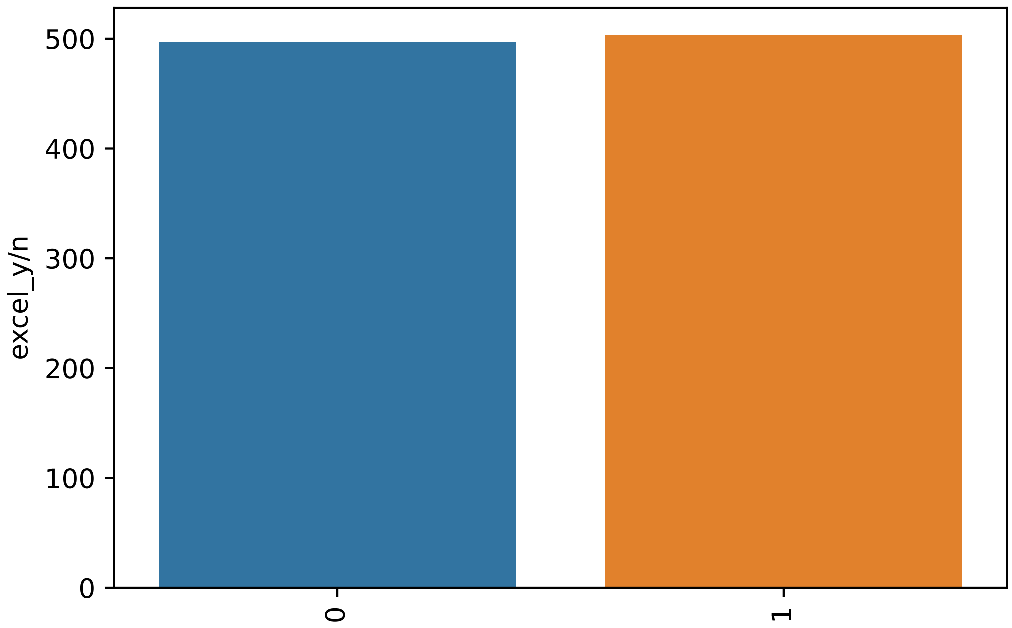

```
graph for spark_y/n: total = 2
```



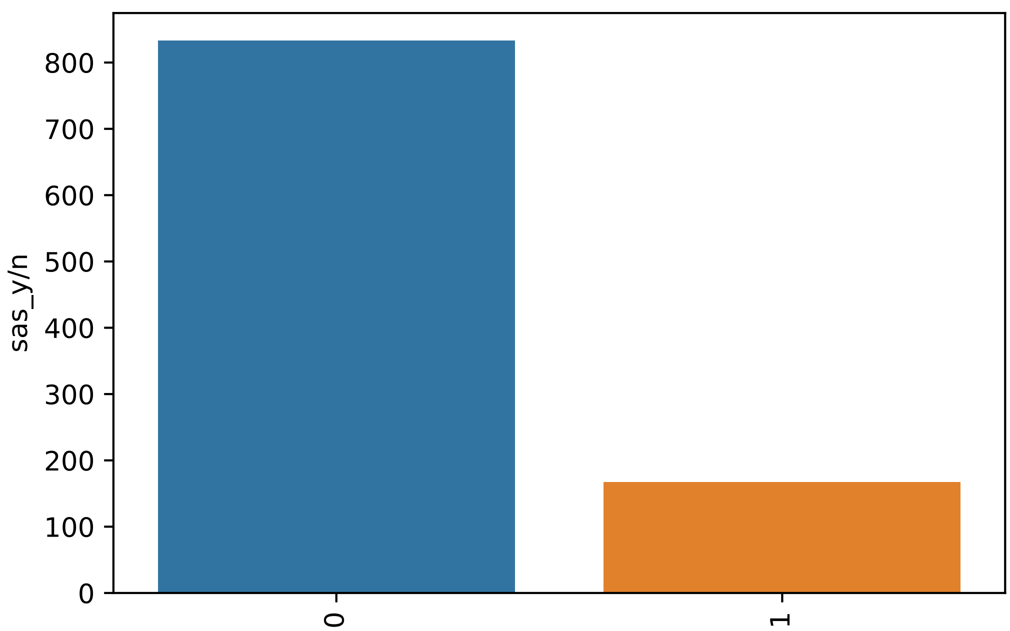

```
graph for aws_y/n: total = 2
```



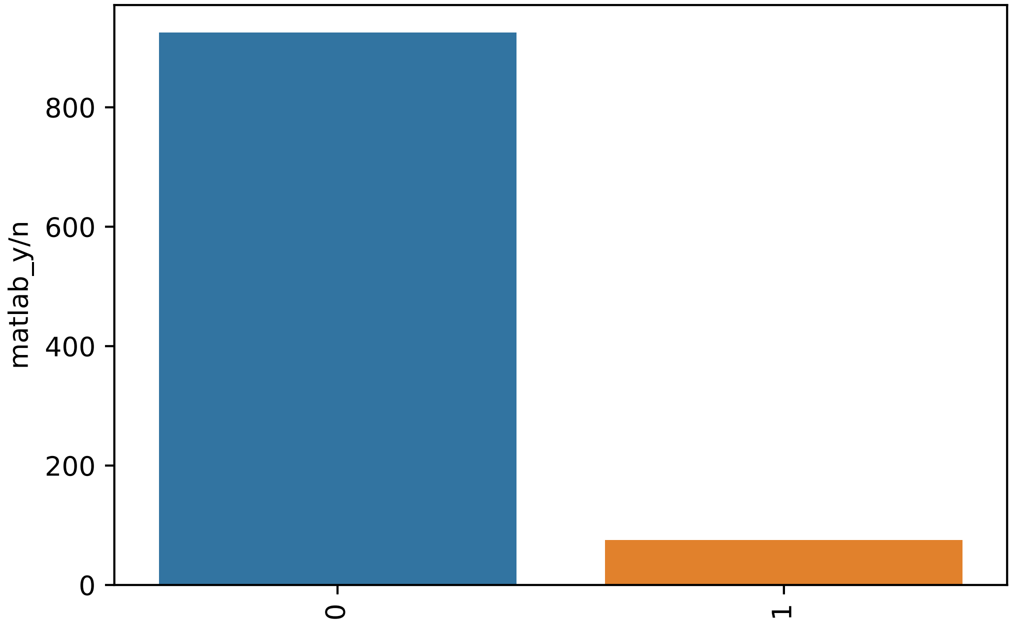

```
graph for excel_y/n: total = 2
```



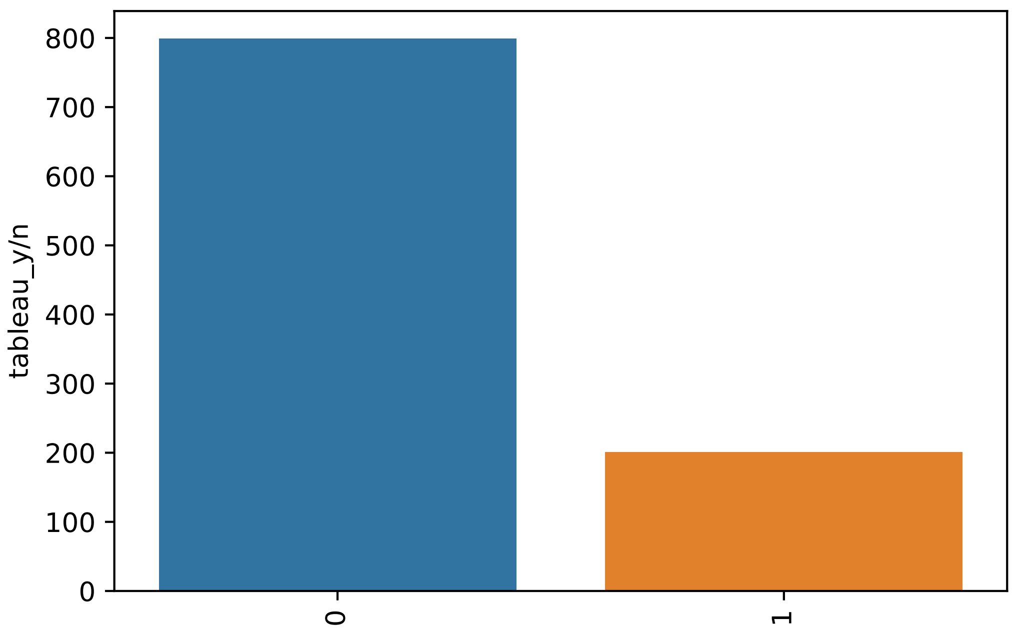

```
graph for sas_y/n: total = 2
```



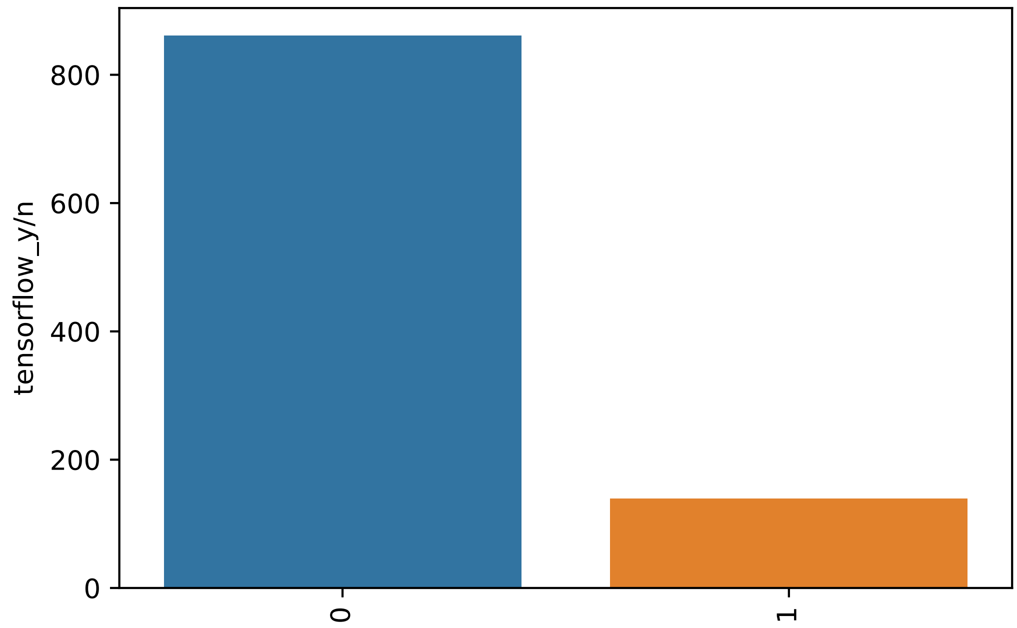

```
graph for matlab_y/n: total = 2
```



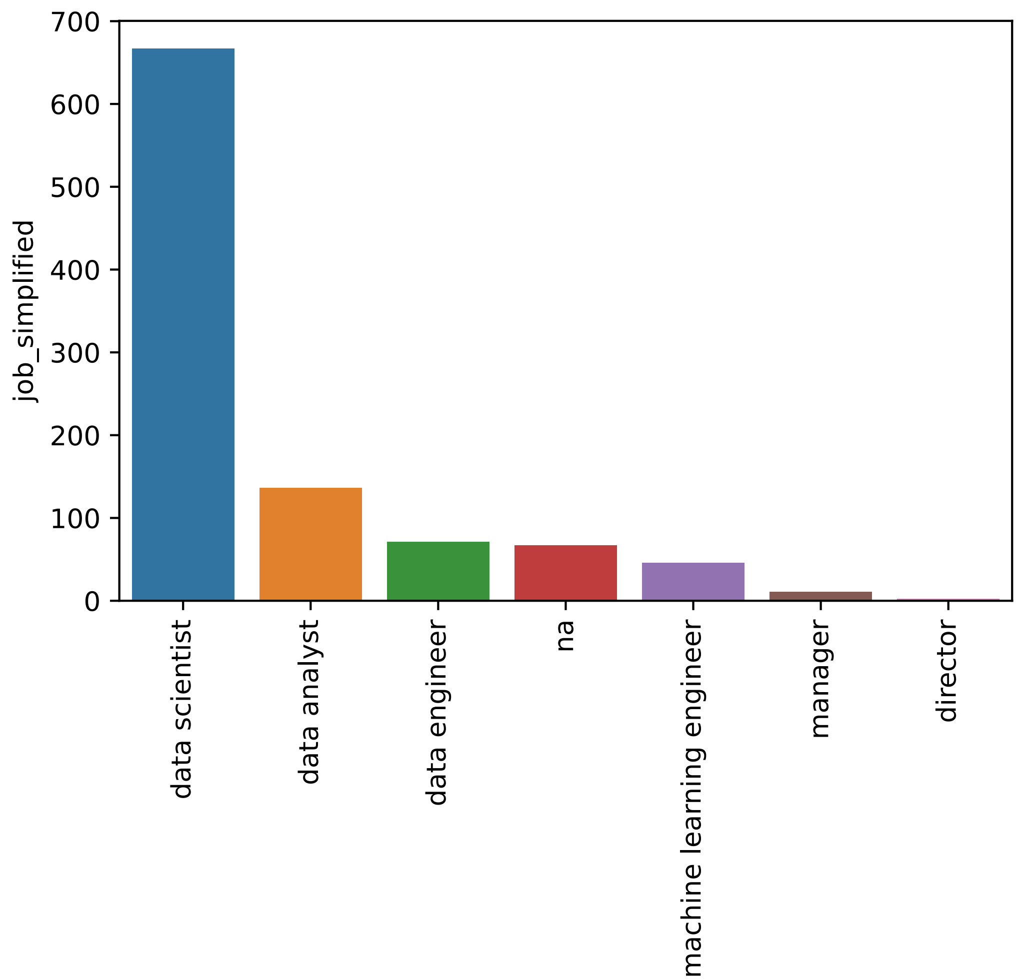

```
graph for tableau_y/n: total = 2
```



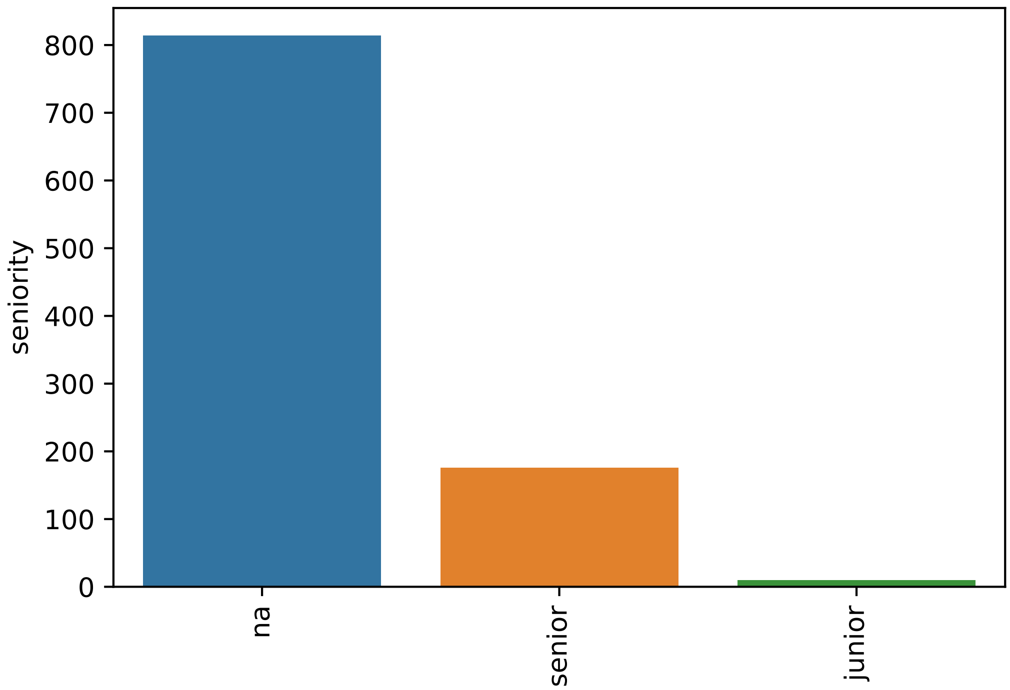

```
graph for tensorflow_y/n: total = 2
```



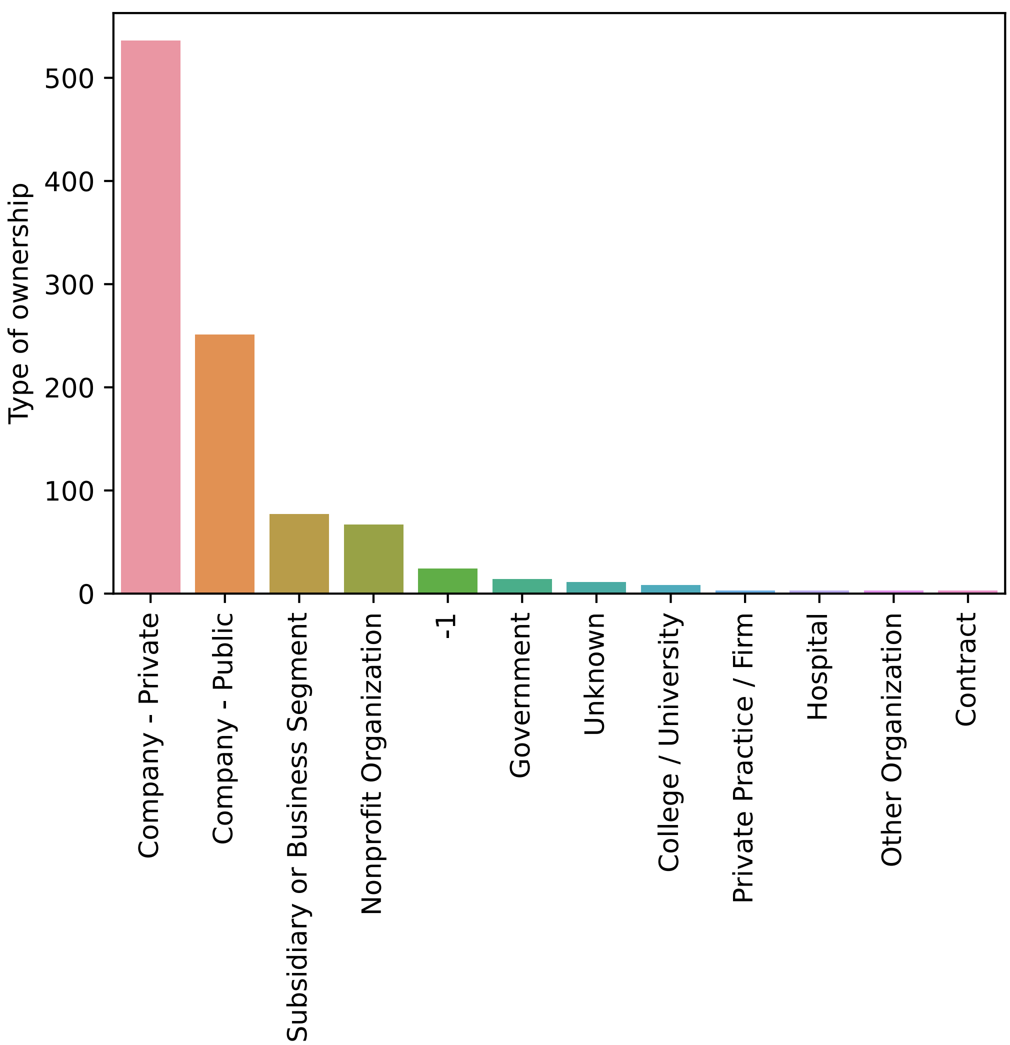
graph for job_simplified: total = 7



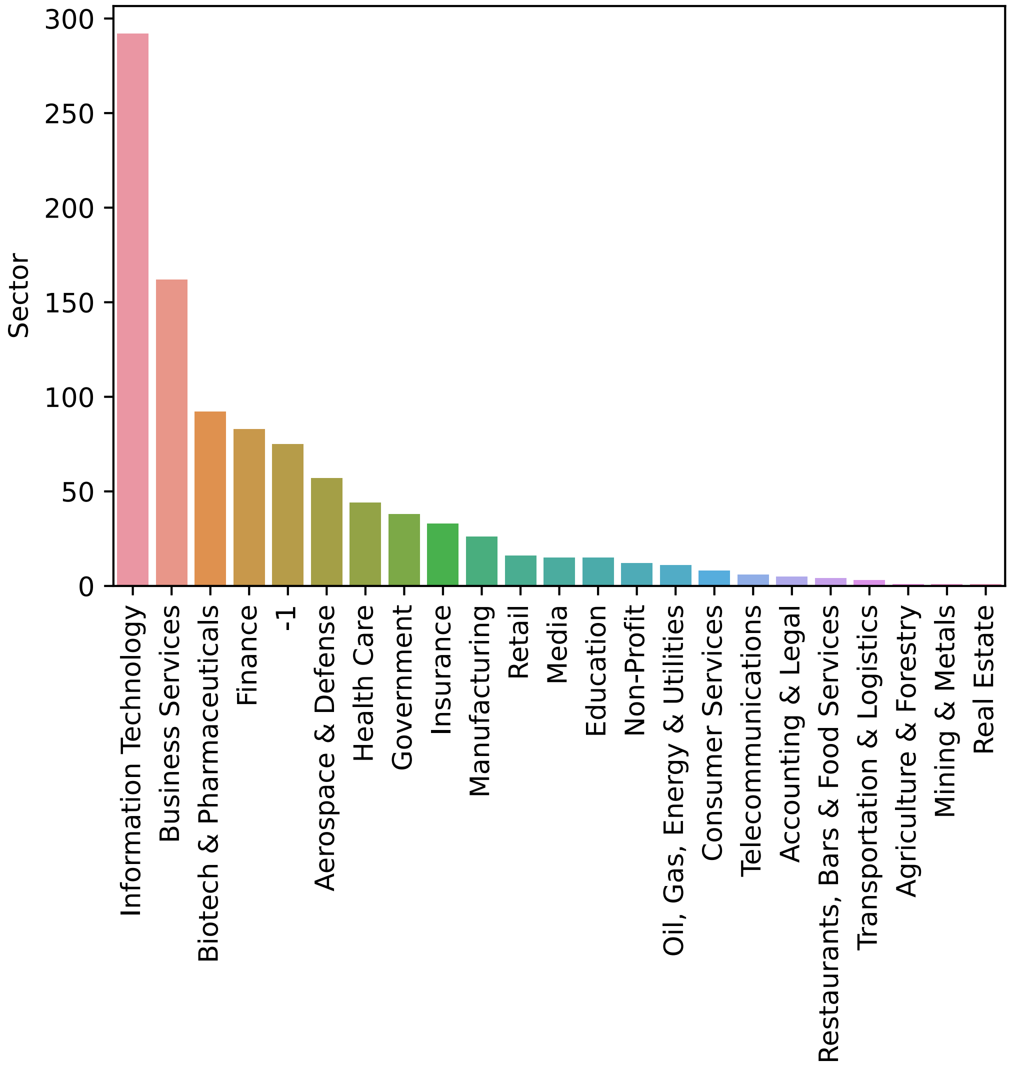
graph for seniority: total = 3



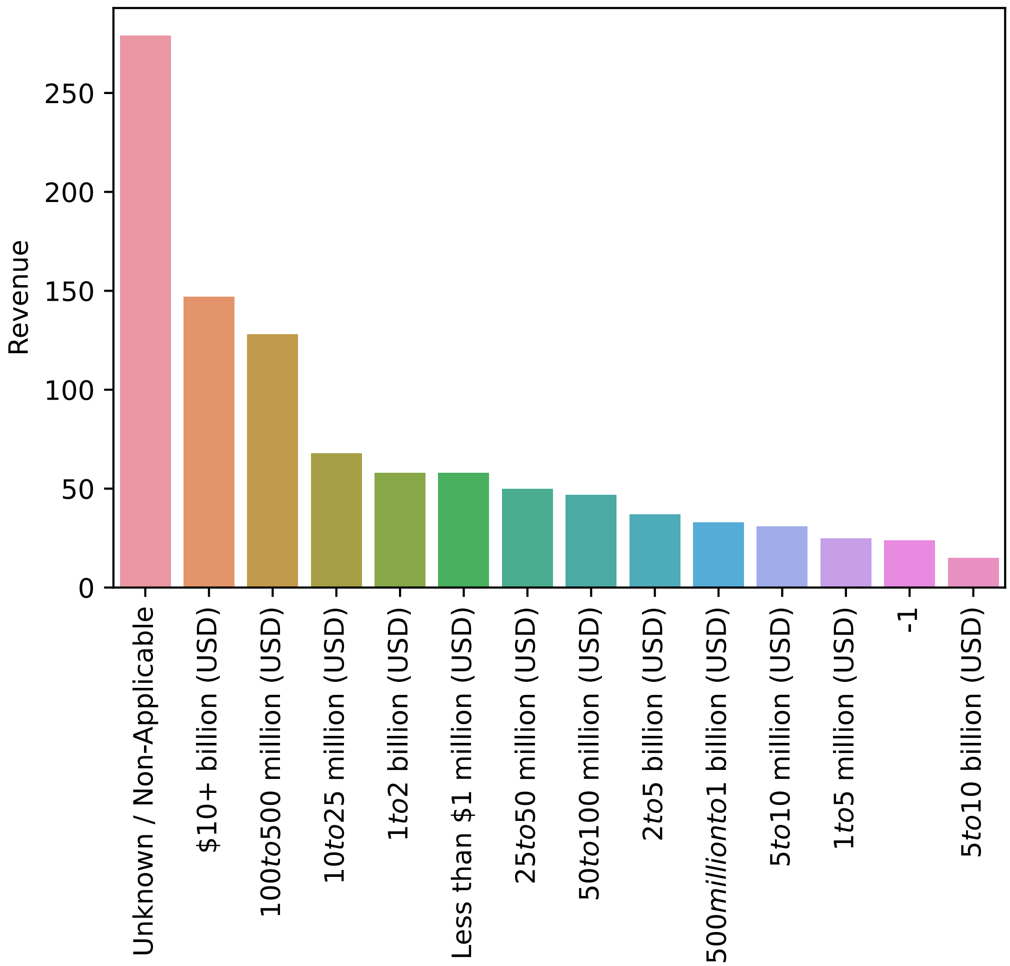
graph for Type of ownership: total = 12



graph for Sector: total = 23



graph for Revenue: total = 14



Revenue
0
50
100
150
200
250
Unknown / Non-Applicable
$10+ billion (USD)
100to500 million (USD)
10to25 million (USD)
1to2 billion (USD)
Less than $1 million (USD)
25to50 million (USD)
50to100 million (USD)
2to5 billion (USD)
500millionto1 billion (USD)
5to10 million (USD)
1to5 million (USD)
-1
5to10 billion (USD)

- ## **Which states have a high demand for data scientists?**

  Companies based on California posts around 1/4 of data scientists jobs. Virginia posts almost 150 out of 1000 data scientists jobs. Because of the pandemic, remote jobs demand is also high.
- ## **Company of which size/industry/type of ownership hire more data scientists?**

  Size: The trend is obvious. The larger the size of a company, the higher the demand for data scientists. Companies with 10,000+ employees post 200 out of 1000 data scientists jobs.

  Industry: Companies from information technology industry posts ~300 out of 1000 data scientists jobs. The second hottest industry for data scientists is business services industry.

  Type of ownership: Not surprisingly, private companies post more than 500 out of 1000 data scientists jobs.
- ## **How many jobs require/prefer applicants to have python/r/spark/aws/excel/sas/matlab/tableau/tensorflow skills?**

  Python: almost 700 out of 1000 jobs require python skills. Python proves to be a "must have" skill for data scientists.

  R: Surprisingly, R studio is not included in any job description. I assume companies do not use "R studio" to mention R skills. However, the letter "r" is used so frequently that my function cannot precisely distingush between R for R languages and "r" in any other words.

  Spark: 200 out of 1000 companies prefer applicants with Spark knowledges.

  Excel: Excel still appeared to be a "must have" skill for data scientists.

  Other skills are not mentioned very frequently in job descriptions.

Pivot tables

I made several pivot tables to show the average salaries for different columns

In [23]:
```
# avg salary for different job titles and seniority
pd.pivot_table(df, index= ["job_simplified", "seniority"], values= "avg_salar
y").sort_values("avg_salary", ascending= False)
```

Out[23]:

job_simplified	seniority	avg_salary
manager	na	132.125000
data engineer	senior	125.500000
na	senior	124.166667
data scientist	junior	123.650000
data analyst	senior	118.987179
data scientist	senior	114.481308
na	na	111.801724
machine learning engineer	na	110.447368
data scientist	na	109.402727
data analyst	na	109.020619
data engineer	na	107.631148
manager	senior	99.666667
machine learning engineer	senior	97.750000
director	na	96.750000

In [29]:
```
# avg salary for different states and job titles
pd.pivot_table(df, index= ["job_state", "job_simplified"], values= "avg_salar
y", aggfunc="count").sort_values(by= "avg_salary", ascending= False)
```

Out[29]:

job_state	job_simplified	avg_salary
CA	data scientist	180
VA	data scientist	106
NY	data scientist	66
MA	data scientist	63
DC	data scientist	38
...	...	...
NY	na	1
OH	na	1
PA	data engineer	1
Remote	manager	1
WY	data analyst	1

105 rows × 1 columns

Data scientists work in California earn more salary than data scientists who work in other states

Wordcloud from 1000 job descriptions

I created a wordcloud from all job descriptions of 1000 jobs to see which words are frequently mentioned by companies

In [30]:
```
comment_words = ''
stopwords = set(STOPWORDS)
# iterate through the csv file
for val in df["Job Description"]:
    # typecaste each val to string
    val = str(val)
    #split the value
    tokens = val.split()
    # converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()
    comment_words += " ".join(tokens)+" "
wordcloud = WordCloud(width= 800, height= 800, background_color= "white", stop
words= stopwords,
                      min_font_size= 10).generate(comment_words)
#plot the image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

Export to CSV

```
In [31]: df.to_csv('data-scientist-salary-eda.csv', index= False)
```

model building

Model Building

I am to build a regression model to help future data scientists get their up-to-date salary estimate.

I will perform:

- Multiple linear regression
- Lasso regression
- Ridge regression
- Random forest regressor

to find out the best-performing model.

I would use **"negative mean absolute error"** to score each model. Since I am trying to predict a numerical value, I think mean absolute error (MAE) would be the most direct score to compare.

Import packages

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_absolute_error
import pickle
```

In [2]:
```
df = pd.read_csv("data-scientist-salary-eda.csv")
```

Select and modify relevant columns

In [3]:
```
df_model = df[['avg_salary', "Rating", "Size", 'Type of ownership', 'Industry'
, 'Revenue', 'job_state', 'age', 'python_y/n', 'spark_y/n', 'aws_y/n', 'excel_
y/n', 'sas_y/n', 'matlab_y/n', 'tableau_y/n', 'tensorflow_y/n', 'job_simplifie
d', 'seniority', "description_length"]]
```

Convert categorical variables to dummy variables

In [4]:
```
df_dum = pd.get_dummies(df_model)
```

In [5]:
```
# replace "-1" in rating column with the average of rating
df_dum["Rating"].replace(-1.0, np.mean(df_dum["Rating"]), inplace= True)
df_dum["Rating"].value_counts()
```

Out[5]:
```
3.900    79
4.400    78
4.100    72
4.000    72
3.800    70
3.700    67
3.200    55
3.500    48
3.655    44
3.600    43
4.200    43
3.300    40
3.400    40
4.500    39
5.000    29
4.300    28
3.100    27
4.600    22
3.000    18
4.800    17
4.700    15
2.800    13
4.900    12
2.900     9
2.700     7
2.400     6
2.500     5
2.200     1
2.000     1
Name: Rating, dtype: int64
```

In [6]:
```
# replace "-1" in age column with the average of age
df_dum["age"].replace(-1, np.mean(df_dum["age"]), inplace= True)
df_dum["age"].value_counts()
```

Out[6]:
```
32.361     143
26.000      39
8.000       39
9.000       36
10.000      34
          ...
0.000        1
57.000       1
61.000       1
80.000       1
158.000      1
Name: age, Length: 110, dtype: int64
```

Split train and test data sets

In [7]:
```
X = df_dum.drop("avg_salary", axis= 1)
y = df_dum["avg_salary"].values
```

In [35]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Multiple linear model

In [9]:
```
lm = LinearRegression()
lm.fit(X_train, y_train)
```

Out[9]: LinearRegression()

In [10]:
```
np.mean(cross_val_score(lm, X_train, y_train, scoring= "neg_mean_absolute_erro
r"))
```

Out[10]: -23.922701323066228

Lasso

In [11]:
```
lm_l = Lasso()
lm_l.fit(X_train, y_train)
np.mean(cross_val_score(lm_l, X_train, y_train, scoring= "neg_mean_absolute_er
ror", cv= 10))
```

Out[11]: -21.897947574524572

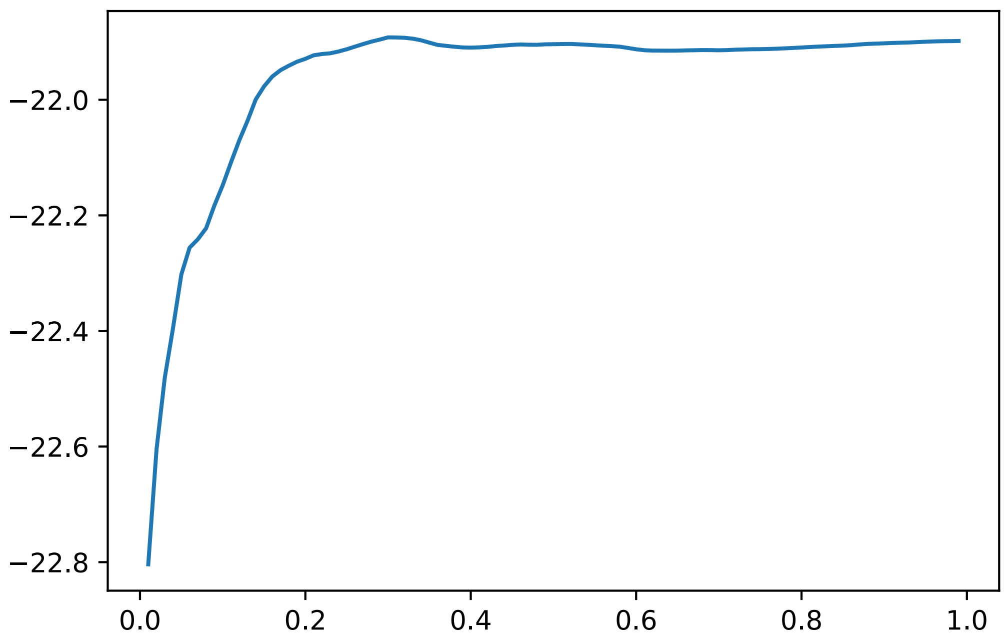
In [12]:
```
# Use a for loop to find the optimal alpha value

lasso_alpha= []
lasso_error= []

for i in range(1,100):
    lasso_alpha.append(i/100)
    lml = Lasso(alpha=(i/100))
    lasso_error.append(np.mean(cross_val_score(lml,X_train,y_train, scoring =
'neg_mean_absolute_error', cv= 10)))

plt.plot(lasso_alpha, lasso_error)
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x150a36b6100>]
```



In [30]:
```
lasso_err = list(zip(lasso_alpha, lasso_error))
lasso_err_df = pd.DataFrame(lasso_err, columns=["alpha", "error"])
lasso_err_df[lasso_err_df["error"] == lasso_err_df.error.max()]
```

Out[30]:

	alpha	error
29	0.3	-21.892004

In [31]:
```
# Modify the lasso classifier with the optimal alpha
lm_l = Lasso(0.3)
lm_l.fit(X_train, y_train)
```

Out[31]:
```
Lasso(alpha=0.3)
```

Ridge regression

In [15]:
```
lm_rid = Ridge()
lm_rid.fit(X_train, y_train)
np.mean(cross_val_score(lm_l, X_train, y_train, scoring= "neg_mean_absolute_er
ror", cv= 10))
```

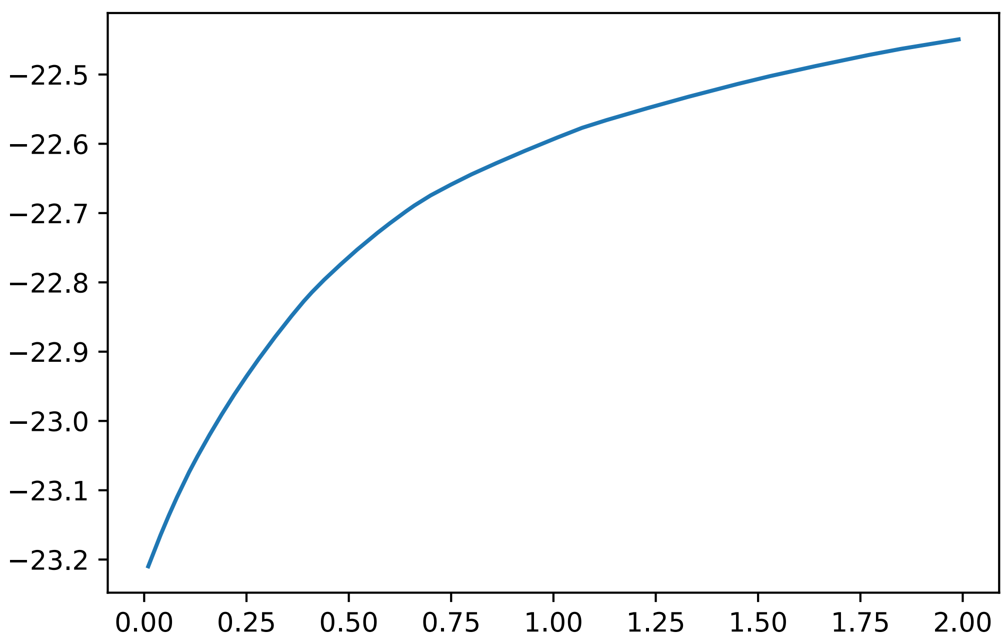
Out[15]: -21.897092978792376

In [16]:
```
# Use a for loop to find the optimal alpha value
ridge_alpha = []
ridge_error = []

for i in range(1,200):
    ridge_alpha.append(i/100)
    lmrid = Ridge(i/100)
    ridge_error.append(np.mean(cross_val_score(lmrid, X_train, y_train, scorin
g= "neg_mean_absolute_error", cv= 10)))

plt.plot(ridge_alpha, ridge_error)
```

Out[16]: [<matplotlib.lines.Line2D at 0x150a38c8910>]



In [17]:
```
ridge_err = list(zip(ridge_alpha, ridge_error))
ridge_err_df = pd.DataFrame(ridge_err, columns=["alpha", "error"])
ridge_err_df[ridge_err_df["error"] == ridge_err_df.error.max()]
```

Out[17]:

	alpha	error
198	1.99	-22.449288

In [18]:
```
# Modify the ridge classifier with the optimal alpha
lm_rid = Ridge(alpha= 1.99)
lm_rid.fit(X_train, y_train)
```

Out[18]:
```
Ridge(alpha=1.99)
```

Random forest regressor

In [19]:
```
rf = RandomForestRegressor()
cross_val_score(rf, X_train, y_train, scoring= "neg_mean_absolute_error", cv=
10)
```

Out[19]:
```
array([-22.59745218, -22.68931432, -22.79800017, -23.86024178,
       -23.56734636, -22.66663076, -19.96379468, -24.138568  ,
       -24.4323657 , -24.56125818])
```

In [20]:
```
# Use RandomizedSearchCV to find the optimal parameters
parameters = {'n_estimators':range(10,300,10),
              'criterion':('mse', 'mae'),
              'max_features':('auto', 'sqrt', 'log2')}
```

In [21]:
```
gs = RandomizedSearchCV(rf, parameters, scoring= "neg_mean_absolute_error", cv
=10)
```

In [22]:
```
gs.fit(X_train, y_train)
```

Out[22]:
```
RandomizedSearchCV(cv=10, estimator=RandomForestRegressor(),
                   param_distributions={'criterion': ('mse', 'mae'),
                                        'max_features': ('auto', 'sqrt',
                                                         'log2'),
                                        'n_estimators': range(10, 300, 10)},
                   scoring='neg_mean_absolute_error')
```

In [23]:
```
gs.best_score_
```

Out[23]:
```
-22.376648437500002
```

In [24]:
```
gs.best_estimator_
```

Out[24]:
```
RandomForestRegressor(criterion='mae', max_features='log2', n_estimators=200)
```

Test the performance of each model

In [36]:
```
tpred_lm = lm.predict(X_test)
tpred_lml = lm_l.predict(X_test)
tpred_ridml = lm_rid.predict(X_test)
tpred_rf = gs.best_estimator_.predict(X_test)
```

In [37]:
```
print("Multiple linear regression MAE: ", mean_absolute_error(y_test, tpred_lm
))
print("Lasso regression MAE: ", mean_absolute_error(y_test, tpred_lml))
print("Ridge regression MAE: ", mean_absolute_error(y_test, tpred_ridml))
print("Random forest regressor MAE: ", mean_absolute_error(y_test, tpred_rf))
```

```
Multiple linear regression MAE:  21.759900893859566
Lasso regression MAE:  22.287458153396003
Ridge regression MAE:  21.971370411902388
Random forest regressor MAE:  16.6944625
```

Save the model

In [28]:
```
pickl = {'model': lml}
pickle.dump( pickl, open( 'model_file' + ".p", "wb" ) )
```