Insights from job and company data from

&

Leo Walker and Anthony Molieri

Dataset

Combined dataset containing job descriptions from Indeed and the corresponding benefits ratings for the companies from Glassdoor.

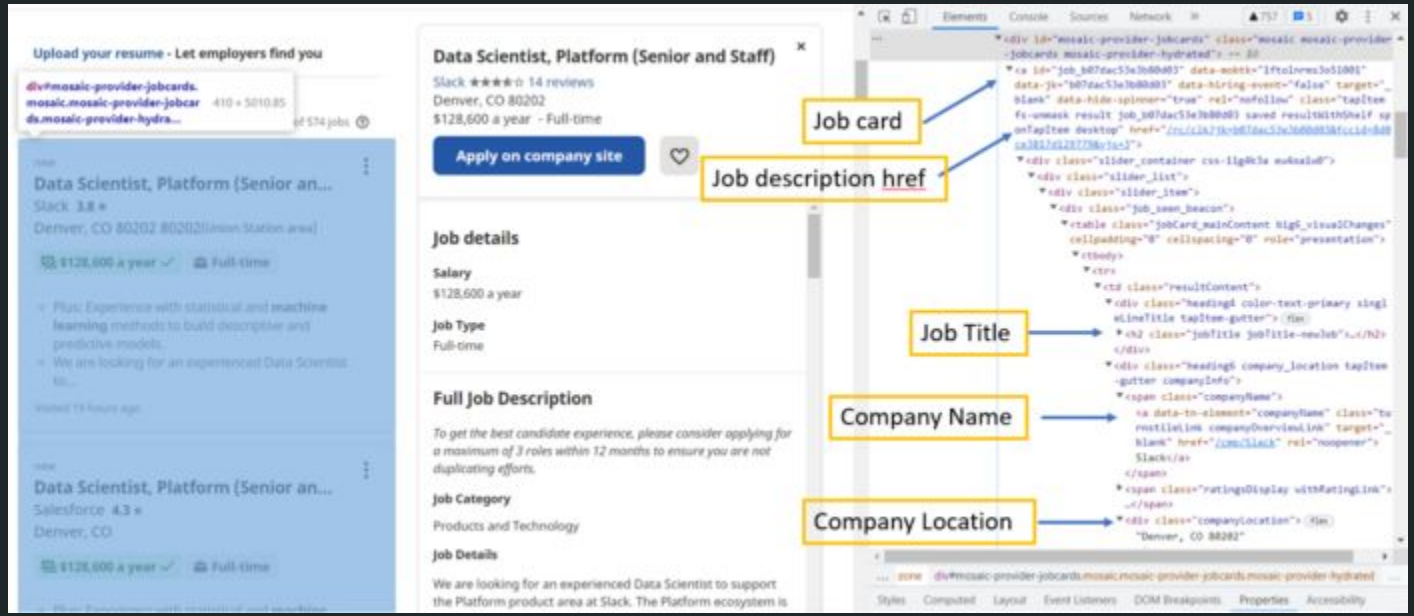
- Scraped Indeed.com for 5 positions from 25 “tech” cities.
  - Data Scientists, Machine Learning Engineer, Data Analyst, BI Analyst, and Data Engineer
  - Atlanta, GA, Austin, TX, Boston, MA, Chicago, IL, Denver, CO, Dallas-Ft. Worth, TX, Los Angeles, CA, NYC, NY, San Francisco, CA, Seattle, WA, Phoenix, AZ, Salt Lake City, UT, San Antonio, TX, San Diego, CA, Jacksonville, FL, Columbus, OH, Boise, ID, Washington DC, Portland, OR, Kansas City, Raleigh, NC, Boulder, CO, Miami, FL, Northern Virginia, Orlando, FL
- Searched, scraped, and aggregated benefits ratings for corresponding companies from glassdoor

Motivation

Get job descriptions and meta data of popular data related jobs to see how they relate to one another and get insights using their metadata.

To help give a better idea of how employees at companies with open positions rate their benefits other than salary.

Scraping Indeed.com



5 job categories * 25 cities = 125 pages
125 pages * ~14 jobs per page = ~1,750 jobs per week

WebScraping API

https://medium.com/@leo-walker
Article: Web Scraping “Data” Jobs with Python

Scraping Glassdoor

- Used company names from the the indeed dataset
- Two loops:
  a. Search for the right company
  b. Scraped the benefits data
- Run multiple times to update table with missing companies

Data Cleaning and Feature Engineering

Indeed:

- Extract State from company location
- Extract Salary from job desc
- Infer job category from job title
- Extract low, high, and pay rate from salary range
- Calculate estimated annual salary

```
indeed_df.loc[:,'state'] = indeed_df.loc[:, 'company_location'].str.extract(r', ([A-Z]{2})')
```

```
def salary_extract(row):
    if row['est_salary']!="No Estimated Salary":
        return row['est_salary']
    else:
        pattern = r"\$([0-9]{5,6}|[0-9]{2,3},[0-9]{3})"
        found = re.findall(pattern, row['job_desc'])
        if len(found)==1:
            return f'{found[0]} a year'
        elif len(found)>1:
```

```
def set_job_category(job_title):
    regex = re.compile('[^a-zA-Z]')
    clean_title = regex.sub('', job_title)
    clean_title = clean_title.lower()
    if all(x in clean_title for x in ["data", "scien"]):
        return "Data Scientist"
    elif all(x in clean_title for x in ["business", "analy"]):
```

```
def salary_low(row):
    if row == None:
        return None
    salary = str(row).split('-')
    pattern = r'([0-9]+\.*,*[0-9]+)'
    low = float(''.join(re.findall(pattern, salary[0])).replace(',',''))
    return low

def salary_high(row):
    if row ==None:
        return None
    salary = str(row).split('-')
    pattern = r'([0-9]+\.*,*[0-9]+)'
    if len(salary)>1:
        high = float(''.join(re.findall(pattern, salary[1])).replace(',',''))
    else:
        high = float(''.join(re.findall(pattern, salary[0])).replace(',',''))
```

```
def calc_salary(row):
    pay_rate=row['pay_rate']
    if (pay_rate == 'Other') or (pay_rate==None):
        return None

    rate_change= {'annual':1, 'month':12, 'week':52, 'day':260, 'hour':2080}
    est_annual_salary = ((row['salary_low']+row['salary_high'])/2)*rate_change[pay_rate]
    return est_annual_salary
indeed_df['est_annual_salary'] = indeed_df.loc[:,['salary_low','salary_high','pay_rate']].apply(calc_salary, axis=1)
```

Data Cleaning and Feature Engineering

Glassdoor:

- Parse rating value
- Calculate weighted averages
- Aggregate ratings from 54 sub-domains up to 6 benefit domains (Family & Parenting, Financial & Retirement, Insurance, Health & Wellness, Perks & Discounts, Professional Support, Vacation & Time Off)
- Overall mean and average number of ratings per company

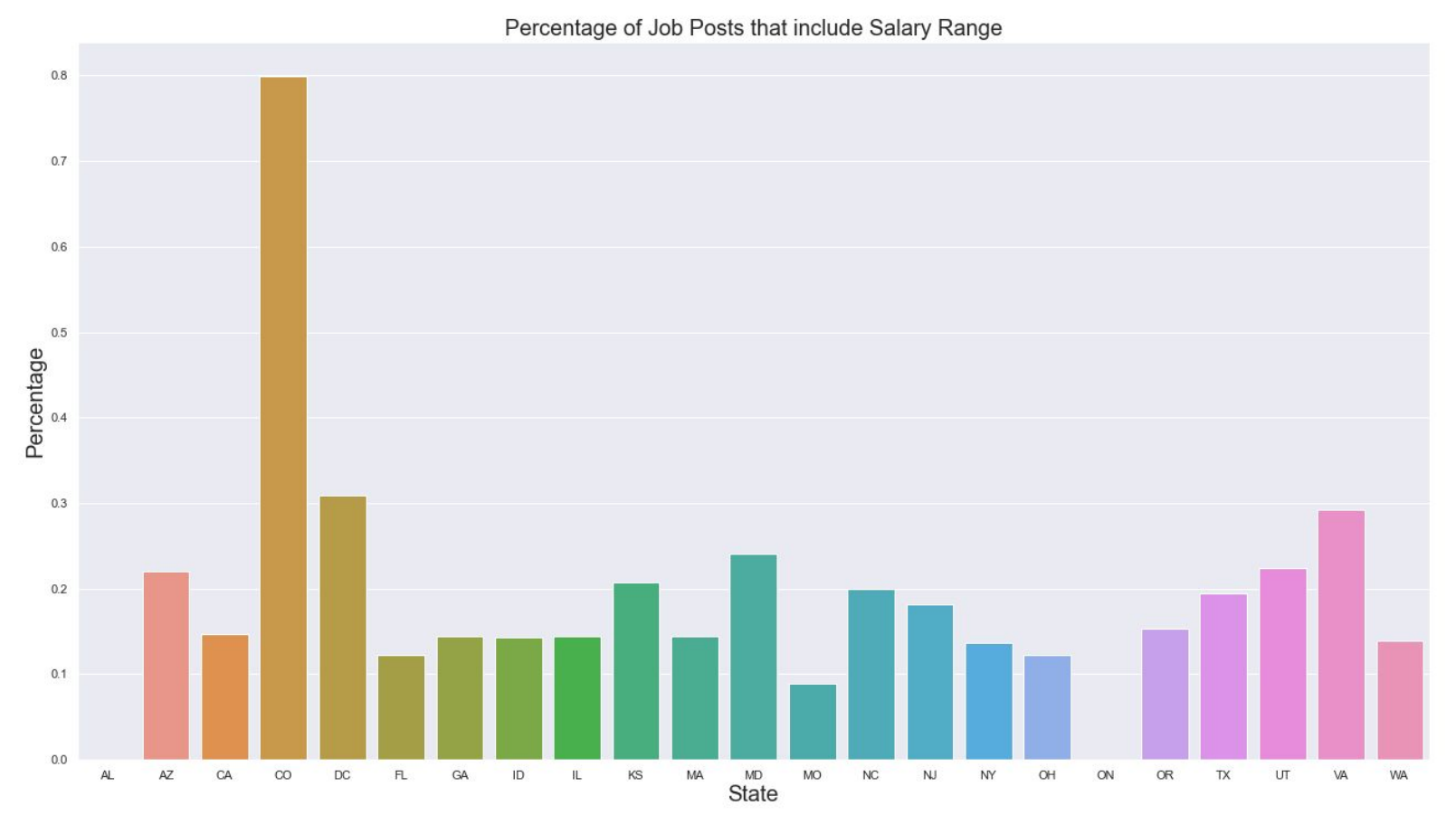
```
# Parse numeric rating from string.
glassdoor_df["count_of_ratings"] = glassdoor_df["count_of_ratings"].str.extract(
    "(\d+)", expand=False
)
```

```
# Calculate weighted average
scaleScores = glassdoor_df.groupby(["companies", "new_type"]).apply(
    lambda x: (x["rating"] * x["count_of_ratings"]).sum() / x["count_of_ratings"].sum()
)
```

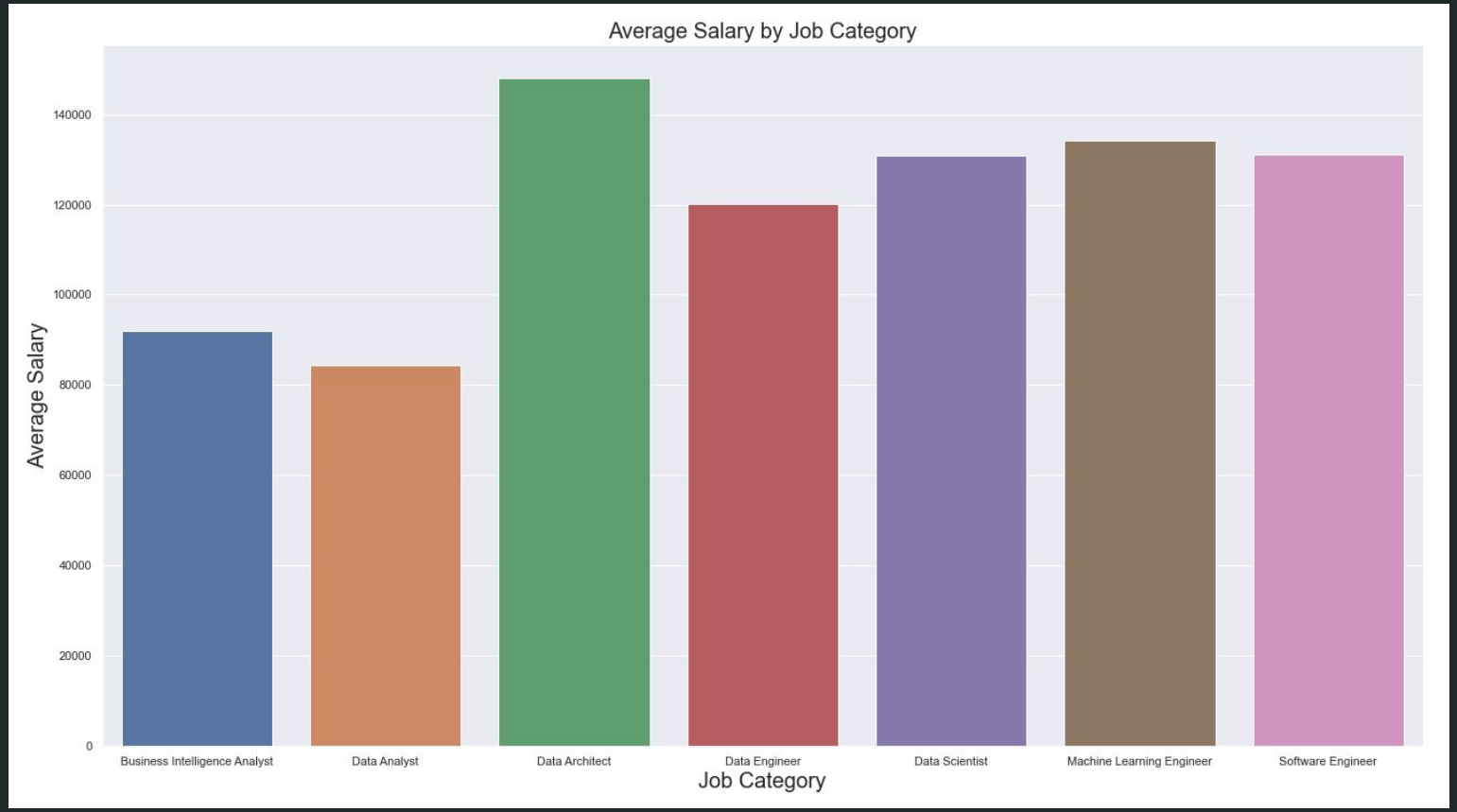
```
# Scale Scores
scaleScore_temp = scaleScores.to_frame().reset_index()

scaleScore_df = scaleScore_temp.pivot_table(0, ["companies"], "new_type")

scaleScore_df.reset_index(drop=False, inplace=True)
scaleScore_df.reindex(
    [
        "companies",
        "Family & Parenting",
        "Financial & Retirement",
        "Insurance, Health & Wellness",
        "Perks & Discounts",
        "Professional Support",
        "Vacation & Time Off",
    ],
    axis=1,
)

# Average of scores
countScore_temp = countScore.to_frame().reset_index()
countScore_temp.rename(columns={0: "ratings_meanCount"}, inplace=True)

scaleScore_df = scaleScore_df.merge(countScore_temp, on="companies")

scaleScore_df
```

Visualizations

Indeed: Percentage of Job Posts that include Salary Range

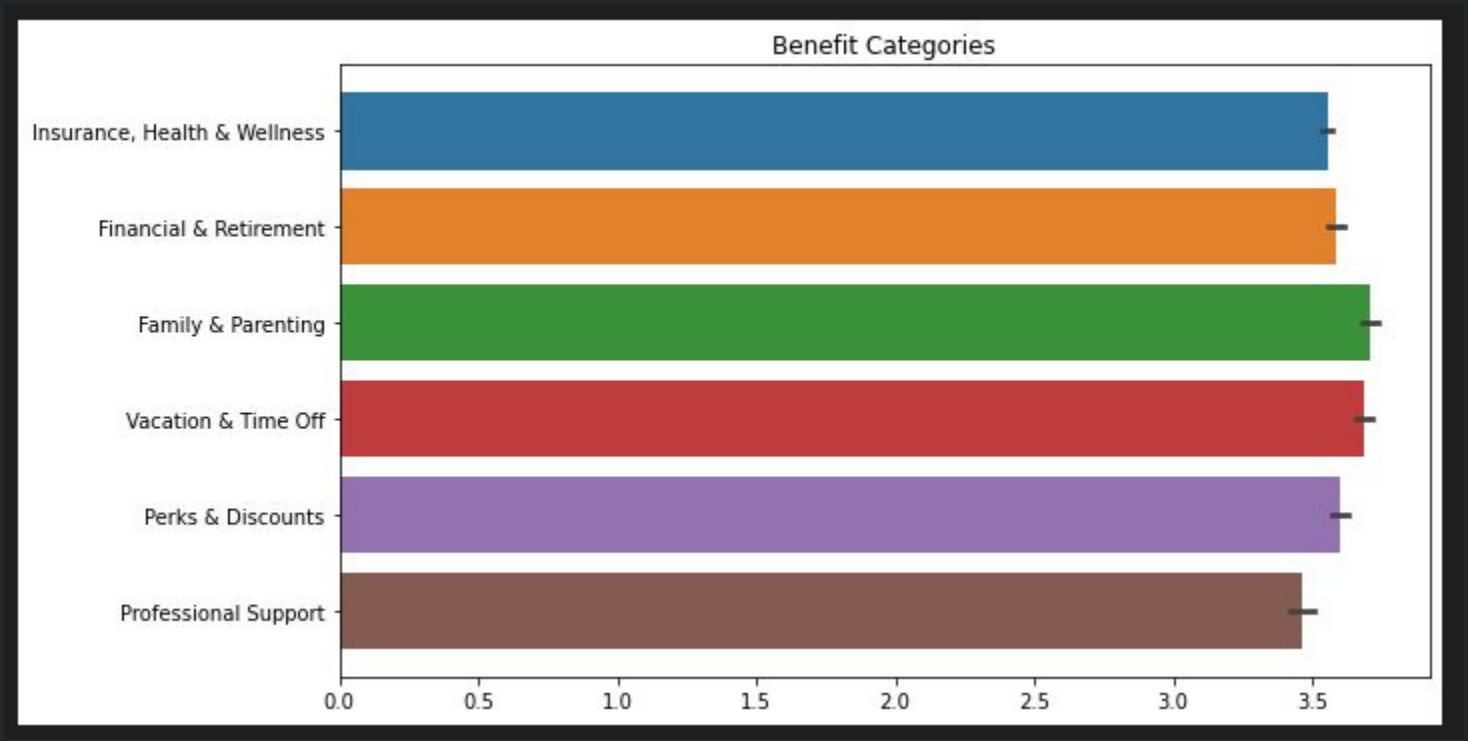


Indeed: Average Salary by Job Category

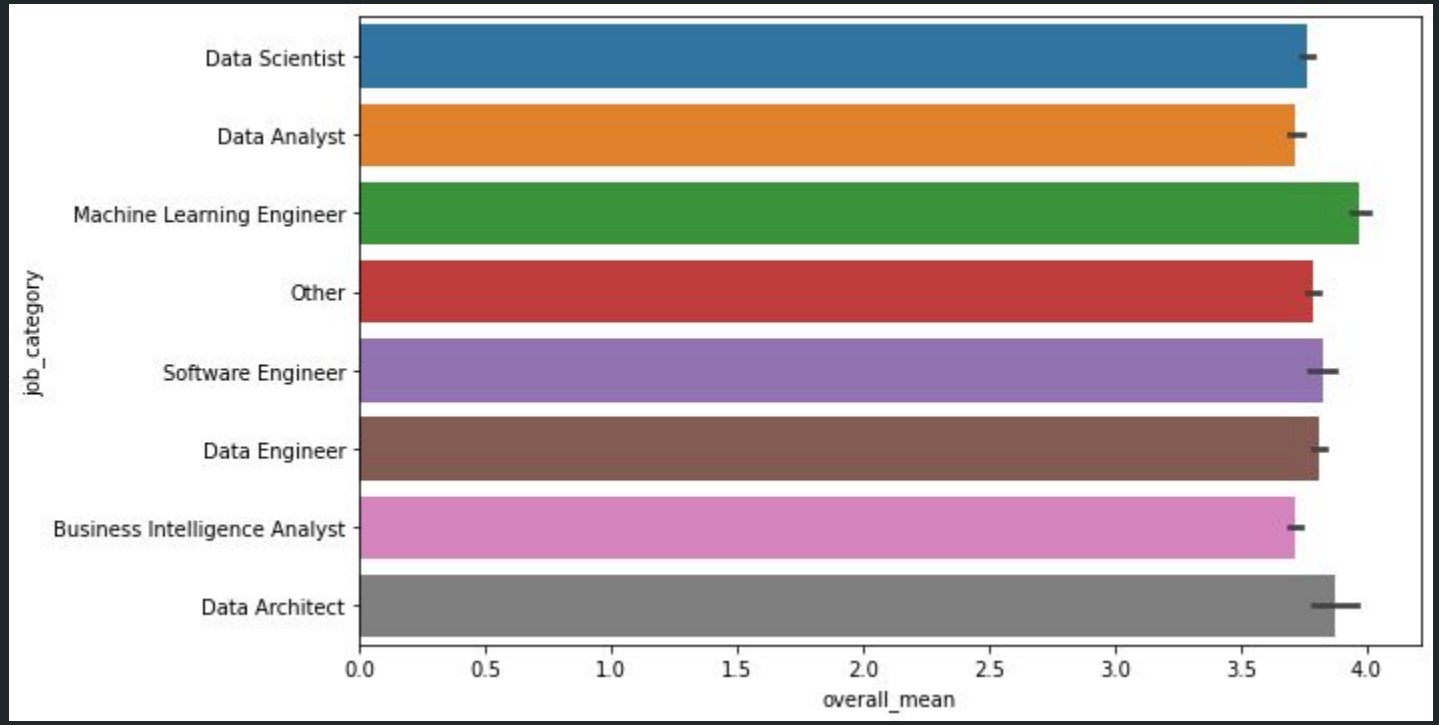


Indeed: Count of Job Posts that include Salary Range

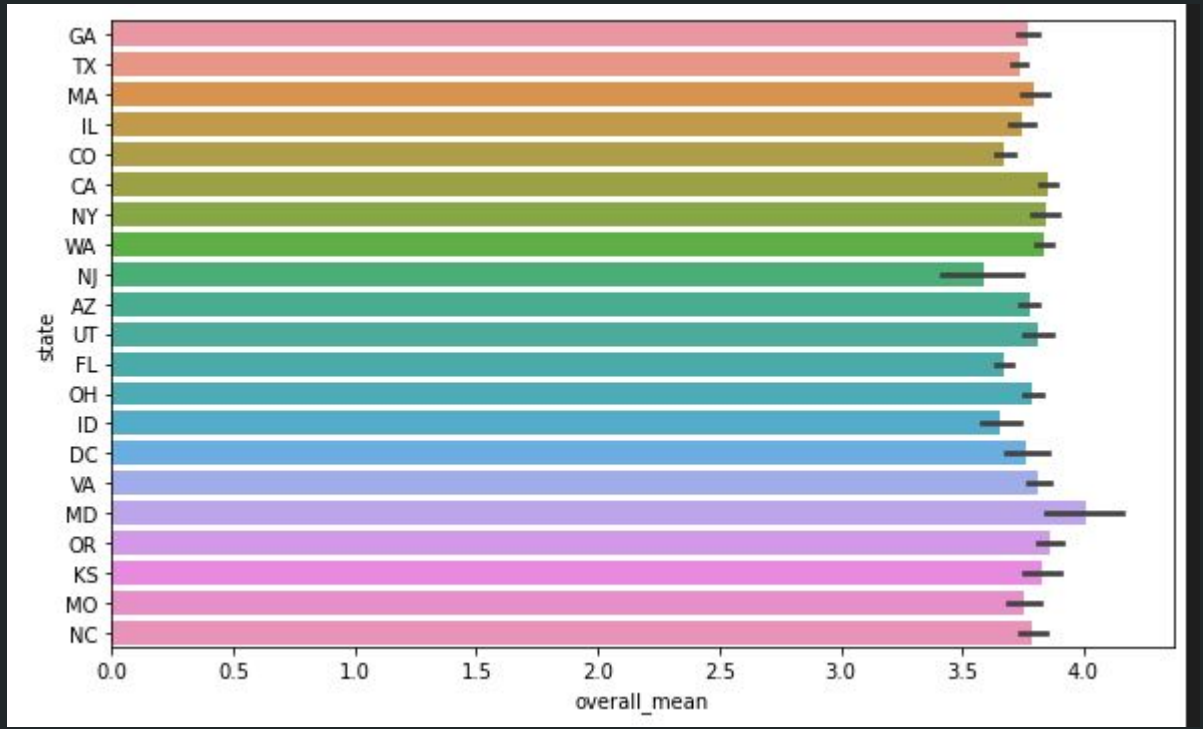
Glassdoor: Main Benefit Categories



Combined: Job Category and Average Rating

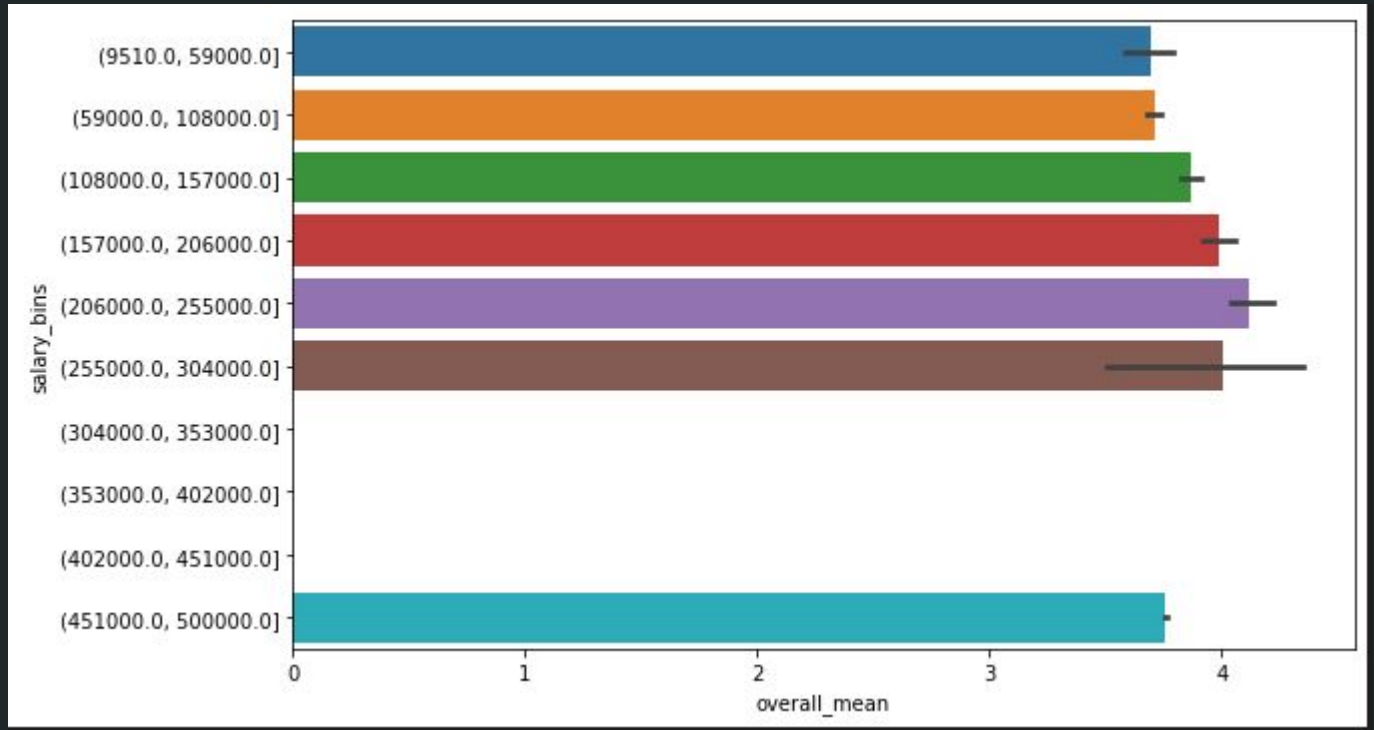


Combined: State* and Average Rating



*States with only one match between the datasets were dropped (Alabama and Ontario Canada)

Combined: Salary and Average Rating



Takeaways

- Salaries tend to be similar to each other within job descriptions
- Higher paid job titles: data architect, machine learning engineer, data scientist
- Ratings from glassdoor tend to be high across different categories with few exceptions

Natural Language Processing (NLP)

K-means and Latent Dirichlet Allocation (LDA)

Text Processing

- Drop blank Job Descriptions
- Filter down to our targeted job categories
- Expand Stop Words list
- Remove Punctuation
- Lemmatize
- Remove Stopwords
- Stem each word

```
# Drop any rows where job description is blank
indeed_df.dropna(subset=['job_desc'], inplace=True)

# Taking a sample of the indeed_df of the initial job cetegories that we were interested in
sample_df = indeed_df[indeed_df.job_category.isin(['Data Scientist', 'Data Analyst', 'Machine Learning Engineer', \
                                                   'Data Engineer','Business Intelligence Analyst', 'Data Architect'])].copy()

# Expand our stoplist
stoplist = ["year", "experience", "opportunity", "employer", "sexual", "orientation", "national", "origin"\
            "veteran", "status", "work", "race", "color", "religion", "orientation", "gender", "regard", "this", "position"\
            "related", "field", "closely", "with", "disability", "federal", "state", "local", "more", "than",\
            "receive", "equal", "opportunity", "veteran", "protect", "employment", "consideration", "accommodation",\
            "role", "employee", "benefit", "applicant", "such","what", "their", "company", "understand", "their",\
            "what", "minimum", "please", "which", "and/or", "vaccine", "vaccination", "understand", "their", "about", "what", \
            "write", "within", "and/or", "qualify", "prefer", "policy", "vaccination", "vaccine", "time", "qualification", \
            "qualify", "be", "without", "want", "prefer", "through", "like", "within", "inclusive", "inclusion", "organization",\
            "leave", "base", "package", "bring", "must", "candidate",  "without", "diversity", ] \
           + [word.lower() for company in indeed_df.company_name.unique() for word in company]
```

```
def preprocess(text):
    """ 1. Removes Punctuations
        2. Removes words smaller than 3 letters
        3. Converts into lowercase
        4. Lemmatizes words
        5. Removes Stopwords
    """
    punctuation= list(string.punctuation)
    doc_tokens= nltk.word_tokenize(text)
    word_tokens= [word.lower() for word in doc_tokens if not (word in punctuation or len(word)<=3)]
    doc_words=[wordnet_lemmatizer.lemmatize(word) for word in word_tokens]
    doc_words= [word for word in doc_words if word not in stoplist]
    stemmer=nltk.stem.PorterStemmer()
    doc_words=[stemmer.stem(word) for word in doc_words]

    return doc_words

job_desc_df_clean = sample_df['job_desc'].apply(preprocess)
```

K-Means Clustering

- We had 6 jobs so set to 6 clusters
- Cluster 0: Machine Learning Engineer
  - ['machin', 'machin learn', 'learn', 'engin', 'model', 'product', 'build', 'scienc', 'softwar', 'technolog', 'comput', 'solut', 'includ', 'problem', 'data scienc']
- Cluster 1: Data Architect
  - ['client', 'technolog', 'manag', 'help', 'process', 'inform', 'servic', 'includ', 'skill', 'solut', 'provid', 'need', 'support', 'peopl', 'analyt']
- Cluster 2: Data Scientist
  - ['data scienc', 'scienc', 'model', 'statist', 'learn', 'analyt', 'machin', 'machin learn', 'engin', 'analysi', 'product', 'use', 'solut', 'build', 'skill']
- Cluster 3: Data Analyst
  - ['report', 'manag', 'process', 'project', 'analysi', 'support', 'analyst', 'posit', 'provid', 'inform', 'abil', 'skill', 'includ', 'perform', 'document']
- Cluster 4: Data Engineer
  - ['engin', 'technolog', 'build', 'design', 'cloud', 'softwar', 'solut', 'product', 'platform', 'work', 'process', 'manag', 'servic', 'includ', 'support']
- Cluster 5: BI Analyst
  - ['analyt', 'insight', 'analysi', 'product', 'report', 'custom', 'market', 'skill', 'analyst', 'manag', 'abil', 'provid', 'support', 'commun', 'drive']

K-Means 6 Clusters after PCA