INTERGO: Platform Seeks to Bridge College to Dream Life with Machine Learning

Durham NC—April 23, 2022—Today Duke graduate student group “The Brainiacs” launched “INTERGO”, an information-and-help comprehensive platform that aims to help graduating and graduated students, who have limited experiences in living their life on their own, to plan their future better. The platform is for everyone, however it was designed with the kind motivation to especially assist international students who are placed in an informational and cultural disadvantaged position in navigating the U.S. society to narrow their knowledge gaps in aspects such as seeking jobs, getting involved in the society, making smarter life choices, etc. The platform uses machine learning to make predictions on the most struggling areas most international students would bump into, including setting an appropriate salary expectation, getting occupational advice, planning their money saving strategies, finding the most desirable residence, and VISA sponsorship requirements information. Reading in a PDF of a resume and extracting vital information, the system is able to pick out information on work experience (in years), experience types, and programming languages skills. And then it uses such information to estimate an hourly salary that is most given with such a combination of work experiences and skill sets.

Most young adults pursue their college degrees in hopes of landing a promising job, which translates into good pay and high life quality. However, not enough of them succeed in maximizing the gains a bachelor’s or graduate degree was believed to promise due to insufficient information sources, higher than expected life expenses, social isolation after relocating, etc. According to the University of Washington, studies show that 53% of college graduates are unemployed or working in a job that doesn't require a bachelor's degree. And on average, it takes college graduates three to six months to secure employment after graduation (2021). Roughly 27% of college graduates chose to pursue a graduate degree to boost their chances of getting a satisfactory job, but master’s degree holders do not necessarily know better about how to play their cards to maximize their gains (educationdata.org). There exists a huge knowledge gap among the graduates with any kinds of degrees where they would benefit greatly from being filled in with the most wanted Information on a single platform. INTERGO understands all the user pain points, and it is dedicated to bring all the wanted information onto the platform as well as offering predictions of salary and housing expenses according to user’s needs. No more jumping between apps and websites, INTERGO has it all covered here!

As of now, there has never been any platform specifically designed for helping graduating and newly graduated students to form a comprehensive prospect about their future job and future dream life. Jumping between apps and websites is exhausting and confusing, and information gaps between the apps and websites creates inconsistency in knowledge acquisition, making it almost impossible to make better decisions.

INTERGO is a free resource for all job-seeking newly graduates when registered with an .edu email address. With a $6.99 subscription fee per month, premium users gain access to expert columns where professionals share their insights on the industry as well as comments about specific aspects of life in their city. INTERGO also works with universities and job seeking websites such as LinkedIn and Handshake to add on the layers of housing information, salary estimation, and job recommendation for users' unique combination of skills and experiences.

INTERGO identifies “your” skill set and experiences, compares such information with its database, and gives out hourly salary estimates as well as job recommendation based on job postings asking for “your” abilities. It is also capable of giving advice on how the user could improve their competitiveness by suggesting languages to master or experiences they would benefit from owning. To care for the newly graduates’ life quality, INTERGO predicts rent in the interested city by using coordinates and real rent information. So the student will have a better understanding of how their life would be like in their interested position in their dream city in terms of the major life expenses and money saving strategies. INTERGO is able to help graduates plan a better life, and it will also help the society to become better informed, wealthier, and less anxious. Looking into the future, INTERGO will expand its service and offer help to any college students by including information about university information, such as ranking, tuition, local life, safety information, etc.

FAQs

Statistical considerations:

Q1: How does the pdf extraction work?

A1: We used the Python pdfplumber to read all the text in the PDF. According to the key words that are relative to skills, we extract the skills that the resume introduced. Based on time-relevant key works, such as years and months, we extract the time information and find out the time durations of work experiences. Then, based on skills, education level, and work experience periods, our program passes these unique sets of skills and experiences to the machine learning training models and finally predicts the salary of the user that is most seen and appropriate. In the future, we are going to expand more features and dimensions of our dataset, so that we can extract more information and skills that can further predict more precise rewards for users' resumes.

Q2: What is the source of data?

A2: What we did is quite innovative so there is no existing data for us to analyze, or at least few existing algorithms to predict salary are open to the public. Therefore, we need to scrape useful information from job search sites and build our own dataset. We currently focus on Indeed as our source of data. Indeed is the top 1 job site in the world with over 250 million unique visitors every month, and for every 10s, a job is added on Indeed globally, making it ideal for as a data provider. Our dataset will be updated monthly to improve the model.

Q3: How does the web scraper work?

A3: There is no nicely formatted job data on Indeed. Although the main part of URLs of jobs posting for analysts positions are the same, suffixes of them are different from each other, and there is no general pattern for suffixes. Therefore, we need to scrape links first, and then we scrape job data based on the links. We use SelectorGadget to interactively figure out what css selector we need to extract desired components from a page, for example, job description, wage, and location. We then used rvest as the scraper to get the raw dataset. For the job description part, we extracted important requirements such as working years, skills, and educational degree requirements of the job. The most difficult part is the processing of job description, because it does not have a uniform format, and the length of job description makes it even harder to extract information. For working years requirements, there are some assumptions that we need to make to simplify this process. We assume that the pattern "number + years" implies the required work experience of jobs based on the pattern we observed by manually reading through many job postings on Indeed. The number mentioned above could be either numeric or text version. We choose the maximum of all the numbers matching this pattern to be the required working experience.

Q4: How does the Machine Learning Model work?

A4: After the detection of user characteristics, such as the skills, education level, and working experience of the resume owner. We first do the feature engineering: An automatic pipline was designed to do the job. Then, a specific ML model is applied to predict the expected salary of the user. On top of that, we will give some advice on how to improve the resume to improve their competitiveness by, for example, suggesting some other skills to master. More specifically, the Catboost model is what we used for predicting the expected salary, more technical issues are welcomed to be asked.

Q5: Could you explain the training process?

A5: The training set is the data that we crawled from the webpage. Roughly speaking, there are 3 steps:

Data exploration and data cleaning: we first do the wholesome investigation and visualization of the data. We handled missing data, identified outliers, then we used histograms to study the distribution of resume features, and used a correlation matrix to identify the features that are best correlated with the label (i.e. salary). Then we split the train set and test set. Test set will not be touched until the final validation. The test set will not involve when we do feature engineering, a pipeline will be applied to the test set to complete the final test.

Feature Engineering: For numerical features such as working experience, we used the standard scaler to scale it, so that the gradient descent will be more efficient. For categorical features, we used one-hot-encoding. One problem with one-hot-encoding is that since the number of categorical features is large, the number of encoded features would be even larger, and there will be some features with little information. To solve this problem, we merged some features and deleted some features with few variations.

Model fitting and hyperparameter tuning: we applied five models: 1. OLS 2. Lasso 3. Ridge 4. Random Forest 5. CatBoost. The CatBoost regression performs best among the models and thus was picked to be the model we use.

Q6: Could you explain the hyperparameter tuning and model selection process?

A6: Here we will briefly explain the model that we used and the method of hyperparamter tuning.

1. The OLS/Lasso/Ridge: The models are standard and effective in our case, since there are a lot of categorical variables in the feature. It turns out that the Lasso regression produces the best result.

Hyperparameter tuning: There is no hyperparameter to tune for the OLS model, but the weight of the regularization term could be tuned for the lasso regression and the ridge regression. To find the best lambda, we used a grid search, and conducted a 10 fold cross validation. After the whole process, we pick the best estimator and calculate the mean squared error. Record the error for future reference.

It makes sense that the Lasso model performs the best. As we mentioned earlier, there could be many features that are not informative. The Lasso regression automatically selected the important features, thus having a good effect doing the prediction. The mean squared error on the test set of Lasso is 356.

2. The Random Forest: This tree based algorithm is expected to handle the regression problem with many categorical variables well. However, after we do a grid search of hyperparameter 'n_estimator' on {3,6,10,15,20,30}, and 'max_features' on {2,4,6,8,10}, with 10-fold cross validation, we find that the best estimator for random forest produces MSE of 384.67. This means that the random forest model performs worse than linear models on the data set, a possible explanation is that the random forest tends to overfit the training data.

3. The Catboost: Catboost: CatBoost is based on gradient boosted decision trees. During training, a set of decision trees is built consecutively. Each successive tree is built with reduced loss compared to the previous trees (https://catboost.ai/en/docs/concepts/algorithmmain-stages). More specifically, Catboost follows the steps:

(a) In each iteration, a decision tree is built with a limit of maximum depth. This tree is called the oblivious tree, with a small improvement from the last decision tree.

(b) If this tree has a smaller loss on the validation set compared to all former trees, the parameter for this model is recorded as the best solution.

(c) Catboost stops and returns the best tree models as the final model after certain iterations.

We made two attempts to tune the hyper-parameters of Catboost: First, with the CatBoostClassifier(plot = True), we tuned the parameter of depth to avoid overfitting; Second, we used Optuna package to automatically tune the parameter;

1. Manual tuning: To tune depth, training on parameter depth= [2, 3, 4, 5, 6, 7, 8] was conducted. The accuracy grows as the maximum depth grows, so should we just use depth = 8? No, because overfitting is one thing we should be concerned about. Observing the learning curve, when iteration is over 1000, the accuracy on the testing set is stable, while the error on the training set goes to zero – it is a pattern of overfitting.

2. Use Catboost: choosing target parameters as loss_function, learning_rate,l2_leaf_reg, colsample_bylevel, depth, boosting_type, bootstrap_type, min_data_in_leaf, one_hot_max_size, random_state, we use the optuna.create_study.optimize() function to let the program searching for best accuracy depending on these parameters, I set the maximize trails as 10000 and the training time is 1 hour. It will output the parameters that make the model have the least mean squared errors. With the best parameter found by Optuna, the MSE is reduced to 282. The performance of this model on the test set is 333.47.

Non-statistical considerations:

Q7: What is the motivation for developing this platform?

A7: We noticed that both graduating and newly-graduated students have trouble finding decent information about schools, local life of the schools, job information, and work sponsorship VISA info, etc. Sure, when we search for each kind of information on the Internet, we can find some related results (for example, U.S. News for school info), but there exists no compositive platform that combines all the information together and shows the clear result. Especially for international students, who experience a greater knowledge gap and usually are in a larger need of accurate information, such research is more nerve-wrecking. At this moment, most international students turn to acquaintances of the same nationalities as them to obtain job or life related information; however, such information does not guarantee accuracy or objectivity.

Q8: What else will you include in the platform?

A8: For the Job search service, now we have built a predictor where users can simply upload their resume pdf and the algorithm will output the expected salary for a data position. Later when we have more data position information from participating companies, we will offer job position recommendations based on the set of experiences a user has. Also, we can further give suggestions on personal development plans to help users get stronger. Later when we expand and open a university section for college students of any year, we will include information such as college ranking, tuition, courses, life expenses, etc. We will extract such information from college websites, and we will also invite alumni to share their precious insights of academic life or social life in a specific column. We look forward to refining the section of campus life, where it provides information on housing situations (price, crime rate, transportation system), local festivals, city vibes, and opportunities. For international students, we will add a visa information part for related content.

Q9: How do you attract consumers?

A9:

1: We authorize each participating school to edit their own parts besides scrapping information by our algorithms. For fairness concerns, we do not accept sponsorships of any colleges or companies. We invite all students to check this platform out for further information for their school life and after. Especially for international students, there are various student associations for all countries of origin, and we rely on them to advertise for us upon their satisfaction with our platform (we can sponsor those associations if it is needed).

2: Students who check the platform need to register with their .edu email addresses for free access. Users can post and discuss questions on the platform.

3: Since we have useful, concise, precise content, students will generally gather on this platform.

Q10: How will the platform get the profit?

A10: Ads, premium member (which is needed for further information and functions), universities, companies and agencies that are eager to collaborate on jobs seeking (similar to LinkedIn)

Q11: What will be the cost of this platform?

A11: Server maintenance fee, forum moderator update fee, platform update fee. Main team size of about 20 people is enough for running this platform.

Q12: What is the market size of this platform?

A12: Referencing to somewhat similar platforms, for example, 1point3acres, a Chinese website for information on studying abroad and jobs seeking in the U.S., which has about 1 million active users per month, and the revenue is about 5 million USD per year, we believe that our platform is looking at a market size around 15 million USD with over 1 million international students in the U.S. (35% of those are Chinese students) as our targeted users, and it can grow bigger with domestic students utilizing our platform as well.

Q13: Are there any potential competitors in the market?

A13: There are websites covering only one part of our services, such as LinkedIn for job applications, US NEWS for college rankings, and 1point3acres for Chinese international students experiences sharing.

Q14: How to compete with other potential competitors?

A14: We are the only comprehensive platform primarily focused on smarter job seeking strategies with salary estimation for each uploaded resume as well as developmental suggestions tailored for each user, and we also provide college information, campus life resources, and first-hand understanding of cities. The most benefited group of our platform would be international students who are poorly informed about resources for college, jobs, and life. However, we thrive to serve all college students of any nationalities or age. We provide multi-languages auto-translation, or we could even build servers for different languages so that students with similar backgrounds can discuss problems with ease.

Q15: What is the plan for the future development of this platform?

A15: Finish other useful functions stated above

Predict Salary

April 23, 2022

[8]:
```
pip install pdfplumber
```

```
Collecting pdfplumber
  Using cached pdfplumber-0.6.1-py3-none-any.whl (33 kB)
Collecting Wand>=0.6.7
  Using cached Wand-0.6.7-py2.py3-none-any.whl (139 kB)
Collecting pdfminer.six==20220319
  Using cached pdfminer.six-20220319-py3-none-any.whl (5.6 MB)
Collecting Pillow>=9.1
  Using cached Pillow-9.1.0-cp38-cp38-macosx_10_9_x86_64.whl (3.1 MB)
Requirement already satisfied: cryptography in
/Users/xuzikai/Downloads/anaconda3/lib/python3.8/site-packages (from
pdfminer.six==20220319->pdfplumber) (3.4.7)
Requirement already satisfied: chardet in
/Users/xuzikai/Downloads/anaconda3/lib/python3.8/site-packages (from
pdfminer.six==20220319->pdfplumber) (4.0.0)
Requirement already satisfied: cffi>=1.12 in
/Users/xuzikai/Downloads/anaconda3/lib/python3.8/site-packages (from
cryptography->pdfminer.six==20220319->pdfplumber) (1.14.5)
Requirement already satisfied: pycparser in
/Users/xuzikai/Downloads/anaconda3/lib/python3.8/site-packages (from
cffi>=1.12->cryptography->pdfminer.six==20220319->pdfplumber) (2.20)
Installing collected packages: Wand, Pillow, pdfminer.six, pdfplumber
  Attempting uninstall: Pillow
    Found existing installation: Pillow 8.2.0
    Uninstalling Pillow-8.2.0:
      Successfully uninstalled Pillow-8.2.0
Successfully installed Pillow-9.1.0 Wand-0.6.7 pdfminer.six-20220319
pdfplumber-0.6.1
Note: you may need to restart the kernel to use updated packages.
```

[9]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

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

0.1 The PDF to feature process

[10]:
```python
def getTextFromPdf(filename) :

  with pdfplumber.open(filename) as pdf:

    first_page = pdf.pages[0]
    this_resume = []
    line = ""
    for word in first_page.extract_text():
      line += word
      if (word == '\n'):
        this_resume.append(line)
        line = ""
    #print(first_page.extract_text());
    return this_resume
```

[ ]:
```python
def classify_features(textlist):
  #list the searchable words
  Feature_dict = {
      "C++": ["C++","c++"],
      "sql": ["sql","SQL"],
      "C" : ["withC","byC","C,","usingC","C."],
      "power": ["power"],
      "sas": ["SAS", "sas"],
      "micro": ["microsoft","office","excel"],
      "spss" :["spss"],
      "xml" :["xml"],
      "r" :["R,", "R.","RStudio"],
      "python":["python","Python"],
      "java" :["java","Java"],
      "comm" :["communication","communicative"],
      "tableau": ["tableau"],
      "Bachelor": ["Bachelor","bachelor"],
      "Graduate": ["Master","master","Doctor","doctor"]
  }
  #create the skill set map
  skill_dict = {
      "C++": 0,
      "sql": 0,
      "C" : 0,
      "power": 0,
      "sas": 0,
```

```
    "micro": 0,
    "spss" :0,
    "xml" :0,
    "r" :0,
    "python":0,
    "java" :0,
    "comm" :0,
    "tableau": 0,
    "expYear":0,
    "Bachelor":0,
    "Graduate":0
  }


  #search the resume and output the features
  for line in textlist:
    skill_dict = searchOneLine(line,Feature_dict,skill_dict);
  skill_dict["expYear"] = getWorkingExperience(textlist)
  return (skill_dict)



  #search one line
def searchOneLine(line,Feature_dict,skill_dict):
  for key in Feature_dict:
    for targetWord in Feature_dict[key]:
      if targetWord in line:
        if skill_dict[key] == 0:
          skill_dict[key] += 1
  return skill_dict

def getWorkingExperience(textlist):
  #list the accepted years as numbers
  yearStr = []
  for i in range(1999,2024):
    yearStr.append(str(i))
  #identify the months
  month = 1/12
  monthlist = {
    1*month : ["Jan","Jan.","January","1/","1."],
    2*month : ["Feb","Feb.","February","2/","2."],
    3*month : ["Mar", "Mar.", "March", "3/", "3."],
    4*month : ["Apr", "Apr.", "April", "April.", "4/", "4."],
    5*month : ["May", "May.", "5/", "5."],
    6*month : ["June", "June.", "6/", "6."],
    7*month : ["July", "July.", "7/", "7."],
    8*month : ["Aug", "August", "8/", "8."],
```

```
    9*month : ["Sep", "Sept", "September", "9/", "9."],
    10*month : ["Oct","October","10/"],
    11*month : ["Nov","November","11/"],
    12*month : ["Dec","December","12/"]

}

#identify the keyword working experience
workPart = False
workwords = ["work", "job", "intern", "fulltime","parttime"];
noworkwords = ["project", ]
for line in textlist:
  for word in workwords:
    if word in line:
      workPart = True

mytotalexp = 0
myyear = []
mymonth = []

for line in textlist:
  tempmonth = []
  for key in monthlist:
    for val in monthlist[key]:
      if val in line:
        tempmonth.append(key)
  if len(tempmonth)>1:
    for e in tempmonth:
      mymonth.append(e)

for line in textlist:
  tempyear = []
  for yr in yearStr:
    if yr in line:
      tempyear.append(int(yr))
  if len(tempmonth)>1:
    for e in tempyear:
      myyear.append(e)


for line in textlist:
  while len(mymonth)>1 and workPart == True:
    end_month = mymonth.pop(len(mymonth)-1)
    start_month = mymonth.pop(len(mymonth)-1)
    mytotalexp = mytotalexp + end_month - start_month

  while len(myyear)>1 and workPart == True:
```

```
        end_year = myyear.pop(len(myyear)-1)
        start_year = myyear.pop(len(myyear)-1)
        mytotalexp = mytotalexp + abs(end_year - start_year)


    return mytotalexp

textlist = getTextFromPdf('Zikai Xu - CV - 220321.pdf')
p = classify_features(textlist)

def get_feature(file):
    info = getWorkingExperience(file)
    info_out = []
    for i in␣
↪['power','c','micro','sas','spss','xml','etl','comm','python','java','r','tableau','spark',
↪
        info_out.append(info[i])
    return info_out
```

1 The model training process

1.1 Data exploration

[42]:
```
df_full = pd.read_csv('jobs.csv')
```

[43]:
```
df_full.head()
```

[43]:
```
   Unnamed: 0 job_remote                                            job_des  \
0           1     Remote  Apply Statistical and Machine Learning methods…
1          18     Remote  Looking to take the next step in your IT caree…
2          20     Remote  Are you an Excel Expert? Are you detail orient…
3          26     Remote  Company Info:\n\nWe help companies test and im…
4          39     Remote  DCI is a rapidly growing media company publish…

   job_loc   job_type  sql  power  c  micro  sas  …  python  java  r  \
0  unknown      mixed    0      0  0      0    0  …       0     0  0
1    south  Full time    0      0  0      0    0  …       0     0  0
2  unknown      mixed    0      0  0      1    0  …       0     0  0
3    south      mixed    0      0  0      0    0  …       0     0  0
4  unknown  Full time    1      0  0      0    0  …       0     0  0

   tableau  spark  working_years        degree  data_analyst  salary  review
0        1      0              0        Master             1   47.22       0
1        0      0              5        Master             1   74.41       2
2        0      0              0  NotSpecified             1   45.00       0
3        0      0              0        Master             1   50.00    1288
```

```
4         0       0               0       Bachelor              0   30.56       8

[5 rows x 24 columns]
```

[44]:
```
df_full.drop(columns = ['Unnamed: 0','review','job_des'], inplace = True)
```

[45]:
```
df_full.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 645 entries, 0 to 644
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   job_remote     645 non-null    object
 1   job_loc        645 non-null    object
 2   job_type       645 non-null    object
 3   sql            645 non-null    int64
 4   power          645 non-null    int64
 5   c              645 non-null    int64
 6   micro          645 non-null    int64
 7   sas            645 non-null    int64
 8   spss           645 non-null    int64
 9   xml            645 non-null    int64
 10  etl            645 non-null    int64
 11  comm           645 non-null    int64
 12  python         645 non-null    int64
 13  java           645 non-null    int64
 14  r              645 non-null    int64
 15  tableau        645 non-null    int64
 16  spark          645 non-null    int64
 17  working_years  645 non-null    int64
 18  degree         645 non-null    object
 19  data_analyst   645 non-null    int64
 20  salary         645 non-null    float64
dtypes: float64(1), int64(16), object(4)
memory usage: 105.9+ KB
```

[46]:
```
df_full.describe()
```

[46]:
```
              sql       power           c       micro         sas        spss  \
count  645.000000  645.000000  645.000000  645.000000  645.000000  645.000000
mean     0.258915    0.040310    0.040310    0.379845    0.029457    0.018605
std      0.438379    0.196838    0.196838    0.485725    0.169216    0.135229
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
50%      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
75%      1.000000    0.000000    0.000000    1.000000    0.000000    0.000000
```

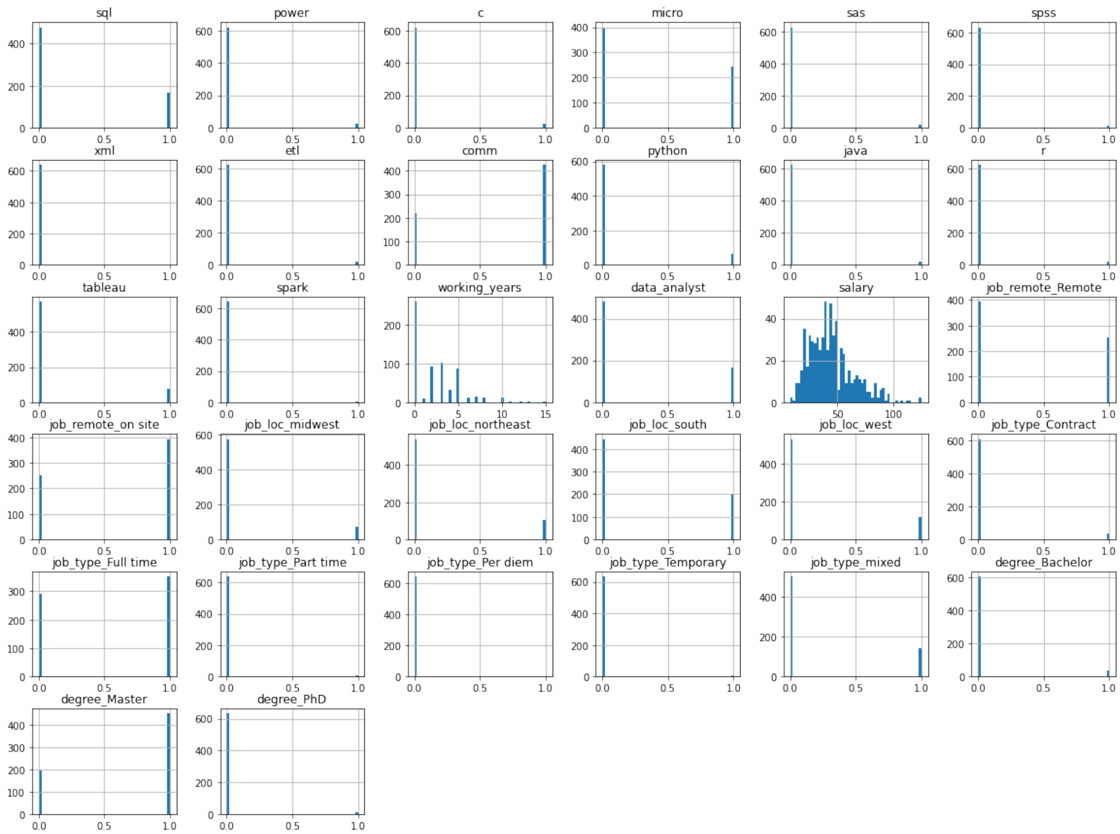
```
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

              xml         etl        comm      python        java           r  \
count  645.000000  645.000000  645.000000  645.000000  645.000000  645.000000
mean     0.007752    0.029457    0.660465    0.099225    0.032558    0.027907
std      0.087771    0.169216    0.473919    0.299196    0.177615    0.164834
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
50%      0.000000    0.000000    1.000000    0.000000    0.000000    0.000000
75%      0.000000    0.000000    1.000000    0.000000    0.000000    0.000000
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

          tableau       spark  working_years  data_analyst      salary
count  645.000000  645.000000     645.000000    645.000000  645.000000
mean     0.114729    0.003101       2.449612      0.255814   44.436558
std      0.318942    0.055641       2.703119      0.436656   19.401807
min      0.000000    0.000000       0.000000      0.000000    8.000000
25%      0.000000    0.000000       0.000000      0.000000   30.000000
50%      0.000000    0.000000       2.000000      0.000000   41.670000
75%      0.000000    0.000000       4.000000      1.000000   54.000000
max      1.000000    1.000000      15.000000      1.000000  125.000000
```

1.2 Feature engineering

[47]:
```
df_full_cat = df_full.loc[:,['job_remote','job_loc','job_type','degree']]
df_full_dum = pd.get_dummies(df_full_cat)
df_full_dum.drop(columns =␣
 ↪['job_loc_unknown','job_type_unknown','degree_NotSpecified'],inplace = True)
```

[48]:
```
df_prepared = pd.concat([df_full.drop(columns =␣
 ↪['job_remote','job_loc','job_type','degree']), df_full_dum],axis = 1)
```

[49]:
```
%matplotlib inline
df_prepared.hist(bins = 50, figsize = (20,15))
plt.show()
```



[50]:
```
# very less PhD samples, we will then merge Master and PhD as Graduate
df_prepared_copy = df_prepared
df_prepared['degree_Graduate'] =␣
↪df_prepared['degree_Master']+df_prepared['degree_PhD']
df_prepared.drop(columns = ['degree_Master','degree_PhD'], inplace = True)
```

[51]:
```
df_prepared.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 645 entries, 0 to 644
Data columns (total 31 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   sql                 645 non-null    int64
 1   power               645 non-null    int64
 2   c                   645 non-null    int64
 3   micro               645 non-null    int64
 4   sas                 645 non-null    int64
 5   spss                645 non-null    int64
 6   xml                 645 non-null    int64
 7   etl                 645 non-null    int64
```

```
 8   comm                645 non-null    int64
 9   python              645 non-null    int64
 10  java                645 non-null    int64
 11  r                   645 non-null    int64
 12  tableau             645 non-null    int64
 13  spark               645 non-null    int64
 14  working_years       645 non-null    int64
 15  data_analyst        645 non-null    int64
 16  salary              645 non-null    float64
 17  job_remote_Remote   645 non-null    uint8
 18  job_remote_on site  645 non-null    uint8
 19  job_loc_midwest     645 non-null    uint8
 20  job_loc_northeast   645 non-null    uint8
 21  job_loc_south       645 non-null    uint8
 22  job_loc_west        645 non-null    uint8
 23  job_type_Contract   645 non-null    uint8
 24  job_type_Full time  645 non-null    uint8
 25  job_type_Part time  645 non-null    uint8
 26  job_type_Per diem   645 non-null    uint8
 27  job_type_Temporary  645 non-null    uint8
 28  job_type_mixed      645 non-null    uint8
 29  degree_Bachelor     645 non-null    uint8
 30  degree_Graduate     645 non-null    uint8
dtypes: float64(1), int64(16), uint8(14)
memory usage: 94.6 KB
```

[52]:
```
## Create Test Set
train_set, test_set = train_test_split(df_prepared, test_size = 0.2,␣
 ↪random_state = 89)
```

[53]:
```
df = train_set.copy()
```

[54]:
```
corr_mat = df.corr()
from pandas.plotting import scatter_matrix
corr_mat['salary'].sort_values(ascending = False)
```

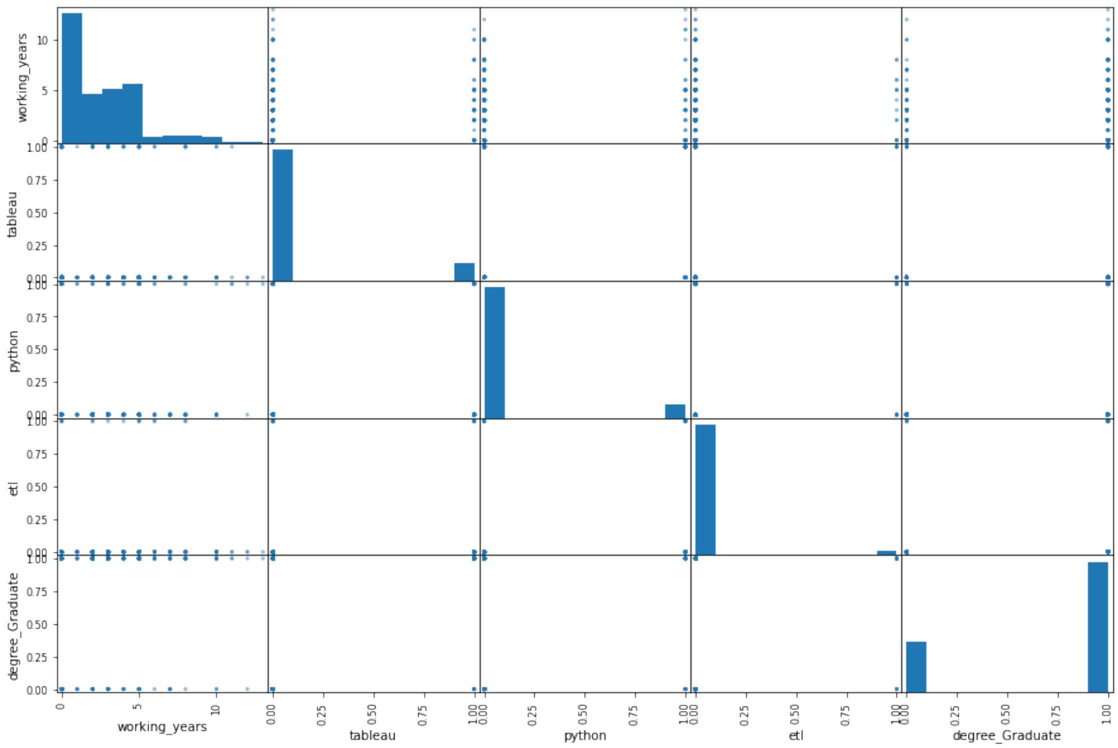
[54]:
```
salary                1.000000
working_years         0.234653
tableau               0.138667
python                0.125498
sql                   0.120978
etl                   0.117459
c                     0.091408
power                 0.091408
java                  0.090637
data_analyst          0.080519
spark                 0.064588
```

```
r                      0.044771
job_type_Part time     0.028053
job_loc_west           0.020144
degree_Graduate        0.017794
job_loc_northeast      0.006338
job_remote_on site     0.003699
job_loc_south         -0.002208
xml                   -0.002933
job_remote_Remote     -0.003699
comm                  -0.005677
job_type_Full time    -0.007815
sas                   -0.008594
spss                  -0.025560
degree_Bachelor       -0.026057
job_type_Contract     -0.030892
job_loc_midwest       -0.037513
job_type_mixed        -0.041313
job_type_Temporary    -0.063323
micro                 -0.076570
job_type_Per diem     -0.077557
Name: salary, dtype: float64
```

[55]:
```python
from pandas.plotting import scatter_matrix
features = ['working_years','tableau','python','etl','degree_Graduate']
scatter_matrix(df[features],figsize = (15,10))
```

[55]:
```
array([[<AxesSubplot:xlabel='working_years', ylabel='working_years'>,
        <AxesSubplot:xlabel='tableau', ylabel='working_years'>,
        <AxesSubplot:xlabel='python', ylabel='working_years'>,
        <AxesSubplot:xlabel='etl', ylabel='working_years'>,
        <AxesSubplot:xlabel='degree_Graduate', ylabel='working_years'>],
       [<AxesSubplot:xlabel='working_years', ylabel='tableau'>,
        <AxesSubplot:xlabel='tableau', ylabel='tableau'>,
        <AxesSubplot:xlabel='python', ylabel='tableau'>,
        <AxesSubplot:xlabel='etl', ylabel='tableau'>,
        <AxesSubplot:xlabel='degree_Graduate', ylabel='tableau'>],
       [<AxesSubplot:xlabel='working_years', ylabel='python'>,
        <AxesSubplot:xlabel='tableau', ylabel='python'>,
        <AxesSubplot:xlabel='python', ylabel='python'>,
        <AxesSubplot:xlabel='etl', ylabel='python'>,
        <AxesSubplot:xlabel='degree_Graduate', ylabel='python'>],
       [<AxesSubplot:xlabel='working_years', ylabel='etl'>,
        <AxesSubplot:xlabel='tableau', ylabel='etl'>,
        <AxesSubplot:xlabel='python', ylabel='etl'>,
        <AxesSubplot:xlabel='etl', ylabel='etl'>,
        <AxesSubplot:xlabel='degree_Graduate', ylabel='etl'>],
       [<AxesSubplot:xlabel='working_years', ylabel='degree_Graduate'>,
```

```
  <AxesSubplot:xlabel='tableau', ylabel='degree_Graduate'>,
  <AxesSubplot:xlabel='python', ylabel='degree_Graduate'>,
  <AxesSubplot:xlabel='etl', ylabel='degree_Graduate'>,
  <AxesSubplot:xlabel='degree_Graduate', ylabel='degree_Graduate'>]],
dtype=object)
```



1.3 Model Fitting and Hyperparameter tuning

[56]:
```python
X = df.drop('salary',axis = 1)
y = df.salary
```

[57]:
```python
# Feature Scaling
#from sklearn.preprocessing import StandardScaler
#std_scaler = StandardScaler()
#X_scaled = std_scaler.fit(X.working_years)
```

[58]:
```python
# Fit OLS, Ridge and Lasso
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn import linear_model
```

[59]:
```
# fit linear model, using Cross Validation to get est of MSE
lin_reg = LinearRegression()
lin_reg.fit(X,y)
lin_score = cross_val_score(lin_reg, X, y, scoring = 'neg_mean_squared_error',␣
↪cv = 10)
print(-lin_score.mean())
```

```
375.1110669796018
```

[60]:
```
# find best Lasso result through grid search
param_grid = {'alpha':np.linspace(0.1,5,num = 100)}
lasso = linear_model.Lasso()
grid_search = GridSearchCV(lasso, param_grid, cv = 5, scoring =␣
↪'neg_mean_squared_error', return_train_score = True)
grid_search.fit(X,y)
```

[60]:
```
GridSearchCV(cv=5, estimator=Lasso(),
             param_grid={'alpha': array([0.1       , 0.14949495, 0.1989899 ,
0.24848485, 0.2979798 ,
       0.34747475, 0.3969697 , 0.44646465, 0.4959596 , 0.54545455,
       0.59494949, 0.64444444, 0.69393939, 0.74343434, 0.79292929,
       0.84242424, 0.89191919, 0.94141414, 0.99090909, 1.04040404,
       1.08989899, 1.13939394, 1.18888889, 1.23838384, 1.28787879,
       1.33737374, 1.38686869, 1…
       3.56464646, 3.61414141, 3.66363636, 3.71313131, 3.76262626,
       3.81212121, 3.86161616, 3.91111111, 3.96060606, 4.01010101,
       4.05959596, 4.10909091, 4.15858586, 4.20808081, 4.25757576,
       4.30707071, 4.35656566, 4.40606061, 4.45555556, 4.50505051,
       4.55454545, 4.6040404 , 4.65353535, 4.7030303 , 4.75252525,
       4.8020202 , 4.85151515, 4.9010101 , 4.95050505, 5.        ])},
             return_train_score=True, scoring='neg_mean_squared_error')
```

[61]:
```
best_lasso = grid_search.best_estimator_
lasso_score = cross_val_score(best_lasso, X, y, scoring =␣
↪'neg_mean_squared_error', cv = 10)
print(-lasso_score.mean())
```

```
356.4805986123157
```

[62]:
```
# Find best Ridge
# find best Lasso result through grid search
param_grid = {'alpha':np.linspace(0.1,5,num = 100)}
ridge = Ridge()
grid_search = GridSearchCV(ridge, param_grid, cv = 5, scoring =␣
↪'neg_mean_squared_error', return_train_score = True)
grid_search.fit(X,y)
best_Ridge = grid_search.best_estimator_
```

```
ridge_score = cross_val_score(best_Ridge, X, y, scoring =␣
↪'neg_mean_squared_error', cv = 10)
print(-ridge_score.mean())
```

368.5679225808803

[63]:
```
# Random Forest
from sklearn.ensemble import RandomForestRegressor

param_grid = [{'n_estimators':[3,6,10,15,20,30], 'max_features':[2,4,6,8,10]}]

forest_reg = RandomForestRegressor()

grid_search = GridSearchCV(forest_reg, param_grid, cv = 5, scoring =␣
↪'neg_mean_squared_error', return_train_score = True)

grid_search.fit(X,y)

best_RF = grid_search.best_estimator_
RF_score = cross_val_score(best_RF, X, y, scoring = 'neg_mean_squared_error',␣
↪cv = 10)
print(-RF_score.mean())
```

384.6684924506811

[64]:
```
from catboost import CatBoostRegressor
Xtrain, Xtest, ytrain, ytest = train_test_split(X,y,test_size = 0.2)
catbst = CatBoostRegressor(iterations = 8000,
                                learning_rate = 0.005,
                                depth = 3,
                                loss_function='RMSE')
catbst.fit(Xtrain, ytrain)
```

```
0:      learn: 19.9075228       total: 308us    remaining: 2.47s
1:      learn: 19.9025419       total: 801us    remaining: 3.21s
2:      learn: 19.8960799       total: 1.06ms   remaining: 2.83s
3:      learn: 19.8868305       total: 1.32ms   remaining: 2.63s
4:      learn: 19.8820527       total: 1.63ms   remaining: 2.6s
5:      learn: 19.8759927       total: 1.95ms   remaining: 2.59s
6:      learn: 19.8694045       total: 2.31ms   remaining: 2.64s
7:      learn: 19.8675870       total: 2.62ms   remaining: 2.61s
8:      learn: 19.8599823       total: 2.9ms    remaining: 2.58s
9:      learn: 19.8562179       total: 3.24ms   remaining: 2.59s
10:     learn: 19.8485510       total: 3.56ms   remaining: 2.59s
11:     learn: 19.8430453       total: 3.9ms    remaining: 2.6s
12:     learn: 19.8378551       total: 4.25ms   remaining: 2.61s
13:     learn: 19.8320871       total: 4.61ms   remaining: 2.63s
```

```
7982:	learn: 14.5034908	total: 2.06s	remaining: 4.38ms
7983:	learn: 14.5034738	total: 2.06s	remaining: 4.12ms
7984:	learn: 14.5034568	total: 2.06s	remaining: 3.86ms
7985:	learn: 14.5034151	total: 2.06s	remaining: 3.6ms
7986:	learn: 14.5031957	total: 2.06s	remaining: 3.35ms
7987:	learn: 14.5028508	total: 2.06s	remaining: 3.09ms
7988:	learn: 14.5028262	total: 2.06s	remaining: 2.83ms
7989:	learn: 14.5027029	total: 2.06s	remaining: 2.57ms
7990:	learn: 14.5024312	total: 2.06s	remaining: 2.32ms
7991:	learn: 14.5024237	total: 2.06s	remaining: 2.06ms
7992:	learn: 14.5011063	total: 2.06s	remaining: 1.8ms
7993:	learn: 14.5010234	total: 2.06s	remaining: 1.54ms
7994:	learn: 14.5008698	total: 2.06s	remaining: 1.29ms
7995:	learn: 14.5004719	total: 2.06s	remaining: 1.03ms
7996:	learn: 14.5004530	total: 2.06s	remaining: 772us
7997:	learn: 14.5004244	total: 2.06s	remaining: 514us
7998:	learn: 14.5004146	total: 2.06s	remaining: 257us
7999:	learn: 14.4987832	total: 2.06s	remaining: 0us
```

[64]: <catboost.core.CatBoostRegressor at 0x7f9a2cd38670>

[65]:
```python
ypred = catbst.predict(Xtest)
print(mean_squared_error(ypred,ytest))
```

```
350.6206180806642
```

[66]:
```python
catbst = CatBoostRegressor(iterations = 100,
                                learning_rate = 0.01,
                                depth = 6,
                                loss_function='RMSE')
catbst.fit(Xtrain, ytrain)
```

```
0:	learn: 19.8979039	total: 707us	remaining: 70ms
1:	learn: 19.8803114	total: 1.71ms	remaining: 83.7ms
2:	learn: 19.8548641	total: 2.79ms	remaining: 90.3ms
3:	learn: 19.8423567	total: 4.49ms	remaining: 108ms
4:	learn: 19.8259774	total: 5.52ms	remaining: 105ms
5:	learn: 19.8068211	total: 6.42ms	remaining: 101ms
6:	learn: 19.7939793	total: 7.31ms	remaining: 97.2ms
7:	learn: 19.7766828	total: 10.3ms	remaining: 118ms
8:	learn: 19.7614935	total: 11.5ms	remaining: 116ms
9:	learn: 19.7458666	total: 12.5ms	remaining: 112ms
10:	learn: 19.7330321	total: 13.5ms	remaining: 109ms
11:	learn: 19.7199350	total: 14.5ms	remaining: 106ms
12:	learn: 19.7074847	total: 15.3ms	remaining: 103ms
13:	learn: 19.6894160	total: 16.2ms	remaining: 99.5ms
14:	learn: 19.6790437	total: 17.1ms	remaining: 96.9ms
15:	learn: 19.6647843	total: 18ms	remaining: 94.3ms
```

```
16:	learn: 19.6447586	total: 18.8ms	remaining: 91.6ms
17:	learn: 19.6313810	total: 19.7ms	remaining: 89.6ms
18:	learn: 19.6163325	total: 20.7ms	remaining: 88.4ms
19:	learn: 19.6102553	total: 21.1ms	remaining: 84.3ms
20:	learn: 19.5969158	total: 22ms	remaining: 82.6ms
21:	learn: 19.5823164	total: 22.9ms	remaining: 81.3ms
22:	learn: 19.5690396	total: 23.7ms	remaining: 79.5ms
23:	learn: 19.5480123	total: 24.7ms	remaining: 78.1ms
24:	learn: 19.5335077	total: 25.5ms	remaining: 76.6ms
25:	learn: 19.5200792	total: 26.3ms	remaining: 74.8ms
26:	learn: 19.5060411	total: 27.1ms	remaining: 73.2ms
27:	learn: 19.4860633	total: 27.9ms	remaining: 71.7ms
28:	learn: 19.4723247	total: 28.9ms	remaining: 70.6ms
29:	learn: 19.4616021	total: 29.7ms	remaining: 69.2ms
30:	learn: 19.4452553	total: 30.4ms	remaining: 67.7ms
31:	learn: 19.4320486	total: 31.2ms	remaining: 66.3ms
32:	learn: 19.4201045	total: 32ms	remaining: 64.9ms
33:	learn: 19.4003171	total: 32.7ms	remaining: 63.6ms
34:	learn: 19.3899136	total: 33.5ms	remaining: 62.3ms
35:	learn: 19.3759470	total: 34.2ms	remaining: 60.9ms
36:	learn: 19.3645036	total: 35ms	remaining: 59.6ms
37:	learn: 19.3537344	total: 35.8ms	remaining: 58.4ms
38:	learn: 19.3454975	total: 36.6ms	remaining: 57.3ms
39:	learn: 19.3374436	total: 37.5ms	remaining: 56.3ms
40:	learn: 19.3284522	total: 38.4ms	remaining: 55.3ms
41:	learn: 19.3170928	total: 39.2ms	remaining: 54.1ms
42:	learn: 19.3048297	total: 40ms	remaining: 53ms
43:	learn: 19.2967004	total: 40.7ms	remaining: 51.8ms
44:	learn: 19.2856606	total: 41.4ms	remaining: 50.6ms
45:	learn: 19.2777595	total: 42.1ms	remaining: 49.4ms
46:	learn: 19.2619740	total: 42.9ms	remaining: 48.3ms
47:	learn: 19.2564477	total: 43.5ms	remaining: 47.2ms
48:	learn: 19.2478886	total: 44.2ms	remaining: 46ms
49:	learn: 19.2359215	total: 44.9ms	remaining: 44.9ms
50:	learn: 19.2255298	total: 45.6ms	remaining: 43.8ms
51:	learn: 19.2183327	total: 46ms	remaining: 42.5ms
52:	learn: 19.2080319	total: 46.6ms	remaining: 41.4ms
53:	learn: 19.1951552	total: 47.3ms	remaining: 40.3ms
54:	learn: 19.1863373	total: 47.9ms	remaining: 39.2ms
55:	learn: 19.1769050	total: 48.6ms	remaining: 38.2ms
56:	learn: 19.1682357	total: 49.2ms	remaining: 37.1ms
57:	learn: 19.1594766	total: 49.9ms	remaining: 36.1ms
58:	learn: 19.1352597	total: 50.5ms	remaining: 35.1ms
59:	learn: 19.1255674	total: 51ms	remaining: 34ms
60:	learn: 19.1188710	total: 51.8ms	remaining: 33.1ms
61:	learn: 19.1121025	total: 52.4ms	remaining: 32.1ms
62:	learn: 19.0974208	total: 53ms	remaining: 31.1ms
63:	learn: 19.0838291	total: 53.6ms	remaining: 30.2ms
```

```
64:     learn: 19.0778225       total: 54.1ms   remaining: 29.1ms
65:     learn: 19.0632779       total: 54.7ms   remaining: 28.2ms
66:     learn: 19.0412788       total: 55.4ms   remaining: 27.3ms
67:     learn: 19.0309707       total: 56.1ms   remaining: 26.4ms
68:     learn: 19.0226340       total: 56.7ms   remaining: 25.5ms
69:     learn: 19.0121968       total: 57.4ms   remaining: 24.6ms
70:     learn: 18.9988301       total: 58ms     remaining: 23.7ms
71:     learn: 18.9906503       total: 58.6ms   remaining: 22.8ms
72:     learn: 18.9826825       total: 59.3ms   remaining: 21.9ms
73:     learn: 18.9651244       total: 59.9ms   remaining: 21ms
74:     learn: 18.9505975       total: 60.5ms   remaining: 20.2ms
75:     learn: 18.9397722       total: 61.1ms   remaining: 19.3ms
76:     learn: 18.9327343       total: 61.8ms   remaining: 18.5ms
77:     learn: 18.9230108       total: 62.4ms   remaining: 17.6ms
78:     learn: 18.9163420       total: 63ms     remaining: 16.8ms
79:     learn: 18.9090288       total: 63.8ms   remaining: 16ms
80:     learn: 18.8987837       total: 64.5ms   remaining: 15.1ms
81:     learn: 18.8839309       total: 65.1ms   remaining: 14.3ms
82:     learn: 18.8775600       total: 65.8ms   remaining: 13.5ms
83:     learn: 18.8664445       total: 66.4ms   remaining: 12.7ms
84:     learn: 18.8618307       total: 66.9ms   remaining: 11.8ms
85:     learn: 18.8469761       total: 67.5ms   remaining: 11ms
86:     learn: 18.8377611       total: 68.2ms   remaining: 10.2ms
87:     learn: 18.8287747       total: 68.8ms   remaining: 9.38ms
88:     learn: 18.8181491       total: 69.5ms   remaining: 8.59ms
89:     learn: 18.8052399       total: 70.1ms   remaining: 7.79ms
90:     learn: 18.7923177       total: 70.8ms   remaining: 7ms
91:     learn: 18.7797292       total: 71.4ms   remaining: 6.21ms
92:     learn: 18.7750977       total: 72.1ms   remaining: 5.43ms
93:     learn: 18.7626564       total: 72.7ms   remaining: 4.64ms
94:     learn: 18.7506154       total: 73.4ms   remaining: 3.86ms
95:     learn: 18.7413441       total: 74.1ms   remaining: 3.08ms
96:     learn: 18.7354536       total: 74.7ms   remaining: 2.31ms
97:     learn: 18.7273633       total: 75.4ms   remaining: 1.54ms
98:     learn: 18.7163793       total: 76ms     remaining: 767us
99:     learn: 18.7113187       total: 76.3ms   remaining: 0us
```

[66]: <catboost.core.CatBoostRegressor at 0x7f9a15f02d60>

[83]:
```python
def objective(trial):
    param = {
        "loss_function": trial.suggest_categorical("loss_function", ['RMSE']),
        "learning_rate": trial.suggest_loguniform("learning_rate", 1e-5, 1e0),
        "l2_leaf_reg": trial.suggest_loguniform("l2_leaf_reg", 1e-2, 1e0),
        "colsample_bylevel": trial.suggest_float("colsample_bylevel", 0.01, 0.
 ↪1),
        "depth": trial.suggest_int("depth", 1, 5),
```

```
        "boosting_type": trial.suggest_categorical("boosting_type", ["Ordered",␣
↪"Plain"]),
        "bootstrap_type": trial.suggest_categorical("bootstrap_type",␣
↪["Bayesian", "Bernoulli", "MVS"]),
        "min_data_in_leaf": trial.suggest_int("min_data_in_leaf", 2, 20),
        "one_hot_max_size": trial.suggest_int("one_hot_max_size", 2, 20),
    }
    # Conditional Hyper-Parameters
    if param["bootstrap_type"] == "Bayesian":
        param["bagging_temperature"] = trial.
↪suggest_float("bagging_temperature", 0, 10)
    elif param["bootstrap_type"] == "Bernoulli":
        param["subsample"] = trial.suggest_float("subsample", 0.1, 1)

    catboost = CatBoostRegressor(**param)
    catboost.fit(Xtrain, ytrain, eval_set=[(Xtest, ytest)], verbose=0,␣
↪early_stopping_rounds=100)
    y_pred = catboost.predict(Xtest)
    score = mean_squared_error(ytest, y_pred, squared=False)
    return score
```

[85]:
```
import optuna
from optuna.samplers import TPESampler
study = optuna.create_study(sampler=TPESampler(), direction="minimize")
study.optimize(objective, n_trials=30, timeout=3600) # Run for 90 minutes
print("Number of completed trials: {}".format(len(study.trials)))
print("Best trial:")
trial = study.best_trial

print("\tBest Score: {}".format(trial.value))
print("\tBest Params: ")
for key, value in trial.params.items():
    print("    {}: {}".format(key, value))
```

```
[I 2022-04-23 17:02:21,047] A new study created in memory with name:
no-name-af8a65be-dabc-40a0-913d-fccded9df285
[I 2022-04-23 17:02:21,378] Trial 0 finished with value:
17.549968781271282 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.0003378570171085431, 'l2_leaf_reg': 0.012254218236650332, 'colsample_bylevel':
0.032668843427496146, 'depth': 4, 'boosting_type': 'Ordered', 'bootstrap_type':
'Bayesian', 'min_data_in_leaf': 8, 'one_hot_max_size': 8, 'bagging_temperature':
4.301687272024077}. Best is trial 0 with value: 17.549968781271282.
[I 2022-04-23 17:02:22,085] Trial 1 finished with value:
17.5763222992879 and parameters: {'loss_function': 'RMSE', 'learning_rate':
3.6864705397327204e-05, 'l2_leaf_reg': 0.5792632174205926, 'colsample_bylevel':
0.05869233775057044, 'depth': 5, 'boosting_type': 'Ordered', 'bootstrap_type':
'MVS', 'min_data_in_leaf': 19, 'one_hot_max_size': 10}. Best is trial 0 with
```

```
value: 17.549968781271282.
[I 2022-04-23 17:02:22,209] Trial 2 finished with value:
17.5044864936749 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.0007975638735147769, 'l2_leaf_reg': 0.1533459102133016, 'colsample_bylevel':
0.05051815404615827, 'depth': 1, 'boosting_type': 'Plain', 'bootstrap_type':
'MVS', 'min_data_in_leaf': 12, 'one_hot_max_size': 18}. Best is trial 2 with
value: 17.5044864936749.
[I 2022-04-23 17:02:22,258] Trial 3 finished with value:
17.046889504859703 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.09020174542550316, 'l2_leaf_reg': 0.2849502164604349, 'colsample_bylevel':
0.031946547252376355, 'depth': 5, 'boosting_type': 'Plain', 'bootstrap_type':
'Bernoulli', 'min_data_in_leaf': 12, 'one_hot_max_size': 20, 'subsample':
0.37446384894206686}. Best is trial 3 with value: 17.046889504859703.
[I 2022-04-23 17:02:22,349] Trial 4 finished with value:
17.57812377595791 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.0007838668299726621, 'l2_leaf_reg': 0.1206029142268605, 'colsample_bylevel':
0.01644138432422239, 'depth': 1, 'boosting_type': 'Ordered', 'bootstrap_type':
'Bernoulli', 'min_data_in_leaf': 6, 'one_hot_max_size': 3, 'subsample':
0.307027425621644}. Best is trial 3 with value: 17.046889504859703.
[I 2022-04-23 17:02:22,380] Trial 5 finished with value:
16.861965968204252 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.09648029250604823, 'l2_leaf_reg': 0.08622986288994734, 'colsample_bylevel':
0.03753199142597104, 'depth': 5, 'boosting_type': 'Plain', 'bootstrap_type':
'MVS', 'min_data_in_leaf': 17, 'one_hot_max_size': 2}. Best is trial 5 with
value: 16.861965968204252.
[I 2022-04-23 17:02:22,542] Trial 6 finished with value:
17.259238669180803 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.0021349834246203936, 'l2_leaf_reg': 0.02254972640198342, 'colsample_bylevel':
0.06795844681347124, 'depth': 2, 'boosting_type': 'Plain', 'bootstrap_type':
'Bayesian', 'min_data_in_leaf': 10, 'one_hot_max_size': 14,
'bagging_temperature': 0.07596266306382882}. Best is trial 5 with value:
16.861965968204252.
[I 2022-04-23 17:02:22,745] Trial 7 finished with value:
17.58417106781951 and parameters: {'loss_function': 'RMSE', 'learning_rate':
1.988707731314606e-05, 'l2_leaf_reg': 0.1570397343129713, 'colsample_bylevel':
0.086407863542927, 'depth': 1, 'boosting_type': 'Ordered', 'bootstrap_type':
'Bernoulli', 'min_data_in_leaf': 19, 'one_hot_max_size': 5, 'subsample':
0.695657575920305}. Best is trial 5 with value: 16.861965968204252.
[I 2022-04-23 17:02:22,796] Trial 8 finished with value:
17.053859616473506 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.08968114773867607, 'l2_leaf_reg': 0.3713464675612779, 'colsample_bylevel':
0.08842139061460393, 'depth': 5, 'boosting_type': 'Plain', 'bootstrap_type':
'Bernoulli', 'min_data_in_leaf': 18, 'one_hot_max_size': 8, 'subsample':
0.5364640023977428}. Best is trial 5 with value: 16.861965968204252.
[I 2022-04-23 17:02:23,185] Trial 9 finished with value:
17.395072037060295 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.00149487653735321, 'l2_leaf_reg': 0.013110337341140942, 'colsample_bylevel':
0.041094082363043107, 'depth': 4, 'boosting_type': 'Ordered', 'bootstrap_type':
```

```
'Bernoulli', 'min_data_in_leaf': 6, 'one_hot_max_size': 11, 'subsample':
0.5479479089828684}. Best is trial 5 with value: 16.861965968204252.
[I 2022-04-23 17:02:23,222] Trial 10 finished with value:
17.5934493511617 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.801984898611091, 'l2_leaf_reg': 0.041698584455644616, 'colsample_bylevel':
0.021477356875551745, 'depth': 3, 'boosting_type': 'Plain', 'bootstrap_type':
'MVS', 'min_data_in_leaf': 2, 'one_hot_max_size': 2}. Best is trial 5 with
value: 16.861965968204252.
[I 2022-04-23 17:02:23,296] Trial 11 finished with value:
17.08432813492705 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.029961611465174877, 'l2_leaf_reg': 0.05279982777675785, 'colsample_bylevel':
0.032605655946926805, 'depth': 5, 'boosting_type': 'Plain', 'bootstrap_type':
'MVS', 'min_data_in_leaf': 14, 'one_hot_max_size': 20}. Best is trial 5 with
value: 16.861965968204252.
[I 2022-04-23 17:02:23,348] Trial 12 finished with value:
17.34601004299848 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.03902740724610852, 'l2_leaf_reg': 0.27246786582289095, 'colsample_bylevel':
0.010461473592230368, 'depth': 4, 'boosting_type': 'Plain', 'bootstrap_type':
'MVS', 'min_data_in_leaf': 15, 'one_hot_max_size': 15}. Best is trial 5 with
value: 16.861965968204252.
[I 2022-04-23 17:02:23,393] Trial 13 finished with value:
17.540818104661525 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.564856331473834, 'l2_leaf_reg': 0.8674072984389363, 'colsample_bylevel':
0.04358145250398163, 'depth': 3, 'boosting_type': 'Plain', 'bootstrap_type':
'Bernoulli', 'min_data_in_leaf': 14, 'one_hot_max_size': 15, 'subsample':
0.12617967779686118}. Best is trial 5 with value: 16.861965968204252.
[I 2022-04-23 17:02:23,499] Trial 14 finished with value:
17.418633121911974 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.011909644622894688, 'l2_leaf_reg': 0.06779565938763005, 'colsample_bylevel':
0.02771674893185637, 'depth': 5, 'boosting_type': 'Plain', 'bootstrap_type':
'Bayesian', 'min_data_in_leaf': 16, 'one_hot_max_size': 6,
'bagging_temperature': 9.905760653657312}. Best is trial 5 with value:
16.861965968204252.
[I 2022-04-23 17:02:23,551] Trial 15 finished with value:
17.012805991093412 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.2780320177769871, 'l2_leaf_reg': 0.29845493337482637, 'colsample_bylevel':
0.06569184054050402, 'depth': 4, 'boosting_type': 'Plain', 'bootstrap_type':
'MVS', 'min_data_in_leaf': 12, 'one_hot_max_size': 18}. Best is trial 5 with
value: 16.861965968204252.
[I 2022-04-23 17:02:23,611] Trial 16 finished with value:
17.104106742153192 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.26628098053416904, 'l2_leaf_reg': 0.08664469488447243, 'colsample_bylevel':
0.07148399109674884, 'depth': 4, 'boosting_type': 'Plain', 'bootstrap_type':
'MVS', 'min_data_in_leaf': 17, 'one_hot_max_size': 13}. Best is trial 5 with
value: 16.861965968204252.
[I 2022-04-23 17:02:23,731] Trial 17 finished with value:
17.065868508011654 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.007738437263460231, 'l2_leaf_reg': 0.03224342033984833, 'colsample_bylevel':
```

0.07740083880225576, 'depth': 4, 'boosting_type': 'Plain', 'bootstrap_type': 'MVS', 'min_data_in_leaf': 20, 'one_hot_max_size': 17}. Best is trial 5 with value: 16.861965968204252.
[I 2022-04-23 17:02:23,777] Trial 18 finished with value: 17.06371949483876 and parameters: {'loss_function': 'RMSE', 'learning_rate': 0.19911422894039876, 'l2_leaf_reg': 0.19963509575169142, 'colsample_bylevel': 0.0995314805632734, 'depth': 3, 'boosting_type': 'Plain', 'bootstrap_type': 'MVS', 'min_data_in_leaf': 9, 'one_hot_max_size': 9}. Best is trial 5 with value: 16.861965968204252.
[I 2022-04-23 17:02:23,929] Trial 19 finished with value: 17.0874105365219 and parameters: {'loss_function': 'RMSE', 'learning_rate': 0.008233377635203279, 'l2_leaf_reg': 0.5491767593820972, 'colsample_bylevel': 0.05976934645990018, 'depth': 2, 'boosting_type': 'Plain', 'bootstrap_type': 'MVS', 'min_data_in_leaf': 13, 'one_hot_max_size': 12}. Best is trial 5 with value: 16.861965968204252.
[I 2022-04-23 17:02:23,973] Trial 20 finished with value: 17.38683020712067 and parameters: {'loss_function': 'RMSE', 'learning_rate': 0.9543264519754303, 'l2_leaf_reg': 0.09780718854743747, 'colsample_bylevel': 0.04979907042014245, 'depth': 4, 'boosting_type': 'Plain', 'bootstrap_type': 'MVS', 'min_data_in_leaf': 16, 'one_hot_max_size': 5}. Best is trial 5 with value: 16.861965968204252.
[I 2022-04-23 17:02:24,041] Trial 21 finished with value: 17.229121499414436 and parameters: {'loss_function': 'RMSE', 'learning_rate': 0.06580216624007723, 'l2_leaf_reg': 0.28501064053976777, 'colsample_bylevel': 0.03993845686636933, 'depth': 5, 'boosting_type': 'Plain', 'bootstrap_type': 'Bernoulli', 'min_data_in_leaf': 11, 'one_hot_max_size': 20, 'subsample': 0.9814785860436358}. Best is trial 5 with value: 16.861965968204252.
[I 2022-04-23 17:02:24,098] Trial 22 finished with value: 17.22023552957454 and parameters: {'loss_function': 'RMSE', 'learning_rate': 0.15239250630639797, 'l2_leaf_reg': 0.3829543129086724, 'colsample_bylevel': 0.02604045422955189, 'depth': 5, 'boosting_type': 'Plain', 'bootstrap_type': 'Bernoulli', 'min_data_in_leaf': 12, 'one_hot_max_size': 18, 'subsample': 0.2852950980571411}. Best is trial 5 with value: 16.861965968204252.
[I 2022-04-23 17:02:24,185] Trial 23 finished with value: 17.38333967875333 and parameters: {'loss_function': 'RMSE', 'learning_rate': 0.025557078848293613, 'l2_leaf_reg': 0.2073574667822005, 'colsample_bylevel': 0.06481602839890459, 'depth': 5, 'boosting_type': 'Plain', 'bootstrap_type': 'Bayesian', 'min_data_in_leaf': 9, 'one_hot_max_size': 17, 'bagging_temperature': 9.876978744200954}. Best is trial 5 with value: 16.861965968204252.
[I 2022-04-23 17:02:24,229] Trial 24 finished with value: 17.034500796824 and parameters: {'loss_function': 'RMSE', 'learning_rate': 0.32990487936507024, 'l2_leaf_reg': 0.7648803870692583, 'colsample_bylevel': 0.049496631465620174, 'depth': 4, 'boosting_type': 'Plain', 'bootstrap_type': 'MVS', 'min_data_in_leaf': 6, 'one_hot_max_size': 20}. Best is trial 5 with value: 16.861965968204252.
[I 2022-04-23 17:02:24,276] Trial 25 finished with value: 17.023184088067357 and parameters: {'loss_function': 'RMSE', 'learning_rate':

```
0.2653919788675027, 'l2_leaf_reg': 0.9927586850416963, 'colsample_bylevel':
0.051920671964858695, 'depth': 4, 'boosting_type': 'Plain', 'bootstrap_type':
'MVS', 'min_data_in_leaf': 3, 'one_hot_max_size': 18}. Best is trial 5 with
value: 16.861965968204252.
[I 2022-04-23 17:02:24,350] Trial 26 finished with value:
16.943308579682697 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.3316987141850231, 'l2_leaf_reg': 0.8835741583407805, 'colsample_bylevel':
0.05527897496840224, 'depth': 3, 'boosting_type': 'Ordered', 'bootstrap_type':
'MVS', 'min_data_in_leaf': 3, 'one_hot_max_size': 16}. Best is trial 5 with
value: 16.861965968204252.
[I 2022-04-23 17:02:24,412] Trial 27 finished with value:
16.585494359253513 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.447280153026815, 'l2_leaf_reg': 0.4972512431115233, 'colsample_bylevel':
0.07658527362012574, 'depth': 2, 'boosting_type': 'Ordered', 'bootstrap_type':
'MVS', 'min_data_in_leaf': 4, 'one_hot_max_size': 16}. Best is trial 27 with
value: 16.585494359253513.
[I 2022-04-23 17:02:24,749] Trial 28 finished with value:
17.558133774307493 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.00010107738126193682, 'l2_leaf_reg': 0.5374782279642493, 'colsample_bylevel':
0.08013881405379375, 'depth': 2, 'boosting_type': 'Ordered', 'bootstrap_type':
'MVS', 'min_data_in_leaf': 4, 'one_hot_max_size': 15}. Best is trial 27 with
value: 16.585494359253513.
[I 2022-04-23 17:02:24,814] Trial 29 finished with value:
17.28571937588651 and parameters: {'loss_function': 'RMSE', 'learning_rate':
0.5396688681155565, 'l2_leaf_reg': 0.6740984282106933, 'colsample_bylevel':
0.09942591051179951, 'depth': 2, 'boosting_type': 'Ordered', 'bootstrap_type':
'Bayesian', 'min_data_in_leaf': 4, 'one_hot_max_size': 13,
'bagging_temperature': 0.21955196000607202}. Best is trial 27 with value:
16.585494359253513.

Number of completed trials: 30
Best trial:
        Best Score: 16.585494359253513
        Best Params:
    loss_function: RMSE
    learning_rate: 0.447280153026815
    l2_leaf_reg: 0.4972512431115233
    colsample_bylevel: 0.07658527362012574
    depth: 2
    boosting_type: Ordered
    bootstrap_type: MVS
    min_data_in_leaf: 4
    one_hot_max_size: 16
```

[67]:
```
ypred = catbst.predict(Xtest)
print(mean_squared_error(ypred,ytest))
```

```
282.9270486589537
```

[68]:
```
## Using the best predictor on the test set
df_test = test_set.copy()
yt = df_test.salary
Xt = df_test
ypred = catbst.predict(test_set)
print(mean_squared_error(ypred,yt))
```

```
333.47732658651654
```

[88]:
```
selected_ctbst = CatBoostRegressor(loss_function = 'RMSE',
                                   learning_rate= 0.0447280153026815,
                                   l2_leaf_reg= 0.4972512431115233,
                                   colsample_bylevel= 0.07658527362012574,
                                   depth= 6,
                                   boosting_type = 'Ordered',
                                   bootstrap_type= 'MVS',
                                   min_data_in_leaf= 4,
                                   one_hot_max_size= 16)
```

[89]:
```
selected_ctbst.fit(Xtrain, ytrain)
ypred = selected_ctbst.predict(Xtest)
print(mean_squared_error(ypred,ytest))
```

```
0:      learn: 19.9020852       total: 389us    remaining: 389ms
1:      learn: 19.8451534       total: 2.09ms   remaining: 1.04s
2:      learn: 19.7847157       total: 2.85ms   remaining: 948ms
3:      learn: 19.7594849       total: 3.49ms   remaining: 868ms
4:      learn: 19.7473243       total: 3.95ms   remaining: 786ms
5:      learn: 19.6675887       total: 8.26ms   remaining: 1.37s
6:      learn: 19.6199515       total: 9.23ms   remaining: 1.31s
7:      learn: 19.5920989       total: 11.5ms   remaining: 1.42s
8:      learn: 19.5292874       total: 13.1ms   remaining: 1.44s
9:      learn: 19.4721260       total: 16.8ms   remaining: 1.66s
10:     learn: 19.3973078       total: 20.3ms   remaining: 1.82s
11:     learn: 19.3973059       total: 20.6ms   remaining: 1.7s
12:     learn: 19.3886192       total: 21ms     remaining: 1.59s
13:     learn: 19.3648143       total: 23ms     remaining: 1.62s
14:     learn: 19.3544353       total: 23.3ms   remaining: 1.53s
15:     learn: 19.3213221       total: 25.7ms   remaining: 1.58s
16:     learn: 19.3141076       total: 26.1ms   remaining: 1.51s
17:     learn: 19.2455051       total: 30.4ms   remaining: 1.66s
18:     learn: 19.2455027       total: 30.7ms   remaining: 1.59s
19:     learn: 19.2181473       total: 34.4ms   remaining: 1.69s
20:     learn: 19.1659889       total: 38.1ms   remaining: 1.77s
21:     learn: 19.1287410       total: 39.8ms   remaining: 1.77s
22:     learn: 19.0924666       total: 42ms     remaining: 1.78s
23:     learn: 19.0318738       total: 46ms     remaining: 1.87s
24:     learn: 19.0300420       total: 46.9ms   remaining: 1.83s
```

```
937:    learn: 14.1526364       total: 1.37s    remaining: 90.6ms
938:    learn: 14.1522511       total: 1.37s    remaining: 89.1ms
939:    learn: 14.1522479       total: 1.37s    remaining: 87.6ms
940:    learn: 14.1504093       total: 1.38s    remaining: 86.2ms
941:    learn: 14.1485101       total: 1.38s    remaining: 84.9ms
942:    learn: 14.1484827       total: 1.38s    remaining: 83.3ms
943:    learn: 14.1484818       total: 1.38s    remaining: 81.8ms
944:    learn: 14.1483922       total: 1.38s    remaining: 80.3ms
945:    learn: 14.1477985       total: 1.38s    remaining: 78.9ms
946:    learn: 14.1473834       total: 1.39s    remaining: 77.6ms
947:    learn: 14.1404334       total: 1.39s    remaining: 76.1ms
948:    learn: 14.1404153       total: 1.39s    remaining: 74.5ms
949:    learn: 14.1404030       total: 1.39s    remaining: 73ms
950:    learn: 14.1309848       total: 1.39s    remaining: 71.5ms
951:    learn: 14.1309358       total: 1.39s    remaining: 70ms
952:    learn: 14.1308173       total: 1.39s    remaining: 68.7ms
953:    learn: 14.1308173       total: 1.39s    remaining: 67.2ms
954:    learn: 14.1278613       total: 1.4s     remaining: 65.8ms
955:    learn: 14.1275908       total: 1.4s     remaining: 64.3ms
956:    learn: 14.1269524       total: 1.4s     remaining: 62.9ms
957:    learn: 14.1258976       total: 1.4s     remaining: 61.4ms
958:    learn: 14.1031714       total: 1.4s     remaining: 60ms
959:    learn: 14.0988593       total: 1.41s    remaining: 58.6ms
960:    learn: 14.0981015       total: 1.41s    remaining: 57.1ms
961:    learn: 14.0976700       total: 1.41s    remaining: 55.7ms
962:    learn: 14.0960785       total: 1.41s    remaining: 54.3ms
963:    learn: 14.0948816       total: 1.42s    remaining: 52.9ms
964:    learn: 14.0947225       total: 1.42s    remaining: 51.4ms
965:    learn: 14.0899201       total: 1.42s    remaining: 50ms
966:    learn: 14.0899020       total: 1.42s    remaining: 48.5ms
967:    learn: 14.0895315       total: 1.42s    remaining: 47ms
968:    learn: 14.0808818       total: 1.43s    remaining: 45.6ms
969:    learn: 14.0800334       total: 1.43s    remaining: 44.2ms
970:    learn: 14.0800294       total: 1.43s    remaining: 42.7ms
971:    learn: 14.0776394       total: 1.43s    remaining: 41.2ms
972:    learn: 14.0663848       total: 1.43s    remaining: 39.8ms
973:    learn: 14.0657293       total: 1.43s    remaining: 38.3ms
974:    learn: 14.0650232       total: 1.44s    remaining: 36.9ms
975:    learn: 14.0640920       total: 1.44s    remaining: 35.4ms
976:    learn: 14.0640852       total: 1.44s    remaining: 33.9ms
977:    learn: 14.0640160       total: 1.44s    remaining: 32.5ms
978:    learn: 14.0640160       total: 1.44s    remaining: 30.9ms
979:    learn: 14.0640160       total: 1.44s    remaining: 29.4ms
980:    learn: 14.0634507       total: 1.45s    remaining: 28ms
981:    learn: 14.0538782       total: 1.45s    remaining: 26.6ms
982:    learn: 14.0538699       total: 1.45s    remaining: 25.1ms
983:    learn: 14.0534197       total: 1.45s    remaining: 23.6ms
984:    learn: 14.0493153       total: 1.45s    remaining: 22.1ms
```

```
985:    learn: 14.0489884       total: 1.45s    remaining: 20.6ms
986:    learn: 14.0489678       total: 1.45s    remaining: 19.1ms
987:    learn: 14.0489678       total: 1.45s    remaining: 17.6ms
988:    learn: 14.0489668       total: 1.45s    remaining: 16.2ms
989:    learn: 14.0484443       total: 1.45s    remaining: 14.7ms
990:    learn: 14.0450998       total: 1.46s    remaining: 13.2ms
991:    learn: 14.0450712       total: 1.46s    remaining: 11.8ms
992:    learn: 14.0450591       total: 1.46s    remaining: 10.3ms
993:    learn: 14.0446795       total: 1.46s    remaining: 8.82ms
994:    learn: 14.0389503       total: 1.46s    remaining: 7.36ms
995:    learn: 14.0383921       total: 1.47s    remaining: 5.88ms
996:    learn: 14.0376311       total: 1.47s    remaining: 4.41ms
997:    learn: 14.0304565       total: 1.47s    remaining: 2.94ms
998:    learn: 14.0289662       total: 1.47s    remaining: 1.47ms
999:    learn: 14.0288183       total: 1.47s    remaining: 0us
332.4508871141036
```

1.4 The Final Model

[ ]:
```
selected_ctbst = CatBoostRegressor(loss_function = 'RMSE',
                                   learning_rate= 0.0447280153026815,
                                   l2_leaf_reg= 0.4972512431115233,
                                   colsample_bylevel= 0.07658527362012574,
                                   depth= 6,
                                   boosting_type = 'Ordered',
                                   bootstrap_type= 'MVS',
                                   min_data_in_leaf= 4,
                                   one_hot_max_size= 16)
```

web scraper

Yesen Chen

2022-04-24

```
library(stringr)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v forcats 0.5.1
## v readr   2.1.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
remotes::install_github("rundel/hayalbaz")
```

```
## Skipping install of 'hayalbaz' from a github remote, the SHA1 (fd92db54) has not changed since last i
##   Use `force = TRUE` to force installation
```

```
remotes::install_github("rstudio/chromote")
```

```
## Skipping install of 'chromote' from a github remote, the SHA1 (1884e3bd) has not changed since last i
##   Use `force = TRUE` to force installation
```

```
library(hayalbaz)
library(chromote)
library(rvest)
```

```
##
## Attaching package: 'rvest'
## The following object is masked from 'package:readr':
##
##     guess_encoding
```

```
library(devtools)
```

```
## Loading required package: usethis
```

```
########## A function to transform word to number, however, this can't handle word larger than 15
w2n <- function(num_word){
  switch(
    num_word,
    "zero" = 0,
    "one" = 1,
    "two" = 2,
    "three" = 3,
    "four" = 4,
```

```
    "five" = 5,
    "six" = 6,
    "seven" = 7,
    "eight" = 8,
    "nine" = 9,
    "ten" = 10,
    "eleven" = 11,
    "twelve" = 12,
    "thirteen" = 13,
    "fourteen" = 14,
    "fifteen" = 15
  )
}
##########

########## A function to extract the number of years from a job description
get_year <- function(jd){
  loc <- str_locate_all(jd, "years")[[1]]

  # If the word "years" is not detected, we regard the job do not require working experience, i.e. work
  if (length(loc) == 0){
    return (0)
  }

  year_vec <- c()

  for (i in 1:nrow(loc)){
    year_tmp <- substr(jd, loc[i, 1] - 10, loc[i, 2]) %>% # Go backward 10 characters to search the num
      str_extract_all("[0-9]+") %>%
      {.[[1]]}

    # If no pattern like [0-9]+ is detected, we continue to detect number written in word like "one", "
    if (length(year_tmp) == 0){
      year_tmp <- substr(jd, loc[i, 1] - 10, loc[i, 2]) %>%
        str_to_lower() %>%
        str_extract_all(" zero | one | two | three | four | five | six | seven | eight | nine | ten | el
        {.[[1]]} %>%
        str_trim()

      # If also no pattern like "one", "two", "three" ... was detected, year_tmp will be a character(0)
      year_tmp <- ifelse(length(year_tmp) == 0, 0, w2n(year_tmp))
    }

    year_vec <- c(year_vec, as.numeric(year_tmp))
  }

  year_vec <- year_vec[year_vec <= 15] # Cut off at 15, because some "years" are not requirement for ap

  return(max(c(year_vec, 0)))
}
##########

########## Functions to extract the required degree and skill sets from a job description
```

```r
detect_sql <- function(des) {
  return(str_detect(des, regex("sql", ignore_case = TRUE)))
}
detect_power <- function(des) {
  return(str_detect(des, regex("power bi | powerbi", ignore_case = TRUE)))
}
detect_c <- function(des) {
  return(str_detect(des, regex("c++ | C++ | C language", ignore_case = TRUE)))
}
detect_micro <- function(des) {
  return(str_detect(des, regex("microsoft | excel | word | powerpoint | power point",
                               ignore_case = TRUE)))
}
detect_sas <- function(des) {
  return(str_detect(des, regex("SAS")))
}
detect_spss <- function(des) {
  return(str_detect(des, regex("SPSS")))
}
detect_xml <- function(des) {
  return(str_detect(des, regex("XML")))
}
detect_etl <- function(des) {
  return(str_detect(des, regex("ETL")))
}
detect_comm <- function(des) {
  return(str_detect(des, regex("communicat", ignore_case = TRUE)))
}
detect_python <- function(des) {
  return(str_detect(des, regex("python", ignore_case = TRUE)))
}
detect_java <- function(des) {
  return(str_detect(des, regex("java", ignore_case = TRUE)))
}
detect_r <- function(des) {
  return(str_detect(des, regex(" R | R language")))
}
detect_tableau <- function(des) {
  return(str_detect(des, regex("tableau", ignore_case = TRUE)))
}
detect_spark <- function(des) {
  return(str_detect(des, regex("spark", ignore_case = TRUE)))
}

detect_bachelor <- function(des) {
  return(str_detect(des, regex("BA | BS | Bachelor", ignore_case = TRUE)))
}

detect_master <- function(des){
  return(str_detect(des, regex("MS | MA | master", ignore_case = TRUE)))
}

detect_phd <- function(des){
```

```r
  return(str_detect(des, regex("phd | Ph.D. |doctor", ignore_case = TRUE)))
}
##########

########## A function used to detect if this job is related to data analyst
detect_data <- function(title) {
  return(str_detect(title, regex("data",ignore_case = TRUE)))
}
##########

########## These four functions detect salary units, four possible values:year, month, day, hour
detect_year <- function(salary) {
  return(str_detect(salary, regex("year",ignore_case = TRUE)))
}
detect_month <- function(salary) {
  return(str_detect(salary, regex("month",ignore_case = TRUE)))
}
detect_day <- function(salary) {
  return(str_detect(salary, regex("day",ignore_case = TRUE)))
}
detect_hour <- function(salary) {
  return(str_detect(salary, regex("hour",ignore_case = TRUE)))
}
##########

########## A function to transform salary unit to hour
get_salary <- function(salary) {
  salary=str_replace_all(salary, ",", "")
  Msalary=max(as.numeric(str_extract_all(salary,"[0-9]+")[[1]]))
  if (detect_year(salary)==1){
    Msalary=Msalary/1800
  }
  if (detect_month(salary)==1){
    Msalary=Msalary/160
  }
  if (detect_day(salary)==1){
    Msalary=Msalary/8
  }
  return (Msalary)
}
##########

########## A function to get the number of reviews
get_review=function(review) {
  review=str_replace_all(review, ",", "")
  review=as.numeric(str_extract_all(review,"[0-9]+")[[1]])
  return (review)
}
##########

########## A function to get the mixed job type
detect_mixed=function(type){
  return(str_detect(type,"-"))
```

```
}
##########

## code to prepare `tidy_data` dataset goes here


#n: the number of pages

scrape_data <- function(n){
  test <- puppet$new()
  # 10 items per page
  starts <- seq(0, n, by = 10)
  full_links <- tibble()
  for(start1 in starts) {
    cat(start1, "\n")
    url <- paste0("https://www.indeed.com/jobs?q=analyst&start=", start1)
    test$goto(url)
    test$wait_on_load()

    a <- test$get_source()

    links <- a %>%
      read_html() %>%
      html_nodes(".mosaic-zone a") %>%
      html_attr("href") %>%
      tibble(link = .) %>%
      filter(str_detect(link, "&vjs="))

    full_links <- rbind(full_links, links)
  }
  # remove duplicated values
  job_links <- tibble(link = unique(full_links$link))
  # add prefix to each link
  job_links <- job_links %>%
    mutate(link = paste0("https://www.indeed.com", link))
  # table used to store each job's information
  job_table <- data.frame()
  for (i in seq(1, nrow(job_links))){
    link <- job_links[i,] %>% pull()
    job_page <- read_html(link)
    tmp <- data.frame(
      job_title = ifelse(length(job_page %>%
                                  html_elements(".jobsearch-JobInfoHeader-title") %>%
                                  html_text2()) == 0, NA, job_page %>%
                           html_elements(".jobsearch-JobInfoHeader-title") %>%
                           html_text2()),
      job_salary = ifelse(length(job_page %>%
                                   html_elements(".jobsearch-JobMetadataHeader-item .icl-u-xs-mr--xs") %
                                   html_text2()) == 0, NA, job_page %>%
                            html_elements(".jobsearch-JobMetadataHeader-item .icl-u-xs-mr--xs") %>%
                            html_text2()),
      job_reviews = ifelse(length(job_page %>%
                                    html_elements(".icl-Ratings-link .icl-Ratings-count") %>%
```

```
                                    html_text2()) == 0, 0, job_page %>%
                             html_elements(".icl-Ratings-link .icl-Ratings-count") %>%
                             html_text2()),
      job_remote = ifelse(length(job_page %>%
                                   html_elements(".jobsearch-DesktopStickyContainer-companyrating~ div+
                                   html_text2()) == 0, "on site", job_page %>%
                            html_elements(".jobsearch-DesktopStickyContainer-companyrating~ div+ div")
                            html_text2()),
      job_des = ifelse(length(job_page %>%
                                html_elements("#jobDescriptionText") %>%
                                html_text2()) == 0, NA, job_page %>%
                         html_elements("#jobDescriptionText") %>%
                         html_text2()),
      job_loc = ifelse(length(job_page %>%
                                html_elements(".jobsearch-DesktopStickyContainer-companyrating+ div") %
                                html_text2()) == 0, NA, job_page %>%
                         html_elements(".jobsearch-DesktopStickyContainer-companyrating+ div") %>%
                         html_text2()),
      job_type = ifelse(length(job_page %>%
                                 html_elements(" .jobsearch-JobMetadataHeader-item.icl-u-xs-mt--xs") %>
                                 html_text2()) == 0, NA, job_page %>%
                          html_elements(" .jobsearch-JobMetadataHeader-item.icl-u-xs-mt--xs") %>%
                          html_text2())

    )
    job_table <- rbind(job_table, tmp)
  }
  # remove NA
  job_table <- job_table[!is.na(job_table$job_salary),]
  # remove duplicated values (based on description)
  job_table <- job_table[!duplicated(job_table$job_des), ]
  return(job_table)
}


#data: a data frame contains job related information (title, salary, review, remote, description, locat

data_preprocessing <- function(data){
  # extract working years, skill sets and required degree from job description
  descr <- data$job_des
  working_years <- c()
  skill_data <- data.frame()
  degree_data <- data.frame()
  for (i in seq_along(descr)) {
    sql <- data.frame(sql = ifelse(detect_sql(descr[i]), 1, 0))
    power <- data.frame(power = ifelse(detect_power(descr[i]), 1, 0))
    c <- data.frame(c = ifelse(detect_power(descr[i]), 1, 0))
    micro <- data.frame(micro = ifelse(detect_micro(descr[i]), 1, 0))
    sas <- data.frame(sas = ifelse(detect_sas(descr[i]), 1, 0))
    spss <- data.frame(spss = ifelse(detect_spss(descr[i]), 1, 0))
    xml <- data.frame(xml = ifelse(detect_xml(descr[i]), 1, 0))
    etl <- data.frame(etl = ifelse(detect_etl(descr[i]), 1, 0))
    # communication
```

```r
  comm <- data.frame(comm = ifelse(detect_comm(descr[i]), 1, 0))
  python <- data.frame(python = ifelse(detect_python(descr[i]), 1, 0))
  java <- data.frame(java = ifelse(detect_java(descr[i]), 1, 0))
  r <- data.frame(r = ifelse(detect_r(descr[i]), 1, 0))
  tableau <- data.frame(tableau = ifelse(detect_tableau(descr[i]), 1, 0))
  spark <- data.frame(spark = ifelse(detect_spark(descr[i]), 1, 0))
  skill <- cbind(sql, power, c, micro, sas, spss, xml, etl, comm, python, java,
               r, tableau, spark)
  skill_data <- rbind(skill_data, skill)

  bachelor <- data.frame(bachelor = ifelse(detect_bachelor(descr[i]), 1, 0))
  master <- data.frame(master = ifelse(detect_master(descr[i]), 2, 0))
  Phd <- data.frame(Phd = ifelse(detect_phd(descr[i]), 3, 0))
  degree <- cbind(bachelor, master, Phd)
  degree_data <- rbind(degree_data, degree)

  working_years <- c(working_years, get_year(descr[i]))
}
# append new columns
new_data <- cbind(data, skill_data, degree_data, working_years)
# create a new column degree and remove original ones
# this new column is categorical and it contains four values:0, 1, 2, 3
# 0 means no degree requirement; 1 means highest required degree is Bachelor
# 2 means highest required degree is Master; 3 means highest required degree is PhD
new_data <- new_data %>%
  mutate(degree = apply(degree_data, MARGIN = 1, max)) %>%
  select(-bachelor, -master, -Phd)
# transform salary unit into hour; extract the number of reviews; identify if it is a data related jo
new_data <- new_data %>%
  mutate(data_analyst=as.numeric(sapply(job_title,detect_data, USE.NAMES = FALSE)),
         salary=round(sapply(job_salary,get_salary, USE.NAMES = FALSE), 2),
         review=sapply(job_reviews,get_review, USE.NAMES = FALSE))
# change job_remote to two types
new_data$job_remote[new_data$job_remote=="Temporarily remote"] <- "Remote"
# change job_type
new_data$job_type[is.na(new_data$job_type)] <- "unknown"
new_data$job_type[new_data$job_type=="- Full-time"] <- "Full time"
new_data$job_type[new_data$job_type=="- Contract"] <- "Contract"
new_data$job_type[new_data$job_type=="- Part-time"] <- "Part time"
new_data$job_type[new_data$job_type=="- Per diem"] <- "Per diem"
new_data$job_type[new_data$job_type=="- Temporary"] <- "Temporary"
new_data$job_type[sapply(new_data$job_type,detect_mixed)] <- "mixed"
# change job_loc to their own states
new_data$job_loc[sapply(new_data$job_loc,FUN=function(i){return (str_detect(i,"DC"))})] <- "DC"
for (abb in state.abb){
  new_data$job_loc[sapply(new_data$job_loc,FUN=function(i){return (str_detect(i,abb))})] <- abb
}
for (j in seq_along(state.name)){
  new_data$job_loc[sapply(new_data$job_loc,FUN=function(i){
    return (str_detect(i,state.name[j]))})] <- state.abb[j]
}
# classify each state to a specific area
west <- c("WA","OR","CA","NV","UT","AZ","ID","MT","WY","CO","NM","AK","HI")
```

```
  midwest <- c("ND","SD","NE","KS","MN","IA","MO","WI","IL","IN","MI","OH")
  south <- c("TX","OK","AR","LA","MS","AL","TN","KY","WV","MD","DE","DC","VA","NC","SC","GA","FL")
  northeast <- c("PA","NY","NJ","CT","RI","MA","VT","NH","ME")
  new_data$job_loc[new_data$job_loc %in% west] <- "west"
  new_data$job_loc[new_data$job_loc %in% midwest] <- "midwest"
  new_data$job_loc[new_data$job_loc %in% south] <- "south"
  new_data$job_loc[new_data$job_loc %in% northeast] <- "northeast"
  # If job location is not in these four areas, we set it to "unknown"
  new_data$job_loc[!(new_data$job_loc %in% c("west", "midwest", "south", "northeast"))] <- "unknown"
  # remove redundant columns
  new_data <- new_data %>%
    select(-job_title, -job_salary, -job_reviews)
  # Transform categorical variables from numerical type to factor type
  degree_set <- c("NotSpecified", "Bachelor", "Master", "PhD")
  new_data <- new_data %>%
    mutate(degree = degree_set[degree + 1] %>% factor()) # Transform the number notation of degree to w

  # transfer variables into factor except working_years, salary and review
  new_data[, -c(which(colnames(new_data) == "working_years"),
              which(colnames(new_data) == "salary"),
              which(colnames(new_data) == "review"))] <- new_data[, -c(which(colnames(new_data) == "wo
                                                                 which(colnames(new_data) == "sa
                                                                 which(colnames(new_data) == "re
                                                          lapply(as.factor)
  return (new_data)
}
```