How to scrape Glassdoor for Company Reviews, Jobs & Salary Data ?

Bruno Helmeczy

14/12/2020

Introduction

The following assignment is in fulfilment of the Coding 2: Web scraping with R course at the Central European University's MSc in Business Analytics curriculum. The assignment was to choose a website related to a topic of interest & scrape "most of the website", at least resulting in providing solutions to a use-case question. My topic of interest is conceived in my personal challenges as a graduate student, new to data science & analytics.

The 'Finding a (good) job as a student' Problem:

Even during my undergraduate studies, I found it difficult to efficiently search job postings online: Many similar but different websites, all wanting you to provide login details, your email address & fill many search parameters, for which they send push notifications, only to find the jobs returned at best tangentially match your interests. Yet, even if websites returned a few relevant job for you, in reality you might need to send out 200 applications until you actually got a job you are satisfied with.

Accepting a job you are happy with however, already assumes you researched company culture, know what the company's internal challenges are, can picture yourself working there, & by the way you successfully negotiated a salary you & your employer are both satisfied with. Now, you can probably tick these off your to-do-list after few years' work experience (or you don't have to, because jobs find you at that point). As a student however, I can read company websites (telling what the company wants you to know, not what e.g. other employees think), maybe 10-20 reviews & ask for 1-2 salary examples from my network, but in essence as a young graduate you are most likely uninformed. It is what I felt at least applying to a Graduate Analyst position in London, facing the question 'What's your expected salary?' I mean, what is a fair amount & how would I know? To summarize the problem, here are the questions I am trying to answer:

1) How do I efficiently find jobs that are actually suitable for me?
2) How do I find out how it is to be working at companies where I found jobs?
3) How do I come up with an expected salary figure?

Formulating a Solution

Now, the term 'suitable' is quite ambiguous & not by accident. You may be able to move countries, or not. You might look to shift careers & you probably want something to match your experience & skill set. These are important search parameters we'll get back to. Finding company reviews once you found jobs, is not the biggest challenge, however you still need to make sense of text data (I outlined a framework for it here). Estimating an expected salary figure however is a matter of both company policy, labour market forces in the area the job is & your experience / skillset.

To formulate a feasible scraping project answering these questions, we will need a multi-step framework. On the one hand, getting all information for all jobs in all companies of all sectors is very computer intensive & vulnerable to errors (the 1000s of read_html requests you'll send will only get your IP address blocked). On the other hand, it also takes way too long & 90% of what you scraped will be irrelevant to your situation. Jobs in Europe won't help you if you must stay in the US, & Senior Software Engineering positions & salaries are not of interest to you as a Business Analytics student with no experience. With these considerations in mind, I bullet point my solution approach below, & show functions to scrape Glassdoor in R in the next section. I plan a web-based solution in the future too. As a final note, I start each function with a brief description on what does the function do / achieve. If you're looking for details, read the paragraphs below too.

1) Find the top X companies (based on the sites' Rating) in selected industry sectors
2) Find Z (optionally most recent) jobs posted for (a subset of) these X companies
3) Find job requirements & description for jobs subset by keyword & location
4) Find company reviews for the companies of your interest
5) Find salary comparison data based on selected job titles, locations & experience

Codes

Finding Companies

So what does the function do? This function posts httr get requests to Glassdoor as part of a double-loop, collecting high-level company data from a given number of pages from the keyword-searched chosen industry sector(s). I most importantly collect companies' shortened name & IDs on Glassdoor, enabling me to recreate their main landing page urls on Glassdoor, also using wildcard characters in the process.

This 1st function generates a series of httr requests for Glassdoor, for the website to return the Names, IDs & importantly, URLs to their Glassdoor overview pages. Function inputs are 1st How many companies to return from any of the selected sectors, & 2nd a (vector of) character string(s), which is searched for among the **SectorTable** data frame, made up of Sector names & their respective IDs. I Felt it simpler to return 100 companies per request, & filter the resulting data frame at the very end, since this request is pretty efficient in comparison to the rest of the functions. I made a double-for-loop to look for the number of pages (resulting from the number of companies requested) for every chosen industry sector.

"https://www.glassdoor.com/Overview/Working-at-Oracle-EI_IE1737.11,17.htm"

The json response is converted using fromJSON(), with the resulting list containing the results data frame, in it the IDs & shortened company names, however not urls to their main pages. The glassdoor url structure is as below. In this case, for Oracle (also its shortname) afterwards, a consistent "EI_IE", then their id number, 1737, then ".11" & a random number. Except the last number (in this case 17), everything else can be specified, & wildcard match with ** ensures the correct url is found.

```
library(rvest)
library(data.table)
library(stringr)
library(jsonlite)
library(XML)
library(dplyr)
library(httr)

GetCompURLs <- function(HowManyCompaniesperSector, WhichSectors) {
  Sectors <- c("Business Services","Construction"
               ,"Aerospace Defense","InfoTech"
               ,"Government","Transport & Logistics"
```

```
              ,"Retail","TeleCommunications"
              ,"HealthCare","Finance"
              ,"Biotech","Consumer Services"
              ,"Media","Accounting & Legal"
              ,"Education","Manufacturing","Arts")
SectorIDs <- paste0(100,c( "06","07","02","13","11"
                          ,"24","22","23","12","10"
                          ,"05","08","16","01","09","15","04"))
SectorTable <- as.data.frame(cbind(Sectors,SectorIDs))

# Define How Many Companies to select & for which industry
HowManyPages <- round(HowManyCompaniesperSector,-2)/100
ChosenIDs <- SectorTable[grep(tolower(WhichSectors)
                              ,tolower(SectorTable$Sectors)),2]

CompanyURLdf <- data.frame()

headers = c(
  `authority` = 'www.glassdoor.com',
  `user-agent` = 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4
  `content-type` = 'application/json',
  `accept` = '*/*',
  `sec-fetch-site` = 'same-origin',
  `sec-fetch-mode` = 'cors',
  `sec-fetch-dest` = 'empty',
  `referer` = 'https://www.glassdoor.com/',
  `accept-language` = 'hu-HU,hu;q=0.9,en-US;q=0.8,en;q=0.7',
  `cookie` = 'JSESSIONID=CDE85AAC2C4E4E7E9236CB6102796332; GSESSIONID=CDE85AAC2C4E4E7E9236CB610279633
)

for (i in 1:HowManyPages) {
  for (j in 1:length(ChosenIDs)) {

    params = list(
      `minRating` = '3.5',
      `maxRating` = '5',
      `numPerPage` = "100",
      `pageRequested` = as.character(i),
      `domain` = 'glassdoor.com',
      `surgeHiring` = 'false',
      `sectorIds` = ChosenIDs[j]
    )
    Res <- httr::GET(url = 'https://www.glassdoor.com/seo/ajax/ugcSearch.htm'
                     , httr::add_headers(.headers=headers), query = params)
    df <-fromJSON( content(res, 'text'))

    Res <- df$employerSearchResponse$results %>%
      select(id,shortName)

    Res$shortName <- gsub(" ","-",Res$shortName)
    Res$url <- paste0("https://www.glassdoor.com/Overview/Working-at-"
                      ,Res$shortName,"-EI_IE",Res$id,".11***.htm")
```

```
      Res$Sector <- SectorTable[which(SectorTable$SectorIDs %in%
                                        params$sectorIds),"Sectors"]

      CompanyURLdf <- rbind(CompanyURLdf,Res)
    }
  }

CompanyURLdf <- unique(CompanyURLdf)
CompanyURLdf$index <- c(1:length(CompanyURLdf$shortName))
 return(head(CompanyURLdf,HowManyCompaniesperSector*length(ChosenIDs)))
}
```

Find Recently Posted Jobs & Descriptions

So what does the function do? This function collects job details from the 1st X number of pages of the Jobs section of chosen companies. There are 40 jobs on 1 page, for 1 company. Collected variables are Job titles, companies, location, links, job requirements & the complete job descriptions.

The get jobs function takes an input of urls (+ the number of pages to get & for which companies), the same as the outputs from the GetCompUrls function. I allow the give vector of index numbers or company names for the WhichCcompanies argument to subset the data frame given as an input. I avoid error possibilities by checking if the number of companies passed as an input, match the number of companies that were recognized. If it matches, I use again a double-for-loop to collect all jobs from the designated number of pages per company.

I start with reading the html links passed as inputs, use the xpath "//*[@id='EIProductHeaders']/div/a[2]" to get the link to the companies jobs section (note, 1-5 instead of the last number lead to the 5 main sub-pages: Reviews, Jobs, Salaries, Interviews & Benefits). I then create links for each sub-jobpage, which I loop through with the 2nd loop & first find the 40 'Box'-es containing all info I need in 1 page. I first find how many days ago a job was posted, the lapply through each Box, collecting into a list the Position title, the Company, City location, City State & Links to the detailed job description for every job posted. I simultaneously rbindlist() the list I collected everything into & column bind it together with the post dates.

I was in dilemma whether to also collect job requirements & the complete description in the same function. Indeed, this increases runtime significantly, hence the "Getting Descriptions takes a while, don't worry" message. Interestingly, I need to 1st read the link found inside JobBox, then find the link to the job again on the resulting page, to be able to collect job requirements & the complete description. When finished, I cbind() job requirements & description to the Jobs_n_Links table obtained earlier.

```
GetCompanyJobs <- function(DataFrame, HowManyPages, WhichCompanies) {
# i = Company    j = Page
# defining empty data frame for compiling all jobs
  jobs_df <- data.frame()

# Subsetting Company URLs & Checking if inputs are correct
  if (is.numeric(WhichCompanies) == T) {
    ChosenURLs <- DataFrame$url[WhichCompanies]

  } else {
    ChosenURLs <- DataFrame$url[which(
      tolower(DataFrame$shortName) %in% tolower(
        gsub(" ", "-", WhichCompanies)))]

  }
```

```
  if ( length(WhichCompanies) != length(ChosenURLs)) {
    print(paste0("You have a spelling error!! You listed "
                 , length(WhichCompanies)
                 ," companies, but only "
                 , length(ChosenURLs)
                 , " matched"))
    print(paste0("Pick from the list below & either"
                 ," pass their full name, "
                 ,"or their index numbers into a vector"))
    print(Comps10[,c(2,5)])
  } else {
    print(paste0("Getting Jobs for: "))
    print(WhichCompanies )
  }
# 40 jobs listed on 1 page
  for (i in 1:length(ChosenURLs)) {
    print(paste0("Starting collection for Company Nr. ", i))
    t <- read_html(ChosenURLs[i])

## Getting Links of Job Pages of 1 Company
    JobLink <- t %>% html_nodes(
      xpath = "//*[@id='EIProductHeaders']/div/a[2]") %>% html_attr('href')

    JobPage <- read_html(paste0("https://www.glassdoor.com",JobLink))

    Pages <- paste0(unlist(str_split(
      paste0("https://www.glassdoor.com",JobLink)
                                     ,".htm"))[1],"_P",1:HowManyPages,".htm")
    for (j in 1:length(Pages)) {
      print(paste0("Getting jobs from page ",j," of Company Nr. ", i))

      JobBox <-  read_html(Pages[j]) %>%
        html_nodes('.JobsListStyles__jobListItem')  %>%
        html_nodes('.d-flex.flex-row') %>%
        html_nodes('.JobDetailsStyles__jobInfoContainer')

      PostDate <- read_html(Pages[j]) %>%
        html_nodes(
        xpath = "//*[@class = 'JobsListStyles__jobListItem']") %>%
        html_nodes(xpath = "//div/div/div/div[2]/span/span[@class = 'd-none d-md-block']") %>%
        html_text()

      Jobs_n_Links <-  cbind(rbindlist(lapply(JobBox, function(x) {
        tl <- list()
        tl[['Position']] <- x  %>%
          html_nodes('.JobDetailsStyles__jobTitle')  %>%
          html_text()
        tl[['Company']] <- trimws(unlist(str_split(
          unlist(str_split(x %>% html_nodes(
            '.JobDetailsStyles__companyInfo') %>%
              html_text(), ", "))[1:2],'-')))[1]
        tl[['City']] <- gsub(paste0(', ', '[a-zA-Z0-9]','[a-zA-Z0-9]'),'',
                             gsub(paste0(tl[['Company']],' - '),'',x %>%
```

```
                                    html_nodes('.JobDetailsStyles__companyInfo') %>%
                                    html_text()))

        tl[['City_State']] <- x %>% html_nodes(
          '.JobDetailsStyles__companyInfo') %>%
          xml_children() %>% html_text()
        tl[['Links']] <- paste0("https://www.glassdoor.com",x %>%
                                  html_nodes('.JobDetailsStyles__jobTitle') %>%
                                  html_attr('href'))

        return(tl)
      })),PostDate)

      print(paste0("Getting job description from page ",j," of Company Nr. ", i))
      jobtexts <- rbindlist(lapply(Jobs_n_Links$Links, function(x) {
        tl <- list()
        TrueLink <- read_html(x )  %>%
          html_nodes(xpath = "//html/body/div[3]/div/div/div[1]/div/div[2]/section/article/div/ul/li/di
          xml_children() %>% html_attr('href')

        TrueJobPage <- read_html(paste0("https://www.glassdoor.com"
                                        ,unique(TrueLink[!is.na(TrueLink)])[1]))

        # Job Description
        ID <- TrueJobPage %>% html_nodes(xpath = "//html/body/div[3]/div/div/div[1]/div/div/div/div") %
        ID <- ID[which(!is.na(ID))]

        tl[['JobReqs']] <- paste(TrueJobPage %>% html_nodes(xpath = paste0("//*[@id='",ID,"']/div/ul/li
                                   html_text(),sep = " ", collapse = " ")
        tl[['JobDesc']] <- trimws(TrueJobPage %>% html_nodes(xpath = paste0("//*[@id='",ID,"']/div")) %
        print("Getting Descriptions takes a while, don't worry")
        return(tl)
      }))

      print(paste0("Compiling Page ",j," of Company ",i))
      df <-  cbind(Jobs_n_Links, jobtexts)
      jobs_df <- rbind(jobs_df, df)
      saveRDS(jobs_df,paste0('CompanyJobs_',Sys.Date(),'.rds'))
    } # End of For-loop for 1 Companies 1 Page
    print(paste0("Done with All pages for Company", i))

  } # End of For-loop for 1 Company
  return(jobs_df)
} # End of whole Function
```

Find Company Reviews for chosen Companies

So what does this function do? Put simply, this function collects the X most popular reviews from the companies you specify to it, assuming you have the starting link to each company. It collects general comments, as well as positives & negatives too, ideal for text analysis inputs.

The GetCompanyReviews function takes among others a data frame as input, to have all companies' starting url to loop through. Here I only formulated which company's index number to start from & how long of

a sequence to go through from there. I read the starting company page as html, use the relative xpath "//*[@id='EIProductHeaders']/div/a[1]" to get to the Reviews pages' link. For interest, I post how many reviews are there on the company & tried but failed to condition how many pages to loop through on a company depending on how many reviews it has (e.g. what to do if I want 100 reviews but there are only 60). Also, by default reviews are sorted by popularity (perhaps a proxy for helpfulness or relevance), so did not think I should sort it.

I serialize the ReviewLink found in the beginning by adding "_P" & natural numbers from 1 to whatever number of pages I wanted to get. Then, I pass these links to an lapply, find 'boxes' holding general Comment & 1-1 positive / negative aspect on a company. The boxes in this case is actually a long vector of character strings with almost every text string on the page. Luckily, headline comments start with "" and pros & cons are themselves strings, after which the pro / con comment is to be found. Therefore, I first find the positions of the 'Pros' strings, then take the strings after them to get my positive comments. 'Cons' are more problematic, because of words like 'consultant', so I first find the containing the string 'Cons', then from those I subset the ones which character length is 4, then take the strings after them. I collect these into a list of lists, then rbindlist it & fill up a data frame with it at the end of each loop. I also start to think my functions might start to send too many requests that might get rejected, so I save the final data frame of interest into an RDS object, which overwrites itself after every loop.

```
#### 1) Reviews #### -> Easiest ####
GetCompanyReviews <- function(StartingCompNr
                              ,HowMany,HowManyReviewsPages
                              ,df_fr_GetCompURLs) {
  AllReviews <- data.frame()

for (i in StartingCompNr:(StartingCompNr+HowMany-1)) {
  t <- read_html(df_fr_GetCompURLs$url[i])
  CompName <- df_fr_GetCompURLs$shortName[i]

  print(paste0("Finding reviews for Company Nr. ",i, ": ", df_fr_GetCompURLs$shortName[i]))

#Reviews -> Do all the below by page
  ReviewLink <- t %>% html_nodes(
    xpath = "//*[@id='EIProductHeaders']/div/a[1]") %>% html_attr('href')
  ReviewPage <- read_html(paste0("https://www.glassdoor.com",ReviewLink))

# Find how many reviews & ReviewPages there are
  RevCount <- ReviewPage %>%
    html_nodes('.active') %>% html_text()
  print(paste0(RevCount[2]))

if (length(grep("k",unlist(str_split(trimws(RevCount[2])," "))[1])) == 1) {
    HowManyRevPages <- HowManyReviewsPages
   } else {
    RevCount <- as.numeric(unlist(str_split(trimws(RevCount[2])," "))[1])
    print(RevCount)
    HowManyRevPages <- HowManyReviewsPages
  }
    RevPages <- paste0("https://www.glassdoor.com"
                       ,unlist(str_split(ReviewLink, ".htm"))[1]
                       ,"_P",1:HowManyRevPages,".htm")
    Reviews <- data.frame()
    print(paste0("Fetching ", HowManyRevPages*10
                 , " Reviews for Company Nr. ", i, ": ", df_fr_GetCompURLs$shortName[i]))
```

```
    AllProsCons <- lapply(RevPages, function(x) {
      tl <- list()
      ProsCons <- read_html(x) %>% html_nodes('.gdReview')  %>%
        html_nodes('.v2__EIReviewDetailsV2__isExpanded , .v2__EIReviewDetailsV2__isCollapsed , .strong')
        html_text()
      tl[['Pros']] <- ProsCons[grep("Pros", ProsCons)+1][1:10]
      tl[['Cons']] <- ProsCons[grep("Cons", ProsCons)+1][which(nchar(ProsCons[grep("Cons", ProsCons)]) =
      tl[['Comments']] <- ProsCons[grep("\"", ProsCons)][1:10]
      tl[['Company']] <- unlist(str_split(unlist(
        str_split(x,"Reviews/"))[2],"-Reviews"))[1]

      return(tl)
    })

    df <- rbindlist(AllProsCons, fill = T)
    Reviews <- rbind(Reviews,df)
    AllReviews <- rbind(AllReviews, Reviews)

    saveRDS(AllReviews,paste0('CompanyReviews_',Sys.Date(),'.rds'))
    print(paste0("Compiling Reviews for companies. "
                 ,length(AllReviews$Comments)
                 , " Reviews collected so far" ))
    next()
  }
 return(AllReviews)
}
```

Find Salary Data for Job Title keyword in chosen Companies

So what does this function do? This function collects location specific salary data from the vector of company names passed on to it, specifically on the job titles matching the search term on the specified number of pages to search for. In the process aggregate salary data on all found positions is also stored. I construct a long series of location specific urls per job title to get specific salaries in different cities / countries, by extracting low-level json objects from the lowest-level urls I found. I ensure function continuity by saving outputs to RDS objects & using try() to skip through iterations with errors, a good solution for imperfect url reconstruction.

The GetSalaryData functions' name is also quite descriptive. It also takes the data frame obtained from GetCompanyURLs (or any filtered version of it), as well as the number of salary pages to look through & the keyword term to look for in the job titles of the information found. I also wanted to get very specific & pinned down salary data by location, therefore was not satisfied without finding city-specific salary information. To do so, I also am sourcing a Country continent table, though proving not very helpful eventually. I do all this because in its raw collected form, countries & cities of respective countries are all appearing as locations, producing duplicates in a final data frame, especially if there is only 1 job location in a given country (note that all available options appear in the filter dropdown object, which a find in the html to get all possibilities).

I start from companies' first landing page. In theory this means I could combine all 4 of these functions & collect first the company information, then from there Jobs, Reviews & Salary data. While I believe there's still value in using it separately, e.g. in a Plotly / Shiny based dashboard, the idea of putting it together failed miserably, the volume & density of html requests got my PC not overloaded, but banned from the website for few days.

I first find grand-children of the html node "//*[@id='EIProductHeaders']" similarly to earlier cases, get the link in there & serialize it to the number of pages I need to find. I loop through all these pages, within a

loop through each company. For 1 page, I fill a list of lists with all Position Titles, Links to further info on the Positions, the Average (Median) pay, & the sample size based on which the average figure is supposedly based. I'm not satisfied however, because by default, all positions only show data for in the United States, neither useful for me as a European, or as an American (California might pay a lot better than Wyoming e.g.). Also, unless I want to compare salaries between various positions as a researcher (so the practical use case of trying to figure out what can I earn where), Software Engineering data is quite useless for me as an Data Analyst / Scientist.

It is for this reason I included the position keyword search, also because some positions are more relevant then others, & same jobs might have different names. The key challenge is constructing the position & location specific hyperlink. For this, lets start with the position specific url & location specific url for comparison:

https://www.glassdoor.com/Salary/Google-Financial-Analyst-Salaries-E9079_D_KO7,24.htm

"https://www.glassdoor.com/Salary/Google-Financial-Analyst-Australia-Salaries-EJI_IE9079.0,6_KO7,24_IL.25,**_IN16.htm"

Actually, up until the position title we are quite fine, then there is a location we need to somehow find, then Salaries & the company id, then a few other IDs (I believe related to office & job title IDs). Importantly, in all mappings from location-general to location specific links, the "KO7" maps to "6_KO7" & "24" maps to "24_IL.25,". Finally, the "N16" in the end is also a location ID for Australia. The 2nd double-for-loop in the function solves precisely this challenge (also using wildcard characters as indicated above), unless the keyword searched for is "Intern" since those sub-pages are organized differently (there are no location-specific urls dedicated to intern salaries).

This is why this double-loop is conditional, & before stepping in the 2nd loop, all locations are found for a position using an absolute & relative xpath combined, as well as every locations' IDs (the last letter-digit combination in the url). The 2nd loop is to create these position & location specific links. Its' components are mentioned above, & are extracted with a long series of string manipulations. All these columns are binded & since locations at different aggregation levels are in the same columns, the Aggreg column denotes whether it is a city or country based on the location ID. The final lapply goes through all location-specific links, & extracts a json object, with company name, title, 10th, 50th & 90th percentile salary estimates & sample size. With the quickly growing number of htmls to read through, I use try() to read html, & if read_html fails, that iteration is skipped, ensuring the function completes & negating possible wrong city name formulations, e.g. Miami, Fort Lauderdale, instead of Miami, resulting is error responses. I also again save everything continually to RDS objects, which overwrite i.e. aggregate.

```
GetSalaryData <- function( Df_w_CompURLs
                           , Nr_Pages_2_CheckperCompany = 3
                           , PosTitleStringKeyword ) {
  source('CountriesContinentsTable.R')
  CountryTable <- GetCountryTable()
  Pages <- Nr_Pages_2_CheckperCompany
  df <- data.frame()

  for (i in 1:length(Df_w_CompURLs$shortName)) {
    tt <- read_html(Df_w_CompURLs$url[i])

    ## 1) Get Salary Pages for 1st Company
    MoneyLinks <-   paste0(
      gsub('.htm','_P',paste0("https://www.glassdoor.com",tt %>%
                              html_nodes(xpath = "//*[@id='EIProductHeaders']/div/a[3]") %>%
                              html_attr('href'))),1:Pages,'.htm')
    ## 2) Get Position-Spec Salary Links
    for (j in 1:length(MoneyLinks)) {
      ttt <- read_html(MoneyLinks[j])
```

```
      PosLinks <- rbindlist(lapply(2:21, function(x) {
        tl <- list()
        tl[['PosLink']] <- ttt %>% html_nodes(
          xpath = paste0("//*[@id='SalariesRef']/div[",x
                         ,"]/div/div[2]/div[3]/div/a")) %>%
          html_attr('href')
        if  (sum(grepl("Intern",tl[['PosLink']])) == 1 ) {
## Intern Positions have wierd URL structure & locations
##      Here a conditioned string decomposition does the trick
          tl[['Position']] <- paste0(unlist(str_split(
            gsub(paste0(Df_w_CompURLs$shortName[i],'-'),'',
                 unlist(str_split(tl[['PosLink']],"Salary/"))[2])
            ,"-Salary-"))[1]," ",

            unlist(str_split(
              unlist(str_split(
                gsub(paste0(Df_w_CompURLs$shortName[i],'-'),'',
                     unlist(str_split(tl[['PosLink']],"Salary/"))[2])
                ,"Salary-"))[2],"="))[2])
        } else {
          tl[['Position']] <- gsub("-"," ",
                                   unlist(str_split(
                                     gsub(paste0(Df_w_CompURLs$shortName[i],'-'),'',
                                          gsub("/Salary/",'',tl[['PosLink']]))
                                     ,'-Salaries'))[1])
        }

        AvgPay  <- gsub(',','',ttt %>% html_nodes(
          xpath = paste0("//*[@id='SalariesRef']/div[",x
                         ,"]/div/div[2]/div[1]/span[1]/span/strong/strong")) %>%
            html_text())

        tl[['AvgPay']] <- as.numeric(substring(AvgPay,2,nchar(AvgPay)))
        tl[['SampleSize']] <- as.numeric(unlist(str_split(ttt %>% html_nodes(
          xpath = paste0("//*[@id='SalariesRef']/div[",x
                         ,"]/div/div[2]/div[3]/div/span/span/strong")) %>%
            html_text()," "))[1])

        return(tl)
      })) # End of PosLinks lapply -> Getting Positions from 1 Page of 1 Company
      print(paste0("Got page ",j," of Company ", i))
      PosLinks$Company <- Df_w_CompURLs$shortName[i]

      df <- rbind(df,PosLinks)
      saveRDS(df, paste0("SalaryDataAggreg_",Sys.Date(),".rds"))
    } # End of j loop -> Getting All positions from 1 Company
  } # End of i loop -> Getting All positions from all selected Companies

  print(paste0("Got all Position Links 4 all Companies"))
  print(paste0("Starting Location Specific SalaryData Collection"))

  # 3) Partial Matching for Keyword
  print(paste0(length(grep(tolower(PosTitleStringKeyword)
```

```
                          ,tolower(df$Position)))
              ," Positions found matching to the term: "
              ,PosTitleStringKeyword))

PosKeyword <- df[grep(tolower(PosTitleStringKeyword)
                      ,tolower(df$Position))]
TruePosLinks <- paste0("https://www.glassdoor.com",PosKeyword$PosLink)

# 4) Getting JSON Outputs
df_fr_js <- data.frame()
for (l in 1:length(TruePosLinks)) {
  PosPage <- read_html(TruePosLinks[l])

  # As commented earlier, Intern salaries are structured differently
  # No location specific URL therefore cant be manipulated so only collected on aggregate
  if (sum(grepl(tolower("Intern"),tolower(TruePosLinks[l]))) == 1) {
    PosJson <- fromJSON(PosPage %>% html_nodes(
      xpath = "//script[@type='application/ld+json']") %>% html_text())

    Comp <- PosJson$hiringOrganization$name    # Company
    Pos <- PosJson$name                        # Position
    SalaryEst <- PosJson$estimatedSalary %>% select(-'@type')          # Salary Estimate
    Sample <- as.numeric(PosJson$sampleSize)  # Sample Size

    SalaryTableperpos <- cbind(Comp,Pos,SalaryEst,Sample)
    print(paste0("Loading Intern Salaries at Position Nr. ",x))

    df_fr_js <- rbind(df_fr_js,SalaryTableperpos)
    df_fr_js <- df_fr_js %>% filter(median != 0)
    saveRDS(df_fr_js, paste0(trimws(PosTitleStringKeyword),"_SalaryData_",Sys.Date(),".rds"))
    next()
  } else {
    ## 4.1) Cities for selected position
    PosAllLocs <- PosPage %>% html_nodes(
      xpath = "//*[@id='nodeReplace']/main/div/div/div[1]/div[3]/div/div[1]/div/div/div/div[2]/div/ul
      xml_children() %>% html_nodes(xpath = "//*[@class = 'dropdownOptionLabel']") %>% html_text()

    LocIDs <- gsub(",",'',gsub("option_",'',PosPage %>% html_nodes(
      xpath = "//*[@id='nodeReplace']/main/div/div/div[1]/div[3]/div/div[1]/div/div/div/div[2]/div/ul
        xml_children() %>% html_attr('id')))

    PosAllLocs <- PosAllLocs[1:length(LocIDs)]
    Pos_Locs <- as.data.frame(cbind(LocIDs,PosAllLocs))

    ## 4.2) Generating Location Specific Salary links per Position
    Cities2Link <- NULL
    LinkEnd <- NULL
    LocSpecLink <- NULL
    for (k in 1:length(Pos_Locs[,1])) {
      Cities2Link[k] <- gsub(' ','-',trimws(unlist(str_split(gsub("-",'',Pos_Locs[k,2]),','))[1]))
      ReplNR1     <-  as.numeric(gsub("KO",''
                                  ,unlist(str_split(
                                    unlist(str_split(
```

```
                                  gsub(".htm","",TruePosLinks[l])
                                  ,"_D_"))[2],',''))))[1]
  ReplNR2      <- as.numeric(gsub("KO",''
                             ,unlist(str_split(
                               unlist(str_split(
                                 gsub(".htm","",TruePosLinks[l])
                                 ,"_D_"))[2],',''))))[2]
  LinkStart      <- unlist(str_split(TruePosLinks[l],'Salaries'))[1]
  CompCodenOff   <- gsub("_D",paste0(".0,",ReplNR1-1)
                         ,gsub("-",'',
                               unlist(str_split(
                                 unlist(str_split(TruePosLinks[l]
                                                  ,'Salaries'))[2],','))[1]))
  LinkEnd[k]        <- paste0(ReplNR2,"_IL.",ReplNR2+1
                              ,",**","_I",Pos_Locs[k,1],".htm")

  LocSpecLink[k] <- paste0(LinkStart , Cities2Link[k]
                           ,'-Salaries-EJI_I',CompCodenOff,',',LinkEnd[k])
}

Pos_Locs$LocSpecLink <- LocSpecLink
Pos_Locs$Locations <- gsub("-",' ',Cities2Link)

Pos_Locs <-  Pos_Locs %>% left_join(
  CountryTable, by = c("Locations" = "Country"))  %>%
  select(-PosAllLocs)
Pos_Locs$Aggreg <- gsub('[[:digit:]]','',gsub("M","City",gsub("N","Country",Pos_Locs$LocIDs)))

## 4.3) Binding JSON outputs into table
FinalTable <- rbindlist(lapply(1:length(Pos_Locs$LocSpecLink),function(x) {
  tl <- list()
  boxx <- try(read_html(Pos_Locs$LocSpecLink[x]))
  if(inherits(boxx, "try-error")) {
    #error handling - skip iteration
    return(tl)
  } else {
    PosJson <- fromJSON(boxx %>% html_nodes(
      xpath = "//script[@type='application/ld+json']") %>% html_text())
    tl[['Company']]         <- PosJson$hiringOrganization$name
    tl[['Position']]        <- PosJson$name
    tl[['SalaryEstimates']] <- PosJson$estimatedSalary %>% select(-'@type')
    tl[['SampleSize']]      <- as.numeric(PosJson$sampleSize)
    tl[['Dims']]            <- Pos_Locs[x,] %>% select(Aggreg,Locations, Continents)

    ## Cant rbindlist() all tl -> SalaryEstimates has 2 rows: Base & Bonus salary
    tl[['Total']]           <- cbind(tl[['Company']]
                                     ,tl$Position,tl$SalaryEstimates
                                     ,tl$SampleSize,tl$Dims)
    print(paste0("Loading Salaries at Location Nr. "
                 ,x, " for Position Nr.",l))
  } # End of Else within FinalTable(lapply)
  return(tl[['Total']])
})) # End of FinalTable(lapply)
```

```
      df_fr_js <- rbind(df_fr_js,FinalTable)
      df_fr_js <- df_fr_js %>% filter(median != 0)
      saveRDS(df_fr_js, paste0(trimws(PosTitleStringKeyword)
                               ,"_SalaryData_",Sys.Date(),".rds"))

    } # End of Else within l for-loop
  } # End of l for-loop -> Getting all salary JSONs for TruePosLinks[l]
  return(df_fr_js)
} #  End of GetSalaryData function
```