STA141B HW4

Kwasie Agbemadon

11/23/2020

Packages used: XML, RCurl, Stringr, ggplot2, and Tidyr

To begin, I started off by loading two required packages: XML and RCurl. Those packages have everything I need to do this assignment.

I will be using Cybercoders to scrape the data. I made 6 functions to do the scraping, one is for getting the needed skills, another for getting the recommended skills, another for getting the job position, location, and pay, and one that combines the skills plus the keyword that is inputted on the website. I also made a function called getNextPage which grabs the link to the next page. The search function uses all of the above functions.

For getting the required skills, the link is inputted in the url parameter, with an additional assigned parameter that parses the webpage using htmlParse. The function obtains the value of the specified node; In this case, the list of needed skills have a div variable called “data-section”, which is set to 7 (that represents the needed skills). I then pasted the text and trimmed off unneeded space.

For getting the Descriptions of the job postings, I scraped text that isn’t a list or doesn’t contain bullet points. I also included the header titles as well. I appended each aspect as it would make up the whole description.

For getting the Preferred Skills, I did the same thing as before, as it was a span node. I then pasted the text with a comma, because those skills had clickable links and I only wanted the text.

Both functions were performed with the getJobSkills function, which takes the url with the info of the job posting and makes vectors with the functions. The function then spits out a list of required and preferred skills that are mapped to that job posting.

I got the title, location, company, and the wage using the getJobDetails function. This function takes the texts of each aspect, which is associated with a node. The location is split into city and state (using strsplit and sapply), but the wage was originally fused with the type of job it was (Full-Time, Part-Time, etc.), and some job postings did not mention wages. I resolved this by using a variable to separate the job type by the wage, and then used an if else statement given that the wage wasn’t mentioned. I then made vectors with those variables. The function spits out a list of all the info.

The search function takes a query with the url for the query to be searched. I used the getForm function, as it allowed me to enter the url and input a keyword. The keyword is sent to the webpage and with the right variable to the node with the textbox, it will acquire a url with that keyword searched up. I used that url on the getJobDetails function.

I used the condition of the length is the list not being zero, because one of the keywords I looked up brought up no results. Aside from that, the code grabs links to the info of each job posting using the getRelativeURL function, in which I input the href from the respected nodes. I then used getJobSkills() on those links.

Most of the time, there is more than one page to a search result. That is why we obtain the link to the next page. I wrote a separate function which does exactly that. It will take the href of that link, given that the variable “rel” has “next” in it, using getNodeSet(). It will then join the main url via the paste0() function. I had to remove the period using the sub function, or else the link is nonexistent.

Once the link to the next page has been grabbed, the same process is repeated, adding all details to each job posting and getting the link to the next page until there none. At this point, we completely scraped every job posting on this website.

Here is the code of the whole process:

```
library(XML)

## Warning: package 'XML' was built under R version 4.0.3

library(RCurl)

## Warning: package 'RCurl' was built under R version 4.0.3

library(stringr)
library(ggplot2)

## Warning: package 'ggplot2' was built under R version 4.0.3

library(tidyr)

## Warning: package 'tidyr' was built under R version 4.0.3

##
## Attaching package: 'tidyr'

## The following object is masked from 'package:RCurl':
##
##     complete

getDesc = function(url, html = htmlParse(readLines(url))) {
  desc = xmlValue(getNodeSet(html, "//div[@class = 'section-
data']/text()[string-length(normalize-space(.)) > 0]"))
  desc = append(desc, xmlValue(getNodeSet(html, "//h4[@class = 'section-
title']"))[1])
  desc = append(desc, xmlValue(getNodeSet(html, "//div[@class = 'section-data
section-data-title'][1]/text()")))
  paste(desc, collapse = '')
}

getReqSkills = function(url, html = htmlParse(readLines(url))) {
```

```r
  req_skills = xmlValue(getNodeSet(html, "//div[@data-section =
'7']/text()"))
  paste(req_skills, collapse = '')
}

getPrefSkills = function(url, html = htmlParse(readLines(url))) {
  pref_skills = xmlValue(getNodeSet(html, "//span[@class = 'skill-name']"))
  paste(pref_skills, collapse = ', ')
}

getJobDetails = function(form) {
  title = xmlValue(getNodeSet(form, "//div[@class = 'job-title']"))
  title = gsub("\n\\s*", "", title)
  location = xmlValue(getNodeSet(form, "//div[@class = 'location']"))
  location = gsub("\\s+{2,}", "", unlist(location))
  l = strsplit(location, ', ')
  city = sapply(l, '[', 1)
  state = sapply(l, '[', 2)
  state = sub(' ', '', state)
  unknown = xmlValue(getNodeSet(form, "//div[@class = 'wage']"))
  a = regmatches(unknown, regexpr(' ', unknown), invert = T)
  type = c()
  wage = c()

  for (i in 1:length(unknown)) {
    type = c(type, ifelse(!'Compensation Unspecified' %in% unknown[i],
                  sapply(a[i], '[', 1),
                  'Type Unspecified'))
    wage = c(wage, ifelse(!'Compensation Unspecified' %in% unknown[i],
                  sapply(a[i], '[', 2),
                  'Compensation Unspecified'))
  }

  return(list(Title = title, Company = rep("Cyber Cobers", length(title)),
City = city, State = state, Type = type, Wage = wage))
}

getJobSkills = function(url, searchTerm) {
  desc = unlist(lapply(url, getDesc))
  reqskills = unlist(lapply(url, getReqSkills))
  prefskills = unlist(lapply(url, getPrefSkills))
  keyword = rep(searchTerm, length(reqskills))
  website = rep("CyberCoders", length(reqskills))
  return(list(Description = desc, `Required Skills` = reqskills, `Preferred
Skills` = prefskills, Keyword = keyword, Website = website))
}

search = function(searchTerm, url = 'https://www.cybercoders.com/search') {
  pg = htmlParse(getForm(url, searchterms = searchTerm))
  d = getJobDetails(pg)
  if (length(d$Title) != 0) {
    link = getRelativeURL(getNodeSet(pg, "//div[@class =   'job-
title']//a/@href"), url)
    Sys.sleep(sample(3:6,1))
    s = getJobSkills(link, searchTerm)
    nextpg = getNextPage(pg, url)
    if(url.exists(nextpg)) {
      while (TRUE) {
        nextpg = htmlParse(readLines(nextpg))
        detailstmp = getJobDetails(nextpg)
        d$Title = c(d$Title, detailstmp$Title)
        d$City = c(d$City, detailstmp$City)
        d$Company = c(d$Company, detailstmp$Company)
        d$State = c(d$State, detailstmp$State)
        d$Wage = c(d$Wage, detailstmp$Wage)
        d$Type = c(d$Type, detailstmp$Type)
        link = getRelativeURL(getNodeSet(nextpg, "//div[@class =   'job-
title']//a/@href"), url)
        skillstmp = getJobSkills(link, searchTerm)
        s$Description = c(s$Description, skillstmp$Description)
        s$`Required Skills` = c(s$`Required Skills`, skillstmp$`Required
Skills`)
        s$`Preferred Skills` = c(s$`Preferred Skills`, skillstmp$`Preferred
Skills`)
        s$Keyword = c(s$Keyword, skillstmp$Keyword)
        s$Website = c(s$Website, skillstmp$Website)
        nextpg = getNextPage(nextpg, url)
        if(!url.exists(nextpg))
          break
      }
    }
      return(c(d, s))
  }
}

getNextPage = function(pg, url) {
  nextPageNode = as.character(getNodeSet(pg, "//a[@rel = 'next']/@href"))[1]
  if (length(nextPageNode) != 0)
    newurl = paste0(url, sub('.', '', nextPageNode))
}
```

Since everything from CyberCoders has been scraped, we can now start our analyzation. I started this off with giving each list a name and using the search function (which spits out a list). I then create the data frame and bind the lists by row using rbind (this function adds entries to a dataframe). I also renamed the columns to make it look more professional.

```r
url = search('data scientist')
url2 = search('data analyst')
url3 = search('statistician')
```

```
url4 = search('machine learning')

data = data.frame(url)
data = rbind(data, data.frame(url2))
data = rbind(data, data.frame(url3))
data = rbind(data, data.frame(url4))
colnames(data) = c("Title", "Company", "City", "State", "Type", "Wage",
"Description", "Required Skills", "Preferred Skills", "Keyword", "Website")
```

Exploration

I am going to discover how many job postings there are with the given keyword(s) and site(s). I used the table function with the keyword and website vector.

```
table(data$Keyword)

##
##     data analyst   data scientist machine learning
##               28               16               43
```

The Machine Learning keyword has more results, followed by Data Analyst, then Data Scientist. It appears that Machine Learning is more widely known that data scientists/analysts.

Next, I will find out how many job postings each state holds. This can be attained using, again, the table function.

```
table(data$State)

##
## AZ CA CO DC FL ID IL MA MD NC NJ NY OH ON OR PA TX UT VA WA
##  1 29  2  2  1  2  7  3  1  2  3  7  2  1  1  3  7  2  8  3
```

From our table, we can see that California holds the majority of job postings, since data jobs are centered in the Bay Area, and the fact that the Bay Area is gentrified. We can also see that New York, Illinois, Virginia, and Texas also have the second most amount of job postings.

I made these two functions to explore the wages and the effects they have. One is called splitWage(), where it takes a column from our dataset and separates the entries that column, the minimum wage, and the maximum wage. I made sure to pick the ones that have wages using the grep function(), because some of postings never mentioned a payment.

The convertToWholeNo() function just formats the numbers from k to a whole one. For example: 15k -> 15000.

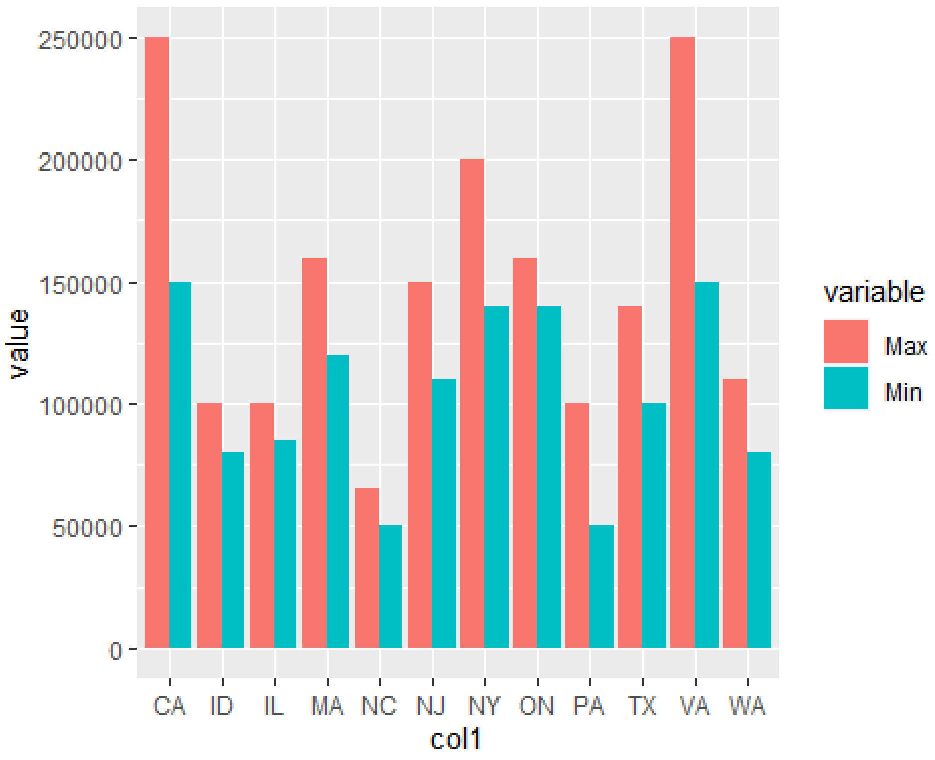
```
splitWage = function(col1, col2 = data$Wage) {
  isSpecified = grep('\\$', col2)
  c = strsplit(col2[isSpecified], ' | - ')
  min = convertToWholeNo(sapply(c, '[', 1))
```

```r
  max = convertToWholeNo(sapply(c, '[', 2))
  col1 = col1[isSpecified]
  return(data.frame(col1, Min = min, Max = max))
}

convertToWholeNo = function(col) {
  return(as.numeric(gsub('\\$|k', '', col)) * 1000)
}
```

Next, I will determine which states hold the highest minimum and maximum wages. I used ggplot to do this because it's more convenient. TO get to that point, I had to use the pivot_longer function from tidyr, which spits out a dataset and maps variables to their correct values. Here is our result:

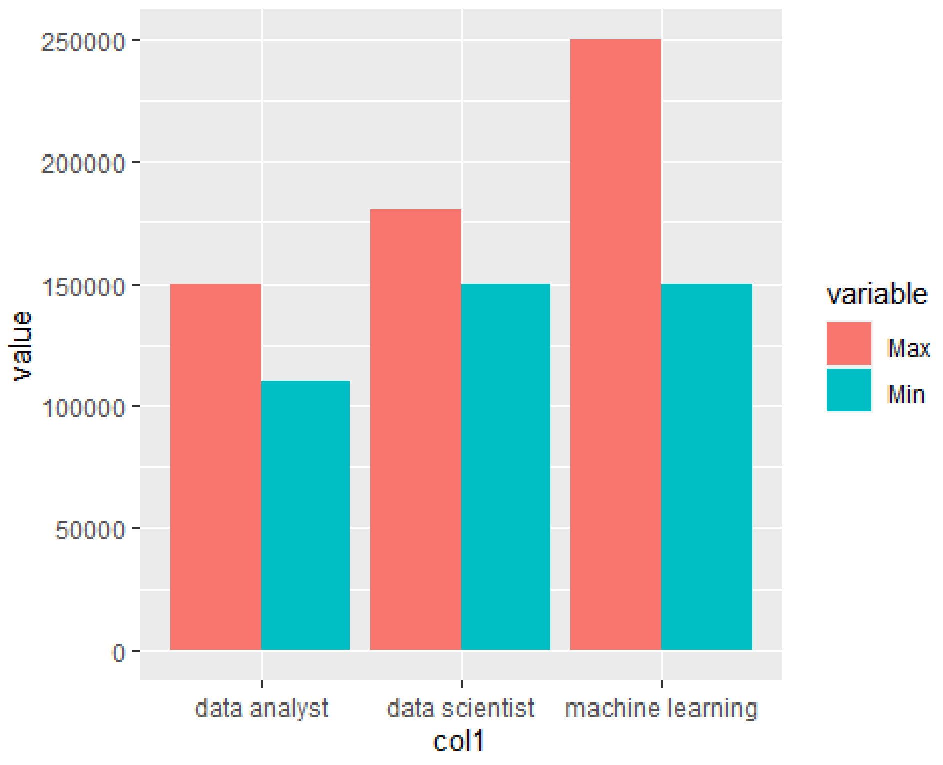
```r
wages_state = pivot_longer(splitWage(data$State), cols = c('Max', 'Min'),
names_to = 'variable', values_to = 'value')
ggplot(wages_state, aes(x = col1, y = value, fill = variable)) +
geom_bar(stat = 'identity', position = 'dodge')
```



California and Virginia hold the highest wages, and they also have a disparity between the maximum and minimum amount possible. California has been known to have the highest salaries when it comes to tech. Virginia also has some tech areas as well, given that most of them are in the DC area.

I can do the same thing with keyword inputs:

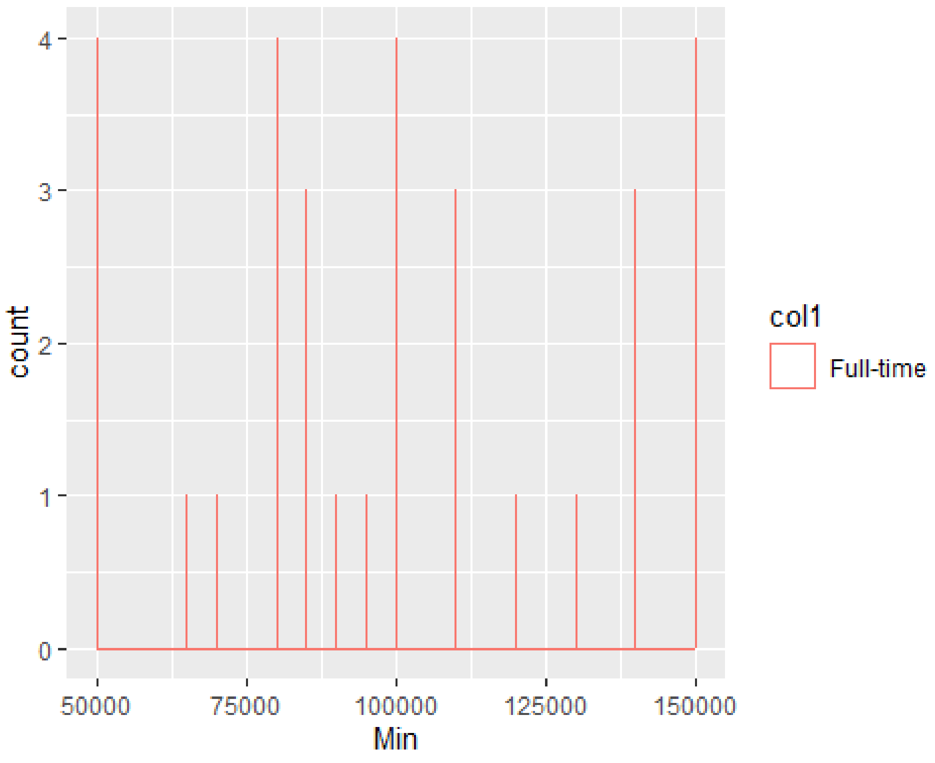
```r
wages_keyword = pivot_longer(splitWage(data$Keyword), cols = c('Min', 'Max'),
names_to = 'variable', values_to = 'value')
ggplot(wages_keyword, aes(x = col1, y = value, fill = variable)) +
geom_bar(stat = 'identity', position = 'dodge')
```



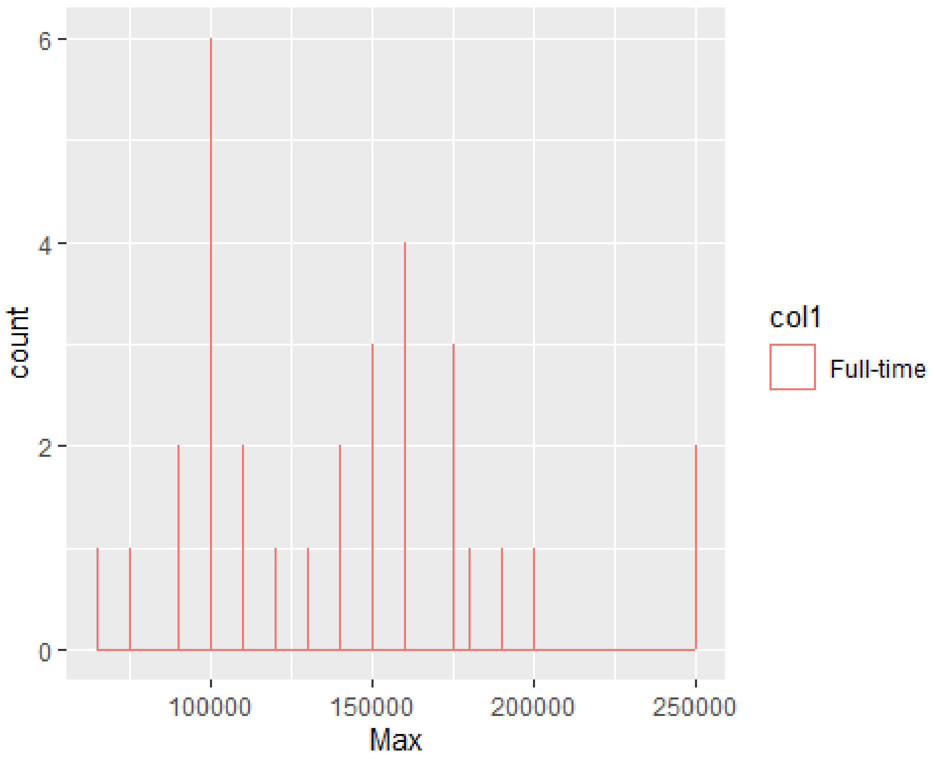
As usual, Machine Learning holds the highest maximum wage, and it also holds the widest gap between the maximum and minimum salaries.

I can also plot the a histogram of the wages. This can also be done with ggplot:

```r
ggplot(splitWage(data$Type), aes(x = Min, color = col1)) +
geom_histogram(fill = 'white', position = 'identity', binwidth = 5)
```



```
ggplot(splitWage(data$Type), aes(x = Max, color = col1)) +
geom_histogram(fill = 'white', position = 'identity', binwidth = 5)
```



For minimum earnings, it turns out that amounts $50k, $100k, and $150k are more frequent.

FOr maximum earnings, $100k is more common.

The years of experience jobs require can be calculated by extracting the skills list of the job posting. In this case, I extracted any numbers that could or could not contain a plus sign because they indicate that it would be the amount of years of experience, then formed the values into a table.

```
yearsOfExperience = str_extract(data$`Required Skills`, '[0-9]\\+*')
yearsOfExperience = yearsOfExperience[which(!is.na(yearsOfExperience))]
table(yearsOfExperience)
```

```
## yearsOfExperience
##   1   2  2+   3  3+   4  4+   5  5+ 5++   6   7  7+  8+
##   3   1   4  13   2   2   4   1  22   2   2   1   1   1
```

Most jobs require at least 3-5 years of experience with prior skills. There are rarely jobs that require 1-2 years, let alone 8+ years.

We can also see which skills are common recommended for jobs. Here I separated each skill by comma, then unlisted the whole column of preferred skills. I then formed a table in descending order to show the top 30 mentions.

```
skills = unlist(strsplit(data$`Preferred Skills`, ', '))
s = table(skills)
head(s[order(-as.numeric(s))], 30)
```

```
## skills
##          Machine Learning                    Python
SQL
##                        42                        37
26
##                         R              Data Science
SPARK
##                        16                        13
12
##                Tensorflow           Computer Vision   Artificial
Intelligence
##                        11                         9
7
##                       NLP                   PyTorch             Data
Analysis
##                         7                         7
6
##       Account Resolutions                       AWS Business Systems
Analysis
##                         5                         5
5
##                       C++                   Chatbot               Complex
```

```
SQL
##                         5                          5
5
##          consumer lending       Data Transformation             Deep
Learning
##                         5                          5
5
##          Indirect Lending          Process Diagrams               Real
Estate
##                         5                          5
5
##        Reporting Services      Statistical Modeling
Data
##                         5                          5
4
##            Data Analytics               Data Mining
Docker
##                         4                          4
4
```

Machine Learning, Python, SQL, R, Data Science, and SPARK are the most common recommended requirements. There are similar terms which are rarely mentioned. Not surprising isn't it? Those terms make the pinnacle of Data Science.