Exercise 5 - Interacting with data using R

Work with a partner to complete the tasks below and submit your results via a pull request on GitHub by the beginning of tutorial next Friday.

To begin this week, one of the partners should fork the TA's Exercise 5 Github repo and provide collaborative access to the other partner. Clone the forked repo so that you have the required files. Be sure to commit regularly to show how you arrived at your solutions and demonstrate coding effort by both partners.

Tutorial

In this short tutorial, we will work with two tabular datasets to extend a couple concepts from the Patient Data exercise from Software Carpentry.

First we will load a simple data table using `read.table()`:

```
data=read.table(file="test.dat",header=FALSE,sep=" ")
data
```

```
##   V1 V2 V3 V4 V5
## 1  1  5  9 13 17
## 2  2  6 10 14 18
## 3  3  7 11 15 19
## 4  4  8 12 16 20
```

In the Patient Data exercise, we saw how we can use square brackets `[]` to index or subset data. We can also use results of logic tests to index our data. This can be really useful when we have a large dataset that we want to access a subset of based on characterstics of the data itself.

```
# we can test for equality using double equal signs
data[,1]==1
```

```
## [1]  TRUE FALSE FALSE FALSE
```

```
# we can test for greater than or less than as well
data[,1]>2
```

```
## [1] FALSE FALSE  TRUE  TRUE
```

```
# the logical values returned by a logic test can be used just like numbers to index a data structure
data[data[,1]>2,]
```

```
##   V1 V2 V3 V4 V5
## 3  3  7 11 15 19
## 4  4  8 12 16 20
```

A nice thing about `dataframes` in `R` is that they can hold more than one data mode (e.g. numbers and characters), like what we saw in `wages.csv` from our exercise last week.

```
#Load wages.csv; the stringsAsFactors argument prevents strings from being treated as factors
wages=read.csv(file="wages.csv",header=TRUE,stringsAsFactors=FALSE)
class(wages)
```

```
## [1] "data.frame"
```

```
dim(wages)
```

```
## [1] 3294    4
```

```
head(wages)
```

```
##   gender yearsExperience yearsSchool     wage
## 1 female               9          13 6.315296
## 2 female              12          12 5.479770
## 3 female              11          11 3.642170
## 4 female               9          14 4.593337
## 5 female               8          14 2.418157
## 6 female               9          14 2.094058
```

We can use square brackets to index a portion of a `dataframe`, but we can also use something called dollarsign notation. This is because a `dataframe` also behaves like a `list` in `R`.

```
# we can extract all of the female wage data using square brackets
females=wages[wages[,1]=="female",]
dim(females)
```

```
## [1] 1569    4
```

```
unique(females[,1])
```

```
## [1] "female"
```

```
# or a mix of square brackets and dollarsign notation
females2=wages[wages$gender=="female",]
dim(females2)
```

```
## [1] 1569    4
```

```
unique(females2$gender)
```

```
## [1] "female"
```

Some `R` users prefer to use a function `subset()`, rather than square bracket indexing, to access a subset of their data.

```
females3=subset(x=wages,wages$gender=="female")
dim(females3)
```

```
## [1] 1569    4
```

```
unique(females$gender)
```

```
## [1] "female"
```

Challenge

Work with your partner to develop a R script that accomplishes the three tasks below. These tasks will require the file "wages.csv", which you should have in your local directory since you cloned the repo you forked from the TA.

1. Write a file containing the unique gender-yearsExperience combinations contained in the file "wages.csv". The file you create should contain gender in the first column and yearsExperience in a second column with a space separating the two columns. The rows should be sorted first by gender and then by yearsExperience, but remember to keep the pairings in a given row intact. Don't worry about column names in the output file. **Hint:** `order()` is likely a useful function for sorting two-dimensional data structures in `R`. Also, the opposite of `read.table()` is `write.table()`.
2. Return the following information to the R console when the script is executed: the gender, yearsExperience, and wage for the highest earner, the gender, yearsExperience, and wage for the lowest earner, and the number of females in the top ten earners in this data set. Be sure to indicate, which output is which when returning them to the console.
3. Return one more piece of information to the console: the effect of graduating college (12 vs. 16 years of school) on the minimum wage for earners in this dataset.

Devise a plan for splitting up the work and generating the required code. Do this in parrallel, not sequentially. Don't forget to check and edit each other's code. Remember to frequently `add-commit` locally and `push-pull` to GitHub to avoid conflicts. Also, remember you don't have to be in the same place at the same time to work on this collaboratively thanks to GitHub!!!

Turning in your assignment via GitHub

Once you have committed all changes to your local Git repos and pushed all of those commits to the forked repo on GitHub, you can "turn in" your assignment using a `pull request`. This can be done from the GitHub repo website. When viewing the forked repo, select "Pull requests" in the upper middle of the screen, then click the green "New pull request" button in the upper right. You'll then see a screen with a history of commits for you and your collaborator, select the green "Create pull request button". In the text box next to your user icon near the top of the page, remove whatever text is there and add "owner's last name - collaborator's last name submission", but obviously substitute your last names. If I and Ann Raiho worked on the project together the text would read "jones-raiho submission". Then click the green "Create pull request" button. **Only one of you will need to create a pull request.**