Biocomp Exercises and Answers

Salinas, Alexa

12/13/2019

## *Exercise 1*

1. List three arguments that can be passed to cp and describe what behavior these arguments induce.

**cp -i one.txt onecopy.txt - to make a copy of one.txt file with the new file name as onecopy.txt (-i to make sure you are asked before something is overwritten); does not include file permission**

**cp -r data-shell data-shell-copy - to make a copy of a whole directory and all of its contents named as data-shell-copy**

**cp -b notes.txt pizza.cfg - to make a backup of pizza.cfg called pizza.cfg~ if the copy will overwrite the original**

1. What is the difference between an absolute and relative path?

**An absolute path is a complete path that stems the desired directory or file from the root. A relative path is a path that stems from the present working directory, not from the root.**

1. What will be returned by ls . after running all of the code below? Assume that your initial working directory exists within another directory that contains the files ‘names.txt’, and ‘addresses.txt’. Hint - feel free to make this toy file structure on your computer and try it!

mkdir newStuff

cd newStuff

cp ../names.txt .

cd ..

mv addresses.txt newStuff

rm -r newStuff

**After running this code, only names.txt (not addresses.txt) is left in the directory that the initial working directory was in.**

1. Assume you have a directory named work. In that directory are two files (one.txt and two.txt). Write the code required to create a directory named data in work, copy two files (one.txt and two.txt) from work to data, and remove the original files from the work directory.

**Once in work directory that contains one.txt and two.txt:**

**mkdir data** **cp *.txt data rm* .txt**

1. What will be returned by ls b\* after running all of the code below? Assume your current directory contains a file called baseball.csv.

mkdir sports

mv baseball.csv sports/

cp sports/baseball.csv sports/hockey.csv

cd sports

cp baseball.csv golf.csv

cp baseball.csv basketball.csv

**Baseball.csv and basketball.csv are returned by ls b\* after running the code.**

## *Exercise 2*

1. What is the difference between > and >> in Unix? What is the difference between \* and ? in Unix?

**In Unix, >> creates a file while > overwrites said file if existent, or creates if not. The \* symbol represents an unlimited amount of characters, while ? is a single character in a file name(s).**

1. Using the file planets.txt from the Software Carpentry example data, describe in your own words what each step of the pipeline at the bottom of this question does. Also, what is the contents of fiveLittlePlanets.txt? Remember that sed is the stream editor and sed 's/"//g' gets rid of all the double quotes in the information from planets.txt. Because some of the planets are missing entries for the second column we need to first make a new version of the input file. You can accomplish this with this short pipeline: cat planets.txt | grep -v ,, > planets2.txt

cat planets2.txt | sed 's/"//g' | cut -d , -f 1,2 | sort -t . -k 2 -n | head -n 5 > fiveLittlePlanets.txt

**The first step concatenates, or reads, the file planets2.txt and redirects the output. The second step calls for the stream editor to remove double quotes in planets2.txt’s information. Step three removes fields in file lines based on a comma as a delimiter so it will remove all fields but the first and second. Step four sorts the file lines by alphabetical and then numerical order based on the first field of all lines. Step five takes the first 5 lines of the file and saves it into a new file called fiveLittlePlanets.txt.** \*TA comment: The cat command accesses information in the file planets2.txt and returns it to stdout, which is then piped as input to the next command (sed). The first five lines returned by the head command consist of a header, followed by the four smallest planets.

1. Imagine you have a working directory that contains two directories - newData and Archive. newData contains 100 .csv files, but you want to move them to Archive for long-term storage. Provide Unix code required to move the 100 files in two different ways. For one of the ways you move the files, you CANNOT use mv and must use a for loop.

mv newData/\*.csv . mv \*.csv Archive

1. (in newData dir)

$ for filename in \*.csv

do

cp \*.csv ../Archive

rm \*.csv

done

1. Give pseudocode and then Unix code to create 100 files. The name of these files should follow the pattern file1.txt, file2.txt, ..., file100.txt. Each file should contain a single line that follows the pattern "This is file number N", where N is the number in the file name. Remember that you can create a sequence of integers from 1 to N with the code {1..N}.

This code will create a new directory in order to create 100 new files named file1.txt, file2.txt…, file100.txt and save a line in each file saying “This is file number N”. (couldn’t figure out how to make N the number in file name)

Start program

Create new directory called Ex2 FOR directory Ex2 Create 100 files with file name format file1.txt

FOR all text files in for loop Do Add same text line to all text files Done

~

mkdir Ex2 cd Ex2 touch file{1..100}.txt

for file in \*.txt do echo "This is file number N" >> $file done

\*TA comment: Good work! However, the output for the echo command does not need to be appended (>>), simply inserted (>) to each txt file. Also, a more simple way to accomplish this task, for N in {1..100} do echo $‘This is file number’ N.txt done

## *Exercise 3*

1. In last week’s exercise, we ran this bit of code to remove planets without any information about their mass from planets.txt: cat planets.txt | grep -v ,, > planets2.txt. Why did this code accomplish that goal?

**In planets.txt, if there were any planets without information about their mass, there would just be “,,” where there would ideally have been a mass value in between the commas. To get rid of these planets, one uses grep to find and -v to say we want to extract all lines that don’t include a “,,” in them. Then, the greater-than symbol places those lines into a file and saves it as planets2.txt.**

1. Write a shell script that can count the number of atoms of a given element (passed as an argument) in all of the .pdb files in the molecules directory of the materials distributed by Software Carpentry. Also provide two lines of code that use your shell script - one that counts hydrogen atoms and one that counts nitrogen atoms.

\*\*# usage: bash molscript.sh \*.pdb\*\*

for file in \*.pdb do echo $file cat $file | grep ATOM | wc -l done

**H atoms:**

for file in \*.pdb do cat $file | grep -w [H] | wc -l done

**C atoms:**

for file in \*.pdb do cat $file | grep -w [C] | wc -l done

\*TA comment:The -w flag on grep or some other method (e.g. cut) to efficiently look only at the third column of the .pdb files is important in this case.

1. Imagine you have a file containing the roster for a large lecture course here at Notre Dame. The file is named fullroster.csv and has columns for first name, last name, email address, and lab section number (1-9). The lab section number corresponds to a lab associated with the lecture course, but these meet in much smaller group sizes than the lecture. There are 9 lab sections, each with their own instructor. Each year you have to make separate rosters for each lab section, and you are sick of copying and pasting in Excel. Write a shell script that generates separate files for each lab section (rostersN.csv, where N is the section number). Feel free to assume that the email addresses do not include numbers.

**# usage: rostersN.csv**

for n in {1..9} do cat fullroster.csv | grep $n > roster$n.csv done

1. You’ve been given a shell script called coolscript.sh with the following content:

\*\*# usage: wc -c $(bash coolscript.sh "\*.txt" 3) | grep -v “total”\*\*

wc -l $(find . -name "$1") | grep -v "total" | sort -n -r | head -n $2 | tr -s [:blank:] | cut -d ' ' -f 3

When you run this as suggested by the usage comment line in data-shell from the Software Carpentry materials the output is:

1053440 ./writing/data/LittleWomen.txt

73861 ./data/sunspot.txt

18034 ./writing/data/two.txt

Describe what happens when you use the script as suggested at the command prompt and also discuss what is happening step-by-step in the pipeline included in coolscript.sh. We will give you one hint tr -s [:blank:]replaces a number of space characters in a row with a single space character.

It prints byte counts and relative paths for three files: LittleWomen.txt, sunspot.txt, and two.txt. In the pipeline, file names ending in .txt are found and lines are counted. A series of commands are done to store command results as an argument. Next, all files without the word total are found. The output is sorted in reverse numerical order. Then, the next step replaces multiple spaces in a row with only one space. Lastly, all but the first three tab-delimited columns are cut. ^TA Comment: When executing the commands in the usage line of the script at the Unix prompt the script call with the two arguments (*.txt and 3) is executed first because it is nested in the $(). Then the output of the script is passed as stdin to ‘wc -c’, which counts the number of bytes in each input file. This output is piped to grep, which removes the line returning the total bytes in all the files and only the size (in bytes) of each .txt files is returned. In the end the script returns the size of the three largest* .txt files anywhere in the file system beneath the working directory.

## *Exercise 4*

1. Provide the Unix code required to return the first five lines containing the word ‘beetle’ from the file insectSightings.txt. Give us two versions of code. One using multiple Unix commands and grep with no flags, and a second version that only uses grep, but can use flags.

cat insectSightings.txt | grep "beetle" | head -n 5

grep -E "beetle" -m 5 insectSightings.txt

1. Provide Unix code to create the reverse complement of the sequence contained in DNA.txt. Give us two versions of code. One version that uses sed and a version that does not use sed.

cat DNA.txt | grep -v ">" | sed 's/ATCG/TAGC/g' | tr -d "\n" | rev | tr 'ATCG' 'TAGC'

cat DNA.txt | grep -v ">" | tr -d "\n" | rev | tr 'ATCG' 'TAGC'

1. Imagine you generated a large amount of sequences of a marker gene for many samples. The sequence ID associated with each sequence in the file contains information about the sample, including the date the sample was collected, but also a bunch of information that only means something to the facility that did the sequencing. You want to generate a list of dates for which you have samples with sequence data. Write a pipeline that generates a list of dates from the file sequences.fasta.

cat seq.txt | grep ">" | cut -f3 -d'\_'

1. Imagine you have a number of tab-delimited text files in a directory. All the files contain 5 columns separated by tabs (\t) and have the file extension .txt. Write a shell script that converts all tab-delimited text files in a directory to comma-delimited files that have the file extension .csv. A couple reminders for you. Recall that (dollarsign)() allows you to assign the output of a Unix pipeline to a shell variable. Also, remember that echo (dollarsign)variable\_name will send the information stored in $variable\_name to stdout, which can be piped into other Unix commands.

**#usage: bash script .sh *.txt >* .csv**

for file in $@

do newFile=$(echo $file | sed 's/.txt/.csv/')

cat $file | tr 't' ',' > $newFile

done

^TA comment: The submitted code returns an error due to the output to \*.csv, which is not defined so the wildcard does not represent anything. The following code converts input tab-delimited .txt files to comma-delimited .csv files: for file in (echo $file | sed ‘s/.txt/.csv/’); cat $file | tr ‘t’ ‘,’ > $newFile; done

1. Write a regular expression that matches the set of character strings or description of character strings below.
2. a sequence motif with variable number of repeats AGCAAAAAAAAAGAT AGGAAAAAAAAAAGAT AGCAAAGAT AGGAAAAAAAAAAGAT AGCAAAAAAGAT
3. any date in either of these formats 2015-05-15 2015/05/15

**a)**

\*\*^"AG{,2}[CG]A{3,}\*GAT"\*\*

**b)**

**“[0-9]{4}(-|/)[0-9]{2}(-|/)[0-9]{2}”**

## *Exercise 5*

Assignment Armed with your new GitHub collaboration skills, work with your partner to develop a single shell 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. 2. Return the following information to stdout when the shell script is executed: 1) the gender, yearsExperience, and wage for the highest earner, 2) the gender, yearsExperience, and wage for the lowest earner, and 3) 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 stdout. 3. Return one more piece of information to stdout: the eﬀect of graduating college (12 vs. 16 years of school) on the minimum wage for earners in this dataset. Hint: you can deﬁne a shell variable with a numeric variable and then us the command bc (“An arbitrary precision calculator language”). If you’ve deﬁned two shell variables (val1 and val2) that are decimal values you can subtract them with the following code: echo “$val1 - $val2” | bc. The owner should start by working on task #1 and collaborator should start by working on task #2. Do this in parrallel, not sequentially. Work together on task #3, and don’t forget to check and edit each other’s code from tasks #1 and #2. 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!!!

**#usage: bash script.sh wages.csv**

cat $1 | grep -v "gender" | cut -d , -f 1,2 | tr ',' ' ' | sort -k1,1d -k2,2n > sorted.txt

echo "highest earner" cat $1 | grep -v 'gender' | sort -t , -k 4 -n | tail -n 1

echo "lowest earner" cat $1 | grep -v 'gender' | sort -t , -k 4 -n | head -n 1

echo "top ten female earners" cat $1 | grep -v 'gender' | sort -t , -k 4 -n | tail -n 10 | grep -c 'female'

echo "effect of graduating college on mimimum wage" VAR1=$(cat $1 | grep -E '[female|male][,][0-9]{1,4}[,]16' | sort -t , -k 4 -n | head -n 1 | cut -d , -f 4) VAR2=$(cat $1 | grep -E '[female|male][,][0-9]{1,4}[,]12' | sort -t, -k 4 -n | head -n 1 | cut -d , -f 4) echo "$VAR1-$VAR2" | bc

\*TA comment: Good work, the solutions for the second and third questions are correct. However, the solution to the first problem is not correctly formatted (-0.25 point). The correct columns are being accessed, but the entries need to be sorted uniquely. For example: cat wages.csv | grep -v gender | cut -d , -f 1,2 | sort -u | tr ‘,’ ’ ’ | sort -k1,1d -k2,2n >> categories.txt.

## *Exercise 6*

1. Write R code that replicates the functionality of the head function we used in Unix. Your code should deﬁne a variable with the ﬁle to return lines from and a variable representing the number of lines to be returned from the top of the indicated ﬁle. The selected ﬁle content should be printed to the terminal in R.
2. Load the data contained in the provided ‘iris.csv’ ﬁle and write R code to do the following things • print the last 2 rows in the last 2 columns to the R terminal • get the number of observations for each species included in the data set • get rows with Sepal.Width > 3.5 • write the data for the species setosa to a comma-delimited ﬁle names ‘setosa.csv’ • calculate the mean, minimum, and maximum of Petal.Length for observations from virginica

#Ex 6  
#load data from iris file  
iris <- read.csv("C:/Users/Owner/Desktop/IBC\_Exercise\_06/iris.csv")  
  
#1  
#iris[y,z] returns value from row y and col z  
iris[1,2]

## [1] 3.5

#code below returns whole first row   
iris[1,1:5]

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1 5.1 3.5 1.4 0.2 setosa

#code below returns whole first 5 rows   
iris[1:5,1:5]

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1 5.1 3.5 1.4 0.2 setosa  
## 2 4.9 3.0 1.4 0.2 setosa  
## 3 4.7 3.2 1.3 0.2 setosa  
## 4 4.6 3.1 1.5 0.2 setosa  
## 5 5.0 3.6 1.4 0.2 setosa

#head(file\_name,number\_of\_lines) where file\_name is n and number\_of\_lines is x; does not include header   
#head(n,x)  
#head(iris,5) returns first 5 lines of iris.csv  
head(iris,5)

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1 5.1 3.5 1.4 0.2 setosa  
## 2 4.9 3.0 1.4 0.2 setosa  
## 3 4.7 3.2 1.3 0.2 setosa  
## 4 4.6 3.1 1.5 0.2 setosa  
## 5 5.0 3.6 1.4 0.2 setosa

#2  
#print last 2 rows in last 2 columns  
iris[149:150,4:5]

## Petal.Width Species  
## 149 2.3 virginica  
## 150 1.8 virginica

#number of observations of each species 50  
iris.species <- iris[, c("Species")]  
iris.spec <- unlist(iris.species)  
table(iris.spec)

## iris.spec  
## setosa versicolor virginica   
## 50 50 50

#get rows with sepal.width > 3.5 [5,6,11,15:17,19,20,22,23,33,34,38,45,47,49,110,118,132]  
  
iris[which(iris[,2]>3.5),]

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 5 5.0 3.6 1.4 0.2 setosa  
## 6 5.4 3.9 1.7 0.4 setosa  
## 11 5.4 3.7 1.5 0.2 setosa  
## 15 5.8 4.0 1.2 0.2 setosa  
## 16 5.7 4.4 1.5 0.4 setosa  
## 17 5.4 3.9 1.3 0.4 setosa  
## 19 5.7 3.8 1.7 0.3 setosa  
## 20 5.1 3.8 1.5 0.3 setosa  
## 22 5.1 3.7 1.5 0.4 setosa  
## 23 4.6 3.6 1.0 0.2 setosa  
## 33 5.2 4.1 1.5 0.1 setosa  
## 34 5.5 4.2 1.4 0.2 setosa  
## 38 4.9 3.6 1.4 0.1 setosa  
## 45 5.1 3.8 1.9 0.4 setosa  
## 47 5.1 3.8 1.6 0.2 setosa  
## 49 5.3 3.7 1.5 0.2 setosa  
## 110 7.2 3.6 6.1 2.5 virginica  
## 118 7.7 3.8 6.7 2.2 virginica  
## 132 7.9 3.8 6.4 2.0 virginica

#write data for species setosa to setosa.csv  
  
setosa.csv <- iris[1:50,]  
  
#find min , mean, max petal lengths for virginica  
  
virginica.csv <- iris[101:150,]  
min(virginica.csv$Petal.Length) #4.5

## [1] 4.5

mean(virginica.csv$Petal.Length)#5.552

## [1] 5.552

max(virginica.csv$Petal.Length)#6.9

## [1] 6.9

#\*TA comments:  
#1. Correct.  
#2. Did not write the setosa data to file (-0.1). You should also be getting the Virginica species using a logical test (-0.05).

## *Exercise 7*

1. Write a function that returns the odd (1, 3, 5, etc.) rows of any dataframe passed as an argument.
2. Repeat a subset of last week’s exercise, but write functions to accomplish these tasks. 2.1. return the number of observations for a given species included in the data set 2.2. return a dataframe for ﬂowers with Sepal.Width greater than a value speciﬁed by the function user 2.3. write the data for a given species to a comma-delimited ﬁle with the given species name as the ﬁle name; Hint: look at paste() to add the .csv extension to your ﬁle in the for-loop.

#Exercise 7 Custom functions in R   
#1. Write a function   
iris <- read.csv("C:/Users/Owner/Desktop/ICB2019\_Exercise07/iris.csv", header = T, stringsAsFactors = F)  
  
#named function only.odd  
only.odd <- function(x){  
 a <- seq(1,nrow(x),2) #assigning the seq as a to use later in for loop; seq(start at row 1,all rows in data frame,every 2 rows(i.e. odd # in this case))   
 x[a, ] #x[sorts odd # rows specific to row]; x[a,1] returns only odd row values for Sepal.Length  
}  
  
#example using iris.csv  
only.odd(iris) #can use function for any data frame in ()

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1 5.1 3.5 1.4 0.2 setosa  
## 3 4.7 3.2 1.3 0.2 setosa  
## 5 5.0 3.6 1.4 0.2 setosa  
## 7 4.6 3.4 1.4 0.3 setosa  
## 9 4.4 2.9 1.4 0.2 setosa  
## 11 5.4 3.7 1.5 0.2 setosa  
## 13 4.8 3.0 1.4 0.1 setosa  
## 15 5.8 4.0 1.2 0.2 setosa  
## 17 5.4 3.9 1.3 0.4 setosa  
## 19 5.7 3.8 1.7 0.3 setosa  
## 21 5.4 3.4 1.7 0.2 setosa  
## 23 4.6 3.6 1.0 0.2 setosa  
## 25 4.8 3.4 1.9 0.2 setosa  
## 27 5.0 3.4 1.6 0.4 setosa  
## 29 5.2 3.4 1.4 0.2 setosa  
## 31 4.8 3.1 1.6 0.2 setosa  
## 33 5.2 4.1 1.5 0.1 setosa  
## 35 4.9 3.1 1.5 0.2 setosa  
## 37 5.5 3.5 1.3 0.2 setosa  
## 39 4.4 3.0 1.3 0.2 setosa  
## 41 5.0 3.5 1.3 0.3 setosa  
## 43 4.4 3.2 1.3 0.2 setosa  
## 45 5.1 3.8 1.9 0.4 setosa  
## 47 5.1 3.8 1.6 0.2 setosa  
## 49 5.3 3.7 1.5 0.2 setosa  
## 51 7.0 3.2 4.7 1.4 versicolor  
## 53 6.9 3.1 4.9 1.5 versicolor  
## 55 6.5 2.8 4.6 1.5 versicolor  
## 57 6.3 3.3 4.7 1.6 versicolor  
## 59 6.6 2.9 4.6 1.3 versicolor  
## 61 5.0 2.0 3.5 1.0 versicolor  
## 63 6.0 2.2 4.0 1.0 versicolor  
## 65 5.6 2.9 3.6 1.3 versicolor  
## 67 5.6 3.0 4.5 1.5 versicolor  
## 69 6.2 2.2 4.5 1.5 versicolor  
## 71 5.9 3.2 4.8 1.8 versicolor  
## 73 6.3 2.5 4.9 1.5 versicolor  
## 75 6.4 2.9 4.3 1.3 versicolor  
## 77 6.8 2.8 4.8 1.4 versicolor  
## 79 6.0 2.9 4.5 1.5 versicolor  
## 81 5.5 2.4 3.8 1.1 versicolor  
## 83 5.8 2.7 3.9 1.2 versicolor  
## 85 5.4 3.0 4.5 1.5 versicolor  
## 87 6.7 3.1 4.7 1.5 versicolor  
## 89 5.6 3.0 4.1 1.3 versicolor  
## 91 5.5 2.6 4.4 1.2 versicolor  
## 93 5.8 2.6 4.0 1.2 versicolor  
## 95 5.6 2.7 4.2 1.3 versicolor  
## 97 5.7 2.9 4.2 1.3 versicolor  
## 99 5.1 2.5 3.0 1.1 versicolor  
## 101 6.3 3.3 6.0 2.5 virginica  
## 103 7.1 3.0 5.9 2.1 virginica  
## 105 6.5 3.0 5.8 2.2 virginica  
## 107 4.9 2.5 4.5 1.7 virginica  
## 109 6.7 2.5 5.8 1.8 virginica  
## 111 6.5 3.2 5.1 2.0 virginica  
## 113 6.8 3.0 5.5 2.1 virginica  
## 115 5.8 2.8 5.1 2.4 virginica  
## 117 6.5 3.0 5.5 1.8 virginica  
## 119 7.7 2.6 6.9 2.3 virginica  
## 121 6.9 3.2 5.7 2.3 virginica  
## 123 7.7 2.8 6.7 2.0 virginica  
## 125 6.7 3.3 5.7 2.1 virginica  
## 127 6.2 2.8 4.8 1.8 virginica  
## 129 6.4 2.8 5.6 2.1 virginica  
## 131 7.4 2.8 6.1 1.9 virginica  
## 133 6.4 2.8 5.6 2.2 virginica  
## 135 6.1 2.6 5.6 1.4 virginica  
## 137 6.3 3.4 5.6 2.4 virginica  
## 139 6.0 3.0 4.8 1.8 virginica  
## 141 6.7 3.1 5.6 2.4 virginica  
## 143 5.8 2.7 5.1 1.9 virginica  
## 145 6.7 3.3 5.7 2.5 virginica  
## 147 6.3 2.5 5.0 1.9 virginica  
## 149 6.2 3.4 5.4 2.3 virginica

#2.1 # of obs for given species   
species.obs<-function(species){  
 speciesobs <- (iris$Species==species)   
 return(sum(speciesobs==T))  
}  
  
#example  
species.obs('versicolor')

## [1] 50

#2.2 get rows with Sepal.Width  
sepalwidth <- function(x,value){  
 x[which(x[,"Sepal.Width"]>value),]  
}  
  
#example  
sepalwidth(iris,3)

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1 5.1 3.5 1.4 0.2 setosa  
## 3 4.7 3.2 1.3 0.2 setosa  
## 4 4.6 3.1 1.5 0.2 setosa  
## 5 5.0 3.6 1.4 0.2 setosa  
## 6 5.4 3.9 1.7 0.4 setosa  
## 7 4.6 3.4 1.4 0.3 setosa  
## 8 5.0 3.4 1.5 0.2 setosa  
## 10 4.9 3.1 1.5 0.1 setosa  
## 11 5.4 3.7 1.5 0.2 setosa  
## 12 4.8 3.4 1.6 0.2 setosa  
## 15 5.8 4.0 1.2 0.2 setosa  
## 16 5.7 4.4 1.5 0.4 setosa  
## 17 5.4 3.9 1.3 0.4 setosa  
## 18 5.1 3.5 1.4 0.3 setosa  
## 19 5.7 3.8 1.7 0.3 setosa  
## 20 5.1 3.8 1.5 0.3 setosa  
## 21 5.4 3.4 1.7 0.2 setosa  
## 22 5.1 3.7 1.5 0.4 setosa  
## 23 4.6 3.6 1.0 0.2 setosa  
## 24 5.1 3.3 1.7 0.5 setosa  
## 25 4.8 3.4 1.9 0.2 setosa  
## 27 5.0 3.4 1.6 0.4 setosa  
## 28 5.2 3.5 1.5 0.2 setosa  
## 29 5.2 3.4 1.4 0.2 setosa  
## 30 4.7 3.2 1.6 0.2 setosa  
## 31 4.8 3.1 1.6 0.2 setosa  
## 32 5.4 3.4 1.5 0.4 setosa  
## 33 5.2 4.1 1.5 0.1 setosa  
## 34 5.5 4.2 1.4 0.2 setosa  
## 35 4.9 3.1 1.5 0.2 setosa  
## 36 5.0 3.2 1.2 0.2 setosa  
## 37 5.5 3.5 1.3 0.2 setosa  
## 38 4.9 3.6 1.4 0.1 setosa  
## 40 5.1 3.4 1.5 0.2 setosa  
## 41 5.0 3.5 1.3 0.3 setosa  
## 43 4.4 3.2 1.3 0.2 setosa  
## 44 5.0 3.5 1.6 0.6 setosa  
## 45 5.1 3.8 1.9 0.4 setosa  
## 47 5.1 3.8 1.6 0.2 setosa  
## 48 4.6 3.2 1.4 0.2 setosa  
## 49 5.3 3.7 1.5 0.2 setosa  
## 50 5.0 3.3 1.4 0.2 setosa  
## 51 7.0 3.2 4.7 1.4 versicolor  
## 52 6.4 3.2 4.5 1.5 versicolor  
## 53 6.9 3.1 4.9 1.5 versicolor  
## 57 6.3 3.3 4.7 1.6 versicolor  
## 66 6.7 3.1 4.4 1.4 versicolor  
## 71 5.9 3.2 4.8 1.8 versicolor  
## 86 6.0 3.4 4.5 1.6 versicolor  
## 87 6.7 3.1 4.7 1.5 versicolor  
## 101 6.3 3.3 6.0 2.5 virginica  
## 110 7.2 3.6 6.1 2.5 virginica  
## 111 6.5 3.2 5.1 2.0 virginica  
## 116 6.4 3.2 5.3 2.3 virginica  
## 118 7.7 3.8 6.7 2.2 virginica  
## 121 6.9 3.2 5.7 2.3 virginica  
## 125 6.7 3.3 5.7 2.1 virginica  
## 126 7.2 3.2 6.0 1.8 virginica  
## 132 7.9 3.8 6.4 2.0 virginica  
## 137 6.3 3.4 5.6 2.4 virginica  
## 138 6.4 3.1 5.5 1.8 virginica  
## 140 6.9 3.1 5.4 2.1 virginica  
## 141 6.7 3.1 5.6 2.4 virginica  
## 142 6.9 3.1 5.1 2.3 virginica  
## 144 6.8 3.2 5.9 2.3 virginica  
## 145 6.7 3.3 5.7 2.5 virginica  
## 149 6.2 3.4 5.4 2.3 virginica

#2.3 data for given species to csv file w given species name as file name  
#Hint: look at paste() to add .csv extension to file in for loop  
species.outputs <- function(species){  
 speciesobs <- iris[iris$Species==species,]  
 write.csv(speciesobs, file = paste(species, ".csv", sep=""))  
}  
  
#to get outputs for each of the 3 species   
species.outputs("virginica")  
species.outputs("versicolor")  
species.outputs("setosa")

## *Exercise 8*

1. Analysis of data surrounding sports teams has grown into a major business for the teams themselves and the media. One cool summary plot that media outlets generate to summarize a game, in this case basketball, is a line graph depicting the cumulative score for each team as a function of time in the game (see below). I ﬁrst saw this plot a number of years ago when reading about the results of a game where the University of Wisconsin (Go Badgers!) played Michigan State University. Using the score-by-score information from this game summarized in “UWvMSU\_1-22-13.txt” generate a graph similar to the one I show above. Don’t worry about how pretty your graph is. Focus more on the control structures required in your script used to generate the plot. A couple tips on this one: • You’ll want to generate a matrix or dataframe with a cumulative score for each team whenever either team scores. • For plotting purposes, keep it simple. There is a function plot(x,y,type=‘l’) in base package, where x and y are vectors and type=‘l’ speciﬁes a line graph. You can add a second line to this graph with lines(x,y). Use the help ﬁle for these functions to ﬁgure out other argument to customize the line types if you would like.
2. Write a game called “guess my number”. The computer will generate a random number between 1 and 100. The user types in a number and the computer replies “lower” if the random number is lower than the guess, “higher” if the random number is higher, and “correct!” if the guess is correct. The player can continue guessing up to 10 times. Here’s an example game where the random number is 79. User inputs are italicized. I’m thinking of a number 1-100… Guess: 40 Higher Guess: 70 Higher Guess: 80 Lower Guess: 77 Higher Guess: 79 Correct! A couple tips on this one: • take a look at the Input/Output reference I gave you for how to get input from the user • sample() is a function that allows for a random selection from a vector containing a set of integers

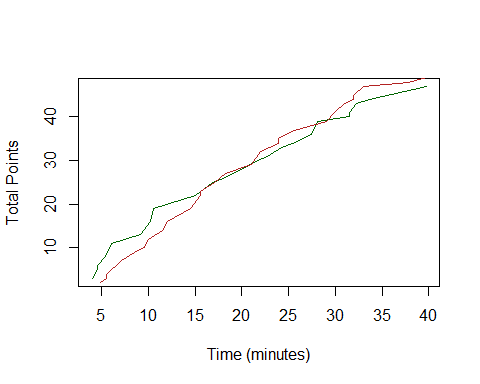
#Exercise 8  
library(ggplot2)  
#move game data to environment  
UWMSUgame <- read.delim("C:/Users/Owner/Desktop/ICB2019\_Exercise08/UWvMSU\_1-22-13.txt")  
  
#a line graph depicting the cumulative score for each team   
#as a function of time in the game, use plot function   
  
#1  
UW<-UWMSUgame[UWMSUgame$team == "UW",] #subset WU   
UW$total<-rep(0,length(UW$score)) #null vector for total  
UW$total[1]<-UW$score[1] #initial  
#For loop   
for(i in 2:length(UW$score)){  
 UW$total[i]<-UW$score[i]+UW$total[i-1] #create running total for total column; adds everything to previous  
}  
#Double check equality   
sum(UW$score) == tail(UW$total,n=1)

## [1] TRUE

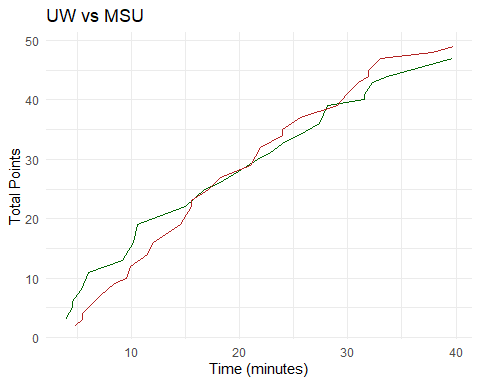
#2  
MSU<-UWMSUgame[UWMSUgame$team == "MSU",] #subset MSU   
MSU$total<-rep(0,length(MSU$score)) #null vector for total  
MSU$total[1]<-MSU$score[1] #initial  
#For loop   
for(i in 2:length(MSU$score)){  
 MSU$total[i]<-MSU$score[i]+MSU$total[i-1] #create running total for total column; adds everything to previous  
}  
#Double check equality   
sum(MSU$score) == tail(MSU$total,n=1)

## [1] TRUE

#plotting with plot  
plot(total ~ time,data=UW,type='l',col='darkgreen', xlab=c("Time (minutes)"), ylab=c("Total Points"))  
lines(total ~ time,data=MSU,col ='firebrick')



#plotting with ggplot  
a=ggplot(data=UW,aes(x=time,y=total))+xlab("Time (minutes)")+ylab("Total Points")+  
 geom\_line(aes(x=time,y=total),colour="darkgreen")+ geom\_line(data=MSU,aes(x=time,y=total),colour="firebrick")+theme\_minimal()+labs(title="UW vs MSU")  
a



#new code  
guessmynum<-function(){   
 n<-readline(prompt="Guess number between 1 and 100: ") #read lines from input  
 n<-as.numeric(n) #creates vector with length equal to n  
 if((as.numeric(n)>0)&(as.numeric(n)<=100)){ #specifies # greater than 0 but less than 100  
 return(as.numeric(n)) #returns the vector  
 }  
 else{  
 cat("Number must be between 1 and 100!")  
 return(guessmynum()) #returns function defined for game variable  
 }  
}  
  
play<-function(){  
 guessmynum<- -100  
 while(guessmynum != num){   
 guess<-guessmynum()  
 if (guess == num) #says that if the guess equals valid number...   
 {  
 cat("Yahoo! You win!", num, "is right!\n") #we get a winning message  
 }  
 else if (guess<num) #if guess is less than valid number...  
 {  
 cat("Shawty, too low. \n") #we get a message prompting us to guess a higher number  
 }  
 else if(guess>num) #if guess is higher than valid number...  
 {  
 cat("Try again, sweetie. Too high. \n")  
 }  
 }   
}  
  
#code for playing game  
#num<-round(runif(1)\*100)  
#play()

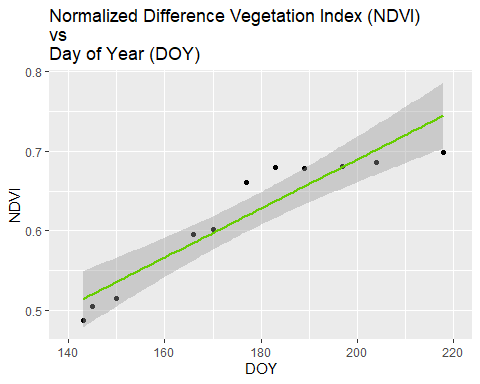
## *Exercise 9*

1. Find some data on two variables that you would expect to be related to each other. These data can come from your own research, your daily life, or the internet. Enter those data into a text ﬁle or Excel and then save a text ﬁle, and write a script that loads this text ﬁle and produces a scatter plot of those two variables that includes a trend line.
2. Given the data in “data.txt”. Write a script that generates two ﬁgures that summarize the data. First, show a barplot of the means of the four populations. Second, show a scatter plot of all of the observations. You may want to “jitter” the points (geom\_jitter()) to make it easier to see all of the observations within a population in your scatter plot. Alternatively, you could also try setting the alpha argument in geom\_scatterplot() to 0.1. Do the bar and scatter plots tell you different stories? Why?

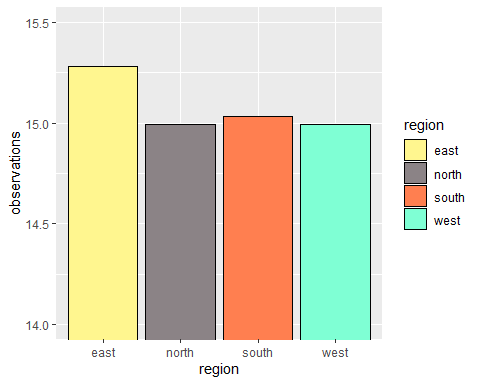
#exercise 09  
  
#1: Use data including 2 related variables; produce a scatter plot w trendline  
#Here, we use NDVI (a vegetation index depicting overall "greenness") and day of year (DOY) up to the onset of senescence (when NDVI begins to decrease)   
#NDVI increases w DOY  
  
asd<-read.csv("C:/Users/Owner/Desktop/IBC\_Exercise\_09/asd\_2019\_ndvi.txt",sep="") #read txt file   
ggplot(data=asd,mapping=aes(x=DOY,y=NDVIm)) +   
 geom\_point() + xlim(140,220) +   
 geom\_smooth(method="lm",color="chartreuse3")+  
 ggtitle("Normalized Difference Vegetation Index (NDVI) \nvs \nDay of Year (DOY)") +   
 ylab("NDVI")

## Warning: Removed 5 rows containing non-finite values (stat\_smooth).

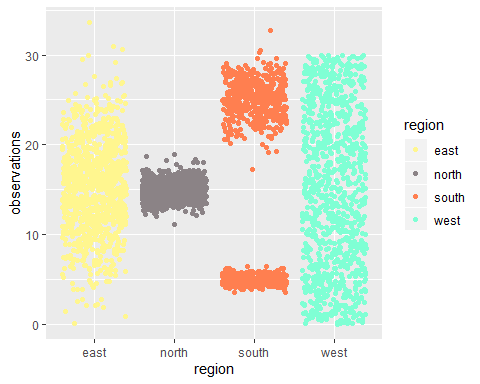
## Warning: Removed 5 rows containing missing values (geom\_point).



#2:figures with data.txt  
#setwd to exercise09 then load in data.txt file   
data <- read.csv("C:/Users/Owner/Desktop/IBC\_Exercise\_09/data.txt")  
  
library(ggplot2)  
# barplot of the means of the four populations  
n <- ggplot(data = data,aes(x=region,y=observations, fill = region))+stat\_summary(geom = "bar", fun.y = "mean",color="black")+  
 scale\_fill\_manual(values = c("khaki1","lavenderblush4","coral","aquamarine"))+  
 coord\_cartesian(ylim=c(14,15.5))  
n



#a scatter plot of all of the observations. "jitter" the points(geom\_jitter())  
u <- ggplot(data = data,aes(x=region,y=observations, color = region))+geom\_jitter()+  
 scale\_color\_manual(values = c("khaki1","lavenderblush4","coral","aquamarine"))  
u

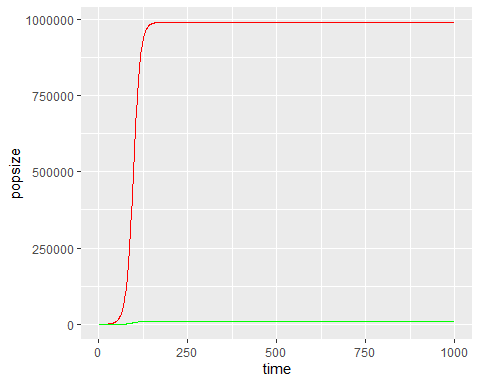


#Do the bar and scatter plots tell you different stories? Why?  
#Answer:  
#Yes, the bar and scatter plots tell us different stories. The bar plot only tells us the mean for each region in order to compare them with each other. We see that they all have similar means. The scatter plot gives us a visual representation of the data distribution within each region. Here, we see that even though they have similar means, the observational scatter for each region varies greatly.

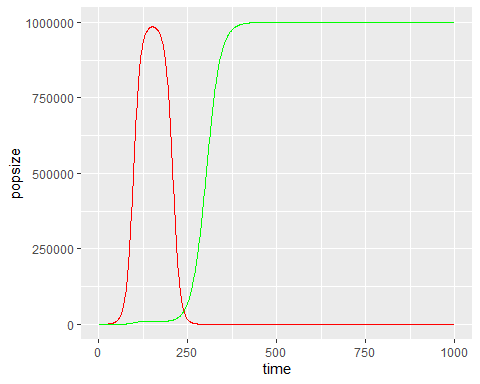
## *Exercise 10*

1. Models not very diﬀerent from the population growth model we worked on this week can be used to explore cancer biology. In this exercise, you will use a population model to investigate evolution of drug resistance in tumors. Imagine a cancer cell in a tumor that spontaneously exhibited a mutation that confers drug resistance. The mutation does not have any positive or negative eﬀects on growth rate of that sub-population when the cancer drug is absent. However, when the cancer drug is present the mutant sub-population grows at 50% of its growth rate in the absence of the drug and the non-mutant sub-population declines rapidly. The model we will use to represent the growth of the two sub-populations is this: Nt = Nt +rNNt(1− (Nt+Mt) K ) Mt = Mt +rMMt(1− (Nt+Mt) K ) Assume in the absence of the cancer drug that the cells grow at a rate of 0.1 per day (rN = rM =0.1) and the carrying capapcity (K) of the tumor is one million cells. The mutation of a single cell occurred early in the tumor growth and when it occurred there were 100 total cells in the tumor. Drug treatment of non-mutant cells results in a negative growth rate of -0.1. Generate a script that simulates growth of the two sub-populations in the tumor to equilibrium followed by drug treatment. Plot your results using a line graph.

#Exercise 10  
#script simulation of pop. and sub pop after cancer drug administration  
#regular pop  
  
#equations for populations non-mutant then mutant  
#N[i] = N[i] + rN\*N[i]\*(1-(N[i]+M[i])/K)  
#M[i] = M[i] + rM\*M[i]\*(1-((N[i]+M[i])/K))  
  
#set parameterrs for model, with drug presence   
N0=99 #starting pop for normal  
M0=1 #starting pop for mutant  
rN=-0.1 #growth rate  
rM=0.1\*(0.5) #growth rate  
K=1e6 #carrying cap  
time=1000  
r=0.1  
  
#changing vector for time to numeric   
N <- numeric(length=time)  
N[1]=N0  
  
M <- numeric(length=time)  
M[1]=M0  
  
#intial loop for regular population growth before drug treatment   
for (i in 1:(time-1)){  
 N[i+1] = N[i] + r\*N[i]\*(1-((N[i]+M[i])/K))  
 M[i+1] = M[i] + r\*M[i]\*(1-((N[i]+M[i])/K))  
}  
#simulate populations before drug to determine equilibrium time to add drug treatment  
simData= data.frame(time=1:time, N=N, M=M)  
ggplot(simData, aes(x=time, y=N))+  
 geom\_line(aes(x=time, y=N), col = "red")+  
 geom\_line(aes(x=time, y = M), col = "green")+  
 labs(x= "time", y= "popsize")



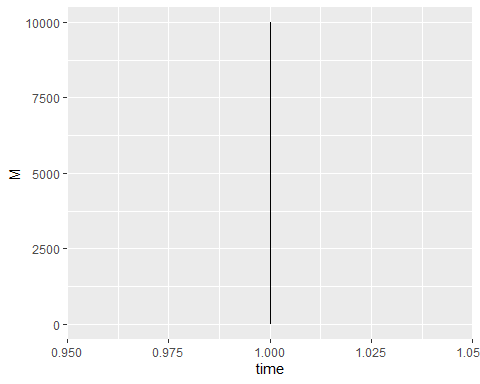
#looks like qrulibrium around 150  
#if else loop to simulate for before and after drug treatment  
#reaches equilirbruim at timestep 200, so drug treatment equation starts at 200 time  
#this should account for new rate in drug treatment   
  
for (i in 1:(time-1)){  
 if(i < 150) {  
 N[i+1] = N[i] + r\*N[i]\*(1-((N[i]+M[i])/K))  
 M[i+1] = M[i] + r\*M[i]\*(1-((N[i]+M[i])/K))  
 }else {  
 N[i+1] = N[i] + rN\*N[i]\*(1-((N[i]+M[i])/K))  
 M[i+1] = M[i] + rM\*M[i]\*(1-((N[i]+M[i])/K))  
 }   
}  
  
  
#simulating data for timesteps  
simData= data.frame(time=1:time, N=N, M=M)  
#practice plots at different times show reaching carrying capacity for  
#nonmutants around 100 so go longer to see effect of drugs,  
#mutants appear at 150 ish so changed time in loop  
#plotting simulation showing the populations over time after drug treatment   
ggplot(simData, aes(x=time, y=N))+  
 geom\_line(aes(x=time, y=N), col = "red")+  
 geom\_line(aes(x=time, y = M), col = "green")+  
 labs(x= "time", y= "popsize")



#Disregard below code, just noted to see thought process for above  
  
#here's what I tried.. kinda like yours but also doesn't work  
N=c() #vectors set for holding values   
M=c()  
N[1]=99 #starting pop for normal  
M[1]=1 #starting pop for mutant  
rN=0.1 #growth rate  
rM=0.1 #growth rate  
K=1e6 #carrying cap  
#f(x) for reg and mutant growth before drug treatment   
for(i in 1:249){  
 N[i+1]=N[i]+rN\*N[i]\*(1-((N[i]+M[i])/K))  
 M[i+1]=M[i]+rM\*M[i]\*(1-((N[i]+M[i])/K))  
}  
#simulate cell model for mutants and nonmutants  
#sim=data.frame(time=1:249,N=N,M=M) #reg and mut in one df  
#plot them  
#a=ggplot()+geom\_line(data=sim,aes(x=time,y=N),color="darkred")+  
#geom\_line(data=sim,aes(x=time,y=M),color="forestgreen")  
#print(a)  
rNd=-0.1 #reg growth rate w drug  
rMd=-0.05 #mut growth rate w drug; 50% of reg  
#f(x) for reg and mut growth after drug treatment  
for(i in 249:999){  
 N[i+1]=N[i]+rNd\*N[i]\*(1-((N[i]+M[i])/K))  
 M[i+1]=M[i]+rMd\*M[i]\*(1-((N[i]+M[i])/K))  
}  
time=seq(1,1000) #set time to end at 1000  
  
#simulate cell model after drug treatment  
#sim2=data.frame(time=1:1000,N=N,M=M)  
#plot  
#b=ggplot()+geom\_line(data=sim,aes(x=time,y=N),color="darkred")+  
#geom\_line(data=sim,aes(x=time,y=M),color="forestgreen")  
#print(b)  
  
library(ggplot2)  
sim=data.frame(time=1:time, M=M, N=N)

## Warning in 1:time: numerical expression has 1000 elements: only the first  
## used

ggplot(sim,aes(x=time,y=M))+geom\_line()



plot(time,N,type="l",col="darkred",)  
lines(time,M,col="forestgreen")

