

The Examples Book

Contents

1	Introduction	5
	How to contribute	5
2	Scholar	15
	Connecting to Scholar	15
	Resources	18
3	Unix	19
	Getting started	19
	Standard utilities	19
	Piping & Redirection	36
	Emacs	39
	Nano	39
	Vim	39
	Writing scripts	39
4	SQL	41
5	R	55
	Getting started	55
	Variables	55
	Logical operators	59
	Lists & Vectors	62
	Basic R functions	65

Data.frames	72
Reading & Writing data	72
Control flow	74
Apply functions	78
Writing functions	79
Plotting	79
RMarkdown	81
Tidyverse	85
data.table	87
SQL in R	87
Scraping	87
shiny	87
6 Python	89
Getting started	89
Lists & Tuples	91
Dicts	91
Control flow	91
Writing functions	91
Reading & Writing data	91
numpy	91
scipy	91
pandas	91
Jupyter notebooks	91
Writing scripts	91
Scraping	91
Plotting	91
Classes	91
tensorflow	91
pytorch	91

7 Tools	93
Docker	93
Tableau	93
GitHub	93
VPNs	106
8 FAQs	107
How do I connect to Scholar from off-campus?	107
Is there an advantage to using the ThinLinc client rather than the ThinLinc web client?	107
GitHub Classroom is not working – can’t authorize the account.	107
In Scholar, on RStudio, my font size looks weird or my cursor is offset.	108
I’m unable to type into the terminal in RStudio.	108
I’m unable to connect to RStudio Server.	108
RStudio initialization error.	108
RStudio crashes when loading a package.	109
RStudio license expired.	109
RStudio is taking a long time to open.	109
How can you run a line of R code in RStudio without clicking the “Run” button?	109
My R session freezes.	110
White screen issue when loading RStudio.	110
Scholar is slow.	110
There are no menus in Scholar.	111
Firefox in Scholar won’t open because multiple instances running.	112
How to transfer files between your computer and Scholar.	113
ThinLinc app says you can’t create any more sessions.	115
How to install ThinLinc on my computer.	116
Forgot my password or password not working with ThinLinc.	116
Jupyter Notebook download error with IE.	116
Jupyter Notebook kernel dying.	116
Python kernel not working, Jupyter Notebook won’t save.	117

Installing <code>my_package</code> for Python	117
Displaying multiple images after a single Jupyter Notebook Python code cell.	118
RMarkdown “Error: option error has NULL value” when knitting“. . .	119
How do you create an RMarkdown file?	119
Problems building an RMarkdown document on Scholar.	119
How can I use SQL in RMarkdown?	120
Copy/paste from terminal inside RStudio to RMarkdown.	121
How do I render an image in a <code>shiny</code> app?	121
The package <code>my_package</code> is not found.	121
Problems installing <code>ggmap</code>	121
Error: <code>object_name</code> is not found	122
Zoom in on <code>ggmap</code>	122
Find the latitude and longitude of a location.	122
Problems saving work as a PDF in R on Scholar.	123
What is a good resource to better understand HTML?	123
Is there a style guide for R code?	123
Is there a guide for best practices using R?	123
Tips for using Jupyter notebooks.	124
What is my username on Scholar?	124
How to submit homework to GitHub without using Firefox?	124
How and why would I need to “escape a character”?	124
How can I fix the error “Illegal byte sequence” when using a UNIX utility like <code>cut</code> ?	125
9 Projects	127
Templates	127
STAT 19000	127
STAT 29000	161
STAT 39000	202
10 Think Summer 2020	241
Project	241

<i>CONTENTS</i>	7
-----------------	---

11 Contributors	249
------------------------	------------

Chapter 1

Introduction

This book contains a collection of examples that students can use to reinforce topics learned in The Data Mine seminar. It is an excellent resource for students to learn what they need to know in order to solve The Data Mine projects.

How to contribute

Contributing to this book is simple:

Small changes and additions

If you have a small change or addition you'd like to make to the book, the easiest way to quickly contribute would be the following method.

1. Navigate to the page or section that needs to be edited
2. Click on the “Edit” button towards the upper left side of the page:



3. You'll be presented with the respective RMarkdown file. Make your modifications.
4. In the “Commit changes” box, select the radio button that says *Create a new branch for this commit and start a pull request*. Give your pull request a title and a detailed description. Name the new branch, and click on “Propose file change”.

5. You’ve successfully submitted a pull request. Our team will review and merge the request shortly thereafter.

Larger changes or additions

If you have larger changes or additions you’d like to make to the book, the easiest way is to edit the contents of the book on your local machine.

Using git in the terminal

1. Setup `git` following the directions here.
2. Start by opening up a terminal and configuring `git` to work with GitHub.
3. Navigate to the directory in which you would like to clone the-examples-book repository. For example, if I wanted to clone the repository in my `~/projects` folder, I’d first execute: `cd ~/projects`.
4. Clone the repository. In this example, let’s assume I’ve cloned the repository into my `~/projects` folder.
5. Navigate into the project folder:

```
cd ~/projects/the-examples-book
```

6. At this point in time your current branch should be the `master` branch. You can verify by running:

```
git branch
```

Note: The highlighted branch starting with “*” is the current branch.

or if you’d like just the name of the branch:

```
git rev-parse --abbrev-ref HEAD
```

7. Create a new branch with whatever name you’d like, and check that branch out. For example, `fix-spelling-errors-01`.
8. Open up RStudio. In the “Files” tab in RStudio, navigate to the repository. In this example, we would navigate to `/Users/kamstut/Documents/GitHub/the-examples-b`. Click on the “More” dropdown and select “Set As Working Directory”.
9. If you do not already have `renv` installed, install it by running the following commands in the console:

```
install.packages("renv")
```

10. Restore the environment by running the following commands in the console:

```
renv::restore()
```

11. In order to compile this book, you must have LaTeX installed. The easiest way to accomplish this is to run the following in the R console:

```
install.packages("tinytex")  
library(tinytex)  
tinytex::install_tinytex()
```

12. In addition, make sure to install both `pandoc` and `pandoc-citeproc` by following the instructions here.
13. Modify the `.Rmd` files to your liking.
14. Click the “Knit” button to compile the book. The resulting “book” is within the “docs” folder.

Important note: If at any point in time you receive an error saying something similar to “there is no package called `my_package`”, simply install the missing package, and try to knit again:

```
install.packages("my_package")  
library(my_package)
```

15. To test the book out, navigate to the “docs” folder and open the `index.html` in the browser of your choice.
16. When you are happy with the modifications you’ve made, commit your changes to the repository.
17. You can continue to make modifications and commit your changes locally. When you are ready, you can push your branch to the remote repository (github.com).
18. At this point in time, you can confirm that the branch has been successfully pushed to github.com by navigating to the repository on github, and click on the “branches” tab:
19. Next, create a pull request. Note that a “Pull Request” is a GitHub-specific concept. You cannot create a pull request using `git`. Navigate to the repository <https://github.com/thedatamine/the-examples-book>, and you should see a message asking if you’d like to create a pull request:

20. Leave a detailed comment about what you've modified or added to the book. You can click on "Preview" to see what your comment will look like. GitHub's markdown applies here. Once satisfied, click "Create pull request".
21. At this point in time, the repository owners will receive a notification and will check and potentially merge the changes into the **master** branch.

Using GitHub Desktop

1. Setup GitHub Desktop following the directions [here](#).
2. When you are presented with the following screen, select "Clone a Repository from the Internet...":



Let's get started!

Add a repository to GitHub Desktop to start collaborating



Create a Tutorial Repository...



Clone a Repository from the Internet...



Create a New Repository on your Hard Drive...



Add an Existing Repository from your Hard Drive...



ProTip! You can drag & drop an existing repository folder here to add it to Desktop

3. Click on the “URL” tab:
 4. In the first field, enter “TheDataMine/the-examples-book”. This is the repository for this book.
 5. In the second field, enter the location in which you’d like the repository to be cloned to. In this example, the repository will be cloned into `/Users/kamstut/Documents/GitHub`. The result will be a new folder

Clone a Repository ×

GitHub.com	GitHub Enterprise Server	URL
Repository URL or GitHub username and repository (hubot/cool-repo) <input type="text" value="URL or username/repository"/>		
Local Path <input type="text" value="/Users/kamstut/Documents/GitHub"/> Choose...		

Cancel Clone

Figure 1.3:

- called `the-examples-book` in `/Users/kamstut/Documents/GitHub`.
6. Click “Clone”.
 7. Upon completion, you will be presented with a screen similar to this:
 8. At this point in time, your current branch will be the `master` branch. Create a new branch with whatever name you’d like. For example, `fix-spelling-errors-01`.
 9. Open up RStudio. In the “Files” tab in RStudio, navigate to the repository. In this example, we would navigate to `/Users/kamstut/Documents/GitHub/the-examples-book`. Click on the “More” dropdown and select “Set As Working Directory”.
 10. If you do not already have `renv` installed, install it by running the following commands in the console:

```
install.packages("renv")
```

11. Restore the environment by running the following commands in the console:

```
renv::restore()
```

12. In order to compile this book, you must have LaTeX installed. The easiest way to accomplish this is to run the following in the R console:



Figure 1.4:

```
install.packages("tinytex")
library(tinytex)
tinytex::install_tinytex()
```

13. In addition, make sure to install both `pandoc` and `pandoc-citeproc` by following the instructions here.
14. Modify the `.Rmd` files to your liking.
15. Click the “Knit” button to compile the book. The resulting “book” is within the “docs” folder.

Important note: If at any point in time you receive an error saying something similar to “there is no package called `my_package`”, simply install the missing package, and try to knit again:

```
install.packages("my_package")
library(my_package)
```

16. To test the book out, navigate to the “docs” folder and open the `index.html` in the browser of your choice.

17. When you are happy with the modifications you've made, commit your changes to the repository.
18. You can continue to make modifications and commit your changes locally. When you are ready, you can publish your branch:



Figure 1.5:

19. Upon publishing your branch, within GitHub Desktop, you'll be presented with the option to create a pull request:
20. At this point in time, the repository owners will receive a notification and will check and potentially merge the changes into the **master** branch.



Figure 1.6:

Chapter 2

Scholar

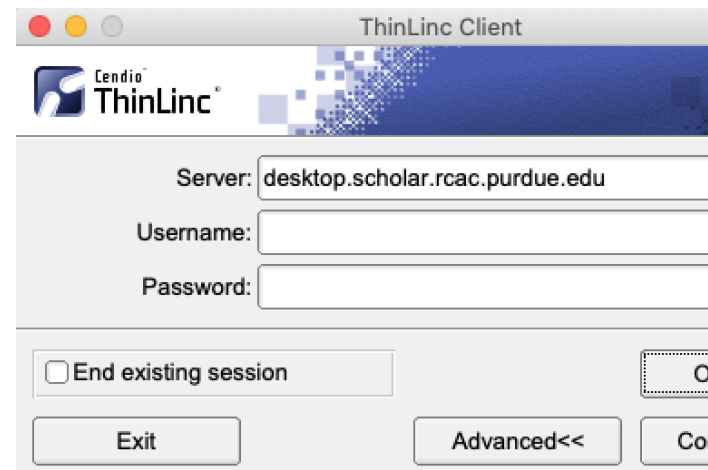
Connecting to Scholar

ThinLinc web client

1. Open a browser and navigating to <https://desktop.scholar.rcac.purdue.edu/>.
2. Login with your Purdue Career Account credentials (**without** BoilerKey).
3. Congratulations, you should now be connected to Scholar using the ThinLinc web client.

ThinLinc client

1. Navigate to <https://www.cendio.com/thinlinc/download>, and download the ThinLinc client application for your operating system.



2. Install and launch the ThinLinc client: **Enter username and password to connect.**
3. Enter your Purdue Career Account information (**without** BoilerKey), as well as the server: `desktop.scholar.rcac.purdue.edu`.
4. Click on “Options...” and fill out the “Screen” tab as shown below:



5. Click “OK” and then “Connect”. **Make sure you are connected to**

Purdue's VPN using AnyConnect before clicking "Connect"!

6. If you are presented with a choice like below, click "Continue".



7. Congratulations, you are now successfully connected to Scholar using the ThinLinc client.

NOTE: If you do accidentally get stuck in full screen mode, the F8 key will help you to escape.

NOTE: The very first time that you log onto Scholar, you will have an option of "use default config" or "one empty panel". PLEASE choose the "use default config".

SSH

Windows

MacOS

Linux

JupyterHub

1. Open a browser and navigate to <https://notebook.scholar.rcac.purdue.edu/>.
2. Enter your Purdue Career Account credentials (**without** BoilerKey).
3. Congratulations, you should now be able to create and run Jupyter notebooks on Scholar!

RStudio Server

1. Open a browser and navigate to <https://rstudio.scholar.rcac.purdue.edu/>.

2. Enter your Purdue Career Account credentials (**without** BoilerKey).
3. Congratulations, you should now be able to create and run R scripts on Scholar!

Resources

Chapter 3

Unix

Getting started

Standard utilities

man

man stand for manual and is a command which presents all of the information you need in order to use a command. To use **man** simply execute **man <command>** where command is the command for which you want to read the manual.

You can scroll up by typing “k” or the up arrow. You can scroll down by typing “j” or the down arrow. To exit the man pages, type “q” (for quit).

How do I show the man pages for the wc utility?

[Click here for solution](#)

```
man wc
```

~ & . & ..

~ represents the location which is in the environment variable \$HOME. If you change \$HOME, ~ also changes. As you are navigating directories, to jump to the most previously visited directory, you can run ~-. For example, if you navigate to /home/\$USER/projects/project1/output, then to /home/\$USER, and you'd like to jump directly back to /home/\$USER/projects/project1/output, simply run ~-. ~- is simply a reference to the location stored in \$OLDPWD.

`.` represents the current working directory. For example, if you are in your home directory `/home/$USER`, `.` means “in this directory”, and `./some_file.txt` would represent a file named `some_file.txt` which is in your home directory `/home/$USER`.

`..` represents the parent directory. For example, `/home` is the parent directory of `/home/$USER`. If you are currently in `/home/$USER/projects` and you want to access some file in the home directory, you could do `../some_file.txt`. `../some_file.txt` is called a *relative* path as it is *relative* to your current location. If we accessed `../some_file.txt` from the home directory, this would be different than accessing `../some_file.txt` from a different directory. `/home/$USER/some_file.txt` is an *absolute* or *full* path of a file `some_file.txt`.

If I am in the directory `/home/kamstut/projects` directory, what is the *relative* path to `/home/mdw`?

[Click here for solution](#)

```
../../mdw
```

If I am in the directory `/home/kamstut/projects/project1`, what is the *absolute* path to the file `../../scripts/runthis.sh`?

[Click here for solution](#)

```
/home/kamstut/scripts/runthis.sh
```

How can I navigate to my `$HOME` directory?

[Click here for solution](#)

```
cd
cd ~
cd $HOME
cd /home/$USER
```

cat

`cat` stands for concatenate and print files. It is an extremely useful tool that prints the entire contents of a file by default. This is especially useful when we want to quickly check to see what is inside of a file. It can be used as a tool to

output the contents of a file and immediately pipe the contents to another tool for some sort of analysis if the other tool doesn't natively support reading the contents from the file.

A similar, but alternative UNIX command that incrementally shows the contents of the file is called **less**. **less** starts at the top of the file and scrolls through the rest of the file as the user pages down.

head

head is a simple utility that displays the first *n* lines of a file, or input.

How do I show the first 5 lines of a file called `input.txt`?

[Click here for solution](#)

```
head -n5 input.txt
```

Alternatively:

```
cat input.txt | head -n5
```

tail

tail is a similar utility to **head**, that displays the last *n* lines of a file, or input.

How do I show the last 5 lines of a file called `input.txt`?

[Click here for solution](#)

```
tail -n5 input.txt
```

Alternatively:

```
cat input.txt | tail -n5
```

ls

ls is a utility that lists files and folders. By default, **ls** will list the files and folders in your current working directory. To list files in a certain directory, simply provide the directory to **ls** as the first argument.

How do I list the files in my \$HOME directory?

[Click here for solution](#)

```
ls $HOME
```

```
# or
```

```
ls ~
```

How do I list the files in the directory /home/\$USER/projects?

[Click here for solution](#)

```
ls /home/$USER/projects
```

How do I list all files and folders, including hidden files and folders in /home/\$USER/projects?

[Click here for solution](#)

```
ls -a /home/$USER/projects
```

How do I list all files and folders in /home/\$USER/projects in a list format, including information like permissions, filesize, etc?

[Click here for solution](#)

```
ls -l /home/$USER/projects
```

How do I list all files and folders, including hidden files and folders in /home/\$USER/projects in a list format, including information like permissions, filesize, etc?

[Click here for solution](#)

```
ls -la /home/$USER/projects
```

```
# or
```

```
ls -al /home/$USER/projects
```

```
# or  
ls -l -a /home/$USER/projects
```

cp

cp is a utility used for copying files and folders from one location to another.

How do I copy /home/\$USER/some_file.txt to /home/\$USER/projects/same_file.txt?

[Click here for solution](#)

```
cp /home/$USER/some_file.txt /home/$USER/projects/same_file.txt  
  
# If currently in /home/$USER  
cd $HOME  
cp some_file.txt projects/same_file.txt  
  
# If currently in /home/$USER/projects  
cd $HOME/projects  
cp ../some_file.txt .
```

mv

mv very similar to cp, but rather than copy a file, mv moves the file. Moving a file removes it from its old location and places it in the new location.

How do I move /home/\$USER/some_file.txt to /home/\$USER/projects/same_file.txt?

[Click here for solution](#)

```
mv /home/$USER/some_file.txt /home/$USER/projects/same_file.txt  
  
# If currently in /home/$USER  
cd $HOME  
mv some_file.txt projects/same_file.txt  
  
# If currently in /home/$USER/projects  
cd $HOME/projects  
mv ../some_file.txt .
```

pwd

pwd stands for print working directory and it does just that – it prints the current working directory to standard output.

type

type is a useful command to find the location of some command, or whether the command is an alias, function, or something else.

Where is the file that is executed when I type `ls`?

[Click here for solution](#)

```
type ls
```

```
## ls is /bin/ls
```

uniq

uniq reads the lines of a specified input file and compares each adjacent line and returns each unique line. Repeated lines in the input will not be detected if they are not adjacent. What this means is you must sort prior to using **uniq** if you want to ensure you have no duplicates.

wc

You can think of **wc** as standing for “word count”. **wc** displays the number of lines, words, and bytes from the input file.

How do I count the number of lines of an input file called `input.txt`?

[Click here for solution](#)

```
wc -l input.txt
```

How do I count the number of characters of an input file called `input.txt`?

[Click here for solution](#)

```
wc -m input.txt
```

How do I count the number of words of an input file called `input.txt`?

[Click here for solution](#)

```
wc -w input.txt
```

ssh

mosh

scp

cut

cut is a tool to cut out parts of a line based on position/character/delimiter/etc and directing the output to stdout. It is particularly useful to get a certain column of data.

How do I get the first column of a csv file called `'office.csv'`?

[Click here for solution](#)

```
cut -d, -f1 office.csv
```

How do I get the first and third column of a csv file called `'office.csv'`?

[Click here for solution](#)

```
cut -d, -f1,3 office.csv
```

How do I get the first and third column of a file with columns separated by the `"|"` character?

[Click here for solution](#)

```
cut -d '|' -f1,3 office.csv
```

awk

awk is a powerful programming language that specializes in processing and manipulating text data.

In awk, a command looks something like this:

```
awk -F, 'BEGIN{ } { } END{ }'
```

The delimiter is specified with the `-F` option (in this case our delimiter is a comma). The `BEGIN` chunk is run only once at the start of execution. The middle chunk is run once per line of the file. The `END` chunk is run only once, at the end of execution.

The `BEGIN` and `END` portions are always optional.

The variables: `$1`, `$2`, `$3`, etc., refer to the 1st, 2nd, and 3rd fields in a line of data. For example, the following would print the 4th field of every row in a csv file:

```
awk -F, '{print $4}'
```

`$0` represents the entire row.

awk is very powerful. We can achieve the same effect as using `cut`:

```
head 5000_products.csv | cut -d, -f3
```

or

```
head 5000_products.csv | awk -F, '{print $3}'
```

Examples

How do I print only rows where the DAYOFWEEK is 5?

[Click here for solution](#)

```
head metadata.csv | awk -F, '{if ($3 == 5) {print $0}}'
```

```
## 01/01/2015,,5,0,0,1,2015,CHRISTMAS PEAK,0,5,nyd,1,,,0,0,CHRISTMAS PEAK,73.02,59.81
## 01/08/2015,,5,7,1,1,2015,CHRISTMAS,8,0,,0,,marwk,,0,1,CHRISTMAS,59.44,38.7,49.07,,0
```

How do I print the first, fourth, and fifth columns of rows where the DAYOFWEEK is 5?

[Click here for solution](#)

```
head metadata.csv | awk -F, '{if ($3 == 5) {print $1, $4, $5}}'
```

```
## 01/01/2015 0 0
## 01/08/2015 7 1
```

How do I print only rows where DAYOFWEEK is 5 OR YEAR is 2015?

[Click here for solution](#)

```
head metadata.csv | awk -F, '{if ($3 == 5 || $7 == 2015) {print $0}}'
```

```
## 01/01/2015,,5,0,0,1,2015,CHRISTMAS PEAK,0,5,nyd,1,,,0,0,CHRISTMAS PEAK,73.02,59.81,66.41,,0,,
## 01/02/2015,,6,1,0,1,2015,CHRISTMAS,2,5,,0,,,0,0,CHRISTMAS,78,60.72,69.36,,0,,0,,0,,0,,0%,0%,0
## 01/03/2015,,7,2,0,1,2015,CHRISTMAS,3,0,,0,,,0,0,CHRISTMAS,83.12,67.31,75.22,,0,,0,,0,,0,,0%,0
## 01/04/2015,,1,3,1,1,2015,CHRISTMAS,4,0,,0,,,0,0,CHRISTMAS,83.93,67.97,75.95,,0,,0,,0,,0,,67%,
## 01/05/2015,,2,4,1,1,2015,CHRISTMAS,5,0,,0,,,0,0,CHRISTMAS,72.3,56.89,64.6,,0,,0,,0,,0,,67%,74
## 01/06/2015,,3,5,1,1,2015,CHRISTMAS,6,0,,0,,,0,0,CHRISTMAS,77.67,54.88,66.28,,0,,0,,0,,0,,86%,
## 01/07/2015,,4,6,1,1,2015,CHRISTMAS,7,0,,0,,marwk,,0,1,CHRISTMAS,67.24,48.56,57.9,,0,,0,,0,,0,,
## 01/08/2015,,5,7,1,1,2015,CHRISTMAS,8,0,,0,,marwk,,0,1,CHRISTMAS,59.44,38.7,49.07,,0,,0,,0,,0,,
## 01/09/2015,,6,8,1,1,2015,CHRISTMAS,9,0,,0,,marwk,,0,1,CHRISTMAS,54.89,45.37,50.13,,0,,0,,0,,0,,
```

How do I print only rows where DAYOFWEEK is 5 AND YEAR is 2015?

[Click here for solution](#)

```
head metadata.csv | awk -F, '{if ($3 == 5 && $7 == 2015) {print $0}}'
```

```
## 01/01/2015,,5,0,0,1,2015,CHRISTMAS PEAK,0,5,nyd,1,,,0,0,CHRISTMAS PEAK,73.02,59.81,66.41,,0,,
## 01/08/2015,,5,7,1,1,2015,CHRISTMAS,8,0,,0,,marwk,,0,1,CHRISTMAS,59.44,38.7,49.07,,0,,0,,0,,0,,
```

How do I get the average of values in a column containing the max temperature, WDWMAXTEMP?

[Click here for solution](#)

Here NR represents the number of rows

```
head metadata.csv | awk -F, '{sum = sum + $19}END{print "Average max temp: " sum/NR}'
```

Or alternatively we could track the number of rows as we go

```
head metadata.csv | awk -F, '{sum = sum + $19; count++}END{print "Average max temp: " sum/count}'
```

```
## Average max temp: 64.961
## Average max temp: 64.961
```

How do I get counts of each unique value in a column, SEASON?

[Click here for solution](#)

```
cat metadata.csv | awk -F, '{seasons[$8]++}END{for (season in seasons) {print season, s
```

```
## SUMMER BREAK 236
## CHRISTMAS 245
## JERSEY WEEK 50
## SEPTEMBER LOW 140
## PRESIDENTS WEEK 55
## FALL 212
## HALLOWEEN 26
## MEMORIAL DAY 20
## CHRISTMAS PEAK 176
## SEASON 1
## COLUMBUS DAY 20
## SPRING 490
## THANKSGIVING 60
## EASTER 95
## MARTIN LUTHER KING JUNIOR DAY 45
## MARDI GRAS 15
## JULY 4TH 25
## WINTER 222
```

sed

grep

It is very simple to get started searching for patterns in files using **grep**.

How do I search for lines with the word “Exact” in the file located /home/john/report.txt?

[Click here for solution](#)

```
grep Exact /home/john/report.txt
```

or

```
grep "Exact" "/home/john/report.txt"
```


How do I search for lines with the word “Exact” or “exact” in the file located `/home/john/report.txt`?

[Click here for solution](#)

```
# The -i option means that the text we are searching for is  
# not case-sensitive. So the following lines will match  
# lines that contain "Exact" or "exact" or "ExAcT".  
grep -i Exact /home/john/report.txt  
  
# or  
  
grep -i "Exact" "/home/john/report.txt"
```

How do I search for lines with a string containing multiple words, like “how do I”?

[Click here for solution](#)

```
# The -i option means that the text we are searching for is  
# not case-sensitive. So the following lines will match  
# lines that contain "Exact" or "exact" or "ExAcT".  
  
# By adding quotes, we are able to search for the entire  
# string "how do i". Without the quotes this would only  
# search for "how".  
grep -i "how do i" /home/john/report.txt
```

How do I search for lines with the word “Exact” or “exact” in the files in the folder and all sub-folders located `/home/john/`?

[Click here for solution](#)

```
# The -R option means to search recursively in the folder  
# /home/john. A recursive search means that it will search  
# all folders and sub-folders starting with /home/john.  
grep -Ri Exact /home/john
```

How do I search for the lines that don’t contain the words “Exact” or “exact” in the folder and all sub-folders located `/home/john/`?

[Click here for solution](#)

```
# The -v option means to search for an inverted match.  
# In this case it means search for all lines of text  
# where the word "exact" is not found.  
grep -Rvi Exact /home/john
```

How do I search for lines where one or more of the words “first” or “second” appears in the current folder and all sub-folders?

[Click here for solution](#)

```
# The "/" character in grep is the logical OR operator.  
# If we do not escape the "/" character with a preceding  
# "\" grep searches for the literal string "first/second"  
# instead of "first" OR "second".  
grep -Ri "first\\|second" .
```

How do I search for lines that begin with the word “Exact” (case insensitive) in the folder and all sub-folders located in the current directory?

[Click here for solution](#) The “^” is called an anchor and indicates the start of a line.

```
grep -Ri "^Exact" .
```

How do I search for lines that end with the word “Exact” (case insensitive) in the files in the current folder and all sub-folders?

[Click here for solution](#) The “\$” is called an anchor and indicates the end of a line.

```
grep -Ri "Exact$" .
```

How do I search for lines that contain only the word “Exact” (case insensitive) in the files in the current folder and all sub-folders?

[Click here for solution](#)

```
grep -Ri "^Exact$" .
```

How do I search for strings or sub-strings where the first character could be anything, but the next two characters are “at”? For example: “cat”, “bat”, “hat”, “rat”, “pat”, “mat”, etc.

Click here for solution The “.” is a wildcard, meaning it matches any character (including spaces).

```
grep -Ri ".at" .
```

How do I search for zero or one of, zero or more of, one or more of, exactly *n* of a certain character using grep and regular expressions?

Click here for solution “*” stands for 0+ of the previous character. “+” stands for 1+ of the previous character. “?” stands for 0 or 1 of the previous character. “{n}” stands for exactly n of the previous character.

```
# Matches any lines with text like "cat", "bat", "hat", "rat", "pat", "mat", etc.  
# Does NOT match "at", but does match " at". The "." indicates a single character.  
grep -i ".at" .
```

```
# Matches any lines with text like "cat", "bat", "hat", "rat", "pat", "mat", etc.  
# Matches "at" as well as " at". The "." followed by the "?" means  
# 0 or 1 of any character.  
grep -i ".*?at" .
```

```
# Matches any lines with any amount of text followed by "at".  
grep -i ".*at" .
```

```
# Only matches words that end in "at": "bat", "cat", "spat", "at". Does not match "spatula".  
grep -i ".*at$" .
```

```
# Matches lines that contain consecutive "e"'s.  
grep -i ".*e{2}.*" .
```

```
# Matches any line. 0+ of the previous character, which in this case is the wildcard "."  
# that represents any character. So 0+ of any character.  
grep -i ".*"
```

Resources

Regex Tester

<https://regex101.com/> is an excellent tool that helps you quickly test and better understand writing regular expressions. It allows you to test four different “flavors” or regular expressions: PCRE (PHP), ECMAScript (JavaScript), Python, and Golang. regex101 also provides a library of useful, pre-made regular expressions.

Lookahead and Lookbehinds

This is an excellent resource to better understand positive and negative lookahead and lookbehind operations using `grep`.

ripgrep

`ripgrep` is a “line-oriented search tool that recursively searches your current directory for a regex pattern.” You can read about why you may want to use `ripgrep` here. Generally, `ripgrep` is frequently faster than `grep`. If you are working with code it has sane defaults (respects `.gitignore`). You can easily search for specific types of files.

How do I exclude a filetype when searching for foo in my_directory?

[Click here for solution](#)

```
# exclude javascript (.js) files
rg -Tjs foo my_directory

# exclude r (.r) files
rg -Tr foo my_directory

# exclude Python (.py) files
rg -Tpy foo my_directory
```

How do I search for a particular filetype when searching for foo in my_directory?

[Click here for solution](#)

```
# search javascript (.js) files
rg -tjs foo my_directory

# search r (.r) files
rg -tr foo my_directory

# search Python (.py) files
rg -tpy foo my_directory
```

How do I search for a specific word, where the word isn't part of another word?

[Click here for solution](#)

```
# this is roughly equivalent to putting \b before and after all search patterns in grep  
rg -w foo my_directory
```

How do I replace every match foo in my_directory with the text given, bar, when printing results?

[Click here for solution](#)

```
rg foo my_directory -r bar
```

How do I trim whitespace from the beginning and ending of each printed line?

[Click here for solution](#)

```
rg foo my_directory --trim
```

How do I follow symbolic links when searching a directory, my_directory?

[Click here for solution](#)

```
rg -L foo my_directory
```

find

find is an aptly named tool that traverses directories and searches for files.

Examples

How do I find a file named foo.txt in the current working directory or subdirectories?

[Click here for solution](#)

```
find . -name foo.txt
```

How do I find a file named `foo.txt` or `Foo.txt` or `Fo0.txt` (i.e. ignoring case) in the current working directory or subdirectories?

[Click here for solution](#)

```
find . -iname foo.txt

# or

find . -i -name foo.txt
```

How do I find a directory named `foo` in the current working directory or subdirectories?

[Click here for solution](#)

```
find . -type d -name foo
```

How do I find all of the Python files in the current working directory or subdirectories?

[Click here for solution](#)

```
find . -name "*.py"
```

How do I find files over 1gb in size in the current working directory or subdirectories?

[Click here for solution](#)

```
find . -size +1G
```

How do I find files under 10mb in size in the current working directory or subdirectories?

[Click here for solution](#)

```
find . -size -10M
```

less

`less` is a utility that opens a page of text from a file and allows the user to scroll forward or backward in the file using “j” and “k” keys or down and up arrows. `less` does not read the entire file into memory at once, and is therefore faster when loading large files.

How do I display the contents of a file, foo.txt?

[Click here for solution](#)

```
less foo.txt
```

How do I scroll up and down in less?

[Click here for solution](#) To scroll down use “j” or the down arrow. To scroll up use “k” or the up arrow.

How do I exit less?

[Click here for solution](#) Press the “q” key on your keyboard.

sort

sort is a utility that sorts lines of text.

Examples**How do I sort a csv, test.csv alphabetically by the 18th column?**

[Click here for solution](#)

```
# the r option sorts ascending
sort -t, -k18,18 test.csv
```

```
## 1990,10,18,7,729,730,847,849,PS,1451,NA,78,79,NA,-2,-1,SAN,ABC,447,NA,NA,0,NA,0,NA,NA,NA,NA,NA,NA
## 1991,10,19,1,749,730,922,849,PS,1451,NA,93,79,NA,33,19,SAN,ABC,447,NA,NA,0,NA,0,NA,NA,NA,NA,NA,NA
## 1991,10,21,3,728,730,848,849,PS,1451,NA,80,79,NA,-1,-2,SAN,ABC,447,NA,NA,0,NA,0,NA,NA,NA,NA,NA,NA
## 1991,10,22,4,728,730,852,849,PS,1451,NA,84,79,NA,3,-2,SAN,ABC,447,NA,NA,0,NA,0,NA,NA,NA,NA,NA,NA
## 1991,10,23,5,731,730,902,849,PS,1451,NA,91,79,NA,13,1,SAN,ABC,447,NA,NA,0,NA,0,NA,NA,NA,NA,NA,NA
## 1991,10,24,6,744,730,908,849,PS,1451,NA,84,79,NA,19,14,SAN,ABC,447,NA,NA,0,NA,0,NA,NA,NA,NA,NA,NA
## Year,Month,DayOfMonth,DayOfWeek,DepTime,CRSDepTime,ArrTime,CRSArrTime,UniqueCarrier,FlightNum,
## 1987,10,14,3,741,730,912,849,PS,1451,NA,91,79,NA,23,11,SAN,SFO,447,NA,NA,0,NA,0,NA,NA,NA,NA,NA,NA
## 1990,10,15,4,729,730,903,849,PS,1451,NA,94,79,NA,14,-1,SAN,SFO,447,NA,NA,0,NA,0,NA,NA,NA,NA,NA,NA
## 1990,10,17,6,741,730,918,849,PS,1451,NA,97,79,NA,29,11,SAN,SFO,447,NA,NA,0,NA,0,NA,NA,NA,NA,NA,NA
```

How do I sort a csv, `test.csv` alphabetically by the 18th column, and then in descending order by the 4th column?

[Click here for solution](#)

```
sort -t, -k18,18 -k4,4r test.csv
```

```
## 1990,10,18,7,729,730,847,849,PS,1451,NA,78,79,NA,-2,-1,SAN,ABC,447,NA,NA,0,NA,0,NA,I
## 1991,10,24,6,744,730,908,849,PS,1451,NA,84,79,NA,19,14,SAN,ABC,447,NA,NA,0,NA,0,NA,I
## 1991,10,23,5,731,730,902,849,PS,1451,NA,91,79,NA,13,1,SAN,ABC,447,NA,NA,0,NA,0,NA,N
## 1991,10,22,4,728,730,852,849,PS,1451,NA,84,79,NA,3,-2,SAN,ABC,447,NA,NA,0,NA,0,NA,N
## 1991,10,21,3,728,730,848,849,PS,1451,NA,80,79,NA,-1,-2,SAN,ABC,447,NA,NA,0,NA,0,NA,I
## 1991,10,19,1,749,730,922,849,PS,1451,NA,93,79,NA,33,19,SAN,ABC,447,NA,NA,0,NA,0,NA,I
## Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDepTime,ArrTime,CRSArrTime,UniqueCarrier
## 1990,10,17,6,741,730,918,849,PS,1451,NA,97,79,NA,29,11,SAN,SFO,447,NA,NA,0,NA,0,NA,I
## 1990,10,15,4,729,730,903,849,PS,1451,NA,94,79,NA,14,-1,SAN,SFO,447,NA,NA,0,NA,0,NA,I
## 1987,10,14,3,741,730,912,849,PS,1451,NA,91,79,NA,23,11,SAN,SFO,447,NA,NA,0,NA,0,NA,I
```

git

See [here](#).

Piping & Redirection

Redirection is the act of writing standard input (stdin) or standard output (stdout) or standard error (stderr) somewhere else. stdin, stdout, and stderr all have numeric representations of 0, 1, & 2 respectively.

Redirection

Examples

For the following examples we use the example file `redirection.txt`. The contents of which are:

```
cat redirection.txt
```

```
## This is a simple file with some text.
## It has a couple of lines of text.
## Here is some more.
```


How do I redirect text from a command like `ls` to a file like `redirection.txt`, completely overwriting any text already within `redirection.txt`?

[Click here for solution](#)

```
# Save the stdout from the ls command to redirection.txt
ls > redirection.txt
```

```
# The new contents of redirection.txt
head redirection.txt
```

```
## 01-scholar.Rmd
## 02-unix.Rmd
## 03-sql.Rmd
## 04-r.Rmd
## 05-python.Rmd
## 06-tools.Rmd
## 07-faqs.Rmd
## 08-projects.Rmd
## 09-think-summer-2020.Rmd
## 10-contributors.Rmd
```

How do I redirect text from a command like `ls` to a file like `redirection.txt`, without overwriting any text, but rather appending the text to the end of the file?

[Click here for solution](#)

```
# Append the stdout from the ls command to the end of redirection.txt
ls >> redirection.txt
```

```
head redirection.txt
```

```
## This is a simple file with some text.
## It has a couple of lines of text.
## Here is some more.
## 01-scholar.Rmd
## 02-unix.Rmd
## 03-sql.Rmd
## 04-r.Rmd
## 05-python.Rmd
## 06-tools.Rmd
## 07-faqs.Rmd
```

How can I redirect text from a file to be used as stdin for another program or command?

[Click here for solution](#)

```
# Let's count the number of words in redirection.txt  
wc -w < redirection.txt
```

```
## 20
```

How can I use multiple redirects in a single line?

[Click here for solution](#)

```
# Here we count the number of words in redirection.txt and then  
# save that value to value.txt.  
wc -w < redirection.txt > value.txt
```

```
head value.txt
```

```
## 20
```

Piping

Piping is the act of taking the output of one or more commands and making the output the input of another command. This is accomplished using the “|” character.

Examples

For the following examples we use the example file `piping.txt`. The contents of which are:

```
cat piping.txt
```

```
## apples, oranges, grapes  
## pears, apples, peaches,  
## celery, carrots, peanuts  
## fruits, vegetables, ok
```

How can I use the output from a `grep` command to another command?

[Click here for solution](#)

```
grep -i "p\{2\}" piping.txt | wc -w
```

```
## 6
```

How can I chain multiple commands together?

[Click here for solution](#)

```
# Get the third column of piping.txt and  
# get all lines that end in "s" and sort  
# the words in reverse order, and append  
# to a file called food.txt.  
cut -d, -f3 piping.txt | grep -i ".*s$" | sort -r > food.txt
```

Resources

Intro to I/O Redirection

A quick introduction to stdin, stdout, stderr, redirection, and piping.

Emacs

Nano

Vim

Writing scripts

Chapter 4

SQL

```
library(RMariaDB)
library(RSQLite)
library(DBI)

# Establish a connection to sqlite databases
chinook <- dbConnect(RSQLite::SQLite(), "chinook.db")
lahman <- dbConnect(RSQLite::SQLite(), "lahman.db")

# Establish a connection to mysql databases
connection <- dbConnect(RMariaDB::MariaDB(),
                        host="your-host.com",
                        db="your-database-name",
                        user="your-username",
                        password="your-password")
```

RDBMS

SQL in R

Examples

Please see [here](#) for a variety of examples demonstrating using SQL within R.

Table 4.1: 8 records

EmployeeId	LastName	FirstName	Title	ReportsTo	BirthDate	Hi
1	Adams	Andrew	General Manager	NA	1962-02-18 00:00:00	200
2	Edwards	Nancy	Sales Manager	1	1958-12-08 00:00:00	200
3	Peacock	Jane	Sales Support Agent	2	1973-08-29 00:00:00	200
4	Park	Margaret	Sales Support Agent	2	1947-09-19 00:00:00	200
5	Johnson	Steve	Sales Support Agent	2	1965-03-03 00:00:00	200
6	Mitchell	Michael	IT Manager	1	1973-07-01 00:00:00	200
7	King	Robert	IT Staff	6	1970-05-29 00:00:00	200
8	Callahan	Laura	IT Staff	6	1968-01-09 00:00:00	200

SQL in Python

Examples

The following examples use the `chinook.db` sqlite database.

```
dbListTables(chinook)
```

```
## [1] "advisors"      "albums"        "artists"        "customers"
## [5] "employees"     "genres"        "invoice_items"  "invoices"
## [9] "media_types"  "playlist_track" "playlists"      "sqlite_sequence"
## [13] "sqlite_stat1"  "students"      "tracks"
```

How do I select all of the rows of a table called employees?

[Click here for solution](#)

```
SELECT * FROM employees;
```

How do I select the first 5 rows of a table called employees?

[Click here for solution](#)

```
SELECT * FROM employees LIMIT 5;
```

Table 4.2: 5 records

EmployeeId	LastName	FirstName	Title	ReportsTo	BirthDate	HireDate
1	Adams	Andrew	General Manager	NA	1962-02-18 00:00:00	2002-08-14 00:00:00
2	Edwards	Nancy	Sales Manager	1	1958-12-08 00:00:00	2002-05-01 00:00:00
3	Peacock	Jane	Sales Support Agent	2	1973-08-29 00:00:00	2002-04-01 00:00:00
4	Park	Margaret	Sales Support Agent	2	1947-09-19 00:00:00	2003-05-03 00:00:00
5	Johnson	Steve	Sales Support Agent	2	1965-03-03 00:00:00	2003-10-17 00:00:00

Table 4.3: 8 records

LastName	FirstName
Adams	Andrew
Edwards	Nancy
Peacock	Jane
Park	Margaret
Johnson	Steve
Mitchell	Michael
King	Robert
Callahan	Laura

How do I select specific rows of a table called employees?

[Click here for solution](#)

```
SELECT LastName, FirstName FROM employees;
```

You can switch the order in which the columns are displayed as well:

```
SELECT FirstName, LastName FROM employees;
```

How do I select only unique values from a column?

[Click here for solution](#)

```
SELECT DISTINCT Title FROM employees;
```

How can I filter that match a certain criteria?

[Click here for solution](#)

Select only employees with a FirstName “Steve”:

Table 4.4: 8 records

FirstName	LastName
Andrew	Adams
Nancy	Edwards
Jane	Peacock
Margaret	Park
Steve	Johnson
Michael	Mitchell
Robert	King
Laura	Callahan

Table 4.5: 5 records

Title
General Manager
Sales Manager
Sales Support Agent
IT Manager
IT Staff

```
SELECT * FROM employees WHERE FirstName='Steve';
```

Select only employees with FirstName “Steve” OR FirstName “Laura”:

```
SELECT * FROM employees WHERE FirstName='Steve' OR FirstName='Laura';
```

Select only employees with FirstName “Steve” AND LastName “Laura”:

```
SELECT * FROM employees WHERE FirstName='Steve' AND LastName='Laura';
```

As expected, there are no results! There is nobody with the full name “Steve Laura”.

List the first 10 tracks from the tracks table.

[Click here for solution](#)

Table 4.6: 1 records

EmployeeId	LastName	FirstName	Title	ReportsTo	BirthDate	Hi
5	Johnson	Steve	Sales Support Agent	2	1965-03-03 00:00:00	200

Table 4.7: 2 records

EmployeeId	LastName	FirstName	Title	ReportsTo	BirthDate	HireDate
5	Johnson	Steve	Sales Support Agent	2	1965-03-03 00:00:00	2003-10-17 00:00:00
8	Callahan	Laura	IT Staff	6	1968-01-09 00:00:00	2004-03-04 00:00:00

Table 4.8: 0 records

EmployeeId	LastName	FirstName	Title	ReportsTo	BirthDate	HireDate	Address	City	State	Country
------------	----------	-----------	-------	-----------	-----------	----------	---------	------	-------	---------

```
SELECT * FROM tracks LIMIT 10;
```

How many rows or records are in the table named tracks?

[Click here for solution](#)

```
SELECT COUNT(*) FROM tracks;
```

Are there any artists with the names: “Elis Regina”, “Seu Jorge”, or “The Beatles”?

[Click here for solution](#)

```
SELECT * FROM artists WHERE Name='Elis Regina' OR Name='Seu Jorge' OR Name='The Beatles';
```

What albums did the artist with ArtistId of 41 make?

[Click here for solution](#)

```
SELECT * FROM albums WHERE ArtistId=41;
```

What are the tracks of the album with AlbumId of 71? Order the results from most Milliseconds to least.

[Click here for solution](#)

Table 4.9: Displaying records 1 - 10

TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer
1	For Those About To Rock (We Salute You)	1	1	1	Angus
2	Balls to the Wall	2	2	1	NA
3	Fast As a Shark	3	2	1	F. Bal
4	Restless and Wild	3	2	1	F. Bal
5	Princess of the Dawn	3	2	1	Deaffy
6	Put The Finger On You	1	1	1	Angus
7	Let's Get It Up	1	1	1	Angus
8	Inject The Venom	1	1	1	Angus
9	Snowballed	1	1	1	Angus
10	Evil Walks	1	1	1	Angus

Table 4.10: 1 records

COUNT(*)
3503

Table 4.11: 2 records

ArtistId	Name
41	Elis Regina
193	Seu Jorge

Table 4.12: 1 records

AlbumId	Title	ArtistId
71	Elis Regina-Minha História	41

Table 4.13: Displaying records 1 - 10

TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes
890	Aprendendo A Jogar	71	1	7	NA	290664	9391041
886	Saudosa Maloca	71	1	7	NA	278125	9059416
880	Dois Pra Lá, Dois Pra Cá	71	1	7	NA	263026	8684639
887	As Aparências Enganam	71	1	7	NA	247379	8014346
882	Romaria	71	1	7	NA	242834	7968525
883	Alô, Alô, Marciano	71	1	7	NA	241397	8137254
889	Maria Rosa	71	1	7	NA	232803	7592504
877	O Bêbado e a Equilibrista	71	1	7	NA	223059	7306143
884	Me Deixas Louca	71	1	7	NA	214831	6888030
878	O Mestre-Sala dos Mares	71	1	7	NA	186226	6180414

Table 4.14: Displaying records 1 - 10

Seconds	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds
290.664	890	Aprendendo A Jogar	71	1	7	NA	290664
278.125	886	Saudosa Maloca	71	1	7	NA	278125
263.026	880	Dois Pra Lá, Dois Pra Cá	71	1	7	NA	263026
247.379	887	As Aparências Enganam	71	1	7	NA	247379
242.834	882	Romaria	71	1	7	NA	242834
241.397	883	Alô, Alô, Marciano	71	1	7	NA	241397
232.803	889	Maria Rosa	71	1	7	NA	232803
223.059	877	O Bêbado e a Equilibrista	71	1	7	NA	223059
214.831	884	Me Deixas Louca	71	1	7	NA	214831
186.226	878	O Mestre-Sala dos Mares	71	1	7	NA	186226

```
SELECT * FROM tracks WHERE AlbumId=71 ORDER BY Milliseconds DESC;
```

What are the tracks of the album with AlbumId of 71? Order the results from longest to shortest and convert Milliseconds to seconds. Use aliasing to name the calculated field Seconds.

[Click here for solution](#)

```
SELECT Milliseconds/1000.0 AS Seconds, * FROM tracks WHERE AlbumId=71 ORDER BY Seconds DESC;
```

What are the tracks that are at least 250 seconds long?

[Click here for solution](#)

Table 4.15: Displaying records 1 - 10

Seconds	TrackId	Name	AlbumId	MediaTypeId	GenreId
343.719	1	For Those About To Rock (We Salute You)	1	1	
342.562	2	Balls to the Wall	2	2	
252.051	4	Restless and Wild	3	2	
375.418	5	Princess of the Dawn	3	2	
263.497	10	Evil Walks	1	1	
263.288	12	Breaking The Rules	1	1	
270.863	14	Spellbound	1	1	
331.180	15	Go Down	4	1	
366.654	17	Let There Be Rock	4	1	
267.728	18	Bad Boy Boogie	4	1	

Table 4.16: Displaying records 1 - 10

Seconds	TrackId	Name	AlbumId
250.017	1992	Lithium	
250.031	3421	Nimrod (Adagio) from Variations On an Original Theme, Op. 36 "Enigma"	
250.070	2090	Romance Ideal	
250.122	2451	Ela Desapareceu	
250.226	2184	Thumbing My Way	
250.253	2728	Pulse	
250.357	974	Edge Of The World	
250.462	1530	Sem Sentido	
250.565	3371	Wooden Jesus	
250.697	2504	Real Love	

```
SELECT Milliseconds/1000.0 AS Seconds, * FROM tracks WHERE Seconds >= 250;
```

What are the tracks that are between 250 and 300 seconds long?

[Click here for solution](#)

```
SELECT Milliseconds/1000.0 AS Seconds, * FROM tracks WHERE Seconds BETWEEN 250 AND 300;
```

What is the GenreId of the genre with name Pop?

[Click here for solution](#)

Table 4.17: 1 records

GenreId
9

Table 4.18: 1 records

avg
229.0341

```
SELECT GenreId FROM genres WHERE Name='Pop';
```

What is the average length (in seconds) of a track with genre “Pop”?

[Click here for solution](#)

```
SELECT AVG(Milliseconds/1000.0) AS avg FROM tracks WHERE genreId=9;
```

What is the longest Bossa Nova track (in seconds)?

[Click here for solution](#)

What is the GenreId of Bossa Nova?

```
SELECT GenreId FROM genres WHERE Name='Bossa Nova';
```

```
SELECT *, MAX(Milliseconds/1000.0) AS Seconds FROM tracks WHERE genreId=11;
```

Get the average price per hour for Bossa Nova music (genreId of 11).

[Click here for solution](#)

Table 4.19: 1 records

GenreId
11

Table 4.20: 1 records

TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Price
646	Samba Da Bênção	52	1	11	NA	409965	1349

Table 4.21: 1 records

Price per Hour
0

```
SELECT AVG(UnitPrice/Milliseconds/1000.0/3600) AS 'Price per Hour' FROM tracks WHERE g
```

Get the average time (in seconds) for tracks by genre.

[Click here for solution](#)

```
SELECT genreId, AVG(Milliseconds/1000.0) AS 'Average seconds per track' FROM tracks GR
```

We can use an INNER JOIN to get the name of each genre as well.

```
SELECT g.Name, track_time.'Average seconds per track' FROM genres AS g INNER JOIN (SEL
```

What is the average price per track for each genre?

[Click here for solution](#)

```
SELECT genreId, AVG(UnitPrice) AS 'Average seconds per track' FROM tracks GROUP BY gen
```

What is the average number of tracks per album?

[Click here for solution](#)

```
SELECT AVG(trackCount) FROM (SELECT COUNT(*) AS trackCount FROM tracks GROUP BY albumI
```

What is the average number of tracks per album per genre?

[Click here for solution](#)

Table 4.22: Displaying records 1 - 10

GenreId	Average seconds per track
1	283.9100
2	291.7554
3	309.7494
4	234.3538
5	134.6435
6	270.3598
7	232.8593
8	247.1778
9	229.0341
10	244.3709

Table 4.23: Displaying records 1 - 10

Name	Average seconds per track
Sci Fi & Fantasy	2911.7830
Science Fiction	2625.5491
Drama	2575.2838
TV Shows	2145.0410
Comedy	1585.2637
Metal	309.7494
Electronica/Dance	302.9858
Heavy Metal	297.4529
Classical	293.8676
Jazz	291.7554

Table 4.24: Displaying records 1 - 10

GenreId	Average seconds per track
1	0.99
2	0.99
3	0.99
4	0.99
5	0.99
6	0.99
7	0.99
8	0.99
9	0.99
10	0.99

Table 4.25: 1 records

AVG(trackCount)
10.0951

Table 4.26: Displaying records 1 - 10

genreId	AVG(trackCount)
1	11.41379
2	10.00000
3	10.90625
4	14.43478
5	12.00000
6	13.85714
7	14.81579
8	15.00000
9	16.00000
10	10.75000

```
SELECT genreId, AVG(trackCount) FROM (SELECT genreId, COUNT(*) AS trackCount FROM track
```

```
SELECT Name, avg_track_count.'Average Track Count' FROM genres AS g INNER JOIN (SELECT
```

The following examples use the `lahman.db` sqlite database.

```
dbListTables(lahman)
```

```
## [1] "allstarfull"      "appearances"      "awardsmanagers"
## [4] "awardsplayers"   "awardssharemanagers" "awardsshareplayers"
## [7] "batting"         "battingpost"      "collegeplaying"
## [10] "divisions"       "fielding"         "fieldingof"
## [13] "fieldingofsplit" "fieldingpost"     "halloffame"
## [16] "homegames"      "leagues"         "managers"
## [19] "managershalf"   "parks"          "people"
## [22] "pitching"       "pitchingpost"    "salaries"
## [25] "schools"        "seriespost"      "teams"
## [28] "teamsfranchises" "teamshalf"
```


Table 4.27: Displaying records 1 - 10

Name	Average Track Count
Rock	11.41379
Jazz	10.00000
Metal	10.90625
Alternative & Punk	14.43478
Rock And Roll	12.00000
Blues	13.85714
Latin	14.81579
Reggae	15.00000
Pop	16.00000
Soundtrack	10.75000

Chapter 5

R

Getting started

Variables

NA

NA stands for not available and, in general, represents a missing value or a lack of data.

How do I tell if a value is NA?

[Click here for solution](#)

```
# Test if value is NA.  
value <- NA  
is.na(value)
```

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

```
# Does is.nan return TRUE for NA?  
is.nan(value)
```

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

NaN

NaN stands for not a number and, in general, is used for arithmetic purposes, for example, the result of 0/0.

How do I tell if a value is NaN?

[Click here for solution](#)

```
# Test if a value is NaN.
value <- NaN
is.nan(value)
```

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

```
value <- 0/0
is.nan(value)
```

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

```
# Does is.na return TRUE for NaN?
is.na(value)
```

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

Dates

`Date` is a class which allows you to perform special operations like subtraction, where the number of days between dates are returned. Or addition, where you can add 30 to a `Date` and a `Date` is returned where the value is 30 days in the future.

You will usually need to specify the `format` argument based on the format of your date strings. For example, if you had a string 07/05/1990, the format would be: `%m/%d/%Y`. If your string was 31-12-90, the format would be `%d-%m-%y`. Replace `%d`, `%m`, `%Y`, and `%y` according to your date strings. A full list of formats can be found [here](#).

How do I convert a string “07/05/1990” to a Date?

[Click here for solution](#)

```
my_string <- "07/05/1990"
my_date <- as.Date(my_string, format="%m/%d/%Y")
my_date
```

```
## [1] "1990-07-05"
```

How do I convert a string “31-12-1990” to a Date?

[Click here for solution](#)

```
my_string <- "31-12-1990"
my_date <- as.Date(my_string, format="%d-%m-%Y")
my_date
```

```
## [1] "1990-12-31"
```

How do I convert a string “31-12-1990” to a Date?

[Click here for solution](#)

```
my_string <- "31-12-1990"
my_date <- as.Date(my_string, format="%d-%m-%Y")
my_date
```

```
## [1] "1990-12-31"
```

Factors

A **factor** is R’s way of representing a categorical variable. Each **factor** is essentially a numeric value with an associated name. They are a useful when a vector has only a few different values it could be, like “Male” and “Female” or “A”, “B”, or “C”.

How do I test whether or not a value is a factor?

[Click here for solution](#)

```
test_factor <- factor("Male")
is.factor(test_factor)
```

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

```
test_factor_vec <- factor(c("Male", "Female", "Female"))
is.factor(test_factor_vec)
```

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

How do I convert a vector of strings to a vector of factors?

[Click here for solution](#)

```
vec <- c("Male", "Female", "Female")
vec <- factor(c("Male", "Female", "Female"))
```

How do I get the unique values a factor could hold, also known as *levels*?

[Click here for solution](#)

```
vec <- factor(c("Male", "Female", "Female"))
levels(vec)
```

```
## [1] "Female" "Male"
```

How can I rename the levels of a vector of factors?

[Click here for solution](#)

```
vec <- factor(c("Male", "Female", "Female"))
levels(vec)
```

```
## [1] "Female" "Male"
```

```
levels(vec) <- c("F", "M")
vec
```

```
## [1] M F F
## Levels: F M
```

```
# Be careful! Order matters, this is wrong:
vec <- factor(c("Male", "Female", "Female"))
levels(vec)
```

```
## [1] "Female" "Male"
```

```
levels(vec) <- c("M", "F")
vec
```

```
## [1] F M M
## Levels: M F
```

Logical operators

Logical operators are symbols that can be used within R to compare values or vectors of values.

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to
!x	negation, not x
x y	x OR y
x&y	x AND y

Examples

What are the values in a vector, `vec` that are greater than 5?

[Click here for solution](#)

```
vec <- 1:10
vec > 5
```

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

What are the values in a vector, `vec` that are greater than or equal to 5?

[Click here for solution](#)

```
vec <- 1:10  
vec >= 5
```

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

What are the values in a vector, `vec` that are less than 5?

[Click here for solution](#)

```
vec <- 1:10  
vec < 5
```

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

What are the values in a vector, `vec` that are less than or equal to 5?

[Click here for solution](#)

```
vec <- 1:10  
vec <= 5
```

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

What are the values in a vector that are greater than 7 OR less than or equal to 2?

[Click here for solution](#)

```
vec <- 1:10  
vec > 7 | vec <= 2
```

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

What are the values in a vector that are greater than 3 AND less than 6?

[Click here for solution](#)


```
vec <- 1:10  
vec > 3 & vec < 6
```

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

How do I get the values in list1 that are in list2?

[Click here for solution](#)

```
list1 <- c("this", "is", "a", "test")  
list2 <- c("this", "a", "exam")  
list1[list1 %in% list2]
```

```
## [1] "this" "a"
```

How do I get the values in list1 that are not in list2?

[Click here for solution](#)

```
list1 <- c("this", "is", "a", "test")  
list2 <- c("this", "a", "exam")  
list1[!(list1 %in% list2)]
```

```
## [1] "is" "test"
```

How can I get the number of values in a vector that are greater than 5?

[Click here for solution](#)

```
vec <- 1:10  
sum(vec>5)
```

```
## [1] 5
```

```
# Note, you do not need to do:  
length(vec[vec>5])
```

```
## [1] 5
```

```
# because TRUE==1 and FALSE==0 in R
TRUE==1
```

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

```
FALSE==0
```

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

Resources

Operators Summary

A quick list of the various operators with a few simple examples.

Lists & Vectors

A vector contains values that are all the same type. The following are some examples of vectors:

```
# A logical vector
lvec <- c(F, T, TRUE, FALSE)
class(lvec)
```

```
## [1] "logical"
```

```
# A numeric vector
nvec <- c(1,2,3,4)
class(nvec)
```

```
## [1] "numeric"
```

```
# A character vector
cvec <- c("this", "is", "a", "test")
class(cvec)
```

```
## [1] "character"
```

As soon as you try to mix and match types, elements are coerced to the simplest type required to represent all the data.

The order of representation is:

logical, numeric, character, list

For example:

```
class(c(F, 1, 2))
```

```
## [1] "numeric"
```

```
class(c(F, 1, 2, "ok"))
```

```
## [1] "character"
```

```
class(c(F, 1, 2, "ok", list(1, 2, "ok")))
```

```
## [1] "list"
```

Lists are vectors that can contain any class of data. For example:

```
list(TRUE, 1, 2, "OK", c(1,2,3))
```

```
## [[1]]  
## [1] TRUE  
##  
## [[2]]  
## [1] 1  
##  
## [[3]]  
## [1] 2  
##  
## [[4]]  
## [1] "OK"  
##  
## [[5]]  
## [1] 1 2 3
```

With lists, there are 3 ways you can index.

```
my_list <- list(TRUE, 1, 2, "OK", c(1,2,3), list("OK", 1,2, F))
```

```
# The first way is with single square brackets [].
# This will always return a list, even if the content
# only has 1 component.
class(my_list[1:2])
```

```
## [1] "list"
```

```
class(my_list[3])
```

```
## [1] "list"
```

```
# The second way is with double brackets [[]].
# This will return the content itself. If the
# content is something other than a list it will
# return the value itself.
class(my_list[[1]])
```

```
## [1] "logical"
```

```
class(my_list[[3]])
```

```
## [1] "numeric"
```

```
# Of course, if the value is a list itself, it will
# remain a list.
class(my_list[[6]])
```

```
## [1] "list"
```

```
# The third way is using $ to extract a single, named variable.
# We need to add names first! $ is like the double bracket,
# in that it will return the simplest form.
my_list <- list(first=TRUE, second=1, third=2, fourth="OK", embedded_vector=c(1,2,3), c(1,2,3))
my_list$first
```

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

```
my_list$embedded_list
```

```
## [[1]]  
## [1] "OK"  
##  
## [[2]]  
## [1] 1  
##  
## [[3]]  
## [1] 2  
##  
## [[4]]  
## [1] FALSE
```

Basic R functions

dim

`dim` returns the dimensions of a matrix or data.frame. The first value is the rows, the second is columns.

Examples

How many dimensions does the data.frame `dat` have?

[Click here for solution](#)

```
dat <- data.frame("col1"=c(1,2,3), "col2"=c("a", "b", "c"))  
dim(dat) # 3 rows and 2 columns
```

```
## [1] 3 2
```

length

`length` allows you to get or set the length of an object in R (for which a method has been defined).

How do I get how many values are in a vector?

[Click here for solution](#)

```
# Create a vector of length 5
my_vector <- c(1,2,3,4,5)

# Calculate the length of my_vector
length(my_vector)
```

```
## [1] 5
```

grep, grepl, etc.

grep allows you to use regular expressions to search for a pattern in a string or character vector, and returns the index where there is a match.

grepl performs the same operation but rather than returning indices, returns a vector of logical TRUE or FALSE values.

Examples

Given a character vector, return the index of any words ending in “s”.

[Click here for solution](#)

```
grep("*.s$", c("waffle", "waffles", "pancake", "pancakes"))
```

```
## [1] 2 4
```

Given a character vector, return a vector of the same length where each element is TRUE if there was a match for any word ending in “s”, and FALSE otherwise.

[Click here for solution](#)

```
grepl("*.s$", c("waffle", "waffles", "pancake", "pancakes"))
```

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

mean

mean is a function that calculates the average of a vector of values.

How do I get the average of a vector of values?

[Click here for solution](#)

```
mean(c(1,2,3,4))
```

```
## [1] 2.5
```

How do I get the average of a vector of values when some of the values are: NA, NaN?

[Click here for solution](#)

Many R functions have the `na.rm` argument available. This argument is “a logical value indicating whether NA values should be stripped before the computation proceeds.”

```
mean(c(1,2,3,NaN), na.rm=T)
```

```
## [1] 2
```

```
mean(c(1,2,3,NA), na.rm=T)
```

```
## [1] 2
```

```
mean(c(1,2,NA,NaN,4), na.rm=T)
```

```
## [1] 2.333333
```

var

`var` is a function that calculate the variance of a vector of values.

How do I get the variance of a vector of values?

[Click here for solution](#)

```
var(c(1,2,3,4))
```

```
## [1] 1.666667
```

How do I get the variance of a vector of values when some of the values are: NA, NaN?

[Click here for solution](#)

```
var(c(1,2,3,NaN), na.rm=T)
```

```
## [1] 1
```

```
var(c(1,2,3,NA), na.rm=T)
```

```
## [1] 1
```

```
var(c(1,2,NA,NaN,4), na.rm=T)
```

```
## [1] 2.333333
```

How do I get the standard deviation of a vector of values?

[Click here for solution](#)

The standard deviation is equal to the square root of the variance.

```
sqrt(var(c(1,2,3,NaN), na.rm=T))
```

```
## [1] 1
```

```
sqrt(var(c(1,2,3,NA), na.rm=T))
```

```
## [1] 1
```

```
sqrt(var(c(1,2,NA,NaN,4), na.rm=T))
```

```
## [1] 1.527525
```

unique

unique “returns a vector, data frame, or array like x but with duplicate elements/rows removed.

Given a vector of values, how do I return a vector of values with all duplicates removed?

[Click here for solution](#)

```
vec <- c(1, 2, 3, 3, 3, 4, 5, 5, 6)
unique(vec)
```

```
## [1] 1 2 3 4 5 6
```

paste and paste0

`paste` is a useful function to “concatenate vectors after converting to character.”

`paste0` is a shorthand function where the `sep` argument is “”.

How do I concatenate two vectors, element-wise, with a comma in between values from each vector?

[Click here for solution](#)

```
vector1 <- c("one", "three", "five")
vector2 <- c("two", "four", "six")
paste(vector1, vector2, sep=",")
```

```
## [1] "one,two"      "three,four"   "five,six"
```

How do I paste together two strings?

[Click here for solution](#)

```
paste0("abra", "kadabra")
```

```
## [1] "abrakadabra"
```

How do I paste together three strings?

[Click here for solution](#)

```
paste0("abra", "kadabra", "alakazam")
```

```
## [1] "abrakadabraalakazam"
```

str

str stands for *structure*. **str** gives you a glimpse at the variable of interest.

How do I get the number of columns or features in a data.frame?

[Click here for solution](#)

As you can see, there are 9 rows or obs. (short for observations), and 29 variables (which can be referred to as columns or features).

```
str(df)
```

str_extract and str_extract_all

str_extract and **str_extract_all** are useful functions from the **stringr** package. You can install the package by running:

```
install.packages("stringr")
```

str_extract extracts the text which matches the provided regular expression or pattern. Note that this differs from **grep** in a major way. **grep** simply returns the index in which a pattern match was found. **str_extract** returns the actual matching text. Note that **grep** typically returns the entire line where a match was found. **str_extract** returns only the part of the line or text that matches the pattern.

For example:

```
text <- c("cat", "mat", "spat", "spatula", "gnat")
# All 5 "lines" of text were a match.
grep(".*at", text)
```

```
## [1] 1 2 3 4 5
```

```
text <- c("cat", "mat", "spat", "spatula", "gnat")
stringr::str_extract(text, ".*at")
```

```
## [1] "cat" "mat" "spat" "spat" "gnat"
```

As you can see, although all 5 words match our pattern and would be returned by `grep`, `str_extract` only returns the actual text that matches the pattern. In this case “spatula” is *not* a “full” match – the pattern “`.*at`” only captures the “spat” part of “spatula”. In order to capture the rest of the word you would need to add something like “`.*`” to the end of the pattern:

```
text <- c("cat", "mat", "spat", "spatula", "gnat")
stringr::str_extract(text, ".*at.*")

## [1] "cat"      "mat"      "spat"     "spatula" "gnat"
```

One final note is that you must double-escape certain characters in patterns because R treats backslashes as escape values for character constants (stack-overflow). For example, to write `\(` we must first escape the `\`, so we write `\\(`. This is true for many character which would normally only be preceded by a single `\`.

Examples

How can I extract the text between parenthesis in a vector of texts?

[Click here for solution](#)

```
text <- c("this is easy for (you)", "there (are) challenging ones", "text is (really awesome) (ok?)")
# Search for a literal "(", followed by any amount of any text other than more parenthesis ([^()]*\\)
stringr::str_extract(text, "\\([^()]*\\)")

## [1] "(you)"      "(are)"      "(really awesome)"
```

To get *all* matches, not just the first match:

```
text <- c("this is easy for (you)", "there (are) challenging ones", "text is (really awesome) more (ok?)")
# Search for a literal "(", followed by any amount of any text (.*), followed by a literal ")".
stringr::str_extract_all(text, "\\(.*\\)")

## [[1]]
## [1] "(you)"
##
## [[2]]
## [1] "(are)"
##
## [[3]]
## [1] "(really awesome)" "(ok?)"
```

Data.frames

How do I sample n rows randomly from a data.frame called `df`?

[Click here for solution](#)

```
df[sample(nrow(df), n),]
```

Alternatively you could use the `sample_n` function from the package `dplyr`:

```
sample_n(df, n)
```

Reading & Writing data

Examples

How do I read a csv file called `grades.csv` into a data.frame?

[Click here for solution](#)

```
dat <- read.csv("./grades.csv")
head(dat)
```

```
##   grade   year
## 1   100  junior
## 2    99 sophom
## 3    75 sophom
## 4    74 sophom
## 5    44  senior
## 6    69  junior
```

How do I read a csv file called `grades2.csv` where instead of being comma-separated, it is semi-colon-separated, into a data.frame?

[Click here for solution](#)

```
dat <- read.csv("./grades_semi.csv", sep=";")
head(dat)
```

```
##   grade   year
## 1   100  junior
```

```
## 2    99 sophomore
## 3    75 sophomore
## 4    74 sophomore
## 5    44   senior
## 6    69   junior
```

How do I prevent R from reading in strings as factors when using a function like `read.csv`?

[Click here for solution](#) In R 4.0+, strings are not read in as factors, so you do not need to do anything special. For R < 4.0, use `stringsAsFactors`.

```
dat <- read.csv("./grades.csv", stringsAsFactors=F)
head(dat)
```

```
##   grade    year
## 1   100   junior
## 2    99 sophomore
## 3    75 sophomore
## 4    74 sophomore
## 5    44   senior
## 6    69   junior
```

How do I specify the type of 1 or more columns when reading in a csv file?

[Click here for solution](#)

```
dat <- read.csv("./grades.csv", colClasses=c("grade"="character", "year"="factor"))
str(dat)
```

```
## 'data.frame':   10 obs. of  2 variables:
## $ grade: chr  "100" "99" "75" "74" ...
## $ year : Factor w/ 4 levels "freshman","junior",...: 2 4 4 4 3 2 2 3 1 2
```

Given a list of csv files with the same columns, how can I read them in and combine them into a single dataframe?

[Click here for solution](#)

```
# We want to read in grades.csv, grades2.csv, and grades3.csv
# into a single dataframe.

list_of_files <- c("grades.csv", "grades2.csv", "grades3.csv")

results <- data.frame()
for (file in list_of_files) {
  dat <- read.csv(file)
  results <- rbind(results, dat)
}
dim(results)
```

```
## [1] 32 2
```

How do I create a data.frame with comma-separated data that I've copied onto my clipboard?

[Click here for solution](#)

```
# For mac
dat <- read.delim(pipe("pbpaste"),header=F,sep=",")

# For windows
dat <- read.table("clipboard",header=F,sep=",")
```

Control flow

If/else statements

If, else if, and else statements are methods for controlling whether or not an operation is performed based on the result of some expression.

How do I print “Success!” if my expression evaluates to TRUE, and “Failure!” otherwise?

[Click here for solution](#)

```
# Randomly assign either TRUE or FALSE to t_or_f.
t_or_f <- sample(c(TRUE,FALSE),1)

if (t_or_f == TRUE) {
```

```

    # If t_or_f is TRUE, print success
    print("Success!")
  } else {
    # Otherwise, print failure
    print("Failure!")
  }

```

```
## [1] "Success!"
```

```

# You don't need to put the full expression.
# This is the same thing because t_or_f
# is already TRUE or FALSE.
# TRUE == TRUE evaluates to TRUE and
# FALSE == TRUE evaluates to FALSE.
if (t_or_f) {
  # If t_or_f is TRUE, print success
  print("Success!")
} else {
  # Otherwise, print failure
  print("Failure!")
}

```

```
## [1] "Success!"
```

How do I print “Success!” if my expression evaluates to TRUE, “Failure!” if my expression evaluates to FALSE, and “Huh?” otherwise?

[Click here for solution](#)

```

# Randomly assign either TRUE or FALSE to t_or_f.
t_or_f <- sample(c(TRUE,FALSE, "Something else"),1)

if (t_or_f == TRUE) {
  # If t_or_f is TRUE, print success
  print("Success!")
} else if (t_or_f == FALSE) {
  # If t_or_f is FALSE, print failure
  print("Failure!")
} else {
  # Otherwise print huh
  print("Huh?")
}

```

```
## [1] "Success!"
```

```

# In this case you need the full expression because
# "Something else" does not evaluate to TRUE or FALSE
# which will cause an error as the if and else if
# statements expect a result of TRUE or FALSE.
if (t_or_f == TRUE) {
  # If t_or_f is TRUE, print success
  print("Success!")
} else if (t_or_f == FALSE) {
  # If t_or_f is FALSE, print failure
  print("Failure!")
} else {
  # Otherwise print huh
  print("Huh?")
}

```

```
## [1] "Success!"
```

For loops

For loops allow us to execute similar code over and over again until we've looped through all of the elements. They are useful for performing the same operation to an entire vector of input, for example.

Using the suite of apply functions is more common in R. It is often said that the apply suite of function are much faster than for loops in R. While this used to be the case, this is no longer true.

Examples

How do I loop through every value in a vector and print the value?

[Click here for solution](#)

```

for (i in 1:10) {
  # In the first iteration of the loop,
  # i will be 1. The next, i will be 2.
  # Etc.
  print(i)
}

```

```

## [1] 1
## [1] 2
## [1] 3
## [1] 4

```



```
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

How do I break out of a loop before it finishes?

[Click here for solution](#)

```
for (i in 1:10) {
  if (i==7) {
    # When i==7, we will exit the loop.
    break
  }
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
```

How do I loop through a vector of names?

[Click here for solution](#)

```
friends <- c("Phoebe", "Ross", "Rachel", "Chandler", "Joey", "Monica")
my_string <- "So no one told you life was gonna be this way, "
for (friend in friends) {
  print(paste0(my_string, friend, "!"))
}
```

```
## [1] "So no one told you life was gonna be this way, Phoebe!"
## [1] "So no one told you life was gonna be this way, Ross!"
## [1] "So no one told you life was gonna be this way, Rachel!"
## [1] "So no one told you life was gonna be this way, Chandler!"
## [1] "So no one told you life was gonna be this way, Joey!"
## [1] "So no one told you life was gonna be this way, Monica!"
```

How do I skip a loop if some expression evaluates to TRUE?

[Click here for solution](#)

```
friends <- c("Phoebe", "Ross", "Mike", "Rachel", "Chandler", "Joey", "Monica")
my_string <- "So no one told you life was gonna be this way, "
for (friend in friends) {
  if (friend == "Mike") {
    # next, skips over the rest of the code for this loop
    # and continues to the next element
    next
  }
  print(paste0(my_string, friend, "!"))
}
```

```
## [1] "So no one told you life was gonna be this way, Phoebe!"
## [1] "So no one told you life was gonna be this way, Ross!"
## [1] "So no one told you life was gonna be this way, Rachel!"
## [1] "So no one told you life was gonna be this way, Chandler!"
## [1] "So no one told you life was gonna be this way, Joey!"
## [1] "So no one told you life was gonna be this way, Monica!"
```

Apply functions

`apply`

`lapply`

`sapply`

`tapply`

`tapply` is described in the documentation as a way to “apply a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.” This is not a very useful description.

An alternative way to think about `tapply`, is as a function that allows you to calculate or apply function to `data1` when `data1` is grouped by `data2`.

```
tapply(data1, data2, function)
```

A concrete example would be getting the *mean* (function) *grade* (data1) when *grade* (data1) is grouped by *year* (data2):

```
grades
```

```
##   grade   year
## 1   100   junior
## 2    99 sophomore
## 3    75 sophomore
## 4    74 sophomore
## 5    44   senior
## 6    69   junior
## 7    88   junior
## 8    99   senior
## 9    90 freshman
## 10   92   junior
```

```
tapply(grades$grade, grades$year, mean)
```

```
##  freshman   junior   senior sophomore
## 90.00000  87.25000  71.50000  82.66667
```

If your function (in this case *mean*), requires extra arguments, you can pass those by name to `tapply`. This is what the `...` argument in `tapply` is for. For example, if we want our *mean* function to remove na's prior to calculating a mean we could do the following:

```
tapply(grades$grade, grades$year, mean, na.rm=T)
```

```
##  freshman   junior   senior sophomore
## 90.00000  87.25000  71.50000  82.66667
```

Writing functions

Plotting

`barplot`

`ggplot`

`ggmap`

`ggmap` is an excellent package that provides a suite of functions that, among other things, allows you to map spatial data on top of static maps.

Getting started

To install `ggmap`, simply run `install.packages("ggmap")`. To load the library, run `library(ggmap)`. When first using this package, you may notice you need an API key to get access to certain functionality. Follow the directions here to get an API key. It should look something like: `mQkzTpiaLYjPqXQBotesgif3EfGL2dbrNV0rogg`.

Once you've acquired the API key, you have two options:

1. Register `ggmap` with Google for the current session:

```
library(ggmap)
register_google(key="mQkzTpiaLYjPqXQBotesgif3EfGL2dbrNV0rogg")
```

2. Register `ggmap` with Google, persistently through sessions:

```
library(ggmap)
register_google(key="mQkzTpiaLYjPqXQBotesgif3EfGL2dbrNV0rogg", write=TRUE)
```

Note that if you choose option (2), your API key will be saved within your `~/.Renviron`.

Examples

How do I get a map of West Lafayette?

[Click here for solution](#)

```
map <- get_map(location="West Lafayette")
ggmap(map)
```

How do I zoom in and out on a map of West Lafayette?

[Click here for solution](#)

```
# zoom way out
map <- get_map(location="West Lafayette", zoom=1)
ggmap(map)

# zoom in
map <- get_map(location="West Lafayette", zoom=12)
ggmap(map)
```

How do I add Latitude and Longitude points to a map of Purdue University?

Click here for solution

```
points_to_add <- data.frame(latitude=c(40.433663, 40.432104, 40.428486), longitude=c(-86.916584,
map <- get_map(location="Purdue University", zoom=14)
ggmap(map) + geom_point(data = points_to_add, aes(x = longitude, y = latitude))
```

RMarkdown

To install RMarkdown simply run the following:

```
install.packages("rmarkdown")
```

Projects in The Data Mine are all written in RMarkdown. You can download the RMarkdown file by clicking on the link at the top of each project page. Each file should end in the “.Rmd” which is the file extension commonly associated with RMarkdown files.

You can find an exemplary RMarkdown file here:

<https://raw.githubusercontent.com/TheDataMine/the-examples-book/master/files/rmarkdown.Rmd>

If you open this file in RStudio, and click on the “Knit” button in the upper left hand corner of IDE, you will get the resulting HTML file. Open this file in the web browser of your choice and compare and contrast the syntax in the `rmarkdown.Rmd` file and resulting output. Play around with the file, make modifications, and re-knit to gain a better understanding of the syntax. Note that similar input/output examples are shown in the RMarkdown Cheatsheet.

Code chunks

Code chunks are sections within an RMarkdown file where you can write, display, and optionally evaluate code from a variety of languages:

## [1] "awk"	"bash"	"coffee"	"gawk"	"groovy"
## [6] "haskell"	"lein"	"mysql"	"node"	"octave"
## [11] "perl"	"psql"	"Rscript"	"ruby"	"sas"
## [16] "scala"	"sed"	"sh"	"stata"	"zsh"
## [21] "highlight"	"Rcpp"	"tikz"	"dot"	"c"
## [26] "cc"	"fortran"	"fortran95"	"asy"	"cat"
## [31] "asis"	"stan"	"block"	"block2"	"js"

```
## [36] "css"          "sql"          "go"           "python"       "julia"
## [41] "sass"         "scss"         "theorem"      "lemma"        "corollary"
## [46] "proposition"  "conjecture"   "definition"   "example"      "exercise"
## [51] "proof"        "remark"       "solution"
```

The syntax is simple:

```
‘‘‘{language, options...}
code here...
‘‘‘
```

For example:

```
‘‘‘{r, echo=TRUE}
my_variable <- c(1,2,3)
my_variable
‘‘‘
```

Which will render like:

```
my_variable <- c(1,2,3)
my_variable
```

```
## [1] 1 2 3
```

You can find a list of chunk options [here](#).

How do I run a code chunk but not display the code above the results?

[Click here for solution](#)

```
‘‘‘{r, echo=FALSE}
my_variable <- c(1,2,3)
my_variable
‘‘‘
```

How do I include a code chunk without evaluating the code itself?

[Click here for solution](#)

```
```{r, eval=FALSE}  
my_variable <- c(1,2,3)
my_variable
```
```

How do I prevent warning messages from being displayed?

[Click here for solution](#)

```
```{r, warning=FALSE}  
my_variable <- c(1,2,3)
my_variable
```
```

How do I prevent error messages from being displayed?

[Click here for solution](#)

```
```{r, error=FALSE}  
my_variable <- c(1,2,3)
my_variable
```
```

How do I run a code chunk, but not include the chunk in the final output?

[Click here for solution](#)

```
```{r, include=FALSE}  
my_variable <- c(1,2,3)
my_variable
```
```

How do I render a figure from a chunk?

[Click here for solution](#)

```
```{r}  
my_variable <- c(1,2,3)
plot(my_variable)
```
```

How do I create a set of slides using RMarkdown?

[Click here for solution](#) Please see the example Rmarkdown file [here](#).

You can change the slide format by changing the yaml header to any of: `ioslides_presentation`, `slidy_presentation`, or `beamer_presentation`.

By default all first and second level headers (`#` and `##`, respectively) will create a new slide. To manually create a new slide, you can use `***`.

Resources

RMarkdown Cheatsheet

An excellent quick reference for RMarkdown syntax.

RMarkdown Reference

A thorough reference manual showing markdown input and expected output. Gives descriptions of the various chunk options, as well as output options.

RStudio RMarkdown Lessons

A set of lessons detailing the ins and outs of RMarkdown.

Markdown Tutorial

RMarkdown uses Markdown syntax for its text. This is a good, interactive tutorial to learn the basics of Markdown. This tutorial is available in multiple languages.

RMarkdown Gallery

This gallery highlights a variety of reproducible and interactive RMarkdown documents. An excellent resource to see the power of RMarkdown.

RMarkdown Chapter

This is a chapter from Hadley Wickham's excellent *R for Data Science* book that details important parts of RMarkdown.

RMarkdown in RStudio

This is a nice article that introduces RMarkdown, and guides the user through creating their own interactive document using RMarkdown in RStudio.

Reproducible Research

This is another good resource that introduces RMarkdown. Plenty of helpful pictures and screenshots.

Tidyverse

lubridate

`lubridate` is a fantastic package that makes the typical tasks one would perform on dates, that much easier.

How do I convert a string “07/05/1990” to a Date?

[Click here for solution](#)

```
library(lubridate)
dat <- "07/05/1990"
dat <- mdy(dat)
class(dat)
```

```
## [1] "Date"
```

How do I convert a string “31-12-1990” to a Date?

[Click here for solution](#)

```
my_string <- "31-12-1990"
dat <- dmy(my_string)
dat
```

```
## [1] "1990-12-31"
```

```
class(dat)
```

```
## [1] "Date"
```

How do I convert a string “31121990” to a Date?

[Click here for solution](#)

```
my_string <- "31121990"
my_date <- dmy(my_string)
my_date
```

```
## [1] "1990-12-31"
```

```
class(my_date)
```

```
## [1] "Date"
```

How do I extract the day, week, month, quarter, and year from a Date?

[Click here for solution](#)

```
my_date <- dmy("31121990")  
day(my_date)
```

```
## [1] 31
```

```
week(my_date)
```

```
## [1] 53
```

```
month(my_date)
```

```
## [1] 12
```

```
quarter(my_date)
```

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

```
year(my_date)
```

```
## [1] 1990
```

Resources

Lubridate Cheatsheet

A comprehensive cheatsheet on `lubridate`. Excellent resource to immediately begin using `lubridate`.

DATA.TABLE

91

data.table

SQL in R

Scraping

shiny

Rendering images

Chapter 6

Python

Getting started

Python on Scholar

Each year we provide students with a working Python kernel that students are able to select and use from within <https://notebook.scholar.rcac.purdue.edu/> as well as within an Rmarkdown document in <https://rstudio.scholar.rcac.purdue.edu/>. We ask that students use this kernel when completing all Python-related questions for the course. This ensures version consistency for Python and all packages that students will use during the academic year. In addition, this enables staff to quickly modify the Python environment for all students should the need arise.

Let's configure this so every time you access <https://notebook.scholar.rcac.purdue.edu/> or <https://rstudio.scholar.rcac.purdue.edu/>, you will have access to the proper kernel, and the default version of python is correct. Navigate to <https://rstudio.scholar.rcac.purdue.edu/>, and login using your Purdue credentials. In the menu, click **Tools > Shell...**

You should be presented with a shell towards the bottom left. Click within the shell, and type the following followed by pressing Enter or Return:

```
/class/datamine/apps/runme
```

After executing the script, in the menu, click **Session > Restart R**.

In order to run Python within <https://rstudio.scholar.rcac.purdue.edu/>, log in to <https://rstudio.scholar.rcac.purdue.edu/> and run the following in the Console or in an R code chunk:

```
datamine_py()  
install.packages("reticulate")
```

The function `datamine_py` “activates” the Python environment we have setup for the course. Any time you want to use our environment, simply run the R function at the beginning of any R Session, *prior* to running anything Python code chunks.

To test if the Python environment is working within <https://rstudio.scholar.rcac.purdue.edu/>, run the following in a Python code chunk:

```
import sys  
print(sys.executable)
```

The python executable should be located in the appropriate folder in the following path: `/class/datamine/apps/python/`.

The `runme` script also adds a kernel to the list of kernels shown in <https://notebook.scholar.rcac.purdue.edu/>.

To test if the kernel is available and working, navigate to <https://notebook.scholar.rcac.purdue.edu/>, login, click on **New**, and select the kernel matching the current year. For example, you would select `f2020-s2021` for the 2020-2021 academic year. Once the notebook has launched, you can confirm the version of Python by running the following in a code cell:

```
import sys  
print(sys.executable)
```

The python executable should be located in the appropriate folder in the following path: `/class/datamine/apps/python/`.

If you already have a Jupyter notebook running at <https://notebook.scholar.rcac.purdue.edu/>, you may need to refresh in order for the kernel to appear as an option in **Kernel > Change Kernel**.

If you would like to use the Python environment that is put together for this class, from within a terminal on Scholar, run the following:

```
source /class/datamine/apps/python.sh
```

This will load the environment and `python` will launch our environment’s interpreter.

Lists & Tuples

Dicts

Control flow

Writing functions

Reading & Writing data

numpy

scipy

pandas

Jupyter notebooks

Writing scripts

argparse

Scraping

Plotting

matplotlib

Resources

plotly

plotnine

pygal

seaborn

bokeh

Classes

tensorflow

pytorch

Chapter 7

Tools

Docker

Tableau

GitHub

Overview

GitHub is a `git` repository hosting service. There are other, less well known repository hosting services such as: GitLab, Bitbucket, and Gitea. `git` itself is a free and open source version-control system for tracking changes in source code during software development.¹

`git`

Install

1. Follow the instructions here to install `git` onto your machine.

Configure `git`

1. Run the following commands:

¹<https://en.wikipedia.org/wiki/Git>

```
git config --global user.name "You name here"  
git config --global user.email "your_email@example.com"
```

2. Next, you need to authenticate with GitHub. Create a public/private keypair:

```
ssh-keygen -t rsa -C "your_email@example.com"
```

This creates two files:

~/.ssh/id_rsa –your private key

and

~/.ssh/id_rsa.pub –your public key

3. Copy your **public** key to your clipboard.
4. Navigate and sign in to <https://github.com>.
5. Go here, and click “New SSH key”.
6. Name the key whatever you’d like in the “Title” field. Usually, I put the name of the computer I’m using.
7. Paste the key in the “Key” field, and click “Add SSH key”.
8. At this point in time you should be good to go. Verify by running the following in your terminal:

```
ssh -T git@github.com
```

You should receive a message like:

```
Hi username! You’ve successfully authenticated, but Github does  
not provide shell access.
```

Clone a repository

If you’ve followed the directions here to configure git with SSH:

1. Open a terminal and navigate into the folder in which you’d like to clone the repository. For example, let’s say I would like to clone this book’s repository into my ~/projects folder:

```
cd ~/projects
```

2. Next, run the following command:

```
git clone git@github.com:TheDataMine/the-examples-book.git
```

3. At this point in time, you should have a new folder called `the-examples-book` inside your `~/projects` folder.

Commit changes to a repository

Creating a commit is simple:

1. Navigate into your project repository folder. For example, let's assume our repository lives: `~/projects/the-examples-book`.

```
cd ~/projects/the-examples-book
```

2. Modify the repository files as you would like, saving the changes.
3. Create your commit, with an accompanying message:

```
git commit -m "Fixed minor spelling error."
```

Fetch remote changes

1. Navigate to the local repository. For example, let's assume our repository lives: `~/projects/the-examples-book`.

```
cd ~/projects/the-examples-book
```

2. Fetch and pull the changes:

```
git fetch  
git pull
```

Push local commits to the remote origin

1. First fetch any remote changes.
2. Then run the following commands:

```
git push
```

Create a new branch

To create a new branch based off of the **master** branch do the following.

1. Checkout the master branch:

```
git checkout master
```

2. Create a new branch named **fix-spelling-errors-01** based off of the master branch and check the new **fix-spelling-errors-01** branch out:

```
git checkout -b fix-spelling-errors-01
```

Publish your branch to GitHub

If your current local branch is not present on its remote origin, git push will publish the branch to GitHub.

Create a pull request

After publishing a local branch to GitHub, in order to create a pull request, simply navigate to the following link:

https://github.com/my_organization/my_repo/pull/new/my_branch_name

Replace **my_organization** with the username or organization name. For example: **thedatamine**.

Replace **my_repo** with the name of the repository. For example: **the-examples-book**.

Replace **my_branch_name** with the name of the branch you would like to have merged into the **master** branch. For example: **fix-spelling-errors-01**.

So at the end, using our examples, you would navigate to:

<https://github.com/TheDataMine/the-examples-book/pull/new/fix-spelling-errors-01>

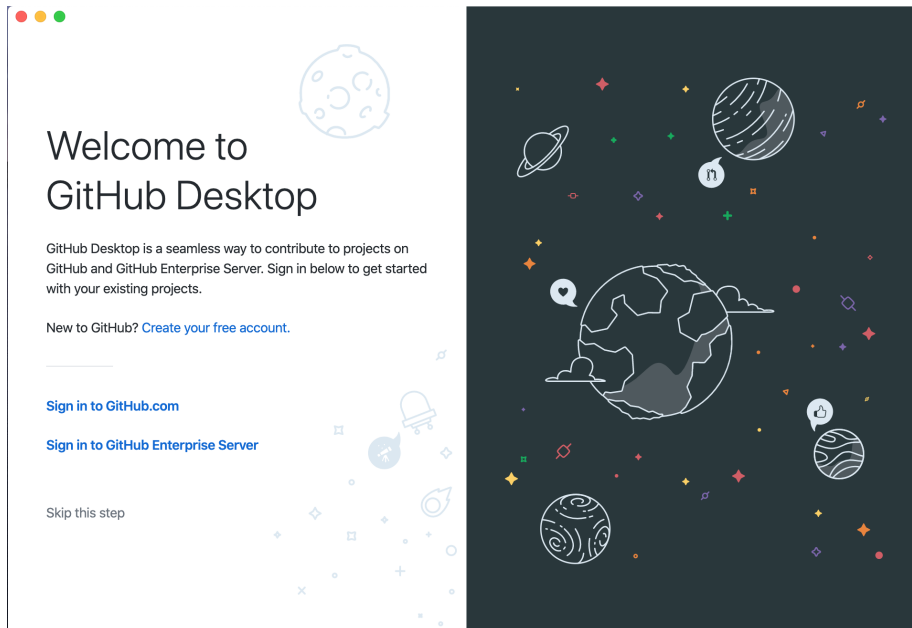
Fill out the information, and click “Create pull request”.

GitHub Desktop

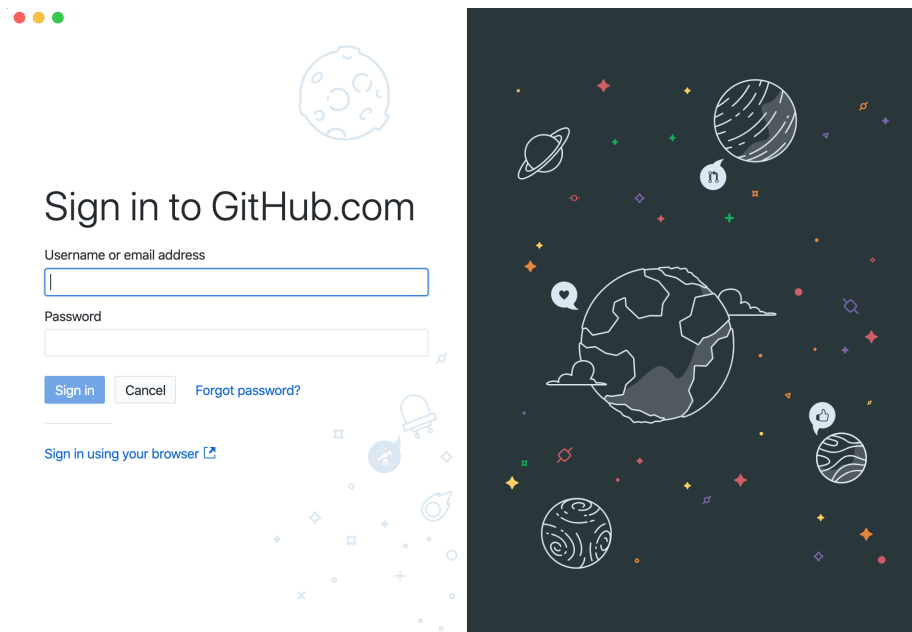
Install

1. Follow the excellent directions here to install GitHub Desktop.

2. Upon the launch of the application, you should be presented with a screen similar to this:



3. Click on "Sign in to GitHub.com."
4. Enter your GitHub credentials in the following screen:

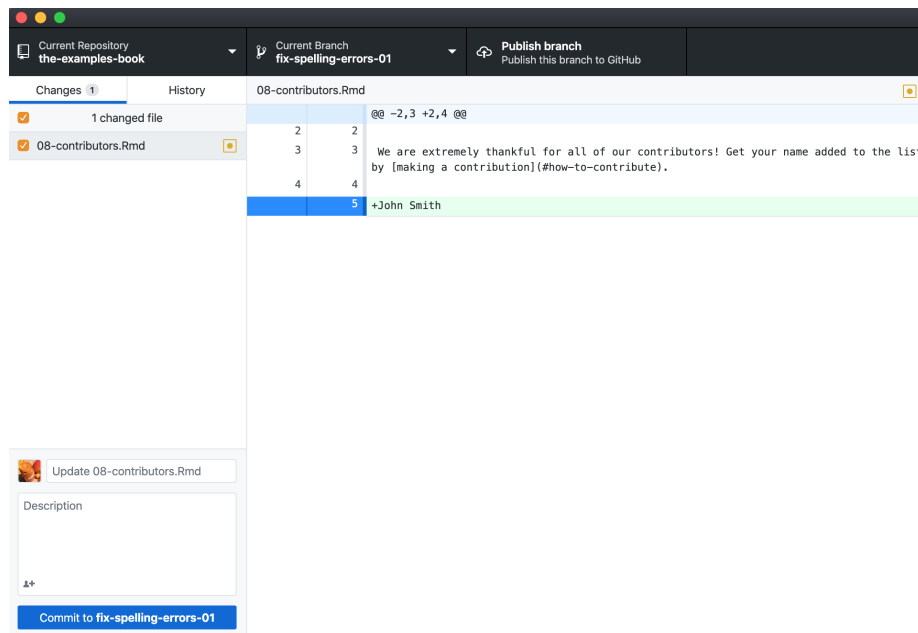


5. Continue the sign in process. You will eventually be presented with a screen

to select a repository. Congratulations! You’ve successfully installed GitHub Desktop.

Commit changes to a repository

1. First, make a change to to a file within the repository. In this example, I added a contributor named John Smith:



2. In the lower left-hand corner of the GUI, add a Commit title and description. Concise and detailed titles and descriptions are best. Click “Commit to **name-of-branch**” in this case, our branch name is **fix-spelling-errors-01**.
3. At this point in time the Commit is only local (on your machine). In order to update the remote repository (on GitHub), you’ll need to publish your branch.

If your branch is already published (present on github.com), you’ll need to push your local commits to the remote origin (which is the remote **fix-spelling-errors-01** branch in this case) by clicking on the “Push origin” button:

Push local commits to the remote origin

1. If you have commits that are ready to be pushed to the remote origin (github.com), you’ll be presented with a screen similar to this:

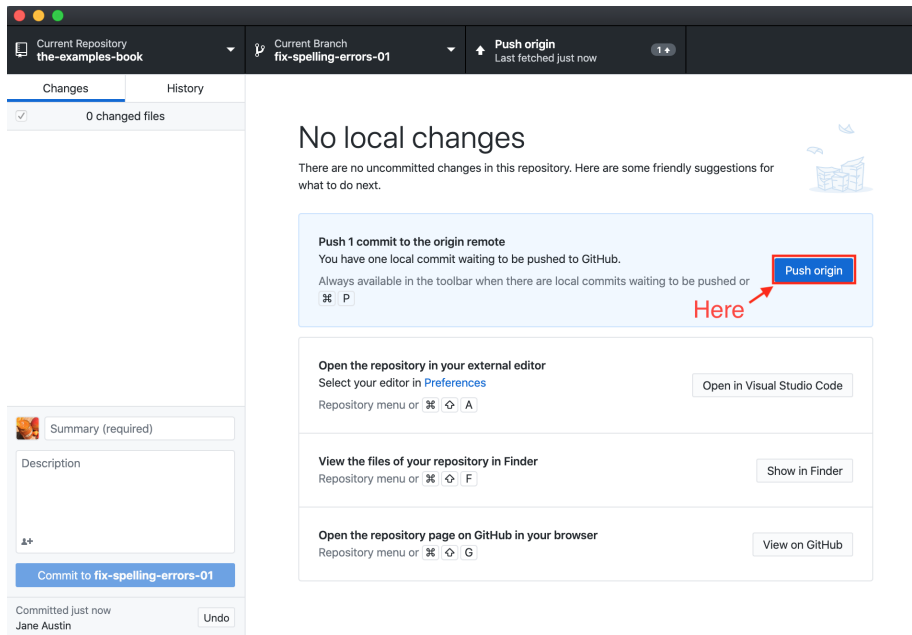


Figure 7.1:

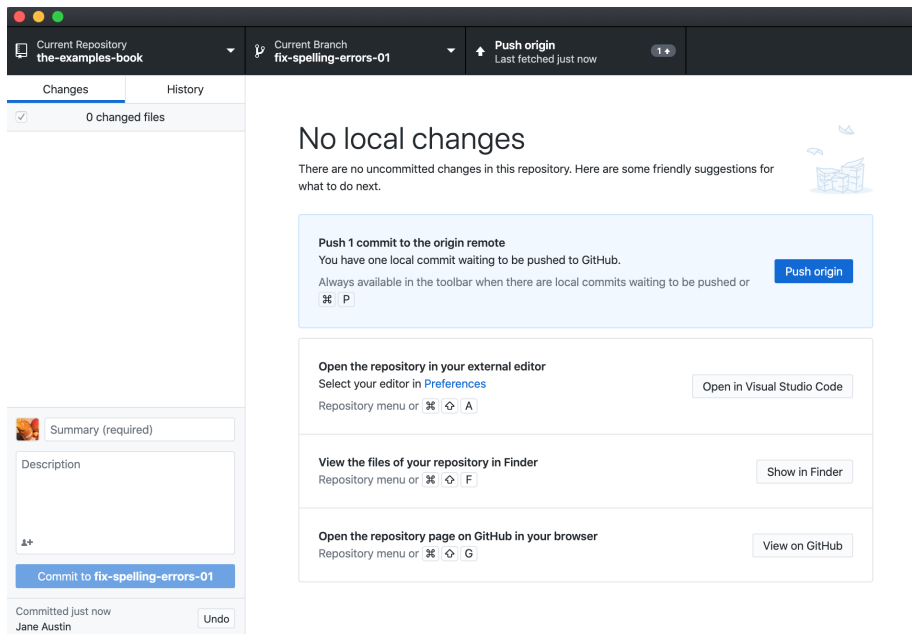


Figure 7.2:

2. Simply click on the “Push origin” button in order to push your local commits to the remote origin (which is in this case, a remote branch called `fix-spelling-errors-01`):

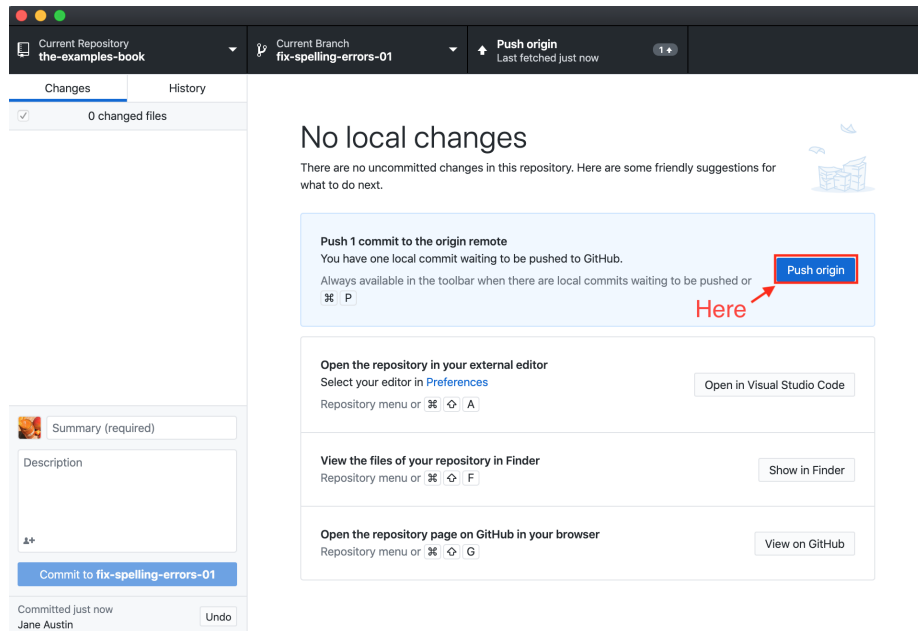
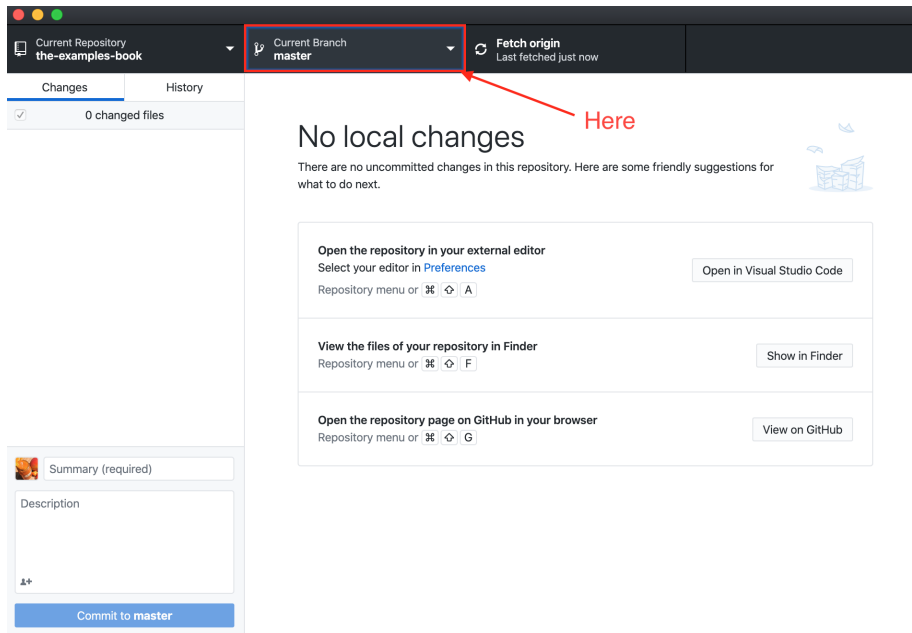


Figure 7.3:

3. You can verify that the changes have been made by navigating to the branch on github.com, and checking the commit history.

Create a new branch

1. In GitHub Desktop, click on the “Current Branch” dropdown:



2. Click on the “New Branch” button:

Branches

Pull Requests

Filter


New Branch

Default Branch


✓ master

5 days ago

Other Branches

 origin/kevinamstutz-patch-1

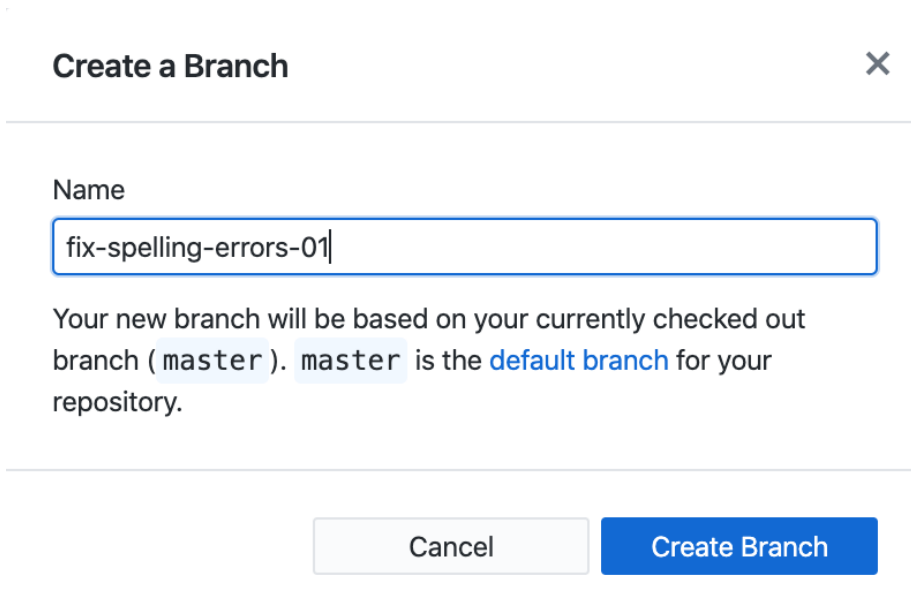
5 days ago

 Choose a branch to merge into **master**

Here



When presented with the following screen, ensure that your new branch will be based on the `master` branch:



Create a Branch ✕

Name

fix-spelling-errors-01

Your new branch will be based on your currently checked out branch (`master`). `master` is the **default branch** for your repository.

Cancel Create Branch

4. Type whatever name you'd like to give the new branch. In this case, we are calling it `fix-spelling-errors-01`. Click "Create Branch". 5. Your current branch should now be `fix-spelling-errors-01` or whatever name you entered in step (4). You can see this in the dropdown:

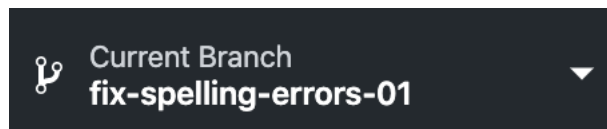
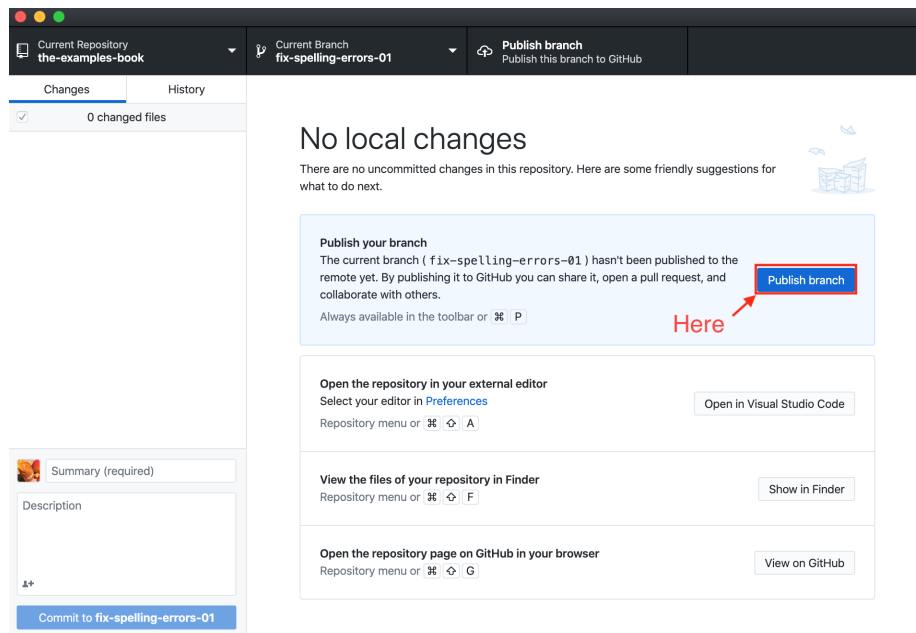


Figure 7.4:

Publish your branch to GitHub

1. If the branch you created is not already present remotely, you'll have a button available to you that says "Publish Branch". Clicking this button will push the branch to the remote repository (on github.com):



2. You can confirm that the branch has been successfully pushed to github.com by navigating to the repository on github, and clicking on the “branches” tab:

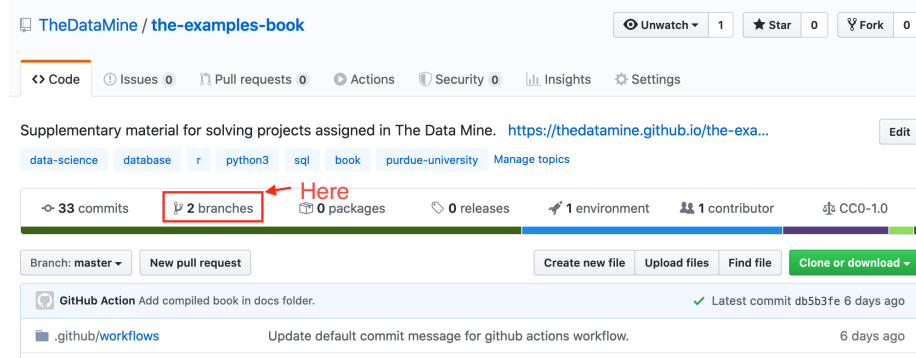


Figure 7.5:

Create a pull request

1. If the branch you are working on is already published remotely, and the remote repository and local repository are both up to date, you will be presented with a screen similar to this:

Note that if your local repository is ahead of the remote repository, you will instead be presented with a screen similar to this:

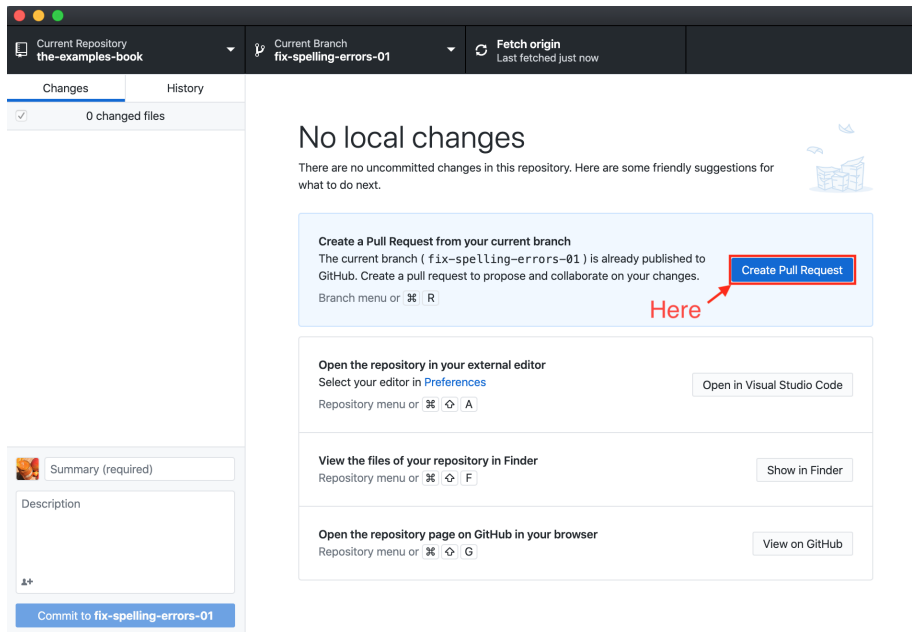


Figure 7.6:

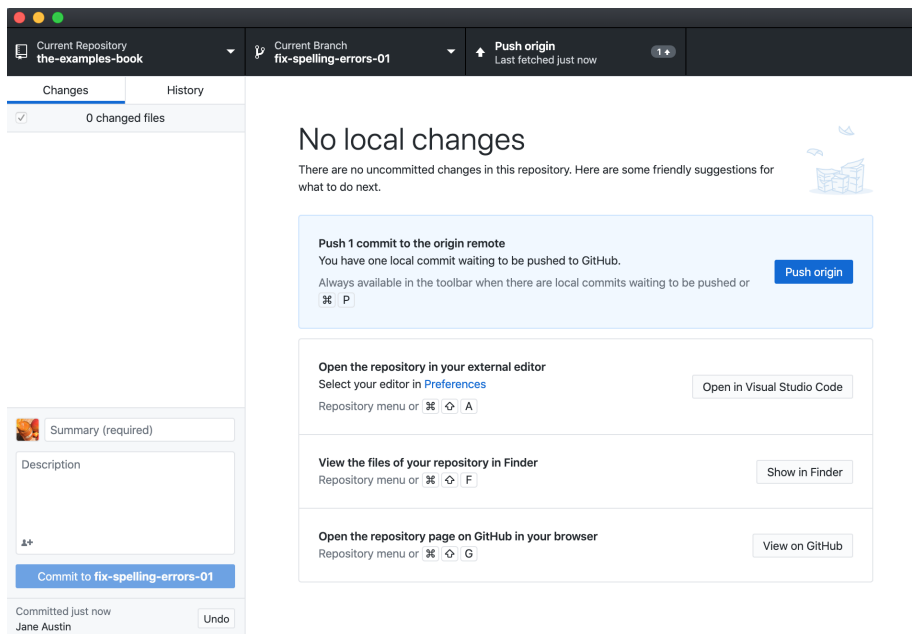


Figure 7.7:

You will first need to push your local commits to the origin (which is the remote `fix-spelling-errors-01` branch in this case) by clicking on the “Push origin” button.

2. Click the “Create Pull Request” button. This will open up a tab in your browser:

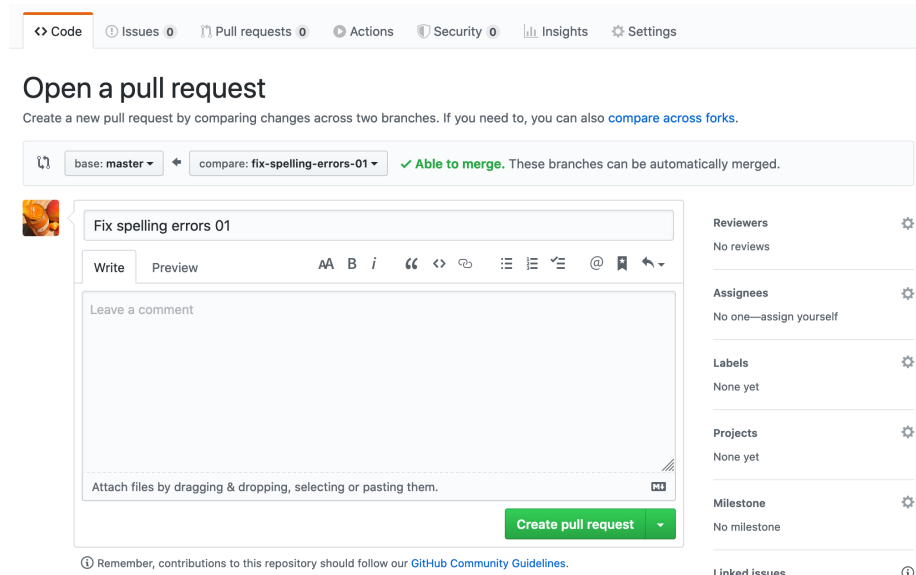


Figure 7.8:

3. Leave a detailed comment about what you’ve modified or added to the book. You can click on “Preview” to see what your comment will look like. GitHub’s markdown applies here. Once satisfied, click “Create pull request”.

Resources

GitHub glossary:

An excellent resource to understand `git` and GitHub specific terminology.

Learn git branching:

An interactive game that teaches you about `git` branching.

VPNs

Chapter 8

FAQs

How do I connect to Scholar from off-campus?

There are a variety of ways to connect to Scholar from off-campus. You can use the ThinLinc web client, or connect using the ThinLinc client application. If you just want to use Jupyter notebooks, you can use JupyterHub. If you just want to use RStudio, you can use RStudio Server.

Is there an advantage to using the ThinLinc client rather than the ThinLinc web client?

Yes. Although it is marginally more difficult to connect with, the ThinLinc client allows the user to copy and paste directly from their native operating system. So for example, if you have an RStudio session opened on your MacBook, you can directly copy and paste code onto Scholar using the ThinLinc client. You are unable to do this via the ThinLinc web client.

Additionally, using the ThinLinc client allows audio to work properly.

GitHub Classroom is not working – can't authorize the account.

This is usually a browser issue. GitHub Classroom does not work well with Microsoft Edge or Internet Explorer. Try Firefox or Safari or Chrome.

In Scholar, on RStudio, my font size looks weird or my cursor is offset.

In scholar, navigate to **Tools > Global Options > Appearance**. You can change your font, including the size and the color scheme. The default, by the way, is the RStudio theme **Modern**, font size 10, and the Editor theme **Textmate**. Make your desired changes, and then click the **Apply** button.

I'm unable to type into the terminal in RStudio.

Try opening a new terminal, try clearing the terminal buffer, or interrupting the current terminal. All these options come from a menu that will pop up when you hit the small down arrow next to the words "Terminal 1" (it might be another number depending on how many terminals are open) which is on the left side right above the terminal in RStudio.

I'm unable to connect to RStudio Server.

Try closing it, clearing your cookies, and using the original link: <https://rstudio.scholar.racac.purdue.edu/>, or for ease of scrolling, try <https://desktop.scholar.racac.purdue.edu>, and open up RStudio from within the ThinLinc web client.

RStudio initialization error.

1. Navigate to <https://desktop.scholar.racac.purdue.edu/> and login using your Purdue Career Account credentials.
2. Open a terminal (*not* RStudio, but rather, a terminal).
3. Run the following commands:

```
/class/datamine/apps/fix_rstudio_initialization_error
```

This script cycles through the frontends (except the current frontend) and kills all of your user processes. At the end, the script kills the user processes on your current frontend, causing you to lose your connection. Once this is complete and you reconnect to Scholar, there should no longer be any lingering processes that prevent you from logging in.

RStudio crashes when loading a package.

Follow the directions under `rstudio` initialization error.

RStudio license expired.

If you are getting this message on a Saturday night, this is due to the Scholar frontends rebooting. Orphan processes are cleaned up and memory reclaimed. This process can cause a disruption in the communication that RStudio needs to do. This disruption is interpreted as a licensing issue. Simply wait and try again the next day.

RStudio is taking a long time to open.

It is possible that you saved a large `.RData` file the last time that you closed RStudio. (It is OK to avoid saving the `.RData` file, for this reason.) If you did save your `.RData` file, and you want to remove it now, you can do the following:

1. Inside RStudio, select the **Terminal** (located near the **Console**; do not use the **Console** itself).
2. Inside the **Terminal**, type: `cd` so that you will be working in your home directory. You can double-check this by typing: `pwd` and it should show you that you are working in `/home/mdw` (but of course `mdw` will be whatever your username is).
3. Type: `rm .RData` (be sure to put a space between `rm` and `.RData`).

Now your R workspace should be fresh when you log out of RStudio (by clicking the log out button in the upper-right-hand corner of RStudio). In other words, next time, you will not have old variables hanging around, from a previous session. Now your RStudio should load more quickly at the beginning.

How can you run a line of R code in RStudio without clicking the “Run” button?

1. Click anywhere on the line (you do not need to highlight the line, and you do not need to click at the start or end of the line; anywhere on the line is ok).
2. Type the “Control” and “Return” keys together to run that line.

My R session freezes.

1. Log out of Scholar.
2. Log back into Scholar using the ThinLinc client.

Before entering your password in ThinLinc, be sure to click on the “End Existing Session” option in the ThinLinc window (to the left of where you type your password). This will resent your Scholar session.

White screen issue when loading RStudio.

1. Log in to Scholar.
2. Open a terminal (click the black box at the bottom of the screen).
3. Run the following commands:

```
# Note: You can use -fe01, -fe02, ..., -fe06 instead of -fe00.  
# Until you find one that works.  
ssh -Y scholar-fe00  
module load rstudio  
rstudio
```

4. When Scholar is reset, load RStudio by opening a terminal and running the following commands:

```
ml gcc  
ml rstudio  
rstudio
```

Scholar is slow.

Possibility one:

Most of the files we use in this class would require dozens of seconds to load using `read.csv()` in R.

Here is another trick to save you some time in data import:

1. Read only the first, say, 10000 rows of data (see instructions below), and complete your code using the smaller dataset. The code works for the subset of data should also work for the complete data. **This output is not your final answer!**

2. Once you complete the code, read in the entire dataset, and run the code to RStudio. You may even close the ThinLinc after submitting the code as long as you do not close your RStudio window. Closing RStudio will stop your code from running. It is also highly recommended to save your code prior to running it.
3. Some time (e.g., a few hours) later, you can come back and check your output. Scholar is a computing facility that is always on, and thus you can leave it do the work.

How do you read the first 10000 rows then? For example, we usually use the following line of code to read all of the election data:

```
myDF <- read.csv('/class/datamine/data/election/itcont2020.txt')
```

Now, with an additional parameter `nrows`, you can decide how many rows to read:

```
myDF_short <- read.csv('/class/datamine/data/election/itcont2020.txt', nrows = 10000)
```

Possibility two:

You could be close to using 100% of your quota on scholar.

1. Log into Scholar using the ThinLinc client.
2. Open a terminal, and run the following command: `myquota`.

Important note: It will ask for your Purdue password (but won't show it to you as you type). If your quota is at or near 100%, you will need to delete some of your files on Scholar. A healthy server needs < 80% full.

There are no menus in Scholar.

Although this is a less common problem, it can happen if you accidentally selected "one empty panel" when you first logged into Scholar. To fix this, do as follows:

1. Open a terminal by clicking on the **Home** icon – it looks like a house –. This will open a window in the **File Manager**. Then, choose from the menu in **File Manager** window: **File > Open Terminal Here**.
2. Run **exactly** the following command in the terminal:

```
cp /etc/xdg/xfce4/panel/default.xml ~/.config/xfce4/xfconf/xfce-perchannel-xml/xfce4-panel.xml
```

3. Log out of Scholar. As this can be hard without menus, run in the terminal:

```
killall -9 -u $USER
```

Running the command above will kill your session. When you log back in, the menu system will work properly.

Firefox in Scholar won't open because multiple instances running.

The easy fix:

1. Open your File Browser in Scholar.
2. Choose the Option View > Show Hidden Files.
3. Inside your home directory, throw away the directory .mozilla.

Now your Firefox should load.

More complicated fix (*if the easy fix doesn't work*):

1. Open a terminal, and run the following commands:

```
cd ~/.mozilla/firefox
rm profiles.ini
```

Alternatively, you can run `rm -rf ./mozilla`.

Important note: *Make sure that you don't leave a space after the period. The period needs to be directly next to the slash.*

2. Log out of Scholar.
3. Log back into Scholar using the ThinLinc client. When logging in, after you type your password in ThinLinc, but before you press the “Connect” button, make sure that you check the box “End Existing Session”.

How to transfer files between your computer and Scholar.

Solution 1: email

Attach the files in an e-mail to yourself. To do so inside Scholar, use the browser to log on to your e-mail client (located in the dock and the icon looks like a blue-and-green picture of the globe).

Solution 2: use scp

To send a file from your computer to Scholar:

1. Open a terminal.
2. Go to the directory where you have the file you want to transfer using the command with updated directory location `/directory/with/file/to/send`

```
cd /directory/with/file/to/send
```

3. Run the following command with the corresponding `filename`, `username`, and `where/to/put/filename` directory

```
scp filename username@scholar.rcac.purdue.edu:/where/to/put/filename
```

Example: Dr. Ward wants to transfer the file titled `my_file.txt` to a folder in his main directory called `my_folder`, he would run:

```
scp my_file.txt mdw@scholar.rcac.purdue.edu:/my_folder/my_file.txt
```

To send a file from Scholar to your computer:

1. Open a terminal.
2. Run the following command with the corresponding `file/to/send/filename`, `username`, and `where/to/put/filename` directory:

```
scp username@scholar.rcac.purdue.edu:/file/to/send/filename /where/to/put
```

Example: If Dr. Ward wants to transfer the file titled `my_file.txt` located in a folder named `my_folder_in_scholar` to a folder in his personal computer called `my_folder` in his main directory, he would run:

```
scp mdw@scholar.rcac.purdue.edu:/my_folder/my_file.txt /my_folder/my_file.txt
```

Solution 3: use FileZilla

1. Download and install the FileZilla Client onto your personal computer. FileZilla uses sftp ([S]SH [F]ile [T]ransfer [P]rotocol) to transfer files to and from Scholar.
2. To connect to Scholar from FileZilla, enter the following information and click “Quickconnect”:

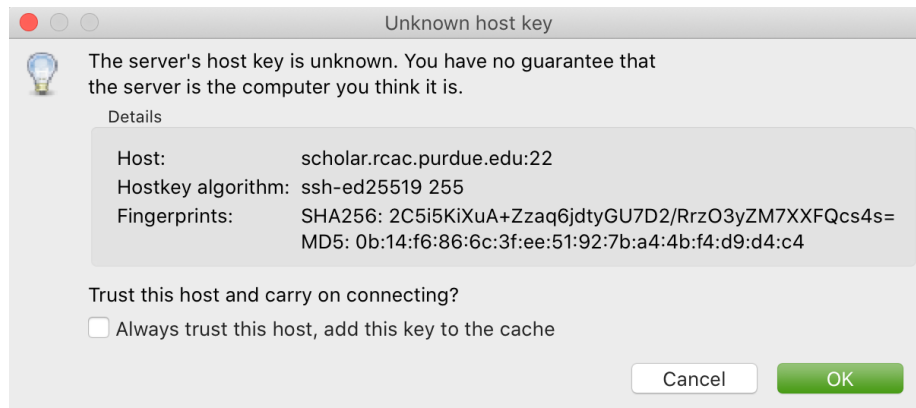
Host: scholar.rcac.purdue.edu

Username: <your_scholar_username> (For example, Dr. Ward’s would be *mdw*. See [here](#).)

Password: <your_scholar_password>

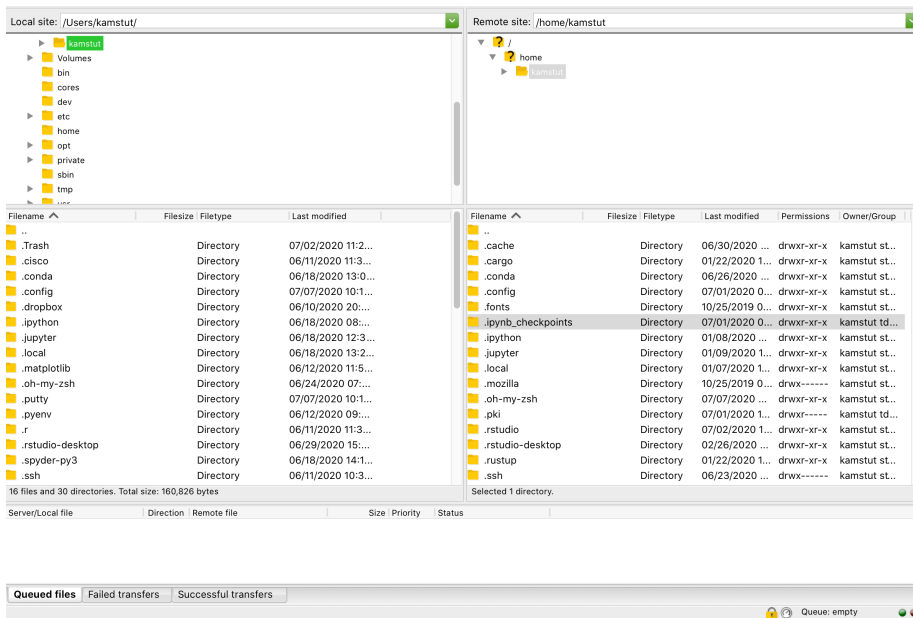
Port: 22

After clicking “Quickconnect” you may be asked something similar to the following:



Select “OK” and establish the connection.

3. The files on the left-hand side are your local computer’s files. The files on the right-hand side are the files in Scholar. To download files from Scholar, right click the file(s) on the Scholar side (right-hand side) and click “Download”. To upload files to Scholar, right click the file(s) on your local machine (left-hand side) and click “Upload”.



Solution 4: use SFTP

On windows:

1. Open your start menu and click on cmd.
2. Type: `sftp username@scholar.rcac.purdue.edu` (replace “username” with your username).
3. Once connected, follow the documentation from RCAC to transfer files.

On mac:

1. Open a terminal.
2. Type: `sftp username@scholar.rcac.purdue.edu` (replace “username” with your username).
3. Once connected, follow the documentation from RCAC to transfer files.

ThinLinc app says you can't create any more sessions.

You will need to close any other sessions that you are running and start a new one. To do so, click on a little box under the password, over on the left-hand side, which says “End existing session”.

How to install ThinLinc on my computer.

See [here](#).

Forgot my password or password not working with ThinLinc.

First, ensure you are typing it correctly by typing it somewhere you can see, and copying and pasting the password back into ThinLinc. Remember that Scholar wants your Career Account credentials, not the Boiler Key.

If you are using the app version of ThinLinc, try using the web version or Jupyter.

If the steps above do not work, you need to change your Career Account password. To do so:

1. Go to Secure Purdue.
2. Click on the option “Change your password”.
3. After logging in, search for the link “Change Password” that “Allows you to change your Purdue Career Account password”.

Jupyter Notebook download error with IE.

Please note that Internet Explorer is **not** a recommended browser. If still want to use Explorer, make sure you download the notebook as “All Files” (or something similar). That is, we need to allow the browser to save in its natural format, and not to convert the notebook when it downloads the file.

Jupyter Notebook kernel dying.

- Make sure you are using the R 3.6 (Scholar) kernel.
- Make sure you are using `https://notebook.scholar.rcac.purdue.edu` and not `https://notebook.brown.rcac.purdue.edu`. (Use Scholar instead of Brown.)
- Try clicking **Kernel > Shutdown**, and then reconnect the kernel.
- If one particular Jupyter Notebook template gives you this error, then create a new R 3.6 (Scholar) file.

*PYTHON KERNEL NOT WORKING, JUPYTER NOTEBOOK WON'T SAVE.*121

- Try re-running the code from an earlier project that you had set up and working using Jupyter Notebooks.
- One student needed to re-run the setup command one time in the terminal:

```
source /class/datamine/data/examples/setup.sh
```

- You could be close to using 100% of your quota on scholar.
1. Log into Scholar using the ThinLinc client.
 2. Open a terminal, and run the following command: `myquota`.

Important note: It will ask for your Purdue password (but won't show it to you as you type). If your quota is at or near 100%, you will need to delete some of your files on Scholar. A healthy server needs < 80% full, aim for that.

Python kernel not working, Jupyter Notebook won't save.

1. Navigate to <https://notebook.scholar.rcac.purdue.edu/>, and login.
2. Click on the “Running” tab and shutdown all running kernels.
3. Log into Scholar using the ThinLinc client.
4. Open a terminal, and run the following commands:

```
pip uninstall tornado(  
/class/datamine/data/examples/setup2.sh
```

5. Go back to <https://notebook.scholar.rcac.purdue.edu/>, click on “Control Panel” in the upper right hand corner.
6. Click the “Stop My Server” button, followed by the green “My Server” button.

Installing my_package for Python

Do **not** install packages in Scholar using:

```
pip install my_package
```

or

```
pip install my_package --user
```

We've tried to provide you with a ready-made kernel with every package you would want or need. If you need a newer version of some package, or need a package not available in the kernel, please send us a message indicating what you need. Depending on the situation we may point you to create your own kernel.

Displaying multiple images after a single Jupyter Notebook Python code cell.

Sometimes it may be convenient to have several images displayed after a single Jupyter cell. For example, if you want to have side-by-side images or graphs for comparison. The following code allows you to place figures side-by-side or in a grid.

Note you will need the included import statement at the very top of the notebook.

```
import matplotlib.pyplot as plt

number_of_plots = 2
fig, axs = plt.subplots(number_of_plots)
fig.suptitle('Vertically stacked subplots', fontsize=12)
axs[0].plot(x, y)
axs[1].imshow(img)
plt.show()

number_of_plots = 3
fig, axs = plt.subplots(1, number_of_plots)
fig.suptitle('Horizontally stacked subplots', fontsize=12)
axs[0].plot(x, y)
axs[1].imshow(img)
axs[2].imshow(img2)
plt.show()

number_of_plots_vertical = 2
number_of_plots_horizontal = 2

# 2 x 2 = 4 total plots
fig, axs = plt.subplots(number_of_plots_vertical, number_of_plots_horizontal)
fig.suptitle('Grid of subplots', fontsize=12)
axs[0][0].plot(x, y) # top left
axs[0][1].imshow(img) # top right
axs[1][0].imshow(img2) # bottom left
```

```
axs[1][1].plot(a, b) # bottom right  
plt.show()
```

RMarkdown “Error: option error has NULL value” when knitting“.

This error message occurs when using the RStudio available on Scholar via ThinLinc, and running a code chunk in RMarkdown by clicking the green “play” button (Run Current Chunk). Do *not* click on the green triangle “play” button. Instead, knit the entire document, using the “knit” button that looks like a ball of yarn with a knitting needle on it.

How do you create an RMarkdown file?

Any text file with the .Rmd file extension can be opened and knitted into a PDF (or other format). If you’d like to create an RMarkdown file in RStudio, you can do so.

1. Open an RStudio session.
2. Click on **File > New File > RMarkdown....**
3. You may put R code into the R blocks (the grey sections of the document), and put any comments into the white sections in between.

This is an excellent guide to RMarkdown, and this is a cheatsheet to get you up and running quickly.

Problems building an RMarkdown document on Scholar.

If you are having problems building an RMarkdown document on Scholar, try the following:

- Dump the previously loaded modules.

1. Open up a terminal.
2. Run the following commands:

```
module purge
ml gcc
ml rstudio
rstudio
```

This will purge (remove) previously loaded modules.

- Remove your R directory:
 1. Open up a terminal.
 2. Run the following commands:

```
cd ~
rm -rf R
```

This will force the removal of your R directory. It will remove your old R libraries. They will reload the newest versions if you install them again, and as you use them.

This is recommended, especially at the start of the academic year.

If your R is taking a long time to open, see [here](#).

How can I use SQL in RMarkdown?

When you use SQL in RMarkdown you can highlight the code in code chunks just like R by writing “sql” instead of “r” in the brackets:

““asis

```
SELECT * FROM table;
```

““

You will notice that all the SQL code chunks provided in the template have the option `eval=F`. The option `eval=F` or `eval=FALSE` means the SQL statements would be shown in your knitted document, but without being executed.

To actually *run* SQL inside RMarkdown see [here](#).

You can read about the different languages that can be displayed in RMarkdown [here](https://bookdown.org/yihui/rmarkdown/language-engines.html): <https://bookdown.org/yihui/rmarkdown/language-engines.html>.

Copy/paste from terminal inside RStudio to RMarkdown.

If you're using the terminal inside the Scholar RStudio at <https://rstudio.scholar.rcac.purdue.edu>, right clicking won't work. A trick that does work (and often works in other situations as well) is the keyboard shortcut ctrl-insert for copy and shift-insert for paste. Alternatively, use the Edit/Copy from the menu in the terminal.

How do I render an image in a shiny app?

There are a variety of ways to render an image in an RShiny app. See [here](#).

The package `my_package` is not found.

The package might not be installed. Try running:

```
install.packages("ggmap")
```

Note that if you have already run this on ThinLinc, there is no need to do it again.

Another possibility is that the library is not loaded, try running:

```
library(ggmap)
```

Problems installing ggmap.

Two possible fixes:

1. Open a terminal and run:

```
rm -rf ~/R
```

After that, re-open RStudio and re-install ggmap:

```
install.packages("ggmap")  
  
# Don't forget to load the package as well  
library(ggmap)
```

2. Open a terminal and run:

```
module load gcc/5.2.0
```

After that, restart all RStudio processes.

Error: object_name is not found

In R if you try to reference an object that does not yet exist, you will receive this error. For example:

```
my_list <- c(1, 2, 3)
mylist
```

In this example you will receive the error `Error: object 'mylist' not found`. The reason is `mylist` doesn't exist, we only created `my_list`.

Zoom in on ggmap.

Run the following code in R:

```
?get_googlemap
```

Under the arguments section you will see the argument `zoom` and can read about what values it can accept. For the zoom level, a map with `zoom=9` would not even show the entire state of California. Try different integers. Larger integers “zoom in” and smaller integers “zoom out”.

Find the latitude and longitude of a location.

1. Install the `ggmap` package.
2. Run the following lines of code to retrieve latitude and longitude of a location:

```
as.numeric(geocode("London"))
```

Replace “London” with the name of your chosen location.

Problems saving work as a PDF in R on Scholar.

Make sure you are saving to your own working directory:

```
getwd()
```

This should result in something like: `/home/<username>/...` where `<username>` is your username. Read this to find your username.

If you don't see your username anywhere in the resulting path, instead try:

1. Specifying a different directory:

```
dev.print(pdf, "/home/<username>/Desktop/project4map.pdf")
```

Make sure you replace `<username>` with your username.

2. Try setting your working directory before saving:

```
setwd("/home/<username>/Desktop")
```

Make sure you replace `<username>` with your username.

What is a good resource to better understand HTML?

<https://www.geeksforgeeks.org/html-course-structure-of-an-html-document/>

Is there a style guide for R code?

<https://style.tidyverse.org/>

Is there a guide for best practices using R?

<https://www.r-bloggers.com/r-code-best-practices/>

1. Comment what you are going to do.
2. Code – what did you do?
3. Comment on the output – what did you get?

Tips for using Jupyter notebooks.

See [here](#).

What is my username on Scholar?

To find your username on Scholar:

1. Open a terminal.
2. Execute the following code:

```
echo $USER
```

How to submit homework to GitHub without using Firefox?

You can submit homework to GitHub without using Firefox by using `git` in a terminal. You can read more about `git` [here](#).

How and why would I need to “escape a character”?

You would need to escape a character any time when you have a command or piece of code where you would like to represent a character literally, but that character has been reserved for some other use. For example, if I wanted to use `grep` to search for the `$` character, literally, I would need to escape that character as its purpose has been reserved as an indicator or anchor for the end of the line.

```
grep -i "$50.00" some_file.txt
```

Without the `\` this code would not work as intended. Another example would be if you wanted to write out $10*10*10 = 1000$ in markdown. If you don't escape the asterisks, the result may be rendered as $101010 = 1000$, which is clearly not what was intended. For this reason, we would type out:

```
10\*10\*10 = 1000
```

Which would then have its intended effect.

*HOW CAN I FIX THE ERROR “ILLEGAL BYTE SEQUENCE” WHEN USING A UNIX UTILITY LIKE CUT?*129

How can I fix the error “Illegal byte sequence” when using a UNIX utility like cut?

Often times this is due to your input having illegal, non-utf-8 values. You can find all lines with illegal values by running:

```
grep -axv '.*' file
```

To fix this issue, you can remove the illegal values by running:

```
iconv -c -t UTF-8 < old_file > new_file
```


Chapter 9

Projects

Templates

Our course project template can be found here, or on Scholar:

```
/class/datamine/apps/templates/project_template.Rmd
```

Students in STAT 19000, 29000, and 39000 are to use this as a template for all project submissions. The template includes a code chunk that “activates” our Python environment, and adjusts some default settings. In addition, it provides examples on how to include solutions for Python, R, Bash, and SQL. Every question should be clearly marked with a third-level header (using 3 #s) followed by **Question X** where X is the question number. Sections for question solutions should be added or removed based on the number of questions in the given project. All code chunks are to be run and solutions displayed for the compiled PDF submission.

Any format or template related questions should be asked in Piazza.

STAT 19000

Project 1

Motivation: In this project we are going to jump head first into The Data Mine. We will load datasets into the R environment, and introduce some core programming concepts like variables, vectors, types, etc. As we will be “living” primarily in an IDE called RStudio, we will take some time to learn how to connect to it, configure it, and run code.

Context: This is our first project as a part of The Data Mine. We will get situated, configure the environment we will be using throughout our time with The Data Mine, and jump straight into working with data!

Scope: r, rstudio, Scholar

Learning objectives:

- Utilize other Scholar resources: rstudio.scholar.rcac.purdue.edu, notebook.scholar.rcac.purdue.edu, desktop.scholar.rcac.purdue.edu, etc.
- Install R and setting up a working environment.
- Explain and demonstrate: positional, named, and logical indexing.
- Read and write basic (csv) data.

Dataset: /class/datamine/data/disney/splash_mountain.csv

1. Read the webpage here. Scholar is the computing cluster you will be using throughout the semester, and your time with The Data Mine. Each *node* is an individual machine with CPUs and memory (RAM). How many *cores* and how much *memory* is available, in total, for our 7 frontend nodes? How about for the sub-clusters? How much is available on your computer or laptop?

Item(s) to submit:

- A sentence explaining how much memory and how many cores your personal computer has.
- A sentence explaining how much memory and how many cores the 7 frontends have combined.
- A sentence explaining how much memory and how many cores the 28 sub-clusters have combined.

2. Navigate and login to <https://rstudio.scholar.rcac.purdue.edu> using your Purdue Career Account credentials (without Boilerkey). This is an instance of RStudio Server running on a Scholar frontend! Frontends are labeled. So, for example, scholar-fe01.rcac.purdue.edu is frontend #1. In the lower left hand side of the RStudio screen, you should be able to see a tab labeled “Console”. You can run R code by typing after the > and pressing enter. Check which frontend you are logged in on by running the following in the “Console” tab: `system("hostname")`. Which frontend are you in?

Relevant topics: running R code

Item(s) to submit:

- The # of the frontend your RStudio Server session is running on.

3. From within RStudio, we can run every type of code that you will need to run throughout your time with The Data Mine: Python, R, Bash, SQL, etc. We've created a script for you to run to help configure settings. These settings will allow us to better serve you during the semester. Follow the directions here. Once complete, in RStudio (<https://rstudio.scholar.rcac.purdue.edu>), click on **Session > Restart R**. Once fully restarted, there should be a message that is printed in your "Console" tab. What does the message say?

Item(s) to submit:

- The sentence that is printed in the RStudio "Console".

4. Projects in The Data Mine should all be submitted using our template found here or on Scholar (/class/datamine/apps/templates/project_template.Rmd). At the beginning of every project, the first step should be downloading and/or copying and pasting the template into a .Rmd file in RStudio. Copy and paste the project template into a new RMarkdown file named project01.Rmd. Code chunks are parts of the RMarkdown file that contains code. You can identify what type of code a code chunk contains by looking at the *engine* in the curly braces "{" and "}". How many of each type of code chunk are in our default template?

Hint: You can read about the template here.

Item(s) to submit:

- A list containing the type of code chunk (r, Python, sql, etc), and how many of each code chunks our default template contains.

5. Fill out the project template, replacing the default information with your own. If a category is not applicable to you, put N/A. This template provides examples of how to run each "type" of code we will run in this course. Look for the second R code chunk, and run it by clicking the tiny green play button in the upper right hand corner of the code chunk. What is the output?

Item(s) to submit:

- The output from running the R code chunk.

6. In question (1) we answered questions about CPUs and RAM for the Scholar cluster. To do so, we needed to perform some arithmetic. Instead of using a calculator (or paper), write these calculations using R. Replace the content of the second R code chunk in our template with your calculations.

Relevant topics: templates

Item(s) to submit:

- The R code chunk with your calculations, and output.

7. In (6) we got to see how you can type out arithmetic and R will calculate the result for you. One constant throughout the semester will be loading datasets into R. Load our dataset into R by running the following code:

```
dat <- read.csv("/class/datamine/data/disney/splash_mountain.csv")
```

Confirm the dataset has been read in by running the `head` function on it. `head` prints the first few rows of data:

```
head(dat)
```

`dat` is a variable which contains our data! We can name this variable anything we want, we do *not* have to name it `dat`. Run our code to read in our dataset, this time, instead of naming our resulting dataset `dat`, name it `splash_mountain`. Place all of your code into a new R code chunk under a new level 3 header (i.e. `### Question 7`).

Relevant topics: reading data in R

Item(s) to submit:

- Code used to answer this question in a code chunk in our template.
- Output of `head`.

8. Let's pretend we are now done with our project. We've written some R code, maybe added some text explaining what we did, and we are ready to turn things in. For this course, we will turn in a variety of work, depending on the project. We will always require a PDF which contains text, code, and code output. Normally we would erase any code chunks from the template that are not used, however, for this project just keep that content. This PDF is generated by "Knitting" a PDF. In addition, if the project uses R code, you will need to copy and paste R code into an R script (file ending with .R). If you are submitting Python code too, you will need to copy and paste Python code into a Python script (file ending with .py). Let's practice. Compile your project to a PDF, create an R script and copy and paste all of your R code from your RMarkdown file (.Rmd), to a new file called project01.R. Include only the R code you wrote. Follow the directions in Brightspace to upload and submit your RMarkdown file, compiled PDF, and R script.

Relevant topics: templates

Item(s) to submit:

- Resulting knitted PDF.
- project01.R script containing all of your code from your R chunks in the .Rmd file.

Project 2

Motivation: The R environment is a powerful tool to perform data analysis. R is a tool that is often compared to Python. Both have their advantages and disadvantages, and both are worth learning. In this project we will dive in head first and learn the basics while solving data-driven problems.

Context: Last project we set the stage for the rest of the semester. We got some familiarity with our project templates, and modified and ran some R code. In this project, we will continue to use R within RStudio to solve problems. Soon you will see how powerful R is and why it is often a more effective tool to use than spreadsheets.

Scope: r, vectors, indexing, recycling

Learning objectives:

- List the differences between lists, vectors, factors, and data.frames, and when to use each.
- Explain and demonstrate: positional, named, and logical indexing.
- Read and write basic (csv) data.
- Demonstrate the ability to use the following functions to solve data-driven problem(s): mean, var, table, cut, paste, rep, seq, sort, order, length, unique, etc.
- Explain what “recycling” is in R and predict behavior of provided statements.

Dataset

The following questions will use the dataset found in Scholar:

`/class/datamine/data/disney/metadata.csv`

A public sample of the data can be found here: `/class/datamine/data/disney/metadata.csv`

1. Use the `read.csv` function to load our dataset into a `data.frame` called `meta`. Note that `read.csv` *by default* loads data into a `data.frame`. We will learn about `data.frames` later, for now, print the first few rows of `meta`.

Relevant topics: reading data in r, head

Item(s) to submit:

- R code used to solve the problem in an R code chunk.

2. We’ve provided you with R code below that will extract a column of our `data.frame` into a vector. What is the first value in the vector? What is the 50th value in the vector? That type of data is in the vector?

```
our_vec <- meta$WDWMAXTEMP
```

Relevant topics: indexing in r, type, creating variables

Item(s) to submit:

- R code used to solve the problem in an R code chunk.
- The values of the first, and 50th element in the vector.
- The type of data in the vector.

3. You can access many elements in a vector at the same time. Create three new vectors named: `first50`, `last50`, and `mix`. `first50` should contain the first 50 values of our `our_vec` vector, and `last50` should contain the last 50 values. `mix` should contain the sum of each element of `first50` being added to each element of `last50`.

Hint: Make sure that `first50` and `last50` are the same length. If you get a warning message that reads “longer object length is not a multiple of shorter object length”, you’ve done something wrong.

Relevant topics: indexing in r, creating variables, length

Item(s) to submit:

- R code used to solve this problem.
- The `head` of each of the three vectors.

4. In (3) we were able to rapidly add values together from two different vectors. Both vectors were the same size, hence, it was obvious which elements in each vector were added together. Create a new vector called `hot` which contains only the values which are greater than or equal to 80 (our vector contains max temperatures for days at Disney World). How many elements are in `hot`? Calculate the sum of `hot` and `first50`, do we get a warning? Read this and then explain what is going on.

Relevant topics: logical indexing, length, recycling

Item(s) to submit:

- R code used to solve this problem.
- 1-2 sentences explaining what is happening when we are adding two vectors of different lengths.

5. Given what we learned in (4) how would we double every odd value in `hot`, and cut in half every even value in `hot`, in a single line of R code?

Relevant topics: recycling

Item(s) to submit:

- R code used to solve this problem.
- head of the result.

6. Run the code below in order to extract the `min_temp`, `max_temp`, and `mean_temp` vectors from our `data.frame`. Are the vectors all the same length? Create a new vector called `meanminmax` that contains the average of the `min_temp` and `max_temp`, element-wise. Calculate the absolute average difference between the `mean_temp` vector, and the `meanminmax` vector. What is the index of the largest difference between `mean_temp` and `meanminmax`? If you replace `INDEX` with the index you found, you will get information about the day: `meta[INDEX,]`. What is the largest difference between the `mean_temp` and `meanminmax` vectors? Explain whether or not that surprises you.

```
min_temp <- meta$WDWMINTMP
mean_temp <- meta$WDWMEANTMP
max_temp <- meta$WDWMAXTEMP
```

7. The following code creates three graphics. The first two are created just using some core R functions, and the last is created using a package called `ggplot`. We will learn more about all of these things later on. For now, pick your favorite graphic, and write 1-2 sentences explaining why it is your favorite, what could be improved, and include any interesting observations (if any).

```
dat <- table(meta$SEASON)
names(dat) <- tolower(names(dat))
par(mar=c(1,8,1,1))
barplot(dat, main="Seasons", xlab="Number of Days in Each Season", las=2, horiz=TRUE, col="black")
```

```
dat <- tapply(meta$WDWMEANTMP, meta$DAYOFYEAR, mean, na.rm=T)
seasons <- tapply(meta$SEASON, meta$DAYOFYEAR, function(x) unique(x)[1])
pal <- c("#4E79A7", "#F28E2B", "#A0CBE8", "#FFBE7D", "#59A14F", "#8CD17D", "#B6992D", "#D95F02")
colors <- factor(seasons)
levels(colors) <- pal
par(oma=c(7,0,0,0), xpd=NA)
barplot(dat, main="Average Temperature", xlab="Jan 1 (Day 0) - Dec 31 (Day 365)", ylab="Average Temperature", las=2, col=colors)
legend(-65, -50, legend=levels(factor(seasons)), lwd=5, col=pal, ncol=3, cex=0.8, box.col="white", box.lty=1)
```

```
library(ggplot2)
library(tidyverse)
summary_temperatures <- meta %>%
  select(MONTHOFYEAR, WDWMAXTEMP:WDWMEANTMP) %>%
  group_by(MONTHOFYEAR) %>%
  summarise_all(mean, na.rm=T)
ggplot(summary_temperatures, aes(x=MONTHOFYEAR)) +
  geom_ribbon(aes(ymin = WDWMINTEMP, ymax = WDWMAXTEMP), fill = "#ceb888", alpha=.5)
```

```
geom_line(aes(y = WDWMEANTEMP), col="#5D8AA8") +  
geom_point(aes(y = WDWMEANTEMP), pch=21,fill = "#5D8AA8", size=2) +  
theme_classic() +  
labs(x = 'Month', y = 'Temperature', title = 'Average temperature range' ) +  
scale_x_continuous(breaks=1:12, labels=month.abb)
```

Project 3

Motivation: `data.frames` are the primary data structure you will work with when using R. It is important to understand how to insert, retrieve, and update data in a `data.frame`.

Context: In the previous project we got our feet wet, and ran our first R code, and learned about accessing data inside vectors. In this project we will continue to reinforce what we've already learned and introduce a new, flexible data structure called `data.frames`.

Scope: `r`, `data.frames`, recycling, factors

Learning objectives:

- Explain what “recycling” is in R and predict behavior of provided statements.
- Explain and demonstrate how R handles missing data: NA, NaN, NULL, etc.
- Demonstrate the ability to use the following functions to solve data-driven problem(s): mean, var, table, cut, paste, rep, seq, sort, order, length, unique, etc.
- Read and write basic (csv) data.
- Explain and demonstrate: positional, named, and logical indexing.
- List the differences between lists, vectors, factors, and `data.frames`, and when to use each.

Dataset: `/class/datamine/data/disney`

1. Read the dataset `/class/datamine/data/disney/metadata.csv` into a `data.frame` called `meta`. Read the dataset `/class/datamine/data/disney/splash_mountain.csv` into a `data.frame` called `splash_mountain`. How many columns, or features are in each dataset? How many rows or observations?

Relevant topics: `str`

Item(s) to include:

- R code used to solve the problem.
- How many columns or features in each dataset?

2. Splash Mountain is a fan favorite ride at Disney World’s Magic Kingdom theme park. `splash_mountain` contains a series of dates and datetimes. For each datetime, `splash_mountain` contains a posted minimum wait time, `SPOSTMIN`, and an actual minimum wait time, `SACTMIN`. What is the average minimum posted wait time for Splash Mountain? What is the standard deviation? Based on the fact that `SPOSTMIN` represents the posted minimum wait time for our ride, does our mean and standard deviation make sense? Explain.

Hint: If you got NA or NaN as a result, see here.

Relevant topics: mean, var, NA, NaN

Item(s) to submit:

- R code used to solve this problem.
- The results of running the R code.
- 1-2 sentences explaining why or why not the results make sense.

3. In (2) we got some peculiar values for the mean and standard deviation. If you read the “attractions” tab in the file `/class/datamine/data/disney/touringplans_data_dictionary.xlsx`, you will find that -999 is used as a value in `SPOSTMIN` and `SACTMIN` to indicate the ride as being closed. Recalculate the mean and standard deviation of `SPOSTMIN`, excluding values that are -999. Does this seem to have fixed our problem?

Relevant topics: NA, mean, var, indexing, which

Item(s) to submit:

- R code used to solve this problem.
- The result of running the R code.
- A statement indicating whether or not the value look reasonable now.

4. `SPOSTMIN` and `SACTMIN` aren’t the greatest feature/column names. An outsider looking at the `data.frame` wouldn’t be able to imme-

diately get the gist of what they represent. Change `SPOSTMIN` to `posted_min_wait_time` and `SACTMIN` to `actual_wait_time`.

Hint: You can always use hard-coded integers to change names manually, however, if you use `which`, you can get the index of the column name that you would like to change. For data.frames like `meta`, this is a lot more efficient than manually counting which column is the one with a certain name.

Relevant topics: `colnames`, `which`

Item(s) to submit:

- R code used to solve the problem.
- The output from executing `names(splash_mountain)` or `colnames(splash_mountain)`.

5. Use the `cut` function to create a new vector called `quarter` that breaks the date column up by quarter. Use the `labels` argument in the `factor` function to label the quarters “q1”, “q2”, ..., “qX” where X is the last quarter. Add `quarter` as a column named `quarter` in `splash_mountain`. How many

Hint: If you have 2 years of data, this will result in 8 quarters: “q1”, ..., “q8”.

Hint: I can generate sequential data using `seq` and `paste0`:

```
paste0("item", seq(1, 5))
```

```
## [1] "item1" "item2" "item3" "item4" "item5"
```

Relevant topics: `cut`, `dates`, `factor`, `paste0`, `seq`, `length`, `unique`

Item(s) to submit:

- R code used to solve the problem.
- The `head` and `tail` of `splash_mountain`.

Project 4

Motivation: Control flow is (roughly) the order in which instructions are executed. We can execute certain tasks or code *if* certain requirements are met using if/else statements. In addition, we can perform operations many times in a loop using for loops. While these are important concepts to grasp, R differs from other programming languages in that operations are usually vectorized and there is little to no need to write loops.

Context: We are gaining familiarity working in RStudio and writing R code. In this project we introduce and practice using control flow in R.

Scope: r, data.frames, recycling, factors, if/else, for

Learning objectives:

- Explain what “recycling” is in R and predict behavior of provided statements.
- Explain and demonstrate how R handles missing data: NA, NaN, NULL, etc.
- Demonstrate the ability to use the following functions to solve data-driven problem(s): mean, var, table, cut, paste, rep, seq, sort, order, length, unique, etc.
- Read and write basic (csv) data.
- Explain and demonstrate: positional, named, and logical indexing.
- List the differences between lists, vectors, factors, and data.frames, and when to use each.
- Demonstrate a working knowledge of control flow in r: if/else statements, while loops, etc.

Dataset: /class/datamine/data/disney

1. In previous project we calculated the mean and standard deviation of the SPOSTMIN (posted minimum wait time). These are vectorized operations (we will learn more about this next project). Instead of using mean, use a loop to calculate the mean, just like the previous project. Do not use sum either.

Hint: Remember, if a value is NA, we don’t want to include it.

Hint: Remember, if a value is -999, it means the ride is closed, we don’t want to include it.

Note: This exercise should make you appreciate the variety of useful functions R has to offer!

Relevant topics: for loops, if/else statements, is.na

Item(s) to submit:

- R code used to solve the problem.
- The mean posted wait time.

2. Choose one of the .csv files containing data for a ride. Use `read.csv` to load the file into a `data.frame` named `ride_name` where “ride_name” is the name of the ride you chose. Use a for loop to loop through the ride file and add a new column called `status`. `status` should contain a string whose value is either “open”, or “closed”. If `SPOSTMIN` or `SACTMIN` is -999, classify the row as “closed”. Otherwise, classify the row as “open”. After `status` is added to your `data.frame`, convert the column to a factor.

Hint: If you want to access two columns at once from a `data.frame`, you can do: `splash_mountain[i, c("SPOSTMIN", "SACTMIN")]`.

Relevant topics: any, for loops, if/else statements, `nrow`

Item(s) to submit:

- R code used to solve the problem.
- The output from running `str` on `ride_name`.

3. Typically you want to avoid using for loops (or even apply functions) when they aren’t needed. Instead you can use vectorized operations and indexing. Repeat (2) without using any for loops or apply functions. Which method was faster?

Hint: To have multiple conditions within the `which` statement, use `|` for logical OR and `&` for logical AND.

Hint: You can start by assigning every value in `status` as “open”, and then change the correct values to “closed”.

Relevant topics: `which`

Item(s) to submit:

- R code used to solve the problem.
- The output from running `str` on `ride_name`.

4. Create a pie chart for open vs. closed for `splash_mountain.csv`. First, use the `table` command to get a count of each status. Use the

resulting table as input to the `pie` function. Make sure to give your pie chart a title that somehow indicates the ride to the audience.

Relevant topics: `pie`, `table`

Item(s) to submit:

- R code used to solve the problem.
- The resulting plot displayed as output in the RMarkdown.

5. Loop through the vector of files we've provided below, and create a pie chart of open vs closed for each ride. Place all 6 resulting pie charts on the same image. Make sure to give each pie chart a title that somehow indicates the ride.

```
ride_names <- c("splash_mountain", "soarin", "pirates_of_caribbean", "expedition_everest")
ride_files <- paste0(c("/class/datamine/data/disney/"), ride_names, ".csv")
```

Hint: To place all of the resulting pie charts in the same image, prior to running the for loop, run `par(mfrow=c(2,3))`.

Relevant topics: `for` loop, `read.csv`, `pie`, `table`

Item(s) to submit:

- R code used to solve the problem.
- The resulting plot displayed as output in the RMarkdown.

Project 5

Motivation: As briefly mentioned in project 4, R differs from other programming languages in that *typically* you will want to avoid using for loops, and instead use vectorized functions and the `apply` suite. In this project we will demonstrate some basic vectorized operations, and how they are better to use than loops.

Context: While it was important to stop and learn about looping and if/else statements, in this project, we will explore the R way of doing things.

Scope: `r`, `data.frames`, recycling, factors, if/else, `for`

Learning objectives:

- Explain what “recycling” is in R and predict behavior of provided statements.
- Explain and demonstrate how R handles missing data: NA, NaN, NULL, etc.
- Demonstrate the ability to use the following functions to solve data-driven problem(s): mean, var, table, cut, paste, rep, seq, sort, order, length, unique, etc.
- Read and write basic (csv) data.
- Explain and demonstrate: positional, named, and logical indexing.
- List the differences between lists, vectors, factors, and data.frames, and when to use each.
- Demonstrate a working knowledge of control flow in r: if/else statements, while loops, etc.
- Demonstrate how apply functions are generally faster than using loops.

Dataset: /class/datamine/data/fars

To get more information on the dataset, see [here](#).

1. The fars dataset contains a series of folders labeled by year. In each folder there is (at least) ACCIDENT.CSV, PERSON.CSV, and VEHICLE.CSV. If you take a peek at ACCIDENT.CSV you’ll notice that the year isn’t complete. Either add a new YEAR column with the *full* year, or fix the variable some other way. Use a loop, and rbind to create a data.frame called accidents. As you are looping through each of the years (from [1975, 1981]), make sure to fix the YEAR.

Relevant topics: rbind, for loops, read.csv

Item(s) to submit:

- R code used to solve the problem.
- The result of `unique(accidents$year)`.

2. How many accidents are there where 1+ drunk drivers were involved in an accident with a school bus?

Hint: Look at the variables DRUNK_DR and SCH_BUS.

Relevant topics: table

Item(s) to submit:

- R code used to solve the problem.
- The result/answer itself.

3. For accidents involving 1+ drunk drivers and a school bus, how many happened in each of the 7 years? Which year had the most qualifying accidents?

Relevant topics: table, which, indexing

Item(s) to submit:

- R code used to solve the problem.
- The results.
- Which year had the most qualifying accidents.

4. Calculate the mean number of motorists involved in an accident (PERSON) with i drunk drivers for i in 0 through 6.

Hint: It is OK that there are no accidents involving just 5 drunk drivers.

Relevant topics: for loops, mean, indexing

Item(s) to submit:

- R code used to solve the problem.
- The output from running your code.

5. Perhaps we have a theory that there are more accidents in cold weather months for Indiana and states around Indiana. Create a barplot that shows the number of accidents by STATE by month (MONTH). First, filter out all data where STATE is not one of: Indiana (18), Illinois (17), Ohio (39), or Michigan (26). What months have the most accidents? Are you surprised by these results? Explain why or why not?

Relevant topics: %in%, barplot

Item(s) to submit:

- R code used to solve the problem.
- The output (plot) from running your code.
- 1-2 sentences explaining which month(s) have the most accidents and whether or not this surprises you.

6. (optional) Spruce up your plot from (5). Do any of the following:

- add vibrant (and preferably colorblind friendly) colors to your plot
- add a title
- add a legend
- add month names or abbreviations instead of numbers

Hint: Here is a resource to get you started.

Item(s) to submit:

- R code used to solve the problem.
- The output (plot) from running your code.

Project 6

Motivation: `tapply` is a powerful function that allows us to group data, and perform calculations on that data in bulk. The “apply suite” of functions provide a fast way of performing operations that would normally require the use of loops. Typically, when writing R code, you will want to use an “apply suite” function rather than a for loop.

Context: The past couple of projects have studied the use of loops and/or vectorized operations. In this project, we will introduce a function called `tapply` from the “apply suite” of functions in R.

Scope: `r`, `for`, `tapply`

Learning objectives:

- Explain what “recycling” is in R and predict behavior of provided statements.
- Explain and demonstrate how R handles missing data: NA, NaN, NULL, etc.
- Demonstrate the ability to use the following functions to solve data-driven problem(s): mean, var, table, cut, paste, rep, seq, sort, order, length, unique, etc.
- Read and write basic (csv) data.
- Explain and demonstrate: positional, named, and logical indexing.
- List the differences between lists, vectors, factors, and data.frames, and when to use each.
- Demonstrate a working knowledge of control flow in r: if/else statements, while loops, etc.
- Demonstrate how apply functions are generally faster than using loops.

Dataset: `/class/datamine/data/fars/7581.csv`

Calculate the number of deaths where there was a drunk driver vs when no drunk driver.

Which state has the most drunk drivers?

1. The dataset, `/class/datamine/data/fars/7581.csv` is the result of question 1 from the previous project. Load up the dataset into a data.frame named `dat`. In the previous project’s question 4, we asked you to calculate the mean number of motorists involved in an accident (PERSON) with `i` drunk drivers for `i` in 0 through 6. Solve this question using `tapply` instead. Which method did you prefer and why?

Relevant topics: `tapply`, `mean`

Item(s) to submit:

- R code used to solve the problem.
- The output/solution.

2. Use `/class/datamine/data/states.csv` to map STATE codes to the names of states.

Hints:

- Make sure to first remove states from `states.csv` that are not in STATE, save the resulting vector as `substate`.

- Create an auxiliary variable containing the `STATE` vector converted to a factor.
- Reorder `substate` by `code` using the `order` function.
- Use the `levels` function to set the levels of our auxiliary variable to the reordered `substate`.

Note: In the next project, we will learn a much more effective way to accomplish this!

Relevant topics: `as.factor`, `order`, `levels`, `%in%`

Item(s) to submit:

- R code used to solve the problem.
- head of `dat`.

3. In the previous project, we calculated how many accidents occurred in 4 selected states, each month. If we wanted to extend this to every state, there would be more steps involved. `tapply` is a perfect fit for such a question. Use `tapply` to calculate the number of accidents (each row/observation is an accident) by month (`MONTH`) for each state (`STATE`). Which state has the most accidents?

Relevant topics: `tapply`

Item(s) to submit:

- R code used to solve the problem.
- The entire output.
- Which state has the most wrecks.

4. Use `tapply` to calculate the percentage of accidents during snowy weather and rainy weather. Use the following image to help you:

Hint: You can solve this using `tapply` twice, or, you can wrap the two conditions you'd like to group by in a `list` by using the `list` function. We will learn more about lists later, however, a `list` is essentially a `vector` containing various types rather than a single type.

Relevant topics: `tapply`, `list`

| 1975-
1979 | 1980-
1981 | 1982-
2006 | 2007-
2009 | 2010-
2012 | 2013-
Later | |
|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|------------------------|--|
| 1 | -- | -- | -- | 1 | 1 | Clear |
| -- | 1 | -- | -- | -- | -- | Normal |
| -- | -- | 1 | 0 | -- | -- | No Adverse Atmospheric Conditions |
| -- | -- | -- | -- | 0 | 0 | No Additional Atmospheric Conditions |
| -- | -- | -- | 1 | -- | -- | Clear/Cloud (No Adverse Conditions) |
| 2 | 2 | -- | -- | 2 | 2 | Rain |
| -- | -- | 2 | 2 | -- | -- | Rain (Mist) |
| 3 | 3 | -- | -- | -- | -- | Sleet |
| -- | -- | 3 | 3 | -- | -- | Sleet (Hail) |
| -- | -- | -- | -- | 3 | -- | Sleet, Hail (Freezing Rain or Drizzle) |
| -- | -- | -- | -- | -- | 3 | Sleet, Hail |
| 4 | 4 | 4 | -- | 4 | 4 | Snow |
| -- | -- | -- | 4 | -- | -- | Snow or Blowing Snow |
| -- | 5 | 5 | -- | -- | -- | Fog |
| -- | -- | -- | 5 | 5 | 5 | Fog, Smog, Smoke |
| -- | -- | 6 | -- | -- | -- | Rain and Fog |
| -- | -- | -- | 6 | 6 | 6 | Severe Crosswinds |
| -- | -- | 7 | -- | -- | -- | Sleet and Fog |
| -- | -- | -- | 7 | 7 | 7 | Blowing Sand, Soil, Dirt |
| -- | 8 | 8 | -- | -- | -- | Other: Smog, Smoke, Blowing Sand or
Dust |
| -- | -- | -- | 8 | 8 | 8 | Other |
| 7 | -- | -- | -- | 10 | 10 | Cloudy |
| -- | -- | -- | -- | 11 | 11 | Blowing Snow |
| -- | -- | -- | -- | -- | 12 | Freezing Rain or Drizzle |
| -- | -- | -- | -- | 98 | 98 | Not Reported |
| 9 | 9 | 9 | 9 | 99 | 99 | Unknown /
Reported as Unknown (<i>Since 2018</i>) |

Figure 9.1:

Item(s) to submit:

- R code used to solve the problem.
- The percentage of accidents during snowy weather.
- The percentage of accidents during rainy weather.

5. According to <https://www.nhtsa.gov/risky-driving/drun-driving>, around 33% of all traffic crash fatalities in the US involve drunk drivers. Jimbob just learned to use `tapply`, and is bound and determined to use it all of the time. He wanted to see if he can confirm a similar number as <https://www.nhtsa.gov> using our `/class/datamine/data/fars/7581.csv` dataset. Examine his code, explain what he is doing wrong, and come up with a *much* simpler solution. Can you confirm the statement from <https://www.nhtsa.gov>?

```
res <- tapply(dat$STATE, list(dat$STATE, dat$DRUNK_DR > 0), length)
mean(res[,2]/(res[,2]+res[,1]))
```

Relevant topics: `tapply`

Item(s) to submit:

- 1-2 sentences explaining what Jimbob is doing wrong.
- The *much* simpler solution to solve the problem.
- Does your solution and result match the findings from <https://www.nhtsa.gov>?

6. Let's put (some of) Jimbob's work to good use, after all, his result, `res` is interesting. Create a data.frame named `myDF` with a column named `state` or `states`, which contains the state names (which you can get from the `row.names` of `res`), and a column named `percent` with the percentage of drunk driving accidents in the associated state. Once complete, generate a map using the code below.

```
library(usmap)
library(ggplot2)
plot_usmap(data = myDF, values = "percent", color = "black") +
  scale_fill_continuous(low = "white", high = "#C28E0E",
                        name = "Drunk driving accidents (%)",
                        label = scales::percent) +
  theme(legend.position = "right")
```

Relevant topics: `data.frame`, `tapply`

Item(s) to submit:

- R code used to solve the problem.
- The resulting plot.

Project 7

Motivation: Three bread-and-butter functions that are a part of the base R are: `subset`, `merge`, and `split`. `subset` provides a more natural way to filter and select data from a `data.frame`. `split` is a useful function that splits a dataset based on one or more factors. `merge` brings the principals of combining data that SQL uses, to R.

Context: We've been getting comfortable working with data in within the R environment. Now we are going to expand our toolset with three useful functions, all the while gaining experience and practice wrangling data!

Scope: `r`, `subset`, `merge`, `split`, `tapply`

Learning objectives:

- Gain proficiency using `split`, `merge`, and `subset`.
- Demonstrate the ability to use the following functions to solve data-driven problem(s): `mean`, `var`, `table`, `cut`, `paste`, `rep`, `seq`, `sort`, `order`, `length`, `unique`, etc.
- Read and write basic (csv) data.
- Explain and demonstrate: positional, named, and logical indexing.
- Demonstrate how to use `tapply` to solve data-driven problems.

Dataset: `/class/datamine/data/goodreads/csv`

1. Load up the following three datasets `goodreads_books.csv`, `goodreads_book_authors.csv`, and `goodreads_interactions.csv` into three `data.frames` `books`, `authors`, and `interactions` respectively. Read in only 1 million rows of the `goodreads_interactions.csv`. How many columns and rows are in each dataset?

Relevant topics: `read.csv`, `dim`

Item(s) to submit:

- R code used to solve the problem.
- The result of running the R code.

2. We want to figure out how book size (`num_pages`) is associated with various metrics. First, let's create a vector called `book_size`, that categorizes books into 4 categories based on `num_pages`: `small` (up to 250 pages), `medium` (250-500 pages), `large` (500-1000 pages), `huge` (1000+ pages).

Relevant topics: `cut`

Item(s) to submit:

- R code used to solve the problem.
- The result of `table(book_size)`.

3. Use `tapply` to calculate the mean `average_rating`, `text_reviews_count`, and `publication_year` by `book_size`. Did any of the result surprise you? Why or why not?

Relevant topics: `tapply`

Item(s) to submit:

- R code used to solve the problem.
- The output from running the R code.

4. Notice in (3) every time we used `tapply` we were re-splitting the data each time. Use `split` to partition the data containing only the following 3 columns: `average_rating`, `text_reviews_count`, and `publication_year`, by `book_size`. Save the result as `books_by_size`. What class is the result? `lapply` is a function that allows you to loop over each item in a list and apply a function. Use `lapply` and `colMeans` to perform the same calculation as in (3).

Relevant topics: `lapply`, `split`, `colMeans`, indexing

Item(s) to submit:

- R code used to solve the problem.
- The copy and pasted output from running the code.

5. We are working with a lot more data than we really want right now. You were provided with the following code to filter out non-English books and only keep columns of interest. Write out the equivalent code using `subset` instead of indexing, and save the result to `res`. Do the dimensions (using `dim`) of the subset version and the version below match? Why or why not?

```
en_books <- books[books$language_code %in% c("en-US", "en-CA", "en-GB", "eng", "en", "en"), ]
```

Hint: If the dimensions don't match, take a look at NA values for the variables used to subset our data.

Relevant topics: indexing, subset, NA, %in%

Item(s) to submit:

- R code used to solve the problem.
- Do the dimensions match?
- 1-2 sentences explaining why or why not.

6. We now have a nice and tidy subset of data, `res`. It would be really nice to get some information on the author (especially the name!). We can find that information in `authors`! In the previous project, we had a similar issue with the states names (in question 2). There is a *much* better way to solve these types of problems. Use the `merge` function to combine `res` and `authors` in a way which appends all information from `author` when there is a match in `res`.

Relevant topics: merge

Item(s) to submit:

- R code used to solve the problem.
- The `dim` of the newly merged data.frame.

7. Look at the `names` of the resulting `data.frame`. Notice that there are two values for `ratings_count` and two values for `average_rating`. The names that have an appended `x` are those values from the first argument to `merge`, and the names that have an appended `y`, are those values from the second argument to `merge`. Rename these columns to indicate if they refer to a book, or an author.

Hint: For example, `ratings_count.x` could be `ratings_count_book` or `ratings_count_author`.

Relevant topics: names

Item(s) to submit:

- R code used to solve the problem.
- The `names` of the new `data.frame`.

8. For an author of your choice (that *is* in the dataset), find the author's highest rated book. Do you agree?

Relevant topics: indexing, subset, which, max

Item(s) to submit:

- R code used to solve the problem.
- The title of the highest rated book (from your author).
- 1-2 sentences explaining why or why not you agree with it being the highest rated book from that author.

Project 8

Motivation: A key component to writing efficient code is writing functions. Functions allow us to repeat and reuse coding steps that we used previously, over and over again. If you find you are repeating code over and over, a function may be a good way to reduce lots of lines of code!

Context: We've been learning about and using functions all year! Now we are going to learn more about some of the terminology and components of a function, as you will certainly need to be able to write your own functions soon.

Scope: r, functions

Learning objectives:

- Gain proficiency using split, merge, and subset.
- Demonstrate the ability to use the following functions to solve data-driven problem(s): mean, var, table, cut, paste, rep, seq, sort, order, length, unique, etc.
- Read and write basic (csv) data.
- Explain and demonstrate: positional, named, and logical indexing.
- Demonstrate how to use tapply to solve data-driven problems.
- Comprehend what a function is, and the components of a function in R.

Dataset: /class/datamine/data/goodreads/csv

1. Read in the same data, in the same way as the previous project (with the same names). We've provided you with the function below. How many arguments does the function have? Name all of the arguments. What is the name of the function? Replace the description column in our books data.frame with the same information, but with stripped punctuation using the function provided.

```
# A function that, given a string (description), returns the string
# without any punctuation.
strip_punctuation <- function(description) {
  # Use regular expressions to identify punctuation.
  # Replace identified punctuation with an empty string ''.
  desc_no_punc <- gsub('[:punct:]]+', '', description)
  # Return the result
  return(desc_no_punc)
}
```

Hint: Since gsub accepts a vector of values, you can pass an entire vector to strip_punctuation.

2. Now its time to write your own function. We want to write a function that counts the words in a string. There are already functions that do this, however, we want to write our own. We plan to use this on our non-punctuated descriptions. Begin by using the strsplit function to split a string by spaces. An examples string is: test_string <- "This is a test string with no punctuation". Use test_string to test out your code. If you counted the words shown in your results, would it be an accurate count? Why or why not?

Relevant topics: strsplit

Item(s) to submit:

- R code used to solve the problem.
- 1-2 sentences explaining why or why not your count would be accurate.

3. Fix the issue in (3), using `which`. You may need to unlist the `strsplit` result first. After you've accomplished this, you can count the remaining words!

Relevant topics: `which`

Item(s) to submit:

- R code used to solve the problem (including counting the words).

4. We are finally to the point where we have code from questions (2) and (3) that we think we may want to use many times. Write a function called `count_words` which, given a string, `description`, returns the number of words in `description`. Test out `count_words` on the `description` from the second row of `books`. How many words are in the `description`?

Relevant topics: functions, `unlist`, indexing, `strsplit`

Item(s) to submit:

- R code used to solve the problem.
- The result of using the function on the `description` from the second row of `books`.

5. Practice makes perfect! Write a function of your own design that is intended on being used with one of our datasets. Test it out and share the results.

Note: You could even pass (as an argument) one of our datasets to your function and calculate a cool statistic or something like that! Maybe your function makes a plot? Who knows?

Relevant topics: functions

Item(s) to submit:

- R code used to solve the problem.
- An example (with output) of using your newly created function.

Project 9

Motivation: A key component to writing efficient code is writing functions. Functions allow us to repeat and reuse coding steps that we used previously, over and over again. If you find you are repeating code over and over, a function may be a good way to reduce lots of lines of code!

Context: We've been learning about and using functions all year! Now we are going to learn more about some of the terminology and components of a function, as you will certainly need to be able to write your own functions soon.

Scope: r, functions

Learning objectives:

- Gain proficiency using split, merge, and subset.
- Demonstrate the ability to use the following functions to solve data-driven problem(s): mean, var, table, cut, paste, rep, seq, sort, order, length, unique, etc.
- Read and write basic (csv) data.
- Explain and demonstrate: positional, named, and logical indexing.
- Demonstrate how to use tapply to solve data-driven problems.
- Comprehend what a function is, and the components of a function in R.

Dataset: /class/datamine/data/goodreads/csv

1. We've provided you with a function below. How many arguments does the function have, and what are their names? You can get a book_id from the URL of a goodreads book's webpage. For example, the book_id from <https://www.goodreads.com/book/show/17332218-words-of-radiance#>, is 17332218. Another example is https://www.goodreads.com/book/show/157993.The_Little_Prince?from__search=true&from__srp=true&qid=JJGqUK9Vp9&rank=1, with a book_id of 157993. Find 2 or 3 book_ids and test out the

function until you get a success or two. Explain in words, what the function is doing, and what options you have.

```
books <- read.csv("/class/datamine/data/goodreads/csv/goodreads_books.csv")
authors <- read.csv("/class/datamine/data/goodreads/csv/goodreads_book_authors.csv")
fun_plot <- function(book_id, display_cover=T) {
  library(imager)
  get_author_name <- function(author_id){
    return(authors[authors$author_id==author_id,'name'])
  }

  book_info <- books[books$book_id==book_id,]
  all_books_by_author <- books[books$author_id==book_info$author_id,]
  author_name <- get_author_name(book_info$author_id)

  img_url <- book_info$image_url
  img <- load.image(img_url)

  if(display_cover){
    par(mfrow=c(1,2))
    plot(img, axes=FALSE)
  }
  plot(all_books_by_author$num_pages, all_books_by_author$average_rating, ylim=c(0,5.1),
       pch=21, bg='grey80',
       xlab='Number of pages', ylab='Average rating', main=paste('Books by', author_name))
  points(book_info$num_pages, book_info$average_rating,pch=21, bg='orange', cex=1.5)
}
```

Relevant topics: functions

Item(s) to submit:

- How many arguments does the function have, and what are their names?
- The result of using the function on 2-3 book_ids.
- 1-2 sentences explaining what the function does (generally), and what (if any) options the function provides you with.

2. You may have encountered a situation where the book_id was not in our dataset, and hence, didn't get plotted. When writing functions, it is usually best to try and foresee issues like this and have the function fail gracefully, instead of showing some ugly (and sometimes unclear) warning. Add a check at the beginning of our function that checks

for the `book_id`, and if it does not exist, prints “Book ID not found.”, and exits the function. Test it out on `book_id=123`.

Hint: Run `?stop` to see if that is a function that may be useful.

Relevant topics: functions, if/else, stop

Item(s) to submit:

- R code with your new and improved function.
- The results from `fun_plot(123)`.
- The results from `fun_plot(19063)`.

3. You may have noticed a function *inside* our `fun_plot` function. It looks like it accepts an `author_id` and returns the name of the author. Try running `get_author_name(6252)`, does it work? Read this and explain in 1-2 sentences what you think is happening.

Relevant topics: scoping

Item(s) to submit:

- The results from `get_author_name(6252)`.
- 1-2 sentences explaining what is happening.

4. Our `fun_plot` requires that the datasets `books` and `authors` have been loaded exactly right (and with the correct names) in the environment. By including objects outside of our function’s scope, within our function (in this case `books` and `authors`) it leaves our `fun_plot` function prone to errors, as any changes to those objects may break our function. Fix this by making the datasets (`books` and `authors`) arguments in the function, and modifying our function accordingly to run based on those arguments.

Relevant topics: functions, `read.csv`, scoping

Item(s) to submit:

- R code with your new and improved function.
- An example using the updated function.

5. Write your own custom function. Make sure your function includes at least 2 arguments. If you access one of our datasets from

within your function (which you *definitely* should do), use what you learned in (4), to avoid future errors dealing with scoping. Your function could output a cool plot, interesting tidbits of information, or anything else you can think of. Get creative and make a function that is fun to use!

Relevant topics: scoping, functions

Item(s) to submit:

- R code used to solve the problem.
- Examples using your function with included output.

Project 10

Motivation: Functions are powerful. They are building blocks to more complex programs and behavior. In fact, there is an entire programming paradigm based on functions called functional programming. In this project, we will learn to *apply* functions to entire vectors of data using **sapply**.

Context: We’ve just taken some time to learn about and create functions. One of the more common “next steps” after creating a function is to use it on a series of data, like a vector. **sapply** is one of the best ways to do this in R.

Scope: r, sapply, functions

Learning objectives:

- Read and write basic (csv) data.
- Explain and demonstrate: positional, named, and logical indexing.
- Utilize apply functions in order to solve a data-driven problem.
- Gain proficiency using split, merge, and subset.

Dataset: /class/datamine/data/okcupid/filtered

1. Load up the the following datasets into data.frames named **users** and **questions**, respectively: /class/datamine/data/okcupid/filtered/users.csv, /class/datamine/okcupid/filtered/questions.csv. This is data from users on OkCupid, on online dating app. In your own words, explain what each file contains and how they are related – its *always* a good

idea to poke around the data to get a better understanding of how things are structured!

Hint: Be careful, just because a file ends in `.csv`, does *not* mean it is comma-separated.

Relevant topics: `read.csv`

Item(s) to submit:

- R code used to solve the problem.
- 1-2 sentences describing what each file contains and how they are related.

2. `grep` is an incredibly powerful tool available to us in R. We will learn more about `grep` in the future, but for now, know that a simple application of `grep` is to find a word in a string. In R, `grep` is vectorized and can be applied to an entire vector of strings. Use `grep` to find a question that references “google”. What is the question?

Hint: If at first you don’t succeed, run `?grep` and check out the `ignore.case` argument.

Relevant topics: `grep`

Item(s) to submit:

- R code used to solve the problem.
- The text of the question that references Google.

3. In (2) we found a pretty interesting question. What is the percentage of users that Google someone before the first date? Does the proportion change by gender (as defined by `gender2`)? How about by `gender_orientation`?

Hint: If you look at the question column for this question, you should notice that this column is a **factor** with two possible answers: “No. Why spoil the mystery?” and “Yes. Knowledge is power!”. If you start by creating a function that calculates the percentage of people who answer each question, you could use `tapply` in combination with this function to break the answer down by gender.

Relevant topics: `functions`, `tapply`, `table`, `prop.table`

Item(s) to submit:

- R code used to solve this problem.
- The results of running the code.
- Written answers to the questions.

4. In project (8) we created a function called `count_words`. Use this function and `apply` to create a vector with the length (in words) of the questions. Call the new column of data `question_length`, and add the column to our `data.frame`.

```
count_words <- function(description) {
  split_desc <- unlist(strsplit(description, " "))
  return(length(split_desc[which(split_desc != "")]))
}
```

Hint: `questions$text` is a factor. Use `as.character` to convert the factor to a character before passing it to `count_words`.

Relevant topics: `apply`

Item(s) to submit:

- R code used to solve this problem.
- The result of `str(questions)`.

5. Write a function called `number_of_options` that accepts the dataset, and a question key (for example `q484`) and counts the number of answer options that the question has. Although each question has 4 option columns, not every column is filled. Consider an option empty if it is NA or blank. What percentage of questions have 1, 2, 3, and 4 options? Add this data to a new column in our `questions` dataset called `number_options`.

Hint: Use `apply` to apply your function to every id in the vector (`questions$X`).

Hint: The way `apply` works is the the first argument is by default the first argument to your function, the second argument is the function you want applied, and after that you can specify arguments by name.

Relevant topics: `table`, `prop.table`, `apply`, functions, `if/else`, indexing, `is.na`

Item(s) to submit:

- R code used to solve this problem.
- The results of the running the code.

6. Does it appear that there is an association between the length of the question and whether or not users answered the question? Assume NA means “unanswered”. First create a function called `percent_answered` that, given a vector, returns the percentage of values that are not NA. Use `percent_answered` and `sapply` to calculate the percentage of users who answer each question. Plot this result, against the length of the questions.

Hint: `length_of_questions <- questions$question_length[grepl("^q", questions$X)]`

Hint: Use the same trick we used in the previous hint, to subset our `users` data.frame before using `sapply` to apply `percent_answered`. `grepl("^q", questions$X)` returns the column index of every column that starts with “q”.

Relevant topics: `sapply`, `is.na`, `length`, `grepl`, `plot`

Item(s) to submit:

- R code used to solve this problem.
- The plot.
- Whether or not you think there may or may not be an association between question length and whether or not the question is answered.

7. *Lots* of questions are asked in this dataset. Explore the dataset, and either calculate an interesting statistic/result using `sapply`, or generate a graphic (with good x-axis and/or y-axis labels, main labels, legends, etc.), or both! Write 1-2 sentences about your analysis and/or graphic, and explain what you thought you’d find, and what you actually discovered.

Relevant topics: plotting, functions, `sapply`

Item(s) to submit:

- R code used to solve this problem.
- The results from running your code.
- 1-2 sentences about your analysis and/or graphic, and explain what you thought you'd find, and what you actually discovered.

Project 11 {#190-p11}

Project 12 {#190-p12}

Project 13 {#190-p13}

Project 14 {#190-p14}

Project 15 {#190-p15}

STAT 29000

Project 1 {#290-p01}

Motivation: In this project we will jump right into an R review. In this project we are going to break one larger data-wrangling problem into discrete parts. There is a slight emphasis on writing functions and dealing with strings. At the end of this project we will have greatly simplified a dataset, making it easy to dig into.

Context: We just started the semester and are digging into a large dataset, and in doing so, reviewing R concepts we've previously learned.

Scope: data wrangling in R, functions

Learning objectives:

- Comprehend what a function is, and the components of a function in R.
- Read and write basic (csv) data.
- Utilize apply functions in order to solve a data-driven problem.

Dataset:

`/class/datamine/data/airbnb`

Often times (maybe even the majority of the time) data doesn't come in one nice file or database. Explore the dataset in `/class/datamine/data/airbnb`.

1. You may have noted that, for each country, city, and date we can find 3 files: `calendar.csv.gz`, `listings.csv.gz`, and `reviews.csv.gz` (for now, we will ignore all files in the “visualisations” folders). Let's take a look at the data in each of the three types of files. Pick a country, city and date, and read the first 50 rows of each of the 3 datasets. Provide 1-2 sentences explaining the type of information found in each, and what variable(s) could be used to join them.

Hint: `read.csv` has an argument to select the number of rows we want to read.

Item(s) to submit:

- Chunk of code used to read the first 50 rows of each dataset.
- 1-2 sentences briefly describing the information contained in each dataset.
- Name(s) of variable(s) that could be used to join them.

To read a compressed csv, simply use the `read.csv` function:

```
dat <- read.csv("/class/datamine/data/airbnb/brazil/rj/rio-de-janeiro/2019-06-19/data/
head(dat)
```

Let's work towards getting this data into an easier format to analyze. From now on, we will focus on the `listings.csv.gz` datasets.

2. Write a function called `get_paths_for_country`, that, given a string with the country name, returns a vector with the full paths for all `listings.csv.gz` files, starting with `/class/datamine/data/airbnb/....`

Some example output from `get_paths_for_country("united-states")`:

```
[1] "/class/datamine/data/airbnb/united-states/ca/los-angeles/2019-07-08/data/listings.csv.gz"
[2] "/class/datamine/data/airbnb/united-states/ca/oakland/2019-07-13/data/listings.csv.gz"
[3] "/class/datamine/data/airbnb/united-states/ca/pacific-grove/2019-07-01/data/listings.csv.gz"
[4] "/class/datamine/data/airbnb/united-states/ca/san-diego/2019-07-14/data/listings.csv.gz"
[5] "/class/datamine/data/airbnb/united-states/ca/san-francisco/2019-07-08/data/listings.csv.gz"
```

Hint: `list.files` is useful with the `recursive=T` option.

Hint: To exclude “visualisations” folders, try: `grep("visualisations", ..., invert=T)`.

Item(s) to submit:

- Chunk of code for your `get_paths_for_country` function.

3. Write a function called `get_data_for_country` that, given a string with the country name, returns a data.frames containing the all listings data for that country. Use your previously written function to help you.

Hint: Use `stringAsFactors=F` in the `read.csv` function.

Hint: Use `do.call(rbind, <listofdataframes>)` to combine a list of dataframes into a single dataframe.

Relevant topics: `rbind`, `lapply`, function

Item(s) to submit:

- Chunk of code for your `get_data_for_country` function.

4. Use your `get_data_for_country` to get the data for a country of your choice, and make sure to name the dataframe `listings`. Take a look at the following columns: `host_is_superhost`, `host_has_profile_pic`, `host_identity_verified`, and `is_location_exact`. What is the data type for each column? Is there a more appropriate type for them? If so, which type would you recommend?

Hint: Remember, there are six types of vectors: logical, integer, double, character, complex, and raw. To see the vector data types of you can use `typeof` or `str`. The function `typeof` will return the type, while `str` will return more information. See some examples below:

```
# Using typeof
typeof(letters)
```

```
## [1] "character"
```

```
typeof(1:10)
```

```
## [1] "integer"
```

```
# Using str
str(letters)
```

```
## chr [1:26] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" ...
```

```
str(1:10)
```

```
## int [1:10] 1 2 3 4 5 6 7 8 9 10
```

5. Write a function called `transform_column` that, given a column similar to the ones in (4) (more or less, lowercase “t”s and “f”s) transforms it to your suggested vector type in (4). Note that NA values for these columns appear as blank (“”), and we need to be careful when transforming the data. Test your function on column `host_is_superhost`.

Relevant topics: `toupper`, `as.logical`

Item(s) to submit:

- Chunk of code for your `transform_column` function.
- Type of `transform_column(listings$host_is_superhost)`.

6. Before we can use your function, we need to determine which columns are similar to `host_is_superhost` (i.e., lowercase “t”s and “f”s) and need transformation. Create a function named `should_be_transformed` that, given a column, determines (returns TRUE or FALSE) if it contains only “t”, “f”, and “” values. Use your newly created function to create a vector named `columns_to_transform` which contains all columns we want to transform using the function in (5). How many columns are in this format?

Relevant topics: `unique`, `%in%`, `any`, `sapply`, `which`

Item(s) to submit:

- Chunk of code for your `should_be_transformed` function.
- Chunk of code used to obtain `columns_to_transform`.

7. Apply your function `transform_column` to all columns in `columns_to_transform` in your `listings` data. Make sure it worked by checking the type of columns `id` and `instant_bookable`. Note that the column `id` should have the same type as before.

Relevant topics: apply

Item(s) to submit:

- Chunk of code to get your new `listings` data.
- Type of columns `id` and `instant_bookable`.

8. Now that we have organized and cleaned our data, let's explore it! Based on your `listings` data, if you are looking at an instant bookable listing (where `instant_bookable` is `TRUE`), would you expect the location to be exact (where `is_exact_location` is `TRUE`)? Why or why not?

Hint: Make a frequency table, and see how many instant bookable listings have exact location.

Relevant topics: table

Item(s) to submit:

- Chunk of code to get a frequency table.
- 1-2 sentences explaining whether or not we would expect the location to be exact if we were looking at a instant bookable listing.

Project 2 {#290-p02}

Motivation: The ability to quickly reproduce an analysis is important. It is often necessary that other individuals will need to be able to understand and reproduce an analysis. This concept is so important there are classes solely on reproducible research! In fact, there are papers that investigate and highlight the lack of reproducibility in various fields. If you are interested in reading about this topic, a good place to start is the paper titled “Why Most Published Research Findings Are False”, by John Ioannidis (2005).

Context: Making your work reproducible is extremely important. We will focus on the computational part of reproducibility. We will learn RMarkdown

to document your analyses so others can easily understand and reproduce the computations that led to your conclusions. Pay close attention as future project templates will be RMarkdown templates.

Scope: Understand Markdown, RMarkdown, and how to use it to make your data analysis reproducible.

Learning objectives:

- Use Markdown syntax within an Rmarkdown document to achieve various text transformations.
- Use RMarkdown code chunks to display and/or run snippets of code.

You can find useful examples that walk you through relevant material in The Examples Book:

<https://thedatamine.github.io/the-examples-book>

It is highly recommended to read through, search, and explore these examples to help solve problems in this project.

Important note: It is highly recommended that you use <https://rstudio.scholar.rcac.purdue.edu/>. Simply click on the link and login using your Purdue account credentials. Use another system at your own risk. The version of RStudio on <https://desktop.scholar.rcac.purdue.edu/> (which uses ThinLinc), is 99.9.9, and is known to have some strange issues when running code chunks.

Don't forget the very useful documentation shortcut `?`. To use, simply type `?` in the console, followed by the name of the function you are interested in.

You can also look for package documentation by using `help(package=PACKAGENAME)`, so for example, to see the documentation for the package `ggplot2`, we could run:

```
help(package=ggplot2)
```

Sometimes it can be helpful to see the source code of a defined function. A function is any chunk of organized code that is used to perform an operation. Source code is the underlying R or C or C++ code that is used to create the function. To see the source code of a defined function, type the function's name without the `()`. For example, if we were curious about what the function `Reduce` does, we could run:

```
Reduce
```

Occasionally this will be less useful as the resulting code will be code that calls C code we can't see. Other times it will allow you to understand the function better.

1. Make the following text (including the asterisks) bold: This needs to be **very** bold. Make the following text (including the underscores) italicized: This needs to be *_very_* italicized.

Hint: Try mixing and matching ways to embolden or italicize text. Alternatively, look up “escaping characters in markdown”, or see [here](#).

Hint: Be sure to check out the *Rmarkdown Cheatsheet* and our section on *Rmarkdown* in the book.

Note: *Rmarkdown* is essentially Markdown + the ability to run and display code chunks. In this question, we are actually using Markdown within Rmarkdown!

Relevant topics: *rmarkdown*, *escaping characters*

Item(s) to submit:

- Fill in the chunk under (1) in the `stat29000project02template.Rmd` file with 2 lines of markdown text. Note that when compiled, this text will be unmodified, regular text.

2. Create an unordered list of your top 3 favorite academic interests (some examples could include: machine learning, operating systems, forensic accounting, etc.). Create another *ordered* list that ranks your academic interests in order of most interested to least interested.

Hint: You can learn what ordered and unordered lists are [here](#).

Note: Similar to (1a), in this question we are dealing with Markdown. If we were to copy and paste the solution to this problem in a Markdown editor, it would be the same result as when we Knit it [here](#).

Relevant topics: *rmarkdown*

Item(s) to submit:

- Create the lists under (2) in the `stat29000project02template.Rmd` file. Note that when compiled, this text will appear as nice, formatted lists.

3. Browse <https://www.linkedin.com/> and read some profiles. Pay special attention to accounts with an “About” section. Write your own personal “About” section using Markdown. Include the following:

- A header (your choice of size) that says “About”.

- A horizontal rule directly underlining the header.
- The text of your personal “About” section that you would feel comfortable uploading to linkedin, including at least 1 link.

Relevant topics: *rmarkdown*

Item(s) to submit:

- Create the described profile under (3) in the `stat29000project02template.Rmd` file.

4. Your co-worker wrote a report, and has asked you to beautify it. Knowing Rmarkdown, you agreed. Spruce up the report found under (4) in `stat29000project02template.Rmd`. At a minimum:

- Make the title pronounced.
- Make all links appear as a word or words, rather than the long-form URL.
- Organize all code into code chunks where code and output are displayed. If the output is really long, just display the code.
- Make the calls to the `library` function and the `install.packages` function be evaluated but not displayed. Make sure all warnings and errors that may eventually occur, do not appear in the final document.

Feel free to make any other changes that make the report more visually pleasing.

Relevant topics: *rmarkdown*

Item(s) to submit:

- Spruce up the “document” under (4) in the `stat29000project02template.Rmd` file.

5. Create a plot using a built-in dataset like `iris`, `mtcars`, or `Titanic`, and display the plot using a code chunk. Make sure the code used to generate the plot is hidden. Include a descriptive caption for the image.

Relevant topics: *rmarkdown*, *plotting in r*

Item(s) to submit:

- Code chunk under (5) that creates and displays a plot using a built-in dataset like `iris`, `mtcars`, or `Titanic`.

6. Insert the following code chunk under (5) in `stat29000project02template.Rmd`. Try knitting the document. Something should go wrong. Fix the problem and knit again. If another problem appears, fix it. What was the first problem? What was the second problem?

```
```{r install-packages}
plot(my_variable)
```
```

Hint: Take a close look at the name we give our code chunk.

Hint: Take a look at the code chunk where `my_variable` is declared.

Relevant topics: *rmarkdown*

Item(s) to submit:

- The modified version of the inserted code that fixes both problems.
- A sentence explaining what the first problem was.
- A sentence explaining what the second problem was.

Project 3 {#290-p03}

Motivation: The ability to navigate a shell, like `bash`, and use some of its powerful tools, is very useful. The number of disciplines utilizing data in new ways is ever-growing, and as such, it is very likely that many of you will eventually encounter a scenario where knowing your way around a terminal will be useful. We want to expose you to some of the most useful `bash` tools, help you navigate a filesystem, and even run `bash` tools from within an RMarkdown file in RStudio.

Context: At this point in time, you will each have varying levels of familiarity with Scholar. In this project we will learn how to use the terminal to navigate a UNIX-like system, experiment with various useful commands, and learn how to execute `bash` commands from within RStudio in an RMarkdown file.

Scope: `bash`, RStudio

Learning objectives:

- Distinguish differences in `/home`, `/scratch`, and `/class`.
- Navigating UNIX via a terminal: `ls`, `pwd`, `cd`, `.`, `..`, `~`, etc.
- Analyzing file in a UNIX filesystem: `wc`, `du`, `cat`, `head`, `tail`, etc.
- Creating and destroying files and folder in UNIX: `scp`, `rm`, `touch`, `cp`, `mv`, `mkdir`, `rmdir`, etc.
- Utilize other Scholar resources: `rstudio.scholar.rcac.purdue.edu`, `notebook.scholar.rcac.purdue.edu`, `desktop.scholar.rcac.purdue.edu`, etc.
- Use `man` to read and learn about UNIX utilities.
- Run `bash` commands from within and RMarkdown file in RStudio.

Dataset: ??

Public: ??

There are a variety of ways to connect to Scholar. In this class, we will *primarily* connect to RStudio Server by opening a browser and navigating to `https://rstudio.scholar.rcac.purdue.edu/`, entering credentials, and using the excellent RStudio interface.

1. Navigate to `https://rstudio.scholar.rcac.purdue.edu/` and login. Take some time to click around and explore this tool. We will be writing and running Python, R, SQL, and `bash` all from within this interface. Navigate to Tools > Global Options Explore this interface and make at least 2 modifications. List what you changed.

Here are some changes Kevin likes:

- Uncheck “Restore .Rdata into workspace at startup”.
- Change tab width 4.
- Check “Soft-wrap R source files”.
- Check “Highlight selected line”.
- Check “Strip trailing horizontal whitespace when saving”.
- Uncheck “Show margin”.

Item(s) to submit:

- List of modifications you made to your Global Options.

2. There are four primary panes, each with various tabs. In one of the panes there will be a tab labeled “Terminal”. Click on that tab. This terminal by default will run a bash shell right within Scholar, the same as if you connected to Scholar using ThinLinc, and opened a terminal. Very convenient! What is the default directory of your bash shell? In our list of relevant topics, we’ve included links to a variety of UNIX commands that may help you solve this problem. Some of the tools are super simple to use, and some are a little bit more difficult.

Hint: Start by reading the section on `man`. `man` stands for manual, and you can find the “official” documentation for the command by typing `man <command_of_interest>`. For example:

```
# read the manual for the 'man' command
# use "k" or the up arrow to scroll up, "j" or the down arrow to scroll down
man man
```

Relevant topics: `man`, `cd`, `pwd`, `ls`, `~`, `..`, `.`

Item(s) to submit:

- The full filepath of default directory (home directory). Ex: Kevin’s is: `/home/kamstut`
- The `bash` code used to show your home directory or current working directory when the `bash` shell is first launched.

3. Learning to navigate away from our home directory to other folders, and back again, is vital. Perform the following actions, in order:

- Write a single command to navigate to the folder containing our full datasets: `/class/datamine/data`.
- Write a command to confirm you are in the correct folder.
- Write a command to list the files and directories within the data directory.
- What are the names of the files? Write another command to return back to your home directory.
- Write a command to confirm you are in the correct folder.

Note: `/` is commonly referred to as the root directory in a linux/unix filesystem. Think of it as a folder that contains *every* other folder in the computer. `/home` is a folder within the root directory. `/home/kamstut` is the full filepath of Kevin’s home directory. There is a folder `home` inside the root directory. Inside `home` is another folder named `kamstut` which is Kevin’s home directory.

Relevant topics: man, cd, pwd, ls, ~, .., .

Item(s) to submit:

- Command used to navigate to the data directory.
- Command used to confirm you are in the data directory.
- Command used to list files and folders.
- List of files and folders in the data directory.
- Command used to navigate back to the home directory.
- Command used to confirm you are in the home directory.

4. Let's learn about two more important concepts. `.` refers to the current working directory, or the directory displayed when you run `pwd`. Unlike `pwd` you can use this when navigating the filesystem! So, for example, if you wanted to see the contents of a file called `my_file.txt` that lives in `/home/kamstut` (so, a full path of `/home/kamstut/my_file.txt`), and you are currently in `/home/kamstut`, you could run: `cat ./my_file.txt`. `..` represents the parent folder or the folder in which your current folder is contained. So let's say I was in `/home/kamstut/projects/` and I wanted to get the contents of the file `/home/kamstut/my_file.txt`. You could do: `cat ../my_file.txt`. When you navigate a directory tree using `.`, `..`, and `~` you create paths that are called *relative* paths because they are *relative* to your current directory. Alternatively, a *full* path or (*absolute* path) is the path starting from the root directory. So `/home/kamstut/my_file.txt` is the *absolute* path for `my_file.txt` and `../my_file.txt` is a *relative* path. Perform the following actions, in order:

- Write a single command to navigate to the data directory.
- Write a single command to navigate back to your home directory using a *relative* path. Do not use `~` or plain `cd`.

Relevant topics: man, cd, pwd, ls, ~, .., .

Item(s) to submit:

- Command used to navigate to the data directory.
- Command used to navigate back to your home directory that uses a *relative* path.

5. In Scholar, when you want to deal with *really* large amounts of data, you want to access scratch (you can read more here). Your

scratch directory on Scholar is located here: `/scratch/scholar/$USER`. `$USER` is an environment variable containing your username. Test it out: `echo /scratch/scholar/$USER`. Perform the following actions:

- Navigate to your scratch directory.
- Confirm you are in the correct location.
- Execute `myquota`.
- Find the location of the `myquota` bash script.
- Output the first 5 and last 5 lines of the bash script.
- Count the number of lines in the bash script.
- How many kilobytes is the script?

Hint: You could use each of the commands in the relevant topics once.

Hint: Commands often have *options*. *Options* are features of the program that you can trigger specifically. You can see the *options* of a command in the DESCRIPTION section of the `man` pages. For example: `man wc`. You can see `-m`, `-l`, and `-w` are all options for `wc`. To test this out:

```
# using the default wc command. "/class/datamine/data/flights/1987.csv" is the first "argument" g
wc /class/datamine/data/flights/1987.csv
# to count the lines, use the -l option
wc -l /class/datamine/data/flights/1987.csv
# to count the words, use the -w option
wc -w /class/datamine/data/flights/1987.csv
# you can combine options as well
wc -w -l /class/datamine/data/flights/1987.csv
# some people like to use a single tack '-'
wc -wl /class/datamine/data/flights/1987.csv
# order doesn't matter
wc -lw /class/datamine/data/flights/1987.csv
```

Hint: The `-h` option for the `du` command is useful.

Relevant topics: `cd`, `pwd`, `type`, `head`, `tail`, `wc`, `du`

Item(s) to submit:

- Command used to navigate to your scratch directory.
- Command used to confirm your location.
- Output of `myquota`.
- Command used to find the location of the `myquota` script.
- Absolute path of the `myquota` script.
- Command used to output the first 5 lines of the `myquota` script.
- Command used to output the last 5 lines of the `myquota` script.
- Command used to find the number of lines in the `myquota` script.
- Number of lines in the script.
- Command used to find out how many kilobytes the script is.
- Number of kilobytes that the script takes up.

6. Perform the following operations:

- Navigate to your scratch directory.
- Copy and paste the file: `/class/datamine/data/flights/1987.csv` to your current directory (scratch).
- Create a new directory called `my_test_dir` in your scratch folder.
- Move the file you copied to your scratch directory, into your new folder.
- Use `touch` to create an empty file named `im_empty.txt` in your scratch folder.
- Remove the directory `my_test_dir` *and* the contents of the directory.
- Remove the `im_empty.txt` file.

Hint: `rmdir` may not be able to do what you think, instead, check out the options for `rm` using `man rm`.

Relevant topics: `cd`, `cp`, `mv`, `mkdir`, `touch`, `rmdir`, `rm`

Item(s) to submit:

- Command used to navigate to your scratch directory.
- Command used to copy the file, `/class/datamine/data/flights/1987.csv` to your current directory (scratch).
- Command used to create a new directory called `my_test_dir` in your scratch folder.
- Command used to move the file you copied earlier `1987.csv` into your new `my_test_dir` folder.
- Command used to create an empty file named `im_empty.txt` in your scratch folder.
- Command used to remove the directory *and* the contents of the directory `my_test_dir`.
- Command used to remove the `im_empty.txt` file.

Project 4 {#290-p04}

Motivation: The need to search files and datasets based on the text held within is common during various parts of the data wrangling process. `grep` is an extremely powerful UNIX tool that allows you to do so using regular expressions. Regular expressions are a structured method for searching for specified patterns. Regular expressions can be very complicated, even professionals can make critical mistakes. With that being said, learning some of the basics is an incredible tool that will come in handy regardless of the language you are working in.

Context: We've just begun to learn the basics of navigating a file system in UNIX using various terminal commands. Now we will go into more depth with one of the most useful command line tools, `grep`, and experiment with regular expressions using `grep`, R, and later on, Python.

Scope: `grep`, regular expression basics, utilizing regular expression tools in R and Python

Learning objectives:

- Use `grep` to search for patterns within a dataset.
- Use `cut` to section off and slice up data from the command line.
- Use `wc` to count the number of lines of input.

You can find useful examples that walk you through relevant material in The Examples Book:

<https://thedatamine.github.io/the-examples-book>

It is highly recommended to read through, search, and explore these examples to help solve problems in this project.

Important note: It is highly recommended that you use <https://rstudio.scholar.rcac.purdue.edu/>. Simply click on the link and login using your Purdue account credentials. Use another system at your own risk. The version of RStudio on <https://desktop.scholar.rcac.purdue.edu/> (which uses ThinLinc), is 99.9.9, and is known to have some strange issues when running code chunks.

Don't forget the very useful documentation shortcut `?`. To use, simply type `?` in the console, followed by the name of the function you are interested in.

You can also look for package documentation by using `help(package=PACKAGENAME)`, so for example, to see the documentation for the package `ggplot2`, we could run:

```
help(package=ggplot2)
```

Sometimes it can be helpful to see the source code of a defined function. A function is any chunk of organized code that is used to perform an operation. Source code is the underlying R or C or C++ code that is used to create the function. To see the source code of a defined function, type the function's name without the `()`. For example, if we were curious about what the function `Reduce` does, we could run:

```
Reduce
```

Occasionally this will be less useful as the resulting code will be code that calls C code we can't see. Other times it will allow you to understand the function better.

Dataset

The following questions will use the dataset found in Scholar:

`/class/datamine/data/movies_and_tv/the_office_dialogue.csv`

A public sample of the data can be found here: `the_office_dialogue.csv`

Answers to questions should all be answered using the full dataset located on Scholar. You may use the public samples of data to experiment with your solutions prior to running them using the full dataset.

`grep` stands for (g)lobally search for a (r)egular (e)xpression and (p)rint matching lines. As such, to best demonstrate `grep`, we will be using it with textual data. You can read about and see examples of `grep` [here](#).

1. Login to Scholar and use `grep` to find the dataset we will use this project. The dataset we will use is the only dataset to have the text “bears. beets. battlestar galactica.”. What is the name of the dataset and where is it located?

Relevant topics: *grep*

Item(s) to submit:

- The `grep` command used to find the dataset.
- The name and location in Scholar of the dataset.
- Use `grep` and `grep1` within R to solve a data-driven problem.

2. `grep` prints the line that the text you are searching for appears in. In project 3 we learned a UNIX command to quickly print the first n lines from a file. Use this command to get the headers for the dataset. As you can see, each line in the tv show is a row in the dataset. You can count to see which column the various bits of data live in.

Write a line of UNIX commands that searches for “bears. beets. battlestar galactica.” and, rather than printing the entire line, prints only the character who speaks the line, as well as the line itself.

Hint: *The result if you were to search for “bears. beets. battlestar galactica.” should be:*

```
"Jim","Fact. Bears eat beets. Bears. Beets. Battlestar Galactica."
```

Hint: *One method to solve this problem would be to pipe the output from `grep` to `cut`.*

Relevant topics: *cut, grep*

Item(s) to submit:

- The line of UNIX commands used to perform the operation.

3. This particular dataset happens to be very small. You could imagine a scenario where the file is many gigabytes and not easy to load completely into R or Python. We are interested in learning what makes Jim and Pam tick as a couple. Use a line of UNIX commands to create a new dataset called `jim_and_pam.csv`. Include only lines that are spoken by either Jim or Pam, or reference Jim or Pam in

any way. Include only the following columns: `episode_name`, `character`, `text`, `text_w_direction`, and `air_date`. How many rows of data are in the new file? How many megabytes is the new file (to the nearest 1/10th of a megabyte)?

Hint: *Redirection.*

Hint: *It is OK if you get an erroneous line where the word “jim” or “pam” appears as a part of another word.*

Relevant topics: *cut, grep, ls, wc, redirection*

Item(s) to submit:

- The line of UNIX commands used to create the new file.
- The number of rows of data in the new file, and the accompanying UNIX command used to find this out.
- The number of megabytes (to the nearest 1/10th of a megabyte) that the new file has, and the accompanying UNIX command used to find this out.

4. Find all lines where either Jim/Pam/Michael/Dwight’s name is followed by an exclamation mark. Use only 1 “!” within your regular expression. How many lines are there?

Relevant topics: *grep*

Item(s) to submit:

- The UNIX command(s) used to solve this problem.
- The number of lines where either Jim/Pam/Michael/Dwight’s name is followed by an exclamation mark.

5. Find all lines that contain the text “that’s what” followed by any amount of any text and then “said”. How many lines are there?

Relevant topics: *grep*

Item(s) to submit:

- The UNIX command used to solve this problem.
- The number of lines that contain the text “that’s what” followed by any amount of text and then “said”.

Regular expressions are really a useful semi language-agnostic tool. What this means is regardless of the programming language your are using, there will be

some package that allows you to use regular expressions. In fact, we can use them in both R and Python! This can be particularly useful when dealing with strings. Load up the dataset you discovered in (1) using `read.csv`. Name the resulting `data.frame` `dat`.

6. The `text_w_direction` column in `dat` contains the characters' lines with inserted direction that helps characters know what to do as they are reciting the lines. Direction is shown between square brackets "[" and "]". Create a new column called `has_direction` that is set to `TRUE` if the `text_w_direction` column has direction, and `FALSE` otherwise. Use regular expressions and the `grepl` function in R to accomplish this.

Hint: Make sure all opening brackets "[" have a corresponding closing bracket "]"

Hint: Think of the pattern as any line that has a [, followed by any amount of any text, followed by a], followed by any amount of any text.

Relevant topics: *grep, grepl*

Item(s) to submit:

- The R code used to solve this problem.

7. Modify your regular expression in (7) to find lines with 2 or more sets of direction.

For example, the following line has 2 directions: `dat$text_w_direction[2789]`. How many lines have more than 2 directions? How many have more than 5?

This is a line with [emphasize this] only 1 direction!

This is a line with [emphasize this] 2 sets of direction, do you see the difference [shrug].

In (6), your solution may have found a match in both lines. In this question we want it to find only lines with 2+ directions, so the first line would not be a match.

Relevant topics: *length, grep*

Item(s) to submit:

- The R code used to solve this problem.
- How many lines have > 2 directions?
- How many lines have > 5 directions?

8. Use the `str_extract_all` function from the `stringr` package to extract the direction(s) as well as the text between direction(s) from each line. Put the strings in a new column called `direction`.

This is a line with [emphasize this] only 1 direction!

This is a line with [emphasize this] 2 sets of direction, do you see the difference [shrug]

In this question, your solution may have extracted:

[emphasize this]

[emphasize this] 2 sets of direction, do you see the difference [shrug]

This is ok.

Note: If you capture text between two sets of direction, this is ok. For example, if we capture “[this] is a [test]” from “if we capture [this] is a [test]”, this is ok.

Relevant topics: `str_extract_all`

Item(s) to submit:

- The R code used to solve this problem.

Project 5 {#290-p05}

Motivation: Becoming comfortable stringing together commands and getting used to navigating files in a terminal is important for every data scientist to do. By learning the basics of a few useful tools, you will have the ability to quickly understand and manipulate files in a way which is just not possible using tools like Microsoft Office, Google Sheets, etc.

Context: We’ve been using UNIX tools in a terminal to solve a variety of problems. In this project we will continue to solve problems by combining a variety of tools using a form of redirection called piping.

Scope: `grep`, regular expression basics, UNIX utilities, redirection, piping

Learning objectives:

- Use `cut` to section off and slice up data from the command line.
- Use piping to string UNIX commands together.
- Use `sort` and its options to sort data in different ways.
- Use `head` to isolate n lines of output.
- Use `wc` to summarize the number of lines in a file or in output.
- Use `uniq` to filter out non-unique lines.
- Use `grep` to search files effectively.

You can find useful examples that walk you through relevant material in The Examples Book:

<https://thedatamine.github.io/the-examples-book>

It is highly recommended to read through, search, and explore these examples to help solve problems in this project.

Important note: It is highly recommended that you use <https://rstudio.scholar.rcac.purdue.edu/>. Simply click on the link and login using your Purdue account credentials. Use another system at your own risk. The version of RStudio on <https://desktop.scholar.rcac.purdue.edu/> (which uses ThinLinc), is 99.9.9, and is known to have some strange issues when running code chunks.

Don't forget the very useful documentation shortcut `?`. To use, simply type `?` in the console, followed by the name of the function you are interested in.

You can also look for package documentation by using `help(package=PACKAGENAME)`, so for example, to see the documentation for the package `ggplot2`, we could run:

```
help(package=ggplot2)
```

Sometimes it can be helpful to see the source code of a defined function. A function is any chunk of organized code that is used to perform an operation. Source code is the underlying R or C or C++ code that is used to create the function. To see the source code of a defined function, type the function's name without the `()`. For example, if we were curious about what the function `Reduce` does, we could run:

```
Reduce
```

Occasionally this will be less useful as the resulting code will be code that calls C code we can't see. Other times it will allow you to understand the function better.

Dataset

The following questions will use the dataset found in Scholar:

```
/class/datamine/data/amazon/amazon_fine_food_reviews.csv
```

A public sample of the data can be found here: `amazon_fine_food_reviews.csv`

Answers to questions should all be answered using the full dataset located on Scholar. You may use the public samples of data to experiment with your solutions prior to running them using the full dataset.

Questions

1. What is the Id of the most helpful review if we consider the review with highest HelpfulnessNumerator to be an indicator of helpfulness (higher is more helpful)?

Relevant topics: *cut, sort, head, piping*

Item(s) to submit:

- Line of UNIX commands used to solve the problem.
- The Id of the most helpful review.

2. What proportion of all Summary's are unique? Use two lines of UNIX commands to find the answer.

Relevant topics: *cut, uniq, sort, wc, piping*

Item(s) to submit:

- Two lines of UNIX commands used to solve the problem.
- The ratio of unique Summary's.

3. Use a simple UNIX command to create a frequency table of Score.

Relevant topics: *cut, uniq, sort, piping*

Item(s) to submit:

- The line of UNIX commands used to solve the problem.
- The frequency table.

4. Who is the user with the highest number of reviews? There are two columns you could use to answer this question, but which column do you think would be most appropriate and why?

Hint: You may need to pipe the output to *sort* multiple times.

Hint: To create the frequency table, read through the *man* pages for *uniq*. *Man* pages are the “manual” pages for UNIX commands. You can read through the *man* pages for *uniq* by running the following:

```
man uniq
```

Relevant topics: *cut, uniq, sort, head, piping, man*

Item(s) to submit:

- The line of UNIX commands used to solve the problem.
- The frequency table.

5. Anecdotally, there seems to be a tendency to leave reviews when we feel strongly (either positive or negative) about a product. For the user with the highest number of reviews, would you say that they follow this pattern of extremes? Let’s consider 5 star reviews to be strongly positive and 1 star reviews to be strongly negative. Let’s consider anything in between neither strongly positive nor negative.

Hint: You may find the solution to problem (3) useful.

Relevant topics: *cut, uniq, sort, grep, piping*

Item(s) to submit:

- The line of UNIX commands used to solve the problem.

6. We want to compare the most helpful review with a Score of 5 with the most helpful review with a Score of 1. Use UNIX commands to calculate these values. Write down the *ProductId* of both reviews. In the case of a tie, write down all *ProductId*’s to get full credit. In this case we are considering the most helpful review to be the review with the highest *HelpfulnessNumerator*.

Hint: You can use multiple lines to solve this problem.

Relevant topics: *sort, head, piping*

Item(s) to submit:

- The lines of UNIX commands used to solve the problem.
- ProductId's of both requested reviews.

7. Using the ProductId's from the previous question, create a new dataset called `reviews.csv` which contains the ProductId's and Score of all reviews with the corresponding ProductId's.

Relevant topics: *grep, redirection*

Item(s) to submit:

- The line of UNIX commands used to solve the problem.

8. Use R to load up `reviews.csv` into a new `data.frame` called `dat`. Create a histogram for each products' Score. Compare the most helpful review Score with the Score's given in the histogram. Based on this comparison, decide (anecdotally) whether you think people found the review helpful because the product is overrated, underrated, or correctly reviewed by the masses.

Relevant topics: *read.csv, hist*

Item(s) to submit:

- R code used to create the histograms.
- 3 histograms, 1 for each ProductId.
- 1-2 sentences explaining whether or not you think people found the review helpful because the produce is overrated, underrated, or correctly reviewed, and why.

Project 6 {#290-p06}

Motivation: A bash script is a powerful tool to perform repeated tasks. RCAC uses bash scripts to automate a variety of tasks. In fact, we use bash scripts on Scholar to do things like link Python kernels to your account, fix potential issues

with Firefox, etc. `awk` is a programming language designed for text processing. The combination of these tools can be really powerful and useful for a variety of quick tasks.

Context: This is the first part in a series of projects that are designed to exercise skills around UNIX utilities, with a focus on writing bash scripts and `awk`. You will get the opportunity to manipulate data without leaving the terminal. At first it may seem overwhelming, however, with just a little practice you will be able to accomplish data wrangling tasks really efficiently.

Scope: `awk`, UNIX utilities, bash scripts

Learning objectives:

- Use `awk` to process and manipulate textual data.
- Use piping and redirection within the terminal to pass around data between utilities.

Dataset:

The following questions will use the dataset found here or in Scholar:

`/class/datamine/data/flights/subset/YYYY.csv`

An example from 1987 data can be found here or in Scholar:

`/class/datamine/data/flights/subset/1987.csv`

Questions

1. In previous projects we learned how to get a single column of data from a csv file. Write 1 line of UNIX commands to print the 17th column, the `Origin`, from `1987.csv`. Write another line, this time using `awk` to do the same thing. Which one do you prefer, and why?

Relevant topics: `cut`, `awk`

Item(s) to submit:

- One line of UNIX commands to solve the problem *without* using `awk`.
- One line of UNIX commands to solve the problem using `awk`.
- 1-2 sentences describing which method you prefer and why.

2. Write a bash script that accepts a year (1987, 1988, etc.) and a column n and returns the n th column of the associated year of data.

Relevant topics: awk, bash scripts

Item(s) to submit:

- The content of your bash script (starting with “#!/bin/bash”) in a code chunk.

3. How many flights came into Indianapolis (IND) in 2008? First solve this problem without using awk, then solve this problem using *only* awk.

Relevant topics: cut, grep, wc, awk, piping

Item(s) to submit:

- One line of UNIX commands to solve the problem *without* using awk.
- One line of UNIX commands to solve the problem using **awk**.
- The number of flights that came into Indianapolis (IND) in 2008.

4. Do you expect the number of unique origins and destinations to be the same? Find out using any command line tool you’d like. Are they indeed the same? How many unique values do we have per category (Origin, Dest)?

Relevant topics: cut, sort, uniq, wc, awk

Item(s) to submit:

- 1-2 sentences explaining whether or not you expect the number of unique origins and destinations to be the same.
- The UNIX command(s) used to figure out if the number of unique origins and destinations are the same.
- The number of unique values per category (Origin, Dest).

5. In (4) we found that there are not the same number of unique Origin’s as Dest’s. Find the IATA airport code for all Origin’s that don’t appear in a Dest and all Dest’s that don’t appear in an Origin.

Hint: https://www.tutorialspoint.com/unix_commands/comm.html

Relevant topics: comm, cut, sort, uniq, redirection

Item(s) to submit:

- The line(s) of UNIX command(s) used to answer the question.
- The list of `Origins` that don't appear in `Dest`.
- The list of `Dests` that don't appear in `Origin`.

6. What was the average number of flights in 2008 per unique `Origin` with the `Dest` of “IND”? How does “PHX” (as a unique `Origin`) compare to the average?

Hint: You manually do the average calculation by dividing the result from (3) by the number of unique `Origin`'s that have a `Dest` of “IND”.

Relevant topics: awk, sort, grep, wc

Item(s) to submit:

- The average number of flights in 2008 per unique `Origin` with the `Dest` of “IND”.
- 1-2 sentences explaining how “PHX” compares (as a unique `Origin`) to the average?

7. Write a bash script that takes a year and IATA airport code and returns the year, and the total number of flights to and from the given airport. Example rows may look like:

```
1987, 12345
1988, 44
```

Run the script with inputs: 1991 and ORD. Include the output in your submission.

Relevant topics: bash scripts, cut, piping, grep, wc

Item(s) to submit:

- The content of your bash script (starting with “#!/bin/bash”) in a code chunk.
 - The output of the script given 1991 and ORD as inputs.
-

Project 7 {#290-p07}

Motivation: A bash script is a powerful tool to perform repeated tasks. RCAC uses bash scripts to automate a variety of tasks. In fact, we use bash scripts on Scholar to do things like link Python kernels to your account, fix potential issues with Firefox, etc. **awk** is a programming language designed for text processing. The combination of these tools can be really powerful and useful for a variety of quick tasks.

Context: This is the first part in a series of projects that are designed to exercise skills around UNIX utilities, with a focus on writing bash scripts and **awk**. You will get the opportunity to manipulate data without leaving the terminal. At first it may seem overwhelming, however, with just a little practice you will be able to accomplish data wrangling tasks really efficiently.

Scope: **awk**, UNIX utilities, bash scripts

Learning objectives:

- Use **awk** to process and manipulate textual data.
- Use piping and redirection within the terminal to pass around data between utilities.

Dataset:

The following questions will use the dataset found in Scholar: `/class/datamine/data/flights/subset/`

An example of the data for the year 1987 can be found [here](#).

Sometimes if you are about to dig into a dataset, it is good to quickly do some sanity checks early on to make sure the data is what you expect it to be.

1. Write a line of code that prints a list of the unique values in the `DayOfWeek` column. Write a line of code that prints a list of the unique values in the `DayOfMonth` column. Write a line of code that prints a list of the unique values in the `Month` column. Use the `1987.csv` dataset. Are the results what you expected?

Relevant topics: `cut`, `sort`

Item(s) to submit:

- 3 lines of code used to get a list of unique values for the chosen columns.
- 1-2 sentences explaining whether or not the results are what you expected.

2. Our files should have 29 columns. Write a line of code that prints any lines in a file that do *not* have 29 columns. Test it on 1987.csv, were there any rows without 29 columns?

Relevant topics: awk

Item(s) to submit:

- Line of code used to solve the problem.
- 1-2 sentences explaining whether or not there were any rows without 29 columns.

3. Write a bash script that, given a “begin” year and “end” year, cycles through the associated files and prints any lines that do *not* have 29 columns.

Relevant topics: awk, bash scripts

Item(s) to submit:

- The content of your bash script (starting with “#!/bin/bash”) in a code chunk.
- The results of running your bash scripts from year 1987 to 2008.

4. awk is a really good tool to quickly get some data and manipulate it a little bit. For example, let’s see the number of kilometers and miles traveled in 1990. To convert from miles to kilometers, simply multiply by 1.609344.

Example output:

```
Miles: 12345
Kilometers: 19867.35168
```

Relevant topics: awk, piping

Item(s) to submit:

- The code used to solve the problem.
- The results of running the code.

5. Use `awk` to calculate the number of `DepDelay` minutes by `DayOfWeek`. Use `2007.csv`.

Example output:

```
DayOfWeek: 0
1: 1234567
2: 1234567
3: 1234567
4: 1234567
5: 1234567
6: 1234567
7: 1234567
```

Note: 1 is Monday.

Relevant topics: `awk`, `sort`, piping

Item(s) to submit:

- The code used to solve the problem.
- The output from running the code.

6. It wouldn't be fair to compare the total `DepDelay` minutes by `DayOfWeek` as the number of flights may vary. One way to take this into account is to instead calculate an average. Modify (5) to calculate the average number of `DepDelay` minutes by the number of flights per `DayOfWeek`. Use `2007.csv`.

Example output:

```
DayOfWeek: 0
1: 1.234567
2: 1.234567
3: 1.234567
4: 1.234567
5: 1.234567
6: 1.234567
7: 1.234567
```

Relevant topics: awk, sort, piping

Item(s) to submit:

- The code used to solve the problem.
- The output from running the code.

7. As a quick follow-up, *slightly* modify (6) to perform the same calculation for ArrDelay. Do the ArrDelays and DepDelays appear to have the highest delays on the same day? Use 2007.csv.

Example output:

```
DayOfWeek: 0
1: 1.234567
2: 1.234567
3: 1.234567
4: 1.234567
5: 1.234567
6: 1.234567
7: 1.234567
```

Relevant topics: awk, sort, piping

Item(s) to submit:

- The code used to solve the problem.
- The output from running the code.
- 1-2 sentences explaining whether or not the ArrDelays and DepDelays appear to have the highest delays on the same day.

Project 8 {#290-p08}

Motivation: A bash script is a powerful tool to perform repeated tasks. RCAC uses bash scripts to automate a variety of tasks. In fact, we use bash scripts on Scholar to do things like link Python kernels to your account, fix potential issues with Firefox, etc. `awk` is a programming language designed for text processing.

The combination of these tools can be really powerful and useful for a variety of quick tasks.

Context: This is the first part in a series of projects that are designed to exercise skills around UNIX utilities, with a focus on writing bash scripts and **awk**. You will get the opportunity to manipulate data without leaving the terminal. At first it may seem overwhelming, however, with just a little practice you will be able to accomplish data wrangling tasks really efficiently.

Scope: awk, UNIX utilities, bash scripts

Learning objectives:

- Use **awk** to process and manipulate textual data.
- Use piping and redirection within the terminal to pass around data between utilities.

Dataset:

The following questions will use the dataset found in Scholar: `/class/datamine/data/flights/subset/`

An example of the data for the year 1987 can be found [here](#).

Let's say we have a theory that there are more flights on the weekend days (Friday, Saturday, Sunday) than the rest of the days, on average. We can use **awk** to quickly check it out and see if maybe this looks like something that is true!

1. Write a line of **awk code that, prints the number of flights on the weekend days, followed by the number of flights on the weekdays for the flights during 2008.**

Relevant topics: awk

Item(s) to submit:

- Line of **awk** code that solves the problem.
- The result: the number of flights on the weekend days, followed by the number of flights on the weekdays for the flights during 2008.

2. Note that in (1), we are comparing 3 days to 4! Write a line of **awk code that, prints the average number of flights on a weekend day, followed by the average number of flights on the weekdays. Continue to use data for 2008.**

Relevant topics: awk

Item(s) to submit:

- Line of `awk` code that solves the problem.
- The result: the average number of flights on the weekend days, followed by the average number of flights on the weekdays for the flights during 2008.

We want to look to see if there may be some truth to the whole “snow bird” concept where people will travel to warmer states like Florida and Arizona during the Winter. Let’s use the tools we’ve learned to explore this a little bit.

3. Take a look at `airports.csv`. In particular run the following:

```
head airports.csv
```

Notice how all of the non-numeric text is surrounded by quotes. The surrounding quotes would need to be escaped for any comparison within `awk`. This is messy and we would prefer to create a new file called `new_airports.csv` without any quotes. Write a line of code to do this.

Hint: You could use `gsub` within `awk` to replace ““with”.

Hint: If you leave out the column number argument to `gsub` it will apply the substitution to every field in every column.

Relevant topics: `awk`, redirection

Item(s) to submit:

- Line of `awk` code used to create the new dataset.

4. Write a line of commands that create a new dataset called `az_fl_airports.txt` that contains a list of airport codes for all airports from both Arizona (AZ) and Florida (FL). Use the file we created in (3), `new_airports.csv`.

Relevant topics: `awk`

Item(s) to submit:

- The line of UNIX commands to create an array called `airports`.

5. Wow! In (4) we discovered a lot of airports! How many airports are there? Did you expect this? Use a line of bash code to answer this question.

Relevant topics: echo, wc, piping

Item(s) to submit:

- Line of UNIX commands used to solve the problem.
- The number of airports.
- 1-2 sentences explaining whether you expected this result and why or why not.

6. Create a new dataset that contains all of the data for flights into or out of Florida and Arizona using 2008.csv, use the newly created dataset, az_fl_airports.txt in (4) to do so.

Hint: <https://unix.stackexchange.com/questions/293684/basic-grep-awk-help-extracting-all-lines-containing-a-list-of-terms-from-one-f>

Relevant topics: grep

Item(s) to submit:

- Line of UNIX commands used to solve the problem.

7. Now that you have code to complete (6), write a bash script that accepts the start year, end year, and filename containing airport codes (az_fl_airports.txt), and outputs the data for flights into or out of any of the airports listed in the provided filename containing airport codes using *all* of the years of data in the provided range. Run the bash script to create a new file called az_fl_flights.csv.

Relevant topics: bash scripts, grep, for loop, redirection

Item(s) to submit:

- The content of your bash script (starting with “#!/bin/bash”) in a code chunk.
 - The line of UNIX code you used to execute the script and create the new dataset.
-

Project 9 {#290-p09}

Project 10 {#290-p10}

Project 11 {#290-p11}

Project 12 {#290-p12}

Motivation: Databases are comprised of many tables. It is imperative that we learn how to combine data from multiple tables using queries. To do so we perform joins! In this project we will explore learn about and practice using joins on a database containing bike trip information from the Bay Area Bike Share.

Context: We've introduced a variety of SQL commands that let you filter and extract information from a database in an systematic way. In this project we will introduce joins, a powerful method to combine data from different tables.

Scope: SQL, sqlite, joins

Learning objectives:

- Briefly explain the differences between left and inner join and demonstrate the ability to use the join statements to solve a data-driven problem.
- Perform grouping and aggregate data using group by and the following functions: count, max, sum, avg, like, having.
- Showcase the ability to filter, alias, and write subqueries.

Dataset

The following questions will use the dataset found in Scholar:

`/class/datamine/data/bay_area_bike_share/bay_area_bike_share.db`

A public sample of the data can be found [here](#).

Questions

1. There are a variety of ways to join data using SQL. With that being said, if you are able to understand and use a **LEFT JOIN** and **INNER JOIN**, you can perform *all* of the other types of joins (**RIGHT JOIN**, **FULL OUTER JOIN**). Given the following two tables, use

RMarkdown to display the result of performing the following query as a table:

```
SELECT * FROM users AS u INNER JOIN dorms AS d ON u.dorm=d.id;
```

users:

| id | first_name | last_name | dorm |
|----|------------|-----------|------|
| 1 | Alice | Smith | 1 |
| 2 | Bob | Johnson | 2 |
| 3 | Susan | Marques | 3 |
| 4 | Amare | Keita | 3 |
| 5 | Kristen | Lakehold | 4 |

dorms:

| id | name | capacity | address |
|----|------------------|----------|--|
| 1 | Windsor Halls | NULL | Windsor Halls, West Lafayette, IN, 47906 |
| 2 | Cary Quadrangle | 1200 | 1016 W Stadium Ave, West Lafayette, IN 47906 |
| 3 | Hillenbrand Hall | NULL | 1301 3rd Street, West Lafayette, IN 47906 |

Relevant topics: sql, inner join

Item(s) to submit:

- RMarkdown table displaying the result of performing the following query as a table.

2. Using the same two tables from (1), use RMarkdown to display the result of performing the following query as a table. Explain the difference between an INNER JOIN and LEFT JOIN.

```
SELECT * FROM users AS u LEFT JOIN dorms AS d ON u.dorm=d.id;
```

Relevant topics: sql, left join

Item(s) to submit:

- RMarkdown table displaying the result of performing the following query as a table.
- 1-2 sentences explaining (in your own words) what the difference between `INNER` and `LEFT JOIN` is.

3. Aliases can be created for tables, fields, and even results of aggregate functions (like `MIN`, `MAX`, `COUNT`, `AVG`, etc.). In addition, you can combine fields using the `sqlite` concatenate operator `||` (see [here](#)). Write a query that returns the first 5 records of information from the `station` table formatted in the following way:

```
(id) name @ (lat, long)
```

For example:

```
(84) Ryland Park @ (37.342725,-121.895617)
```

Relevant topics: aliasing, concat

Item(s) to submit:

- SQL query used to solve this problem.
- The first 5 records of information from the `station` table.

4. There is a variety of interesting weather information in the `weather` table. Write a query that finds the average `mean_temperature_f` by `zip_code`. Which is on average the warmest `zip_code`?

Use aliases to format the result in the following way:

```
Zip Code|Avg Temperature
94041|61.3808219178082
```

Relevant topics: aliasing, group by, avg

Item(s) to submit:

- SQL query used to solve this problem.
- The results of the query copy and pasted.

5. From (4) we can see that there are only 5 `zip_codes` with weather information. How many unique `zip_codes` do we have in the `trip` table? Write a query that finds the number of unique `zip_codes` in the `trip` table. Write another query that lists the `zip_code` and count of the number of times the `zip_code` appears. If we had originally assumed that the `zip_code` was related to the location of the trip itself, we were wrong. Can you think of a likely explanation?

Relevant topics: group by, count

Item(s) to submit:

- SQL queries used to solve this problem.
- 1-2 sentences explaining what a possible explanation for the `zip_codes` could be.

6. In (4) we wrote a query that finds the average `mean_temperature_f` by `zip_code`. What if we want to tack on to our results information from each row in the `station` table based on the `zip_codes`? To do, use an `INNER JOIN`. `INNER JOIN` combines tables based on specified fields, and returns only rows where there is a match in both the “left” and “right” tables.

Hint: Use the query from (4) as a sub query within your solution.

Relevant topics: inner join, subqueries, aliasing

Item(s) to submit:

- SQL query used to solve this problem.

7. In (5) we eluded that the `zip_codes` in the `trip` table aren’t very consistent. Users can enter a `zip code` when using the app. This means that `zip_code` can be from anywhere in the world! With that being said, if the `zip_code` is one of the 5 `zip_codes` for which we have weather data (from question 4), we can add that weather information to matching rows of the `trip` table. In (6) we used an `INNER JOIN` to append some weather information to each row in the `station` table. For this question, write a query that performs an `INNER JOIN` and appends weather data from the `weather` table to the `trip` data from the `trip` table. Limit your solution to 5 lines.

Hint: You will want to wrap your dates and datetimes in `sqlite’s date` function prior to comparison.

Important note: Notice that the weather data has about 1 row of weather information for each date and each zip code. This means you may have to join your data based on multiple constraints instead of just 1 like in (6).

Relevant topics: inner join, aliasing

Item(s) to submit:

- SQL query used to solve this problem.
- First 5 lines of output.

Project 13 {#290-p13}

Project 14 {#290-p14}

Project 15 {#290-p15}

Motivation: We've done a lot of work with SQL this semester. Let's review concepts in this project and mix and match R, Python, and SQL to solve data-driven problems.

Context: In this project, we will reinforce topics you've already learned, with a focus on SQL.

Scope: SQL, sqlite, R, Python

Learning objectives:

- Write and run SQL queries in `sqlite` on real-world data.
- Use SQL from within R and Python.

Dataset

The following questions will use the dataset found in Scholar:

`/class/datamine/data/movies_and_tv/imdb.db`

A public sample of the data can be found [here](#).

In this project we want to offer the flexibility of using your choice of R and/or Python. To keep things as consistent as possible, please use Rmarkdown on <https://rstudio.scholar.rcac.purdue.edu/>. See [here](#) to learn how to run Python in this environment.

Questions

1. What is the first year where our database has > 1000 titles? Use the `premiered` column in the `titles` table as our year. What year has the most titles?

Relevant topics: count, group by, order by, desc

Item(s) to submit:

- 1 or more SQL queries used to answer the questions.
- What year is the first year to have > 1000 titles?
- What year has the most titles?

2. How many, and what are the unique types from the `titles` table? From the year from (1) with the most titles, how many titles of each type are there?

Item(s) to submit:

- 1 or more SQL queries used to answer the questions.
- How many and what are the unique types from the `titles` table?
- A list of type and count for the year (`premiered`) 2017.

F.R.I.E.N.D.S is a popular tv show. They have an interesting naming convention for the names of their episodes. They all begin with the text “The One ...”. There are 6 primary characters in the show: Chandler, Joey, Monica, Phoebe, Rachel, and Ross. Let’s use SQL and R to take a look at how many times each characters’ names appear in the title of the episodes.

3. Write a query that gets the `episode_title_id`, `primary_title`, `rating`, and `votes`, of all of the episodes of Friends (`title_id` is `tt0108778`).

Hint: You can slightly modify the solution to question (7) in project 13.

Relevant topics: inner join, subqueries, aliasing

Item(s) to submit:

- SQL query used to answer the question.
- First 5 results of the query.

The next couple of questions should be complete in the same language. You can use either R or Python, but you must use the same for both questions.

4. Now that you have a working query, connect to the database and run the query to get the data into an R or pandas data frame. In previous projects, we learned how to use regular expressions to search for text. For each character, how many episodes `primary_titles` contained their name?

Relevant topics: SQL in R, SQL in Python, grep

Item(s) to submit:

- R or Python code in a code chunk that was used to find the solution.
- The solution pasted below the code chunk.

5. Create a graphic showing our results in (2) using your favorite package. Make sure the plot has a good title, x-label, y-label, and try to incorporate some of the following colors: `#273c8b`, `#bd253a`, `#016f7c`, `#f56934`, `#016c5a`, `#9055b1`, `#eaab37`.

Relevant topics: plotting

Item(s) to submit:

- The R or Python code used to generate the graphic.
- The graphic in a png or jpg/jpeg format.

6. Use any combination of SQL, R, and Python you'd like in order to find which of the following 3 genres has the highest average rating for movies (see `type` column from `titles` table): Romance, Comedy, Animation. In the `titles` table, you can find the genres in the `genres` column. There may be some overlap (i.e. a movie may have more than one genre), this is ok.

To query rows which have the genre Action as one of its genres:

```
SELECT * FROM titles WHERE genres LIKE '%action%';
```

Relevant topics: like, inner join

Item(s) to submit:

- Any code you used to solve the problem in a code chunk.
 - The average rating of each of the genres listed for movies.
-

STAT 39000

Project 1 {#390-p01}

Project 2 {#390-p02}

Motivation: The ability to quickly reproduce an analysis is important. It is often necessary that other individuals will need to be able to understand and reproduce an analysis. This concept is so important there are classes solely on reproducible research! In fact, there are papers that investigate and highlight the lack of reproducibility in various fields. If you are interested in reading about this topic, a good place to start is the paper titled “Why Most Published Research Findings Are False”, by John Ioannidis (2005).

Context: Making your work reproducible is extremely important. We will focus on the computational part of reproducibility. We will learn RMarkdown to document your analyses so others can easily understand and reproduce the computations that led to your conclusions. Pay close attention as future project templates will be RMarkdown templates.

Scope: Understand Markdown, RMarkdown, and how to use it to make your data analysis reproducible.

Learning objectives:

- Use Markdown syntax within an Rmarkdown document to achieve various text transformations.
- Use RMarkdown code chunks to display and/or run snippets of code.

You can find useful examples that walk you through relevant material in The Examples Book:

<https://thedatamine.github.io/the-examples-book>

It is highly recommended to read through, search, and explore these examples to help solve problems in this project.

Important note: It is highly recommended that you use <https://rstudio.scholar.rcac.purdue.edu/>. Simply click on the link and login using your Purdue account credentials. Use another system at your own risk. The version of RStudio on <https://desktop.scholar.rcac.purdue.edu/> (which uses ThinLinc), is 99.9.9, and is known to have some strange issues when running code chunks.

Don’t forget the very useful documentation shortcut `?<function>`. To use, simply type `?<function>` in the console, followed by the name of the function you are interested in.

You can also look for package documentation by using `help(package=PACKAGENAME)`, so for example, to see the documentation for the package `ggplot2`, we could run:

```
help(package=ggplot2)
```

Sometimes it can be helpful to see the source code of a defined function. A function is any chunk of organized code that is used to perform an operation. Source code is the underlying R or c or c++ code that is used to create the function. To see the source code of a defined function, type the function's name without the `()`. For example, if we were curious about what the function `Reduce` does, we could run:

```
Reduce
```

Occasionally this will be less useful as the resulting code will be code that calls c code we can't see. Other times it will allow you to understand the function better.

1. Make the following text (including the asterisks) bold: This needs to be *very* bold. Make the following text (including the under-scores) italicized: This needs to be very italicized.

Hint: Try mixing and matching ways to embolden or italicize text. Alternatively, look up “escaping characters in markdown”, or see [here](#).

Hint: Be sure to check out the *Rmarkdown Cheatsheet* and our section on *Rmarkdown in the book*.

Note: *Rmarkdown is essentially Markdown + the ability to run and display code chunks. In this question, we are actually using Markdown within Rmarkdown!*

Relevant topics: *rmarkdown, escaping characters*

Item(s) to submit:

- Fill in the chunk under (1) in the `stat29000project02template.Rmd` file with 2 lines of markdown text. Note that when compiled, this text will be unmodified, regular text.

2. Create an unordered list of your top 3 favorite academic interests (some examples could include: machine learning, operating systems, forensic accounting, etc.). Create another *ordered* list that ranks your academic interests in order of most interested to least interested.

Hint: You can learn what ordered and unordered lists are [here](#).

Note: Similar to (1a), in this question we are dealing with Markdown. If we were to copy and paste the solution to this problem in a Markdown editor, it would be the same result as when we Knit it here.

Relevant topics: *rmarkdown*

Item(s) to submit:

- Create the lists under (2) in the `stat29000project02template.Rmd` file. Note that when compiled, this text will appear as nice, formatted lists.

3. Browse <https://www.linkedin.com/> and read some profiles. Pay special attention to accounts with an “About” section. Write your own personal “About” section using Markdown. Include the following:

- A header (your choice of size) that says “About”.
- A horizontal rule directly underlining the header.
- The text of your personal “About” section that you would feel comfortable uploading to linkedin, including at least 1 link.

Relevant topics: *rmarkdown*

Item(s) to submit:

- Create the described profile under (3) in the `stat29000project02template.Rmd` file.

4. LaTeX is a powerful editing tool where you can create beautifully formatted equations and formulas. Replicate the equation found [here](#) as closely as possible.

Hint: Lookup “*latex mid*” and “*latex frac*”.

Item(s) to submit:

- Replicate the equation using LaTeX under (1d) in the `stat39000project02template.Rmd` file.

5. Your co-worker wrote a report, and has asked you to beautify it. Knowing Rmarkdown, you agreed. Spruce up the report found under (4) in `stat29000project02template.Rmd`. At a minimum:

- Make the title pronounced.
- Make all links appear as a word or words, rather than the long-form URL.
- Organize all code into code chunks where code and output are displayed. If the output is really long, just display the code.
- Make the calls to the `library` function and the `install.packages` function be evaluated but not displayed. Make sure all warnings and errors that may eventually occur, do not appear in the final document.

Feel free to make any other changes that make the report more visually pleasing.

Relevant topics: *rmarkdown*

Item(s) to submit:

- Spruce up the “document” under (4) in the `stat29000project02template.Rmd` file.

6. Create a plot using a built-in dataset like `iris`, `mtcars`, or `Titanic`, and display the plot using a code chunk. Make sure the code used to generate the plot is hidden. Include a descriptive caption for the image.

Relevant topics: *rmarkdown*, *plotting in r*

Item(s) to submit:

- Code chunk under (5) that creates and displays a plot using a built-in dataset like `iris`, `mtcars`, or `Titanic`.

7. Insert the following code chunk under (5) in `stat29000project02template.Rmd`. Try knitting the document. Something should go wrong. Fix the problem and knit again. If another problem appears, fix it. What was the first problem? What was the second problem?

```
```{r install-packages}
plot(my_variable)
```
```

Hint: Take a close look at the name we give our code chunk.

Hint: Take a look at the code chunk where `my_variable` is declared.

Relevant topics: *rmarkdown*

Item(s) to submit:

- The modified version of the inserted code that fixes both problems.
- A sentence explaining what the first problem was.
- A sentence explaining what the second problem was.

8. RMarkdown is also an excellent tool to create a slide deck. Use the information [here](#) or [here](#) to convert your solutions into a slide deck rather than the regular PDF. You may use `slidy`, `ioslides` or `beamer`. Make any needed modifications to make the solutions knit into a well-organized slide deck. Modify (2) so the bullets are incrementally presented as the slides progresses.

Relevant topics: *rmarkdown*

Item(s) to submit:

- The modified version of the inserted code that fixes both problems.
- A sentence explaining what the first problem was.
- A sentence explaining what the second problem was.

Project 3 {#390-p03}

Project 4 {#390-p04}

Motivation: The need to search files and datasets based on the text held within is common during various parts of the data wrangling process. `grep` is an extremely powerful UNIX tool that allows you to do so using regular expressions. Regular expressions are a structured method for searching for specified patterns. Regular expressions can be very complicated, even professionals can make critical mistakes. With that being said, learning some of the basics is an incredible tool that will come in handy regardless of the language you are working in.

Context: We've just begun to learn the basics of navigating a file system in UNIX using various terminal commands. Now we will go into more depth with one of the most useful command line tools, **grep**, and experiment with regular expressions using **grep**, R, and later on, Python.

Scope: grep, regular expression basics, utilizing regular expression tools in R and Python

Learning objectives:

- Use **grep** to search for patterns within a dataset.
- Use **cut** to section off and slice up data from the command line.
- Use **wc** to count the number of lines of input.

You can find useful examples that walk you through relevant material in The Examples Book:

<https://thedatamine.github.io/the-examples-book>

It is highly recommended to read through, search, and explore these examples to help solve problems in this project.

Important note: It is highly recommended that you use <https://rstudio.scholar.rcac.purdue.edu/>. Simply click on the link and login using your Purdue account credentials. Use another system at your own risk. The version of RStudio on <https://desktop.scholar.rcac.purdue.edu/> (which uses ThinLinc), is 99.9.9, and is known to have some strange issues when running code chunks.

Don't forget the very useful documentation shortcut `?.` To use, simply type `?.` in the console, followed by the name of the function you are interested in.

You can also look for package documentation by using `help(package=PACKAGENAME)`, so for example, to see the documentation for the package `ggplot2`, we could run:

```
help(package=ggplot2)
```

Sometimes it can be helpful to see the source code of a defined function. A function is any chunk of organized code that is used to perform an operation. Source code is the underlying R or C or C++ code that is used to create the function. To see the source code of a defined function, type the function's name without the `()`. For example, if we were curious about what the function `Reduce` does, we could run:

```
Reduce
```

Occasionally this will be less useful as the resulting code will be code that calls C code we can't see. Other times it will allow you to understand the function better.

Dataset

The following questions will use the dataset found in Scholar:

`/class/datamine/data/movies_and_tv/the_office_dialogue.csv`

A public sample of the data can be found here: `the_office_dialogue.csv`

Answers to questions should all be answered using the full dataset located on Scholar. You may use the public samples of data to experiment with your solutions prior to running them using the full dataset.

`grep` stands for (g)lobally search for a (r)egular (e)xpression and (p)rint matching lines. As such, to best demonstrate `grep`, we will be using it with textual data. You can read about and see examples of `grep` here.

1. Login to Scholar and use `grep` to find the dataset we will use this project. The dataset we will use is the only dataset to have the text “bears. beets. battlestar galactica.”. What is the name of the dataset and where is it located?

Relevant topics: *grep*

Item(s) to submit:

- The `grep` command used to find the dataset.
- The name and location in Scholar of the dataset.
- Use `grep` and `grep1` within R to solve a data-driven problem.

2. `grep` prints the line that the text you are searching for appears in. In project 3 we learned a UNIX command to quickly print the first n lines from a file. Use this command to get the headers for the dataset. As you can see, each line in the tv show is a row in the dataset. You can count to see which column the various bits of data live in.

Write a line of UNIX commands that searches for “bears. beets. battlestar galactica.” and, rather than printing the entire line, prints only the character who speaks the line, as well as the line itself.

Hint: *The result if you were to search for “bears. beets. battlestar galactica.” should be:*

`"Jim","Fact. Bears eat beets. Bears. Beets. Battlestar Galactica."`

Hint: *One method to solve this problem would be to pipe the output from `grep` to `cut`.*

Relevant topics: *cut, grep*

Item(s) to submit:

- The line of UNIX commands used to perform the operation.

3. This particular dataset happens to be very small. You could imagine a scenario where the file is many gigabytes and not easy to load completely into R or Python. We are interested in learning what makes Jim and Pam tick as a couple. Use a line of UNIX commands to create a new dataset called `jim_and_pam.csv`. Include only lines that are spoken by either Jim or Pam, or reference Jim or Pam in any way. Include only the following columns: `episode_name`, `character`, `text`, `text_w_direction`, and `air_date`. How many rows of data are in the new file? How many megabytes is the new file (to the nearest 1/10th of a megabyte)?

Hint: *Redirection.*

Hint: *It is OK if you get an erroneous line where the word “jim” or “pam” appears as a part of another word.*

Relevant topics: *cut, grep, ls, wc, redirection*

Item(s) to submit:

- The line of UNIX commands used to create the new file.
- The number of rows of data in the new file, and the accompanying UNIX command used to find this out.
- The number of megabytes (to the nearest 1/10th of a megabyte) that the new file has, and the accompanying UNIX command used to find this out.

4. Find all lines where either Jim/Pam/Michael/Dwight’s name is followed by an exclamation mark. Use only 1 “!” within your regular expression. How many lines are there?

Relevant topics: *grep*

Item(s) to submit:

- The UNIX command(s) used to solve this problem.
- The number of lines where either Jim/Pam/Michael/Dwight’s name is followed by an exclamation mark.

5. Find all lines that contain the text “that’s what” followed by any amount of any text and then “said”. How many lines are there?

Relevant topics: *grep*

Item(s) to submit:

- The UNIX command used to solve this problem.
- The number of lines that contain the text “that’s what” followed by any amount of text and then “said”.

6. Find all of the lines where Pam is called “Beesley” instead of “Pam” or “Pam Beesley”.

Hint: *A negative lookbehind would be one way to solve this.*

Relevant topics: *grep*

Item(s) to submit:

- The UNIX command used to solve this problem.

Regular expressions are really a useful semi language-agnostic tool. What this means is regardless of the programming language you are using, there will be some package that allows you to use regular expressions. In fact, we can use them in both R and Python! This can be particularly useful when dealing with strings. Load up the dataset you discovered in (1) using `read.csv`. Name the resulting `data.frame` `dat`.

7. The `text_w_direction` column in `dat` contains the characters’ lines with inserted direction that helps characters know what to do as they are reciting the lines. Direction is shown between square brackets “[” “]”. Create a new column called `has_direction` that is set to `TRUE` if the `text_w_direction` column has direction, and `FALSE` otherwise. Use regular expressions and the `grepl` function in R to accomplish this.

Hint: *Make sure all opening brackets “[” have a corresponding closing bracket “]”.*

Hint: *Think of the pattern as any line that has a [, followed by any amount of any text, followed by a], followed by any amount of any text.*

Relevant topics: *grep, grepl*

Item(s) to submit:

- The R code used to solve this problem.

8. Modify your regular expression in (7) to find lines with 2 or more sets of direction.

For example, the following line has 2 directions: `dat$text_w_direction[2789]`.
How many lines have more than 2 directions? How many have more than 5?

This is a line with [emphasize this] only 1 direction!

This is a line with [emphasize this] 2 sets of direction, do you see the difference [shrug].

In (7), your solution may have found a match in both lines. In this question we want it to find only lines with 2+ directions, so the first line would not be a match.

Relevant topics: *length, grep*

Item(s) to submit:

- The R code used to solve this problem.
- How many lines have > 2 directions?
- How many lines have > 5 directions?

9. Use the `str_extract_all` function from the `stringr` package to extract the direction(s) as well as the text between direction(s) from each line. Put the strings in a new column called `direction`.

This is a line with [emphasize this] only 1 direction!

This is a line with [emphasize this] 2 sets of direction, do you see the difference [shrug].

In this question, your solution may have extracted:

[emphasize this]

[emphasize this] 2 sets of direction, do you see the difference [shrug]

This is ok.

Note: If you capture text between two sets of direction, this is ok. For example, if we capture “[this] is a [test]” from “if we capture [this] is a [test]”, this is ok.

Relevant topics: *str_extract_all*

Item(s) to submit:

- The R code used to solve this problem.

10. Repeat (9) but this time make sure you only capture the brackets and text within the brackets. Save the results in a new column called `direction_correct`. You can test to see if it is working by running the following code:

```
dat$direction_correct[747]
```

This is a line with [emphasize this] only 1 direction!

This is a line with [emphasize this] 2 sets of direction, do you see the difference [shrug]

In (7), your solution may have extracted:

```
[emphasize this]
```

```
[emphasize this] 2 sets of direction, do you see the difference [shrug]
```

This is ok for (7). In this question, however, we want to fix this to only extract:

```
[emphasize this]
```

```
[emphasize this] [shrug]
```

Hint: *This regular expression will be hard to read.*

Hint: *The pattern we want is: literal opening bracket, followed by 0+ of any character other than the literal [or literal], followed by a literal closing bracket.*

Relevant topics: `str_extract_all`

Item(s) to submit:

- The R code used to solve this problem.

Project 5 {#390-p05}

Motivation: Becoming comfortable stringing together commands and getting used to navigating files in a terminal is important for every data scientist to do. By learning the basics of a few useful tools, you will have the ability to quickly understand and manipulate files in a way which is just not possible using tools like Microsoft Office, Google Sheets, etc.

Context: We've been using UNIX tools in a terminal to solve a variety of problems. In this project we will continue to solve problems by combining a variety of tools using a form of redirection called piping.

Scope: grep, regular expression basics, UNIX utilities, redirection, piping

Learning objectives:

- Use `cut` to section off and slice up data from the command line.
- Use piping to string UNIX commands together.
- Use `sort` and it's options to sort data in different ways.
- Use `head` to isolate n lines of output.
- Use `wc` to summarize the number of lines in a file or in output.
- Use `uniq` to filter out non-unique lines.
- Use `grep` to search files effectively.

You can find useful examples that walk you through relevant material in The Examples Book:

<https://thedatamine.github.io/the-examples-book>

It is highly recommended to read through, search, and explore these examples to help solve problems in this project.

Important note: It is highly recommended that you use <https://rstudio.scholar.rcac.purdue.edu/>. Simply click on the link and login using your Purdue account credentials. Use another system at your own risk. The version of RStudio on <https://desktop.scholar.rcac.purdue.edu/> (which uses ThinLinc), is 99.9.9, and is known to have some strange issues when running code chunks.

Don't forget the very useful documentation shortcut `?<function>`. To use, simply type `?<function>` in the console, followed by the name of the function you are interested in.

You can also look for package documentation by using `help(package=PACKAGENAME)`, so for example, to see the documentation for the package `ggplot2`, we could run:

```
help(package=ggplot2)
```

Sometimes it can be helpful to see the source code of a defined function. A function is any chunk of organized code that is used to perform an operation. Source code is the underlying R or c or c++ code that is used to create the function. To see the source code of a defined function, type the function's name without the (). For example, if we were curious about what the function `Reduce` does, we could run:

```
Reduce
```

Occasionally this will be less useful as the resulting code will be code that calls c code we can't see. Other times it will allow you to understand the function better.

Dataset

The following questions will use the dataset found in Scholar:

```
/class/datamine/data/amazon/amazon_fine_food_reviews.csv
```

A public sample of the data can be found here: `amazon_fine_food_reviews.csv`

Answers to questions should all be answered using the full dataset located on Scholar. You may use the public samples of data to experiment with your solutions prior to running them using the full dataset.

Questions

1. What is the Id of the most helpful review if we consider the review with highest HelpfulnessNumerator to be an indicator of helpfulness (higher is more helpful)?

Relevant topics: *cut, sort, head, piping*

Item(s) to submit:

- Line of UNIX commands used to solve the problem.
- The Id of the most helpful review.

2. What proportion of all Summaries are unique? Use two lines of UNIX commands to find the answer.

Relevant topics: *cut, uniq, sort, wc, piping*

Item(s) to submit:

- Two lines of UNIX commands used to solve the problem.
- The ratio of unique `Summary`'s.

3. Use a simple UNIX command to create a frequency table of `Score`.

Relevant topics: *cut, uniq, sort, piping*

Item(s) to submit:

- The line of UNIX commands used to solve the problem.
- The frequency table.

4. Who is the user with the highest number of reviews? There are two columns you could use to answer this question, but which column do you think would be most appropriate and why?

Hint: *You may need to pipe the output to `sort` multiple times.*

Hint: *To create the frequency table, read through the `man` pages for `uniq`. `Man` pages are the “manual” pages for UNIX commands. You can read through the `man` pages for `uniq` by running the following:*

```
man uniq
```

Relevant topics: *cut, uniq, sort, head, piping, man*

Item(s) to submit:

- The line of UNIX commands used to solve the problem.
- The frequency table.

5. Anecdotally, there seems to be a tendency to leave reviews when we feel strongly (either positive or negative) about a product. For the user with the highest number of reviews, would you say that they follow this pattern of extremes? Let's consider 5 star reviews to be strongly positive and 1 star reviews to be strongly negative. Let's consider anything in between neither strongly positive nor negative.

Hint: *You may find the solution to problem (3) useful.*

Relevant topics: *cut, uniq, sort, grep, piping*

Item(s) to submit:

- The line of UNIX commands used to solve the problem.

6. We want to compare the most helpful review with a Score of 5 with the most helpful review with a Score of 1. Use UNIX commands to calculate these values. Write down the ProductId of both reviews. In the case of a tie, write down all ProductId's to get full credit. In this case we are considering the most helpful review to be the review with the highest HelpfulnessNumerator.

Hint: *You can use multiple lines to solve this problem.*

Relevant topics: *sort, head, piping*

Item(s) to submit:

- The lines of UNIX commands used to solve the problem.
- ProductId's of both requested reviews.

7. Using the ProductId's from the previous question, create a new dataset called `reviews.csv` which contains the ProductId's and Score of all reviews with the corresponding ProductId's.

Relevant topics: *cut, grep, redirection*

Item(s) to submit:

- The line of UNIX commands used to solve the problem.

8. If we didn't use `cut` prior to searching for the ProductId's in (7), we would get unwanted results. Modify the solution to (7) and explore. What is happening?

Relevant topics: *cat, grep, redirection*

Item(s) to submit:

- The line of UNIX commands used to solve the problem.
- 1-2 sentences explaining why we need to use `cut` first.
- 1-2 sentences explaining whether or not you think people found the review helpful because the produce is overrated, underrated, or correctly reviewed, and why.

9. Use R to load up `reviews.csv` into a new `data.frame` called `dat`. Create a histogram for each products' Score. Compare the most helpful review Score with the Score's given in the histogram. Based on this comparison, decide (anecdotally) whether you think people found the review helpful because the product is overrated, underrated, or correctly reviewed by the masses.

Relevant topics: *read.csv*, *hist*

Item(s) to submit:

- R code used to create the histograms.
- 3 histograms, 1 for each `ProductId`.

Project 6 {#390-p06}

Motivation: A bash script is a powerful tool to perform repeated tasks. RCAC uses bash scripts to automate a variety of tasks. In fact, we use bash scripts on Scholar to do things like link Python kernels to your account, fix potential issues with Firefox, etc. `awk` is a programming language designed for text processing. The combination of these tools can be really powerful and useful for a variety of quick tasks.

Context: This is the first part in a series of projects that are designed to exercise skills around UNIX utilities, with a focus on writing bash scripts and `awk`. You will get the opportunity to manipulate data without leaving the terminal. At first it may seem overwhelming, however, with just a little practice you will be able to accomplish data wrangling tasks really efficiently.

Scope: `awk`, UNIX utilities, bash scripts

Learning objectives:

- Use `awk` to process and manipulate textual data.
- Use piping and redirection within the terminal to pass around data between utilities.
- Use output created from the terminal to create a plot using R.

Dataset:

The following questions will use the dataset found in Scholar: `/class/datamine/data/flights/subset/`.
An example of the data for the year 1987 can be found [here](#).

Questions

1. In previous projects we learned how to get a single column of data from a csv file. Write 1 line of UNIX commands to print the 17th column, the `Origin`, from `1987.csv`. Write another line, this time using `awk` to do the same thing. Which one do you prefer, and why?

Relevant topics: `cut`, `awk`

Item(s) to submit:

- One line of UNIX commands to solve the problem *without* using `awk`.
- One line of UNIX commands to solve the problem using `awk`.
- 1-2 sentences describing which method you prefer and why.

2. Write a bash script that accepts a year (1987, 1988, etc.) and a column *n* and returns the *n*th column of the associated year of data.

Relevant topics: `awk`, bash scripts

Item(s) to submit:

- The content of your bash script (starting with “`#!/bin/bash`”) in a code chunk.

3. How many flights came into Indianapolis (IND) in 2008? First solve this problem without using `awk`, then solve this problem using *only* `awk`.

Relevant topics: cut, grep, wc, awk, piping

Item(s) to submit:

- One line of UNIX commands to solve the problem *without* using `awk`.
- One line of UNIX commands to solve the problem using `awk`.
- The number of flights that came into Indianapolis (IND) in 2008.

4. Do you expect the number of unique origins and destinations to be the same? Find out using any command line tool you'd like. Are they indeed the same? How many unique values do we have per category (Origin, Dest)?

Relevant topics: cut, sort, uniq, wc, awk

Item(s) to submit:

- 1-2 sentences explaining whether or not you expect the number of unique origins and destinations to be the same.
- The UNIX command(s) used to figure out if the number of unique origins and destinations are the same.
- The number of unique values per category (Origin, Dest).

5. In (4) we found that there are not the same number of unique Origin's as Dest's. Find the IATA airport code for all Origin's that dont appear in a Dest and all Dest's that don't appear in an Origin.

Hint: https://www.tutorialspoint.com/unix_commands/comm.htm

Relevant topics: comm, cut, sort, uniq, redirection

Item(s) to submit:

- The line(s) of UNIX command(s) used to answer the question.
- The list of Origin's that don't appear in Dest.
- The list of Dest's that don't appear in Origin.

6. What was the average number of flights in 2008 per unique Origin with the Dest of "IND"? How does "PHX" (as a unique Origin) compare to the average?

Hint: You manually do the average calculation by dividing the result from (3) by the number of unique Origin's that have a Dest of "IND".

Relevant topics: awk, sort, grep, wc

Item(s) to submit:

- The average number of flights in 2008 per unique `Origin` with the `Dest` of “IND”.
- 1-2 sentences explaining how “PHX” compares (as a unique `Origin`) to the average?

7. Write a bash script that takes a year and IATA airport code and returns the year, and the total number of flights to and from the given airport. Example rows may look like:

```
1987, 12345
1988, 44
```

Run the script with inputs: 1991 and ORD. Include the output in your submission.

Relevant topics: bash scripts, cut, piping, grep, wc

Item(s) to submit:

- The content of your bash script (starting with “#!/bin/bash”) in a code chunk.
- The output of the script given 1991 and ORD as inputs.

8. Pick your favorite airport and get its IATA airport code. Write a bash script that, given the first year, last year, and airport code, runs the bash script from (7) for all years in the provided range for your given airport, or loops through all of the files for the given airport, appending all of the data to a new file called `my_airport.csv`.

Relevant topics: bash scripts, cut, grep, wc, for loops, echo, redirection

Item(s) to submit:

- The content of your bash script (starting with “#!/bin/bash”) in a code chunk.

9. In R, load `my_airport.csv` and create a line plot showing the year-by-year change. Label your x-axis “Year”, your y-axis “Num Flights”, and your title the name of the IATA airport code. Write 1-2 sentences with your observations.

Relevant topics: `read.csv`, `lines`

Item(s) to submit:

- Line chart showing year-by-year change in flights into and out of the chosen airport.
- R code used to create the chart.
- 1-2 sentences with your observations.

Project 7 {#390-p07}

Motivation: A bash script is a powerful tool to perform repeated tasks. RCAC uses bash scripts to automate a variety of tasks. In fact, we use bash scripts on Scholar to do things like link Python kernels to your account, fix potential issues with Firefox, etc. `awk` is a programming language designed for text processing. The combination of these tools can be really powerful and useful for a variety of quick tasks.

Context: This is the first part in a series of projects that are designed to exercise skills around UNIX utilities, with a focus on writing bash scripts and `awk`. You will get the opportunity to manipulate data without leaving the terminal. At first it may seem overwhelming, however, with just a little practice you will be able to accomplish data wrangling tasks really efficiently.

Scope: `awk`, UNIX utilities, bash scripts

Learning objectives:

- Use `awk` to process and manipulate textual data.
- Use piping and redirection within the terminal to pass around data between utilities.

Dataset:

The following questions will use the dataset found in Scholar:

```
/class/datamine/data/flights/subset/YYYY.csv
```

An example of the data for the year 1987 can be found [here](#).

Sometimes if you are about to dig into a dataset, it is good to quickly do some sanity checks early on to make sure the data is what you expect it to be.

Questions

1. Write a line of code that prints a list of the unique values in the `DayOfWeek` column. Write a line of code that prints a list of the unique values in the `DayOfMonth` column. Write a line of code that prints a list of the unique values in the `Month` column. Use the `1987.csv` dataset. Are the results what you expected?

Relevant topics: `cut`, `sort`

Item(s) to submit:

- 3 lines of code used to get a list of unique values for the chosen columns.
- 1-2 sentences explaining whether or not the results are what you expected.

2. Our files should have 29 columns. Write a line of code that prints any lines in a file that do *not* have 29 columns. Test it on `1987.csv`, were there any rows without 29 columns?

Relevant topics: `awk`

Item(s) to submit:

- Line of code used to solve the problem.
- 1-2 sentences explaining whether or not there were any rows without 29 columns.

3. Write a bash script that, given a “begin” year and “end” year, cycles through the associated files and prints any lines that do *not* have 29 columns.

Relevant topics: `awk`, bash scripts

Item(s) to submit:

- The content of your bash script (starting with “#!/bin/bash”) in a code chunk.
- The results of running your bash scripts from year 1987 to 2008.

4. `awk` is a really good tool to quickly get some data and manipulate it a little bit. For example, let’s see the number of kilometers and miles traveled in 1990. To convert from miles to kilometers, simply multiply by 1.609344.

Example output:

```
Miles: 12345
Kilometers: 19867.35168
```

Relevant topics: `awk`, piping

Item(s) to submit:

- The code used to solve the problem.
- The results of running the code.

5. Use `awk` to calculate the number of `DepDelay` minutes by `DayOfWeek`. Use `2007.csv`.

Example output:

```
DayOfWeek: 0
1: 1234567
2: 1234567
3: 1234567
4: 1234567
5: 1234567
6: 1234567
7: 1234567
```

Note: 1 is Monday.

Relevant topics: `awk`, sort, piping

Item(s) to submit:

- The code used to solve the problem.
- The output from running the code.

6. It wouldn't be fair to compare the total DepDelay minutes by DayOfWeek as the number of flights may vary. One way to take this into account is to instead calculate an average. Modify (5) to calculate the average number of DepDelay minutes by the number of flights per DayOfWeek. Use 2007.csv.

Example output:

```
DayOfWeek: 0
1: 1.234567
2: 1.234567
3: 1.234567
4: 1.234567
5: 1.234567
6: 1.234567
7: 1.234567
```

Relevant topics: awk, sort, piping

Item(s) to submit:

- The code used to solve the problem.
- The output from running the code.

7. As a quick follow-up, *slightly* modify (6) to perform the same calculation for ArrDelay. Do the ArrDelays and DepDelays appear to have the highest delays on the same day? Use 2007.csv.

Example output:

```
DayOfWeek: 0
1: 1.234567
2: 1.234567
3: 1.234567
4: 1.234567
5: 1.234567
6: 1.234567
7: 1.234567
```

Relevant topics: awk, sort, piping

Item(s) to submit:

- The code used to solve the problem.
- The output from running the code.
- 1-2 sentences explaining whether or not the ArrDelays and DepDelays appear to have the highest delays on the same day.

8. Anyone who has flown knows how frustrating it can be waiting for takeoff, or deboarding the aircraft. These roughly translate to TaxiOut and TaxiIn respectively. If you were to fly into or out of IND what is your expected total taxi time? Use 2007.csv.

Note: Taxi times are in minutes.

Relevant topics: awk, grep

Item(s) to submit:

- The code used to solve the problem.
- The output from running the code.

9. What are the IATA airport codes of the 5 airports with the greatest total taxi time for 2007? Show the total taxi time for each.

Example output:

```
DayOfWeek: 0
IND: 1234567
IND: 1234567
IND: 1234567
IND: 1234567
IND: 1234567
```

Relevant topics: awk, head, sort

Item(s) to submit:

- The code used to solve the problem.
 - The output from running the code.
-

Project 8 {#390-p08}

Motivation: A bash script is a powerful tool to perform repeated tasks. RCAC uses bash scripts to automate a variety of tasks. In fact, we use bash scripts on Scholar to do things like link Python kernels to your account, fix potential issues with Firefox, etc. **awk** is a programming language designed for text processing. The combination of these tools can be really powerful and useful for a variety of quick tasks.

Context: This is the first part in a series of projects that are designed to exercise skills around UNIX utilities, with a focus on writing bash scripts and **awk**. You will get the opportunity to manipulate data without leaving the terminal. At first it may seem overwhelming, however, with just a little practice you will be able to accomplish data wrangling tasks really efficiently.

Scope: awk, UNIX utilities, bash scripts

Learning objectives:

- Use **awk** to process and manipulate textual data.
- Use piping and redirection within the terminal to pass around data between utilities.

Dataset:

The following questions will use the dataset found in Scholar:

`/class/datamine/data/flights/subset/YYYY.csv`

An example of the data for the year 1987 can be found [here](#).

Let's say we have a theory that there are more flights on the weekend days (Friday, Saturday, Sunday) than the rest of the days, on average. We can use **awk** to quickly check it out and see if maybe this looks like something that is true!

1. Write a line of **awk code that, prints the number of flights on the weekend days, followed by the number of flights on the weekdays for the flights during 2008.**

Relevant topics: awk

Item(s) to submit:

- Line of `awk` code that solves the problem.
- The result: the number of flights on the weekend days, followed by the number of flights on the weekdays for the flights during 2008.

2. Note that in (1), we are comparing 3 days to 4! Write a line of `awk` code that, prints the average number of flights on a weekend day, followed by the average number of flights on the weekdays. Continue to use data for 2008.

Relevant topics: `awk`

Item(s) to submit:

- Line of `awk` code that solves the problem.
- The result: the average number of flights on the weekend days, followed by the average number of flights on the weekdays for the flights during 2008.

We want to look to see if there may be some truth to the whole “snow bird” concept where people will travel to warmer states like Florida and Arizona during the Winter. Let’s use the tools we’ve learned to explore this a little bit.

3. Take a look at `airports.csv`. In particular run the following:

```
head airports.csv
```

Notice how all of the non-numeric text is surrounded by quotes. The surrounding quotes would need to be escaped for any comparison within `awk`. This is messy and we would prefer to create a new file called `new_airports.csv` without any quotes. Write a line of code to do this.

Hint: You could use `gsub` within `awk` to replace “” with “”.

Hint: If you leave out the column number argument to `gsub` it will apply the substitution to every field in every column.

Relevant topics: `awk`, redirection

Item(s) to submit:

- Line of `awk` code used to create the new dataset.

4. Write a line of commands that create a new dataset called `az_fl_airports.txt` that contains a list of airport codes for all airports from both Arizona (AZ) and Florida (FL). Use the file we created in (3), `new_airports.csv`.

Relevant topics: `awk`

Item(s) to submit:

- The line of UNIX commands to create an array called `airports`.

5. Wow! In (4) we discovered a lot of airports! How many airports are there? Did you expect this? Use a line of bash code to answer this question.

Relevant topics: `echo`, `wc`, piping

Item(s) to submit:

- Line of UNIX commands used to solve the problem.
- The number of airports.
- 1-2 sentences explaining whether you expected this result and why or why not.

6. Create a new dataset that contains all of the data for flights into or out of Florida and Arizona using `2008.csv`, use the newly created dataset, `az_fl_airports.txt` in (4) to do so.

Hint: <https://unix.stackexchange.com/questions/293684/basic-grep-awk-help-extracting-all-lines-containing-a-list-of-terms-from-one-f>

Relevant topics: `grep`

Item(s) to submit:

- Line of UNIX commands used to solve the problem.

7. Now that you have code to complete (6), write a bash script that accepts the start year, end year, and filename containing airport codes (`az_fl_airports.txt`), and outputs the data for flights into or out of any of the airports listed in the provided filename containing airport codes using *all* of the years of data in the provided range. Run the bash script to create a new file called `az_fl_flights.csv`.

Relevant topics: bash scripts, grep, for loop, redirection

Item(s) to submit:

- The content of your bash script (starting with “#!/bin/bash”) in a code chunk.
- The line of UNIX code you used to execute the script and create the new dataset.

8. Use the newly created `az_fl_flights.csv` dataset to calculate the total number of flights into and out of both states by month, and by year, for a total of 3 columns (year, month, flights). Export this information to a new file called `snowbirds.csv`.

Relevant topics: awk, redirection

Item(s) to submit:

- The line of `awk` code used to create the new dataset, `snowbirds.csv`.

9. Load up your newly created dataset and use either R or Python (or some other tool) to create a graphic that illustrates whether or not we believe the “snowbird effect” effects flights. Include a description of your graph, as well as your (anecdotal) conclusion.

Item(s) to submit:

- Code used to create the visualization in a code chunk.
- The generated plot as either a png or jpg/jpeg.
- 1-2 sentences describing your plot and your conclusion.

Project 9 {#390-p09}

Project 10 {#390-p10}

Project 11 {#390-p11}

Motivation: Being able to use results of queries as tables in new queries (also known as writing sub-queries), and calculating values like MIN, MAX, and AVG in aggregate are key skills to have in order to write more complex queries. In this project we will learn about aliasing, writing sub-queries, and calculating aggregate values.

Context: We are in the middle of a series of projects focused on working with databases and SQL. In this project we introduce aliasing, sub-queries, and calculating aggregate values using a much larger dataset!

Scope: sql, sql in R

Learning objectives:

- Demonstrate the ability to interact with popular database management systems within R.
- Solve data-driven problems using a combination of SQL and R.
- Basic clauses: select, order by, limit, desc, asc, count, where, from, etc.
- Showcase the ability to filter, alias, and write subqueries.
- Perform grouping and aggregate data using group by and the following functions: count, max, sum, avg, like, having. Explain when to use having, and when to use where.

Dataset

elections database & /class/datamine/data/election/itcontYYYY.txt
(for example, data for year 1980 would be /class/datamine/data/election/itcont1980.txt)

A public sample of the data can be found here:

<https://www.datadepot.rcac.purdue.edu/datamine/data/election/itcontYYYY.txt>
(for example, data for year 1980 would be <https://www.datadepot.rcac.purdue.edu/datamine/data/election/itcont1980.txt>)

Up until now, you've been working with a neatly organized database containing baseball data. As fantastic as this database is, it would be trivial to load up the entire database in R or Python and do your analysis using `merge`-like functions. Now, we are going to deal with a much larger set of data.

1. Approximately how large was the lahman database (use the sqlite database in Scholar: /class/datamine/data/lahman/lahman.db)? Use UNIX utilities you've learned about this semester to write a line of code to return the amount of data (in MB) in the elections folder /class/datamine/data/election/. How much data (in MB) is there?

The data in that folder has been added to the `elections` database in the `elections` table. Write a SQL query that returns how many rows of data are in the database. How many rows of data are in the database?

Hint: This will take some time! Be patient.

Relevant topics: sql, sql in R, awk, ls

Item(s) to submit:

- Approximate size of the lahman database in mb.
- Line of code (bash/awk) to calculate the size (in mb) of the entire elections dataset in `/class/datamine/data/election`.
- The size of the elections data in mb.
- SQL query used to find the number of rows of data in the `elections` table in the `elections` database.
- The number of rows in the `elections` table in the `elections` database.

2. Write a SQL query using the LIKE command to find a unique list of zip_codes that start with “479”. How many unique zip_codes are there that begin with “479”?

Hint: Make sure you only select `zip_codes`.

Relevant topics: sql, like

Item(s) to submit:

- SQL queries used to answer the question.
- The first 5 results from running the query.

3. Write a SQL query that counts the number of donations (rows) that are from Indiana. How many donations are from Indiana? Rewrite the query and create an *alias* for our field so it doesn’t read COUNT(*) but rather Indiana Donations.

Relevant topics: sql, where, aliasing

Item(s) to submit:

- SQL query used to answer the question.
- The result of the SQL query.

4. Rewrite the query in (3) so the result is displayed like the following:

```
+-----+
| Donations |
+-----+
| IN: 1111778 |
+-----+
```

Hint: Use CONCAT and aliasing to accomplish this.

Relevant topics: sql, aliasing, concat

Item(s) to submit:

- SQL query used to answer the question.

5. In (2) we wrote a query that returns a unique list of `zip_codes` that start with “479”. In (3) we wrote a query that counts the number of donations that are from Indiana. Use our query from (2) as a subquery to find how many donations come from areas with `zip_codes` starting with “479”. What percent of donations in Indiana come from said `zip_codes`?

Relevant topics: sql, aliasing, subqueries

Item(s) to submit:

- SQL queries used to answer the question.
- The percentage of donations from Indiana from `zip_codes` starting with “479”.

6. In (3) we wrote a query that counts the number of donations that are from Indiana. When running queries like this, a natural “next question” is to ask the same question about another state. SQL gives us the ability to calculate functions in aggregate when grouping by a certain column. Write a SQL query that returns the state, number of donations from each state, the sum of the donations (`transaction_amt`). Which 5 states gave the most donations (highest count)? Order you result from most to least.

Hint: You may want to create an alias in order to sort.

Relevant topics: sql, group by

Item(s) to submit:

- SQL query used to answer the question.
- Which 5 states gave the most donations?

7. Write a query that gets the number of donations, and sum of donations, by year, for Indiana. Create one or more graphics that highlights the year-by-year changes. Write a short 1-2 sentences explaining your graphic(s).

Relevant topics: sql in R, group by

Item(s) to submit:

- SQL query used to answer the question.
- R code used to create your graphic(s).
- 1 or more graphics in png/jpeg format.
- 1-2 sentences summarizing your graphic(s).

Project 12 {#390-p12}

Project 13

Motivation: Databases you will work with won't necessarily come organized in the way that you like. Getting really comfortable writing longer queries where you have to perform many joins, alias fields and tables, and aggregate results, is important. In addition, gaining some familiarity with terms like *primary key*, and *foreign key* will prove useful when you need to search for help online. In this project we will write some more complicated queries with a fun database. Proper preparation prevents poor performance, and that means practice!

Context: We are towards the end of a series of projects that give you an opportunity to practice using SQL. In this project, we will reinforce topics you've already learned, with a focus on subqueries and joins.

Scope: SQL, sqlite

Learning objectives:

- Write and run SQL queries in `sqlite` on real-world data.
- Identify primary and foreign keys in a SQL table.

Dataset

`/class/datamine/data/movies_and_tv/imdb.db`

A public sample of the data can be found [here](#).

Questions

1. A primary key is a field in a table which uniquely identifies a row in the table. Primary keys *must* be unique values, and this is enforced at the database level. A foreign key is a field whose value matches a primary key in a different table. A table can have 0-1 primary key, but it can have 0+ foreign keys. Examine the `titles` table. Do you think there are any primary keys? How about foreign keys?

Relevant topics: primary key, foreign key

Item(s) to submit:

- List any primary or foreign keys in the `episodes` table.

2. Examine the `episodes` table. Based on observation and the column names, do you think there are any primary keys? How about foreign keys?

Relevant topics: primary key, foreign key

Item(s) to submit:

- List any primary or foreign keys in the `episodes` table.

If you paste a `title_id` to the end of the following url, it will pull up the page for the title. For example, <https://www.imdb.com/title/tt0413573> leads to the page for the TV series *Grey's Anatomy*.

3. Write a query to confirm that the `title_id` `tt0413573` does indeed belong to *Grey's Anatomy*.

Relevant topics: select, where

Item(s) to submit:

- SQL query used to solve the problem in a code chunk.
- Output of the query.

4. The `episode_title_id` column in the `episodes` table references titles of individual episodes of a tv series. The `show_title_id` references the titles of the show itself. With that in mind, write a query that gets a list of all of the episodes and titles of Grey's Anatomy.

Relevant topics: inner join

Item(s) to submit:

- SQL query used to solve the problem in a code chunk.

5. Like we explained in (3), you can find the `title_id` of a tv show, a tv show episodes, or a movie by browsing `imdb.com` and getting the `title_id` directly from the url. Browse `imdb.com` and find your favorite tv show. Get the `title_id` from the url and run the following query to confirm that the tv show is in our database:

```
SELECT * FROM titles WHERE title_id='<title id here>';
```

Make sure to replace '

' with the `title_id` of your favorite show. If your show does not appear, or has only a single season, pick another show until you find one we have in our database with multiple seasons.

Item(s) to submit:

- The `title_id` of your favorite tv show.
- The output from running the provided (modified) query.

6. We want to write a query that returns the title and rating of the highest rated episode of the tv show you chose in (5). In order to do so, first write a query that returns a list of `episode_title_ids` (found in the `episodes` table), with the `primary_title` (found in the `titles` table) of the episode.

Relevant topics: inner join, aliasing

Item(s) to submit:

- SQL query used to solve the problem in a code chunk.
- The first 5 results from your query.

7. Write a query that adds the rating to the end of each episode. To do so, use the query you wrote in (6) as a subquery. Was this also your favorite episode?

Relevant topics: inner join, aliasing, subqueries, desc, limit, order by

Item(s) to submit:

- SQL query used to solve the problem in a code chunk.
- The `episode_title_id`, `primary_title`, and `rating` of the top rated episode from the tv series from (5).
- A statement saying whether it is also your favorite episode.

8. Write a query that returns the `season_number` (from the `episodes` table), and average `rating` (from the `ratings` table) for each season. Write another query that only returns the season number and `rating` for the highest rated season. Consider the highest rated season the season with the highest average.

Relevant topics: inner join, aliasing, group by, having, avg

Item(s) to submit:

- The 2 SQL queries used to solve the problems in a code chunk.

9. Write a query that returns the `primary_title`, and `rating` of the highest rated episode per season for your tv show from (5).

Relevant topics: max, subqueries, group by, having, inner join, aliasing

Item(s) to submit:

- The SQL query used to solve the problem.
 - The output from your query.
 - 1-2 sentences explaining whether or not you agree.
-

Project 14 {#390-p14}**Project 15 {#390-p15}**

Motivation: We've done a lot of work with SQL this semester. Let's review concepts in this project and mix and match R, Python, and SQL to solve data-driven problems.

Context: In this project, we will reinforce topics you've already learned, with a focus on SQL.

Scope: SQL, sqlite, R, Python

Learning objectives:

- Write and run SQL queries in `sqlite` on real-world data.
- Use SQL from within R and Python.

Dataset

```
/class/datamine/data/movies_and_tv/imdb.db
```

A public sample of the data can be found [here](#).

In this project we want to offer the flexibility of using your choice of R and/or Python. To keep things as consistent as possible, please use Rmarkdown on <https://rstudio.scholar.rcac.purdue.edu/>. See [here](#) to learn how to run Python in this environment.

F.R.I.E.N.D.S is a popular tv show. They have an interesting naming convention for the names of their episodes. They all begin with the text "The One ...". There are 6 primary characters in the show: Chandler, Joey, Monica, Phoebe, Rachel, and Ross. Let's use SQL and R to take a look at how many times each characters' names appear in the title of the episodes.

Questions

1. Write a query that gets the `episode_title_id`, `primary_title`, `rating`, and `votes`, of all of the episodes of Friends (`title_id` is `tt0108778`).

Hint: You can slightly modify the solution to question (7) in project 13.

Relevant topics: inner join, subqueries, aliasing

Item(s) to submit:

- SQL query used to answer the question.
- First 5 results of the query.

The next couple of questions should be complete in the same language. You can use either R or Python, but you must use the same for both questions.

2. Now that you have a working query, connect to the database and run the query to get the data into an R or pandas data frame. In previous projects, we learned how to use regular expressions to search for text. For each character, how many episodes primary_titles contained their name?

Relevant topics: SQL in R, SQL in Python, grep

Item(s) to submit:

- R or Python code in a code chunk that was used to find the solution.
- The solution pasted below the code chunk.

3. Create a graphic showing our results in (2) using your favorite package. Make sure the plot has a good title, x-label, y-label, and try to incorporate some of the following colors: #273c8b, #bd253a, #016f7c, #f56934, #016c5a, #9055b1, #eaab37.

Relevant topics: plotting

Item(s) to submit:

- The R or Python code used to generate the graphic.
- The graphic in a png or jpg/jpeg format.

4. Use any combination of SQL, R, and Python you'd like in order to find which of the following 3 genres has the highest average rating for movies (see type column from titles table): Romance, Comedy, Animation. In the titles table, you can find the genres in the genres column. There may be some overlap (i.e. a movie may have more than one genre), this is ok.

To query rows which have the genre Action as one of its genres:

```
SELECT * FROM titles WHERE genres LIKE '%action%';
```

Relevant topics: like, inner join

Item(s) to submit:

- Any code you used to solve the problem in a code chunk.
- The average rating of each of the genres listed for movies.

5. Write a function called `top_episode` in R or in Python which accepts the path to the `imdb.db` database, as well as the `title_id` of a tv series (for example, “tt0108778” or “tt1266020”), and returns the `season_number`, `episode_number`, `primary_title`, and `rating` of the highest rated episode in the series. Test it out on some of your favorite series, and share the results.

Relevant topics: functions, inner join, order by

Item(s) to submit:

- Any code you used to solve the problem in a code chunk.
- The results for at least 3 of your favorite tv series.

Chapter 10

Think Summer 2020

Project

Submission

Students need to submit an RMarkdown file with all of the required code and output by **Wednesday, July 8th at 12:00 PM EST** through Gradescope inside Brightspace.

You can find an Rmarkdown template which you can modify and use a starting point for your project [here](#), and the resulting, compiled PDF [here](#).

Motivation: SQL is an incredibly powerful tool that allows you to process and filter massive amounts of data – amounts of data where tools like spreadsheets start to fail. You can perform SQL queries directly within the R environment, and doing so allows you to quickly perform ad-hoc analyses.

Context: This project is specially designed for Purdue University’s Think Summer program, in conjunction with Purdue University’s integrative data science initiative, The Data Mine.

Scope: SQL, SQL in R

Learning objectives:

- Demonstrate the ability to interact with popular database management systems within R.
- Solve data-driven problems using a combination of SQL and R.
- Use basic SQL commands: select, order by, limit, desc, asc, count, where, from.
- Perform grouping and aggregate data using group by and the following functions: count, max, sum, avg, like, having.

You can find useful examples that walk you through relevant material in The Examples Book:

<https://thedatamine.github.io/the-examples-book>

It is highly recommended to read through, search, and explore these examples to help solve problems in this project.

Important note: It is highly recommended that you use <https://rstudio.scholar.rcac.purdue.edu/>. Simply click on the link and login using your Purdue account credentials. Use another system at your own risk. The version of RStudio on <https://desktop.scholar.rcac.purdue.edu/> (which uses ThinLinc), is 99.9.9, and is known to have some strange issues when running code chunks.

Don't forget the very useful documentation shortcut `?`. To use, simply type `?` in the console, followed by the name of the function you are interested in.

You can also look for package documentation by using `help(package=PACKAGENAME)`, so for example, to see the documentation for the package `ggplot2`, we could run:

```
help(package=ggplot2)
```

Sometimes it can be helpful to see the source code of a defined function. A function is any chunk of organized code that is used to perform an operation. Source code is the underlying R or C or C++ code that is used to create the function. To see the source code of a defined function, type the function's name without the `()`. For example, if we were curious about what the function `Reduce` does, we could run:

```
Reduce
```

Occasionally this will be less useful as the resulting code will be code that calls C code we can't see. Other times it will allow you to understand the function better.

Dataset

The following questions will use the `imdb` database found in Scholar. The credentials to the database are:

Username: imdb_user

Password: movie\$Rkool

This database has 6 tables, namely:

akas, crew, episodes, people, ratings, and titles.

To connect to the database from a terminal in Scholar, execute the following:

```
mysql -u imdb_user -h scholar-db.rcac.purdue.edu -p
```

You will be asked for the password. Type the provided password and press enter. Note that it will look like nothing is being typed as you type, this is OK, you are indeed typing the password.

To connect to the database from Rstudio, open a browser and navigate to <https://rstudio.scholar.rcac.purdue.edu/>, and login using your Purdue Career Account credentials.

To establish a connection with the MySQL database within Rstudio, run the following:

```
install.packages("RMariaDB")
library(RMariaDB)

host <- "scholar-db.rcac.purdue.edu"
user <- "imdb_user"
password <- "movie$Rkool"
database <- "imdb"

db <- dbConnect(RMariaDB::MariaDB(), host=host, db=database, user=user, password=password)
```

After running the code above, you should be successfully connected to the database. From here, you can either use the package `RMariaDB` to query our database:

```
result <- dbGetQuery(db, "SELECT * FROM titles LIMIT 5;")
```

Or you can execute SQL directly in an Rmarkdown file. For example, copy and paste the following code chunks in an RMarkdown file:

This code chunk initiates a connection to the database.

```
````{r}
install.packages("RMariaDB")
library(RMariaDB)

host <- "scholar-db.rcac.purdue.edu"
```

```

user <- "imdb_user"
password <- "movie$Rkool"
database <- "imdb"

db <- dbConnect(RMariaDB::MariaDB(), host=host, db=database, user=user, password=password)
'''

```

This code chunk demonstrates how to run SQL queries from within R.

```

'''{r}
result <- dbGetQuery(db, "SELECT * FROM titles LIMIT 5;")
'''

```

This code chunk demonstrates how to use the SQL connection to run SQL queries directly within a code chunk.

```

'''{sql, connection=db}
SELECT * FROM titles LIMIT 5;
'''

```

**1. Explore the 6 tables. State an interesting fact (of your choice) that you find about at least one of the tables.**

**Relevant topics:** *sql, sql in R*

**Item(s) to submit:**

- A sentence describing at least 1 interesting fact about at least one of the tables.

**2. Find the title\_id, rating, and number of votes for all movies that received at least 2 million votes.**

**Hint:** *Use the ratings table.*

**Relevant topics:** *sql, sql in R*

**Item(s) to submit:**

- SQL query used to solve this problem.
- Output from running the SQL query.



**3. Now use the information you found, about the movies that received at least 2 million votes, to identify the titles of these movies, using the titles table.**

**Hint:** *You will probably recognize the names of these movies.*

**Relevant topics:** *sql, sql in R*

**Item(s) to submit:**

- SQL query used to solve this problem.
- Output from running the SQL query.

**4. Find the names, birth years, and death years, for all actors and actresses who lived more than 115 years.**

**Hint:** \*You can use this clause in your SQL query:\*

```
WHERE died - born > 115
```

**Relevant topics:** *sql, sql in R*

**Item(s) to submit:**

- SQL query used to solve this problem.
- Output from running the SQL query.

**5. In the titles table, the genres column specifies the genre of each movie. Use the COUNT function to find how many movies of each genre occur in the database.**

**Hint:** *You can use the same strategy from the SUM of transactions examples in the election database. Just use COUNT instead of SUM.*

**Relevant topics:** *sql, sql in R*

**Item(s) to submit:**

- SQL query used to solve this problem.

**6. In the titles table, the premiered column specifies the year that a movie was premiered. Use the COUNT function to find how many movies premiered in each year in the database.**

**Relevant topics:** *sql, sql in R*

**Item(s) to submit:**

- SQL query used to solve this problem.

**7. One movie has a strange premiere year. Which movie is this?**

**Relevant topics:** *sql, sql in R*

**Item(s) to submit:**

- SQL query used to solve this problem.
- Output from running the SQL query.

**8. Make a dotchart that shows how many movies premiered in each year since the year 2000.**

**Relevant topics:** *sql, sql in R*

**Item(s) to submit:**

- SQL query used to gather the data used in the dotchart.
- A dotchart that shows how many movies premiered in each year since the year 2000, in png or jpg/jpeg format.

**9. The title ‘The Awakening’ has been used very often! How many times has this been used as a title?**

**Relevant topics:** *sql, sql in R*

**Item(s) to submit:**

- SQL query used to solve this problem.
- Output from running the SQL query.

**10. Investigate all of the occurrences of these titles called ‘The Awakening’. Find an interesting fact about the entries with these titles.**

**Relevant topics:** *sql, sql in R*

**Item(s) to submit:**

- SQL query used to solve this problem.
- Output from running the SQL query.
- 1-2 sentences describing the interesting fact you found about the entries with these titles.



## Chapter 11

# Contributors

We are extremely thankful for all of our contributors! Get your name added to the list by making a contribution.