

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	30
	Emacs	33
	Nano	33
	Vim	33
	Writing scripts	33
4	SQL	35
5	R	49
	Getting started	49
	Variables	49
	Logical operators	49
	Lists & Vectors	49
	Basic R functions	49

Data.frames	52
Reading & Writing data	53
Control flow	53
Apply functions	53
Writing functions	53
Plotting	53
RMarkdown	53
Tidyverse	57
data.table	57
SQL in R	57
Scraping	57
shiny	57
6 Python	59
Getting started	59
Lists & Tuples	61
Dicts	61
Control flow	61
Writing functions	61
Reading & Writing data	61
numpy	61
scipy	61
pandas	61
Jupyter notebooks	61
Writing scripts	61
Scraping	61
Plotting	61
Classes	61
tensorflow	61
pytorch	61

<i>CONTENTS</i>	5
7 Tools	63
Docker	63
Tableau	63
GitHub	63
VPNs	76
8 FAQs	77
How do I connect to Scholar from off-campus?	77
Is there an advantage to using the ThinLinc client rather than the ThinLinc web client?	77
GitHub Classroom is not working – can’t authorize the account. . . .	77
In Scholar, on RStudio, my font size looks weird or my cursor is offset.	78
I’m unable to type into the terminal in RStudio.	78
I’m unable to connect to RStudio Server.	78
RStudio initialization error.	78
RStudio crashes when loading a package.	79
RStudio license expired.	79
RStudio is taking a long time to open.	79
How can you run a line of R code in RStudio without clicking the “Run” button?	79
My R session freezes.	80
White screen issue when loading RStudio.	80
Scholar is slow.	80
There are no menus in Scholar.	81
Firefox in Scholar won’t open because multiple instances running. . .	82
How to transfer files between your computer and Scholar.	83
ThinLinc app says you can’t create any more sessions.	85
How to install ThinLinc on my computer.	86
Forgot my password or password not working with ThinLinc.	86
Jupyter Notebook download error with IE.	86
Jupyter Notebook kernel dying.	86
Python kernel not working, Jupyter Notebook won’t save.	87

Installing <code>my_package</code> for Python	87
Displaying multiple images after a single Jupyter Notebook Python code cell.	88
RMarkdown “Error: option error has NULL value” when knitting“. . .	89
How do you create an RMarkdown file?	89
Problems building an RMarkdown document on Scholar.	89
How can I use SQL in RMarkdown?	90
Copy/paste from terminal inside RStudio to RMarkdown.	91
How do I render an image in a <code>shiny</code> app?	91
The package <code>my_package</code> is not found.	91
Problems installing <code>ggmap</code>	91
Error: <code>object_name</code> is not found	92
Zoom in on <code>ggmap</code>	92
Find the latitude and longitude of a location.	92
Problems saving work as a PDF in R on Scholar.	93
What is a good resource to better understand HTML?	93
Is there a style guide for R code?	93
Is there a guide for best practices using R?	93
Tips for using Jupyter notebooks.	94
What is my username on Scholar?	94
How to submit homework to GitHub without using Firefox?	94
How and why would I need to “escape a character”?	94
How can I fix the error “Illegal byte sequence” when using a UNIX utility like <code>cut</code> ?	95
9 Projects	97
Templates	97
STAT 19000	97
STAT 29000	97
STAT 39000	135
10 Think Summer 2020	167
Project	167

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

11 Contributors	175
------------------------	------------

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:



Figure 1.1:

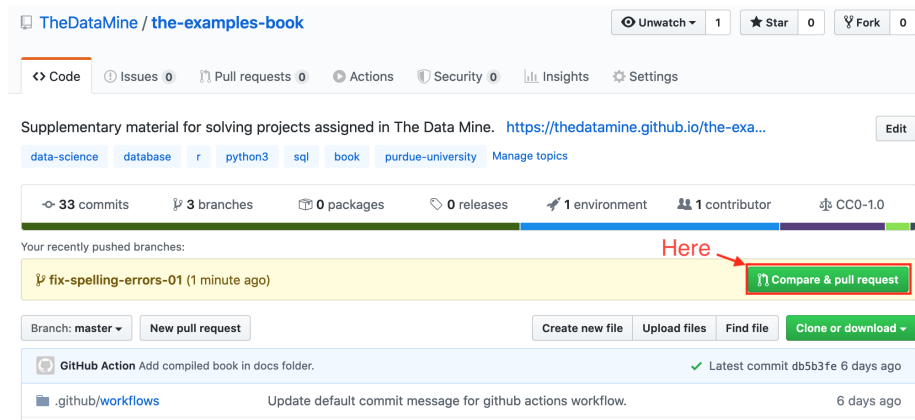


Figure 1.2:

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`.
- Click “Clone”.
 - Upon completion, you will be presented with a screen similar to this:
 - 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`.
 - 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”.
 - If you do not already have `renv` installed, install it by running the following commands in the console:

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

- Restore the environment by running the following commands in the console:

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

- 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.

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

find

fd

top

less & more

sort

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

How to install R (windows/mac/linux) How to install RStudio How to connect to RStudio Server on Scholar How to get help (?, help(), get function itself)

Variables

Logical operators

Lists & Vectors

Basic R functions

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
```

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 a
# Search for a literal "(", followed by any amount of any text other than more parenth
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 a
# Search for a literal "(", followed by any amount of any text (.*), followed by a lit
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 create a `data.frame` with comma-separated data that I've copied?

[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

Apply functions

`apply`

`sapply`

`lapply`

`tapply`

Writing functions

Plotting

`ggplot`

`ggmap`

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**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>

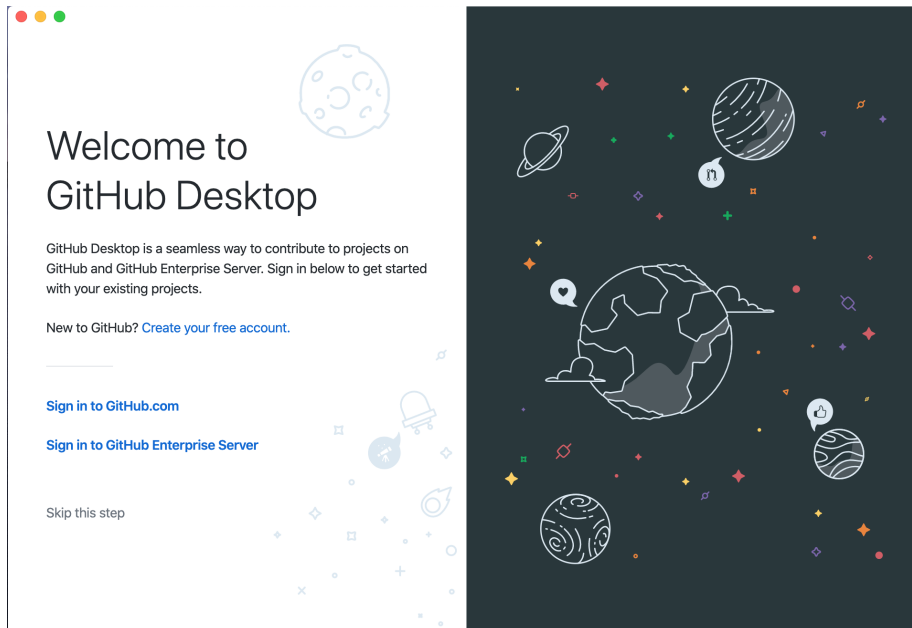
Fill out the information, and click “Create pull request”.

GitHub Desktop

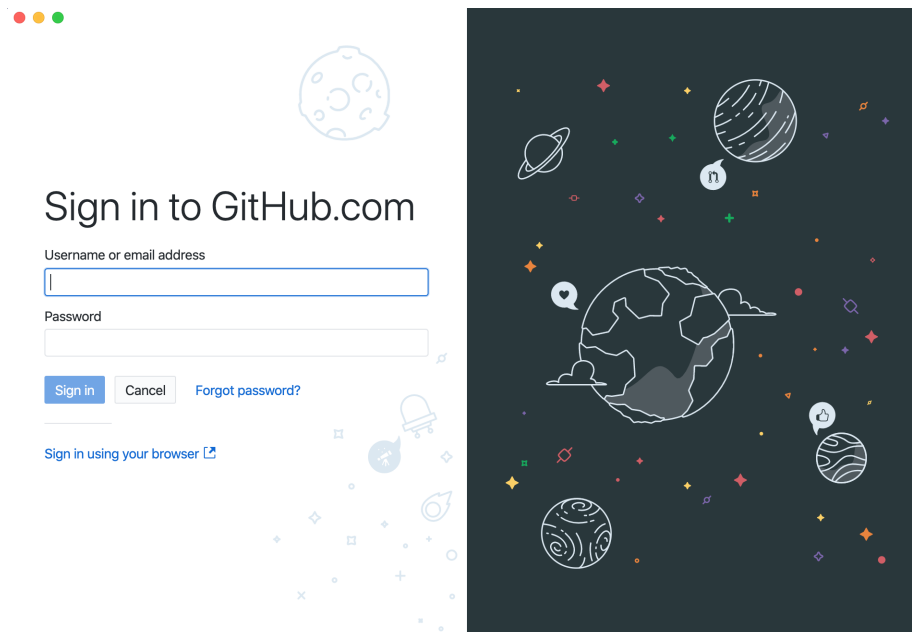
Install

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:

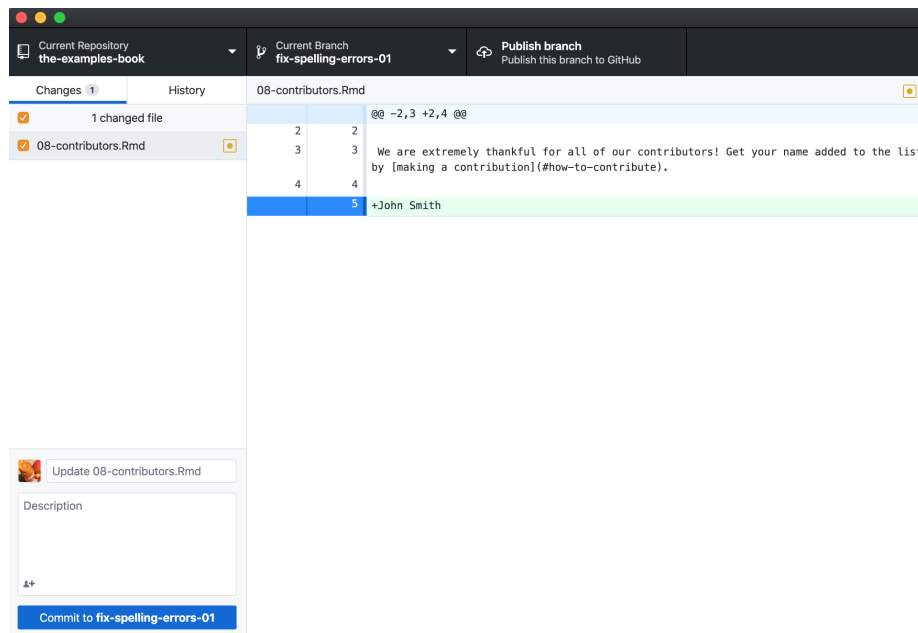


5. Continue the sign in process. You will eventually be presented with a screen

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 respository (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:

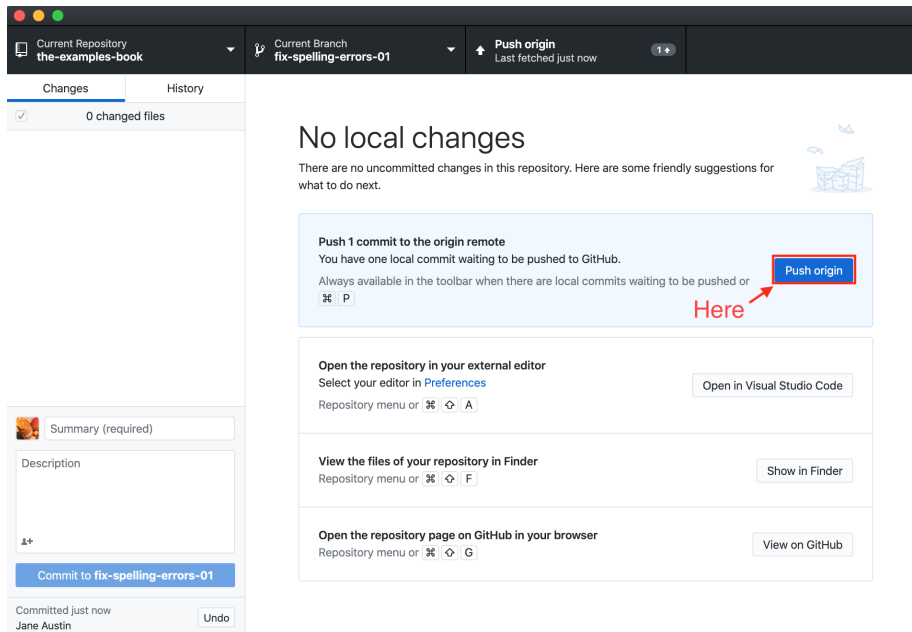


Figure 7.1:

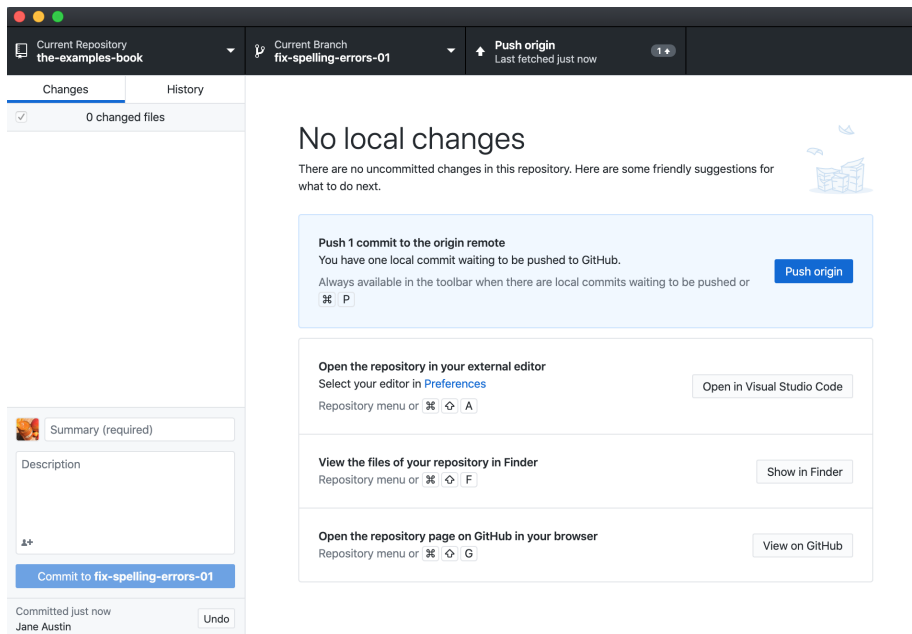


Figure 7.2:

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`):

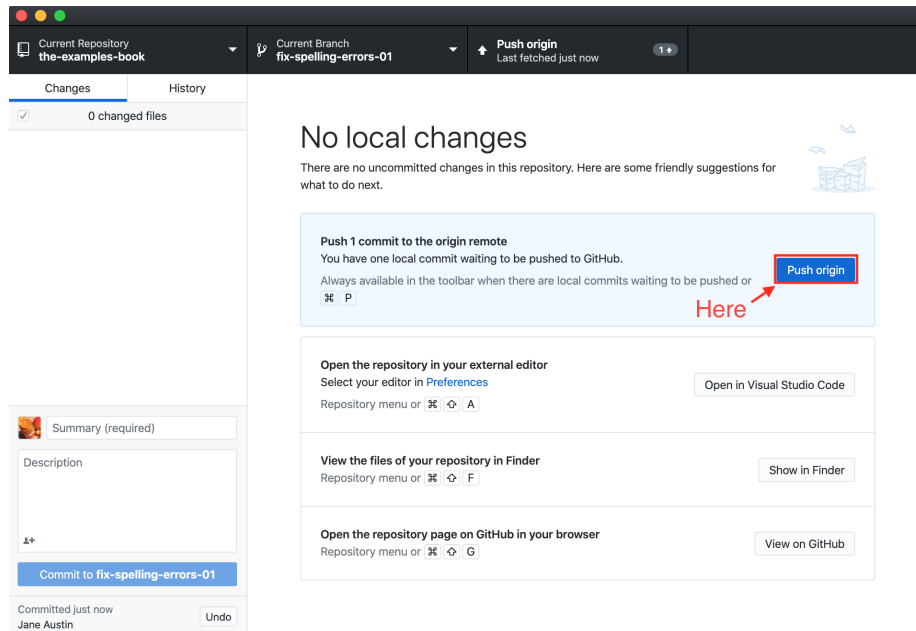
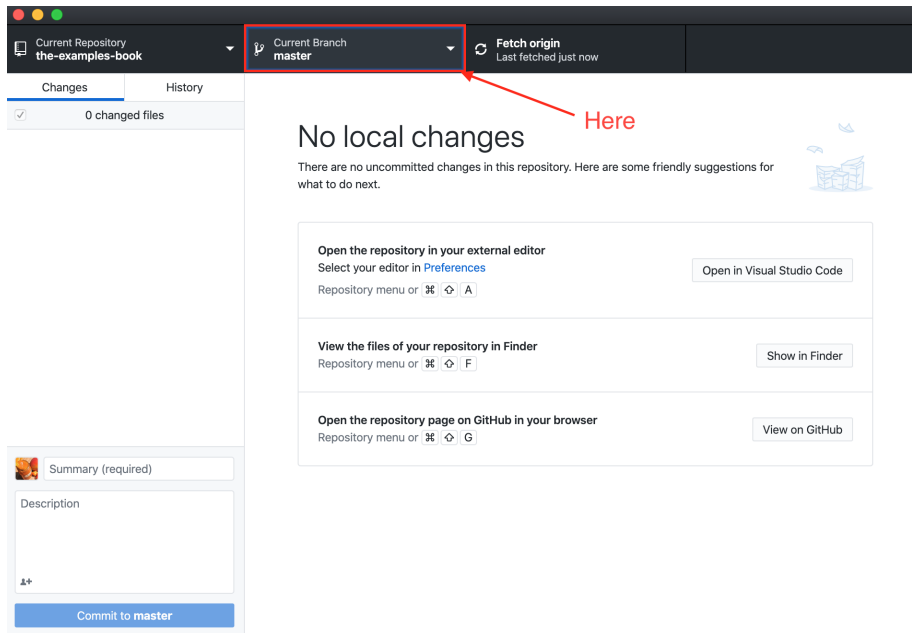


Figure 7.3:

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

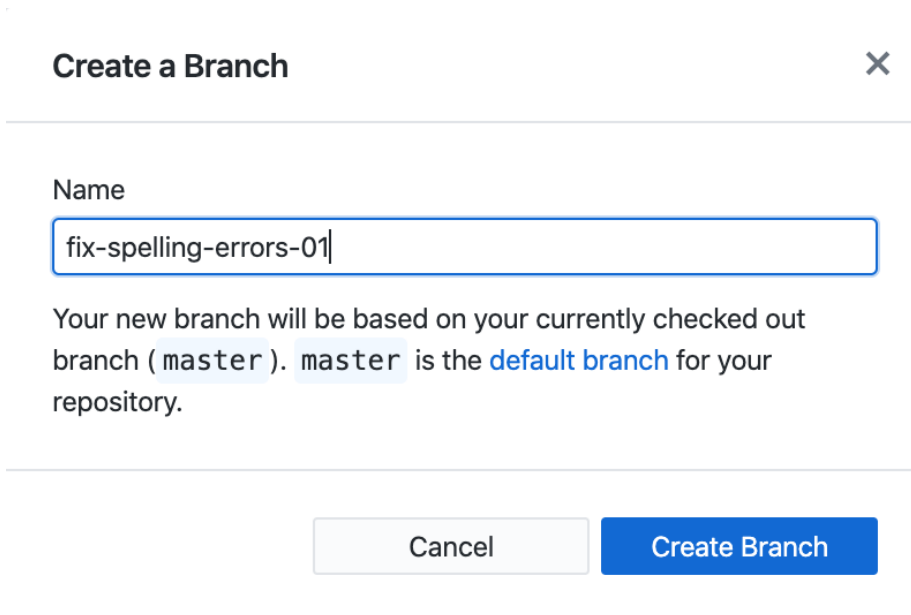
5 days ago

 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:

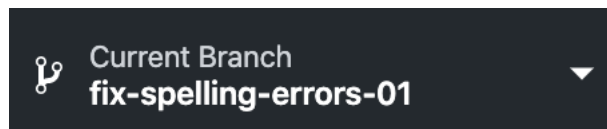
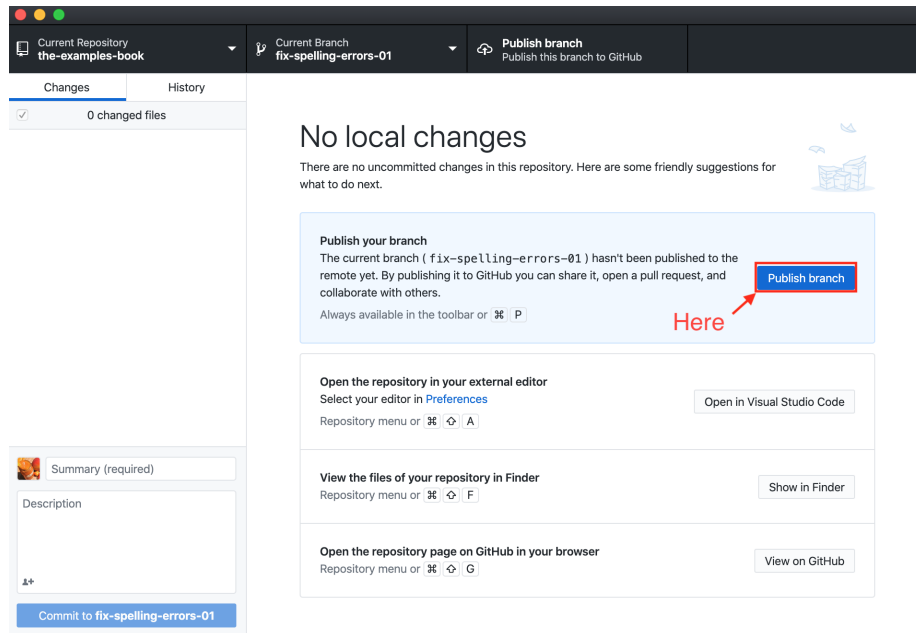


Figure 7.4:

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:

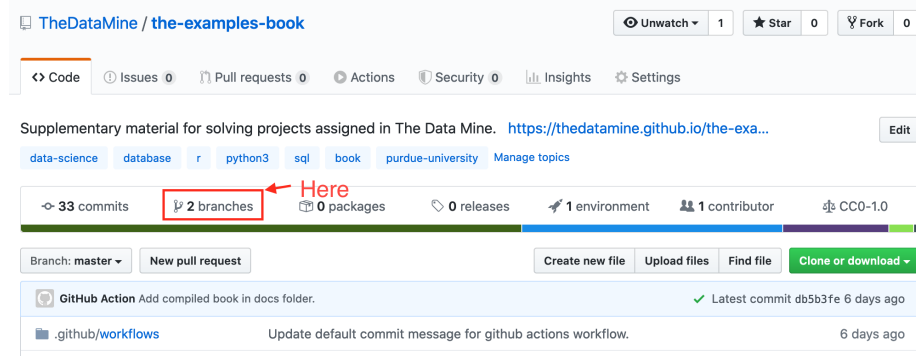


Figure 7.5:

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:

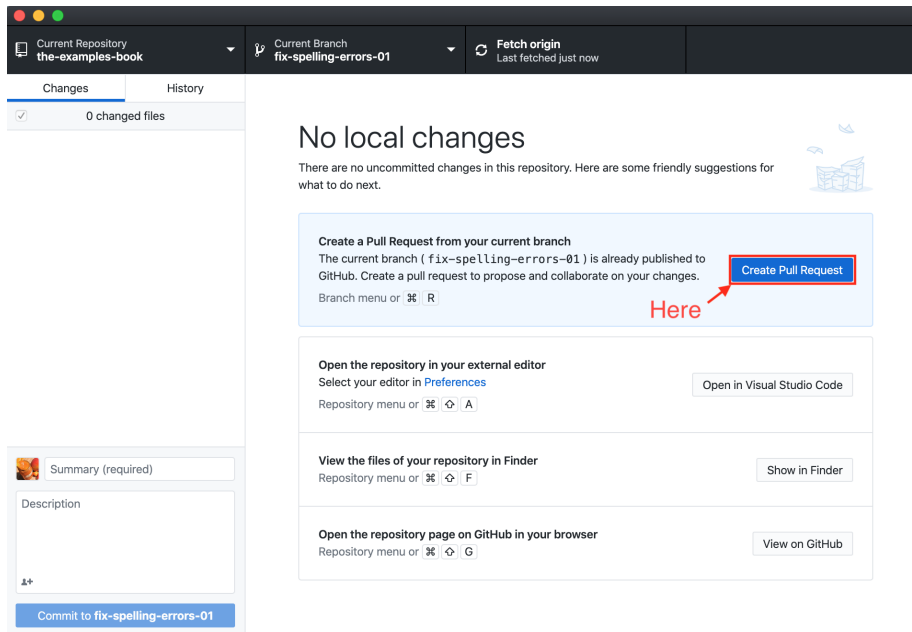


Figure 7.6:

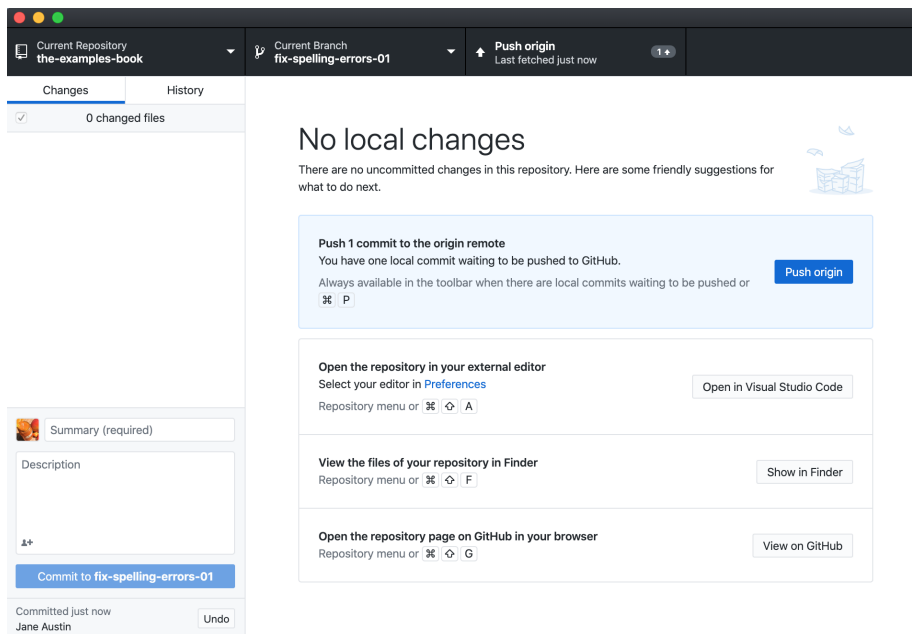


Figure 7.7:

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:

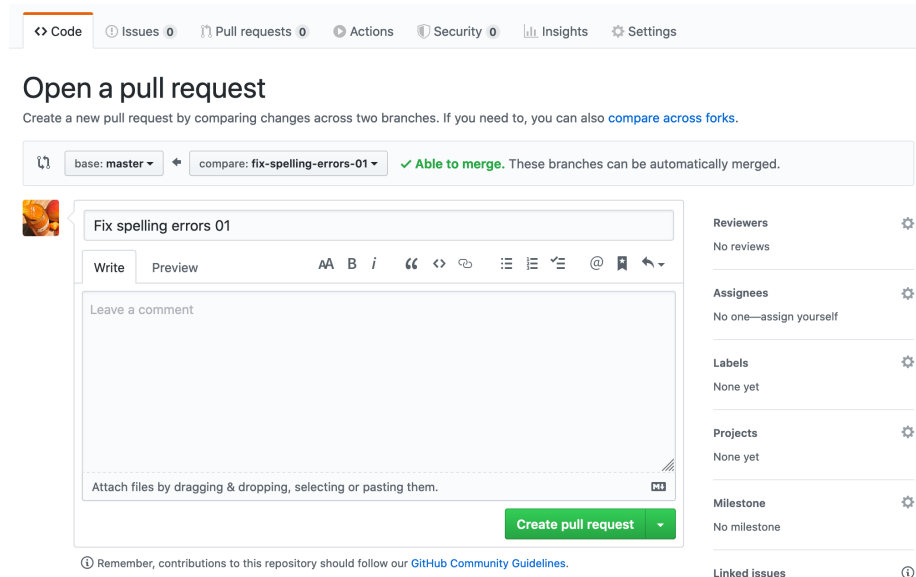


Figure 7.8:

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”:

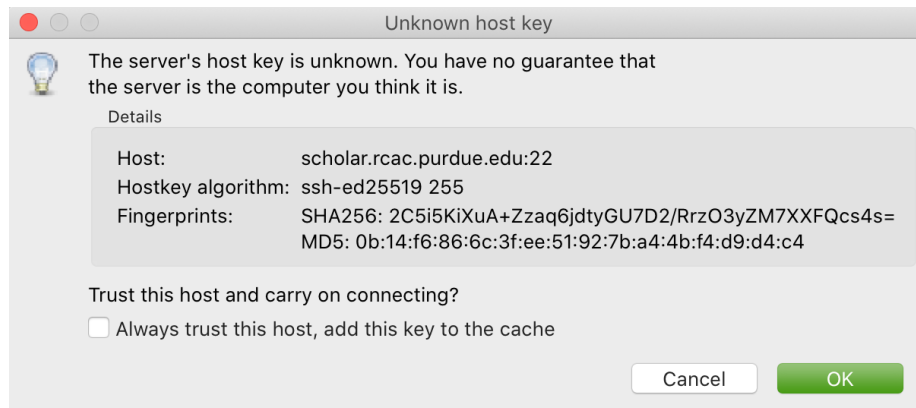
Host: scholar.rcac.purdue.edu

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

Password: <your_scholar_password>

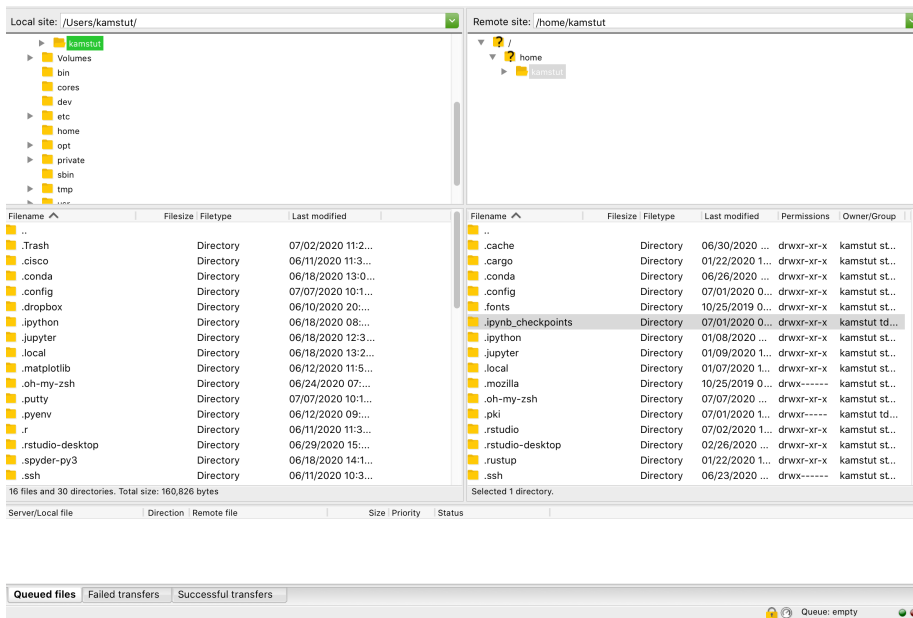
Port: 22

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.*⁹¹

- 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?*⁹⁹

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

STAT 29000

Project 1

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 data.frame `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 a 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

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 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

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

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 "[]". 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

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

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

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

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 12

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 and **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.

STAT 39000

Project 2

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 `?<code>`. To use, simply type `?<code>` 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. 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 4

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

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 `?<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 `Summarys` 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

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/YYYY.csv`
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 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.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 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.

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

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

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 11

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 your 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).
-

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.