The Examples Book

Contents

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 succesfully 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:

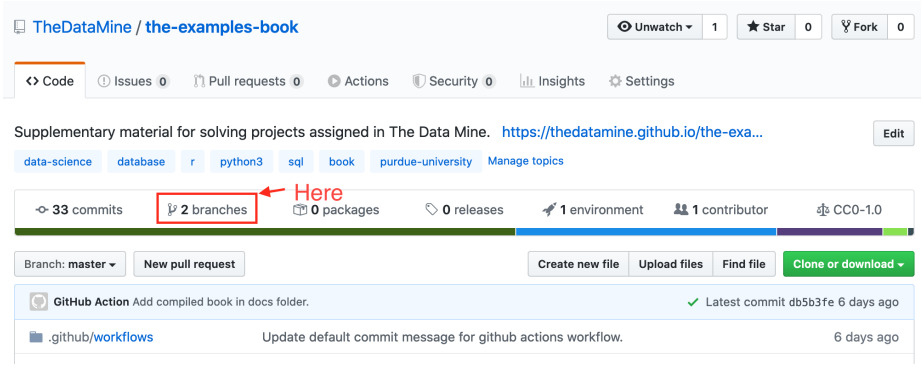


Figure 1.1:

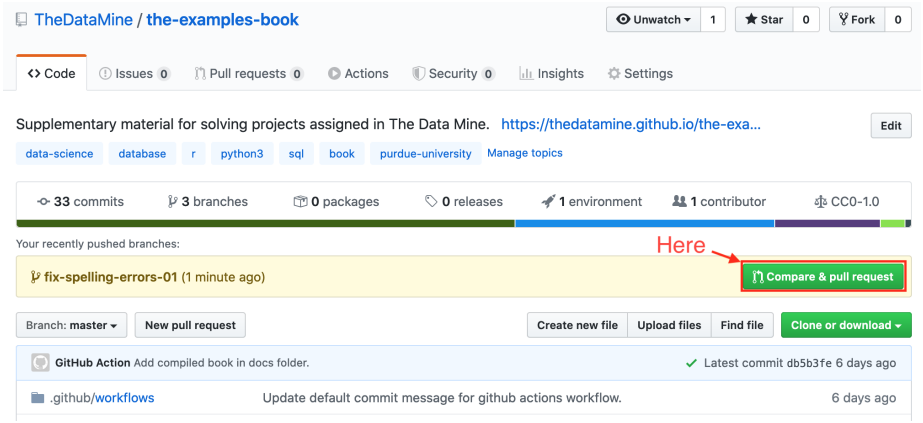


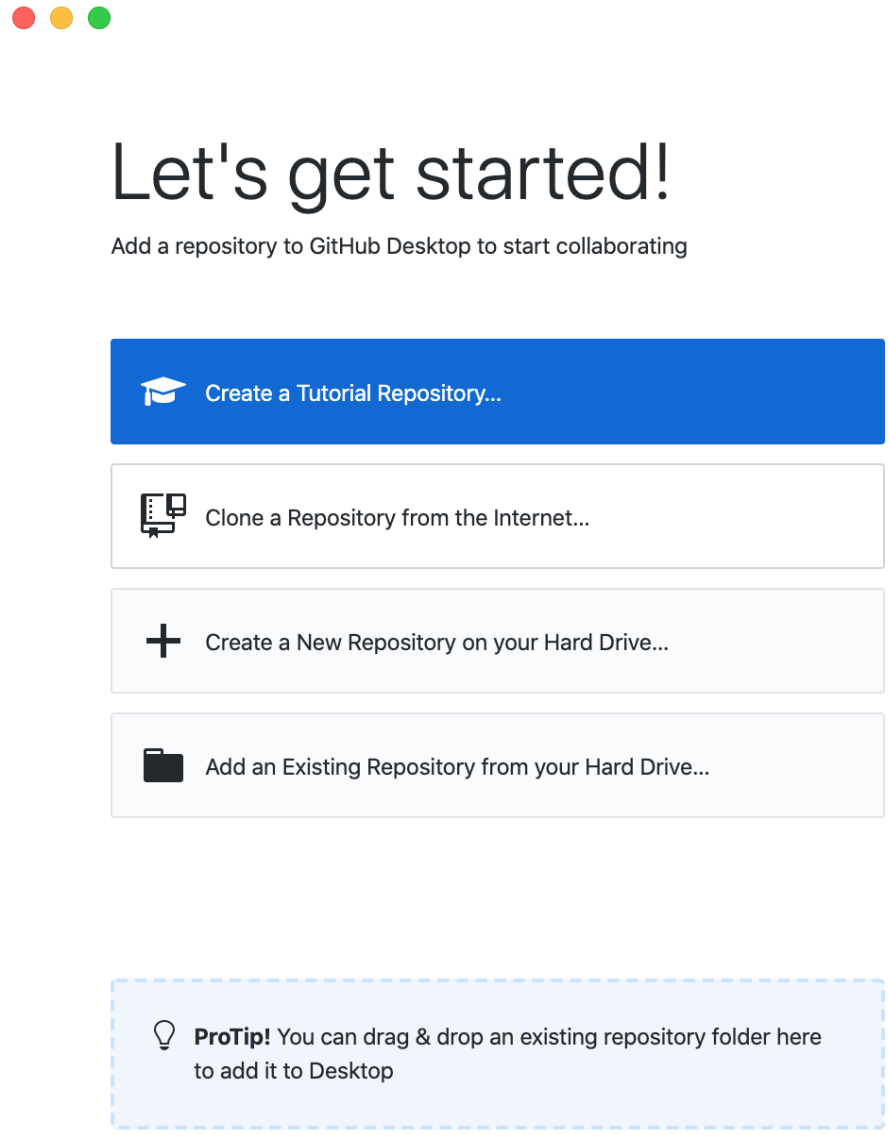
Figure 1.2:

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...":



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

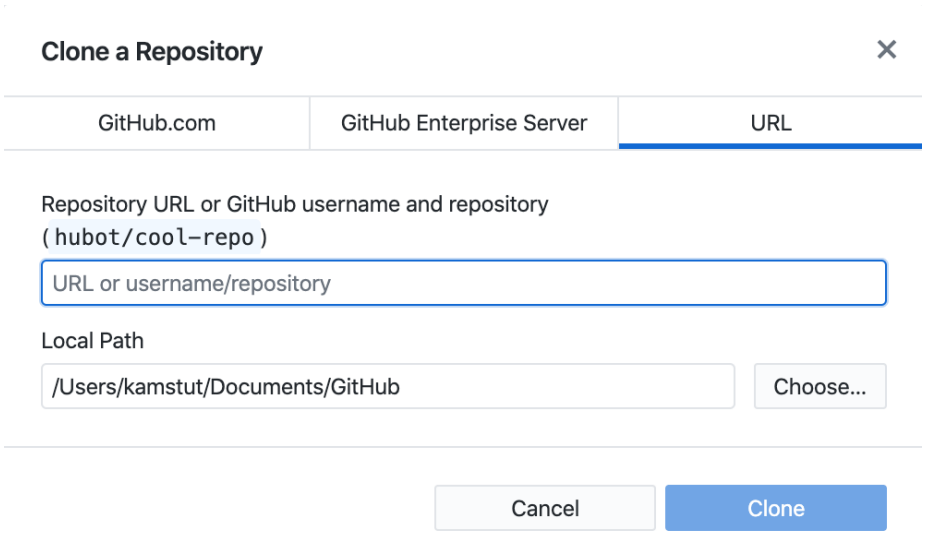


Figure 1.3:

called `the-examples-book` in `/Users/kamstut/Documents/GitHub`.
6. Click "Clone".
7. Upon completion, you will be presented with a screen similar to this:

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

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

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

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

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

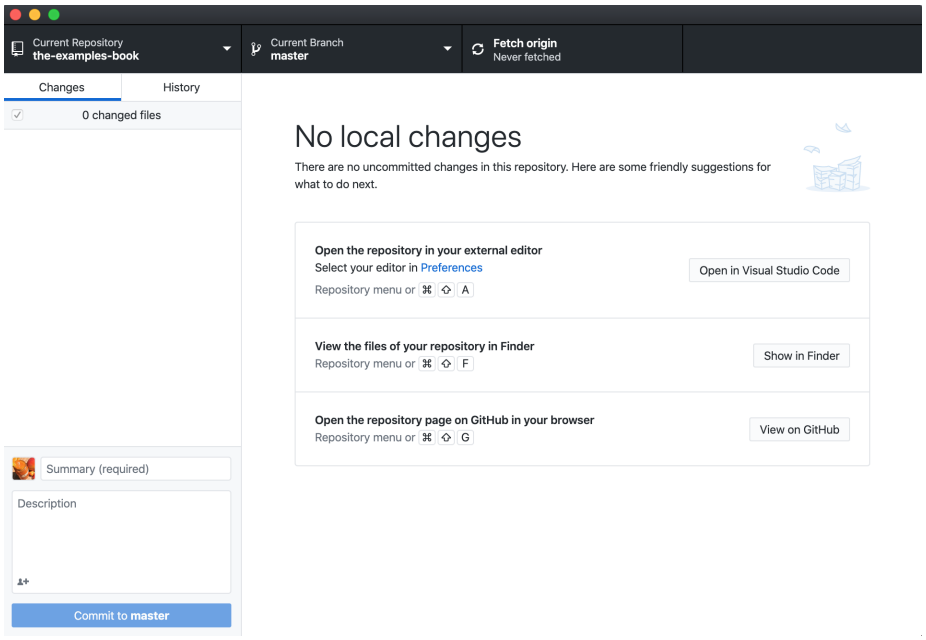


Figure 1.4:

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

13. In addition, make sure to install both `pandoc` and `pandoc-citeproc` by following the instructions here.

14. Modify the `.Rmd` files to your liking.

15. Click the "Knit" button to compile the book. The resulting "book" is within the "docs" folder.

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

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

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

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

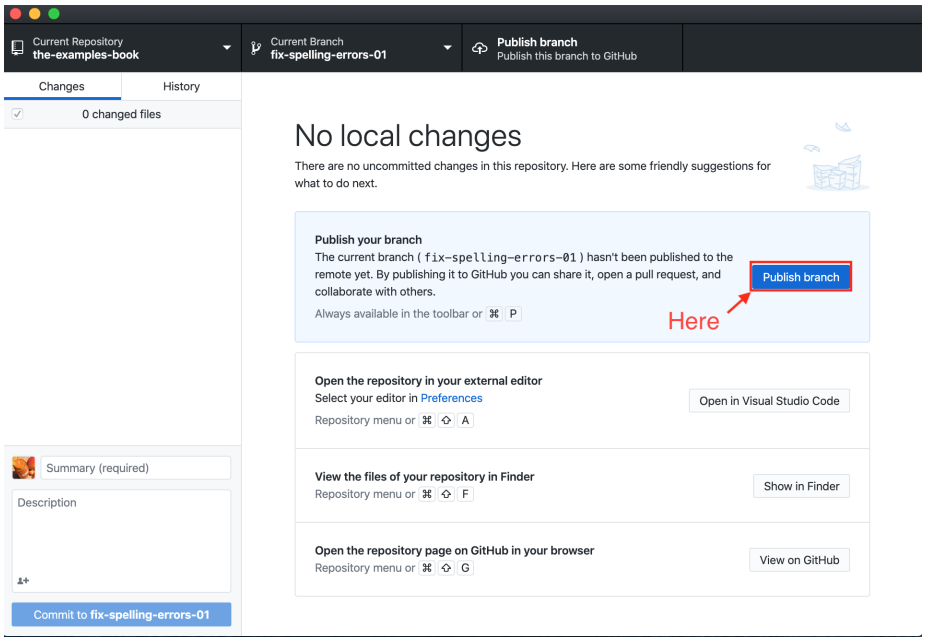


Figure 1.5:

19. Upon publishing your branch, within GitHub Desktop, you'll be presented with the option to create a pull request:

20. At this point in time, the repository owners will receive a notification and will check and potentially merge the changes into the `master` branch.

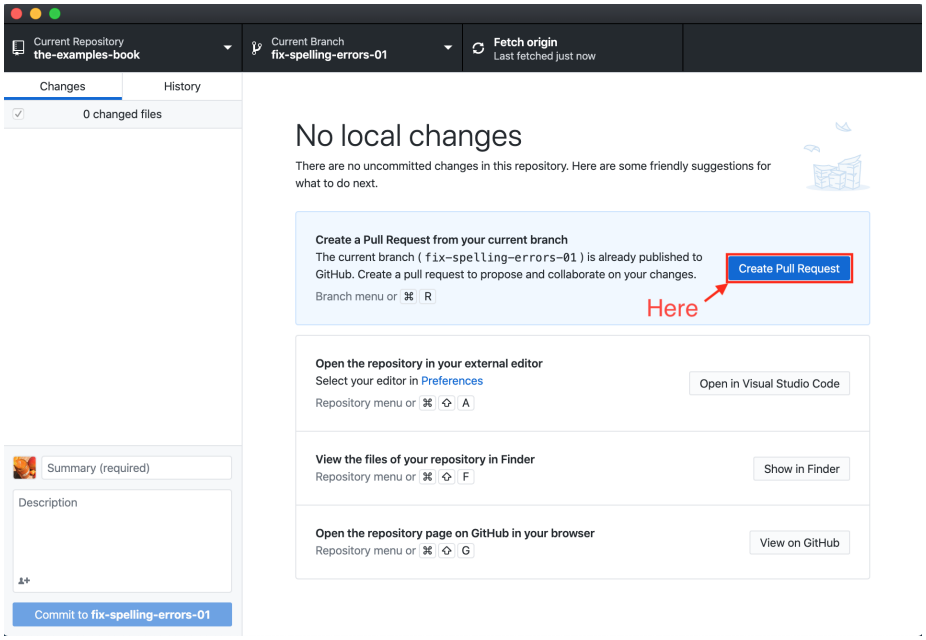

Current Repository
the-examples-book
Current Branch
fix-spelling-errors-01
Fetch origin
Last fetched just now
Changes
History
0 changed files
No local changes
There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.
Create a Pull Request from your current branch
The current branch (fix-spelling-errors-01) is already published to GitHub. Create a pull request to propose and collaborate on your changes.
Branch menu or ⌘ R
Create Pull Request
Here
Open the repository in your external editor
Select your editor in Preferences
Repository menu or ⌘ ⇧ A
Open in Visual Studio Code
View the files of your repository in Finder
Repository menu or ⌘ ⇧ F
Show in Finder
Open the repository page on GitHub in your browser
Repository menu or ⌘ ⇧ G
View on GitHub
Summary (required)
Description
Commit to fix-spelling-errors-01

Figure 1.6:

Chapter 2

Scholar

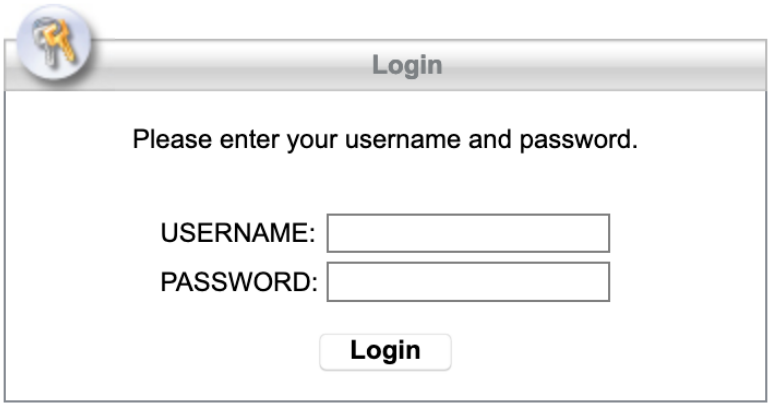
Connecting to Scholar

ThinLinc web client

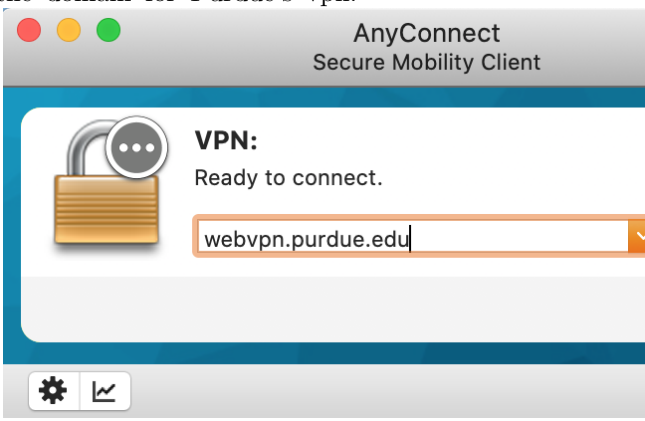
1. Open a browser and navigating to https://desktop.scholar.rcac.purdue.edu/.
2. Login with your Purdue Career Account credentials (**without** BoilerKey).
3. Congratulations, you should now be connected to Scholar using the ThinLinc web client.

ThinLinc client

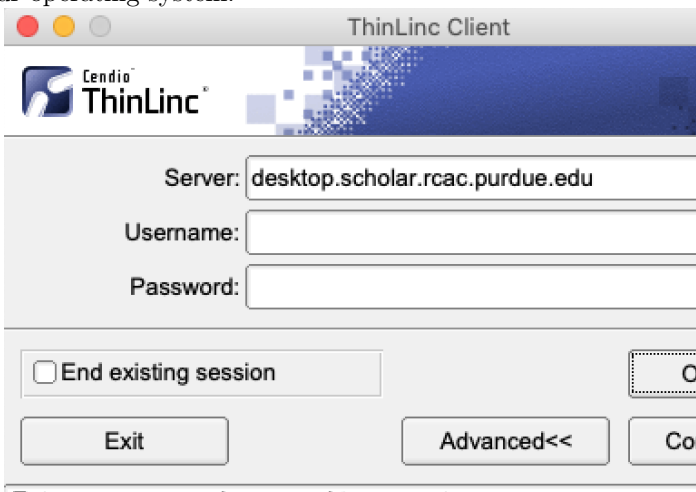
1. Navigate to https://webvpn.purdue.edu/. You should see a login screen:



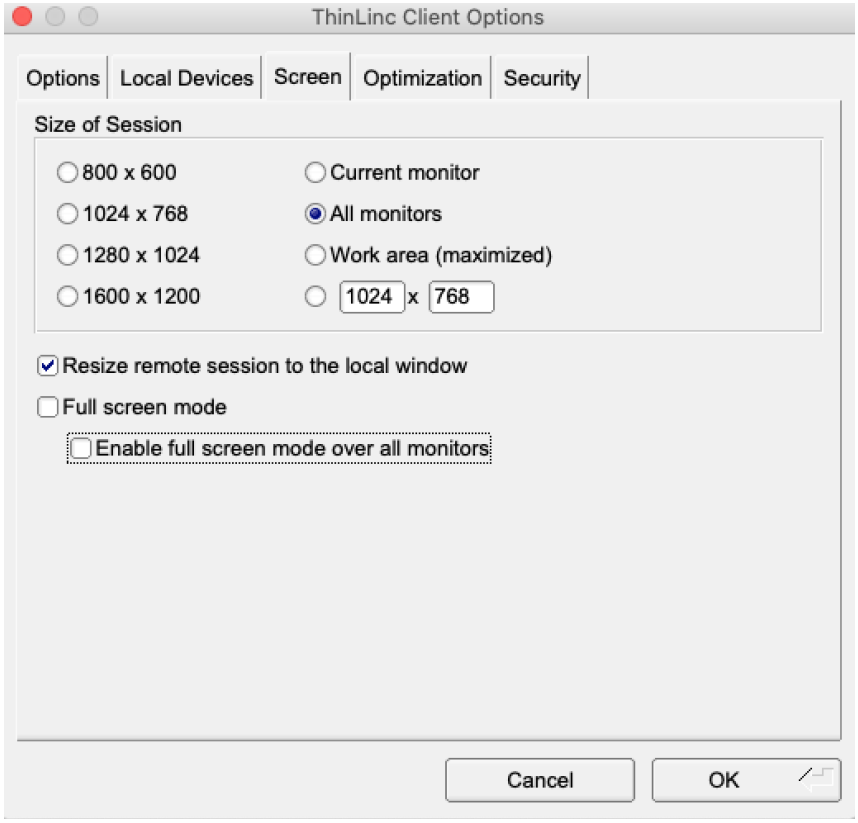
2. Enter your Purdue Career Account credentials (**with** BoilerKey).
3. Download and install the Cisco AnyConnect Secure Mobility Client.
4. Open the AnyConnect client and enter the domain for Purdue's vpn: `webvpn.purdue.edu`, and click "Connect":



5. When prompted, enter your Purdue Career Account credentials (**with** BoilerKey).
6. You should be successfully connected to Purdue's VPN! You can read more about VPNs here.
7. Navigate to https://www.cendio.com/thinlinc/download, and download the ThinLinc client application for your operating system.
8. Install and launch the ThinLinc client:



9. Enter your Purdue Career Account information (**without** BoilerKey), as well as the server: `desktop.scholar.rcac.purdue.edu`.
10. Click on "Options..." and fill out the "Screen" tab as shown below:



11. Click "OK" and then "Connect". **Make sure you are connected to Purdue's VPN using AnyConnect before clicking "Connect"!**
12. If you are presented with a choice like below, click "Continue".

13. 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`

`tldr`

~ & . & ..

`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
```

ls

cp

mv

pwd

touch

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

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-contributors.Rmd
## README.md
```

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

RDBMS

SQL in R

SQL in Python

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

```r
# 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

```r
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

```r
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 (stackoverflow). 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 au

# 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 au

# 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

Lists & Tuples

Dicts

Control flow

Writing functions

Reading & Writing data

`numpy`

`scipy`

`pandas`

Jupyter notebooks

Writing scripts

`argparse`

Scraping

Plotting

`matplotlib`

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

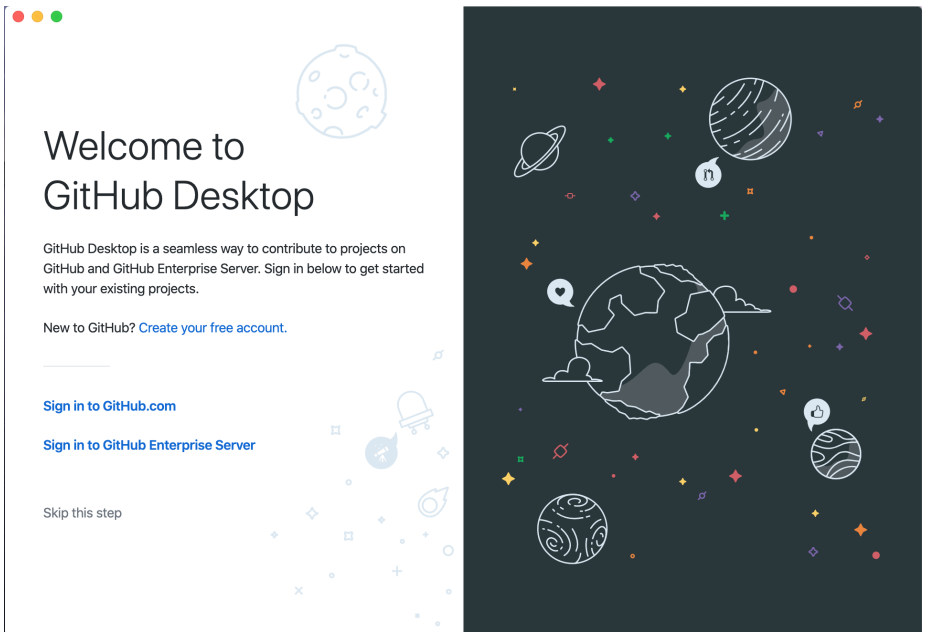
Fill out the information, and click "Create pull request".

GitHub Desktop

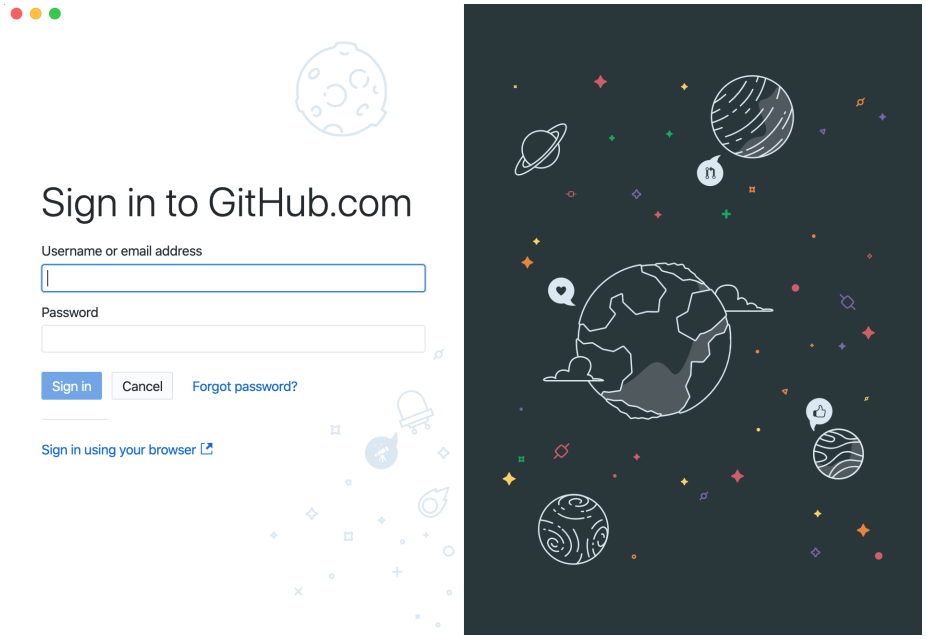
Install

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:

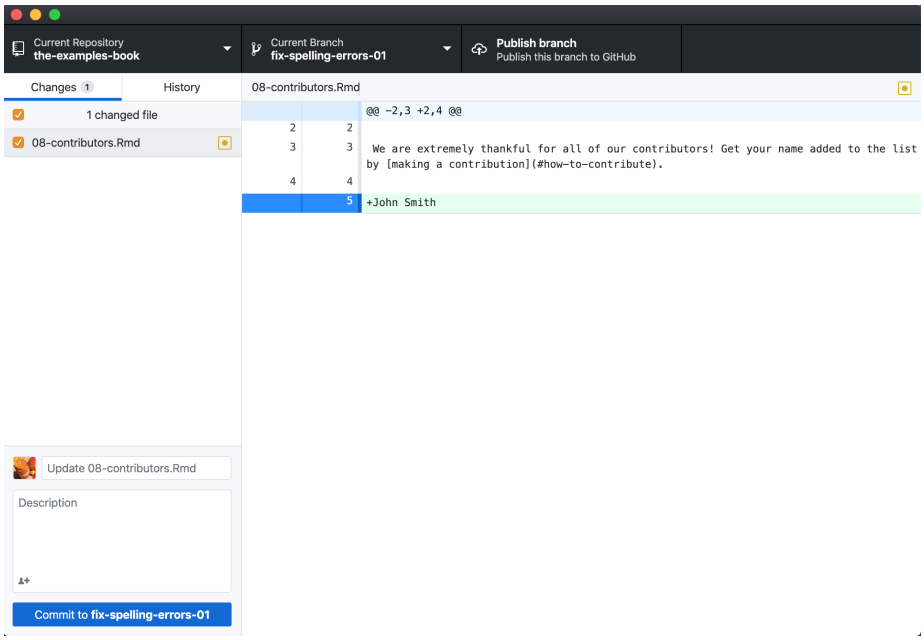


5. Continue the sign in process. You will eventually be presented with a screen

to select a repository. Congratulations! You've succesfully 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:

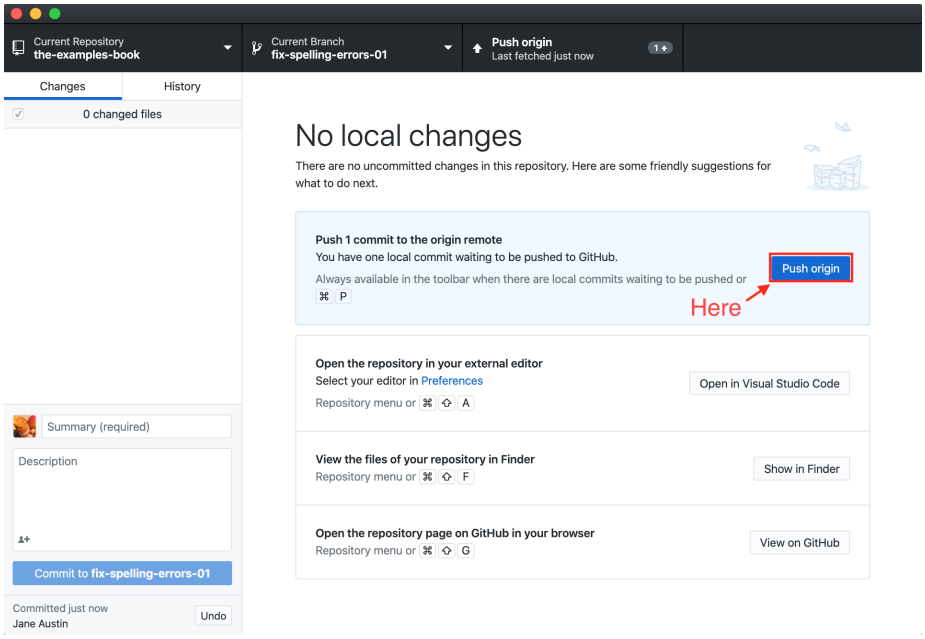


Figure 7.1:

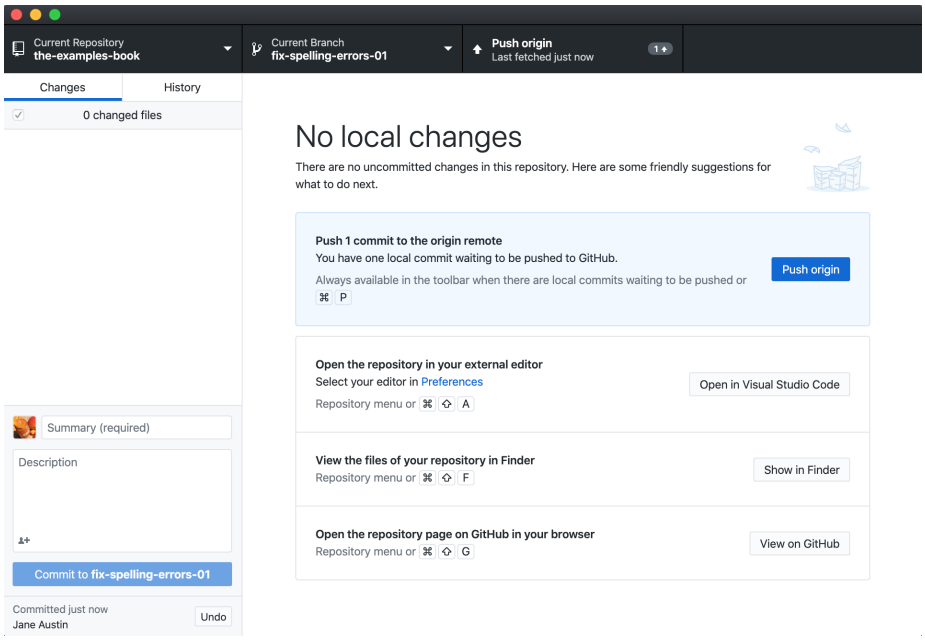


Figure 7.2:

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`):

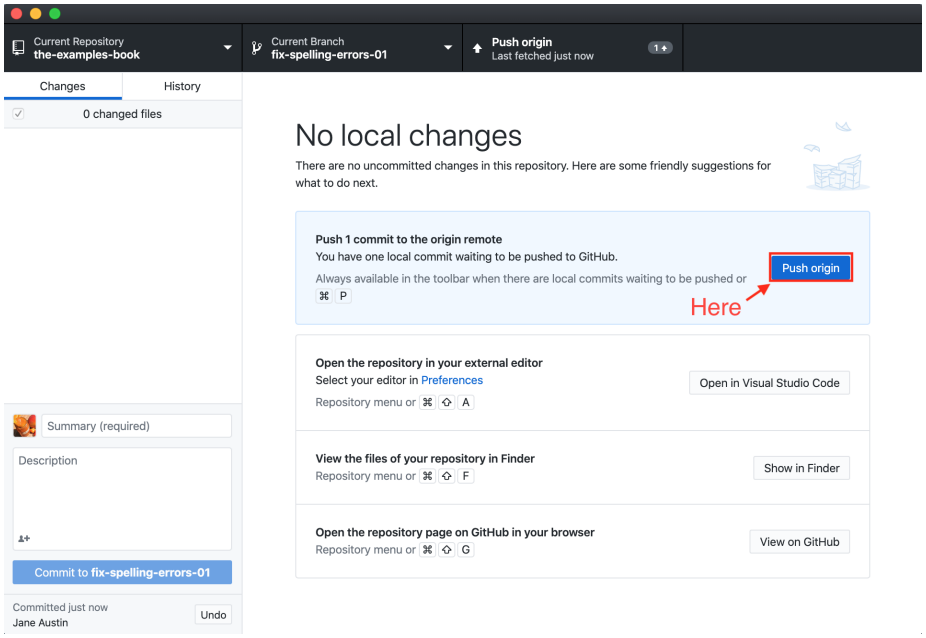
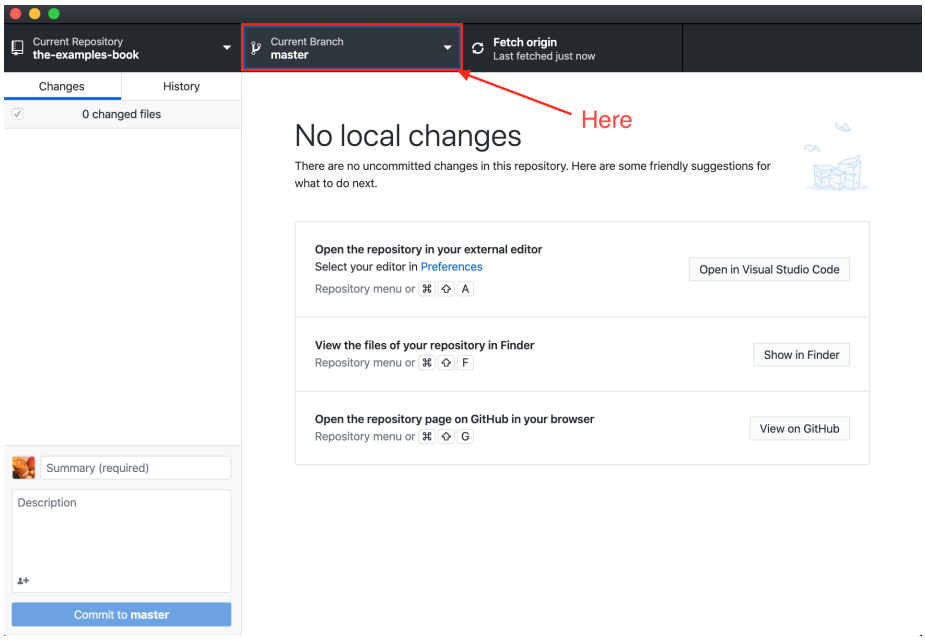


Figure 7.3:

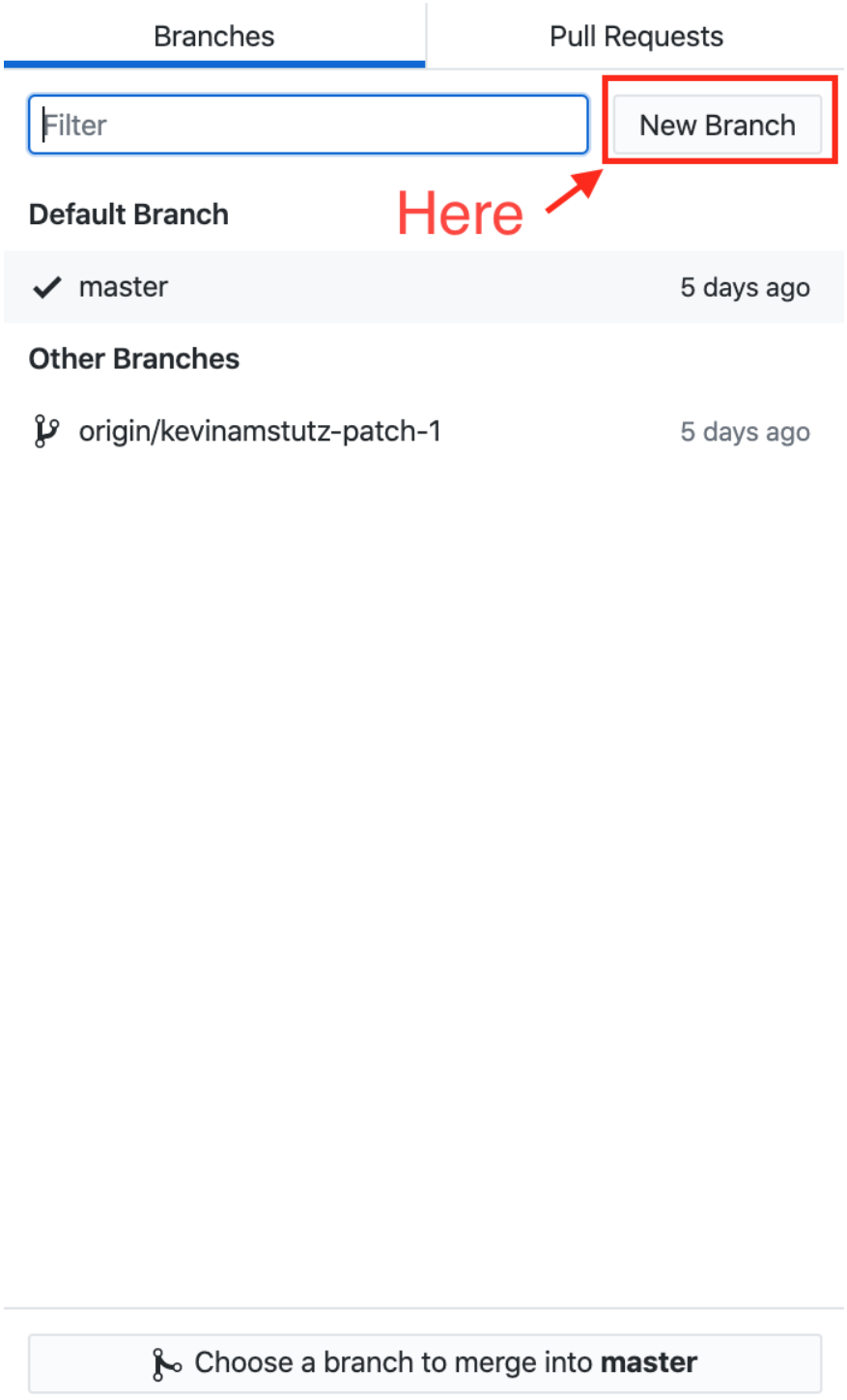
3. You can verify that the changes have been made by navigating to the branch on github.com, and checking the commit history.

Create a new branch

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

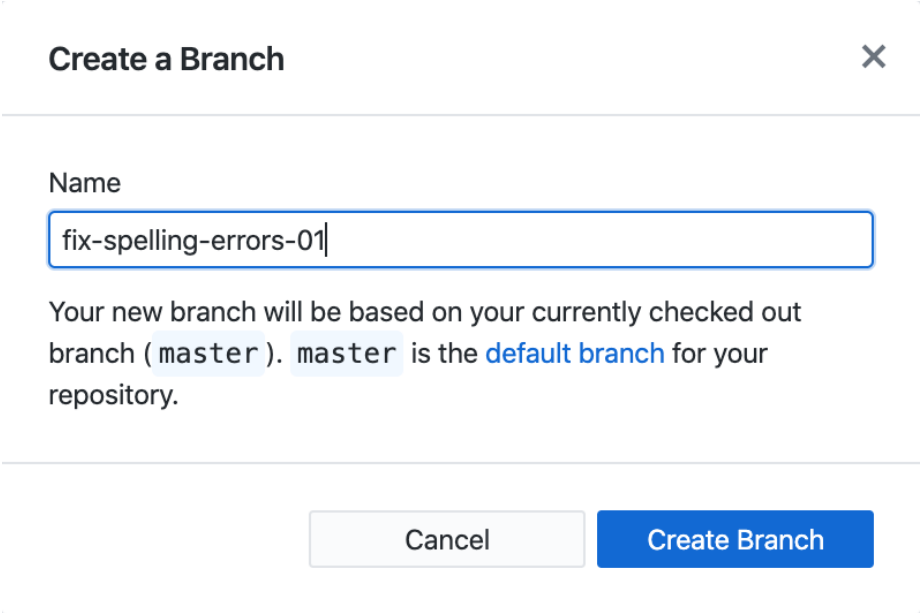


2. Click on the "New Branch" button:


Branches
Pull Requests
Filter
New Branch
Here
Default Branch
master
5 days ago
Other Branches
origin/kevinamstutz-patch-1
5 days ago
Choose a branch to merge into master

3.

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

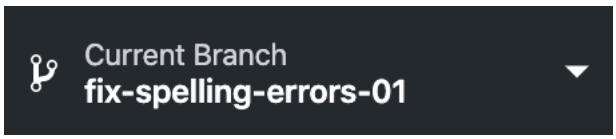
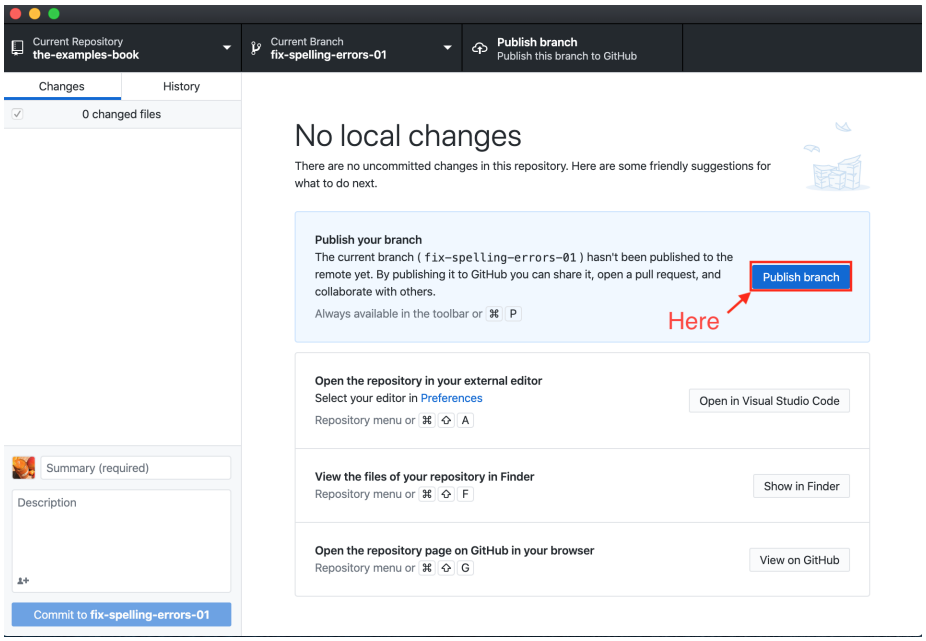


Figure 7.4:

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:

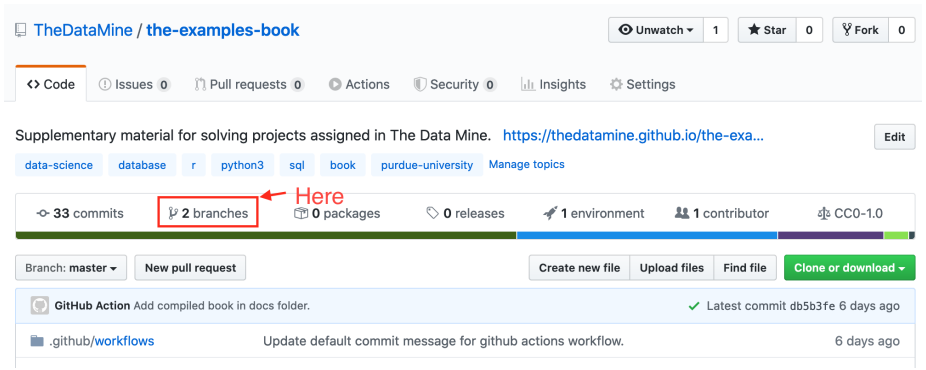


Figure 7.5:

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:

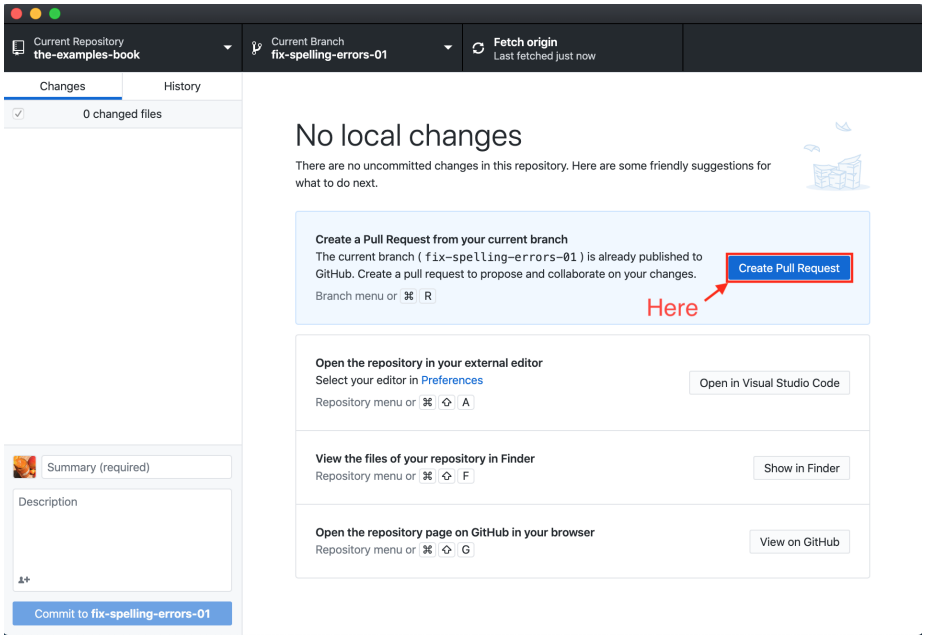


Figure 7.6:

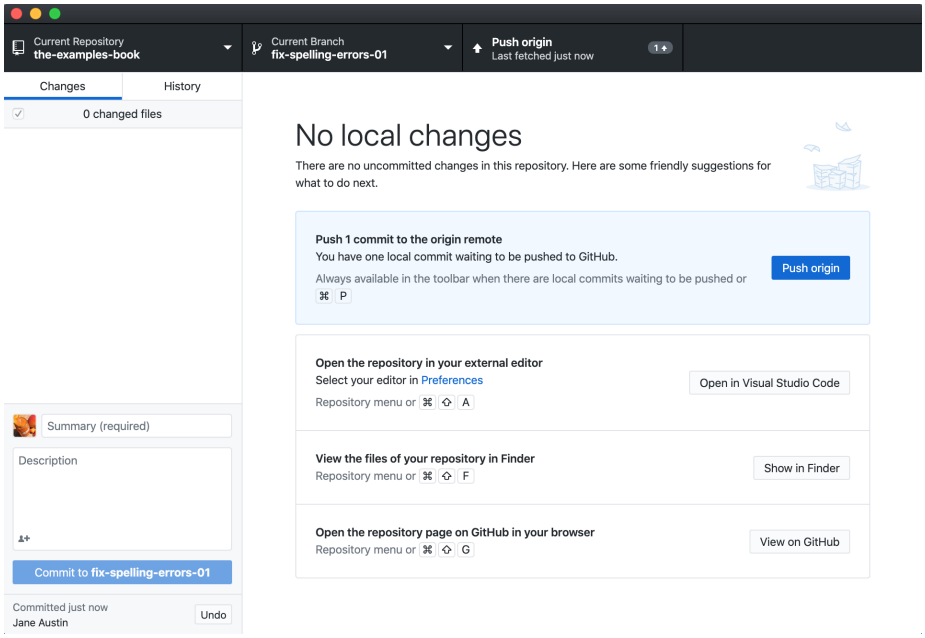


Figure 7.7:

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:

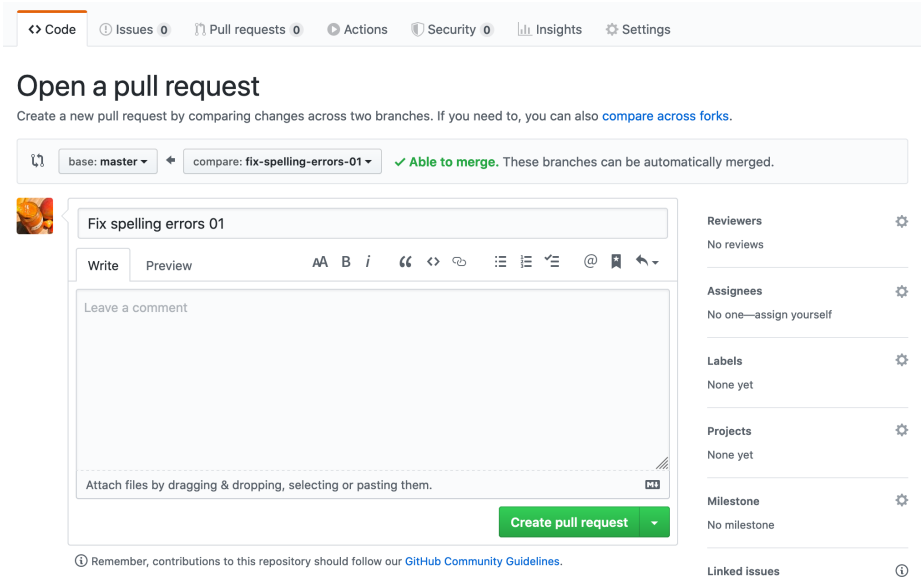


Figure 7.8:

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 setup a VPN connection to Purdue's VPN, and 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.

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, my font size looks weird or my cursor is offset.

In scholar, navigate to `Tools > Global Options > Appearance`, and select the "Monospace" font, and click the "Apply" button.

How do I make the ThinLinc window bigger without going to the dreaded "full screen" mode?

See here.

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.rcac.purdue.edu, and open up RStudio from within the ThinLinc web client.

RStudio initialization error.

1. Navigate to https://desktop.scholar.rcac.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, one at a time. Be sure to replace *yourusername* with your actual Purdue alias. These commands connect you to the various Scholar frontends and kill all of the processes running under your username.

```
ssh scholar-fe00;
killall -9 -u yourusername;
```

```
ssh scholar-fe01;
killall -9 -u yourusername;

ssh scholar-fe02;
killall -9 -u yourusername;

ssh scholar-fe03;
killall -9 -u yourusername;

ssh scholar-fe04;
killall -9 -u yourusername;

ssh scholar-fe05;
killall -9 -u yourusername;

ssh scholar-fe06;
killall -9 -u yourusername;
```

During one of these 7 processes (from 00 to 06) you will get logged out of Scholar. When this happens, repeat steps **(1)** and **(2)** and continue to execute remaining commands. Once you have killed your processes on all 7 machines, 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. Close RStudio.
2. Open the File Manager (it is located in the dock, and looks like a filing cabinet).
3. Select your home directory (Dr. Ward's is called "mdw", for instance).
4. You need to see the "hidden files", which you can do by pressing Control-H or by choosing View / Show Hidden Files in the menu at the top of the File Manager window.
5. Drag your `.RData` file to the Trash.
6. Click on the Trash and then choose File / Empty Trash from the File Manager window, and then click "Empty Trash" to confirm that you want to empty the Trash.

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/xf
```

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

```
killall -9 -u [PUT USERNAME HERE]
```

   Example: Dr. Ward would type:

```
killall -9 -u mdw
```

Running the command above will kill your session. When youd 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
```

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.
- 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.

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 the 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 10*10*10 = 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`?

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

R & Bash

Template

Students in STAT 19000, 29000, and 39000 are to use this as a template for non-python project solutions. All code should live inside code chunks with the proper language code (r, bash, sql, etc.). Every question should be clearly marked with a third-level header (using 3 "#"'s). 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 prior to submission.

Any format or template related questions should be asked in Piazza.

Python

Template

Students in STAT 19000, 29000, and 39000 are to use this as a template for python project solutions. Every question should be clearly marked with a second-level header (using 3 "#"'s) in a Markdown cell. Sections for question solutions should be added or removed based on the number of questions in the given project. All cells are to be run and solutions displayed prior to submission.

Any format or template related questions should be asked in Piazza.

STAT 19000

STAT 29000

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 `?`. To use, simply type `?` in the console, followed by the name of the function you are interested in.

You can also look for package documentation by using `help(package=PACKAGENAME)`, so for example, to see the documentation for the package `ggplot2`, we could run:

```
help(package=ggplot2)
```

Sometimes it can be helpful to see the source code of a defined function. A function is any chunk of organized code that is used to perform an operation. Source code is the underlying `R` or `c` or `c++` code that is used to create the function. To see the source code of a defined function, type the function's name without the `()`. For example, if we were curious about what the function `Reduce` does, we could run:

```
Reduce
```

Occasionally this will be less useful as the resulting code will be code that calls `c` code we can't see. Other times it will allow you to understand the function better.

1. Make the following text (including the asterisks) bold: `This needs to be **very** bold`. Make the following text (including the underscores) italicized: `This needs to be _very_ italicized.`

Hint: *Try mixing and matching ways to embolden or italicize text. Alternatively, look up "escaping characters in markdown", or see here.*

Hint: *Be sure to check out the Rmarkdown Cheatsheet and our section on Rmarkdown in the book.*

Note: *Rmarkdown is essentially Markdown + the ability to run and display code chunks. In this question, we are actually using Markdown within Rmarkdown!*

Relevant topics: *rmarkdown, escaping characters*

Item(s) to submit:

- Fill in the chunk under (1) in the `stat29000project02template.Rmd` file with 2 lines of markdown text. Note that when compiled, this text will be unmodified, regular text.

2. Create an unordered list of your top 3 favorite academic interests (some examples could include: machine learning, operating systems, forensic accounting, etc.). Create another *ordered* list that ranks your academic interests in order of most interested to least interested.

Hint: *You can learn what ordered and unordered lists are here.*

Note: *Similar to (1a), in this question we are dealing with Markdown. If we were to copy and paste the solution to this problem in a Markdown editor, it would be the same result as when we Knit it here.*

Relevant topics: *rmarkdown*

Item(s) to submit:

- Create the lists under (2) in the `stat29000project02template.Rmd` file. Note that when compiled, this text will appear as nice, formatted lists.

3. Browse https://www.linkedin.com/ and read some profiles. Pay special attention to accounts with an "About" section. Write your own personal "About" section using Markdown. Include the following:

- A header (your choice of size) that says "About".
- A horizontal rule directly underlining the header.
- The text of your personal "About" section that you would feel comfortable uploading to linkedin, including at least 1 link.

Relevant topics: *rmarkdown*

Item(s) to submit:

- Create the described profile under (3) in the `stat29000project02template.Rmd` file.

4. Your co-worker wrote a report, and has asked you to beautify it. Knowing Rmarkdown, you agreed. Spruce up the report found under (4) in `stat29000project02template.Rmd`. At a minimum:

- Make the title pronounced.
- Make all links appear as a word or words, rather than the long-form URL.
- Organize all code into code chunks where code and output are displayed. If the output is really long, just display the code.
- Make the calls to the `library` function and the `install.packages` function be evaluated but not displayed. Make sure all warnings and errors that may eventually occur, do not appear in the final document.

Feel free to make any other changes that make the report more visually pleasing.

Relevant topics: *rmarkdown*

Item(s) to submit:

- Spruce up the "document" under (4) in the `stat29000project02template.Rmd` file.

5. Create a plot using a built-in dataset like `iris`, `mtcars`, or `Titanic`, and display the plot using a code chunk. Make sure the code used to generate the plot is hidden. Include a descriptive caption for the image.

Relevant topics: *rmarkdown, plotting in r*

Item(s) to submit:

- Code chunk under (5) that creates and displays a plot using a built-in dataset like `iris`, `mtcars`, or `Titanic`.

6. Insert the following code chunk under (5) in `stat29000project02template.Rmd`. Try knitting the document. Something should go wrong. Fix the problem and knit again. If another problem appears, fix it. What was the first problem? What was the second problem?

````
```{r install-packages}
plot(my_variable)
```
````

Hint: *Take a close look at the name we give our code chunk.*

Hint: *Take a look at the code chunk where `my_variable` is declared.*

Relevant topics: *rmarkdown*

Item(s) to submit:

- The modified version of the inserted code that fixes both problems.
- A sentence explaining what the first problem was.
- A sentence explaining what the second problem was.

Project 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 `grepl` 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 [sl
```

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 [sl
```

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 it's options to sort data in different ways.
- Use `head` to isolate n lines of output.
- Use `wc` to summarize the number of lines in a file or in output.
- Use `uniq` to filter out non-unique lines.
- Use `grep` to search files effectively.

You can find useful examples that walk you through relevant material in The Examples Book:

https://thedatamine.github.io/the-examples-book

It is highly recommended to read through, search, and explore these examples to help solve problems in this project.

Important note: It is highly recommended that you use https://rstudio.scholar.rcac.purdue.edu/. Simply click on the link and login using your Purdue account credentials. Use another system at your own risk. The version of RStudio on https://desktop.scholar.rcac.purdue.edu/ (which uses ThinLinc), is 99.9.9, and is known to have some strange issues when running code chunks.

Don't forget the very useful documentation shortcut `?`. 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

Project 7

Project 8

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 `?`. To use, simply type `?` in the console, followed by the name of the function you are interested in.

You can also look for package documentation by using `help(package=PACKAGENAME)`, so for example, to see the documentation for the package `ggplot2`, we could run:

```
help(package=ggplot2)
```

Sometimes it can be helpful to see the source code of a defined function. A function is any chunk of organized code that is used to perform an operation. Source code is the underlying `R` or `c` or `c++` code that is used to create the function. To see the source code of a defined function, type the function's name without the `()`. For example, if we were curious about what the function `Reduce` does, we could run:

```
Reduce
```

Occasionally this will be less useful as the resulting code will be code that calls `c` code we can't see. Other times it will allow you to understand the function better.

1. Make the following text (including the asterisks) bold: `This needs to be **very** bold`. Make the following text (including the underscores) italicized: `This needs to be _very_ italicized.`

Hint: *Try mixing and matching ways to embolden or italicize text. Alternatively, look up "escaping characters in markdown", or see here.*

Hint: *Be sure to check out the Rmarkdown Cheatsheet and our section on Rmarkdown in the book.*

Note: *Rmarkdown is essentially Markdown + the ability to run and display code chunks. In this question, we are actually using Markdown within Rmarkdown!*

Relevant topics: *rmarkdown, escaping characters*

Item(s) to submit:

- Fill in the chunk under (1) in the `stat29000project02template.Rmd` file with 2 lines of markdown text. Note that when compiled, this text will be unmodified, regular text.

2. Create an unordered list of your top 3 favorite academic interests (some examples could include: machine learning, operating systems, forensic accounting, etc.). Create another *ordered* list that ranks your academic interests in order of most interested to least interested.

Hint: *You can learn what ordered and unordered lists are here.*

Note: *Similar to (1a), in this question we are dealing with Markdown. If we were to copy and paste the solution to this problem in a Markdown editor, it would be the same result as when we Knit it here.*

Relevant topics: *rmarkdown*

Item(s) to submit:

- Create the lists under (2) in the `stat29000project02template.Rmd` file. Note that when compiled, this text will appear as nice, formatted lists.

3. Browse https://www.linkedin.com/ and read some profiles. Pay special attention to accounts with an "About" section. Write your own personal "About" section using Markdown. Include the following:

- A header (your choice of size) that says "About".
- A horizontal rule directly underlining the header.
- The text of your personal "About" section that you would feel comfortable uploading to linkedin, including at least 1 link.

Relevant topics: *rmarkdown*

Item(s) to submit:

- Create the described profile under (3) in the `stat29000project02template.Rmd` file.

4. 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 `grepl` 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 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`.

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 [sl
```

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 it's options to sort data in different ways.
- Use `head` to isolate n lines of output.
- Use `wc` to summarize the number of lines in a file or in output.
- Use `uniq` to filter out non-unique lines.
- Use `grep` to search files effectively.

You can find useful examples that walk you through relevant material in The Examples Book:

https://thedatamine.github.io/the-examples-book

It is highly recommended to read through, search, and explore these examples to help solve problems in this project.

Important note: It is highly recommended that you use https://rstudio.scholar.rcac.purdue.edu/. Simply click on the link and login using your Purdue account credentials. Use another system at your own risk. The version of RStudio on https://desktop.scholar.rcac.purdue.edu/ (which uses ThinLinc), is 99.9.9, and is known to have some strange issues when running code chunks.

Don't forget the very useful documentation shortcut `?`. 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 `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

Project 7

Project 8

Chapter 10

Contributors

We are extremely thankful for all of our contributors! Get your name added to the list by making a contribution.