Jai-rmarkdown-tutorial

To use Rmarkdown, we first need to install it

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

Best Practices for Scientific Computing Summary

2.1. List down any concepts or words from the paper that you did not know of, along with their definitions.

Retractions - this occurs when a paaper that has been published has been removed from a journal which may be caused by aspects such as scientific misconduct, error, plagiarism, or other situations where the credibility of the work is lost.

Modularize - modeularizing is when the functions of a program are separated into separate sub-programs. Accuracy is maintained, functionality of the program makes the code easier to comprehend, and the modularized code can allow repurposing of other projects with more ease.

Prototype - the working model of the product utilized for testing or something that can undergo replication to allow individuals tto lelarn from it. It can be made as a part of development, and although it is not complete it can represent what the final product will be like.

2.2. For each number in Box 1. Summary of Best Practices, identify and write down the best practices that you have implemented so far, in general and for this class.

Write programs for people, not computers.

- A program should not require its readers to hold more than a handful of facts in memory at once. *I have done this by ensuring that I am commenting on my code; commenting on the code in a meaningful way can allow clear and concise understanding for the reader, rather than cause an overwhelming need of memorization at one time.*
- Make names consistent, distinctive, and meaningful. *I have done this by making sure that the names are concise, and fully explain what is going on.*
- Make code style and formatting consistent. *I have done this by making sure that I do not change the format that I write my code in; instead I keep the same format to ensure that everything flows and remains consistent.*

Let the computer do the work.

- Make the computer repeat tasks. *I have done this by rerunning the code numerous times to ensure the output remains consistent.*
- Save recent commands in a file for re-use. *I have done this by ensuring that all of my code is saved in it's appropriate file, so it can be found and utilized easily without any issues.*
- Use a build tool to automate workflows. *I have done this by being specific with the files and the data so when I run the code, I can allow something to be generated again at the appropriate times.*

Make incremental changes.

- Work in small steps with frequent feedback and course correction. *I write the a few lines of code and run it rather than writing everything all together, so any mistakes or discrepancies can be identified and eliminated before enrgy is wasted continuing the code.*
- Use a version control system. *I have used Git as a control system, and its erves as a VCS and places the code in a repositories so modification and commitment of the code and changes can be done once results are ready to be shared.*
- Put everything that has been created manually in version control. *I placed the files in version control myself to ensure that the code can be reproduced efficiently.*

Don't repeat yourself (or others).

- Every piece of data must have a single authoritative representation in the system. *I ensured that the representation of the data in the code is seen once within the system to maximize conciseness and efficiency.*
- Modularize code rather than copying and pasting. *Rather than making the codes of the clone, I can make sure to modularize it so that it can be more efficient, and individuals can reproduce it in an easier way.*
- Re-use code instead of rewriting it. *Rather than trying to create my own code, I have utilized initial codes that can be made by others so that the problems can be solved by an established package.*

Plan for mistakes.

- Add assertions to programs to check their operation. *I utilized the help console on the R as well as those on the terminal and git to receive assistance when needed.*
- Use an off-the-shelf unit testing library. *I utilized the libraries that were readily available to me so that everything is uniform and the inputs are initialized.*
- Turn bugs into test cases. *I fixed the code when I saw that there were mistakes to make sure that if the same bug was to occur again, there is already something set up to stop it.*
- Use a symbolic debugger. *Using a debugger can track down the bugs which can make the fixing of the bugs more accessible and easy.*

Optimize software only after it works correctly.

- Use a profiler to identify bottlenecks. *Identification of bottlnecks through the use of a profiler allows the code to be fast and work efficiently.**
- Write code in the highest-level language possible. *Using the highest-level language possible can allow an increase in productivity to make sure that more code is written at the same time.*

Document design and purpose, not mechanics.

- Document interfaces and reasons, not implementations. *I have included a clear description in the behining of the function so I can show why the input and output is useful and the way that it can be utilized.*
- Refactor code in preference to explaining how it works. *Rather than creating a large paragraph to explain a an entire complex code, splitting the code in a way in which it doesn't need an explanation is easier.*
- Embed the documentation for a piece of software in that software. *This can allow the reference documentation to be maintain so it can increase the chances that a change in the code will allow the document to be updated at the same time.*

Collaborate.

- Use pre-merge code reviews. *This allows the code to be reviewed before or after it has been committed to a version control repository that has been shared.*
- Use pair programming when bringing someone new up to speed and when tackling particularly tricky problems. *This is when someone writes a code and the other provides feedback and tracks to ensure consistency; this can increase productivity, but can bring issues in terms of intrusiveness.*
- Use an issue tracking tool. *This can allow the organization of what exactly needs to be reviewed so that duplication of effects can be avoided by keeping a list of things that need to completed to increase productivity.*

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```