# Problem Set 1

Due January 31, 2023

### Instructions

- Read all of these instructions closely.
- This problem set is due Tuesday, January 31, 2023 at 4pm. We will talk in class on Monday, January 30 about how to submit via Github.
- Submit files via Github:
    1. the .Rmd (R Markdown) file
    2. the knitted .pdf file
    3. anything else the particular problem set might require
- Use a copy of this file, perhaps with your name or initials appended to the file name, to write your answers to the questions. You'll see there is a designated space where your answers should begin.
- Knitting the .Rmd file to a .pdf file *as you work* will ensure your code runs without errors and is working how you expect. Knit early and often. You've already read the instruction that a knitted .pdf is required when you submit. (Question 1 practices this.)
- Per the syllabus, I will not accept any late work. Keep in mind the two lowest problem set scores are dropped. Turn in what you have.

## Question 1–Getting Started with RMarkdown

This question will help orient you to RMarkdown if you haven't used it before.

### 1a

In your own words, what is the difference between R, RStudio, and RMarkdown?

Answer: R is a programming langauge. Like any language, it has rules that we have to follow. In a commonly spoken langauge, such as English, we have to put certain words in order for them to make sense. I cannot say "cat is the hat inside" because that combination of words does not make sense in that order. Instead I should probably say "The cat is inside the hat" where the subject (cat) is followed by a series of verb and positioning (is inside) and finally an object (hat). We have to communicate using R with its own set of rules.

RStudio is the interface that we use in order to actually use the language in a way that makes sense. It makes things more stremlined and user-friendly.

RMarkdown provides the user a way to take the code and written material that we create in RStudio and put it into reproducible files (probably a PDF) that can be shared easily and read to most audiences.

### 1b

When we're working in an RMarkdown file (like this one), we can easily write text around our R code. To distinguish was is R code, we put in what is called "code chunks." To see how it works, put `2+2` in the grey

area, or "code chunk," below. Then, run the code. Make sure you see the answer *both* in the RStudio console and right below the code chunk.

Answer:

```
2+2
```

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

### 1c

We can edit and run code as we're working in the .Rmd file in RStudio (that is what we did in 1b). We can further "knit" the file into a .pdf. Knitting will compile the text and run the code you've written, starting at the top of the file from scratch and working its way down. (This means it doesn't have access to objects in the global environment. Everything you need must be written in the .Rmd).

Before proceeding any further in the assignment, try knitting. Click the "Knit" button in RStudio and a .pdf of the same name should be created in the location of where this .Rmd is saved. If you can't find the "Knit" button, it is displayed as #3 on this cheatsheet.

Find the .pdf file and check that the R code in 1b was executed and is correct.

Note:

1. You may need to install an additional library or two for the .pdf to knit. I've included that code below. Ping me in Slack as needed to debug any issues you might encounter on your individual computer.
2. If you want to see someone walk through these steps, I recommend this short video.

```
#tinytex::install_tinytex()
```

## Question 2–Agility with Vectors and Functions

We learned several functions in class, like `rep` and `seq`. This question asks you to use a few more, like `paste0`, to practice working with vectors and functions. Developing agility with these R basics will help you with broader skills like cleaning data.

Hint: Look at the helpfiles for these functions if they are new to you by typing, for example, `?paste0` in the console and viewing the file that populates in the "Help" tab in the bottom right of RStudio.

### 2a

Create a vector called `x` of the elements 1 to 100 (i.e., $[1, 100]$). Then, make every odd-indexed element of the vector $x$ negatively signed.

Answer:

```
x <- seq(1,100)
#this code will make every other character negative
x[c(seq(1,100,by=2))] <- -x[c(seq(1,100,by=2))]
list(x)
```

```
## [[1]]
##   [1]  -1   2  -3   4  -5   6  -7   8  -9  10 -11  12 -13  14 -15  16 -17  18
##  [19] -19  20 -21  22 -23  24 -25  26 -27  28 -29  30 -31  32 -33  34 -35  36
##  [37] -37  38 -39  40 -41  42 -43  44 -45  46 -47  48 -49  50 -51  52 -53  54
##  [55] -55  56 -57  58 -59  60 -61  62 -63  64 -65  66 -67  68 -69  70 -71  72
##  [73] -73  74 -75  76 -77  78 -79  80 -81  82 -83  84 -85  86 -87  88 -89  90
##  [91] -91  92 -93  94 -95  96 -97  98 -99 100
```

### 2b

Use the `seq()` and `paste0()` functions to create the vector called `varnames` with the following structure: `c("Var10", "Var20", "Var30", "Var40", "Var50", "Var60")`

Answer:

```
varnames <- c(paste0("Var", seq(10,60, by= 10)))
varnames
```

```
## [1] "Var10" "Var20" "Var30" "Var40" "Var50" "Var60"
```

### 2c

Using the vector you created in 2b, write code that removes the leading "Var" from each element of the vector. Call the new vector `varnames_str`, and the resulting vector should be the following strings: `c("10", "20", "30", "40", "50", "60")`

Hint: you could use the `substr` function.

```
varnames_str <- substring(varnames, 4)
varnames_str
```

```
## [1] "10" "20" "30" "40" "50" "60"
```

### 2d

Use the vector `varnames_str` created in 2c and change the string elements into numeric values, naming this vector `varnames_num`. The new vector should look like this: `c(10, 20, 30, 40, 50, 60)`

```
varnames_num <- as.numeric(varnames_str)
varnames_num
```

```
## [1] 10 20 30 40 50 60
```

### 2e

Create a subset the vector `varnames_num` from 2d that only includes the numbers divisible by 4. Name this vector `varnames_d4`.

Hint: you might use the modulo function, `%%`.

```
varnames_d4 <- varnames_num[varnames_num %% 4 == 0]
varnames_d4
```

```
## [1] 20 40 60
```

### 2f

Uncomment and run the following code and describe exactly what is happening (what R functionality and calculations) to get this result.

Hint: it might help to write the math out for each element.

```
varnames_num - varnames_d4
```

```
## [1] -10 -20 -30  20  10   0
```

In the following code we are taking the vector varnames_num which is a series of numbers 10-60 by 10s and subtracting the vector varnames_d4 whcih is the varnames_num vector but only the numbers divisible by 4.

(10 - 20) = -10 (20 - 40) = -20 (30 - 60) = -30 (40 - 20) = 20 (50 - 40) = 10 (60 - 60) = 0

## Question 3–Math in R

Like the inclass activity, this question asks you to practice translating math equations into R. While you might not plan to do a lot of math in R in the future, this question helps make R rules and syntax second-nature e.g., closing all open parentheses, knowing how to work with vectorized operations, etc.

Hint: look at the compiled .pdf file I provided to see the math in an easily-readable format and ignore the code between the pink dollar signs.

### 3a

$\frac{3}{7} \times 2^{\frac{1}{5}} \times \sum_{i=1}^{50} i$

```
(3/7)*(2^(1/5))*(sum(1:50))
```

```
## [1] 627.6816
```

### 3b

$\sum_{i=1}^{50} (i + 10)$

```
sum(1:50 +10)
```

```
## [1] 1775
```

### 3c

$\forall i \in [1, 10],\ i^{\sqrt{i+2}}$

```
threec <- function(x){
  return(x**sqrt(x+2))
}

threec(1:10)
```

```
##  [1]    1.00000    4.00000   11.66475   29.83594   70.68069  158.83423
##  [7]  343.00000  717.49898 1461.72058 2911.39824
```

## Question 4–Writing Functions

### 1a

Write a function named `sample_var` that takes one input, a numeric vector `x`, and calculates the standard sample variance formula. Have the function return the answer.

$$\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

```
sample_var <- function(x){
  xbar <- mean(x)
  n <- length(x)
  ss <- sum((x-xbar)^2)
  var <- ss/(1/(n-1))
  return(var)
}
```

### 1b

Use the function you wrote in 1a to calculate the sample variance of the following vector. You can use the built-in function in R to double check your function is correct

```
vec <- c(2, 5, 10, 12, 3, 3, 4, 1, 20)
sample_var(vec)
```

```
## [1] 2464
```

### 1c

Now, try running your function on the following vector. Why won't it work?

```
vec_str <- c("2", "5", "10", "12", "3", "3", "4", "1", "20")
```

Likely because these are in string format. We could use the command as.numeric() to change the values and then run the function on that vector