Homework 1

Samriddh Gupta

1. (2 pts) In the second variant of rescale01(), infinite values are left unchanged. Rewrite rescale01() so that -Inf is mapped to 0, and Inf is mapped to 1. Add error checking to return the message "Error: input must be numeric" if the input is not all numeric.

```
rescale01 <- function(x){
  if(is.numeric(x)==FALSE){
    stop("Error: input must be numeric")
  }
  x=replace(x,x==Inf,1)
  x=replace(x,x==-Inf,0)
  rng <- range(x, na.rm = TRUE, finite=F)
  (x - rng[1]) / (rng[2] - rng[1])
}

#rescale01(c(0,10,2,"a")) ## gives you the error we want
rescale01(c(3,4,10,Inf,-Inf))
```

```
## [1] 0.3 0.4 1.0 0.1 0.0
```

```
rescale01(c(3,4,1,Inf,-Inf))
```

```
## [1] 0.75 1.00 0.25 0.25 0.00
```

2. Write both_na(), a function that takes two vectors of the same length and returns the number of positions that have an NA in both vectors.

```
both_na<- function(x,y){
  if (length(x) != length(y)) {
    stop("`x` and `y` must be the same length", call. = FALSE)
  }
  which(is.na(x) & is.na(y))
}


both_na(c(1, 2, NA, 2, NA), c(1, 2, 3, 4, NA))
```

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

```
both_na(c(1, 2, NA, 2, NA), c(1, 2, NA, 4, NA))
```

```
## [1] 3 5
```

3. Read the source code for each of the following three functions, puzzle out what they do, and then brainstorm better names.

```r
f1 <- function(string, prefix) {
  substr(string, 1, nchar(prefix)) == prefix
}
f2 <- function(x) {
  if (length(x) <= 1) return(NULL)
  x[-length(x)]
}
f3 <- function(x, y) {
  rep(y, length.out = length(x))
}

f1("Mr Samriddh gupta","mr")
```

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

```r
f1("Mr Samriddh gupta","Mr")
```

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

```r
f1("Helloworld","Hello")
```

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

```r
# f1 tells you whether the string is has the first part same as prefix or second string.


f2(c(1,2,3,4,5))
```

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

```r
f2(c(1,2,3,4))
```

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

```r
f2("hello")
```

```
## NULL
```

```r
#f2 returns the strng with less than 1 last character. If there is only one character, then it returns

f3("samriddh12","helloworld")
```

```
## [1] "helloworld"
```

```r
f3("samriddh","hello")
```

```
## [1] "hello"
```

```r
f3("samriddh12","hello")
```

```
## [1] "hello"
```

```r
# f3 replaces the length of the second string with first string.
```

4. Write a greeting function that says "good morning", "good afternoon", or "good evening", depending on the time of day. (Hint: use a time argument that defaults to lubridate::now(). That will make it easier to test your function.

```r
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':
##
##     date
```

```r
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x lubridate::as.difftime() masks base::as.difftime()
## x lubridate::date()        masks base::date()
## x dplyr::filter()          masks stats::filter()
## x lubridate::intersect()   masks base::intersect()
## x dplyr::lag()             masks stats::lag()
## x lubridate::setdiff()     masks base::setdiff()
## x lubridate::union()       masks base::union()
```

```r
greeting <- function(now = now()) {
  if (between(hour(now), 6, 12)) {# my morning is from 6 to 12
    print("Good morning")
  }
  else if (between(hour(now), 12, 15)) { # my afternoon is from 12 to 3 pm
    print("Good afternoon")
  }
  else if(between(hour(now),15,21)){ # my evening is from 3 pm to 9 pm
```

```r
    print("Good evening")
  }
  else{
    print("Good Night") ## my edition to good night
  }
}
greeting(now())
```

```
## [1] "Good morning"
```

5. Implement a fizzbuzz function. It takes a single number as input. If the number is divisible by three, it returns "fizz". If it's divisible by five it returns "buzz". If it's divisible by three and five, it returns "fizzbuzz". Otherwise, it returns the number. Make sure you first write working code before you create the function.

```r
fizzbuzz<- function(x){
  if(x%%3==0){
    if (x%%5==0){
      print("fizzbuzz")
    }
    else{
      print("fizz")
    }
  }
  else if (x%%5==0){
    print("buzz")
  }
  else{
    return(x)
  }
}



fizzbuzz(21)
```

```
## [1] "fizz"
```

```r
fizzbuzz(85)
```

```
## [1] "buzz"
```

```r
fizzbuzz(45)
```

```
## [1] "fizzbuzz"
```

```r
fizzbuzz(76)
```

```
## [1] 76
```

6. How could you use cut() to simplify this set of nested if-else statements? if (temp <= 0) { "freezing" } else if (temp <= 10) { "cold" } else if (temp <= 20) { "cool" } else if (temp <= 30) { "warm" } else { "hot" } How would you change the call to cut() if I'd used < instead of <=? What is the other chief advantage of cut() for this problem? (Hint: what happens if you have many values in temp?)

```
weather<-function(temp){
  cut(temp, breaks = seq(-10, 40, 10),
    labels = c("freezing", "cold", "cool", "warm", "hot"))
}

weather(30)
```

```
## [1] warm
## Levels: freezing cold cool warm hot
```

```
weather(-2)
```

```
## [1] freezing
## Levels: freezing cold cool warm hot
```

```
weather(10)
```

```
## [1] cold
## Levels: freezing cold cool warm hot
```

if we need to use < instead of <= we need to add a liitle change to the code

```
weather2<- function(temp){
  cut(temp, breaks = seq(-10, 40, 10),
    right = FALSE,
    labels = c("freezing", "cold", "cool", "warm", "hot"))
}
weather2(30)
```

```
## [1] hot
## Levels: freezing cold cool warm hot
```

```
weather2(-2)
```

```
## [1] freezing
## Levels: freezing cold cool warm hot
```

```
weather2(10)
```

```
## [1] cool
## Levels: freezing cold cool warm hot
```

the advantages of using cut() is that the code gets shorter and we do need to change at many places if we are changing the deminesions in the future.

Reference

I took a liitle bit of help from this website https://github.com/cimentadaj/R4DS-Solutions/blob/master/ch15.Rmd