setwd("C:/Users/Adrian/OneDrive - Northeastern University/R folder") Refer to R-cokbook! http://www.cookbook-r.com/ (http://www.cookbook-r.com/) —

Math in R

```
1+1
```

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

```
sqrt(2)
```

```
## [1] 1.414214
```

```
log(1000)
```

```
## [1] 6.907755
```

```
# R can also do many other things using built-in method; more math operation; look to
```

Working with variables: Create two vectors named v1 and v2

```
# To assign a variable, rather than use the = sign (though that also works), we use  <
-
v1 <- c(2:6)
v2 <- c(5:9)
v1
```

```
## [1] 2 3 4 5 6
```

```
v2
```

```
## [1] 5 6 7 8 9
```

```
# to remove it
rm(v2)
```

Working with data types

```
# class will tell us what type something already is:
class(2.8)
```

```
## [1] "numeric"
```

```
# changig data types:
## inquire about specific types with is.TYPE: is.numeric; is.factor(); is.character()
is.character("2014-08-13")
```

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

```
## coerce a variable of one type into another with as.TYPE: as.numeric; as.character
(); as.factor; as.Date
properdate <- as.Date("2014-08-13")
as.character(properdate)
```

```
## [1] "2014-08-13"
```

```
## Logical (TRUE/FALSE) is another type, and is the output of many basic tests - equal
ity ==, inequality !=, greater than >, less-than-or-equal-to <=, etc.
1 == 2
```

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

Working with Vectors

```
#A vector is an ordered sequence of numbers. One way to create a vector is with  c()
v <- c(1,5,9,7,2)
v
```

```
## [1] 1 5 9 7 2
```

```
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
# vector properties

##Length of a vector
length(v)
```

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

```
##pick out specific elements from the vector:
v[3] # what is the elements in the third position of the vector
```

```
## [1] 9
```

```
v[3:5] # the elements from positon 3 to 5
```

```
## [1] 9 7 2
```

```
#modify vector
v[3:5] <- c(17,0,11) # reassign values from position 3 to 5
v
```

```
## [1]  1  5 17  0 11
```

```
v[c(2,4)] # The result is a new vector consisting of the second and fourth elements fr
om v
```

```
## [1] 5 0
```

```
#vector math
3+v # Standard mathematical operations on vectors affect each element individually
```

```
## [1]  4  8 20  3 14
```

```
w <- 1:5 #Vectors of the same length can be added

which(v < 3) # do logical checks on vectors
```

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

```
# Replace the elements in v1+v2 that are greater than 10 with the number 0. Show that
vector.
v1 <- c(2:6)
v2 <- c(5:9)
u <- v1+v2
u[u > 10] <- 0 # using logical function
u1 <- ifelse(v1+v2 > 10, 0, v1+v2) # using ifelse statement
u2 <- replace(v1+v2, v1+v2 > 10, 0) # using the raplce function
u1
```

```
## [1] 7 9 0 0 0
```

```
u2
```

```
## [1] 7 9 0 0 0
```

Working with matrix

```
v1 <- c(2:6)
m1 <- matrix(1:25,nrow=5,ncol=5)
m1
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    6   11   16   21
## [2,]    2    7   12   17   22
## [3,]    3    8   13   18   23
## [4,]    4    9   14   19   24
## [5,]    5   10   15   20   25
```

```
dim(m1) # note the dimention of a matrix
```

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

```
ncol(m1)
```

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

```
nrow(m1)
```

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

```
# matrix algebra
m1 %*% v1 # m1 times v1?
```

```
##      [,1]
## [1,]  270
## [2,]  290
## [3,]  310
## [4,]  330
## [5,]  350
```

```
v1 %*% m1 #  v1 times m1?
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   70  170  270  370  470
```

```
m <- rbind(m1, v1) # bind two vectors as rows together

rbind(m1,m1) # we can do the same with matrices, and then it matters: cbind() glues th
em side-by-side, and rbind() glues them above and below.
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    1    6   11   16   21
##  [2,]    2    7   12   17   22
##  [3,]    3    8   13   18   23
##  [4,]    4    9   14   19   24
##  [5,]    5   10   15   20   25
##  [6,]    1    6   11   16   21
##  [7,]    2    7   12   17   22
##  [8,]    3    8   13   18   23
##  [9,]    4    9   14   19   24
## [10,]    5   10   15   20   25
```

```
m1 %*% t(m1) # m1 times the transpose of m1?
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]  855  910  965 1020 1075
## [2,]  910  970 1030 1090 1150
## [3,]  965 1030 1095 1160 1225
## [4,] 1020 1090 1160 1230 1300
## [5,] 1075 1150 1225 1300 1375
```

```
# pick elements or subsets from matrixs
m1[2, 1:2] # pick elements from second row, first to second coloum
```

```
## [1] 2 7
```

```
m1[ , 1:2] # pick elements from all rows, first to second coloum
```

```
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    3    8
## [4,]    4    9
## [5,]    5   10
```

```
m1 == 3
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]
## [1,] FALSE FALSE FALSE FALSE FALSE
## [2,] FALSE FALSE FALSE FALSE FALSE
## [3,]  TRUE FALSE FALSE FALSE FALSE
## [4,] FALSE FALSE FALSE FALSE FALSE
## [5,] FALSE FALSE FALSE FALSE FALSE
```

```
which(m1 == 3)
```

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

```
m1[which(m1 == 3)] <- 5 # Note that the elements in a matrix are ordered as we saw the
m assigned: down first, then across.
```

Working with data frame

```
# Create a date frame with at least five rows and three columns.

options(stringsAsFactors=F) #set the global options in R so that the stringsAsFactors argument in functions that use it is always false
dates <- seq.Date(from=Sys.Date(), by=1, length.out=5) # the second variable should be strings (characters)
str_dates <- as.character(dates) # Data type coercion, a double-edged sword in R
nums <- 1:5 # the third variable should be numbers
df2 <- data.frame(dates=dates, str_dates=str_dates, nums=nums) # The left hand side of each '=' is a name for the new column;  The right hand side is a vector that you've already defined.

colnames(df2) <- c("Date","Strings","Num") # change the column name
df2
```

```
##         Date    Strings Num
## 1 2019-01-17 2019-01-17   1
## 2 2019-01-18 2019-01-18   2
## 3 2019-01-19 2019-01-19   3
## 4 2019-01-20 2019-01-20   4
## 5 2019-01-21 2019-01-21   5
```

```
str(df2) # Use str() to show that your data frame is appropriately structured.
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ Date   : Date, format: "2019-01-17" "2019-01-18" ...
##  $ Strings: chr  "2019-01-17" "2019-01-18" "2019-01-19" "2019-01-20" ...
##  $ Num    : int  1 2 3 4 5
```

```
ourcol <- df2$Date # To work with one of the columns, we use the $ to pick it out of the data frame

ourcol # ourcol is now a new vector with values taken from the first column of the data frame.
```

```
## [1] "2019-01-17" "2019-01-18" "2019-01-19" "2019-01-20" "2019-01-21"
```

```
df2[2:3,2:3] # pull out subsets of the dataframe
```

```
##      Strings Num
## 2 2019-01-18   2
## 3 2019-01-19   3
```

```
df2[2:3,c("Date","Strings")] # pull out the same using column names
```

```
##         Date    Strings
## 2 2019-01-18 2019-01-18
## 3 2019-01-19 2019-01-19
```

```
df2[c(1,3),"Date"]
```

```
## [1] "2019-01-17" "2019-01-19"
```

```
df2$Date[c(1,3)] #these two do the same
```

```
## [1] "2019-01-17" "2019-01-19"
```

```
# edit the dataframe
df2$Num <- as.character(df2$Num)
str(df2)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ Date   : Date, format: "2019-01-17" "2019-01-18" ...
##  $ Strings: chr  "2019-01-17" "2019-01-18" "2019-01-19" "2019-01-20" ...
##  $ Num    : chr  "1" "2" "3" "4" ...
```

```
View(df2) #view certain datasets in a graphical manner
```

Save it as a csv file, and then reload the data from the csv file. R can also read many non-text data formats, including those from SPSS, STATA, SAS, and others. To do so you need to have installed and load the foreign package, and then use (for instance) read.dta() rather than read.table().

```
write.table(df2,file="homework1df.csv",row.names=FALSE,sep=",")
newdf <- read.table(file="homework1df.csv",header=TRUE,sep=",",stringsAsFactors=FALSE)
newdf
```

```
##         Date    Strings Num
## 1 2019-01-17 2019-01-17   1
## 2 2019-01-18 2019-01-18   2
## 3 2019-01-19 2019-01-19   3
## 4 2019-01-20 2019-01-20   4
## 5 2019-01-21 2019-01-21   5
```

```
head(newdf) # to quickly check a file
```

```
##         Date    Strings Num
## 1 2019-01-17 2019-01-17   1
## 2 2019-01-18 2019-01-18   2
## 3 2019-01-19 2019-01-19   3
## 4 2019-01-20 2019-01-20   4
## 5 2019-01-21 2019-01-21   5
```

practice: Replace all the even numbers in the original data frame with 0.

```
df <- data.frame(dates=dates, str_dates=str_dates, nums=nums)
df[df[,3] %% 2 == 0,3] <- 0 # We can do it by column number
df$Num[df$nums %% 2 == 0] <- 0 # Or better, by column name
df$nums <- ifelse(df$nums %% 2 == 0, 0, df$nums) #less elegantly, using ifelse
df$nums <- replace(df$nums, df$nums %% 2 == 0, 0) # or with replace
```

Working with list

```
list1 <- list(v1,v2,m1,df2) #Create a list with v1, v2, m1, and your data frame.

list1
```

```
## [[1]]
## [1] 2 3 4 5 6
##
## [[2]]
## [1] 5 6 7 8 9
##
## [[3]]
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    6   11   16   21
## [2,]    2    7   12   17   22
## [3,]    5    8   13   18   23
## [4,]    4    9   14   19   24
## [5,]    5   10   15   20   25
##
## [[4]]
##         Date    Strings Num
## 1 2019-01-17 2019-01-17   1
## 2 2019-01-18 2019-01-18   2
## 3 2019-01-19 2019-01-19   3
## 4 2019-01-20 2019-01-20   4
## 5 2019-01-21 2019-01-21   5
```

```
list1[[2]][2] # the objects in a list are indexed with the double bracket [[ ]]; in th
is case, it pick the second object in the list, and 2rd element in that vector
```

```
## [1] 6
```

```
names(list1) <- c("item1","item2", "item3", "item4") # name the object in the list

list1$item3[2] #pick out the third item's second item.
```

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

Using latex equation notation in your .Rmd file, write out the quadratic formula

ax^2 +bx +c=0

Remember that you can find more latex symbols and tips online or in the cribsheet here: http://faculty.cbu.ca/srodney/ShortSymbInd.pdf (http://faculty.cbu.ca/srodney/ShortSymbInd.pdf) .

Working with functions and packages

```
# install.packages()
# library()

# define the function named "doubleit"
## The first line specifies the name of the function ("doubleit") and the arguments o
f the function (its inputs x).
## The input gets assigned temporarily to "x", and then multiplied by 2 with the outpu
t stored temporarily in "doubled"
## The final line tells the function that the value of "doubled" is the output from th
e function.
doubleit <- function(x){
  doubled <- x*2
  return(doubled)
}

# now use the function with an input of 7 and save it as sevendoubled
sevendoubled <- doubleit(7)
sevendoubled
```

```
## [1] 14
```

```
#here is a function that takes two numbers and calculates both their sum and differenc
e:
sumdiff <- function(a,b){
  nsum <- a + b
  ndif <- a - b
  return(c(nsum,ndif))
}
```

Working with logical function and loop

```
#use of brackets  { } and parentheses to let the computer know where chunks start and
end.
#R will ignore blank space in your script, so choooose a stule that you like the most

# if ... else function

if(3 < 4){
  print("yup")
}
```

```
## [1] "yup"
```

```
# write if in a function
isitthree <- function(x){
  if(x == 3){
    return(TRUE)
  }else{
    return(FALSE)
  }
}
isitthree(3)
```

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

```
## this code can be shorten using ifelse
isitless <- ifelse(3<4,1,0) # the first argument is the test, the second is the outpu
t for if it is true, and the third is the output for if the test is false. But ifelse
() is only a good substitute of if() { } else {} if you are conducting element-wise lo
gical tests.

ifelse(c(TRUE,TRUE,FALSE,TRUE), v1, 0)
```

```
## [1] 2 3 0 5
```

Working with loops:

```
# for loop is the most common loop function

for(i in 1:3){
  print(2*i)
}
```

```
## [1] 2
## [1] 4
## [1] 6
```

```
## i takes all the value in 1:3, then print out all value 2*i

## Here i, like in a function, is a local variable, used within the loop only.
## After the "in" comes a vector (eg, 1:3); it can be any vector, including a column o
f data.

# while loops

i <- 1
while(i <= 2){
  print(3*i)
  i <- i + 1
}
```

```
## [1] 3
## [1] 6
```

```
# using break to interrupts the loop if some condition is met partway
i <- 1
for(i in 1:3){
  if(i == 2){
    break
  }
  print(i)
}
```

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

Apply function

```
m <- matrix(1:6,nrow=2,ncol=3)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
apply(m,2,mean) # m is the input data, 2 means we apply the function over columns (1 i
s rows), and mean is the function we are applying. We can use any function we want her
e, including those of our own creation.
```

```
## [1] 1.5 3.5 5.5
```

```
summary(m)
```

```
##        V1             V2             V3
##  Min.   :1.00   Min.   :3.00   Min.   :5.00
##  1st Qu.:1.25   1st Qu.:3.25   1st Qu.:5.25
##  Median :1.50   Median :3.50   Median :5.50
##  Mean   :1.50   Mean   :3.50   Mean   :5.50
##  3rd Qu.:1.75   3rd Qu.:3.75   3rd Qu.:5.75
##  Max.   :2.00   Max.   :4.00   Max.   :6.00
```

```
## There are many other apply functions designed for different input data.
```