First R project

Initialize Project

1. To begin open RStudio
2. Click **File -> New project**
3. Choose **New directory**
4. Choose **New project**

    a. **Make sure to check start new session box in the bottom right corner**

5. Choose a location for your project

IMPORTANT

In deciding where you are going to put this project you should keep in mind a couple things. First, you should have a central location for all your code. This is not just a good organizational idea, but it will help you and others quickly find what is needed. It is also a a requirement of this lab/class. Think of it as a digital lab notebook.For example, all of my lab code is located here: /Users/cgaulke/Documents/research/. Each of my projects gets a new subdirectory, for this exercise it is: /Users/cgaulke/Documents/research/R_training.

6. After you have selected a location select **create project**

    a. If everything has gone right you should end up with a new RStudio window with three tabs (Console, environment, and project).

7. Now select **File –> New File –> R Script**

    a. This is a code document, which is basically a glorified text document with syntax highlighting. This is where you will right and save your code.

    Note: Code executed in the console is not saved, so you do need a R Script

8. Now lets go over some basics. . .

Comments

In your R Script everything is code, which can be bad if you need to explain what you are doing.

```
This assigns 2 to x
x <- 2
```

```
## Error: <text>:1:6: unexpected symbol
## 1: This assigns
##         ^
```

This creates a error, which is bad in R because it usually stops whatever code you are running without returning the value you asked for. If we what to explain ourselves we can use a comment

```
#This assigns 2 to x
x <- 2
x
```

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

The comment character "#" allows us to tell the interpreter (computer) to ignore that bit of code. Not all code needs to be commented, but as a general rule if you cant glance at it and figure out whats going on you should use a comment

---

R Data types

There are many types of R-objects. We will use the following frequently

- Vectors
- Lists
- Matrices
- Data Frames
- Arrays
- Factors

Each data type has various strengths and purposes.

Vectors

The simplest of these objects is the vector object. When these objects are made from the basic (atomic) data types (logical, integer, real, complex, string and raw) they are termed atomic vectors. Below we will take a closer look at these data types

```
#Logical aka Boolean, TRUE/FALSE
v <- TRUE
print(class(v))
```

```
## [1] "logical"
```

```
#Numeric ... Just what it sounds like
v <- 23.5
print(class(v))
```

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

```
#Integer Note that R integers are 32 bit... and that sucks
v <- 2L
print(class(v))
```

```
## [1] "integer"
```

```r
#Complex imaginary
v <- 2+5i
print(class(v))
```

```
## [1] "complex"
```

```r
#Character
v <- "TRUE"
print(class(v))
```

```
## [1] "character"
```

```r
#Raw (don't worry about this one yet)
v <- charToRaw("Hello")
print(class(v))
```

```
## [1] "raw"
```

These basic (atomic) data types can be combined into larger data types such as data frames, matrices, etc. If the length is a vector is more than one we create it with the special c() function.

```r
#Logical aka Boolean, TRUE/FALSE
v <- c(1,2,3)
print(class(v))
```

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

```r
length(v)
```

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

Vectors, can be any length, but only one type. If there is more than one type then the vector is coerced to a single type as above.

```r
#make a mixed vector with character
v <- c(1,2,"3")
print(class(v))
```

```
## [1] "character"
```

```r
v
```

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

```r
#make a mixed vector with decimal
v <- c(1,2.1,3)
print(class(v))
```

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

```
v
```

```
## [1] 1.0 2.1 3.0
```

```
#make a mixed vector with integer
v <- c(1,2.1,3L)
print(class(v))
```

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

```
v
```

```
## [1] 1.0 2.1 3.0
```

Lists

If you need to store different types and sizes of data together you can uses lists

```
# Create a list.
list1 <- list(c(1,2,3),"chris",sin)

# Print the list.
print(list1)
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "chris"
##
## [[3]]
## function (x)  .Primitive("sin")
```

```
class(list1)
```

```
## [1] "list"
```

Matrices

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.

```
# Create a matrix.
M = matrix( c('a','a','b','c','b','a'), nrow = 2, ncol = 3, byrow = TRUE)
print(M)
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "a"  "b"
## [2,] "c"  "b"  "a"
```

Arrays

While matrices are confined to two dimensions, arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension. In the below example we create an array with two elements which are 3x3 matrices each.

```r
# Create an array.
a <- array(c('boston','terriers'),dim = c(3,3,2))
print(a)
```

```
## , , 1
##
##      [,1]       [,2]       [,3]
## [1,] "boston"   "terriers" "boston"
## [2,] "terriers" "boston"   "terriers"
## [3,] "boston"   "terriers" "boston"
##
## , , 2
##
##      [,1]       [,2]       [,3]
## [1,] "terriers" "boston"   "terriers"
## [2,] "boston"   "terriers" "boston"
## [3,] "terriers" "boston"   "terriers"
```

Factors

Factors are vectors of values and labels. They are very useful for coding data for plotting or statistical analysis, the labels are always character irrespective of its original data type.

Factors are created using the factor() function. The nlevels functions gives the count of levels.

```r
# Create a vector.
eye_colors <- c('green','green','brown','blue','hazel','blue','green')

# Create a factor object.
factor_ec <- factor(eye_colors)

# Print the factor.
print(factor_ec)
```

```
## [1] green green brown blue  hazel blue  green
## Levels: blue brown green hazel
```

```r
print(nlevels(factor_ec))
```

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

Data Frames

Data frames are data objects in which each column can contain different types of data. Essentially it is list of vectors of equal length.

Data Frames are created using the data.frame() function.

```
# Make data frame.
BMI <-  data.frame(
   gender = c("Male", "Female","Female"),
   height = c(152, 150, 165),
   weight = c(81,70, 78),
   Age = c(33,22,57)
)
print(BMI)
```

```
##    gender height weight Age
## 1   Male    152     81  33
## 2 Female    150     70  22
## 3 Female    165     78  57
```

Data frames are really important in data analysis so it is worth spending a little extra time exploring their
features. We will start with how to view data frames

```
#make a larger data frame

df <-   data.frame(
   letter1 = letters[1:20],
   letter2 = LETTERS[1:20],
   num1 = 1:20,
   num2 = 21:40,
   num3 = 41:60
)

#View in a new window
View(df)
```

```
## Error in check_for_XQuartz(): X11 library is missing: install XQuartz from xquartz.macosforge.org
```

```
#for very large data frames it can sometimes be useful to look at a few entries
#and not the whole thing to do this we can

#get the first 10 rows
head(df, 10)
```

```
##    letter1 letter2 num1 num2 num3
## 1        a       A    1   21   41
## 2        b       B    2   22   42
## 3        c       C    3   23   43
## 4        d       D    4   24   44
## 5        e       E    5   25   45
## 6        f       F    6   26   46
## 7        g       G    7   27   47
## 8        h       H    8   28   48
## 9        i       I    9   29   49
## 10       j       J   10   30   50
```

```
#get the last 10 rows
tail(df, 10)
```

```
##    letter1 letter2 num1 num2 num3
## 11       k       K   11   31   51
## 12       l       L   12   32   52
## 13       m       M   13   33   53
## 14       n       N   14   34   54
## 15       o       O   15   35   55
## 16       p       P   16   36   56
## 17       q       Q   17   37   57
## 18       r       R   18   38   58
## 19       s       S   19   39   59
## 20       t       T   20   40   60
```

```
#you can change the number 10 to any number you like even if it is larger than
#the number of rows
```

Its also nice to be able to get some general information about data tables because they are often large

```
# Make data frame.
BMI <-  data.frame(
    gender = c("Male", "Female","Female"),
    height = c(152, 150, 165),
    weight = c(81,70, 78),
    Age = c(33,22,57)
)

#get class
class(BMI)
```

```
## [1] "data.frame"
```

```
#Print column names
colnames(BMI)
```

```
## [1] "gender" "height" "weight" "Age"
```

```
#Print row names
rownames(BMI)
```

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

```
#row and column names are stored invisibly as data object attributes. When we use
#the functions to return the row or colnames we are actually accessing these data.
#To get all the names at once we can use
attributes(BMI)
```

```
## $names
## [1] "gender" "height" "weight" "Age"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] 1 2 3
```

```
#How many rows and columns?
dim(BMI)  #gives the number of rows and columns
```

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

```
nrow(BMI) #gives only rows
```

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

```
ncol(BMI) #gives only columns
```

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

```
#note length does not work as expected here
length(BMI)
```

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

There are many ways to access the data inside a data frame and, as usual, they all have their strengths and limitations. The most versatile is the subset function '['. Using this notation a data frame, df, is split by indices you provide i.e., df[row_index, column_index]. You must include both indices even if you are only interested in a column or row. Examples are provided below

```
# Make data frame.
BMI <-  data.frame(
   gender = c("Male", "Female","Female"),
   height = c(152, 150, 165),
   weight = c(81,70, 78),
   Age = c(33,22,57)
)

BMI
```

```
##    gender height weight Age
## 1    Male    152     81  33
## 2 Female    150     70  22
## 3 Female    165     78  57
```

```r
#get the first entry in row 1 column 1
BMI[1,1]
```

```
## [1] "Male"
```

```r
#alternatively call by row and column names
BMI[1,"gender"]
```

```
## [1] "Male"
```

```r
#what if we want to get all data in a column?
#to get all data in column 4 (Age)
BMI[,4]
```

```
## [1] 33 22 57
```

```r
#you can also use the quoted column name to get the same info
BMI[,"Age"]
```

```
## [1] 33 22 57
```

```r
# as a bonus in data frames we can also use the '$' operator to get column data
BMI$Age
```

```
## [1] 33 22 57
```

```r
#to access all data in a row
BMI[1,]
```

```
##   gender height weight Age
## 1   Male    152     81  33
```

```r
#just to confirm that data frames can contain multiple data types
class(BMI$gender)
```

```
## [1] "character"
```

```r
class(BMI$weight)
```

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

These techniques can be useful if you need to add to or change data in a data frame

```r
# Make data frame.
BMI <-  data.frame(
   gender = c("Male", "Female","Female"),
   height = c(152, 150, 165),
   weight = c(81,70, 78),
   Age = c(33,22,57)
)

BMI
```

```
##    gender height weight Age
## 1   Male    152     81  33
## 2 Female    150     70  22
## 3 Female    165     78  57
```

```r
#We mistakenly coded patient 2 as a female and need to change it. To do this we
#can use the 'gets' (aka, assign) operator '<-'

BMI[2,1] <- "Male"
BMI
```

```
##    gender height weight Age
## 1   Male    152     81  33
## 2   Male    150     70  22
## 3 Female    165     78  57
```

```r
#we can also add whole columns or rows if we want. Lets say we enrolled a new patient in our study

BMI[4,] <- list("Female",140, 72,68 ) #not we use a list ... Why ?

#Now we also forgot that we collected information on smoking status. We can add
#this too
BMI[,5] <- c(0,1,0,1)

#But the column names are now odd
colnames(BMI)
```

```
## [1] "gender" "height" "weight" "Age"    "V5"
```

```r
#we can fix this too using the colnames function and the assignment operator
colnames(BMI) <- c("gender", "height", "weight", "Age", "smoker")

#but why is age the only thing capitalized, thats weird, but we can fix it
class(colnames(BMI))
```

```
## [1] "character"
```

```r
#looks like colnames are just a character vector and just like data frames these
# can be subset using '[' . The only difference is there is only one dimension
colnames(BMI)[4] <- "age"
```

```r
#We can do the same thing with row names, say we wanted to add patient IDs to the
#data

rownames(BMI) <- c("Patient1","Patient2","Patient3","Patient4")

BMI
```

```
##          gender height weight age smoker
## Patient1   Male    152     81  33      0
## Patient2   Male    150     70  22      1
## Patient3 Female    165     78  57      0
## Patient4 Female    140     72  68      1
```

```r
#now to get the gender of Patient1 we can use

BMI["Patient1", "gender"]
```

```
## [1] "Male"
```

It should be noted that all of the approaches outlined for data frames will work with matrices except for the '$' operator. But, it should be noted that matrices can only contain 1 data type unlike data frames which can encode columns with different data types