Student Exercises | Intro to R

your name here

```
library(knitr)
opts_chunk$set(tidy.opts=list(width.cutoff=60),tidy=TRUE)
```

Exercise 1.1: Exploring R Studio

1. Take a few minutes to familiarize yourself with the R studio environment by locating the following features:

- The windows clockwise from top left are: the code editor, the workspace and history, the plots and files window, and the R console.
- In the plots and files window, click on the packages and help tabs to see what they offer.
- See what types of new files can be made in R studio by clicking the top left icon- open a new R script.

2. Now open the file called 'Exercises_for_R_Lectures.Rmd'. This file will serve as your digital notebook for parts of the workshop and contains the other exercises.

Exercise 1.2: Intro to `R Markdown` Files

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.

1. 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
```

2. Each code chunk begins and ends in the same way- with a fence (three dashes). You can further specify what you want to show up in your final document using the `echo` and `eval` commands in the opening line. Insert a few code chunks below using the `insert` tab at the top of this window. Then, change the `echo` and `eval` arguments to `TRUE` or `FALSE` and see how different combinations of these arguments change the output when you knit. I have done the first one for you. Notice too that each `R` code chunk requires a unique title argument (here 'cars variant 1'), or the Rmd will not knit.

```
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
```

```
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
```

3. What do you think `echo` and `eval` do, based on your manipulations?
    - `echo` repeats the code in the final document, `eval` tells R to actually run the code and print the results.

- What are the defaults for `echo` and `eval`, based on your manipulations?
    - defaults are `echo=FALSE` and `eval=TRUE`

Excercise 1.3: RMarkdown advanced

Getting more familiar with RMarkdown

- If you want to beautify your output, it always starts here.
- There are many options, and a few are laid out below.
- The `knitr` package has lots of options explained
    - here and
    - here in detail.
- Part of configuring your script is loading the correct packages. Always load all packages together at the top in your `config` chunk. That way future users will know exactly what they need to install.

```
library(scales)
library(knitr)
opts_chunk$set(background = "gray80", tidy = FALSE, cache = FALSE,
    comment = "", dpi = 72, fig.path = "RMDfigs/", fig.width = 4,
    fig.height = 4)
```

2. Generate fake data

- The `x` value is just numbers 1-100 for an x axis value. This might be time or distance, etc.

- For the response variable, generate a random normal distribution with the `rnorm` function, and then add a trend with the `seq` function.
- Then we'll add some fake treatments with `letters`.

```
# setwd('~/Desktop')

x <- 1:100
y <- rnorm(100, sd=3) + seq(10.05, 20, 10/100)
z <- factor(rep(letters[1:5], each=20))
dat <- data.frame(x, y, z)
```

3. Tables in `knitr`

- This is an ugly way to preview data or display tables.

```
head(dat)
```

```
  x         y z
1 1 16.347160 a
2 2 13.750655 a
3 3  8.697976 a
4 4  8.616812 a
5 5 12.045210 a
6 6 12.002517 a
```

- The `knitr` package has a simple built-in function for dealing with tables. This works well in either html or pdf output.

```
kable(head(dat))
```

x	y	z
1	16.347160	a
2	13.750655	a
3	8.697976	a
4	8.616812	a
5	12.045210	a
6	12.002517	a

```
# remove a few samples that we don't want to analyze.
# dat <- dat[-c(3, 4, 5, 12), ]
```

4. R commands embedded in prose

- One of the best features in `knitr` and RMarkdown generally, is the ability to embed real R commands in sentences, so that you can report actual values instead of constantly copying and pasting when results change a little bit.
- This table has 100 rows and 3 columns. The 'x' variable starts at 1 and ends at 100.
- 0.0132895

5. Formatting text in RMarkdown

- Create a formatted list with 2 levels and 2 sub levels; make one of the sub levels italic and the main levels bold
- Create a quote from one of your new workshop friends

6. LaTeX in RMarkdown

- Insert three symbols of your choice
- Recreate an equation from the slides and insert it in-line and fenced

---

Exercise 1.4: Basic Mathematics in R

Insert a code chunk below and complete the following tasks:

1. Add and subtract

```
2+2
```

```
[1] 4
```

```
2-1
```

```
[1] 1
```

2. Multiply and divide

```
2*2
```

```
[1] 4
```

```
2/1
```

```
[1] 2
```

3. Raise a number to a power using the ^ symbol

```
2^2
```

```
[1] 4
```

4. Create a more complex equation involving all of these operations to convince yourself that `R` follows the normal priority of mathematical evaluation

```
(4-1)^2*3
```

```
[1] 27
```

Exercise 1.5: Assigning Variables and Arithmetic Functions in `R`

Insert a code chunk below and complete the following tasks:

1. Assign three variables using basic mathematical operations

```
x<-2*3
y<-5
z<-5-1
```

2. Take the log of your three variables

```
log(x)
```

```
[1] 1.791759
```

```
log(y)
```

```
[1] 1.609438
```

```
log(z)
```

```
[1] 1.386294
```

3. Use the print function to display your most complex variable

```
print(z)
```

```
[1] 4
```

4. Use the `concatenate` function to print a sentence

```
q<-c("I","love","stats")
print(q)
```

```
[1] "I"     "love"  "stats"
```

Exercise 1.6: Vectors and Factors

Insert a code chunk below and complete the following tasks:

1. Create a numeric vector using the `c` function

```
n<-c(2,4,6,8,10)
```

2. Create a multi-level character factor using the `c` function

```
vec1<-c("I", "am", "great","at","R")
fac1<-as.factor(vec1)
print(fac1)
```

```
[1] I     am    great at    R
Levels: am at great I R
```

3. Use `str` and `class` to evaluate your variables

```
class(fac1)
```

```
[1] "factor"
```

```
str(fac1)
```

```
 Factor w/ 5 levels "am","at","great",..: 4 1 3 2 5
```

Exercise 1.7: Basic Statistics

Insert a code chunk below and complete the following tasks:

1. Create a vector and calculate the `mean`, `sd`, `sum`, `length`, and `var`

```
n<-c(2,4,6,8,10)
mean(n)
```

```
[1] 6
```

```
sd(n)
```

```
[1] 3.162278
```

```
sum(n)
```

```
[1] 30
```

```
length(n)
```

```
[1] 5
```

```
var(n)
```

```
[1] 10
```

2. Use the `log` and `sqrt` functions on your vector

```
n<-c(2,4,6,8,10)
log(n)
```

```
[1] 0.6931472 1.3862944 1.7917595 2.0794415 2.3025851
```

```
sqrt(n)
```

```
[1] 1.414214 2.000000 2.449490 2.828427 3.162278
```

3. What happens when you try to apply these functions to a factor?
   - Error! log not meaningful for factors
4. Type the first couple letters of a function within your R code chunk, then hit tab- what happens?

- What if you press tab with you cursor inside the function parentheses?
  - Tab complete is the BEST! Use it to help you with functions and the arguments for each function!

Exercise 1.8: Creating Larger Vectors and Random Sampling

Complete the following tasks in the codechunk below: - Note: If you ever want someone else to be able to perfectly reproduce your results, always set the random seed at the top. Any number will do. Note that it never hurts to set the seed, *but* robust results should always stand up to random number generators.

1. Create a vector with 100 elements using the `seq` function and calculate two basic statistics on your vector
2. Create a variable and `sample` it with equal probability

- Can you figure out what the arguments in the parentheses mean?
- Try varying the arguments to see what happens.

3. Create a normally distributed variable of 10000 elements using the `rnorm` function then `sample` that distribution with and without replacement
4. Use `hist` to plot your normally distributed variable

```
set.seed(1415)
#1
s<-seq(from=1, to=1000,length.out = 100)
mean(s)
```

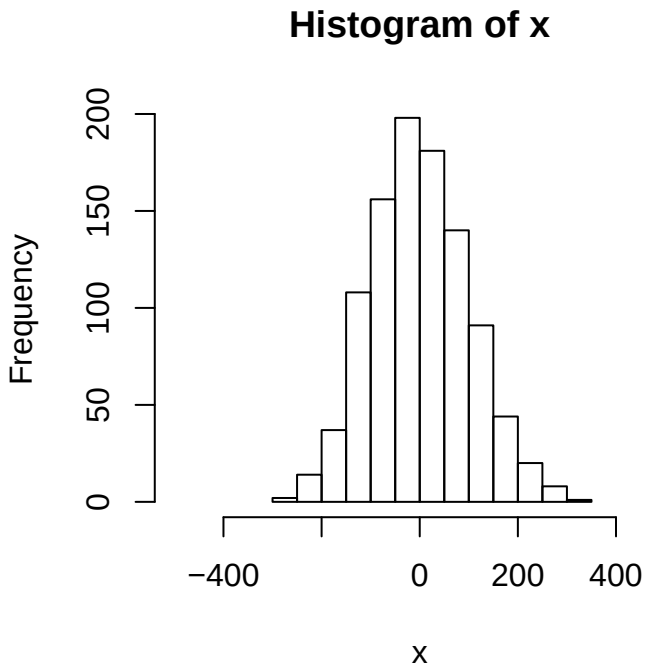
```
[1] 500.5
```

```
sqrt(s)
```

```
  [1]  1.000000  3.330302  4.602371  5.592202  6.431457  7.173182  7.845091
  [8]  8.463827  9.040314  9.582180 10.095003 10.583005 11.049476 11.497035
 [15] 11.927813 12.343567 12.745766 13.135656 13.514302 13.882625 14.241425
 [22] 14.591405 14.933185 15.267315 15.594288 15.914545 16.228482 16.536461
 [29] 16.838808 17.135821 17.427773 17.714914 17.997475 18.275667 18.549688
 [36] 18.819719 19.085930 19.348479 19.607512 19.863168 20.115575 20.364854
 [43] 20.611118 20.854474 21.095023 21.332860 21.568074 21.800751 22.030970
 [50] 22.258808 22.484338 22.707628 22.928743 23.147747 23.364698 23.579652
 [57] 23.792665 24.003788 24.213069 24.420558 24.626298 24.830333 25.032706
 [64] 25.233455 25.432620 25.630238 25.826343 26.020971 26.214153 26.405922
 [71] 26.596309 26.785342 26.973051 27.159462 27.344602 27.528498 27.711173
 [78] 27.892651 28.072957 28.252112 28.430138 28.607056 28.782886 28.957649
 [85] 29.131364 29.304049 29.475722 29.646401 29.816103 29.984845 30.152642
 [92] 30.319511 30.485466 30.650523 30.814695 30.977998 31.140444 31.302048
 [99] 31.462821 31.622777
```

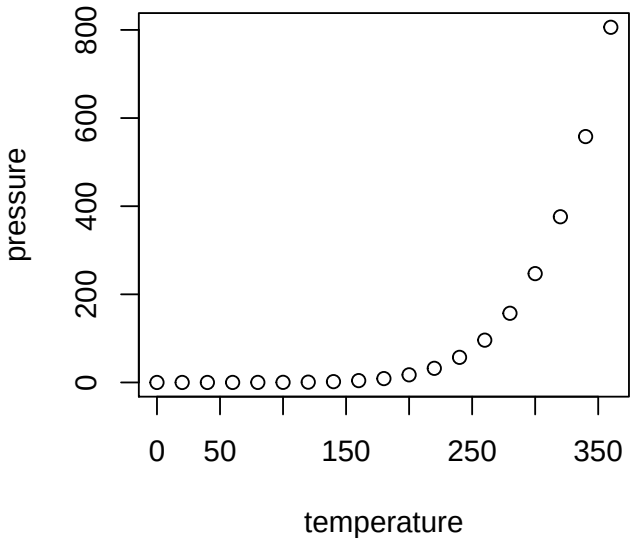
```
#2
t<-sample(x=s,size=50,replace = TRUE)
```

```r
#3
u <- rnorm (n = 10000, mean = 0, sd = 10)
v <- sample (u, 10000, replace = T)
w <- sample (u, 10000, replace = F)
#4
x <-rnorm(1000, 0, 100)
hist(x, xlim = c(-500,500))
```



Including Plots

You can also embed plots in your pdf document (`knit` to view), for example:



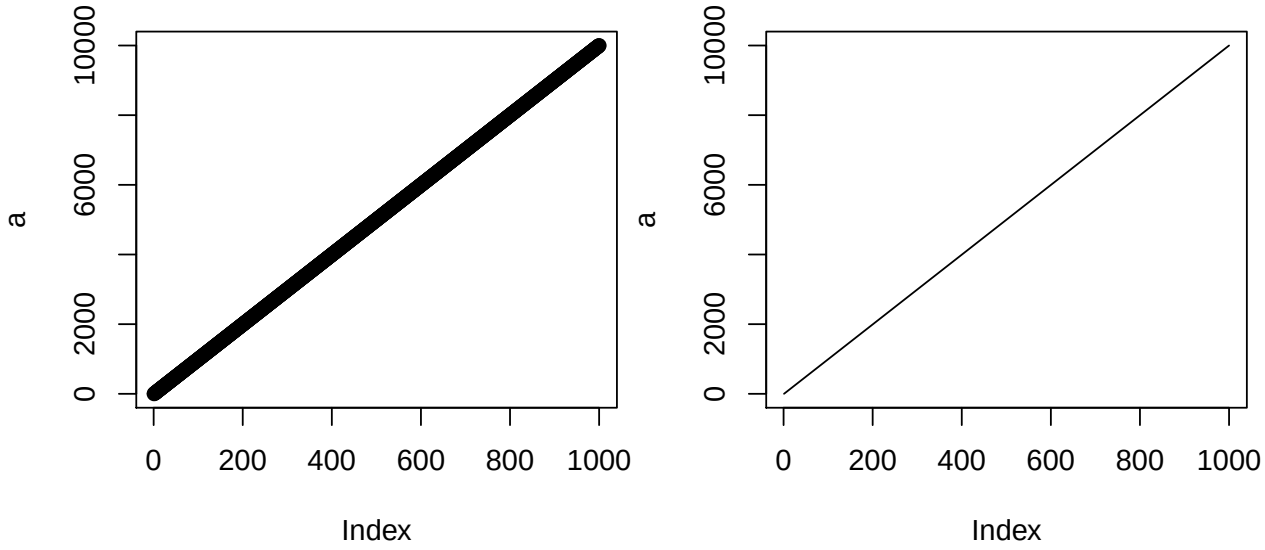
- Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

- Note that you can also alter the size of the plot in the chunk header (`{}`) section.

Exercise 1.9: Basic Visualization

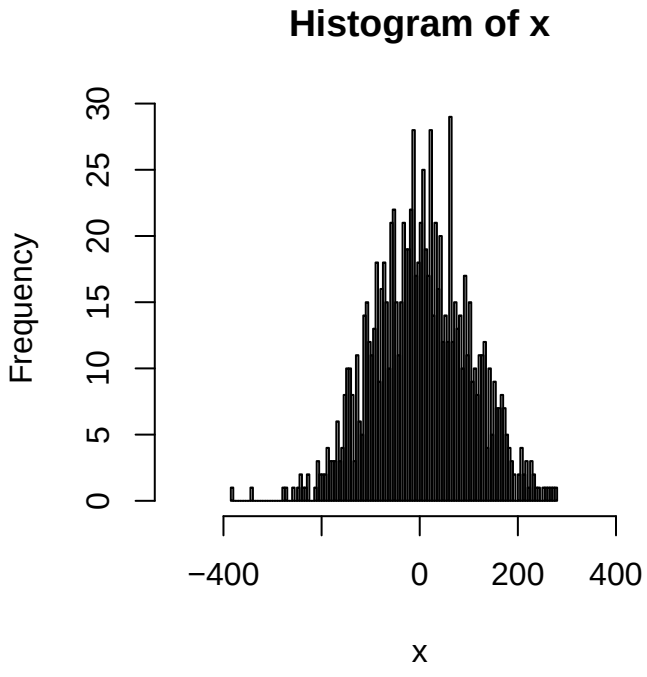
Insert a code chunk below and complete the following tasks, make sure to label all plot axes and have fun with colors!

1. Create a variable using `seq` and make two different plots by changing the `type` argument

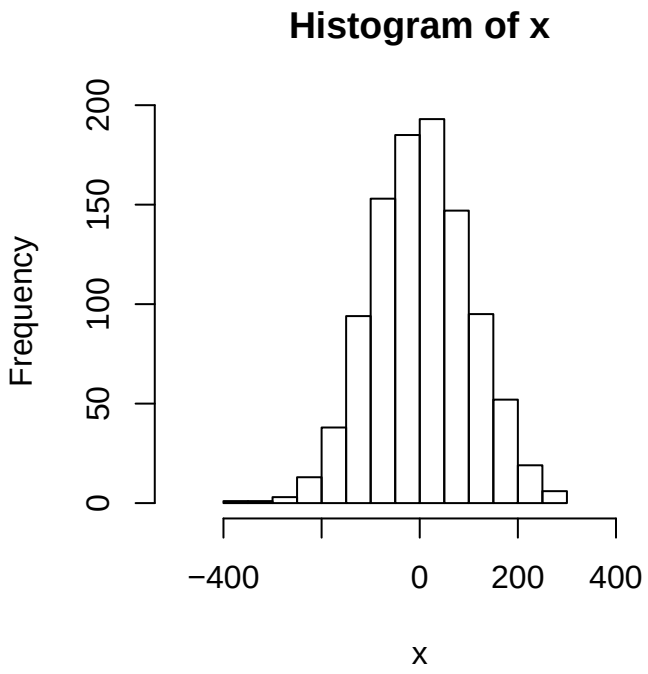


2. Create a normally distributed variable using `rnorm` and make two different plots using `hist` by varying the `breaks` argument (what does `breaks` appear to do?)

```
x <-rnorm(1000, 0, 100)
hist(x, xlim = c(-500,500), breaks=100)
```

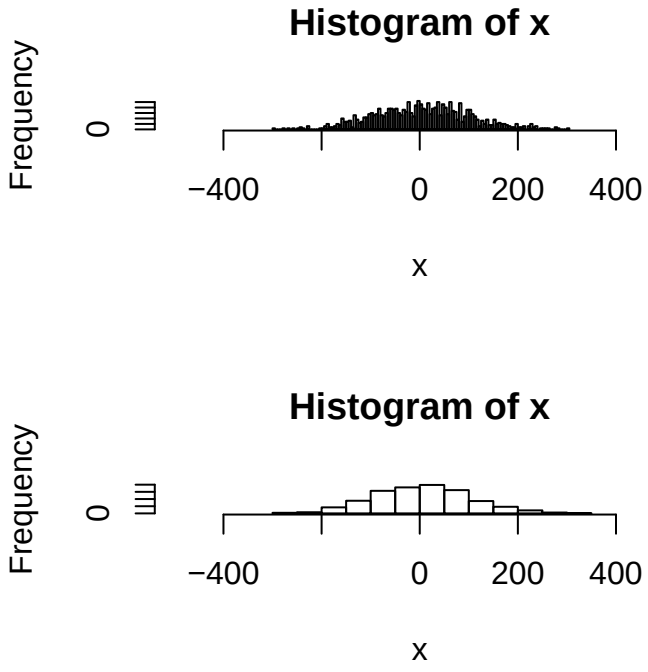


```
hist(x, xlim = c(-500,500), breaks=10)
```



3. Modify your `par()` arguments to create a composite figure of the above graphs.

```
par(mfrow=c(2,1))
x <-rnorm(1000, 0, 100)
hist(x, xlim = c(-500,500), breaks=100)
hist(x, xlim = c(-500,500), breaks=10)
```



Exercise 1.10: Creating a Data Frame and Evaluating Class

Insert a code chunk below and complete the following tasks:

1. Recreate the dataframe from the slides by creating each vector then using `data.frame`

```
habitat <- factor(c("mixed", "wet", "wet", "wet", "dry", "dry", "dry","mixed"))
temp <- c(3.4, 3.4, 8.4, 3, 5.6, 8.1, 8.3, 4.5)
elevation <- c(0, 9.2, 3.8, 5, 5.6, 4.1, 7.1, 5.3)
mydata <- data.frame(habitat, temp, elevation)
```

2. Assign rownames to your dataframe using `rownames` and `c`

```
row.names(mydata) <- c("Reedy Lake", "Pearcadale", "Warneet", "Cranbourne", "Lysterfield", "Red Hill",
```

3. Get class assignments for your whole dataframe using `str`

```
str(mydata)
```

```
'data.frame':   8 obs. of  3 variables:
 $ habitat  : Factor w/ 3 levels "dry","mixed",..: 2 3 3 3 1 1 1 2
 $ temp     : num  3.4 3.4 8.4 3 5.6 8.1 8.3 4.5
 $ elevation: num  0 9.2 3.8 5 5.6 4.1 7.1 5.3
```

4. Calculate the `mean` of each numeric variable

```
mean(mydata$temp)
```

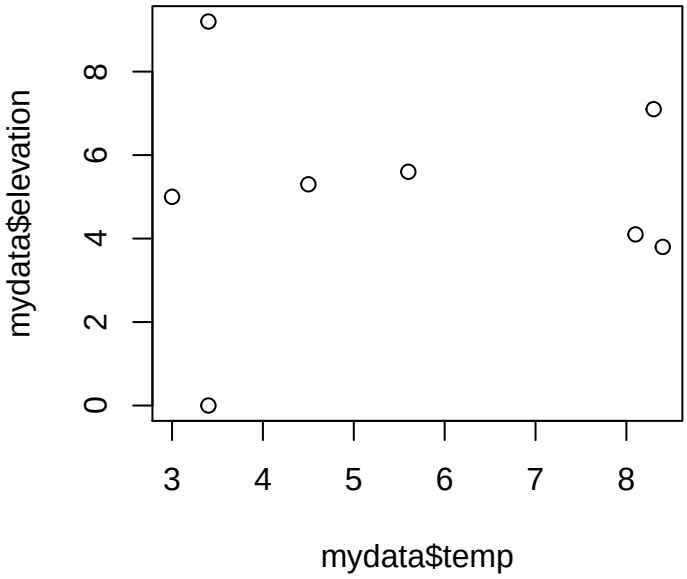
```
[1] 5.5875
```

```
mean(mydata$elevation)
```

```
[1] 5.0125
```

5. Make a descriptive plot of your choosing

```
plot(mydata$temp,mydata$elevation)
```



6. What happens when you use the functions `head` and `tail` on your dataframe?

```
head(mydata)
```

```
            habitat temp elevation
Reedy Lake    mixed  3.4       0.0
Pearcadale      wet  3.4       9.2
Warneet         wet  8.4       3.8
Cranbourne      wet  3.0       5.0
Lysterfield     dry  5.6       5.6
Red Hill        dry  8.1       4.1
```

```
tail(mydata)
```

```
            habitat temp elevation
Warneet         wet  8.4       3.8
Cranbourne      wet  3.0       5.0
Lysterfield     dry  5.6       5.6
Red Hill        dry  8.1       4.1
Devilbend       dry  8.3       7.1
Olinda        mixed  4.5       5.3
```

Exercise 1.11: Datasets and Indexing

By opening this .Rmd file, you have automatically set your working directory to the folder containing it. Now, you can access data from this directory or a sub-directory in this folder. You can do this by including that part of the path in the `read.csv` function. Insert a code chunk below and complete the following tasks:

1. Save the file we created together in a sub-directory of your current working directory
2. Use `read.csv` to read your file in

```
YourFile <- read.csv("yourfile_key.csv", header=T, row.names=1,sep=',')
```

3. Use `str` and `head` to view your data structure

```
str(YourFile)
head(YourFile)
```

4. Use the `$` and `[ ]` operators to select out different parts of the dataframe.

```
YourFile$temp
```

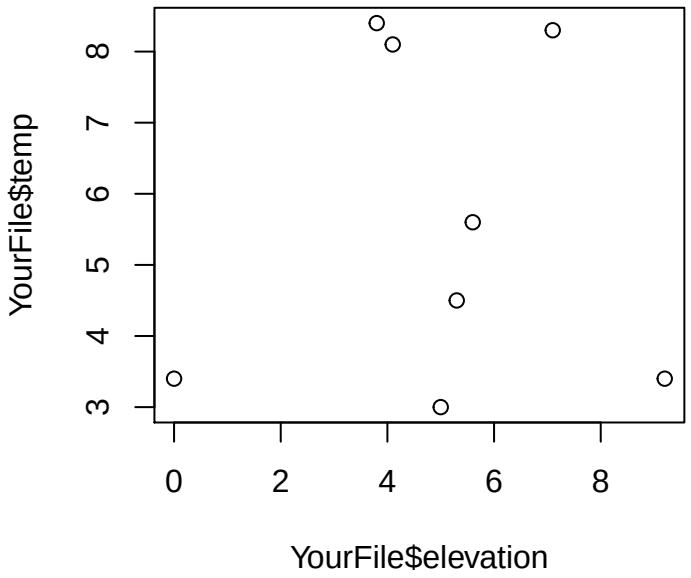
```
[1] 3.4 3.4 8.4 3.0 5.6 8.1 8.3 4.5
```

```
YourFile[3]
```

```
  elevation
1       0.0
2       9.2
3       3.8
4       5.0
5       5.6
6       4.1
7       7.1
8       5.3
```

5. Plot temperature over elevation using `$`.

```
plot(YourFile$elevation,YourFile$temp)
```



6. Use the `tapply` function to calculate the `mean` and `var` of temp by habitat type and temp by elevation.

```
tapply(YourFile$temp, YourFile$habitat, mean)
```

```
     dry    mixed      wet
7.333333 3.950000 4.933333
```

```
tapply(YourFile$temp, YourFile$habitat, var)
```

```
     dry    mixed      wet
2.263333 0.605000 9.053333
```

7. Export your data frame with a different file name

```
write.csv(YourFile, "yourfile_2.csv", quote=F, row.names=T, sep=",")
```
