Lab 2: Indexing and Iteration

Name: Rufus Petrie

This week's agenda: basic indexing, with a focus on matrices; some more basic plotting; vectorization; using `for()` loops.

Back to some R basics

- **1a.** Let's start easy by working through some R basics, to continue to brush up on them. Define a variable `x.vec` to contain the integers 1 through 100. Check that it has length 100. Report the data type being stored in `x.vec`. Add up the numbers in `x.vec`, by calling a built-in R function. How many arithmetic operations did this take? **Challenge**: show how Gauss would have done this same calculation as a 7 year old, using just 3 arithmetic operations.

```r
x.vec <- 1:100
length(x.vec)
```

```
## [1] 100
```

```r
sum(x.vec)
```

```
## [1] 5050
```

Gauss would have solved this by using:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{100(101)}{2} = 5050$$

- **1b.** Convert `x.vec` into a matrix with 20 rows and 5 columns, and store this as `x.mat`. Here `x.mat` should be filled out in the default order (column major order). Check the dimensions of `x.mat`, and the data type as well. Compute the sums of each of the 5 columns of `x.mat`, by calling a built-in R function. Check (using a comparison operator) that the sum of column sums of `x.mat` equals the sum of `x.vec`.

```r
x.mat <- matrix(x.vec, ncol=5)
dim(x.mat)
```

```
## [1] 20  5
```

```r
typeof(x.mat)
```

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

```r
colMeans(x.mat)
```

```
## [1] 10.5 30.5 50.5 70.5 90.5
```

```r
sum(x.vec) == sum(colSums(x.mat))
```

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

- **1c.** Extract and display rows 1, 5, and 17 of `x.mat`, with a single line of code. Answer the following questions, each with a single line of code: how many elements in row 2 of `x.mat` are larger than 40?

How many elements in column 3 are in between 45 and 50? How many elements in column 5 are odd? Hint: take advantage of the `sum()` function applied to Boolean vectors.

```
x.mat[c(1,5,17),]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1   21   41   61   81
## [2,]    5   25   45   65   85
## [3,]   17   37   57   77   97
```

```
sum(x.mat[2,]>40)
```

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

```
sum(x.mat[,3]>45 & x.mat[,3]<50)
```

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

```
sum(x.mat[,5]%%2 != 0)
```

```
## [1] 10
```

- **1d.** Using Boolean indexing, modify `x.vec` so that every even number in this vector is incremented by 10, and every odd number is left alone. This should require just a single line of code. Print out the result to the console. **Challenge**: show that `ifelse()` can be used to do the same thing, again using just a single line of code.

```
x.vec <- 1:100
x.vec[x.vec%%2 == 0]  <- x.vec[x.vec%%2 == 0] + 10
x.vec
```

```
##   [1]   1  12   3  14   5  16   7  18   9  20  11  22  13  24  15  26  17  28
##  [19]  19  30  21  32  23  34  25  36  27  38  29  40  31  42  33  44  35  46
##  [37]  37  48  39  50  41  52  43  54  45  56  47  58  49  60  51  62  53  64
##  [55]  55  66  57  68  59  70  61  72  63  74  65  76  67  78  69  80  71  82
##  [73]  73  84  75  86  77  88  79  90  81  92  83  94  85  96  87  98  89 100
##  [91]  91 102  93 104  95 106  97 108  99 110
```

```
x.vec <- 1:100
x.vec <- ifelse(x.vec%%2 == 0, x.vec +10, x.vec)
x.vec
```

```
##   [1]   1  12   3  14   5  16   7  18   9  20  11  22  13  24  15  26  17  28
##  [19]  19  30  21  32  23  34  25  36  27  38  29  40  31  42  33  44  35  46
##  [37]  37  48  39  50  41  52  43  54  45  56  47  58  49  60  51  62  53  64
##  [55]  55  66  57  68  59  70  61  72  63  74  65  76  67  78  69  80  71  82
##  [73]  73  84  75  86  77  88  79  90  81  92  83  94  85  96  87  98  89 100
##  [91]  91 102  93 104  95 106  97 108  99 110
```

- **1e.** Consider the list `x.list` created below. Complete the following tasks, each with a single line of code: extract all but the second element of `x.list`—seeking here a list as the final answer. Extract the first and third elements of `x.list`, then extract the second element of the resulting list—seeking here a vector as the final answer. Extract the second element of `x.list` as a vector, and then extract the first 10 elements of this vector—seeking here a vector as the final answer. Note: pay close attention to what is asked and use either single brackets `[ ]` or double brackets `[[ ]]` as appropriate.

```
x.list = list(rnorm(6), letters, sample(c(TRUE,FALSE),size=4,replace=TRUE))
x.list[-2]
```

```
## [[1]]
## [1]  0.62574164 -0.21866568 -0.04284963  0.43079570  0.22386291  0.41265474
```

```
##
## [[2]]
## [1] FALSE FALSE FALSE  TRUE
```

```
x.list[-2][[2]]
```

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

```
x.list[[2]][1:10]
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

Prostate cancer data set

We're going to look at a data set on 97 men who have prostate cancer (from the book The Elements of Statistical Learning). There are 9 variables measured on these 97 men:

1. `lpsa`: log PSA score
2. `lcavol`: log cancer volume
3. `lweight`: log prostate weight
4. `age`: age of patient
5. `lbph`: log of the amount of benign prostatic hyperplasia
6. `svi`: seminal vesicle invasion
7. `lcp`: log of capsular penetration
8. `gleason`: Gleason score
9. `pgg45`: percent of Gleason scores 4 or 5

To load this prostate cancer data set into your R session, and store it as a matrix `pros.dat`:

```
pros.dat =
  as.matrix(read.table("http://www.stat.cmu.edu/~ryantibs/statcomp/data/pros.dat"))
```

Basic indexing and calculations

- **2a.** What are the dimensions of `pros.dat` (i.e., how many rows and how many columns)? Using integer indexing, print the first 6 rows and all columns; again using integer indexing, print the last 6 rows and all columns.

```
dim(pros.dat)
```

```
## [1] 97  9
```

```
pros.dat[1:6,]
```

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
## 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0  0.7654678
```

```
pros.dat[92:97,]
```

```
##      lcavol  lweight age       lbph svi       lcp gleason pgg45     lpsa
## 92 2.532903 3.677566  61  1.3480732   1 -1.386294       7    15 4.129551
## 93 2.830268 3.876396  68 -1.3862944   1  1.321756       7    60 4.385147
## 94 3.821004 3.896909  44 -1.3862944   1  2.169054       7    40 4.684443
## 95 2.907447 3.396185  52 -1.3862944   1  2.463853       7    10 5.143124
## 96 2.882564 3.773910  68  1.5581446   1  1.558145       7    80 5.477509
## 97 3.471966 3.974998  68  0.4382549   1  2.904165       7    20 5.582932
```

- **2b.** Using the built-in R functions `head()` and `tail()` (i.e., do *not* use integer indexing), print the first 6 rows and all columns, and also the last 6 rows and all columns.

```
head(pros.dat, 6)
```

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
## 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0  0.7654678
```

```
tail(pros.dat, 6)
```

```
##      lcavol  lweight age       lbph svi       lcp gleason pgg45     lpsa
## 92 2.532903 3.677566  61  1.3480732   1 -1.386294       7    15 4.129551
## 93 2.830268 3.876396  68 -1.3862944   1  1.321756       7    60 4.385147
## 94 3.821004 3.896909  44 -1.3862944   1  2.169054       7    40 4.684443
## 95 2.907447 3.396185  52 -1.3862944   1  2.463853       7    10 5.143124
## 96 2.882564 3.773910  68  1.5581446   1  1.558145       7    80 5.477509
## 97 3.471966 3.974998  68  0.4382549   1  2.904165       7    20 5.582932
```

- **2c.** Does the matrix `pros.dat` have names assigned to its rows and columns, and if so, what are they? Use `rownames()` and `colnames()` to find out. Note: these would have been automatically created by the `read.table()` function that we used above to read the data file into our R session. To see where `read.table()` would have gotten these names from, open up the data file: http://www.stat.cmu.edu/~ryantibs/statcomp/data/pros.dat in your web browser. Only the column names here are actually informative.

```
rownames(pros.dat)
```

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

```
colnames(pros.dat)
```

```
## [1] "lcavol"  "lweight" "age"     "lbph"    "svi"     "lcp"     "gleason"
## [8] "pgg45"   "lpsa"
```

- **2d.** Using named indexing, pull out the two columns of `pros.dat` that measure the log cancer volume and the log cancer weight, and store the result as a matrix `pros.dat.sub`. (Recall the explanation of variables at the top of this lab.) Check that its dimensions make sense to you, and that its first 6 rows are what you'd expect. Did R automatically assign column names to `pros.dat.sub`?

```
pros.dat.sub <- pros.dat[,c("lcavol", "lweight")]
dim(pros.dat.sub)
```

```
## [1] 97  2
```

```
head(pros.dat.sub, 6)
```

```
##       lcavol  lweight
## 1 -0.5798185 2.769459
## 2 -0.9942523 3.319626
## 3 -0.5108256 2.691243
## 4 -1.2039728 3.282789
## 5  0.7514161 3.432373
## 6 -1.0498221 3.228826
```

```
colnames(pros.dat.sub)
```

```
## [1] "lcavol"  "lweight"
```

- **2e.** Using the log cancer weights and log cancer volumes, calculate the log cancer density for the 97 men in the data set (note: by density here we mean weight divided by volume). There are in fact two different ways to do this; the first uses three function calls and one arithmetic operation; the second just uses one arithmetic operation. Note: in either case, you should be able to perform this computation for all 97 men *with a single line of code*, taking advantage of R's ability to vectorize. Write code to do it both ways, and show that both ways lead to the same answer, using `all.equal()`.

```
ldens <- pros.dat.sub[,'lweight'] / pros.dat.sub[,'lcavol']
```

I'm not sure what we're looking for here. It sounds like I'm supposed to do exp() of the two variables and take the log of the quotient, but that doesn't equal the simple quotient in this scenario.

- **2f.** Append the log cancer density to the columns of `pros.dat`, using `cbind()`. The new `pros.dat` matrix should now have 10 columns. Set the last column name to be `ldens`. Print its first 6 rows, to check that you've done all this right.

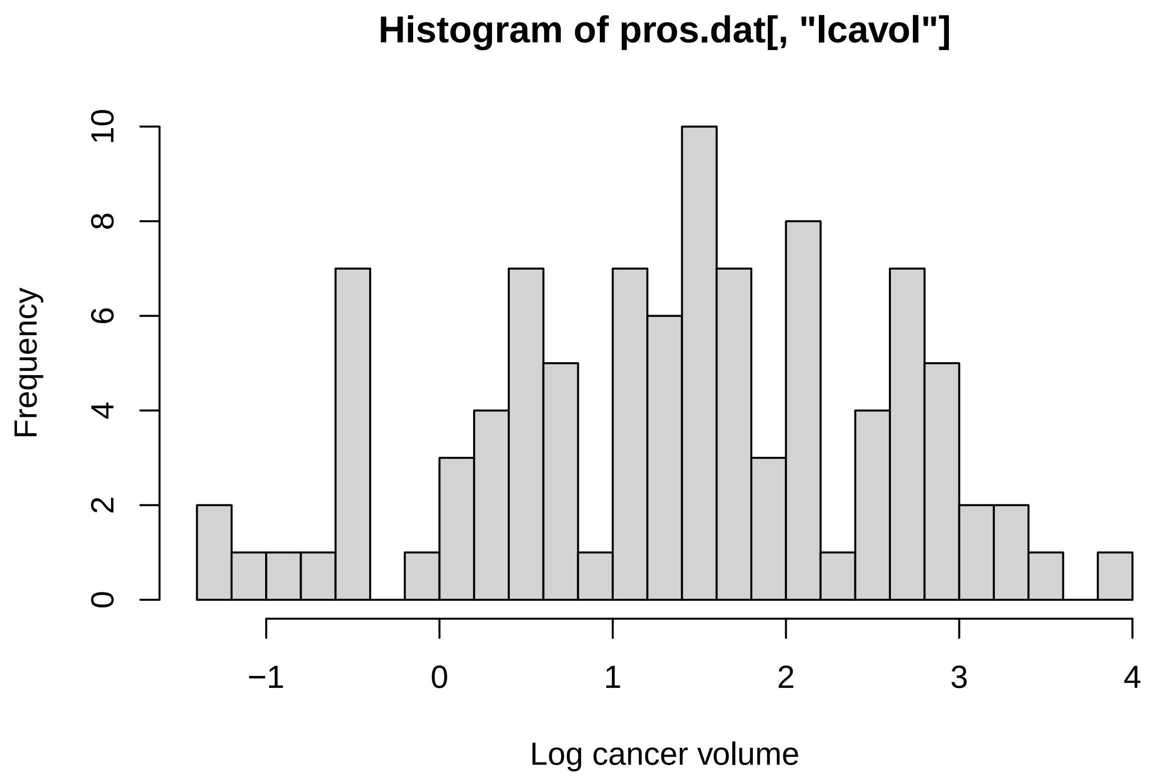
```
pros.dat <- cbind(pros.dat, ldens)
head(pros.dat, 6)
```

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
## 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0  0.7654678
##        ldens
## 1 -4.776424
## 2 -3.338817
## 3 -5.268418
## 4 -2.726631
## 5  4.567873
## 6 -3.075593
```

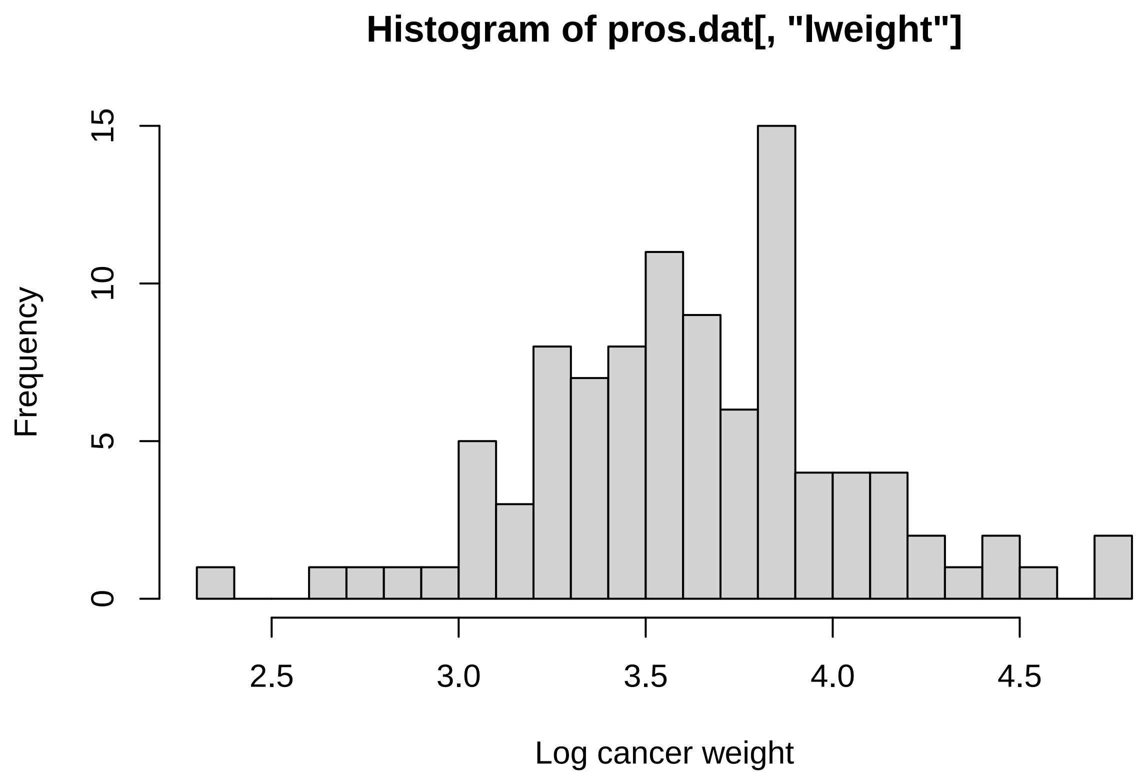
Exploratory data analysis with plots

- **3a.** Using `hist()`, produce a histogram of the log cancer volume measurements of the 97 men in the data set; also produce a histogram of the log cancer weight. In each case, use `breaks=20` as an arugment to `hist()`. Comment just briefly on the distributions you see. Then, using `plot()`, produce a scatterplot of the log cancer volume (y-axis) versus the log cancer weight (x-axis). Do you see any kind of relationship? Would you expect to? **Challenge**: how would you measure the strength of this relationship formally? Note that there is certainly more than one way to do so. We'll talk about statistical modeling tools later in the course.

```
hist(pros.dat[,'lcavol'], breaks=20, xlab="Log cancer volume")
```

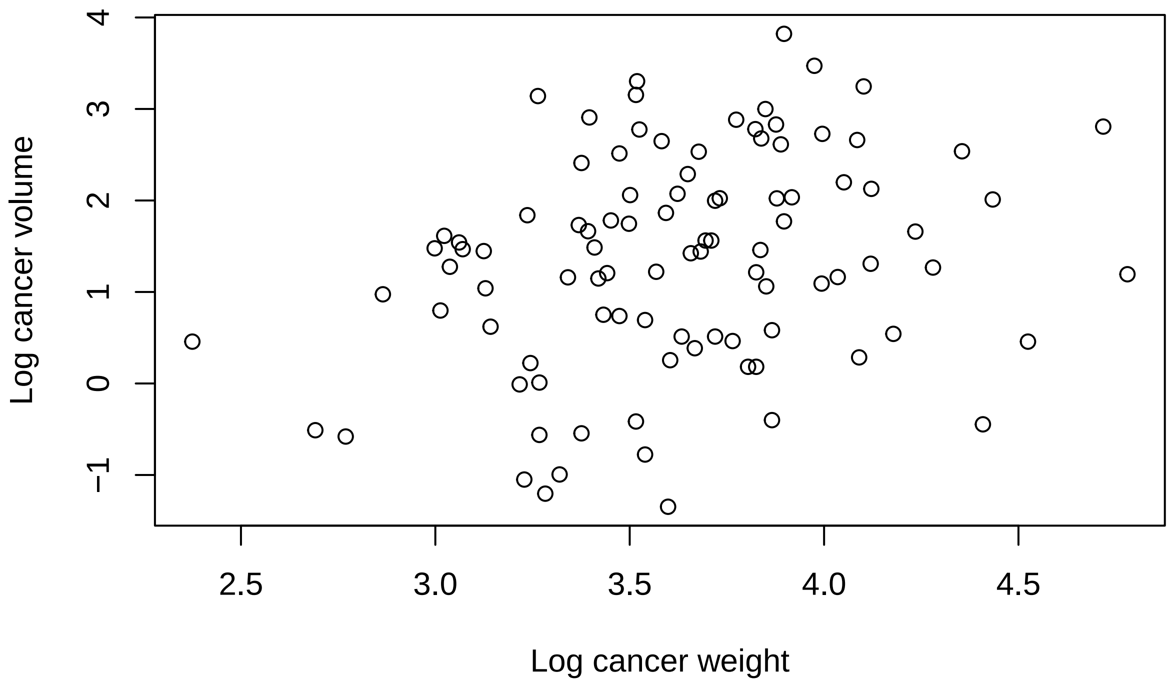


```
hist(pros.dat[,'lweight'], breaks=20, xlab="Log cancer weight")
```



Both of these plots roughly resemble normal distributions, but the log cancer weight distribution is fairly flat.

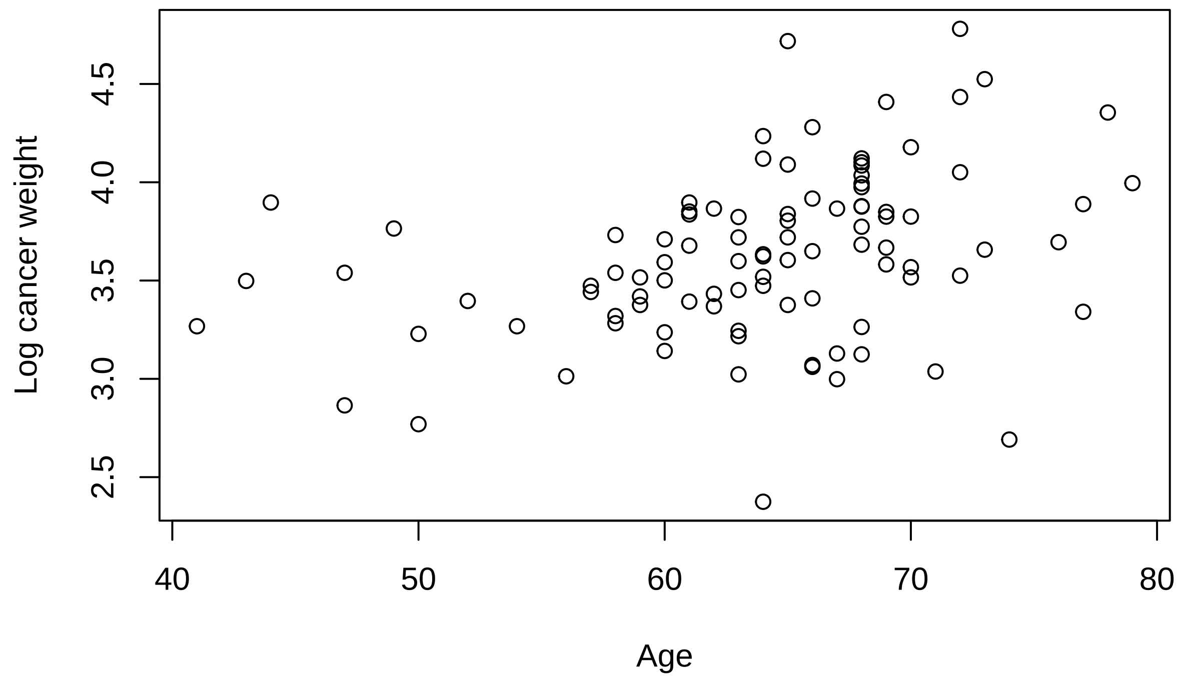
```
plot(pros.dat[,'lweight'], pros.dat[,'lcavol'], xlab="Log cancer weight",
     ylab="Log cancer volume")
```



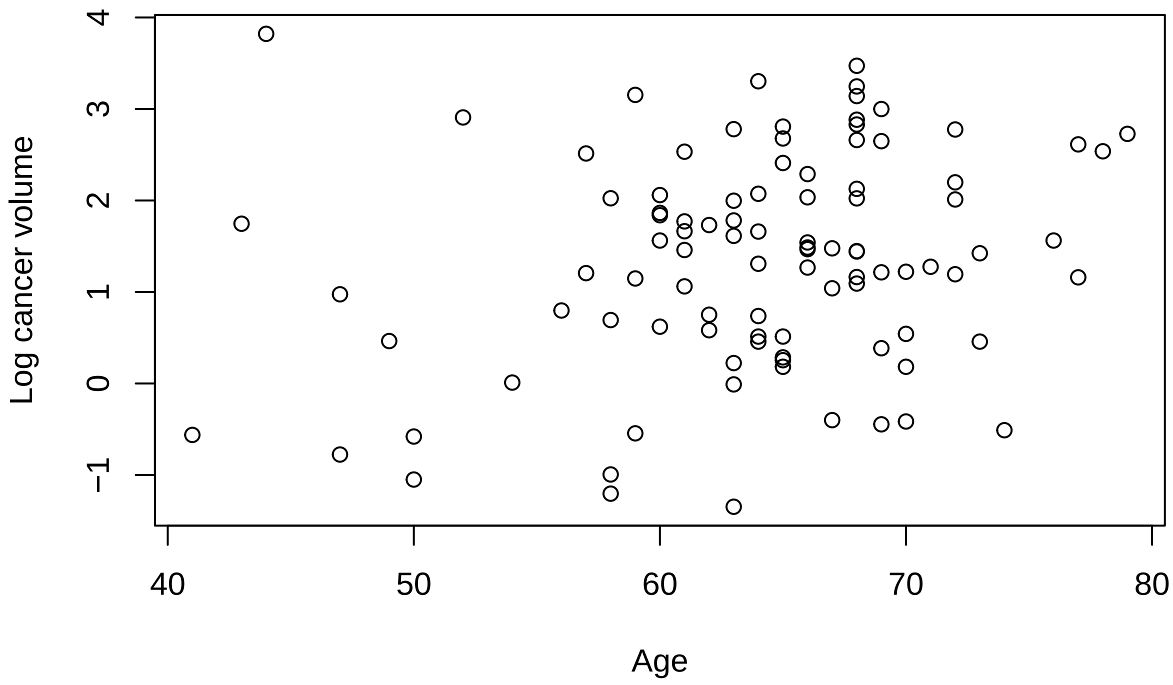
There appears to be a positive correlation between log cancer volume and weight. To measure this relationship formally, we might want to make a regression line.

- **3b.** Produce scatterplots of log cancer weight versus age, and log cancer volume versus age. Do you see relationships here between the age of a patient and the volume/weight of his cancer?

```
plot(pros.dat[,'age'], pros.dat[,'lweight'], xlab="Age", ylab="Log cancer weight")
```



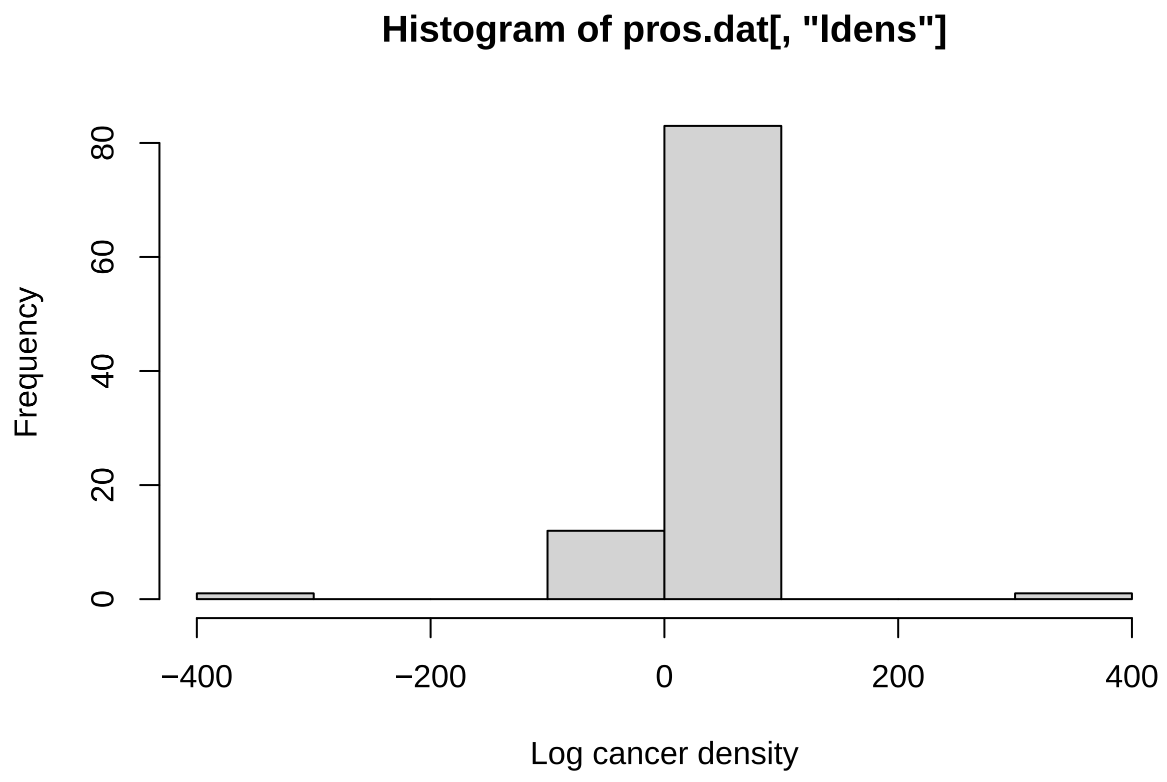
```r
plot(pros.dat[,'age'], pros.dat[,'lcavol'], xlab="Age", ylab="Log cancer volume")
```



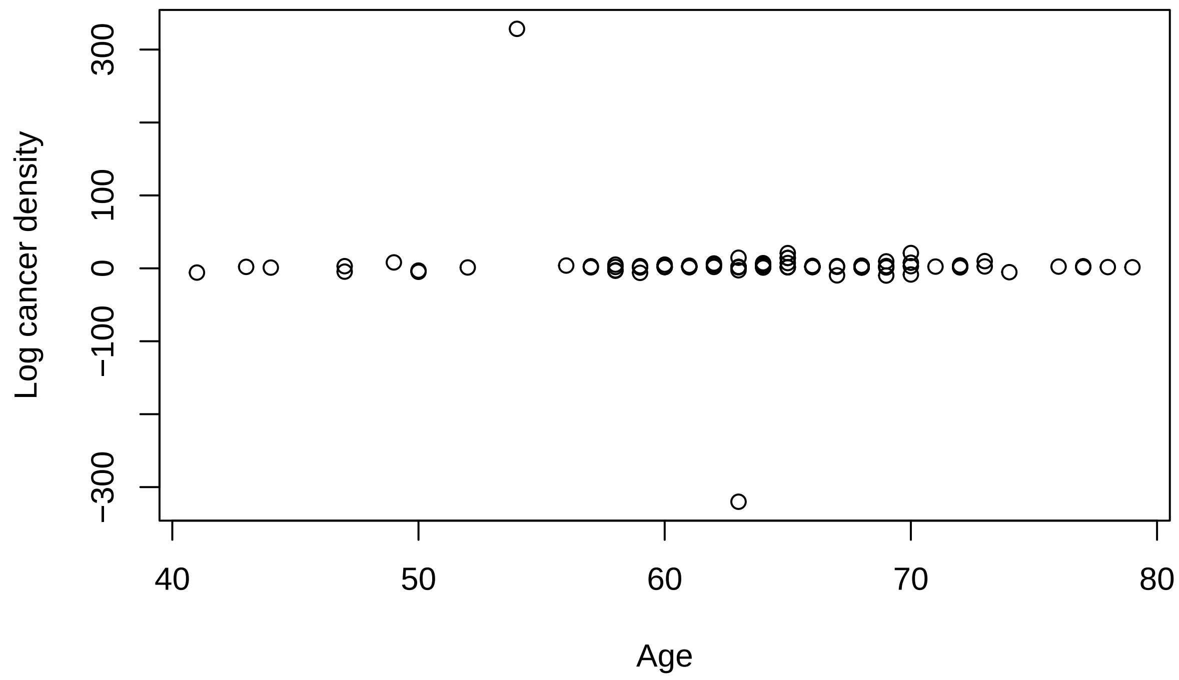
Both the log cancer weight and log cancer volume appear to have a positive correlation with the age of the patient.

- **3c.** Produce a histogram of the log cancer density, and a scatterplot of the log cancer density versus age. Comment on any similarities/differences you see between these plots, and the corresponding ones you produced above for log cancer volume/weight.

```
hist(pros.dat[,'ldens'], xlab="Log cancer density")
```



```r
plot(pros.dat[,'age'], pros.dat[,'ldens'], xlab="Age", ylab="Log cancer density")
```



Unlike log cancer volume and weight, log cancer density does not have much variation other than two noticeable outliers, as the histogram has most observations appearing in one bar instead of being a bell curve. Similarly, it appears that the log density is uncorrelated with age, as it does not vary as age increases.

- **3d.** Delete the last column, corresponding to the log cancer density, from the `pros.dat` matrix, using negative integer indexing.

```
pros.dat <- pros.dat[,-10]
```

A bit of Boolean indexing never hurt anyone

- **4a.** The `svi` variable in the `pros.dat` matrix is binary: 1 if the patient had a condition called "seminal vesicle invasion" or SVI, and 0 otherwise. SVI (which means, roughly speaking, that the cancer invaded into the muscular wall of the seminal vesicle) is bad: if it occurs, then it is believed the prognosis for the patient is poorer, and even once/if recovered, the patient is more likely to have prostate cancer return in the future. Compute a Boolean vector called `has.svi`, of length 97, that has a `TRUE` element if a row (patient) in `pros.dat` has SVI, and `FALSE` otherwise. Then using `sum()`, figure out how many patients have SVI.

```
has.svi <- pros.dat[,"svi"] == 1
sum(has.svi)
```

```
## [1] 21
```

- **4b.** Extract the rows of `pros.dat` that correspond to patients with SVI, and the rows that correspond to patients without it. Call the resulting matrices `pros.dat.svi` and `pros.dat.no.svi`, respectively. You can do this in two ways: using the `has.svi` Boolean vector created above, or using on-the-fly

Boolean indexing, it's up to you. Check that the dimensions of `pros.dat.svi` and `pros.dat.no.svi` make sense to you.

```
pros.dat.svi <- pros.dat[has.svi,]
pros.dat.no.svi <- pros.dat[!has.svi,]
dim(pros.dat.svi)
```

```
## [1] 21  9
```

```
dim(pros.dat.no.svi)
```

```
## [1] 76  9
```

- **4c.** Using the two matrices `pros.dat.svi` and `pros.dat.no.svi` that you created above, compute the means of each variable in our data set for patients with SVI, and for patients without it. Store the resulting means into vectors called `pros.dat.svi.avg` and `pros.dat.no.svi.avg`, respectively. Hint: for each matrix, you can compute the means with a single call to a built-in R function. What variables appear to have different means between the two groups?

```
pros.dat.svi.avg <- colMeans(pros.dat.svi)
pros.dat.svi.avg
```

```
##    lcavol   lweight       age      lbph       svi       lcp   gleason     pgg45
##  2.551959  3.754927 65.523810 -0.135346  1.000000  1.601858  7.190476 48.809524
##      lpsa
##  3.715360
```

```
pros.dat.no.svi.avg <- colMeans(pros.dat.no.svi)
pros.dat.no.svi.avg
```

```
##     lcavol    lweight        age       lbph        svi        lcp    gleason
##  1.0178918  3.5941312 63.4078947  0.1654837  0.0000000 -0.6715458  6.6315789
##      pgg45       lpsa
## 17.6315789  2.1365916
```

Most of the variables look fairly different apart from lweight and age.

Computing standard deviations using iteration

- **5a.** Take a look at the starter code below. The first line defines an empty vector `pros.dat.svi.sd` of length `ncol(pros.dat)` (of length 9). The second line defines an index variable `i` and sets it equal to 1. Write a third line of code to compute the standard deviation of the `i`th column of `pros.dat.svi`, using a built-in R function, and store this value in the `i`th element of `pros.dat.svi.sd`.

```
pros.dat.svi.sd = vector(length=ncol(pros.dat))
i = 1
pros.dat.svi.sd[i] = sd(pros.dat.svi[,i])
```

- **5b.** Repeat the calculation as in the previous question, but for patients without SVI. That is, produce three lines of code: the first should define an empty vector `pros.dat.no.svi.sd` of length `ncol(pros.dat)` (of length 9), the second should define an index variable `i` and set it equal to 1, and the third should fill the `i`th element of `pros.dat.no.svi.sd` with the standard deviation of the `i`th column of `pros.dat.no.svi`.

```
pros.dat.no.svi.sd = vector(length=ncol(pros.dat))
i = 1
pros.dat.no.svi.sd[i] = sd(pros.dat.no.svi[,i])
```

- **5c.** Write a `for()` loop to compute the standard deviations of the columns of `pros.dat.svi` and `pros.dat.no.svi`, and store the results in the vectors `pros.dat.svi.sd` and `pros.dat.no.svi.sd`, respectively, that were created above. Note: you should have a single `for()` loop here, not two for loops. And if it helps, consider breaking this task down into two steps: as the first step, write a `for()` loop that iterates an index variable `i` over the integers between 1 and the number of columns of `pros.dat` (don't just manually write 9 here, pull out the number of columns programmatically), with an empty body. As the second step, paste relevant pieces of your solution code from Q5a and Q5b into the body of the `for()` loop. Print out the resulting vectors `pros.dat.svi.sd` and `pros.dat.no.svi.sd` to the console. Comment, just briefly (informally), by visually inspecting these standard deviations and the means you computed in Q4c: which variables exhibit large differences in means between the SVI and non-SVI patients, relative to their standard deviations?

```
pros.dat.svi.sd = vector(length=ncol(pros.dat))
for(i in 1:ncol(pros.dat)){
  pros.dat.svi.sd[i] = sd(pros.dat.svi[,i])
}
pros.dat.no.svi.sd = vector(length=ncol(pros.dat))
for(i in 1:ncol(pros.dat)){
  pros.dat.no.svi.sd[i] = sd(pros.dat.no.svi[,i])
}
pros.dat.svi.sd
```

```
## [1]  0.6707867  0.3275689  7.8715885  1.3545258  0.0000000  1.0452899  0.6015852
## [8] 25.7344498  0.9251229
```

```
pros.dat.no.svi.sd
```

```
## [1]  1.0685730  0.4479291  7.3105907  1.4782007  0.0000000  1.0379398  0.7088414
## [8] 25.0667600  0.9646403
```

When comparing difference in means relative to the standard deviations, lcavol and lpsa appear quite different.

- **5d.** The code below computes the standard deviations of the columns of `pros.dat.svi` and `pros.dat.no.svi`, and stores them in `pros.dat.svi.sd.master` and `pros.dat.no.svi.sd.master`, respectively, using `apply()`. (We'll learn `apply()` and related functions a bit later in the course.) Remove `eval=FALSE` as an option to the Rmd code chunk, and check using `all.equal()` that the standard deviations you computed in the previous question equal these "master" copies. Note: use `check.names=FALSE` as a third argument to `all.equal()`, which instructs it to ignore the names of its first two arguments. (If `all.equal()` doesn't succeed in both cases, then you must have done something wrong in computing the standard deviations, so go back and fix them!)

```
pros.dat.svi.sd.master = apply(pros.dat.svi, 2, sd)
pros.dat.no.svi.sd.master = apply(pros.dat.no.svi, 2, sd)
all.equal(pros.dat.svi.sd, pros.dat.svi.sd.master, check.names=FALSE)
```

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

```
all.equal(pros.dat.no.svi.sd, pros.dat.no.svi.sd.master, )
```

```
## [1] "names for current but not for target"
```

Computing t-tests using vectorization

- **6a.** Recall that the **two-sample (unpaired) t-statistic** between data sets $X = (X_1, \dots, X_n)$ and $Y = (Y_1, \dots, Y_m)$ is:

$$T = \frac{\bar{X} - \bar{Y}}{\sqrt{\frac{s_X^2}{n} + \frac{s_Y^2}{m}}},$$

where $\bar{X} = \sum_{i=1}^n X_i/n$ is the sample mean of X, $s_X^2 = \sum_{i=1}^n (X_i - \bar{X})^2/(n-1)$ is the sample variance of X, and similarly for $\bar{Y}$ and s_Y^2. We will compute these t-statistics for all 9 variables in our data set, where X will play the role of one of the variables for SVI patients, and Y will play the role of this variable for non-SVI patients. Start by computing a vector of the denominators of the t-statistics, called `pros.dat.denom`, according to the formula above. Take advantage of vectorization; this calculation should require just a single line of code. Make sure not to include any hard constants (e.g., don't just manually write 21 here for n); as always, programmatically define all the relevant quantities. Then compute a vector of t-statistics for the 9 variables in our data set, called `pros.dat.t.stat`, according to the formula above, and using `pros.dat.denom`. Again, take advantage of vectorization; this calculation should require just a single line of code. Print out the t-statistics to the console.

```
nrow(pros.dat.no.svi)
```

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

```
pros.dat.denom <- sqrt((pros.dat.no.svi.sd/nrow(pros.dat.no.svi))
                       +(pros.dat.svi.sd/nrow(pros.dat.svi)))
pros.dat.t.stat <- (pros.dat.no.svi.avg - pros.dat.svi.avg) / pros.dat.denom
pros.dat.t.stat
```

```
##     lcavol    lweight        age       lbph        svi        lcp    gleason
##  -7.152441  -1.096814  -3.083004   1.038262       -Inf  -9.026504  -2.868072
##      pgg45       lpsa
## -25.000217  -6.627511
```

- **6b.** Given data X and Y and the t-statistic T as defined the last question, the **degrees of freedom** associated with T is:

$$\nu = \frac{(\frac{s_X^2}{n} + \frac{s_Y^2}{m})^2}{\frac{(\frac{s_X^2}{n})^2}{n-1} + \frac{(\frac{s_Y^2}{m})^2}{m-1}}.$$

Compute the degrees of freedom associated with each of our 9 t-statistics (from our 9 variables), storing the result in a vector called `pros.dat.df`. This might look like a complicated/ugly calculation, but really, it's not too bad: it only involves arithmetic operators, and taking advantage of vectorization, the calculation should only require a single line of code. Hint: to simplify this line of code, it will help to first set short variable names for variables/quantities you will be using, as in `sx = pros.dat.svi.sd`, `n = nrow(pros.dat.svi)`, and so on. Print out these degrees of freedom values to the console.

```
sx <-  pros.dat.svi.sd
n <- nrow(pros.dat.svi)
sy <- pros.dat.no.svi.sd
m <- nrow(pros.dat.no.svi)
pros.dat.df <- (sx/n + sy/m)^2 / ((sx/n)^2/(n-1) + (sy/m)^2/(m-1))
pros.dat.df
```

```
## [1] 39.44410 36.57657 31.03698 33.07830      NaN 31.84132 34.17715 31.60416
## [9] 32.46631
```

- **6c.** The function `pt()` evaluates the distribution function of the t-distribution. E.g.,

```
pt(x, df=v, lower.tail=FALSE)
```

returns the probability that a t-distributed random variable, with `v` degrees of freedom, exceeds the value `x`. Importantly, `pt()` is vectorized: if `x` is a vector, and so is `v`, then the above returns, in vector format: the probability that a t-distributed variate with `v[1]` degrees of freedom exceeds `x[1]`, the probability that a t-distributed variate with `v[2]` degrees of freedom exceeds `x[2]`, and so on.

Call `pt()` as in the above line, but replace `x` by the absolute values of the t-statistics you computed for the 9 variables in our data set, and `v` by the degrees of freedom values associated with these t-statistics. Multiply the output by 2, and store it as a vector `pros.dat.p.val`. These are called **p-values** for the t-tests of mean difference between SVI and non-SVI patients, over the 9 variables in our data set. Print out the p-values to the console. Identify the variables for which the p-value is smaller than 0.05 (hence deemed to have a significant difference between SVI and non-SVI patients). Identify the variable with the smallest p-value (the most significant difference between SVI and non-SVI patients).

```
pvals <- pt(abs(pros.dat.t.stat), df=pros.dat.df, lower.tail=FALSE)
pvals
```

```
##       lcavol      lweight          age         lbph          svi          lcp
## 6.220770e-09 1.399458e-01 2.138606e-03 1.533392e-01          NaN 1.366155e-10
##      gleason        pgg45         lpsa
## 3.514386e-03 1.089864e-22 8.333242e-08
```

```
pvals[pvals<0.05]
```

```
##       lcavol          age         <NA>          lcp      gleason        pgg45
## 6.220770e-09 2.138606e-03           NA 1.366155e-10 3.514386e-03 1.089864e-22
##         lpsa
## 8.333242e-08
```

```
min(pvals[-5])
```

```
## [1] 1.089864e-22
```

lcavol, age, lcp, gleason, pgg45, and lpsa all have p values lower than 0.05. pgg45 has the strongest statistical relationship with SVI.