Open Intro Biostat R Companion: Chapter 1

Morgan F. Breitmeyer

Contents

This document is intended to be a companion source to the textbook *Introductory Statistics for the Life and Biomedical Sciences* that goes through all of the *R* code and operations seen in the text.

We have created a package that stores all of the datasets seen in the text. You can download the package using the following command:

```r
install.packages("OIBioStat")  ## make sure to include the quotations
```

The above command only needs to be run once, as after that, the package will be in your computer. However, each time you want to use the package, you must alert *R* using the following command:

```r
require(OIBioStat) ## note the lack of quotations here
```

1 Case study: preventing peanut allergies

The first dataset encountered in the text is from the "Learning Early About Peanut Allergy" (LEAP) study. The dataset contains the results of a study conducted in 2015, which was reported in the *New England Journal of Medicine.* There are several basic things we can do to learn about our dataset. Firstly, we can see the help files that come with it, using the following command:

```r
help(LEAP)
```

Running this, you should see a file appear, which contains several useful pieces of information:

- **Description:** This is a very short description of the dataset. It typically contains a short overview of what is contained, but few details.
- **Format:** This is typically the most useful section of the help file. It contains the names and descriptions of every variable in the dataset. It should contain information such as the units and the type of variable seen. A variable can take two main types: a *factor* meaning it can be one of several set categories which do not have quantitative meaning (such as male and female or racial identification) or a *number* meaning the variable can take on a range of quantitative values, which typically have some sort of meaning (such as height, weight, or frequency).
- **Details:** This section is an extension of the Description section seen at the beginning of the help file. It will typically offer more details about where the data came from and its previous or intended uses.

The next important thing to know about datasets is how to take a look at the dataset itself. The simplest way to do this is as follows:

```r
View(LEAP)
```

This command should cause a new window to pop up that shows you the whole dataset. Here you can see the variable names you saw in your help file across the top as the names of the columns. Also, you can see the entries in the dataset, as the various values within the rows and columns. Note that the farthest left column should show what are called indeces, these are computer generated values that allow you to access specific rows in the dataset, but are not meaningful values in the dataset. It is important to note that these indeces typically have no meaning that corresponds with the data, but are very useful for working with the data.

Another simple way to view a small portion of the data is as follows:

```
LEAP[1:5,1:6]

##   participant.ID    treatment.group age.months    sex primary.ethnicity
## 1    LEAP_100522 Peanut Consumption     6.0780 Female             Black
## 2    LEAP_103358 Peanut Consumption     7.5893 Female             White
## 3    LEAP_105069   Peanut Avoidance     5.9795   Male             White
## 4    LEAP_105328 Peanut Consumption     7.0308 Female             White
## 5    LEAP_106377   Peanut Avoidance     6.4066   Male             White
##   overall.V60.outcome
## 1            PASS OFC
## 2            PASS OFC
## 3            PASS OFC
## 4            PASS OFC
## 5            PASS OFC
```

This command will print out in the console the first 5 rows of the data. The bracket notation above after the dataset name implies location, so you are telling *R* at what location you would like to look. The notation here is *dataset[rows,columns]*, so the 1:5 dictates the first 5 rows and the 1:6 dictates the first 6 columns. You can use this notation to access any combination of rows and columns in the dataset that you wish. If you wanted to tell *R* to take all the columns, but only the first five rows you could do that with the following:

```
## note the space (or lack of text after the comma)
LEAP[1:5, ]

##   participant.ID    treatment.group age.months    sex primary.ethnicity
## 1    LEAP_100522 Peanut Consumption     6.0780 Female             Black
## 2    LEAP_103358 Peanut Consumption     7.5893 Female             White
## 3    LEAP_105069   Peanut Avoidance     5.9795   Male             White
## 4    LEAP_105328 Peanut Consumption     7.0308 Female             White
## 5    LEAP_106377   Peanut Avoidance     6.4066   Male             White
##   overall.V60.outcome
## 1            PASS OFC
## 2            PASS OFC
## 3            PASS OFC
## 4            PASS OFC
## 5            PASS OFC
```

This output actually gives the same as the above command, but with a different command. This is a common theme in *R* - there are multiple ways to do everything.

The final and most complex way to look at part of the data can be seen below and in Table 1.1 in the text. In this case, rather than doing a list of numbers sequentially, we create a list using the command *c*() which binds the values inside the parentheses together into a list. We can see the command below has first a list of row indexes (numbers as described above) and then a list of column names.

```
## Table 1.1
LEAP[c(1,2,3,529, 530),c("participant.ID", "treatment.group",
                         "overall.V60.outcome")]

##     participant.ID    treatment.group overall.V60.outcome
## 1      LEAP_100522 Peanut Consumption            PASS OFC
## 2      LEAP_103358 Peanut Consumption            PASS OFC
## 3      LEAP_105069   Peanut Avoidance            PASS OFC
## 639    LEAP_994047   Peanut Avoidance            PASS OFC
## 640    LEAP_997608 Peanut Consumption            PASS OFC
```

Another good way to visualize data is using a table, which summarizes the data using two variables, one on each axis, and then provides the number of counts matching the varaible categories. We can see this with the below command, which corresponds to Table 1.2 in the text. Using the *table* command, the first variable put in goes to the x-axis (meaning the row names) and the second variable goes to the y-axis (meaning the column names). The addition of the *addmargins()* command prints the sums of the rows and columns on the sides of the table as seen below,

```
## Table 1.2
table(LEAP$treatment.group, LEAP$overall.V60.outcome)

##
##                      FAIL OFC PASS OFC
##   Peanut Avoidance         36      227
##   Peanut Consumption        5      262

addmargins(table(LEAP$treatment.group, LEAP$overall.V60.outcome))

##
##                      FAIL OFC PASS OFC Sum
##   Peanut Avoidance         36      227 263
##   Peanut Consumption        5      262 267
##   Sum                      41      489 530
```

2 Data Basics

We can partition part of the entire data to make what is called a dataframe. We do this as follows below, where the brackets again indicate the location and the values inside these brackets indicate rows 1-3 and 150 and all columns for those rows. This output corresponds to Table 1.3 in the text. By separating this part of the dataframe as its own variable, here called *data.matrix*, we can then call this specific part of the data for later operations.

```
## Table 1.3
data.matrix = frog.altitude.data[c(1:3, 150),]
data.matrix
```

```
##     altitude latitude clutch.size body.size clutch.volume egg.size
## 1   3,462.00    34.82    181.9701  3.630781      177.8279 1.949845
## 2   3,462.00    34.82    269.1535  3.630781      257.0396 1.949845
## 3   3,462.00    34.82    158.4893  3.715352      151.3561 1.949845
## 150 2,597.00    34.05    537.0318        NA      776.2471 2.238721
```

We can perform a similar operation on the *famuss* data to get Table 1.5 in the text:

```
## Table 1.5
famuss[c(1,2,3,595),c( "sex", "age", "race", "height", "weight", "actn3.r577x",
                       "ndrm.ch")]

##         sex age      race height weight actn3.r577x ndrm.ch
## 1    Female  27 Caucasian   65.0    199          CC    40.0
## 2      Male  36 Caucasian   71.7    189          CT    25.0
## 3    Female  24 Caucasian   65.0    134          CT    40.0
## 1348 Female  30 Caucasian   64.0    134          CC    43.8
```

4 Numerical Data

Now that we know how to look at the data itself, we may want to consider some methods to numerically summarize the data. Doing this, we can get a good idea of the patterns seen in the data with just a few key values. Each of these values can be calculated by hand using a mathematical formula or in *R* using a function which calculates the value for you.

4.1 Measures of Center – Mean and Median

The first numerical summary value we will consider is the **mean**, which provides the average value of the data. The mathematical formula to do this by hand is

$$\bar{X} = \frac{X_1 + X_2 + \cdots + X_n}{n}$$

where n is the number of X observations in the data. We can perform this calculation in R with the following.

```
## Solving by hand
# First solve for n - the number of observations in the data
n = length(frog.altitude.data$clutch.volume)
n

## [1] 431

x.bar.h = sum(frog.altitude.data$clutch.volume)/n
x.bar.h

## [1] 882.474
```

Alternatively, we can use an R provided function that will automatically calculate this value with no work on our part, as follows:

```
## Compare to Using R function
x.bar.r = mean(frog.altitude.data$clutch.volume)
x.bar.r

## [1] 882.474

## We can also round to get the exact values seen in text
x.bar.rounded = round(x.bar.r, 1)
x.bar.rounded

## [1] 882.5
```

Finally, we could also consider getting the **median** value of the dataset, which is the center value in the data. You could think of it as the observation $X_{n/2}$, but the best way to obtain this value is using the following R function:

```
median(frog.altitude.data$clutch.volume)

## [1] 831.7638
```

4.2 Measures of Spread – Standard Deviation and Variance

Now that we have an idea of the center of the data, it is of interest to know how the data spreads away from that center. This is where the **standard deviation** comes in. The **variance** is another measure of spread and is simply equal to the square of the standard deviation. To calculate the variance, you can follow a few steps:

1. Obtain the deviation of each data point from the mean, i.e. $x_i - \bar{x}$
2. Square each of these deviations, i.e. $(x_i - \bar{x})^2$
3. Sum all of the squares, i.e. $\sum_{i=1}^{n}(x_i - \bar{x})^2$
4. To obtain the final value for variance, divide this sum by the number of observations minus 1, $n-1$, i.e. $(\frac{1}{n-1})\sum_{i=1}^{n}(x_i - \bar{x})^2$
5. To obtain the standard deviation, take the square root of the variance, i.e $\sqrt{(\frac{1}{n-1})\sum_{i=1}^{n}(x_i - \bar{x})^2}$

We go through these steps by hand here and compare to the R function's output.

```
## Looking at a few sample values from text
frog.altitude.data$clutch.volume[c(1,2,3,431)]-x.bar.r

## [1] -704.64604 -625.43440 -731.11786   50.78032
## Calculating variance by hand
# Step 1: calculating all deviations
diffs = frog.altitude.data$clutch.volume-x.bar.r

# Steps 2-4:
variance = sum(diffs^2)/(n-1)
variance

## [1] 143680.9

## Comparing to the R function's value
var(frog.altitude.data$clutch.volume)

## [1] 143680.9

## Step 5: obtaining standard deviation
stand.dev = sqrt(variance)
stand.dev

## [1] 379.0527

sd(frog.altitude.data$clutch.volume)        ## compare to R function value

## [1] 379.0527
```

A third measure of spread is the **interquartile range (IQR)**, which measures variability. The IQR is equal to the third quartile (the 75th percentile), labeled Q_3, minus the first quartile (the 25th percentile), labeled Q_1, i.e. $IQR = Q_3 - Q_1$. The IQR can be obtained in R using the following function,

```
IQR(frog.altitude.data$clutch.volume)

## [1] 486.9009
```

4.3 Summary

A final way to get a lot of the above information about the numerical data can be obtained using the *summary* command in R as follows,

```
summary(frog.altitude.data$clutch.volume)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   151.4   609.6   831.8   882.5  1096.0  2630.0
```

4.4 With and Without Extreme Outliers

```
# Table 1.13
loc = (frog.altitude.data$clutch.volume<= 2000)

## Robust estimates
median(frog.altitude.data$clutch.volume[loc])

## [1] 831.7638

IQR(frog.altitude.data$clutch.volume[loc])

## [1] 493.9186

## Non-robust estimates
mean(frog.altitude.data$clutch.volume[loc])

## [1] 867.9425

sd(frog.altitude.data$clutch.volume[loc])

## [1] 349.1596
```

4.5 Visualizing Data

One of the most important functions of R is in data visualization. As we saw above, most numerical summaries of data can be caclulated by hand or in R, but creating plots and other visualizations are much more difficult by hand.

4.5.1 Histograms

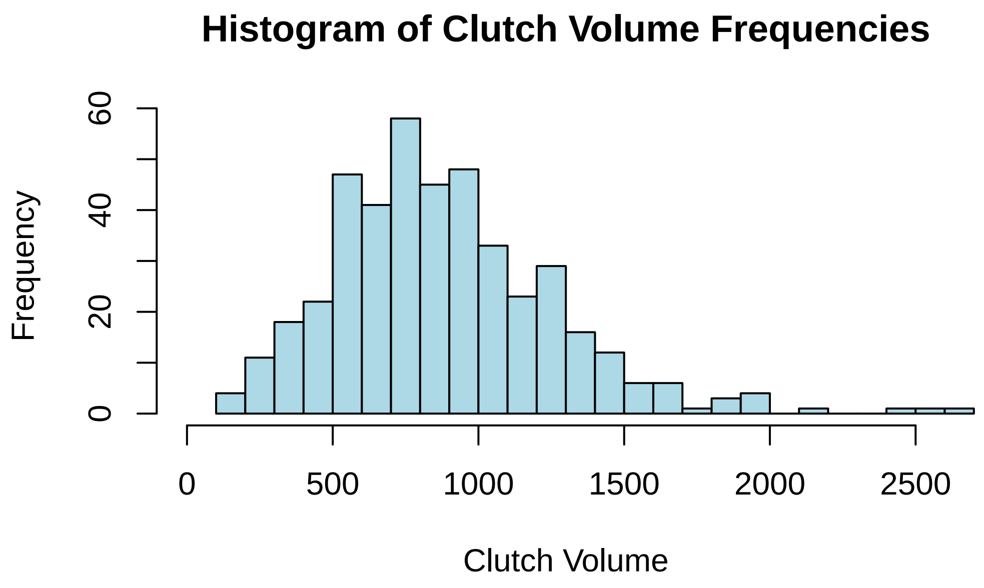
The first type of plot we will consider is a **histogram**, which plots the frequency of the various values of a variable. The command for doing this is *hist()*, and this function takes several arguments:

- x: the variable you want to visualize
- $breaks$: the number of vertical boxes (called bins) you want the plot to have; higher numbers will equal more boxes
- col: the color you want your plot to be; if unspecified, it will be white; must be in quotation marks
- $xlab$: the label you would like the x-axis to have; must be in quotation marks
- $ylab$: the label you would like the y-axis to have; must be in quotation marks
- $xlim$: the range of values you would like the x-axis to have; of the form $c(lowerbound, upperbound)$

- *ylim*: the range of values you would like the y-axis to have; of the form $c(lowerbound, upperbound)$
- *main*: the title you would like the whole plot to have; must be in quotation marks

In order to use each argument, you write the argument name followed by an equals sign, followed by what information you want to give it. Note that not all arguments must be used - only the x argument is necessary, the rest are just optional preferences. An example of this can be seen below, plotting Figure 1.15 from the text:

```
## Figure 1.15
hist(x = frog.altitude.data$clutch.volume, breaks = 20, col = "lightblue",
     xlab = "Clutch Volume", ylab = "Frequency", xlim = c(0, 2600),
     main = "Histogram of Clutch Volume Frequencies")
```



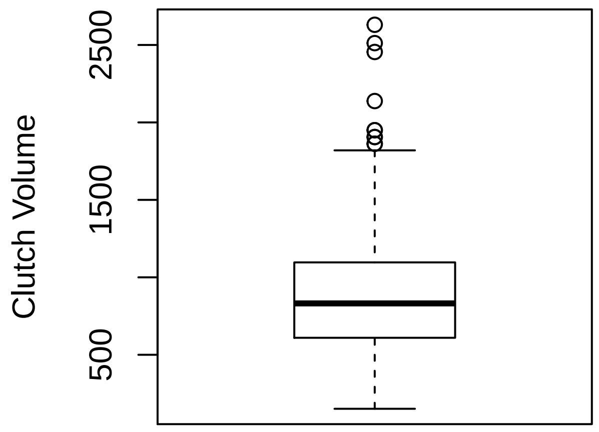
4.5.2 Boxplots

Another way to visualize data is with a boxplot. The command for doing this in *R* is *boxplot()*, and it takes the following arguments:

- *x*: the variable you want to visualize
- *axes*: a TRUE/FALSE indicator that determines whether or not numbers are shown on the axes
- *col*: the color you want the fill inside your boxplot to be; if unspecified, it will be white; must be in quotation marks
- *xlab*: the label you would like the x-axis to have; must be in quotation marks

- *ylab*: the label you would like the y-axis to have; must be in quotation marks
- *xlim*: the range of values you would like the x-axis to have; of the form $c(lowerbound, upperbound)$; fairly meaningless for a boxplot
- *ylim*: the range of values you would like the y-axis to have; of the form $c(lowerbound, upperbound)$; be careful not to specify too much here because you can eliminate data of importance to the boxplot
- *main*: the title you would like the whole plot to have; must be in quotation marks

```
## Simplified Figure 1.17
boxplot(x = frog.altitude.data$clutch.volume, ylab = 'Clutch Volume', axes = TRUE,
        ylim = range(frog.altitude.data$clutch.volume))
```



4.5.3 Scatterplots

Scatterplots can be used to visualize the relationship between two variables or to visualize just one variable individually.

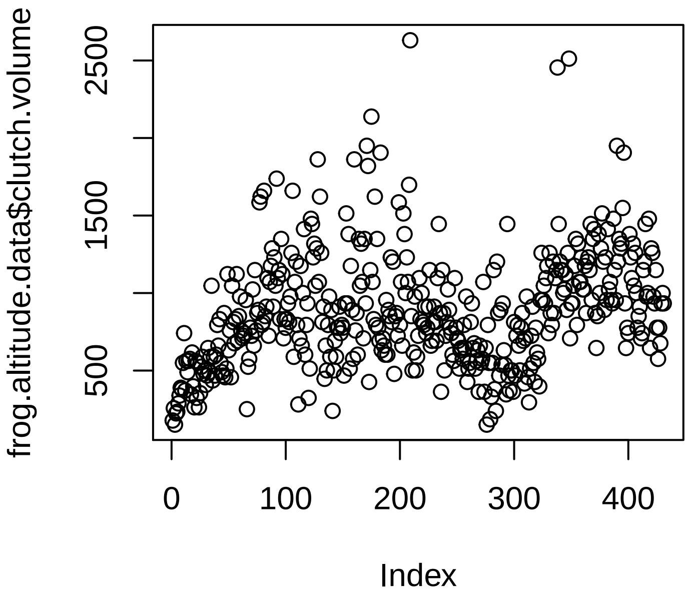
The command for doing this is *plot()*, and this function takes several arguments:

- x: the first variable you want to visualize; *necessary to include*
- y: the second variable you may want to visualize; included if you want to consider two variables *not necessary to include*

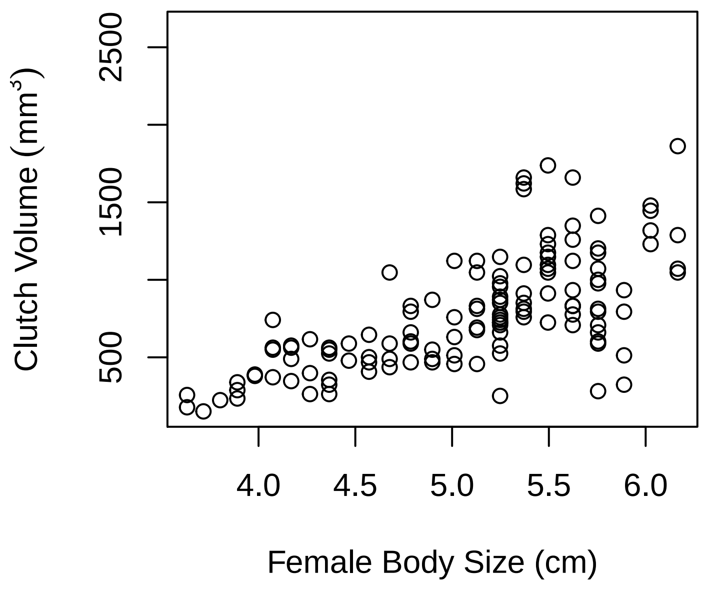
- *col*: the color you want your dots on the plot to be; if unspecified, they will be black; must be in quotation marks
- *type*: specifies how you want R to plot the data; if unspecified, you will end up with dots (in a scatterplot); other options include "*l*" for lines between data points
- *xlab*: the label you would like the x-axis to have; must be in quotation marks
- *ylab*: the label you would like the y-axis to have; must be in quotation marks
- *xlim*: the range of values you would like the x-axis to have; of the form $c(lowerbound, upperbound)$
- *ylim*: the range of values you would like the y-axis to have; of the form $c(lowerbound, upperbound)$
- *main*: the title you would like the whole plot to have; must be in quotation marks

The *plot* command has an interesting feature that you can either specify your x and y variables by running $plot(x, y)$ or by running $plot(y\ x)$. Either command will give the same result. Below provides an example of using the *plot* command for one and two variables (corresponds to Figure 1.18 in the text):

```r
# For one variable
plot(frog.altitude.data$clutch.volume)
```



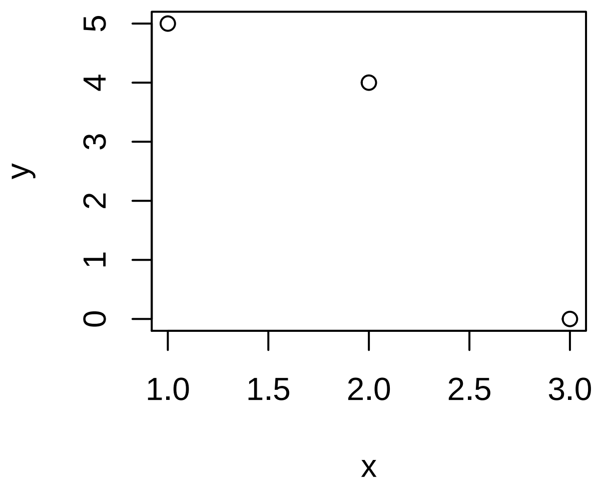
```r
## Figure 1.18
# For two variables,
plot(frog.altitude.data$clutch.volume~frog.altitude.data$body.size,
     xlab = "Female Body Size (cm)", ylab = expression("Clutch Volume" ~ (mm^3)))
```



4.6 Correlation

Correlation can be calculated by hand but it's a tedious calculation, so doing it using *R* is often the best bet. It can be done with the command *cor()* which takes as its arguments your two variables of interest. Using the example from the text, we can create Figure 1.22 and calculate the correlation as follows:

```r
## Figure 1.22
x = c(1, 2, 3)
y = c(5, 4, 0)

plot(x,y)
```



```
cor(x,y)

## [1] -0.9449112
```

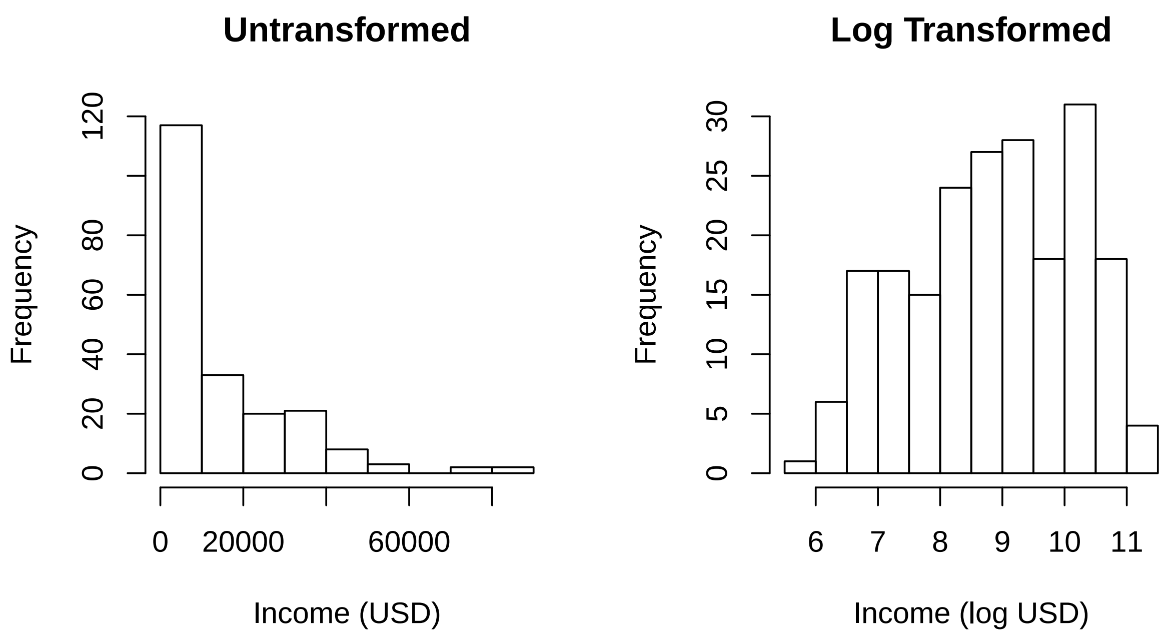
Figure 1.21 can also be seen with the following with the correlation calculated as stated in the book to be 0.63. Note that this correlation command contains the argument *use = "complete.obs"*, which simply allows for the correlation calculation by ignoring any NA values in the dataset.

```
cor(life.expectancy.income$income, life.expectancy.income$life.expectancy,
    use = "complete.obs")

## [1] 0.6308783
```

4.7 Transforming Data

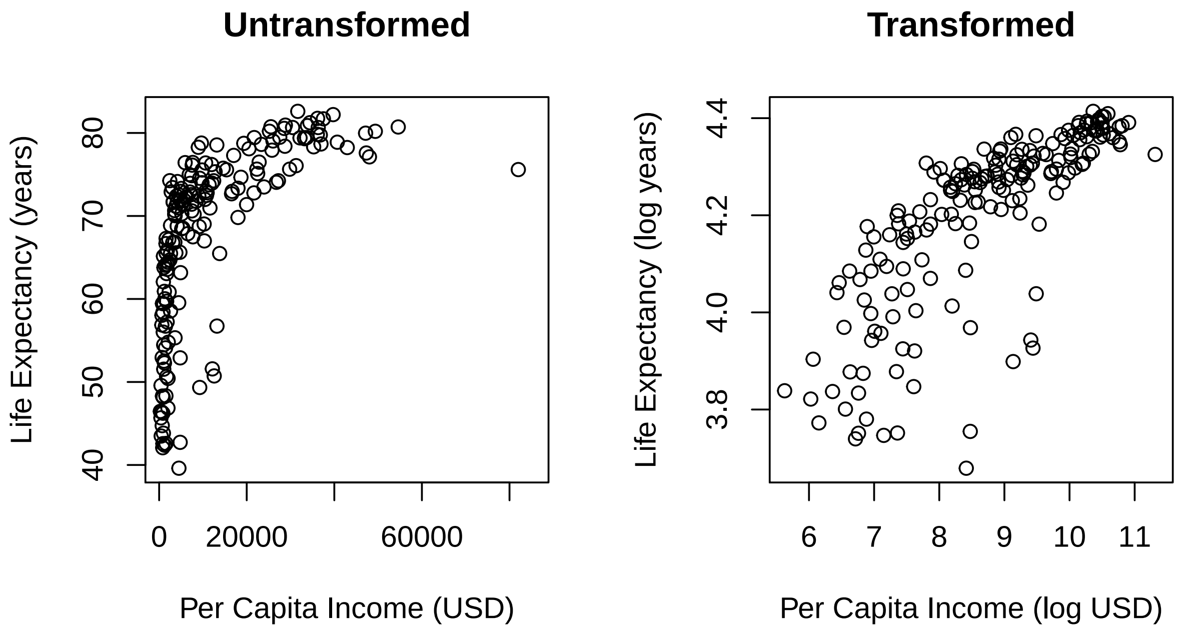
Transformations of data are often of interest, and performing them in *R* is quite easy. The plot below shows the before and after of a log transformation on the *life expectancy-income* data. The *log* command in *R* perform a natural log, or ln, on the data, despite being written as log.

```
## Figure 1.24
par(mfrow = c(1, 2))  ## this line just allows two plots to print at the same time
hist(life.expectancy.income$income, breaks = 12, xlab = "Income (USD)",
    ylab = "Frequency", ylim = c(0, 120), main = "Untransformed")

hist(log(life.expectancy.income$income), breaks = 12, xlab = "Income (log USD)",
    ylab = "Frequency", ylim = c(0, 30), main = "Log Transformed")
```



Figures 1.21 and 1.25 in the text can be seen below next to each other to see how the log transformation changes a scatterplot. The function *par()* creates partitions in the graphing output. In this case, we get 1 row and 2 columns, as indicated by *mfrow = c(1,2)*

```
## Figure 1.21 and Figure 1.25
par(mfrow = c(1, 2))
plot(life.expectancy.income$income,
    life.expectancy.income$life.expectancy,
    ylab = "Life Expectancy (years)",
    xlab = "Per Capita Income (USD)",
    main = "Untransformed")
plot(log(life.expectancy.income$income),
    log(life.expectancy.income$life.expectancy),
    ylab = "Life Expectancy (log years)",
    xlab = "Per Capita Income (log USD)",
    main = "Transformed")
```



5 Categorical Data

5.1 Contingency Tables

Shown below, Table 1.26 shows a **frequency table** for genotypes of the actn3.r577x gene. As a frequency table, it gives the counts of each genotype within the dataset.

```
## Table 1.26
addmargins(table(famuss$actn3.r577x))

##
##  CC  CT  TT Sum
## 173 261 161 595
```

Table 1.27 shows a frequency table on two variables, genotype of the actn3.r577x gene and race from the *famuss* dataset. Using the command *addmargins* here gives us **marginal totals**. Because this data summarizes two categorical variables, it is called a **contingency table**.

```
## Table 1.27
addmargins(table(famuss$race, famuss$actn3.r577x))

##
##                CC  CT  TT Sum
##   African Am   16   6   5  27
##   Asian        21  18  16  55
##   Caucasian  125 216 126 467
##   Hispanic     4  10   9  23
##   Other        7  11   5  23
##   Sum        173 261 161 595
```

If we instead wanted a contingency table with proportions rather than counts, we could do that in one of two ways - either row proportions or column proportions. Table 1.28 shows the row proportions and the column proportions are calculated just below.

```
## Table 1.28
row.prop.table = table(famuss$race, famuss$actn3.r577x)[1:5,]
row.prop.table / rep(rowSums(row.prop.table), 3)

##
##                     CC        CT        TT
##   African Am 0.5925926 0.2222222 0.1851852
##   Asian      0.3818182 0.3272727 0.2909091
##   Caucasian  0.2676660 0.4625268 0.2698073
##   Hispanic   0.1739130 0.4347826 0.3913043
##   Other      0.3043478 0.4782609 0.2173913
```

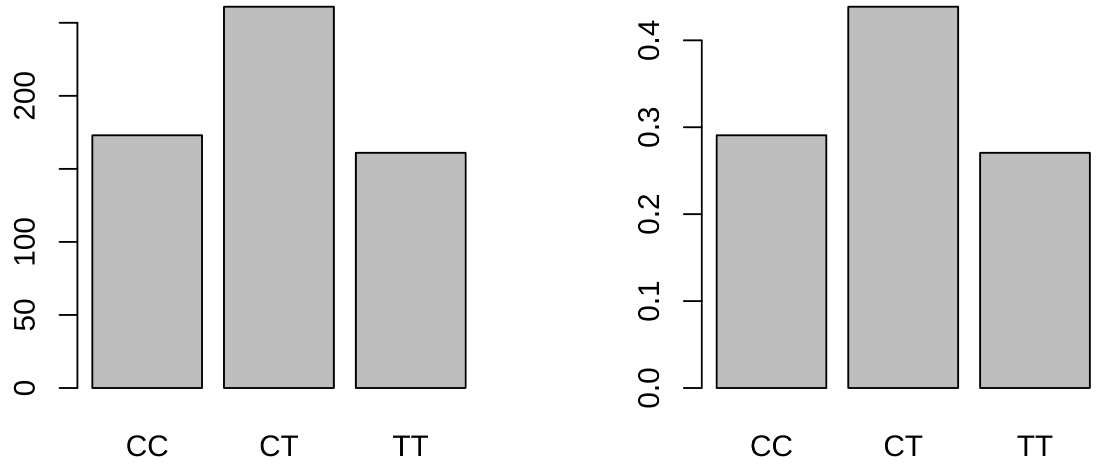
Alternatively, we could build a similar table but performing column proportions rather than row proportions as seen in Table 1.29:

```
## Table 1.29
col.prop.table = table(famuss$race, famuss$actn3.r577x)[1:5,]
col.prop.table / rep(colSums(col.prop.table), 3)

##
##                      CC         CT         TT
##   African Am 0.09248555 0.03726708 0.01915709
##   Asian      0.08045977 0.10404624 0.09937888
##   Caucasian  0.77639752 0.82758621 0.72832370
##   Hispanic   0.02312139 0.06211180 0.03448276
##   Other      0.02681992 0.06358382 0.03105590
```

5.2 Bar Plots

Another way to visualize categorical data is using a **bar plot**, which can present the data by its categories, and either the count or proportion of occurences within each category. Note how the *barplot* command takes as its input a table, rather than just a variable. This is because the data needs to be sorted by its categories before plotting. We can see Figure 1.30 below which shows two barplots of the same data, sorted on the left by counts and on the right by frequencies.

```r
## Figure 1.30
par(mfrow = c(1, 2))
barplot(table(famuss$actn3.r577x))  ## count barplot
barplot(table(famuss$actn3.r577x)/sum(table(famuss$actn3.r577x)))
```



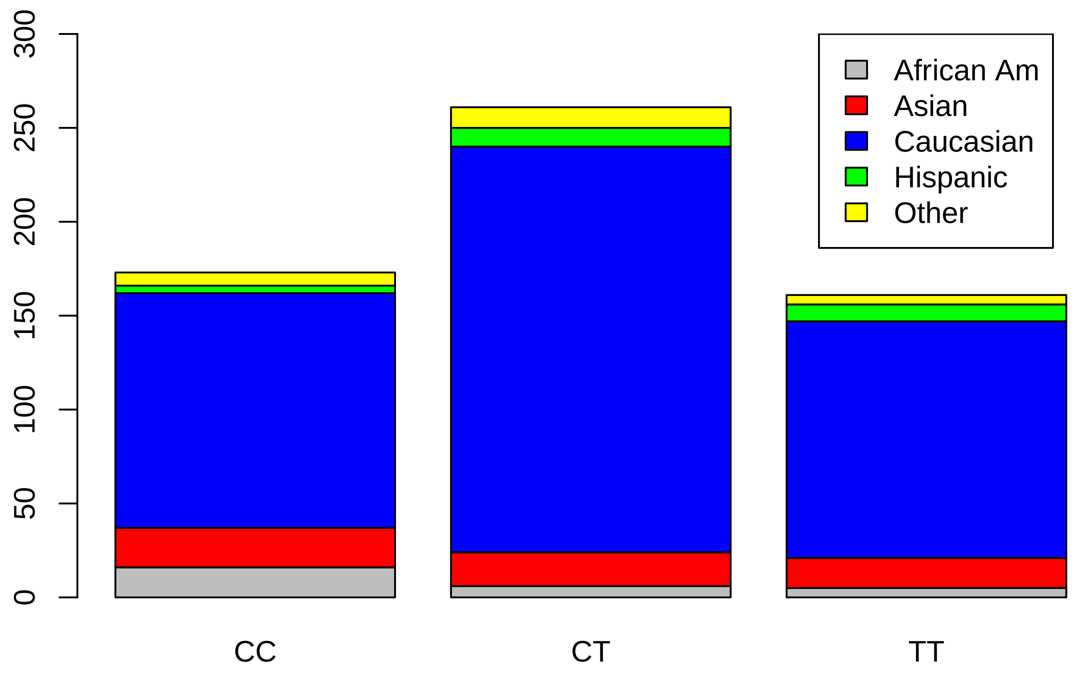
```r
## frequency barplot
```

5.2.1 Segmented Bar Plot

A segmented bar plot is a more sophisticated version of the barplots seen above. The data is now sorted on two categorical variables, so This code is the most sophisticated thus far, so try to as best as possible work through it line by line.

```r
## Figure 1.31a first, create a table of the data that is
## sorted
genotype.race = matrix(table(famuss$actn3.r577x, famuss$race),
    ncol = 3, byrow = T)

# second, change the column and row names on the table
colnames(genotype.race) = c("CC", "CT", "TT")
rownames(genotype.race) = c("African Am", "Asian", "Caucasian",
    "Hispanic", "Other")

# third, plot the barplot where colors are specified
```

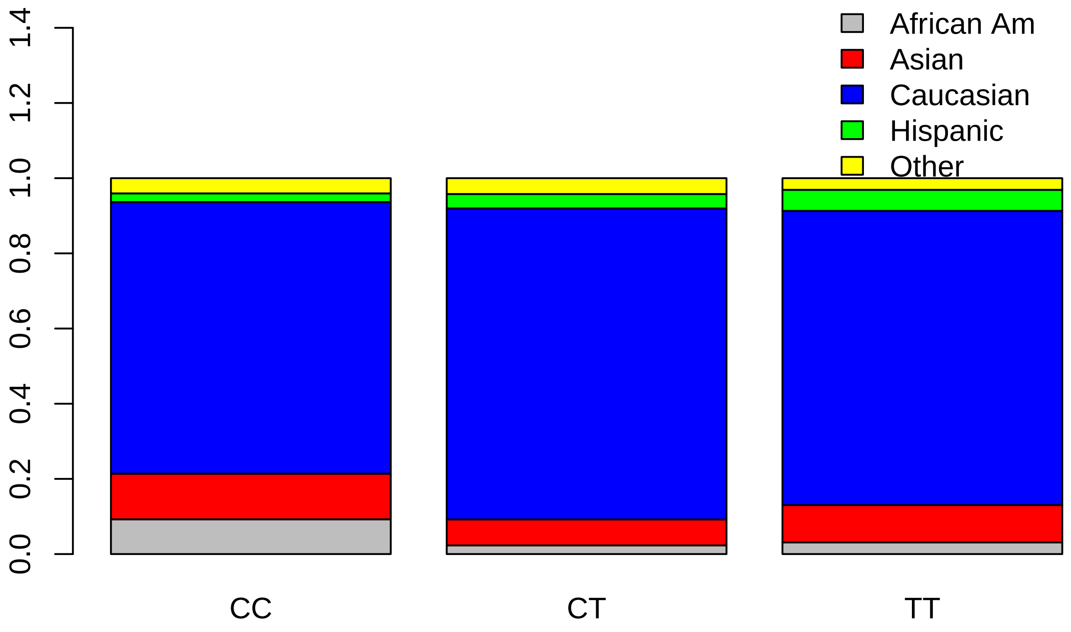
```r
barplot(genotype.race, col = c("grey", "red", "blue", "green",
    "yellow"), ylim = c(0, 300), width = 2)
# lastly, include a legend for intepretation of your plots
legend("topright", inset = c(0.05, 0), fill = c("grey", "red",
    "blue", "green", "yellow"), legend = rownames(genotype.race))
```



If we instead wanted to make a **standardized segmented bar plot**, we could do that by creating a table of proportions (instead of counts) using the command *prop.table*. This command is similar to the *table* command but it calculates the proportions for you. The only difference in the following code from the section above is the use of a proportion table in the first step.

```r
## Figure 1.31b first, create table of proportions
prop.genotype.race <- prop.table(genotype.race, 2)

# second, change the column and row names on the table
colnames(prop.genotype.race) = c("CC", "CT", "TT")
rownames(prop.genotype.race) = c("African Am", "Asian", "Caucasian",
    "Hispanic", "Other")

# second, plot the output
```

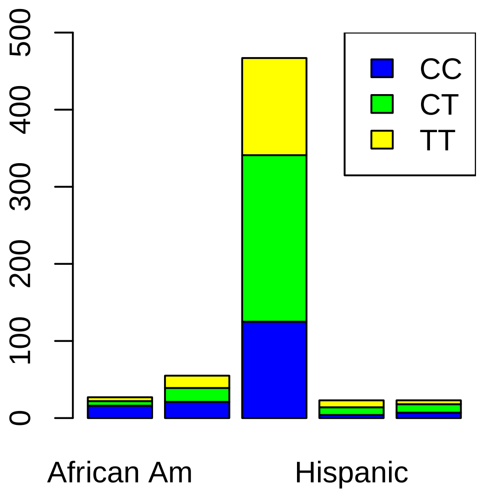
```r
barplot(prop.genotype.race, col = c("grey", "red", "blue", "green",
    "yellow"), ylim = c(0, 1.5), width = 2)

# lastly, include a legend for intepretation of your plots
legend("topright", inset = c(0.05, -0.005), fill = c("grey",
    "red", "blue", "green", "yellow"), legend = rownames(prop.genotype.race),
    bty = "n")
```

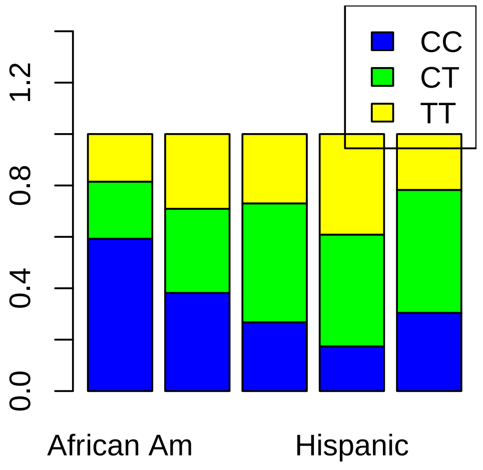


Note: In the *legend* line, the command *inset=* controls where on the plot the legend prints out. Sometimes this specification can be a little bit bizarre so it may require some alteration to find the right spot for the legend.

If we wanted to reproduce Figure 1.32, where the bars indicate race instead of genotype, we can do the same process as above, with one exception. In the first step, of creating the table, we must instead put the race variable first as the rows and the gene variable as the columns, as follows:

```r
## Figure 1.32 Plotting two graphs next to each other
par(mfrow = (c(1, 2)))

# Setting up the table and changing column/row names
race.genotype = matrix(table(famuss$race, famuss$actn3.r577x),
    ncol = 5, byrow = T)
colnames(race.genotype) = c("African Am", "Asian", "Caucasian",
    "Hispanic", "Other")
rownames(race.genotype) = c("CC", "CT", "TT")

# Creating segmented bar plot with a legend
barplot(race.genotype, col = c("blue", "green", "yellow"), ylim = c(0,
    500), width = 2)
legend("topright", inset = c(0, 0), fill = c("blue", "green",
    "yellow"), legend = rownames(race.genotype))

# Creating standardized segmented bar plot with a legend
prop.race.genotype <- prop.table(race.genotype, 2)
barplot(prop.race.genotype, col = c("blue", "green", "yellow"),
    ylim = c(0, 1.5), width = 2)
legend("topright", inset = c(0, 0), fill = c("blue", "green",
    "yellow"), legend = rownames(race.genotype))
```

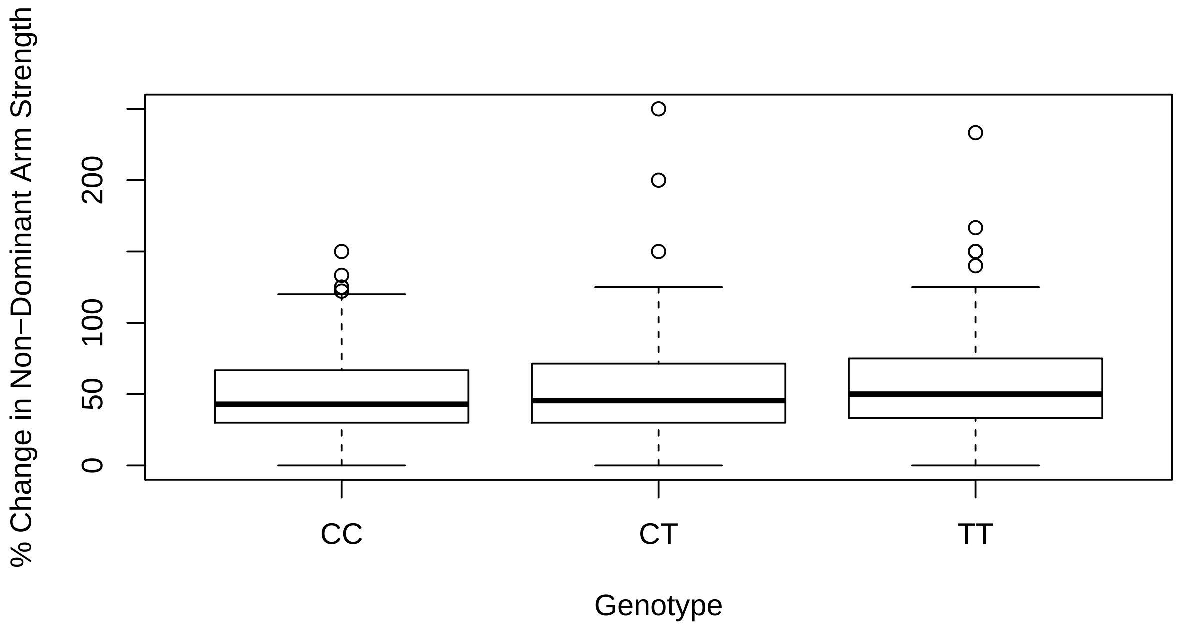




5.2.2 Side-by-Side Boxplots

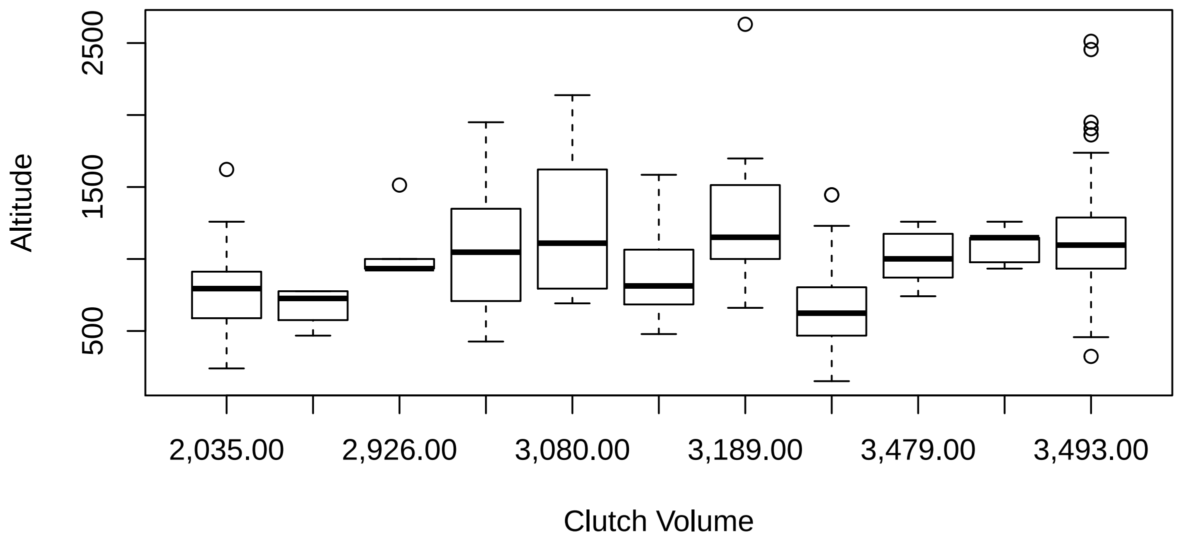
With categorical data, you can also create a **side-by-side boxplot**, as seen in Figure 1.33, which plots the boxplots for corresponding categories on the same plot. It can be plotted as follows,

```r
## Figure 1.33(a)
boxplot(famuss$ndrm.ch ~ famuss$actn3.r577x,
    ylab = "% Change in Non-Dominant Arm Strength",
    xlab = "Genotype")
```



Another example of a larger side-by-side boxplot can be seen in Figure 1.34, created as follows,

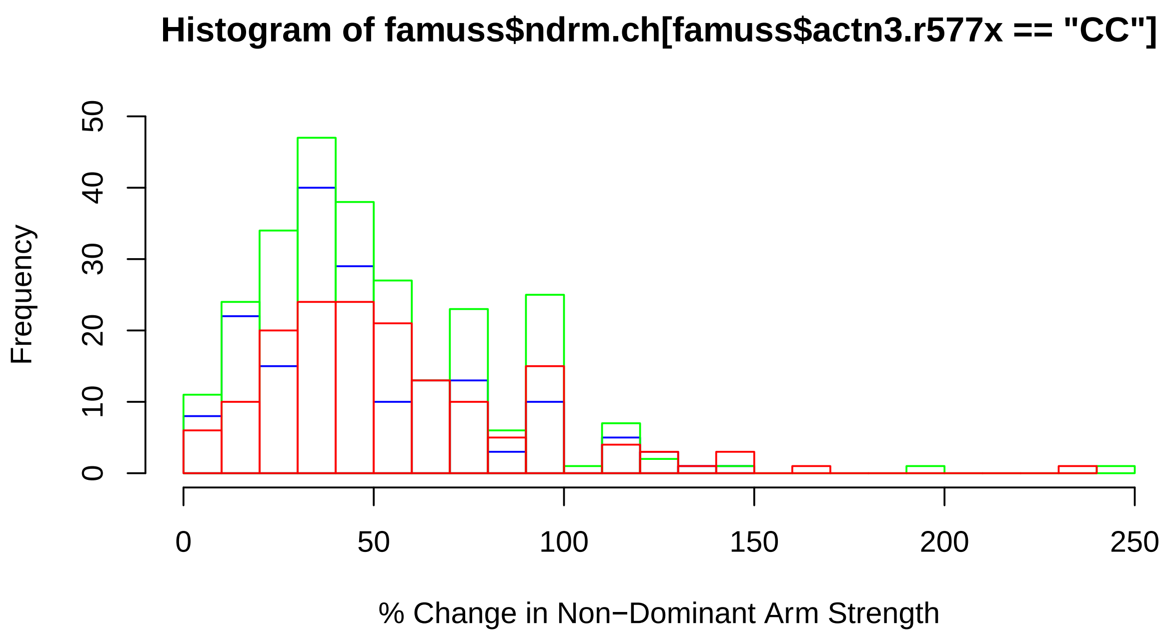
```r
## Figure 1.34
boxplot(frog.altitude.data$clutch.volume ~ frog.altitude.data$altitude,
    xlab = "Clutch Volume", ylab = "Altitude")
```



5.2.3 Hollow Histogram

Using categorical data, we can also plot histograms for each category on top of each other on one single plot, using a **hollow histogram**. This can be done as follows,

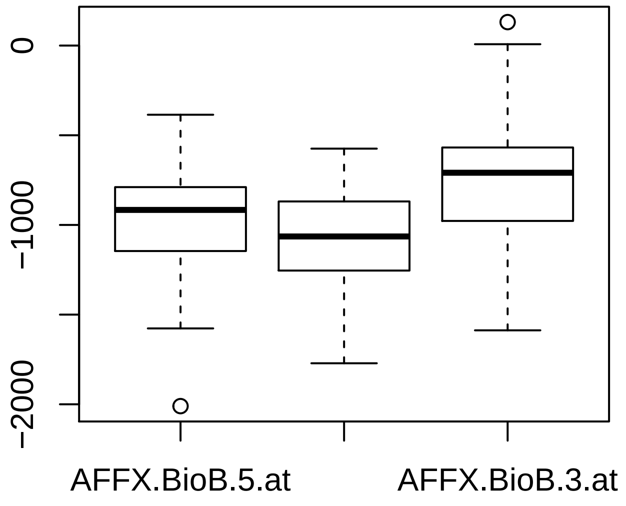
```
## Figure 1.33(b)
hist(famuss$ndrm.ch[famuss$actn3.r577x == "CC"], breaks = 20, border = "blue",
    xlab = "% Change in Non-Dominant Arm Strength", ylim = c(0, 50), xlim = c(0,
        250))
hist(famuss$ndrm.ch[famuss$actn3.r577x == "CT"], breaks = 20, border = "green",
    add = T)
hist(famuss$ndrm.ch[famuss$actn3.r577x == "TT"], breaks = 20, border = "red",
    add = T)
```



6 Genomic Data

```
## Table 1.36
subset.golub = Golub[c(1, 2, 3,
    4, 5, 6), c("Samples", "Gender",
    "cancer", "AFFX.BioB.5.at",
    "AFFX.BioB.M.at", "AFFX.BioB.3.at")]
subset.golub

##    Samples Gender cancer AFFX.BioB.5.at AFFX.BioB.M.at AFFX.BioB.3.at
## 39      39      F   allB     -1363.2764      -1058.585    -541.469194
## 40      40      F   allB      -796.2851      -1167.103       7.538493
## 42      42      F   allB      -679.1392      -1069.832    -690.301829
## 47      47      M   allB     -1164.4002      -1109.940    -990.127218
## 48      48      F   allB     -1299.6538      -1401.999   -1077.543813
## 49      49      M   allB     -1251.6207      -1236.344   -1587.702224
```

```
## Figure 1.37
boxplot(Golub[, 7:9])
```


0
−1000
−2000
AFFX.BioB.5.at
AFFX.BioB.3.at