Book Title

Book Subtitle

Morgan F. Breitmeyer

Harvard University
Cambridge, MA
September 29, 2016

Contents

0 Preface **2**

1 Introduction to Basic Techniques **3**
1.1 Case study: preventing peanut allergies 3
1.2 Data Basics 5
1.3 Numerical Data 6
1.4 Categorical Data 17
1.5 Genomic Data 27

2 Using R for Probabilities **28**
2.1 Bayes' Theorem 28
2.2 Contingency Table 29
2.3 Simulation 30

3 Distributions **33**
3.1 Normal Distribution 33
3.2 Binomial Distribution 37
3.3 Continuous and Discrete Distributions 38
3.4 Other Distributions 39

4 Chapter 4 - Name this **41**

Chapter 0

Preface

This text is a companion to *Introductory Statistics for the Life and Biomedical Sciences*; while the main text focuses on statistical concepts and ideas, this supplement provides details about how to use the statistical computing language R.

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

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

Chapter 1

Introduction to Basic Techniques

Contents

1.1 Case study: preventing peanut allergies 3
1.2 Data Basics 5
1.3 Numerical Data 6
1.3.1 Measures of center: mean and median 6
1.3.2 Measures of spread: standard deviation and variance 7
1.3.3 Robust statistics 8
1.3.4 Visualizing distributions of data 8
1.3.5 Scatterplots and correlation 12
1.3.6 Correlation 14
1.3.7 Transforming Data 16
1.4 Categorical Data 17
1.4.1 Contingency Tables 17
1.4.2 Bar Plots 18
1.4.3 Segmented Bar Plot 20
1.4.4 Comparing numerical data across groups 25
1.5 Genomic Data 27

This chapter introduces basic commands for manipulating datasets, including how to calculate numerical summaries, create tables from data, and make graphical plots.

1.1 Case study: preventing peanut allergies

The `LEAP` dataset contains the results of the "Learning Early About Peanut Allergy" (LEAP) study, an experiment conducted to assess whether early exposure to peanut products reduces the probability of peanut allergies developing. To access the documentation associated with the dataset, run the following command:

```
help(LEAP)
```

A file will appear in the Help pane that provides some basic information about the dataset:

- **Description:** A general overview of the dataset.
- **Usage:** Instructions for how to load the dataset.

- **Format:** The names and descriptions of each variable in the dataset, including information about variable type and measurement units.
- **Details:** Additional details about the conditions under which the data were collected.
- **Source**: Information about where the data originates from.

To view the dataset itself, run:

```
View(LEAP)
```

In the main console pane, a window will appear that shows the entire dataset. The variable names are displayed across the top, as the names of the columns. Data for each case in the study are contained within the rows. Note that the farthest left column shows the **indices**, which are computer-generated values that allow specific rows in the dataset to be accessed. These can be used to view a specific portion of the data.

For example, to print out data contained in the first 5 rows and first 3 columns of the data, use the following syntax:

```
LEAP[1:5,1:3]

##   participant.ID    treatment.group age.months
## 1    LEAP_100522 Peanut Consumption     6.0780
## 2    LEAP_103358 Peanut Consumption     7.5893
## 3    LEAP_105069   Peanut Avoidance     5.9795
## 4    LEAP_105328 Peanut Consumption     7.0308
## 5    LEAP_106377   Peanut Avoidance     6.4066
```

The bracket notation after the dataset name implies location; the syntax `dataset[rows,columns]` specifies rows 1 through 5 and columns 1 through 6. To access only the first 5 rows, but all the columns, leave an empty space where the columns would usually be specified:

```
## 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
```

Alternatively, since there are 6 columns in the dataset, the command `LEAP[1:5,1:6]` would also achieve the same result. This is a common theme in R – there can be several ways to accomplish a desired result.

OI Biostat Table 1.1 shows the participant ID, treatment group, and overall outcome for five patients. It is not specified in the main text, but the data are specifically from rows 1, 2, 3, 529, and 530. The command `c()` can be used to bind the row numbers into a list, as well as to create a list of the desired columns. Columns can be referred to by name, instead of by number:

```
## OI Biostat 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
```

Two-way summary tables organize the data according to two variables and display the number of counts matching each combination of variable categories. The following code corresponds to *OI Biostat* Table 1.2, which groups participants into categories based on treatment group and overall outcome. In the `table()` command, the first variable specifies the rows and the second variable specifies the columns. The addition of the `addmargins()` command prints the sums of the rows and columns on the sides of the table.

```
## 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
```

1.2 Data Basics

Entire datasets can be partitioned into data frames using bracket notation. For example, *OI Biostat* Table 1.3 shows a data frame consisting of rows 1-3 and 150 (and all columns) from the `frog.altitude` dataset. The data frame can then be given a specific name, such as `frog.df`, and directly called on for later operations.

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

##     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
```

Similarly, matrix notation can be used to create *OI Biostat* Table 1.5.

```
## 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
```

1.3 Numerical Data

Numerical summaries can be quickly and easily calculated using R. The `summary()` command is one way to access several numerical summaries at once, including the minimum and maximum values of a variable:

```
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
```

1.3.1 Measures of center: mean and median

The **mean** of a numerical value is the sum of all observations divided by the number of observations, where $x_1, x_2, \ldots, x_n$ represent the n observed values:

$$\overline{x} = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

This calculation can be completed in R the same way as a hand calculation is done:

```
# identify n, the number of observations in the data
n = length(frog.altitude.data$clutch.volume)
n

## [1] 431

# calculate the sum of all observations and divide by n
sum(frog.altitude.data$clutch.volume)/n

## [1] 882.474
```

Alternatively, the R function `mean()` can be directly applied to the variable of interest:

```
x.bar = mean(frog.altitude.data$clutch.volume)
x.bar

## [1] 882.474

## round to the first decimal
round(x.bar, 1)

## [1] 882.5
```

To identify the **median** value, use the following command:

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

## [1] 831.7638
```

1.3.2 Measures of spread: standard deviation and variance

The distance of a single observation from the mean is its **deviation**. The following command produces the deviations for the 1st, 2nd, 3rd, and 431th observations in the `clutch.volume` variable.

```
frog.altitude.data$clutch.volume[c(1,2,3,431)]-x.bar

## [1] -704.64604 -625.43440 -731.11786   50.78032
```

The sample **standard deviation** is computed as the square root of the **variance**, which is the sum of squared deviations divided by the number of observations minus 1.

$$s = \sqrt{\frac{(x_1 - \overline{x})^2 + (x_2 - \overline{x})^2 + \cdots + (x_n - \overline{x})^2}{n-1}}$$

The following steps illustrate how to calculate the variance and standard deviation using the formula:

```
# calculate all deviations
dev = frog.altitude.data$clutch.volume - x.bar

# sum the squares of the deviations and divide by n - 1
var = (sum(dev^2))/(n-1)
var

## [1] 143680.9

# take the square root of the variance
sd = sqrt(var)
sd

## [1] 379.0527
```

Alternatively, use the R functions `var()` and `sd()`:

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

## [1] 143680.9

sd(frog.altitude.data$clutch.volume)

## [1] 379.0527
```

Variability can also be measured using the **interquartile range (IQR)**, which equals the third quartile (the 75th percentile) minus the first quartile (the 25th percentile).

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

## [1] 486.9009
```

1.3.3 Robust statistics

In the `frog.altitude` dataset, there are four extreme values for clutch volume that are larger than 2,000 mm^3. To illustrate how the summary statistics are influenced by extreme values, *OI Biostat* Table 1.15 shows summary statistics for the data without the four largest observations.

The subset of the data with values less than 2,000 mm^3 can be pulled out by first specifying a logical condition, which assigns `TRUE` or `FALSE` to each entry.

```
# logical condition
less.than.2000 = frog.altitude.data$clutch.volume<= 2000

# view the first 5 values
less.than.2000[1:5]

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

Bracket notation can then be used to calculate summary statistics specifically for the values in `clutch.volume` that satisfy the logical condition.

```
## robust estimates
median(frog.altitude.data$clutch.volume[less.than.2000])

## [1] 831.7638

IQR(frog.altitude.data$clutch.volume[less.than.2000])

## [1] 493.9186

## non-robust estimates
mean(frog.altitude.data$clutch.volume[less.than.2000])

## [1] 867.9425

sd(frog.altitude.data$clutch.volume[less.than.2000])

## [1] 349.1596
```

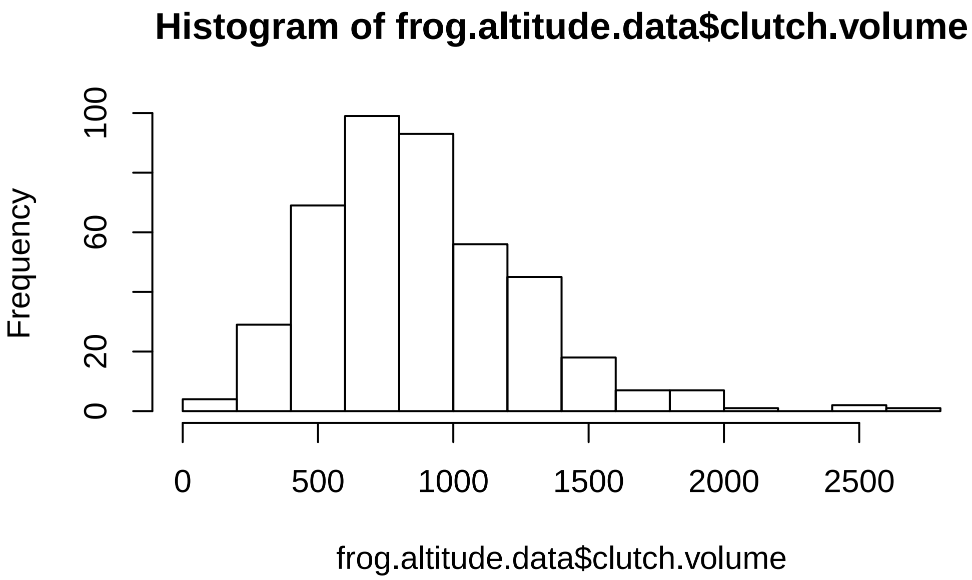
1.3.4 Visualizing distributions of data

While most numerical summaries can be calculated by hand, R is essential for creating graphical summaries.

Histograms

A **histogram** provides a view of data density. Observations are sorted into different bins based on their value, and the histogram shows the number of observations in each bin. The following command produces a histogram of the `clutch.volume` variable.

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



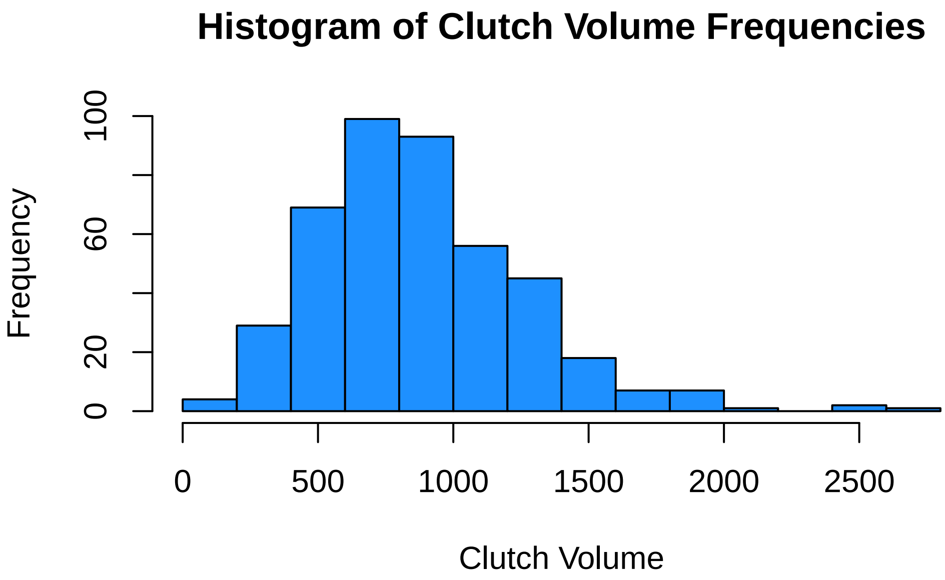
The `hist()` function takes several arguments:

- `x`: variable of interest
- `breaks`: number of bins
- `col`: color of the bars, enclosed in `""`
- `xlab`: x-axis label, enclosed in `""`
- `ylab`: y-axis label, enclosed in `""`
- `xlim`: range of values for the x-axis, in the form `c(lower bound, upper bound)`
- `ylim`: range of values for the y-axis, in the form `c(lower bound, upper bound)`
- `main`: main title of the plot, enclosed in `""`
- `plot`: if `TRUE` (default), a histogram is plotted; otherwise, data about the histogram is returned

As seen above, not all arguments must be specified; only the `x` argument is necessary. When options are not specified, R uses the default options, such as the default color of white for the histogram bars.

The following command reproduces *OI Biostat* Table 1.16 and Figure 1.17.

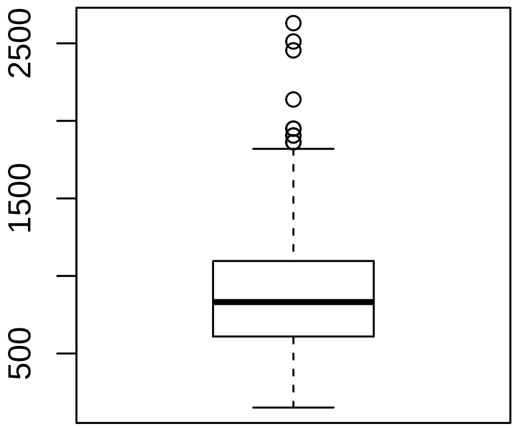
```
## Table 1.16
hist(frog.altitude.data$clutch.volume, breaks = 14, plot = FALSE)$counts

##  [1]  4 29 69 99 93 56 45 18  7  7  1  0  2  1

## Figure 1.17
hist(x = frog.altitude.data$clutch.volume, breaks = 14, col = "dodgerblue",
     xlab = "Clutch Volume", ylab = "Frequency", ylim = c(0, 100),
     main = "Histogram of Clutch Volume Frequencies")
```



Boxplots

A **boxplot** uses five statistics to summarize a dataset, in addition to showing unusual observations. The following command produces a boxplot of the `clutch.volume` variable.

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

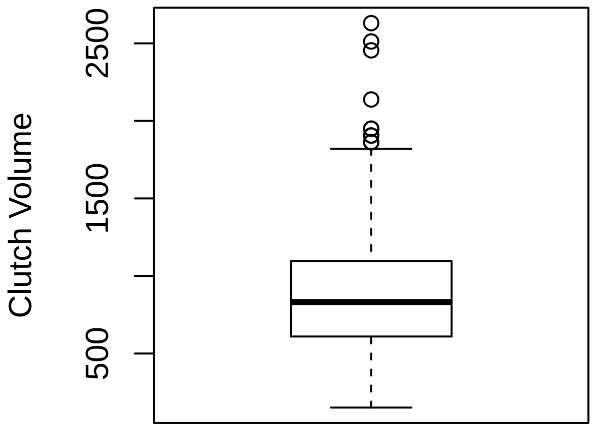


Like the hist() function, the boxplot() function also takes several arguments that allow for certain parameters to be specified:

- x: variable of interest
- axes: if TRUE, numbers are shown on the axes
- col: color of the boxplot, enclosed in ""
- xlab: x-axis label, enclosed in ""
- ylab: y-axis label, enclosed in ""
- xlim: range of values for the x-axis, in the form c(lower bound, upper bound)
- ylim: range of values for the y-axis, in the form c(lower bound, upper bound)
- main: main title of the plot, enclosed in ""

The following command reproduces a simplified version of *OI Biostat* Figure 1.19.

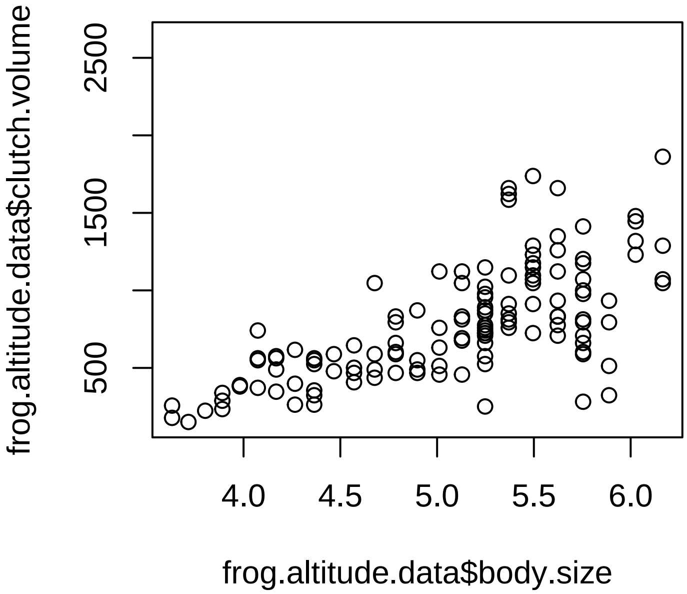
```r
## Figure 1.19
boxplot(x = frog.altitude.data$clutch.volume, ylab = 'Clutch Volume', axes = TRUE,
        ylim = range(frog.altitude.data$clutch.volume))
```



1.3.5 Scatterplots and correlation

Scatterplots can be used to visualize the relationship between two numerical variables. In the `plot()` command, either a comma or a tilde can be used between the variable names; i.e., `plot(x,y)` versus `plot(y ~ x)`.

```
plot(frog.altitude.data$body.size, frog.altitude.data$clutch.volume)
```

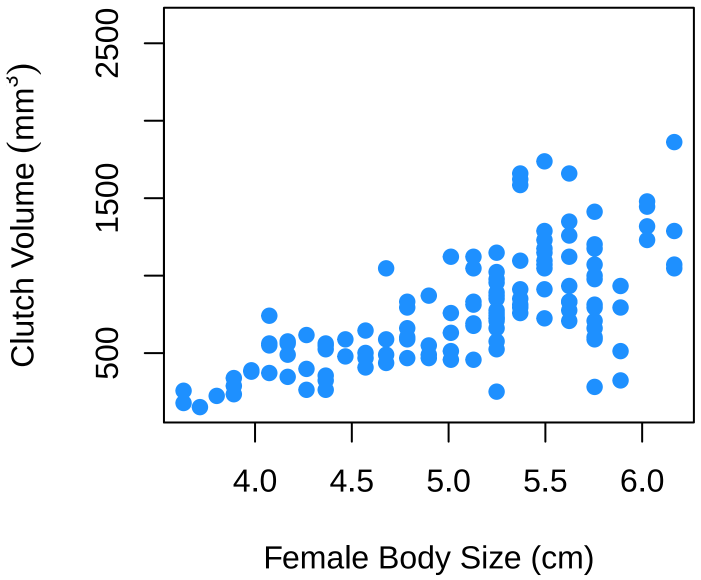


The `plot()` command takes the following arguments:

- `x`: variable defining the *x*-coordinates
- `y`: variable defining the *y*-coordinates
- `col`: color of the dots, enclosed in `""`
- `type`: by default, data points are marked with dots; other options include `"l"` which draws a line chart, or `"b"` which includes both dots and lines
- `xlab`: *x*-axis label, enclosed in `""`
- `ylab`: *y*-axis label, enclosed in `""`
- `xlim`: range of values for the *x*-axis, in the form `c(lower bound, upper bound)`
- `ylim`: range of values for the *y*-axis, in the form `c(lower bound, upper bound)`
- `main`: main title of the plot, enclosed in `""`

The following command reproduces a simplified version of *OI Biostat* Figure 1.20. The additional argument pch changes the appearance of the dots to filled dots, which is specified by `19`.

```
## Figure 1.20
plot(frog.altitude.data$clutch.volume~frog.altitude.data$body.size, col = "dodgerblue",
     pch = 19, xlab = "Female Body Size (cm)", ylab = expression("Clutch Volume" ~ (mm^3)))
```



Simplified versions of *OI Biostat* Figures 1.21, 1.22, and 1.23 can also be reproduced using `plot()`.

1.3.6 Correlation

Correlation is a numerical measure of the strength of a linear relationship. The formula for correlation uses the pairing of the two variables, just as the scatterplot does in a graph, so the data used in calculating correlation is a set of n ordered pairs $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$.

The correlation between two variables x and y is given by:

$$r = \frac{1}{n-1} \sum_{i=1}^{n} \left(\frac{x_i - \overline{x}}{s_x} \right) \left(\frac{y_i - \overline{y}}{s_y} \right)$$

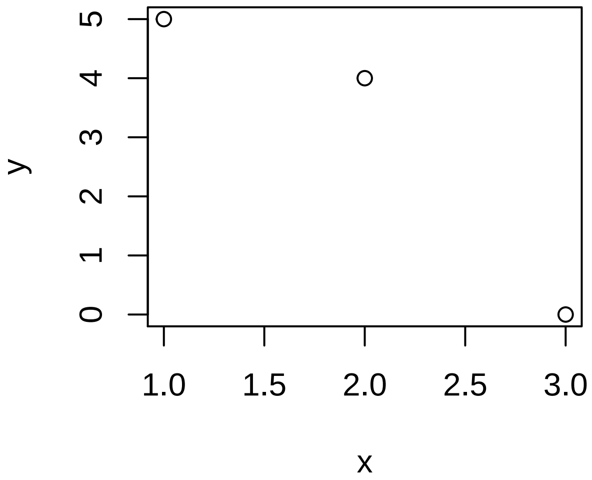
where $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ are the n paired values of x and y, and s_x and s_y are the sample standard deviations of the x and y variables, respectively.

The following illustrates the calculations for finding the correlation coefficient of (1,5), (2, 4), and (3,0), as shown in *OI Biostat* Example 1.13.

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

# calculate sample means
x.bar = mean(x)
y.bar = mean(y)

# calculate sample sd's
sd.x = sd(x)
sd.y = sd(y)

# calculate products and sum of products
x.component = (x - x.bar)/(sd.x)
y.component = (y - y.bar)/(sd.y)
products = x.component * y.component
products.sum = sum(products)

# divide by n - 1
n = length(x)   ## n also equals length(y)
r = products.sum / (n - 1)
r

## [1] -0.9449112
```

It is much easier to use the `cor()` function to find correlation.

```
## Figure 1.22
cor(x,y)

## [1] -0.9449112

plot(x,y)
```



The correlation can also be calculated for the income versus life expectancy data plotted in *OI Biostat* Figure 1.23. In this case, the command includes the argument `use = "complete.obs"` to allow for the computation to disregard any missing values in the dataset.

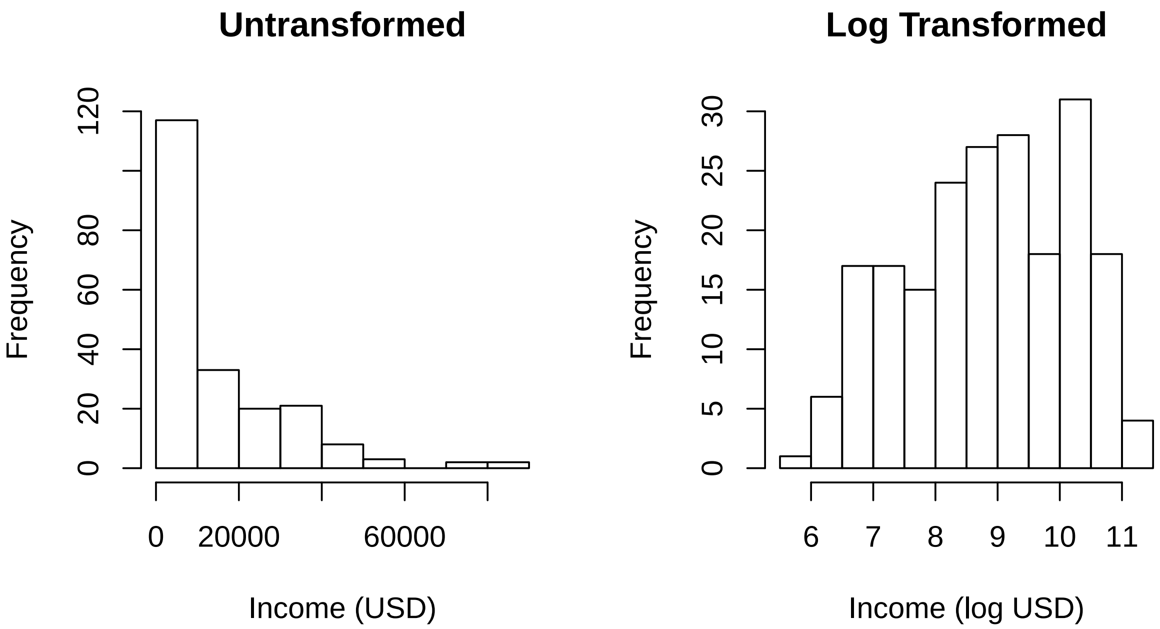
```
cor(life.expectancy.income$income, life.expectancy.income$life.expectancy,
    use = "complete.obs")

## [1] 0.6308783
```

1.3.7 Transforming Data

A **transformation** is a rescaling of data using a function. The figure below shows the original plot of income data as well as the plot of the log-transformed data. Note that the log command in R computes natural logarithms.

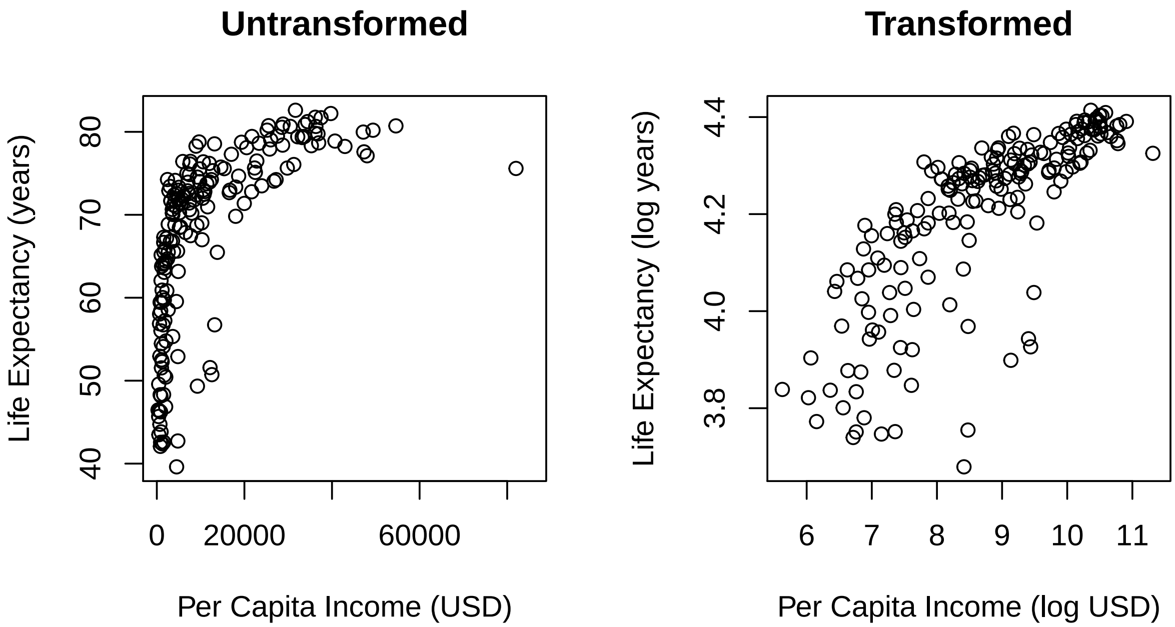
The function par() creates partitions in the graphing output. In this case, specifying mfrow = c(1,2) produces 1 row and 2 columns.

```r
## Figure 1.26
par(mfrow = c(1, 2))  ## this line 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")
```



Transformations can also be applied to both variables when exploring a relationship. The following figure shows simplified versions of *OI Biostat* Figures 1.23 and 1.27.

```r
## Figure 1.23 and Figure 1.27
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")
```



1.4 Categorical Data

1.4.1 Contingency Tables

A **frequency table** shows the counts for each category within a variable. The following `table()` command produces a frequency table for the `actn3.r577x` variable.

```
## Table 1.28
addmargins(table(famuss$actn3.r577x))

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

A **contingency table** summarizes data for two categorical variables, such as race and genotype in *OI Biostat* Table 1.29.

```
## Table 1.29
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
```

OI Biostat Table 1.30 shows a contingency table with row proportions, computed as the counts divided by their row totals. The `prop.table()` command produces a table with row proportions, as specified by the 1 in the argument.

```
## Table 1.30
counts.table = table(famuss$race, famuss$actn3.r577x)
row.prop.table = prop.table(counts.table, 1)
row.prop.table

##
##                     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
```

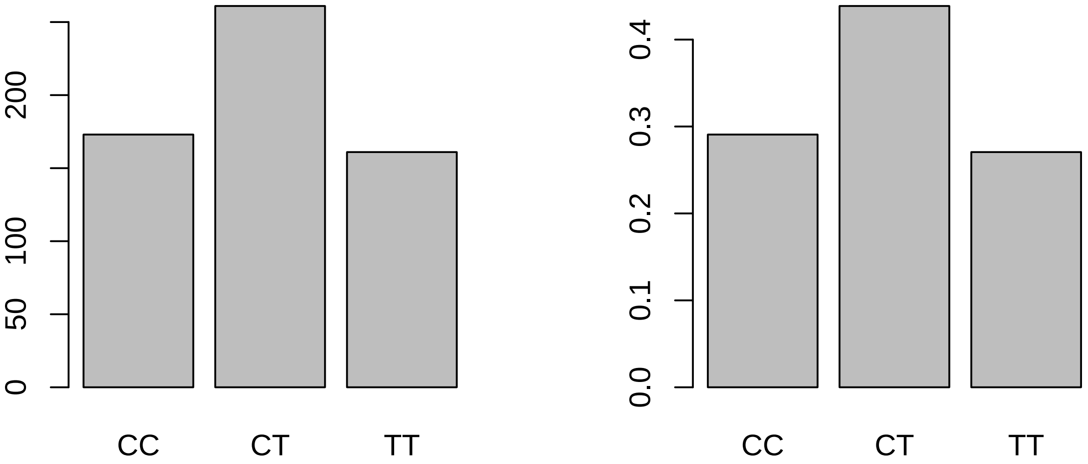
Alternatively, a contingency table can be created with column proportions by changing the 1 in `prop.table()` to a 2.

```
## Table 1.31
counts.table = table(famuss$race, famuss$actn3.r577x)
col.prop.table = prop.table(counts.table, 2)
col.prop.table

##
##                      CC         CT         TT
##   African Am 0.09248555 0.02298851 0.03105590
##   Asian      0.12138728 0.06896552 0.09937888
##   Caucasian  0.72254335 0.82758621 0.78260870
##   Hispanic   0.02312139 0.03831418 0.05590062
##   Other      0.04046243 0.04214559 0.03105590
```

1.4.2 Bar Plots

A **bar plot** is a common way to display a single categorical variable, either from count data or from proportions. The `barplot()` command requires values to be input in a table format. In the below example, the `table()` command is nested within the `barplot()` command.

```
## Figure 1.32
par(mfrow = c(1, 2))
barplot(table(famuss$actn3.r577x))  ## count barplot
barplot(table(famuss$actn3.r577x)/sum(table(famuss$actn3.r577x)))  ## frequency barplot
```

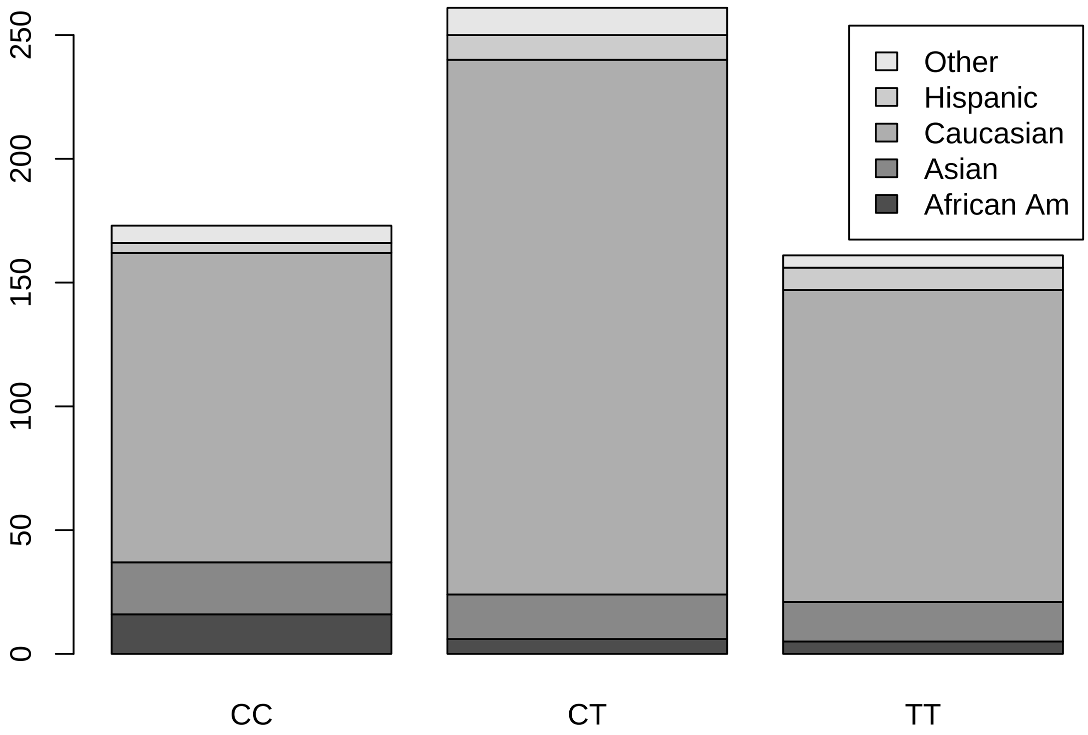

0
50
100
200
CC
CT
TT
0.0
0.1
0.2
0.3
0.4
CC
CT
TT

1.4.3 Segmented Bar Plot

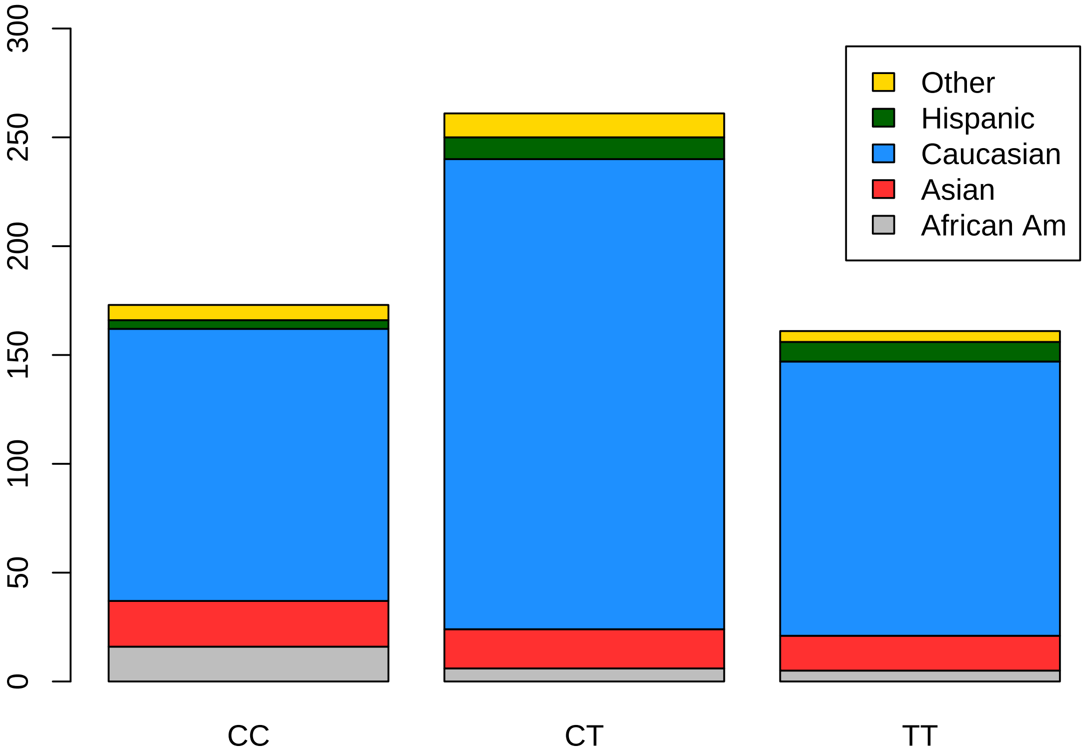
A **segmented bar plot** draws from a contingency table to display information about two categorical variables. The following code reproduces *OI Biostat* Figure 1.33a, in which a bar plot was created using the actn3.r577x variable, with each group divided by the levels of race. The simplified version of the plot uses the default greyscale shading; it is also possible to specify a list of colors using c().

The argument legend can be used to specify whether the row names or the column names are used for the legend.

```
## Figure 1.31a
counts.table = (table(famuss$race, famuss$actn3.r577x))
barplot(counts.table, legend = rownames(counts.table))
```

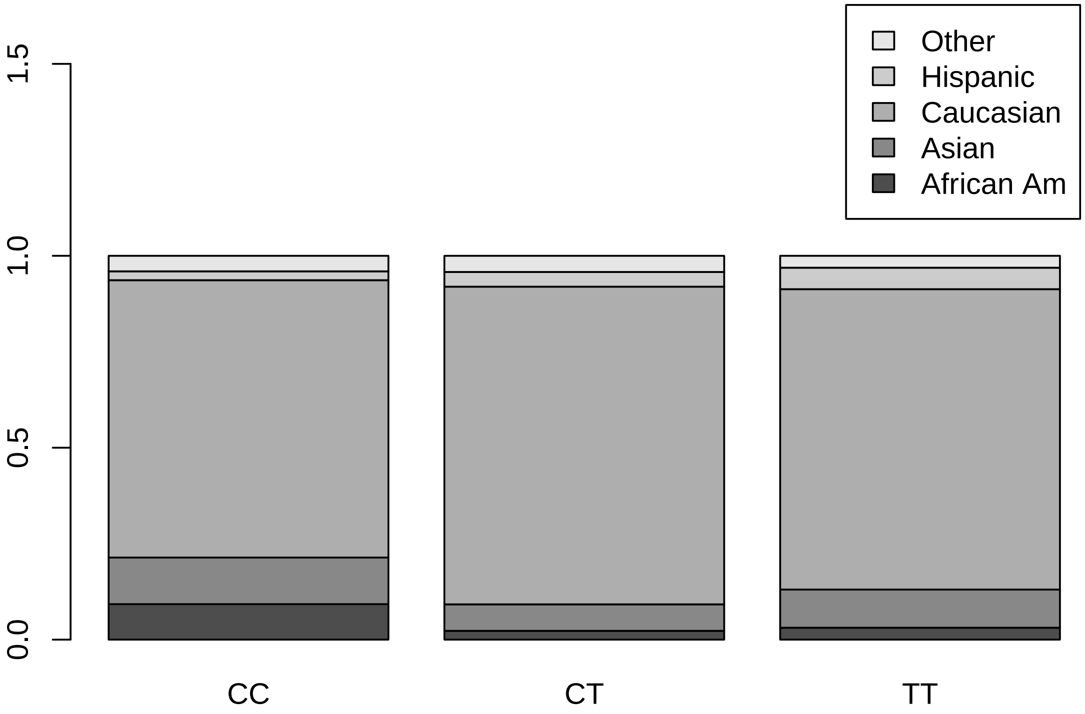


```
barplot(counts.table, col = c("gray", "firebrick1", "dodgerblue",
    "darkgreen", "gold"), ylim = c(0, 300), legend = rownames(counts.table))
```

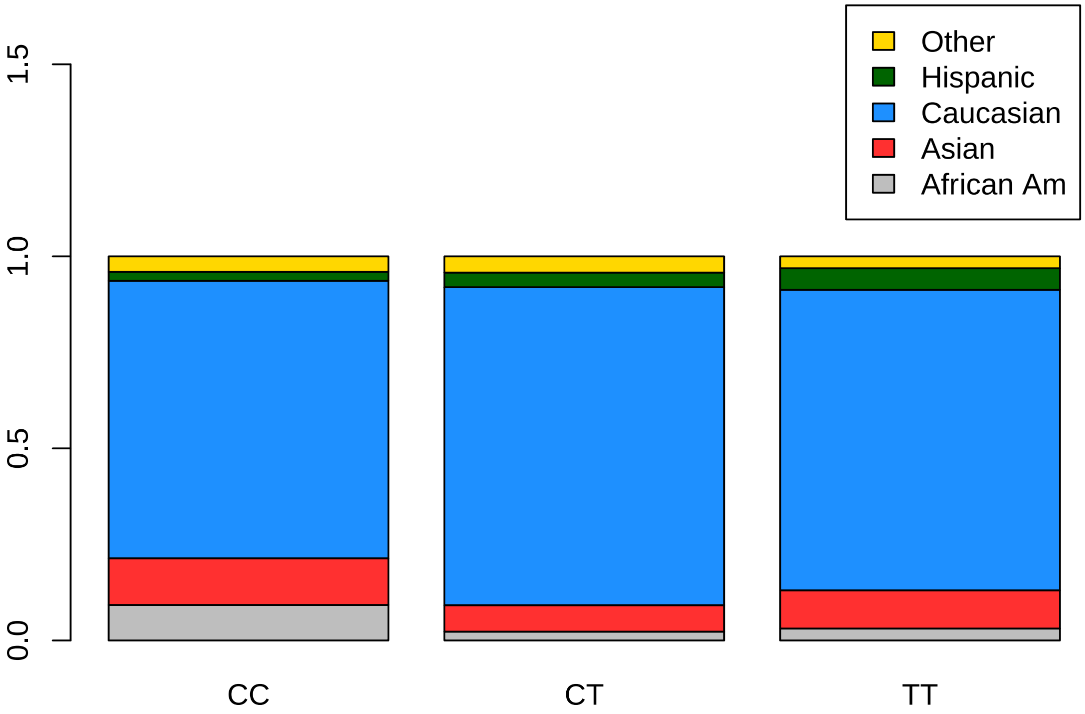

300
250
200
150
100
50
0
Other
Hispanic
Caucasian
Asian
African Am
CC
CT
TT

A **standardized segmented bar plot** uses proportions to scale the data, and draws from a contingency table with proportions. Setting `ylim` from 0 to 1.7 allows for enough empty space on the plot so that the legend does not overlap the bars.

```
## Figure 1.33b
counts.table = (table(famuss$race, famuss$actn3.r577x))
row.prop.table = prop.table(counts.table, 2)
barplot(row.prop.table, ylim = c(0, 1.7), legend = rownames(counts.table))
```

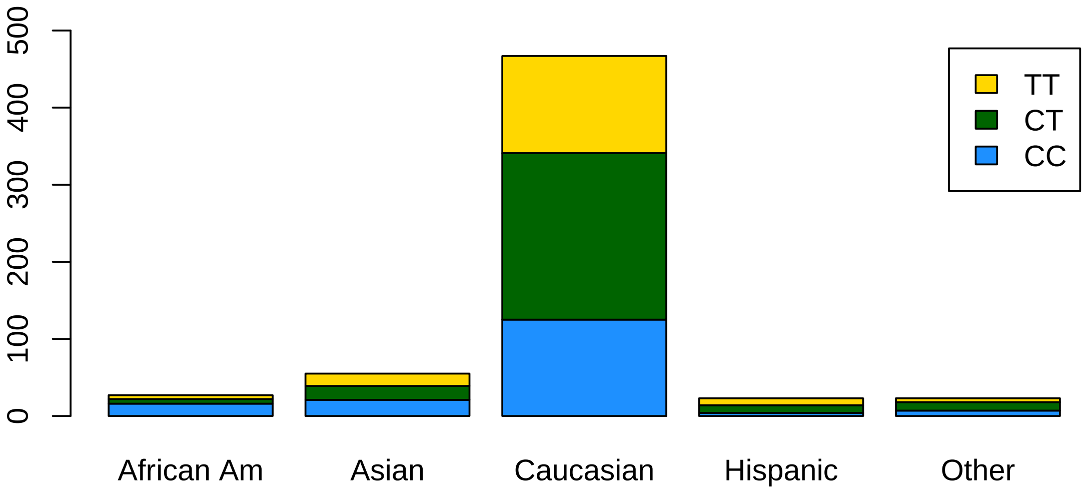


```
barplot(row.prop.table, col = c("gray", "firebrick1", "dodgerblue",
    "darkgreen", "gold"), ylim = c(0, 1.7), legend = rownames(counts.table))
```

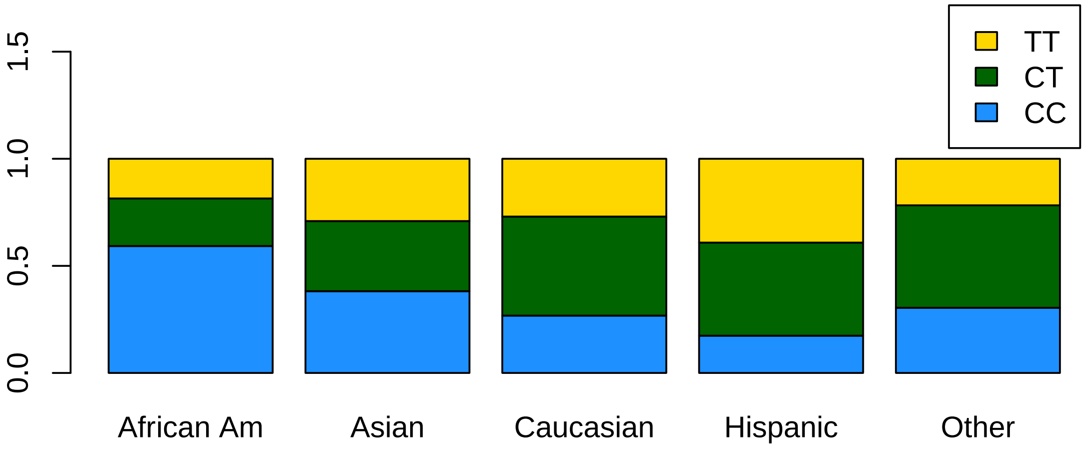


In *OI Biostat* Figure 1.34, the data from the contingency table are organized differently, with each bar representing a level of race. To make this change, reverse the order of the variables in `counts.table`

```
## Figure 1.34a
counts.table = (table(famuss$actn3.r577x, famuss$race))  ## change variable order

barplot(counts.table, col = c("dodgerblue", "darkgreen", "gold"),
    ylim = c(0, 500), legend = rownames(counts.table))
```



```r
## Figure 1.34b
row.prop.table = prop.table(counts.table, 2)
barplot(row.prop.table, col = c("dodgerblue", "darkgreen", "gold"),
    ylim = c(0, 1.8), legend = rownames(counts.table))
```

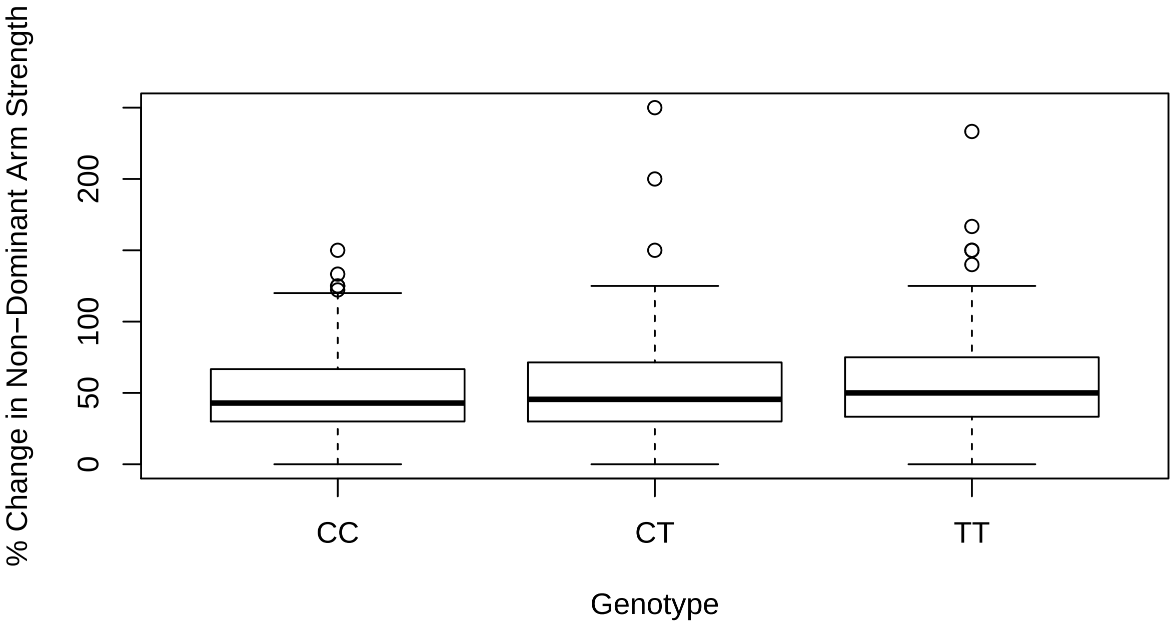


1.4.4 Comparing numerical data across groups

Side-by-side boxplots

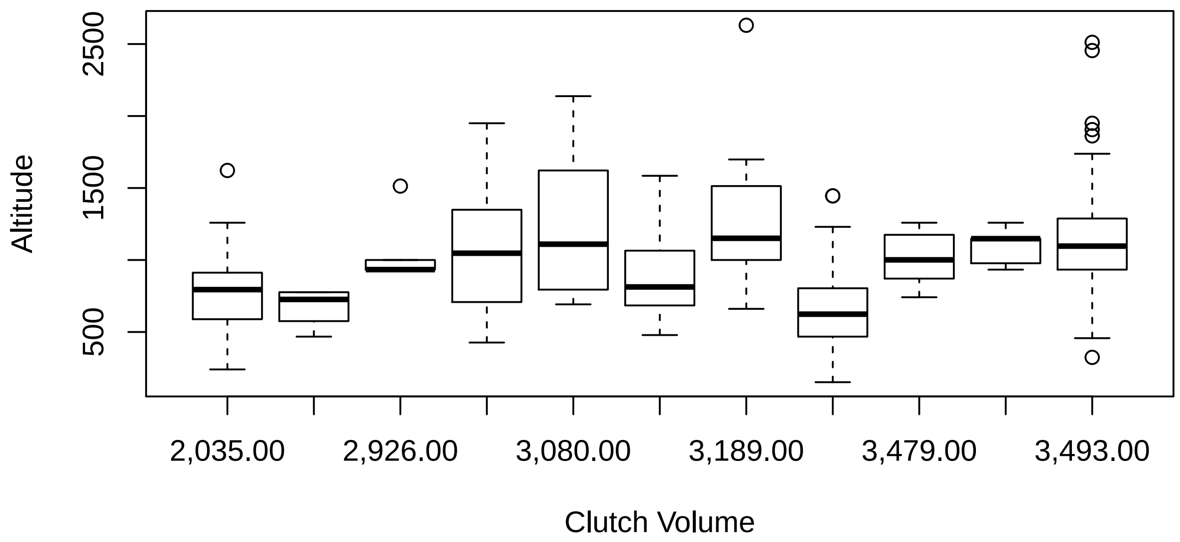
The **side-by-side** boxplot is a useful tool for comparing the distribution of numerical data across categories. *OI Biostat* Figure 1.35a shows a side-by-side boxplot for percent change in non-dominant arm strength grouped by genotype. The response variable y comes before the tilde and the explanatory variable x; in this case, the response variable is percent change in strength.

```
## Figure 1.335(a)
boxplot(famuss$ndrm.ch ~ famuss$actn3.r577x,
    ylab = "% Change in Non-Dominant Arm Strength",
    xlab = "Genotype")
```



OI Biostat Figure 1.36 shows a larger side-by-side boxplot which compares the distribution of frog clutch sizes for different altitudes.

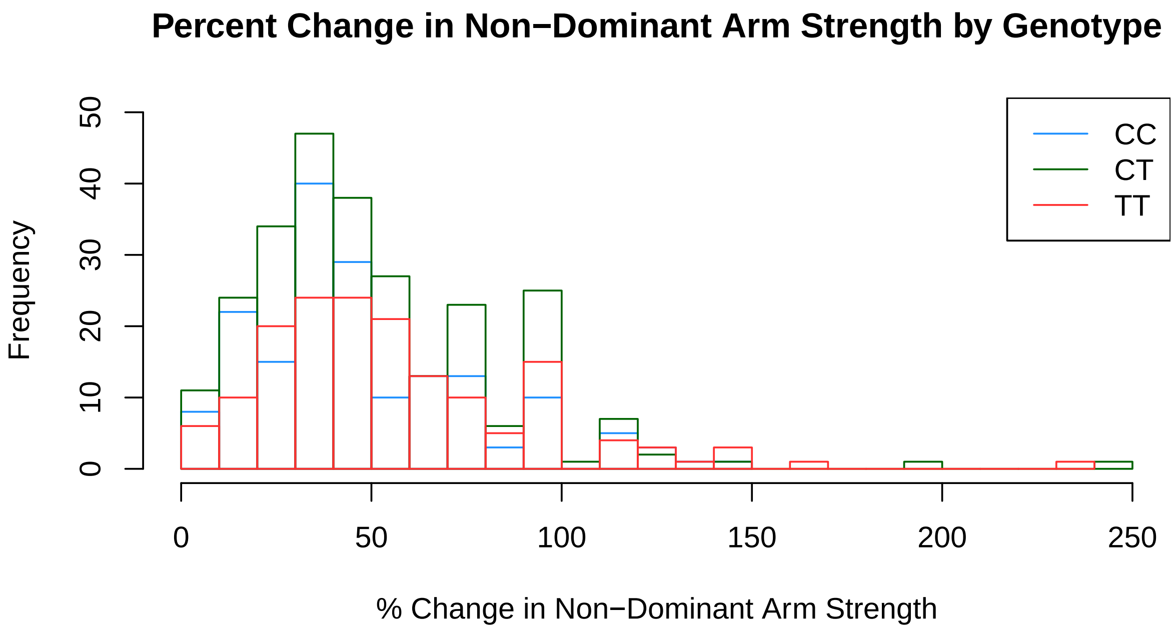
```
## Figure 1.36
boxplot(frog.altitude.data$clutch.volume ~ frog.altitude.data$altitude,
    xlab = "Clutch Volume", ylab = "Altitude")
```



Hollow Histograms

A **hollow histogram** plots the outlines of histograms for each group onto the same axes. To plot a hollow histogram, specify any axis limits and labels in the command for the first histogram; the subsequent histograms require the argument `add = T` in order for them to be overlaid on top of the first plot.

```
## Figure 1.35(b)
hist(famuss$ndrm.ch[famuss$actn3.r577x == "CC"], breaks = 20, border = "dodgerblue",
     xlab = "% Change in Non-Dominant Arm Strength", ylim = c(0,50), xlim = c(0,250),
     main = "Percent Change in Non-Dominant Arm Strength by Genotype")
hist(famuss$ndrm.ch[famuss$actn3.r577x == "CT"], breaks = 20, border = "darkgreen",
     add = T)
hist(famuss$ndrm.ch[famuss$actn3.r577x == "TT"], breaks = 20, border = "firebrick1",
     add = T)

legend('topright',
       col = c("dodgerblue", "darkgreen", "firebrick1"),
       lwd = c(1, 1, 1),      ## line width
       legend = c('CC', 'CT', 'TT'))    ##legend labels
```



1.5 Genomic Data

```
boxplot(golub.exprs.pheno[, 7:9])

## Error in boxplot(golub.exprs.pheno[, 7:9]): object 'golub.exprs.pheno' not found
```

Chapter 2

Using R for Probabilities

Contents

2.1 **Bayes' Theorem** 28
2.2 **Contingency Table** 29
2.3 **Simulation** 30

This chapter focuses on the material covered in *OI Biostat* Chapter 2, Section 2.5. While simple conditional probability problems are easily solved by hand with a basic calculator, R can be useful for more elaborate scenarios like those that involve Bayes' Theorem.

What are the chances that a woman with a positive mamogram has breast cancer? This question can be rephrased as the conditional probability that a woman has breast cancer, given that her mammogram is abnormal, otherwise known as the **positive predictive value** of a mammogram. Two methods discussed in this chapter, using Bayes' Theorem and creating a contingency table, are also explained in *OI Biostat*. This chapter will also cover an additional approach: modeling the problem scenario by running a simulation.

> ***OI Biostat* Example 2.37.** In Canada, about 0.35% of women over 40 will develop breast cancer in any given year. A common screening test for cancer is the mammogram, but it is not perfect. In about 11% of patients with breast cancer, the test gives a **false negative**: it indicates a woman does not have breast cancer when she does have breast cancer. Similarly, the test gives a **false positive** in 7% of patients who do not have breast cancer: it indicates these patients have breast cancer when they actually do not. If a randomly selected woman over 40 is tested for breast cancer using a mammogram and the test is positive – that is, the test suggests the woman has cancer – what is the probability she has breast cancer?

2.1 Bayes' Theorem

Bayes' Theorem states that the conditional probability $P(A_1|B)$ can be identified as the following fraction:

$$\frac{P(A_1 \text{and } B)}{P(B)} = \frac{P(B|A_1)P(A_1)}{P(B|A_1)P(A_1) + P(B|A_2)P(A_2) + \cdots + P(B|A_k)P(A_k)}$$

where A_2, A_3, ..., and A_k represent all other possible outcomes of the first variable.

The expression can also be written in terms of diagnostic testing language, where D = {has disease}, D^c = {does not have disease}, T^+ = {positive test result}, and T^- = {negative test result}.

$$
\begin{aligned}
P(D|T^{+}) &= \frac{P(D \text{ and } T^{+})}{P(D)} \\
&= \frac{P(T^{+}|D) \times P(D)}{[P(T^{+}|D) \times P(D)] + [P(T^{+}|D^{c}) \times P(D^{c})]} \\
\text{PPV} &= \frac{\text{sensitivity } \times \text{ prevalence}}{[\text{sensitivity } \times \text{ prevalence}] + [(1 \text{ - specificity}) \times (1 \text{ - prevalence})]}
\end{aligned}
$$

R can be used to store values for prevalence, sensitivity, and specificity so that calculations are less tedious. Recall that the **sensitivity** is the probability of a positive test result when disease is present, which is the complement of a false negative. The **specificity** is the probability of a negative test result when disease is absent, which is the complement of a false positive.

```
prevalence = 0.0035
sensitivity = 1 - 0.11
specificity = 1 - 0.07

ppv.num = (sensitivity*prevalence)  ## numerator
ppv.den = ppv.num + ((1-specificity)*(1-prevalence))  ## denominator
ppv = ppv.num / ppv.den

ppv

## [1] 0.04274736
```

2.2 Contingency Table

The PPV can also be calculated by constructing a two-way contingency table for a hypothetical population and calculating conditional probabilities by conditioning on rows or columns. While this method results in an estimate of PPV, using a large enough population size such as 100,000 produces an empirical estimate that is very close to the exact value found through using Bayes' Theorem.

	D+	D-	Total
T+			
T-			
Total			100,000

First, calculate the expected number of disease cases and non-disease cases in the population:

```
population.size = 100000
expected.cases = prevalence * population.size
expected.cases

## [1] 350

expected.noncases = (1 - prevalence) * population.size
expected.noncases

## [1] 99650
```

	D+	D-	Total
T+			
T-			
Total	350	99,650	100,000

Next, calculate the expected number of cases of true positives and the expected number of cases of false positives:

```
expected.true.positives = expected.cases * sensitivity
expected.true.positives

## [1] 311.5

expected.false.positives = expected.noncases * (1 - specificity)
expected.false.positives

## [1] 6975.5

total.expected.positives = expected.true.positives + expected.false.positives
total.expected.positives

## [1] 7287
```

	D+	D-	Total
T+	311.5	6,975.5	7,287
T-			
Total	350	99,650	100,000

Finally, calculate the positive predictive value:

```
ppv = expected.true.positives/total.expected.positives
ppv

## [1] 0.04274736
```

2.3 Simulation

R can be used to simulate a population of 100,000 individuals that fits the parameters specified by the problem, i.e., a population where 0.35% of women have breast cancer. Afterwards, using the known specificity and sensitivity of the diagnostic test, individuals can be assigned a test result of either positve or negative. This results in a simulated dataset of 100,000 individuals that each have a disease status and test result.

1. Define the parameters of the simulation: disease prevalence, test sensitivity, test specificity, and hypothetical population size.

2. In order for the results of the simulation to be reproducible, it is necessary to use `set.seed()` to associate the particular set of results with a "seed". Any integer can be used as the seed. Different seeds will produce a different set of results.

3. Using the `vector()` command, create two empty lists that will be used to store the results of the simulation: one will store disease status and the other will store test outcome. The results will be in the form of numbers; specifically, either `0` or `1`.

4. Assign disease status by using the `sample()` command, which takes a sample of a specified size from a list `x`. In this context, the goal is to sample between `0` and `1` 100,000 times, where `0` represents an individual without breast cancer and `1` an individual with breast cancer. The argument `prob` defines the probability that either number is sampled; a `1` should be sampled with the same probability as the prevalence, since the prevalence indicates how many members of the population have disease. Similarly, a `0` should be sampled with probability `(1 - prevalence)`. The argument `replace = TRUE` allows for sampling with replacement; in other words, allowing the numbers `0` and `1` to be assigned multiple times.

5. Assign test result by using the `sample()` command in combination with a `for` loop and conditional statements. In this step, a test result is assigned with different probability depending on whether the disease status is a `0` or a `1`; this relates to the sensitivity and specificity of the diagnostic test. The loop allows for this process to be repeated for each of the 100,000 individuals being simulated.

   - The first line sets up the loop, with a variable `ii` that starts at 1 and finishes at `population.size`, which was specified earlier as 100,000. This allows for each test result to be assigned one at a time.
   - Two `if` statements are in the loop which direct R to take a different action depending on the value of `disease.status`. The double equals signs == imply a conditional statement, allowing `if(disease.status[ii] == 1)` to make the statement "For this loop, if disease status is equal to 1, then do the following...".
   - Depending on disease status, R will execute one of the two `sample()` functions in the loop. One sample function has probabilities weighted based on sensitivity and the other has probabilities defined by specificity.

6. Make calculations using the two lists, `disease.status` and `disease.outcome`, which are now filled with values. Since the value 1 was assigned to a positive test result and to an individual with disease, the `sum()` command can be used to determine the total number of individuals with disease and the number of positive test results.

```
## 1. set parameters
prevalence = 0.0035
sensitivity = 1 - 0.11
specificity = 1 - 0.07
population.size = 100000

## 2. set seed
set.seed(2016)

## 3. create empty lists
disease.status = vector(”numeric”, population.size)
test.outcome = vector(”numeric”, population.size)

## 4. assign disease status
disease.status = sample(x = c(0,1), size = population.size,
                        prob = c(1 - prevalence, prevalence),  ## matches order of x
                        replace = TRUE)

## 5. assign test result
for (ii in 1:population.size) {
  if (disease.status[ii] == 1) {  ## note the ==
    test.outcome[ii] = sample(c(0, 1), size = 1, ## test results assigned 1 at a time
                              prob = c(1 - sensitivity, sensitivity))}
  if (disease.status[ii] == 0) {  ## note the ==
    test.outcome[ii] = sample(c(0, 1), size = 1, ## test results assigned 1 at a time
                              prob = c(specificity, 1 - specificity))}

}

## 6. calculate ppv
num.disease = sum(disease.status)  ## total number of individuals with disease
num.pos.test = sum(test.outcome)   ## total number of positive tests

# identify the number of individuals with disease who tested positive
num.true.pos = sum(test.outcome[disease.status == 1])

# calculate ppv
ppv = num.true.pos / num.pos.test
ppv

## [1] 0.04273973
```

Chapter 3

Distributions

Contents

3.1 Normal Distribution . . . 33

3.1.1 Direct Plot . . . 33

3.1.2 Through Simulation . . . 34

3.1.3 A Comparison . . . 35

3.1.4 Probability Functions . . . 35

3.2 Binomial Distribution . . . 37

3.2.1 Accessing the Binomial Distribution . . . 37

3.2.2 Comparison of the Methods . . . 38

3.3 Continuous and Discrete Distributions . . . 38

3.4 Other Distributions . . . 39

3.4.1 Poisson Distribution . . . 39

3.4.2 Geometric Distribution . . . 39

3.4.3 Negative Binomial Distribution . . . 40

Chapter 3 presents the idea of several known and commonly used distributions. R is a powerful source for accessing and using these distributions.

3.1 Normal Distribution

We have learned that the **normal distribution** can be written as

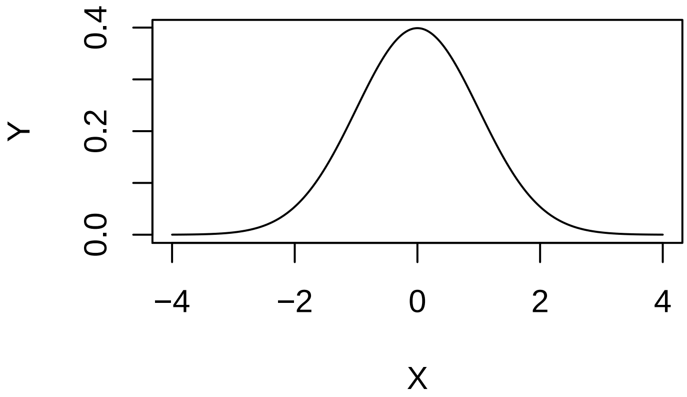
$$N(\mu, \sigma)$$

where μ is the mean of the distribution and σ is the standard deviation. Using this idea, we can create a standard normal distribution in one of two ways:

3.1.1 Direct Plot

We can use a function in R called the *dnorm* function which for a given posible value of the distribution, returns the probability of the distribution holding that value.

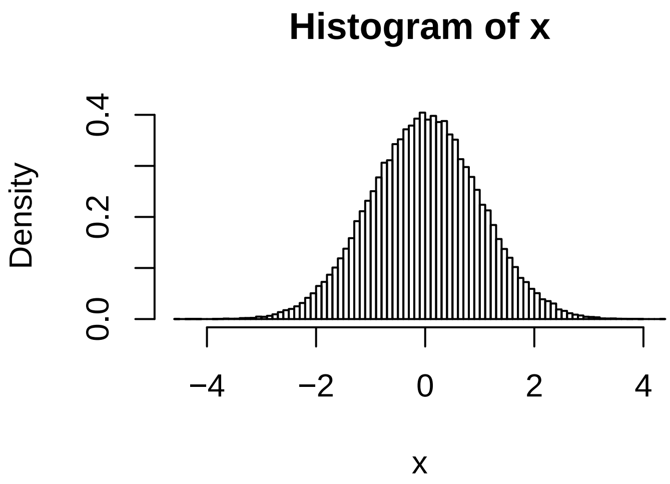
```
## getting a list of values to evaluate distribution
X = seq(-4, 4, 0.01)
## getting normal distribution value of the given X values
Y = dnorm(X, mean = 0, sd = 1)
## plotting results
plot(X,Y, type = "l")
```



3.1.2 Through Simulation

The second method involves sampling randomly from the known normal distribution to simulate the probabilities that we drew directly in the above method. In this case, we use the function *rnorm*, which takes *n* number of random samples from the normal distribution with the given mean and standard deviation. In the *hist* command, we include the specification that *freq = FALSE* to make the plot reflect percentages rather than frequency counts.

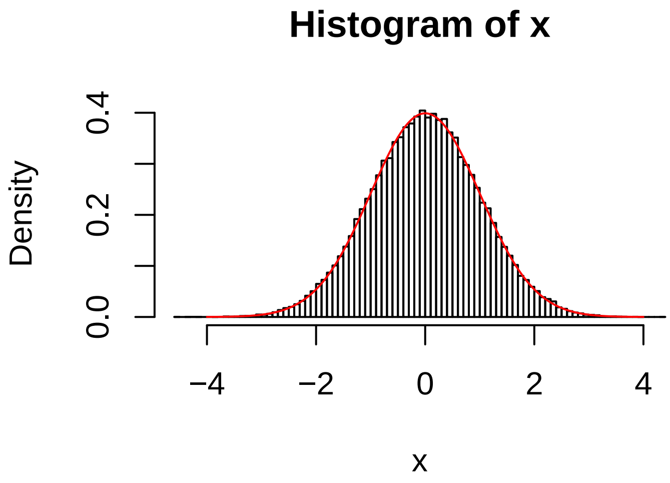
```
set.seed(1)
x <- rnorm(n = 100000, mean = 0, sd = 1)
hist(x, breaks = 100, freq = FALSE)
```



3.1.3 A Comparison

To see how comparable the two methods are, we can plot them on the same graph as follows.

```
hist(x, freq = FALSE, breaks = 100)
lines(X,Y, col = "red")
```



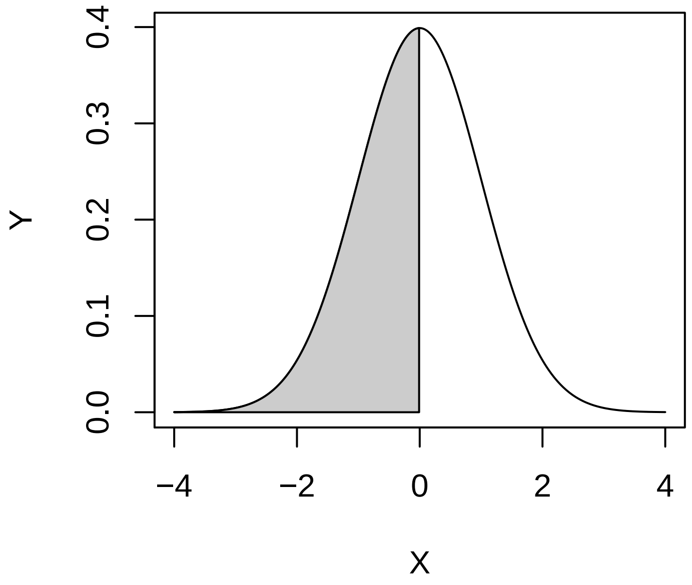
Note that the two show very similar results. For practical purposes, we typically use *dnorm* as it only requires one command and is the asymptotic result of simulations, rather than an approximation.

3.1.4 Probability Functions

For the normal distribution, there are two more functions that are very useful, *pnorm* and *qnorm*.

pnorm

The function *pnorm* tells you the percentage of the normal distribution that is less than or equal to the value you put in. This corresponds to the gray area in the below plot and can be written as $pnorm(x, mean = \mu, sd = \sigma)$ which is equivalent to $P(X < x)$ where $X \sim N(\mu, \sigma)$ and x is the point of interest.



For some example scenarios, where we want to find a proportion of the normal distribution given x or z, we can use the following configurations of *pnorm()*:

- $pnorm(z) = P(Z \leq z)$
- $pnorm(z, lower.tail = FALSE) = P(Z > z)$
- $pnorm(x, \mu, \sigma) = P(X \leq x)$ where $X = \sigma Z + \mu$
- $pnorm(x, \mu, \sigma, lower.tail = FALSE) = P(X > x)$ where $X = \sigma Z + \mu$

qnorm

On the other hand, the *qnorm* function is the inverse function, instead giving the X value such that the given percentage of the distribution is less than or equal to that value. This function takes a known probability and returns a value on the normal distribution that is corresponding.

Some sample configurations of using *qnorm()* can be as follows,

- $qnorm(p) = P(Z \leq z)$
- $qnorm(p, lower.tail = FALSE) = P(Z > z)$
- $qnorm(p, \mu, \sigma) = P(X \leq x)$ where $X = \sigma Z + \mu$
- $qnorm(p, \mu, \sigma, lower.tail = FALSE) = P(X > x)$ where $X = \sigma Z + \mu$

3.2 Binomial Distribution

We can write a random variable which follows a binomial distribution as

$$X \sim Binom(n, p)$$

where n represents the sample size and p represents the probability of success.

The functions we saw above for the normal distribution can be similarly used for the **binomial distribution** as follows where the necessary arguments are *size* and *prob*, corresponding respectively to n and p in our above representation of the binomial.

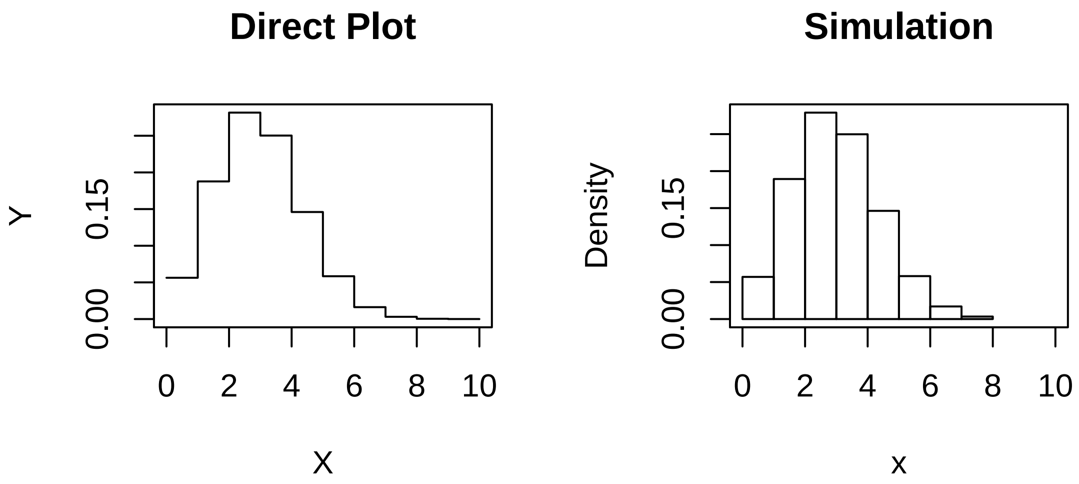
- *dbinom*
- *rbinom*
- *pbinom*
- *qbinom*

Similar to the above example with the normal distribution, we can use these functions in multiple ways to work with the binomial distribution.

3.2.1 Accessing the Binomial Distribution

As we did above, we can directly access the binomial distribution using the *dbinom* function as follows:

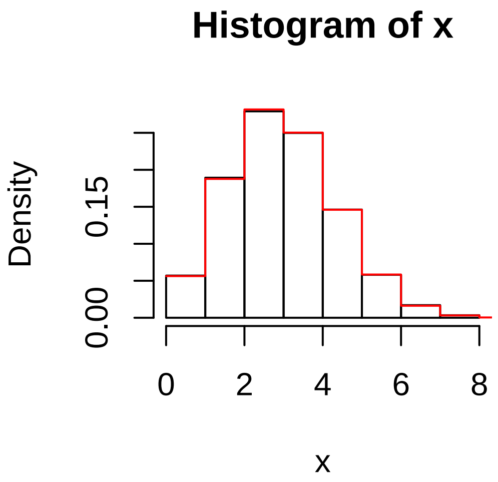
```
## Splitting the graphics window into two panes
par(mfrow=c(1,2))
## Directly plotting the binomial distribution
 # getting a list of values to evaluate distribution
X = seq(0, 10, 1)
 # getting normal distribution value of the given X values
Y = dbinom(X, size = 10, prob = .25)
 # plotting results
plot(X,Y, type = "s", main = "Direct Plot", xlim = c(0,10))

## Instead Using Simulation
set.seed(1)
x <- rbinom(n = 100000, size = 10, prob = .25)
hist(x, breaks = 10, freq = FALSE,right = FALSE, main = "Simulation", xlim = c(0,10))
box()
```



3.2.2 Comparison of the Methods

Instead of side by side, we can superimpose the two graphs for direct comparison.

```
hist(x, breaks = 10, freq = FALSE,right = FALSE)
lines(X,Y, type = "s", col = "red")
```



3.3 Continuous and Discrete Distributions

Using the binomial distribution highlights an important feature of distributions that we must consider: whether the random variable is **discrete** or **continuous**. A discrete distribution can only take on specified values, typically the integers. The binomial is a great example of this - it can be any integer greater than or equal to zero. For example, the possible values of a binomial are {0, 1, 2, 3, ...}.

On the other hand, a continuous distribution can take on any real number within a specified range. For example, looking at the standard normal distribution, a list of some possible values could be {0, 0.1, 0.01, 0.001, 0.0001, ...}. This is a limited list of the infinite list of possible values that a continuous distribution could take on.

The reason we care about the classification of a random variable as discrete or continous is that it affects how we use our probability functions in *R*. Let's walk through an example to see the difference. Suppose we have a normally distributed random variable, X, and a binomially distributed random variable, Y.

$$X \sim N(0,1) \qquad Y \sim Binom(15, 0.5)$$

If we wanted to know for these two distributions, the probability of being less than 2, we would consider $P(X < 2)$ and $P(Y < 2)$. However, for the binomial distribution $P(Y < 2) = P(Y \leq 1) = P(Y = 0) + P(Y = 1)$ because there is a discrete set of possible values that Y can take on. Y can only take on the integer values, and less than 2, that just leaves 0 and 1, whereas the normal distribution can take on any non-integer value in that interval, which is an infinite number of values. For the normal distribution, solving for $P(Y < 2)$ is mathematically equivalent to $P(Y \leq 2)$ because it is continuous.

So to solve for these probabilities, we would use *pnorm* and *pbinom*, but we have to be careful with the discrete binomial case because these functions automatically give the less than or equal to probability, not just the less than probability. The correct commands to use would be

```r
pnorm(2)

## [1] 0.9772499

pbinom(1, size = 15, prob=0.5)

## [1] 0.0004882812
```

To summarize, we would use the binomial probaility functions as follows,

- $dbinom(x, n, p) = P(X = x)$
- $pbinom(x, n, p) = P(X \leq x)$
- $pbinom(x, n, p, lower.tail = FALSE) = P(X > x)$

3.4 Other Distributions

3.4.1 Poisson Distribution

Similarly, we can build out the functions for the **poisson distribution** as the following where the necessary argument is *lambda*

- $dpoisson(x, lambda) = P(X = x)$
- $ppois(x, lambda) = P(X \leq x)$
- $ppois(x, lambda, lower.tail = FALSE) = P(X > x)$

3.4.2 Geometric Distribution

We can follow the same pattern for the **geometric distribution** to get the following where the necessary argument is *prob*

- $dgeom(x, prob) = P(X = x)$
- $pgeom(x, prob) = P(X \leq x)$

3.4.3 Negative Binomial Distribution

Finally, we can follow the same pattern for the **negative binomial distribution** to get the following where the necessary arguments are *prob* and *size*

- $dnbinom(x, n, p) = P(X = x)$
- $pnbinom(x, n, p) = P(X \leq x)$

Chapter 4

Chapter 4 - Name this