

# R and Its Applications

Traci Blonquist

Robert Thompson

Spring 2010

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	What is R? . . . . .	1
1.2	How to get R . . . . .	1
1.2.1	Download . . . . .	1
1.2.2	Install . . . . .	1
1.3	Running R (Windows) . . . . .	1
<b>2</b>	<b>Commands</b>	<b>2</b>
2.1	Basic Functions . . . . .	2
2.1.1	Simple Arithmetic . . . . .	2
2.1.2	Standard Operations . . . . .	2
2.1.3	Trigonometric Functions . . . . .	3
2.1.4	Vectors . . . . .	4
2.1.5	Factors . . . . .	4
2.1.6	Lists . . . . .	5
2.1.7	Data Frames . . . . .	5
2.1.8	Statistics . . . . .	5
2.2	Input . . . . .	6
2.2.1	How to Read In Files . . . . .	6
2.3	Basic Graphics . . . . .	7
<b>3</b>	<b>Plots</b>	<b>8</b>
3.1	Stem-and-Leaf Plots . . . . .	8
3.2	Histograms . . . . .	9
3.3	Boxplot . . . . .	9
<b>4</b>	<b>Distributions</b>	<b>11</b>
4.1	The Beta Distribution . . . . .	11
4.1.1	Random Number Generation: Beta . . . . .	11
4.1.2	Density: Beta . . . . .	11
4.1.3	Distribution Function: Beta . . . . .	12
4.2	The Binomial Distribution . . . . .	12
4.2.1	Random Number Generation: Binomial . . . . .	12
4.2.2	Density: Binomial . . . . .	12
4.2.3	Distribution Function: Binomial . . . . .	13
4.3	The Cauchy Distribution . . . . .	13
4.3.1	Random Number Generation: Cauchy . . . . .	13
4.3.2	Density: Cauchy . . . . .	13
4.3.3	Distribution Function: Cauchy . . . . .	14
4.4	The Chi-Squared Distribution . . . . .	14

4.4.1	Random Number Generation: Chi-Square	14
4.4.2	Density: Chi-Square	14
4.4.3	Distribution Function: Chi-Square	15
4.5	The Exponential Distribution	15
4.5.1	Random Number Generation: Exponential	15
4.5.2	Density: Exponential	15
4.5.3	Distribution Function: Exponential	16
4.6	Fisher's F Distribution	16
4.6.1	Random Number Generation: Fisher's F	16
4.6.2	Density: Fisher's F	16
4.6.3	Distribution Function: Fisher's F	17
4.7	The Gamma Distribution	17
4.7.1	Random Number Generation: Gamma	17
4.7.2	Density: Gamma	17
4.7.3	Distribution Function: Gamma	18
4.8	The Geometric Distribution	18
4.8.1	Random Number Generation: Geometric	18
4.8.2	Density: Geometric	18
4.8.3	Distribution Function: Geometric	19
4.9	The Hypergeometric Distribution	19
4.9.1	Random Number Generation: Hypergeometric	19
4.9.2	Density: Hypergeometric	19
4.9.3	Distribution Function: Hypergeometric	20
4.10	The Lognormal Distribution	20
4.10.1	Random Number Generation: Lognormal	20
4.10.2	Density: Lognormal	20
4.10.3	Distribution Function: Lognormal	21
4.11	The Logistic Distribution	21
4.11.1	Random Number Generation: Logistic	21
4.11.2	Density: Logistic	21
4.11.3	Distribution Function: Logistic	22
4.12	The Negative Binomial Distribution	22
4.12.1	Random Number Generation: Negative Binomial	22
4.12.2	Density: Negative Binomial	22
4.12.3	Distribution Function: Negative Binomial	23
4.13	The Normal Distribution	23
4.13.1	Random Number Generation: Normal	23
4.13.2	Density: Normal	23
4.13.3	Distribution Function: Normal	24
4.14	The Poisson Distribution	24
4.14.1	Random Number Generation: Poisson	24
4.14.2	Density: Poisson	24
4.14.3	Distribution Function: Poisson	25
4.15	The Wilcoxon Signed Rank Statistic	25
4.15.1	Random Number Generation: Wilcoxon Signed	25
4.15.2	Density: Wilcoxon Signed	25
4.15.3	Distribution Function: Wilcoxon Signed	25
4.16	The Student's T Distribution	26
4.16.1	Random Number Generation: Student T	26
4.16.2	Density: Student T	26
4.16.3	Distribution Function: Student T	26

4.17	The Uniform Distribution . . . . .	27
4.17.1	Random Number Generation: Uniform . . . . .	27
4.17.2	Density: Uniform . . . . .	27
4.17.3	Distribution Function: Uniform . . . . .	27
4.18	The Weibull Distribution . . . . .	28
4.18.1	Random Number Generation: Weibull . . . . .	28
4.18.2	Density: Weibull . . . . .	28
4.18.3	Distribution Function: Weibull . . . . .	28
4.19	The Wilcoxon Rank Sum . . . . .	29
4.19.1	Random Number Generation: Wilcoxon Rank . . . . .	29
4.19.2	Density: Wilcoxon Rank . . . . .	29
4.19.3	Distribution Function: Wilcoxon Rank . . . . .	29
<b>5</b>	<b>Tests</b>	<b>30</b>
5.1	Discrete Response . . . . .	30
5.1.1	Binomial Probability of Success . . . . .	30
5.1.2	Multiple Proportions of Equality . . . . .	30
5.1.3	Test for association (Fisher Test) . . . . .	31
5.1.4	Test for association (Chi-Squared) . . . . .	32
5.2	Continuous Response . . . . .	32
5.2.1	T-Test . . . . .	32
5.2.2	Test for Correlation between Two Samples . . . . .	33
5.2.3	Comparaision of Two Variances: F-test . . . . .	34
5.3	Nonparametric Tests . . . . .	34
5.3.1	Wilcoxon Rank Sum . . . . .	34
<b>6</b>	<b>Regression</b>	<b>36</b>
6.1	Correlation Coefficient . . . . .	36
6.2	Plot of variables in question . . . . .	36
6.3	Linear Model . . . . .	37
6.3.1	Summary of data . . . . .	37
6.3.2	Plotting a linear model . . . . .	38
6.3.3	Adding a line to the original plot: . . . . .	38
6.3.4	Other useful things . . . . .	39
6.4	ANOVA Table . . . . .	39
	<b>References</b>	<b>40</b>

# Chapter 1

## Introduction

### 1.1 What is R?

R is both a programming language and environment used for statistical computation and graphics. It is a part of the GNU project, which was established to create “*a sufficient body of software [...] to get along without any software that is not free.*” This means that R is available as Free Software the GNU General Public License for use and distribution, making it especially ideal for students.

R is similar to the S language and environment, which was created at Bell Laboratories. R can be considered to be a different implementation of S. R provides a wide variety of statistical methods and graphical techniques, and is also easily extended via packages that can be created and downloaded with a more specific goal in mind.

### 1.2 How to get R

#### 1.2.1 Download

Go to <http://cran.r-project.org/> and click on the appropriate link corresponding to the operating system to be used. From there, select the installation package that will best suit your needs and download it to an easily accessible place.

#### 1.2.2 Install

After you have downloaded the installer, run the installer and follow the steps as they appear onscreen.

### 1.3 Running R (Windows)

To run R in Windows, click ‘Start,’ ‘Programs,’ ‘R,’ and then ‘R’ followed by the version number you installed.

# Chapter 2

## Commands

### 2.1 Basic Functions

#### 2.1.1 Simple Arithmetic

To perform basic arithmetic operations, R follows an intuitive means of calculation; given two numbers  $a$  and  $b$ :

Operation	Input
Addition	$a+b$
Subtraction	$a-b$
Multiplication	$a*b$
Division	$a/b$
Exponentiation	$a ^ b, a**b$

#### Example

```
> 14+9
[1] 23
> 14-9
[1] 5
> 14*9
[1] 126
> 14/9
[1] 1.555556
> 14^9
[1] 20661046784
> 14**9
[1] 20661046784
```

#### 2.1.2 Standard Operations

R handles standard operations as well:

Operation	Input
Square Root	<code>sqrt()</code>
Absolute Value	<code>abs()</code>
Factorial	<code>factorial()</code>
Logarithm	<code>log()</code>
Base-10 Logarithm	<code>log10()</code>
Constant $\pi$	<code>pi</code>
Exponential	<code>exp()</code>

### Example

```
> sqrt(225)
[1] 15
> abs(-24)
[1] 24
> factorial(6)
[1] 720
> log(23)
[1] 3.135494
> log10(23)
[1] 1.361728
> pi
[1] 3.141593
> exp(1)
[1] 2.718282
```

### 2.1.3 Trigonometric Functions

R can also handle standard trigonometric functions, with angle input in radians:

Operation	Input
Sine	<code>sin()</code>
Cosine	<code>cos()</code>
Tangent	<code>tan()</code>
Arcsine	<code>asin()</code>
Arccosine	<code>acos()</code>
Arctangent	<code>atan()</code>

### Example

```
> sin(pi/4)
[1] 0.7071068
> cos(pi/6)
[1] 0.8660254
> tan(pi/3)
[1] 1.732051
> asin(1)
[1] 1.570796
> acos(0.5)
[1] 1.047198
> atan(4)
[1] 1.325818
```

## 2.1.4 Vectors

The construct `c(...)` is used to define a vector. There are 3 types of vectors:

Type	Definition
Numeric	String of Numbers
Character	Text Strings
Logical	Take values true, false, NA

### Numeric Vector

```
> c(1,2,3,4)
[1] 1 2 3 4
> c(1:4)
[1] 1 2 3 4
```

### Character Vector

```
> c('Mo', 'Curly', 'Larry')
[1] "Mo"      "Curly"  "Larry"
```

### Logical Vector

```
> c(T,T,F,F)
[1] TRUE TRUE FALSE FALSE
```

## 2.1.5 Factors

Factors are a data structure that assigns names to categories. It is necessary in distinguishing between numerical and categorical variables in R. A factor has a set of levels which are a combination of a numeric vector and a categorical vector. The vector of integers is associated with the character vector in that the character vector explains what the numerical vector means. The connection is defined with a few lines of coding:

1. Create a numeric vector of a particular category of interest
2. Create a factor of the category of interest and label the levels numerically
3. Create a categorical vector explaining what the levels mean

Note: Each vector created needs to be named in order for the coding to work. For the last step, the name has to be `levels(factor vector)`. It is also possible to convert the levels back to a numerical form by using `as.numeric`.

### Example

```
> happy<-c(0,1,2)
> fhappy<-factor(happy,levels=0:2)
> levels(fhappy)<-c('not happy', 'happy', 'very happy')
> fhappy
[1] not happy  happy      very happy
Levels: not happy happy very
> as.numeric(fhappy)
[1] 1 2 3
```

## 2.1.6 Lists

Lists combine a collection of objects into a larger composite object. A list is constructed using the function *list*. A list is created by combining a series of vectors. To do this:

1. Make vectors of variables you want apart of your list
2. Use the list function to combine the vectors under new headings

### Example

```
> food <- c('pizza', 'toast', 'brownies', 'apples')
> drink <- c('water', 'soda', 'juice', 'milk')
> partylist <- list(food=food, drink=drink)
> partylist
$food
[1] "pizza"    "toast"    "brownies" "apples"

$drink
[1] "water" "soda"   "juice"  "milk"
```

## 2.1.7 Data Frames

A data frame is the same thing as a data matrix or a data set. It is a list of vectors and/or factors of the same length that are related across by the same experimental unit. It therefore has a unique set of rows. The dataframe can be created from pre-existing variables using the code `data.frame`:

### Example

```
> snacks<-data.frame(food,drink)
> snacks
   food drink
1  pizza water
2  toast  soda
3 brownies juice
4  apples  milk
```

## 2.1.8 Statistics

R—as it is designed to do—can perform basic statistical functions that are built-in, with the input in the form of a list, vector, or data set:

Operation	Input
Mean	<code>mean()</code>
Standard Deviation	<code>sd()</code>
Variance	<code>var()</code>
Median	<code>median()</code>
Summary	<code>summary()</code>

### Example

```
> Numbers <- c(1,23,4,5,67,4,93,50,34,15)
> mean(Numbers)
[1] 29.6
> sd(Numbers)
```

```

[1] 31.34822
> var(Numbers)
[1] 982.7111
> median(Numbers)
[1] 19
> summary(Numbers)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.00   4.25   19.00   29.60   46.00   93.00

```

## 2.2 Input

### 2.2.1 How to Read In Files

*NOTE: It is important to know what type of file you are trying to read in R (ie: .txt, .csv, .xls, etc.). It is also important to know if the variables in your dataset have headers; that is to say, if the variables are titled.*

#### Use `read.table()`

*NOTE: It is important to have saved file as a tab-delimited file.*

General format:

- `read.table('C:/name of folder/ name of file.txt', header = T)`
- Make sure quotes surround destination and file name, use / instead of \, and if headers are in the include the statement `header=T`.
- Make sure the data set is set equal to something so it can be called up easier
- If headers are included in dataset, after loading the data type use the command `attach(dataset)`
- Example:

```

> data <- read.table('C:/data.txt', header = T)
> attach(data)

```

#### Comma separated files

General format:

- The same as `read.table()`, except the command is `read.csv()`.
- Example:

```

> Data <- read.csv('C:/data.csv', header = T)

```

#### Semi-separated files

General format:

- The same as `read.table()`, except the command is `read.csv2()`.

#### Browse to find a file using `file.choose()`

General format:

- `read.table(file.choose(), header=T)`
- Make sure the command is set equal to a variable name

## 2.3 Basic Graphics

- To add a title, add the line `main = 'title'`
- To change color, add the line `col = 'color'`
- To change the point, add the line `pch = 'number'`
- To add title to  $x$ -axis, add the line `xlab = 'x-axis'`
- To add title to  $y$ -axis, add the line `ylab = 'y-axis'`
- Example:

```
> plot(x,y, main = 'example', col = 'red', pch = 2, xlab = 'x', ylab='y')
```

# Chapter 3

## Plots

This material is useful for people working with one-sample data. Operations performed in this section is from the dataset `airquality` available in R. The section that will be looked at most is `Ozone`, which can be called by `airquality$Ozone`, which will be called `Ozone`.

### 3.1 Stem-and-Leaf Plots

Command: `stem(x, scale = 1, width = 80, atom = 1e-08)`

- `x` is a numeric vector
- `scale` controls the plot length
- `atom` is the tolerance

#### Example

Based on the `Ozone` data available in R, the following stem-and-leaf plot is produced:

```
> stem(Ozone)
```

The decimal point is 1 digit(s) to the right of the |

```
0 | 1467778999
1 | 01112233334444666688889
2 | 0000111123333334478889
3 | 001222455667799
4 | 01444556789
5 | 0299
6 | 134456
7 | 13367889
8 | 024559
9 | 1677
10 | 8
11 | 058
12 | 2
13 | 5
14 |
15 |
16 | 8
```

## 3.2 Histograms

Command: `hist(x, ...,)`

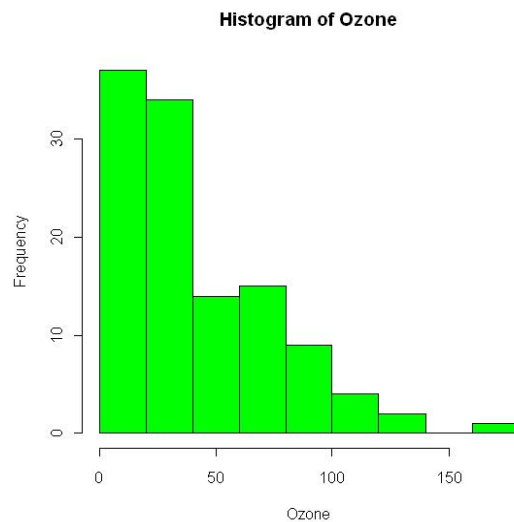
- `x` is a numeric vector

Many things can be included in the command including adding titles, changing the colors, and/or adding a legend.

### Example

Based on the Ozone data available in R, the following stem-and-leaf plot is produced:

```
> hist(Ozone, main='Histogram of Ozone', col = 'green')
```



## 3.3 Boxplot

Command: `boxplot(x, ...)`

- `x` is a numeric vector

As with the Histogram, many things can be added to the boxplot command.

For the following examples, a data set obtainable from the Data Analysis Storage Library called Labor Force is used. It has information on the number of women in the workforce for certain cities in 1972 and 1968.

### Single Boxplot

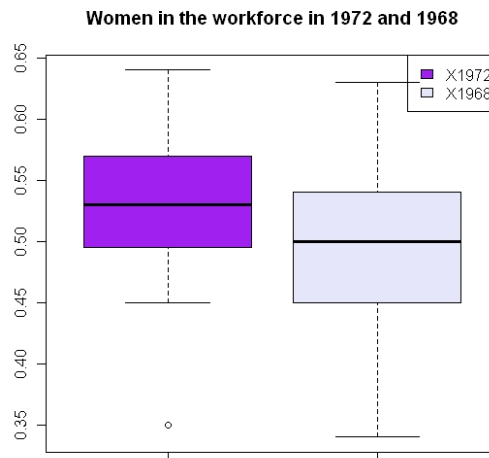
```
> labor <- read.table(file.choose(), header = T)
> attach(labor)
> boxplot(X1972, main = 'Women in the workforce in 1972', col='springgreen')
```



### Side-By-Side Boxplot

The boxplot can also be used to compare two different samples, say the number of women in the workforce in 1972 and 1968.

```
> boxplot(X1972,X1968, col = c('purple','lavender'),
+ main='Women in the workforce in 1972 and 1968')
> legend('topright',legend=c('X1972','X1968'),fill=c('purple','lavender'))
```



# Chapter 4

## Distributions

For each distribution, the density function, distribution function, and random numbers can be obtained.

### 4.1 The Beta Distribution

#### 4.1.1 Random Number Generation: Beta

For generating numbers with shape 1 and shape 2 parameters.

Command: `rbeta(n, shape1, shape2, ncp = 0)`

- `n` is the sample size
- `shape1/shape2` are positive parameters of beta distribution
- `ncp` is the non-centrality parameter (default is `ncp = 0`)

**Example**

```
> rbeta(10,2,3)
[1] 0.1347721 0.4084822 0.5058341 0.2205493 0.1150583 0.2107388 0.1186759
[8] 0.5659488 0.1868174 0.4739991
```

#### 4.1.2 Density: Beta

Command: `dbeta(x, shape1, shape2, ncp = 0, log = FALSE)`

- `x` is the vector of quantiles
- `shape1/shape2` are positive parameters of beta distribution
- `ncp` is the non-centrality parameter (default is `ncp = 0`)
- `log` is logical; if *TRUE*, probabilities  $p$  are given as  $\log(p)$

**Example**

```
> x <- seq(0, 3, length=25)
> dbeta(x,1,1)
[1] 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

### 4.1.3 Distribution Function: Beta

Command: `pbeta(q, shape1, shape2, ncp = 0, lower.tail = TRUE, log.p = FALSE)`

- `q` is the vector of quantiles
- `shape1/shape2` are positive parameters of beta distribution
- `ncp` is the non-centrality parameter (default is `ncp = 0`)
- `lower.tail` is logical; if *TRUE* (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log.p` is logical; if *TRUE*, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> x <- seq(0, 3, length=25)
> pbeta(x,1,1)
[1] 0.000 0.125 0.250 0.375 0.500 0.625 0.750 0.875 1.000 1.000 1.000 1.000
[13] 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
[25] 1.000
```

## 4.2 The Binomial Distribution

### 4.2.1 Random Number Generation: Binomial

For generating numbers based on sample size and probability of success.

Command: `rbinom(n, size, prob)`

- `n` is the sample size/observations
- `size` is the number of trials
- `prob` is the probability of success on each trial

#### Example

```
> rbinom(20,10,.5)
[1] 9 9 5 5 2 5 5 8 6 6 6 7 1 5 6 6 3 4 5 37
```

### 4.2.2 Density: Binomial

Command: `dbinom(x, size, prob, log = FALSE)`

- `x` is the vector of quantiles
- `size` is the number of trials
- `prob` is the probability of success on each trial
- `log` logical (default *FALSE*); if *TRUE*, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> y <- c(1:10)
> dbinom(y,4,0.25)
[1] 0.42187500 0.21093750 0.04687500 0.00390625 0.00000000 0.00000000
[7] 0.00000000 0.00000000 0.00000000 0.00000000
```

### 4.2.3 Distribution Function: Binomial

Command: `pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)`

- `q` is the vector of quantiles
- `size` is the number of trials
- `prob` is the probability of success on each trial
- `lower.tail` is logical; if *TRUE* (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log.p` is logical; if *TRUE*, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> y <- c(1:10)
> pbinom(y,4,0.25)
[1] 0.7382812 0.9492188 0.9960938 1.0000000 1.0000000 1.0000000 1.0000000
[8] 1.0000000 1.0000000 1.0000000
```

## 4.3 The Cauchy Distribution

### 4.3.1 Random Number Generation: Cauchy

For generating numbers based on location and scale; distribution in which the mean is not defined.

Command: `rcauchy(n, location, scale)`

- `n` is the number of observations
- `location/scale` are the input parameters (default `location = 0` and `scale = 1`)

#### Example

```
> rcauchy(10, 2, 5)
[1] -23.714410 -0.751092 11.044266 -0.472979 1.744287 -11.188024
[7] 14.210118 -26.883512 2.905069 4.764892
```

### 4.3.2 Density: Cauchy

Command: `dcauchy(x, location = 0, scale = 1, log = FALSE)`

- `x` is a vector of quantiles
- `location` is the location parameter
- `scale` is the scale parameter
- `log` is logical; if *TRUE*, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> z <- c(1:4)
> dcauchy(z)
[1] 0.15915494 0.06366198 0.03183099 0.01872411
```

### 4.3.3 Distribution Function: Cauchy

Command: `pcauchy(q, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)`

- `q` is the vector of quantiles
- `location` is the location parameter
- `scale` is the scale parameter
- `lower.tail` logical; if *TRUE* (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log.p` logical; if *TRUE*, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> z <- c(1:4)
> pcauchy(z)
[1] 0.7500000 0.8524164 0.8975836 0.9220209
```

## 4.4 The Chi-Squared Distribution

### 4.4.1 Random Number Generation: Chi-Square

For generating numbers based on degrees of freedom on the chi-squared distribution.

Command: `rchisq(n, df, ncp)`

- `n` is the number of observations
- `df` is the degrees of freedom
- `ncp` is the non-centrality parameter (default `ncp = 0`)

#### Example

```
> rchisq(10, 3)
[1] 5.3565010 2.6391035 0.8358416 3.1199989 3.3054772
[6] 4.5369201 2.5707984 0.6990311 3.5688367 2.9273041
```

### 4.4.2 Density: Chi-Square

Command: `dchisq(x, df, ncp=0, log = FALSE)`

- `x` is the vector of quantiles
- `df` is the degrees of freedom
- `ncp` is the non-centrality parameter
- `log` logical; if *TRUE*, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> dchisq(1,8)
[1] 0.006318028
```

### 4.4.3 Distribution Function: Chi-Square

Command: `pchisq(q, df, ncp=0, lower.tail = TRUE, log.p = FALSE)`

- `q` is the vector of quantiles
- `df` is the degree of freedom
- `ncp` is the non-centrality parameter
- `lower.tail` logical; if *TRUE* (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log.p` logical; if *TRUE*, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> pchisq(1,8)
[1] 0.001751623
```

## 4.5 The Exponential Distribution

For generating numbers based on rate or exponential.

### 4.5.1 Random Number Generation: Exponential

Command: `rexp(n, rate)`

- `n` is the number of observations
- `rate` (optional) is the vector of rates (default `rate = 1`)

#### Example

```
> rexp(10, 3)
[1] 1.20841539 0.37497390 0.31481535 0.03435592 0.74110316 0.18012204
[7] 0.16400281 0.59892393 0.18943655 0.82281728
```

### 4.5.2 Density: Exponential

Command: `dexp(x, rate = 1, log = FALSE)`

- `x` is the vector of quantiles
- `rate` is the vector of rates
- `log` logical; if *TRUE*, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> dexp(2)
[1] 0.1353353
```

### 4.5.3 Distribution Function: Exponential

Command: `pexp(q, rate = 1, lower.tail = TRUE, log.p = FALSE)`

- `q` is the vector of quantiles
- `rate` is the vector of rates
- `lower.tail` logical; if *TRUE* (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log.p` logical; if *TRUE*, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> pexp(2)
[1] 0.8646647
```

## 4.6 Fisher's F Distribution

### 4.6.1 Random Number Generation: Fisher's F

For generating numbers based on numerator and denominator degrees of freedom, given that the F distribution is based off of the ratio of two chi-square distributions.

Command: `rf(n, df1, df2, ncp)`

- `n` is the number of observations
- `df1/df2` are the degrees of freedom, infinity (`inf`) is allowed
- `ncp` is the non-centrality parameter; omit and central F is assumed

#### Example

```
> rf(10,3,4)
[1] 2.7747548 1.3747736 1.2948525 2.7941093 0.1954332
[6] 1.7913453 3.5808341 2.7321801 1.2642066 7.2637125
```

### 4.6.2 Density: Fisher's F

Command: `df(x, df1, df2, ncp, log = FALSE)`

- `x` is the vector of quantiles
- `df1,df2` is the numerator and denominator degrees of freedom
- `ncp` is the non-centrality parameter
- `log` logical; if *TRUE*, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> f <- c(1:30)
> df(4,2,3)
[1] 0.03884377
```

### 4.6.3 Distribution Function: Fisher's F

Command: `pf(q, df1, df2, ncp, lower.tail = TRUE, log.p = FALSE)`

- `q` is the vector of quantiles
- `df1,df2` is the numerator and denominator degrees of freedom
- `ncp` is the non-centrality parameter
- `lower.tail` logical; if *TRUE* (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log.p` logical; if *TRUE*, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> f <- c(1:30)
> pf(4,2,3)
[1] 0.8575728
```

## 4.7 The Gamma Distribution

### 4.7.1 Random Number Generation: Gamma

Generating numbers with a shape parameter

Command: `rgamma(n, shape, rate, scale)`

- `n` is the number of observations
- `rate` is an alternative way to specify scale (default = 1)
- `shape/scale` are the parameters (default `scale = 1/rate`)

#### Example

```
> rgamma(10,2,2)
[1] 0.5565034 0.2070850 1.0515220 0.7913443 0.3169567
[6] 0.1318344 2.9374537 1.0177702 3.4482931 1.7604411
```

### 4.7.2 Density: Gamma

Command: `dgamma(x, shape, rate = 1, scale = 1/rate, log = FALSE)`

- `x` is the vector of quantiles
- `shape` is the shape parameter; must be positive
- `rate` is an alternative way to specify scale
- `scale` is the scale parameter; must be positive
- `log` logical; if *TRUE*, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> g <- c(2:20)
> dgamma(g, 2, 1)
[1] 2.706706e-01 1.493612e-01 7.326256e-02 3.368973e-02 1.487251e-02
[6] 6.383174e-03 2.683701e-03 1.110688e-03 4.539993e-04 1.837187e-04
[11] 7.373055e-05 2.938428e-05 1.164140e-05 4.588535e-06 1.800563e-06
[16] 7.037894e-07 2.741396e-07 1.064531e-07 4.122307e-08
```

### 4.7.3 Distribution Function: Gamma

Command: `pgamma(q, shape, rate = 1, scale = 1/rate, lower.tail = TRUE, log.p = FALSE)`

- `q` is the vector of quantiles
- `shape` is the shape parameter; must be positive
- `rate` is an alternative way to specify scale
- `scale` is the scale parameter; must be positive
- `lower.tail` logical; if `TRUE` (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log.p` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> g <- c(2:20)
> pgamma(g, 2, 1)
[1] 0.5939942 0.8008517 0.9084218 0.9595723 0.9826487 0.9927049 0.9969808
[8] 0.9987659 0.9995006 0.9997996 0.9999201 0.9999684 0.9999875 0.9999951
[15] 0.9999981 0.9999993 0.9999997 0.9999999 1.0000000
```

## 4.8 The Geometric Distribution

### 4.8.1 Random Number Generation: Geometric

Generating numbers based on the probability

Command: `rgeom(n, prob)`

- `n` is the number of observations
- `prob` is the probability of success in each trial

#### Example

```
> rgeom(10, .2)
[1] 0 2 2 0 10 4 9 3 4 2
```

### 4.8.2 Density: Geometric

Command: `dgeom(x, prob, log = FALSE)`

- `x` is the trial of first success
- `prob` is the probability of success
- `log` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> dgeom(4, 0.3)
```

### 4.8.3 Distribution Function: Geometric

Command: `pgeom(q, prob, lower.tail = TRUE, log.p = FALSE)`

- `q` is the trial of first success
- `prob` is the probability of success
- `lower.tail` logical; if `TRUE` (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log.p` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> pgeom(4, 0.34)
[1] 0.8747667
```

## 4.9 The Hypergeometric Distribution

### 4.9.1 Random Number Generation: Hypergeometric

Generating numbers that describes the number of successes in a sequence of  $n$  draws from a finite population without replacement

Command: `rhyper(nn, m, n, k)`

- `nn` is the number of observations
- `m` is the number of one variable present
- `n` is the number of another variable present
- `k` is the number of sample drawn

#### Example

```
> rhyper(20,7,13,5)
[1] 2 2 1 1 0 2 1 1 2 3 1 1 1 1 2 3 1 2 3 2
```

### 4.9.2 Density: Hypergeometric

Command: `dhyper(x, m, n, k, log = FALSE)`

- `x` is the vector of numbers drawn without replacement
- `m` is the number of one thing available
- `n` is the number of one thing available
- `k` is the total number drawn
- `log` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> x<-c(2,3,5)
> dhyper(x,5,8,3)
[1] 0.27972028 0.03496503 0.00000000
```

### 4.9.3 Distribution Function: Hypergeometric

Command: `phyper(q, m, n, k, lower.tail = TRUE, log.p = FALSE)`

- `q` is the vector of numbers drawn without replacement
- `m` is the number of one thing available
- `n` is the number of one thing available
- `k` is the total number drawn
- `lower.tail` logical; if TRUE (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log.p` logical; if TRUE, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> x<-c(2,3,5)
> phyper(x,5,8,3)
[1] 0.965035 1.000000 1.000000
```

## 4.10 The Lognormal Distribution

### 4.10.1 Random Number Generation: Lognormal

Generating numbers using the parameters of mean and standard deviation

Command: `rlnorm(n, meanlog, sdlog)`

- `n` is the number of observations
- `meanlog/sdlog` is the mean and standard deviation of the distribution on a log scale (default `meanlog = 0` and `sdlog = 1`)

#### Example

```
> rlnorm(10)
[1] 1.4103584 1.3293625 1.4945715 0.3513362 0.9626091
[6] 0.7831300 4.3321923 0.8815737 0.2840601 0.8262417
```

### 4.10.2 Density: Lognormal

Command: `dlnorm(x, meanlog = 0, sdlog = 1, log = FALSE)`

- `x` is the vector of quantiles
- `meanlog` is the mean of the distribution on the log scale
- `sdlog` is the standard deviation of the distribution on the log scale
- `log` logical; if TRUE, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> dlnorm(4)
[1] 0.03815346
```

### 4.10.3 Distribution Function: Lognormal

Command: `plnorm(q, meanlog = 0, sdlog = 1, lower.tail = TRUE, log.p = FALSE)`

- `q` is the vector of quantiles
- `meanlog` is the mean of the distribution on the log scale
- `sdlog` is the standard deviation of the distribution on the log scale
- `lower.tail` logical; if `TRUE` (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log.p` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> plnorm(4)
[1] 0.9171715
```

## 4.11 The Logistic Distribution

### 4.11.1 Random Number Generation: Logistic

Generating numbers based on location and scale

Command: `rlogis(n, location, scale)`

- `n` is the number of observations
- `location/scale` are the parameters (default `location = 0` and `scale = 1`)

#### Example

```
> rlogis(10)
[1] -0.59868089 -0.39991247  0.13783443 -4.43324328 -0.08282099
[6] -1.09163327  0.91784355 -4.17507810 -1.18040060  3.08569926
```

### 4.11.2 Density: Logistic

Command: `dlogis(x, location = 0, scale = 1, log = FALSE)`

- `x` is the vector of quantiles
- `location` is the location parameter
- `scale` is the scale parameter
- `log` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> x <- c(1:5)
> dlogis(x)
[1] 0.196611933 0.104993585 0.045176660 0.017662706 0.006648057
```

### 4.11.3 Distribution Function: Logistic

Command: `plogis(q, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)`

- `x` is the vector of quantiles
- `location` is the location parameter
- `scale` is the scale parameter
- `lower.tail` logical; if `TRUE` (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> x <- c(1:5)
> plogis(x)
[1] 0.7310586 0.8807971 0.9525741 0.9820138 0.9933071
```

## 4.12 The Negative Binomial Distribution

### 4.12.1 Random Number Generation: Negative Binomial

Generating numbers based on size and probability

Command: `rnbinom(n, size, prob, mu)`

- `n` is the number of observations
- `size` is the target number of successful trials
- `prob` is the probability of success in each trial
- `mu` is an alternative parameterization using the mean

#### Example

```
> rnbinom(10,7,.3)
[1] 9 17 29 10 47 13 16 13 19 13
```

### 4.12.2 Density: Negative Binomial

Command: `dnbinom(x, size, prob, mu, log = FALSE)`

- `x` is the vector of quantiles
- `size` is the target for successful trials
- `prob` is the probability of success in each trial
- `mu` is alternative parameterization
- `log` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> dnbinom(5, 2, 0.3)
[1] 0.0907578
```

### 4.12.3 Distribution Function: Negative Binomial

Command: `pnbinom(q, size, prob, mu, lower.tail = TRUE, log.p = FALSE)`

- `q` is the vector of quantiles
- `size` is the target for successful trials
- `prob` is the probability of success in each trial
- `mu` is alternative parameterization
- `lower.tail` logical; if `TRUE` (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log.p` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> pnbinom(5, 2, 0.3)
[1] 0.6705828
```

## 4.13 The Normal Distribution

### 4.13.1 Random Number Generation: Normal

Generating numbers with exact mean standard deviation

Command: `rnorm(n, mean, sd)`

- `n` is the sample size
- `mean` is the desired mean
- `sd` is the desired standard deviation

#### Example

```
> rnorm(10,2,1)
[1] 2.660770 1.584287 2.941271 1.571650 2.422331 2.190849 1.412084 3.916315
[9] 1.620669 1.230931
```

### 4.13.2 Density: Normal

Command: `dnorm(x, mean = 0, sd = 1, log = FALSE)`

- `x` is the vector of quantiles
- `mean` is the vector of means
- `sd` is the vector of standard deviations
- `log` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> dnorm(6, 39, 15)
[1] 0.002364973
```

### 4.13.3 Distribution Function: Normal

Command: `pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)`

- `q` is the vector of quantiles
- `mean` is the vector of means
- `sd` is the vector of standard deviations
- `lower.tail` logical; if `TRUE` (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log.p` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> pnorm(6, 39, 15)
[1] 0.01390345
```

## 4.14 The Poisson Distribution

### 4.14.1 Random Number Generation: Poisson

Generating numbers based on the mean

Command: `rpois(n, lambda)`

- `n` is the number of random values to return
- `lambda` is the mean

#### Example

```
> rpois(10,3)
[1] 5 0 1 1 5 1 1 0 1 2
```

### 4.14.2 Density: Poisson

Command: `dpois(x, lambda, log = FALSE)`

- `x` is the vector of quantiles
- `lambda` is the vector of means
- `log` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> dpois(2,4)
[1] 0.1465251
```

### 4.14.3 Distribution Function: Poisson

Command: `ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)`

- `x` is the vector of quantiles
- `lambda` is the vector of means
- `lower.tail` logical; if `TRUE` (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log.p` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> ppois(2,4)
[1] 0.2381033
```

## 4.15 The Wilcoxon Signed Rank Statistic

### 4.15.1 Random Number Generation: Wilcoxon Signed

Generating numbers with a particular sample size

Command: `rsignrank(nn, n)`

- `nn` is the number of observations
- `n` is the number of observations in the sample

#### Example

```
> rsignrank(10,5)
[1] 6 0 11 3 12 8 5 11 14 5
```

### 4.15.2 Density: Wilcoxon Signed

Command: `dsignrank(x, n, log = FALSE)`

- `x` is the vector of quantiles
- `n` is the number of observations in the sample
- `log` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> dsignrank(4,5)
[1] 0.0625
```

### 4.15.3 Distribution Function: Wilcoxon Signed

Command: `psignrank(q, n, lower.tail = TRUE, log.p = FALSE)`

- `q` is the vector of quantiles
- `n` is the number of observations in the sample
- `lower.tail` logical; if `TRUE` (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$

- `log.p` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

### Example

```
[1] 0.0625
> psignrank(4,5)
[1] 0.21875
```

## 4.16 The Student's T Distribution

### 4.16.1 Random Number Generation: Student T

Generating numbers with degrees of freedom as the parameter  
 Command: `rt(n, df, ncp)`

- `n` is the number of observations
- `df` is the degrees of freedom; infinity (`Inf`) is allowed
- `ncp` is the non-centrality parameter; if omitted assumed central t

### Example

```
> rt(10,1)
[1] 0.92811572 -0.08416517 6.28972773 0.34112849 -1.15486428
[6] 1.66572962 1.09059932 2.36878479 19.88746057 -1.10718180
```

### 4.16.2 Density: Student T

Command: `dt(x, df, ncp, log = FALSE)`

- `x` is the vector of quantiles
- `df` is the degrees of freedom
- `ncp` is the non-centrality parameter
- `log` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

### Example

```
> dt(5, 13)
[1] 0.0002146272
```

### 4.16.3 Distribution Function: Student T

Command: `pt(q, df, ncp, lower.tail = TRUE, log.p = FALSE)`

- `q` is the vector of quantiles
- `df` is the degree of freedom
- `ncp` is the non-centrality parameter
- `lower.tail` logical; if `TRUE` (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log.p` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

### Example

```
> pt(5, 13)
[1] 0.9998785
```

## 4.17 The Uniform Distribution

### 4.17.1 Random Number Generation: Uniform

Generating numbers with the minimum and maximum as parameters

Command: `runif(n, min, max)`

- `n` is the number of observations
- `min`/`max` are the upper and lower limits of the distribution (default `min = 0` and `max = 1`)

#### Example

```
> runif(10,3,6)
[1] 5.936356 4.055628 4.982546 3.937450 5.411350
[6] 5.897618 3.599752 3.943389 4.321685 3.328836
```

### 4.17.2 Density: Uniform

Command: `dunif(x, min=0, max=1, log = FALSE)`

- `x` is the vector of quantiles
- `min` is the lower limit of the distribution
- `max` is the upper limit of the distribution
- `log` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> x<-c(0.5, 0.24, 0.009)
> dunif(x)
[1] 1 1 1
```

### 4.17.3 Distribution Function: Uniform

Command: `punif(q, min=0, max=1, lower.tail = TRUE, log.p = FALSE)`

- `x` is the vector of quantiles
- `min` is the lower limit of the distribution
- `max` is the upper limit of the distribution
- `lower.tail` logical; if `TRUE` (default), probabilities are  $P[X \leq x]$ , otherwise,  $P[X > x]$
- `log` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> x<-c(0.5, 0.24, 0.009)
> punif(x)
[1] 0.500 0.240 0.009
```

## 4.18 The Weibull Distribution

### 4.18.1 Random Number Generation: Weibull

Generating numbers with a particular shape

Command: `rweibull(n, shape, scale)`

- `n` is the number of observations
- `shape/scale` are the parameters (default `scale = 1`)

#### Example

```
> rweibull(10,2)
[1] 0.48313911 1.23519967 0.37096164 0.17545258 0.98799180
[6] 0.83823075 0.47142624 0.56597357 0.01436115 0.28473754
```

### 4.18.2 Density: Weibull

Command: `dweibull(x, shape, scale = 1, log = FALSE)`

- `x` is the vector of quantiles
- `shape` is the shape parameter
- `scale` is the scale parameter
- `log` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> x <- c(1:4)
> dweibull(x, 1)
[1] 0.36787944 0.13533528 0.04978707 0.01831564
```

### 4.18.3 Distribution Function: Weibull

Command: `pweibull(q, shape, scale = 1, lower.tail = TRUE, log.p = FALSE)`

- `x` is the vector of quantiles
- `shape` is the shape parameter
- `scale` is the scale parameter
- `lower.tail` logical; if `TRUE` (default), probabilities are  $P(X \leq x)$ ; otherwise,  $P(X > x)$
- `log` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> x <- c(1:4)
> pweibull(x, 1)
[1] 0.6321206 0.8646647 0.9502129 0.9816844
```

## 4.19 The Wilcoxon Rank Sum

### 4.19.1 Random Number Generation: Wilcoxon Rank

For generating numbers of two sample sizes. Command: `rwilcox(nn, m, n)`

- `nn` is the number of observations
- `m/n` are the number of observations in the first and second samples

#### Example

```
> rwilcox(10,5,5)
[1] 12 16 10 14 8 7 13 10 9 9
```

### 4.19.2 Density: Wilcoxon Rank

Command: `dwilcox(x, m, n, log = FALSE)`

- `x` is the vector of quantiles
- `m` is the number of observations in the first sample
- `n` is the number of observations in the second sample
- `log` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> dwilcox(3, 5, 7)
[1] 0.003787879
```

### 4.19.3 Distribution Function: Wilcoxon Rank

Command: `pwilcox(q, m, n, lower.tail = TRUE, log.p = FALSE)`

- `x` is the vector of quantiles
- `m` is the number of observations in the first sample
- `n` is the number of observations in the second sample
- `lower.tail` logical; if `TRUE` (default), probabilities are  $P(X \leq x)$ ; otherwise,  $P(X > x)$
- `log` logical; if `TRUE`, probabilities  $p$  are given as  $\log(p)$

#### Example

```
> pwilcox(3, 5, 7)
[1] 0.008838384
```

# Chapter 5

## Tests

### 5.1 Discrete Response

#### 5.1.1 Binomial Probability of Success

Command: `binom.test(x, n, p = 0.5, alternative = c('two.sided', 'less', 'greater'), conf.level=0.95)`

- `x` is the number of successes
- `n` is the number of trials
- `p` is the probability of success (default setting is `p = 0.5`)
- `alternative` is what the alternative hypothesis states
- `conf.level` is the confidence level for the confidence interval (default setting is `conf.level = 0.95`)

#### Example

```
> binom.test(20, 200, p = .85, alternative = 'two.sided', conf.level = 0.9)
```

```
Exact binomial test
```

```
data: 20 and 200
number of successes = 20, number of trials = 200, p-value < 2.2e-16
alternative hypothesis: true probability of success is not equal to 0.85
90 percent confidence interval:
 0.0672614 0.1419893
sample estimates:
probability of success
                0.1
```

#### 5.1.2 Multiple Proportions of Equality

Command: `prop.test(x, n, p = NULL, alternative = c('two.sided', 'less', 'greater'), conf.level = 0.95, correct = TRUE)`

- `x` is the vector counts of successes
- `n` is the vector number of trials

- `p` is the probability of success
- `alternative` is what the alternative hypothesis states (only used for single proportion tests)
- `conf.level` is the confidence level used
- `correct` applies continuity correction (+/ - 0.5)

### Multiple Proportion Test

```
> prop.test(x=c(120, 175, 200, 100), n=c(150, 200, 250, 150))
```

```
4-sample test for equality of proportions without continuity correction
```

```
data: c(120, 175, 200, 100) out of c(150, 200, 250, 150)
X-squared = 22.9229, df = 3, p-value = 4.191e-05
alternative hypothesis: two.sided
sample estimates:
  prop 1    prop 2    prop 3    prop 4
0.8000000 0.8750000 0.8000000 0.6666667
```

### Single Proportion Test

```
> prop.test(50, 90, .5, alternative = 'less')
```

```
1-sample proportions test with continuity correction
```

```
data: 50 out of 90, null probability 0.5
X-squared = 0.9, df = 1, p-value = 0.8286
alternative hypothesis: true p is less than 0.5
95 percent confidence interval:
 0.000000 0.644123
sample estimates:
      p
0.5555556
```

### 5.1.3 Test for association (Fisher Test)

Best used with  $2 \times 2$  tables, with inference based on odds ratio; it gives an exact  $p$ -value.

Command: `fisher.test(x, y = NULL, alternative = 'two.sided', conf.int = TRUE, conf.level = 0.95)`

- `x` is a contingency table in matrix form
- `y` is a factor object (ignored if `x` is a matrix)
- `alternative` is the alternative hypothesis (only used in a  $2 \times 2$  case)
- `conf.int` tells it to compute a confidence interval
- `conf.level` is the confidence level for confidence interval (only definable in a  $2 \times 2$  case)

### Example

```
> mat=matrix(c(10, 20, 30, 40 , 50, 60), nrow=2)
> fisher.test(mat)
```

Fisher's Exact Test for Count Data

```
data: mat
p-value = 0.5177
alternative hypothesis: two.sided
```

### 5.1.4 Test for association (Chi-Squared)

Used mostly for larger dimensioned tables.

Command: `chisq.test(x, y = NULL, correct = TRUE)`

- `x` is a vector or a matrix
- `y` is a vector (ignored if `x` is a matrix)
- `correct` adds a continuity correction for  $2 \times 2$  tables

#### Example

```
> mat1=matrix(c(500, 300, 200, 250, 450, 700, 600, 550), nrow=2, byrow=T)
> chisq.test(mat1)
```

Pearson's Chi-squared test

```
data: mat1
X-squared = 180.3453, df = 3, p-value < 2.2e-16
```

## 5.2 Continuous Response

### 5.2.1 T-Test

T-tests are based on the assumption that the data comes from the Normal distribution. It is primarily used on larger sample sizes.

#### One-Sample t-test

Command: `t.test(x, alternative, mu)`

- `x` is a numeric vector of data values
- `alternative` is the alternative hypothesis; *'two.sided'* (default), *'greater'*, *'less'*
- `mu` is the true value of the mean

#### Example

The women in the workforce data set will be used again. We can perform a t-test on the number of women in the workforce in 1972.

```
> t.test(X1972)
```

One Sample t-test

```

data: X1972
t = 32.4388, df = 18, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.4927208 0.5609634
sample estimates:
mean of x
0.5268421

```

## Two-Sample t-test

Command: `t.test(x, y, alternative, mu, paired, var.equal)`

- `x` is a numeric vector of data values
- `y` is an optional numeric vector of data values
- `alternative` is the alternative hypothesis; `'two.sided'` (default), `'greater'`, `'less'`
- `mu` is the mean difference in means between the 2 samples
- `paired` is a logical variable indicating a paired t-test; `FALSE` (default)
- `var.equal` is a logical variable indicating whether of not to treat the 2 variances as equal; if `TRUE`, pooled variance is used; if `FALSE`, an approximation to the degrees of freedom is used

### Example

Using again the women in the work force data set, a two sample (paired) t-test can be performed.

```
> t.test(X1972, X1968, paired = T)
```

Paired t-test

```

data: X1972 and X1968
t = 2.4577, df = 18, p-value = 0.02435
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.004889895 0.062478527
sample estimates:
mean of the differences
 0.03368421

```

## 5.2.2 Test for Correlation between Two Samples

Command: `cor.test(x, y, alternative, method)`

- `x` is a numeric vector of data; needs to be the same length as `y`
- `y` is a numeric vector of data; needs to be the same length as `x`
- `alternative` indicates the alternative hypothesis; `'two.sided'`, `'greater'`, `'less'`
- `method` is a character string signifying which correlation coefficient should be used; `'pearson'` (default), `'kendall'`, `'spearman'`

### Example

Using the vapor pressure of mercury as a function of temperature dataset titled `pressure` available in R, a correlation test can be performed.

```
> cor.test(temperature, pressure)

Pearson's product-moment correlation

data: temperature and pressure
t = 4.7885, df = 17, p-value = 0.0001710
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.4629176 0.9016552
sample estimates:
      cor
0.7577923
```

### 5.2.3 Comparison of Two Variances: F-test

Command: `var.test(x, y, ratio, alternative)`

- `x` is a numeric vector of data values
- `y` is a numeric vector of data values
- `ratio` is the hypothesized ratio of the population variances for  $x$  and  $y$ ; `ratio = 1` (default)
- `alternative` specifies the alternative hypothesis; `'two.sided'`, `'greater'`, `'less'`

### Example

We can first randomly generate two data sets for  $x$  and  $y$ ; their variances can then be compared.

```
> x <- rnorm(20, mean = 3, sd = 2.4)
> y <- rnorm(50, mean = 2, sd = 3)
> var.test(x,y)

F test to compare two variances

data: x and y
F = 0.3743, num df = 19, denom df = 49, p-value = 0.02188
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.1854621 0.8602850
sample estimates:
ratio of variances
      0.374306
```

## 5.3 Nonparametric Tests

### 5.3.1 Wilcoxon Rank Sum

#### One-sample Wilcoxon Test

Command: `wilcox.test(x, alternative, correct)`

- `x` is the numeric vector of data values
- `y` is an optional numeric vector of data values (the same length as  $x$ )
- `alternative` is the alternative hypothesis; `'two.sided'`, `'greater'`, `'less'`
- `correct` is logical indicating whether or not to apply the continuity correction for the normal approximation

### Example

To follow along, the `airquality` dataset available in R will be used.

```
> wilcox.test(Ozone, Temp)
```

```
Wilcoxon rank sum test with continuity correction
```

```
data: Ozone and Temp
```

```
W = 2891, p-value < 2.2e-16
```

```
alternative hypothesis: true location shift is not equal to 0
```

### Two-sample Wilcoxon Test (aka Mann-Whitney Test)

Command: `wilcox.test(x, y, alternative, paired, correct)`

- `x` is the numeric vector of data values
- `y` is an optional numeric vector of data values (does not need to be the same length as  $x$ )
- `alternative` is the alternative hypothesis; `'two.sided'`, `'greater'`, `'less'`
- `paired` is logical indicating whether or not a paired test is to be used
- `correct` is logical indicating whether or not to apply the continuity correction for the normal approximation

**Example** Two vectors will be created for the purpose of this example:

```
> x <- c(1.43, 2.54, 1.80, 2.34, 3.53, 4.23)
```

```
> y <- c(2.34, 4.21, 4.54, 3.43)
```

```
> wilcox.test(x,y)
```

```
Wilcoxon rank sum test with continuity correction
```

```
data: x and y
```

```
W = 6.5, p-value = 0.2850
```

```
alternative hypothesis: true location shift is not equal to 0
```

Warning message:

```
In wilcox.test.default(x, y) : cannot compute exact p-value with ties
```

## Chapter 6

# Regression

To follow along, the dataset titled `women` containing the average heights and weights of American women as made available in R will be used in all examples.

### 6.1 Correlation Coefficient

Command: `cor(x,y)`

#### Example

A look at the relationship between the height and weight of American women:

```
> cor(height,weight)
[1] 0.9954948
```

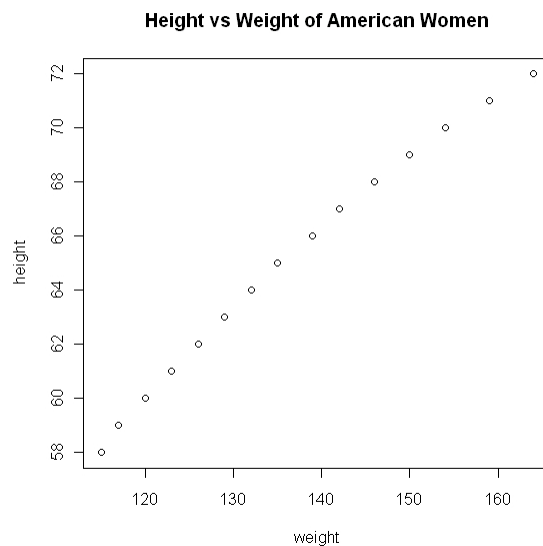
### 6.2 Plot of variables in question

Command: `Plot(y~x)`

#### Example

A look at the relationship between the height and weight of American women:

```
> plot(height~weight, main = 'Height vs Weight of American Women')
```



## 6.3 Linear Model

To find the slope and intercept for the given variables. Command: `lm(y~x)`

### Example

A look at the relationship between the height and weight of American women

```
> lm(height~weight)
```

Call:

```
lm(formula = height ~ weight)
```

Coefficients:

(Intercept)	weight
25.7235	0.2872

### 6.3.1 Summary of data

Command: `Summary(lm(y~x))`

### Example

```
> x <- lm(height~weight)
```

```
> summary(x)
```

Call:

```
lm(formula = height ~ weight)
```

Residuals:

Min	1Q	Median	3Q	Max
-0.83233	-0.26249	0.08314	0.34353	0.49790

Coefficients:

```

              Estimate Std. Error t value Pr(>|t|)
(Intercept) 25.723456   1.043746   24.64 2.68e-12 ***
weight      0.287249   0.007588   37.85 1.09e-14 ***
---

```

Signif. codes: 0 \*\*\* 0.001 \*\* 0.01 \* 0.05 . 0.1 1

Residual standard error: 0.44 on 13 degrees of freedom  
Multiple R-squared: 0.991, Adjusted R-squared: 0.9903  
F-statistic: 1433 on 1 and 13 DF, p-value: 1.091e-14

### 6.3.2 Plotting a linear model

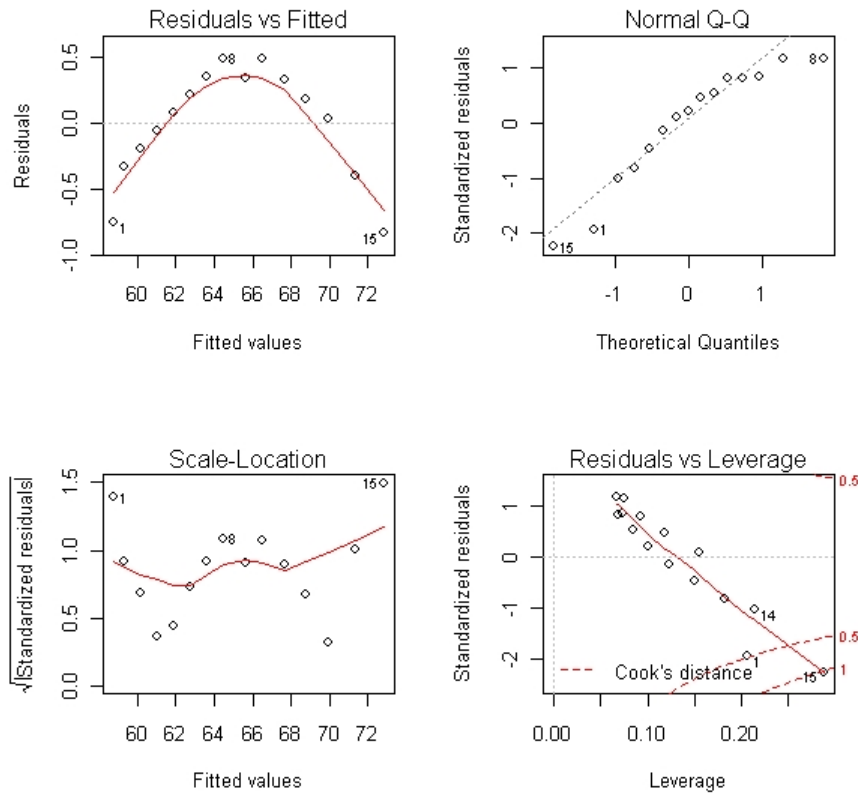
By plotting the linear model, many graphs are available for viewing, including Residuals vs Fitted Values, Normal Q-Q plot, scale location, and residuals vs leverage.

#### Example

```

> layout(matrix(c(1,2,3,4), nrow=2, byrow=T))
> plot(x)

```

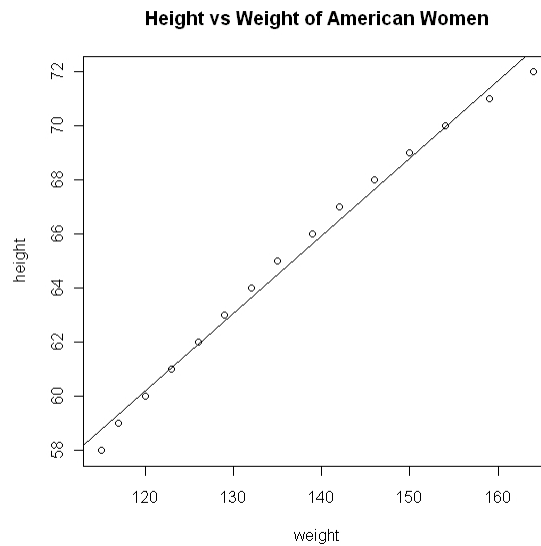


### 6.3.3 Adding a line to the original plot:

Command: `abline(lm(y~x))`

## Example

```
> plot(height~weight, main='Height vs Weight of American Women')
> abline(lm(height~weight))
```



### 6.3.4 Other useful things

- `resid(lm(y~x))` will list the residuals

- Example:

```
> resid(lm(height~weight))
      1      2      3      4      5      6
-0.75711680 -0.33161526 -0.19336294 -0.05511062  0.08314170  0.22139402
      7      8      9     10     11     12
 0.35964634  0.49789866  0.34890175  0.48715407  0.33815716  0.18916026
     13     14     15
 0.04016335 -0.39608278 -0.83232892
```

- `predict(lm(y~x))` will list the predicted values based off the linear model

- Example:

```
> predict(lm(height~weight))
      1      2      3      4      5      6      7      8
58.75712 59.33162 60.19336 61.05511 61.91686 62.77861 63.64035 64.50210
      9     10     11     12     13     14     15
65.65110 66.51285 67.66184 68.81084 69.95984 71.39608 72.83233
```

## 6.4 ANOVA Table

Command: `anova(lm(y~x))`

### Example

```
> anova(x)
Analysis of Variance Table

Response: height
      Df Sum Sq Mean Sq F value    Pr(>F)
weight  1 277.483 277.483 1433.0 1.091e-14 ***
Residuals 13  2.517  0.194
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

# Bibliography

- [1] Crawley, Michael J. *The R Book*. Chichester, England: Wiley, 2007. Print.
- [2] Dalgaard, Peter. *Introductory Statistics with R*. New York: Springer, 2008. Print.
- [3] R Development Core Team. *An Introduction to R*. R Foundation for Statistical Computing, 2009. *The R Manuals*. The Comprehensive R Archive Network, Apr. 2010. Web. 22 Apr. 2010. <<http://cran.r-project.org/doc/manuals/R-intro.pdf>>.
- [4] R Development Core Team. *R Data Import/Export*. R Foundation for Statistical Computing, 2009. *The R Manuals*. The Comprehensive R Archive Network. Web. 22 Apr. 2010. <<http://cran.r-project.org/doc/manuals/R-data.pdf>>.
- [5] R Development Core Team. *R Installation and Administration*. R Foundation for Statistical Computing, 2009. *The R Manuals*. The Comprehensive R Archive Network. Web. 22 Apr. 2010. <<http://cran.r-project.org/doc/manuals/R-admin.pdf>>.
- [6] R Development Core Team. *The R Reference Index*. R Foundation for Statistical Computing, 2009. *The R Manuals*. The Comprehensive R Archive Network. Web. <<http://cran.r-project.org/doc/manuals/fullrefman.pdf>>.