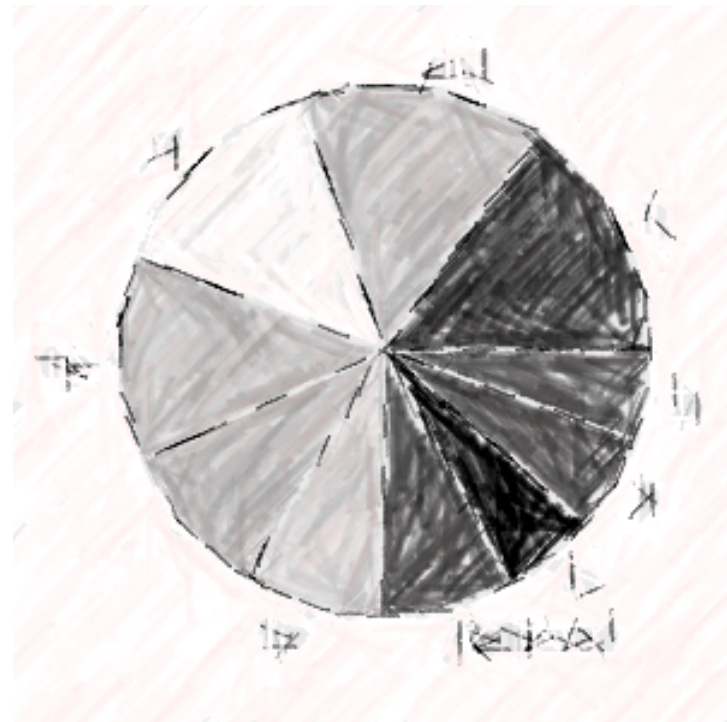


Introduction to Computing for Economics and Management

Lecture 9: Graphics Continued



Acknowledgement

- These slides are adapted from Bert Arnrich's R lecture.



Recap: scatter plots of data frames

```
> person.data
```

	Name	Height	Weight	BMI	above22.5
1	Can	1.70	65	22.49	FALSE
2	Cem	1.75	66	21.55	FALSE
3	Hande	1.62	61	23.24	TRUE
4	Lale	1.76	64	20.66	FALSE
5	Arda	1.78	63	19.88	FALSE
6	Bilgin	1.77	84	26.81	TRUE
7	Cem	1.69	75	26.26	TRUE
8	Ozlem	1.75	65	21.22	FALSE
9	Ali	1.73	75	25.06	TRUE
10	Haluk	1.71	81	27.70	TRUE

Recap: scatter plots of data frames

- We can plot each dimension of a data frame against each other by providing the data frame as argument to the `plot()` function

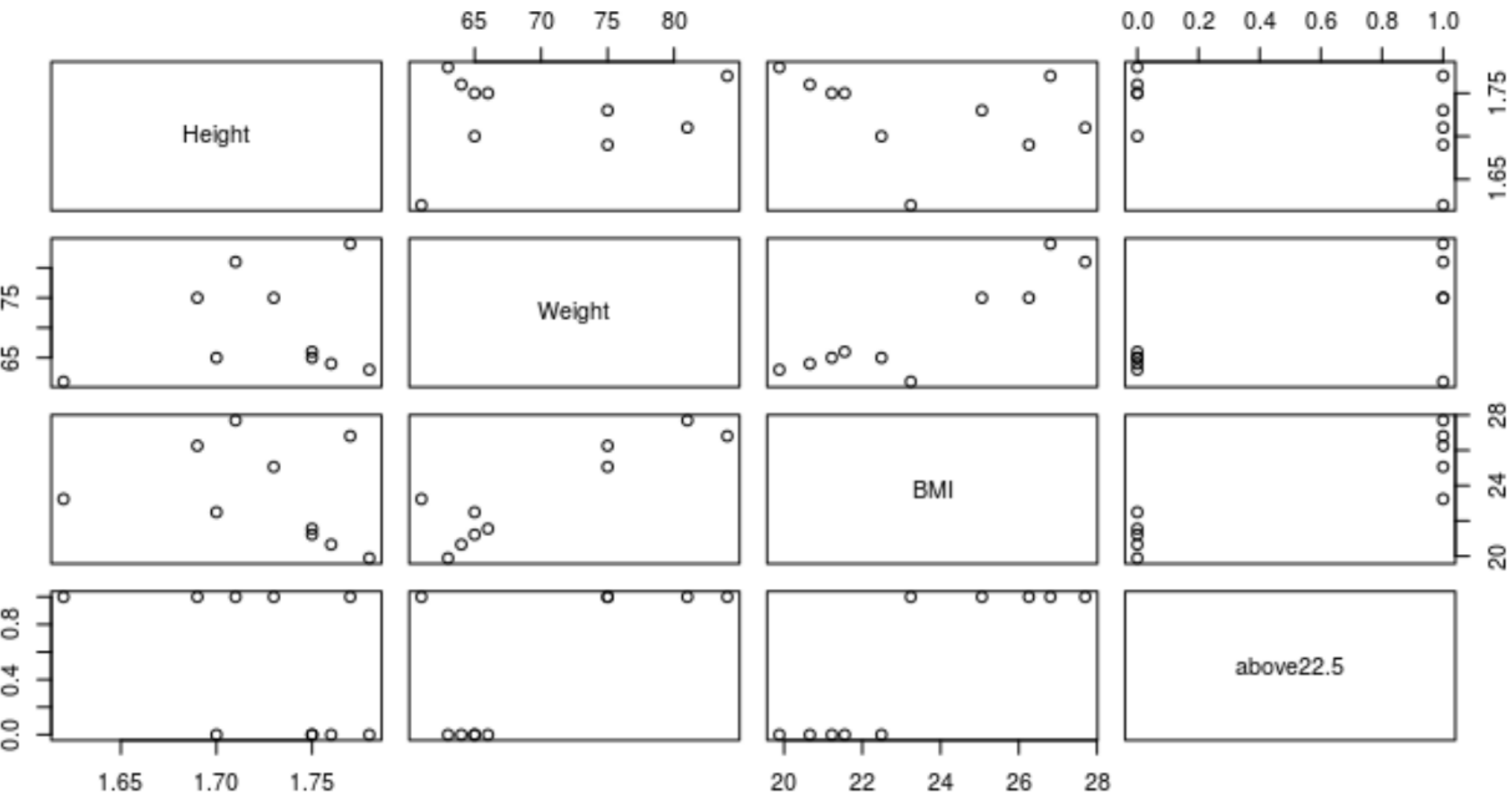
In our example, we plot the numeric and Boolean columns 2, 3, 4 and 5

- `> plot(person.data[,2:5])`

- `> scatter.plot(person.data[,2:5])`

Error: could not find function "scatter.plot"

Recap: scatter plots of data frames



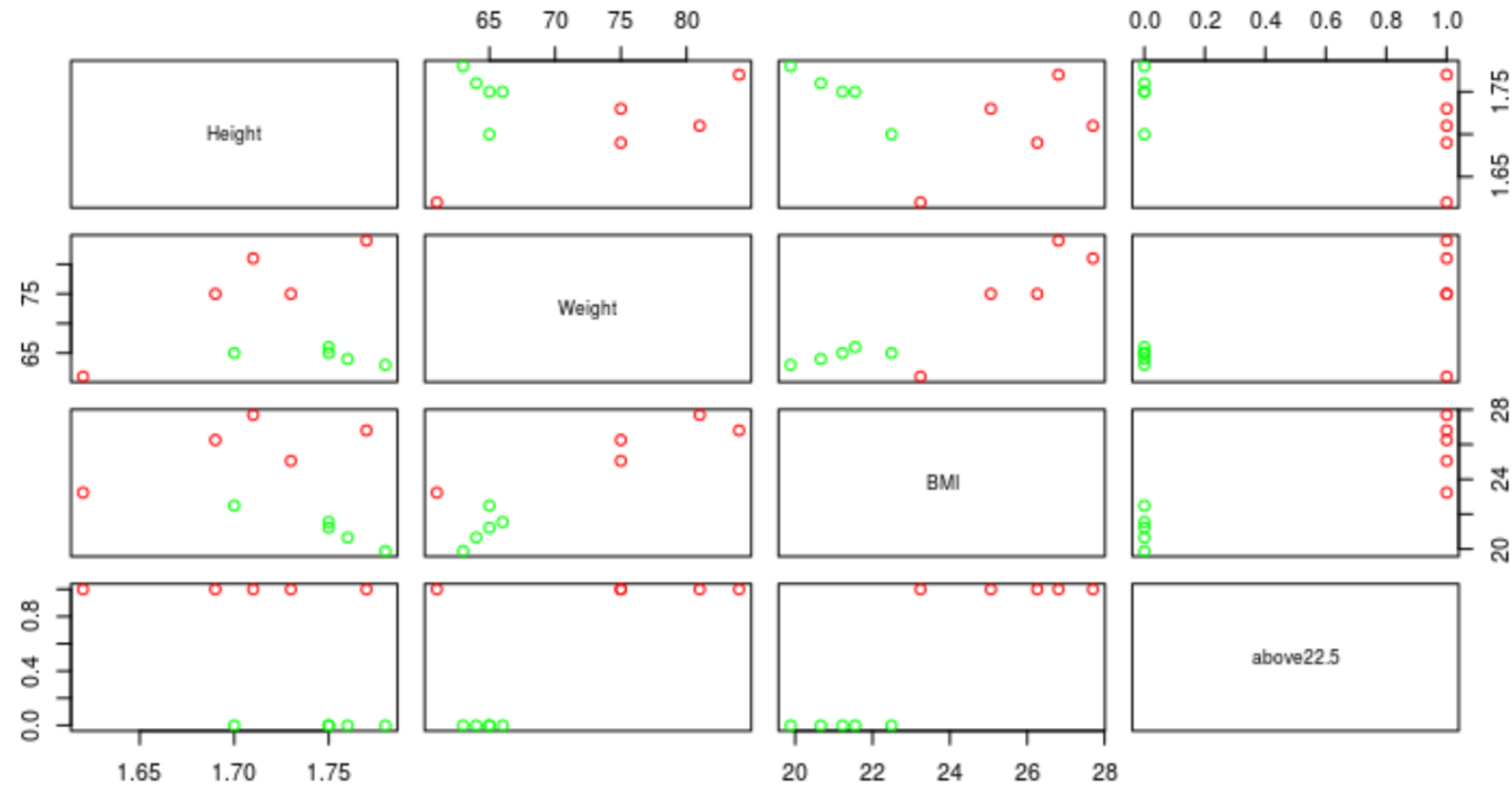
Recap: scatter plots of data frames

- In the previous plot we can observe how the numeric dimensions `Height`, `Weight`, and `BMI` are related to each other
- In order to better observe the relation between the Boolean group variable `above22.5` we use a color coding
- We can specify a color coding with the argument `col`

We use `ifelse` to specify color red for `above22.5=TRUE` and color green otherwise

- ```
> plot(person.data[,2:5],
 col=ifelse(person.data$above22.5, "red", "green"))
```

# Recap: scatter plots of data frames



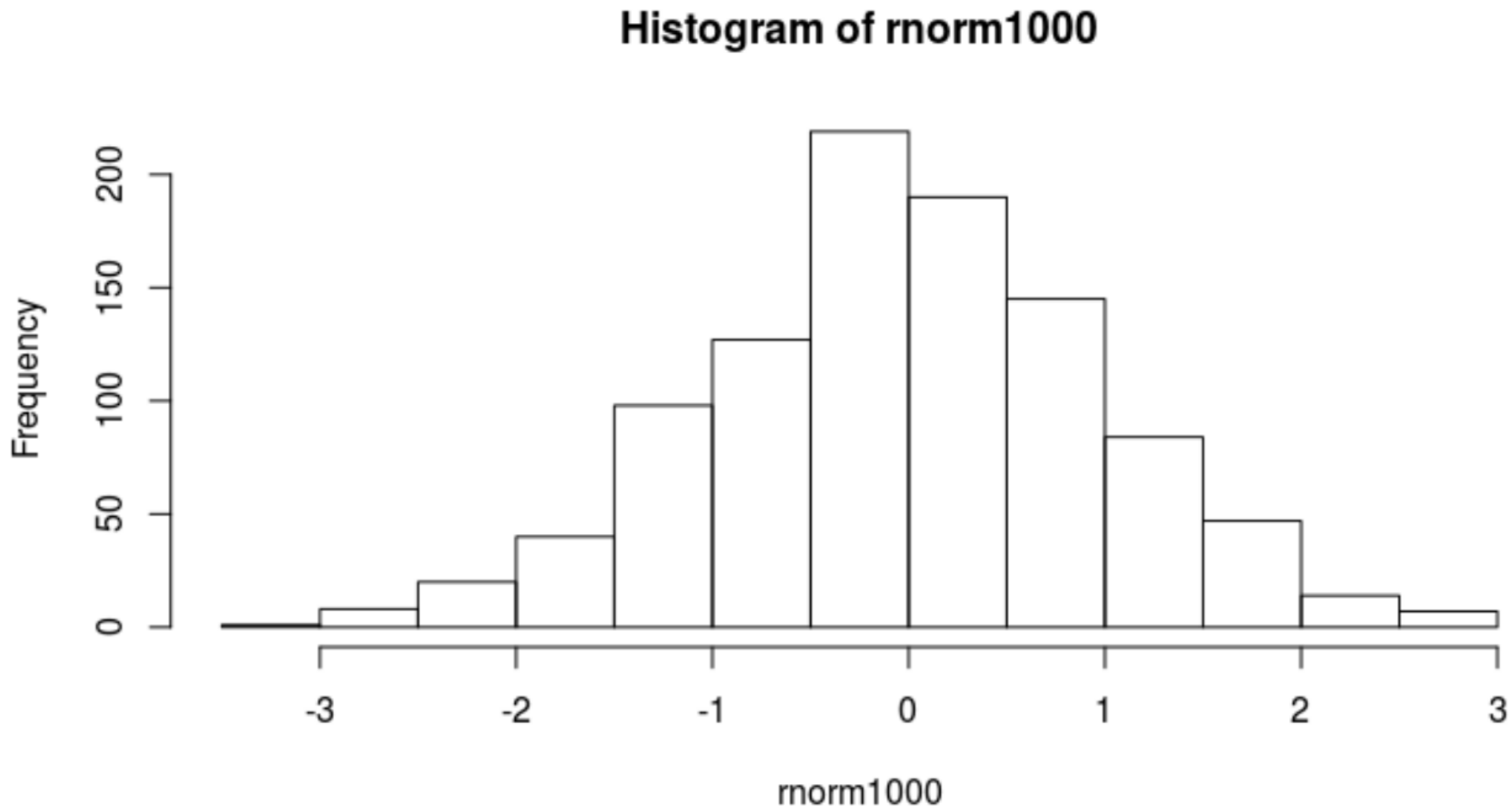
# Recap: histogram

- A histogram is similar to a barplot since we visualize how many observations fall within specified divisions called “bins”
- In R we simply call the `hist` function
- `hist` creates the bins automatically, counts the number of observations that fall within the bins and plots the result

A histogram of normal distributed random numbers with mean 0 is created with

```
> rnorm1000 <- rnorm(1000)
▪ > hist(rnorm1000)
```

# Recap: histogram



# Recap: histogram

- Let's now create a histogram of positive random numbers

We use our function `positive.rnd.numbers` from previous lecture to create positive random numbers

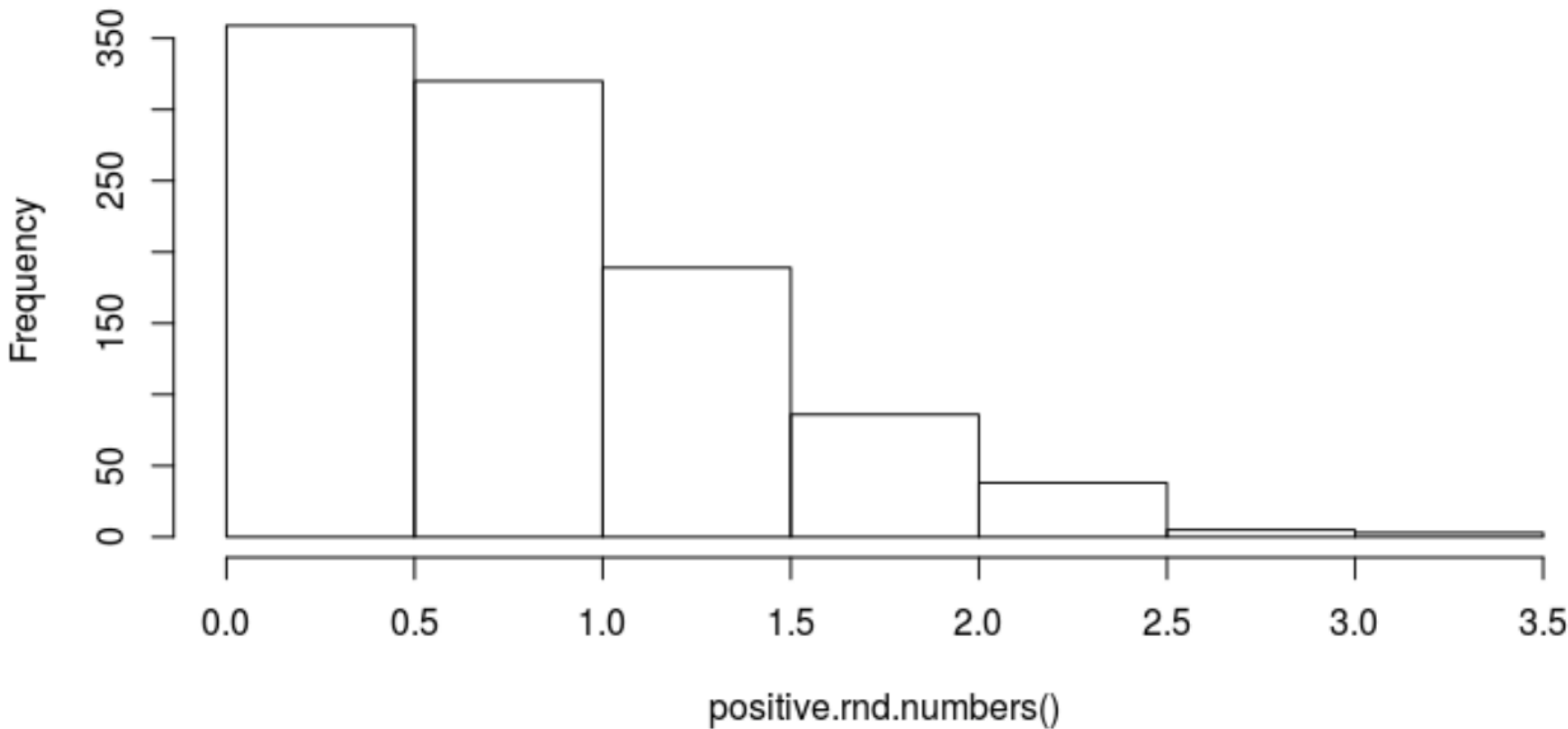
```
> hist(positive.rnd.numbers())
```

# Recap: Random numbers with the while loop

```
generate n positive random numbers
positive.rnd.numbers <- function(n=1000)
{
 i <- 1
 rnd.vec <- vector()
 while(TRUE)
 {
 # generate a single random number
 rnd.number <- rnorm(1)
 # if random number is positive add it to vector rnd.vec
 if(rnd.number > 0)
 {
 rnd.vec <- c(rnd.vec, rnd.number)
 i <- i + 1
 }
 # return rnd.vec after n positive random number
 if(i > n) {return(rnd.vec)}
 }
}
```

# Recap: histogram

Histogram of `positive.rnd.numbers()`



# Recap: figure array

- Sometimes we need to place several figures in the same plot
- For example, we would like to have the plot of the normal distributed random numbers and the corresponding histogram in one plot
- We use `par` to specify how several figures will be drawn in an number of rows by number of columns array
  - `par(mfrow=c(nr, nc))` specifies that figures will be drawn in an `nr`-by-`nc` array by rows
  - `par(mfcol=c(nr, nc))` specifies that figures will be drawn in an `nr`-by-`nc` array by columns

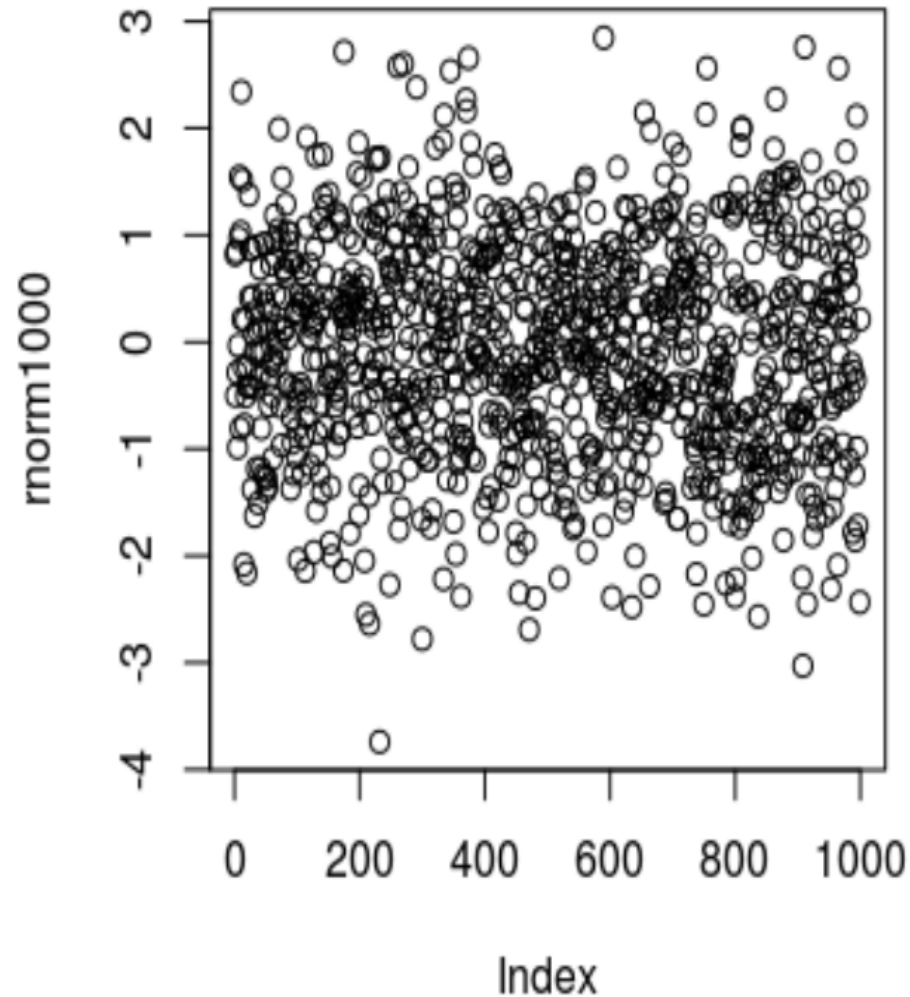
## Recap: figure array

We place the plot of the normal distributed random numbers and the corresponding histogram side by side

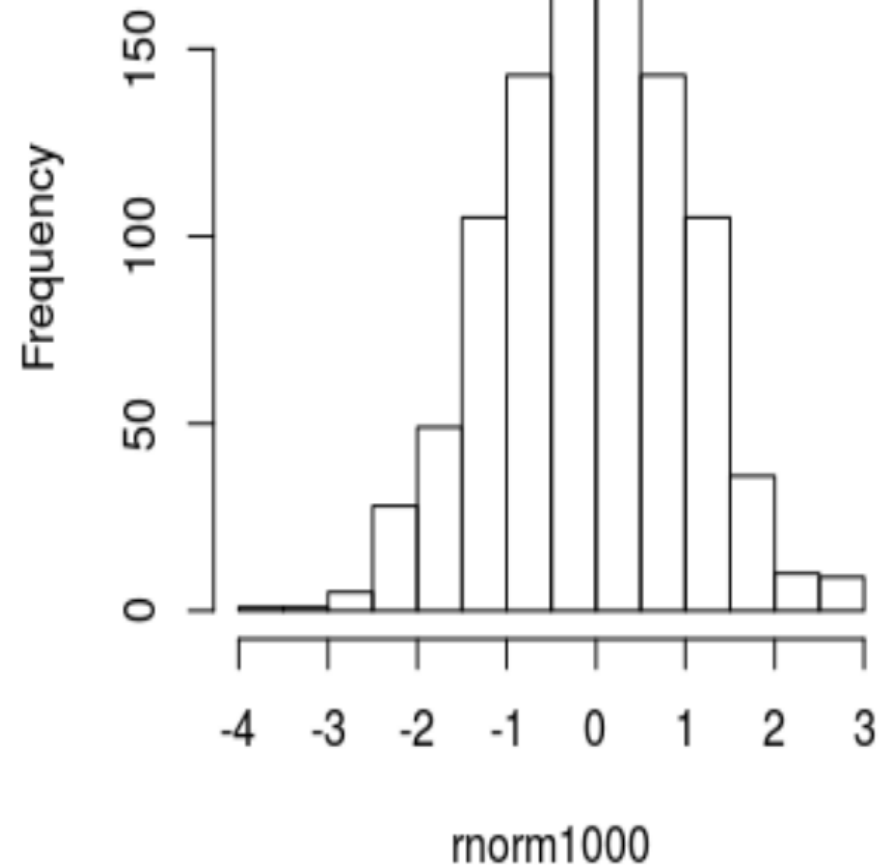
```
> rnorm1000 <- rnorm(1000)
> par(mfrow=c(1,2))
> plot(rnorm1000)
> hist(rnorm1000)
> par(mfrow=c(1,1))
```

- The last line resets `mfrow` to its default value

# Recap: figure array



Histogram of rnorm1000

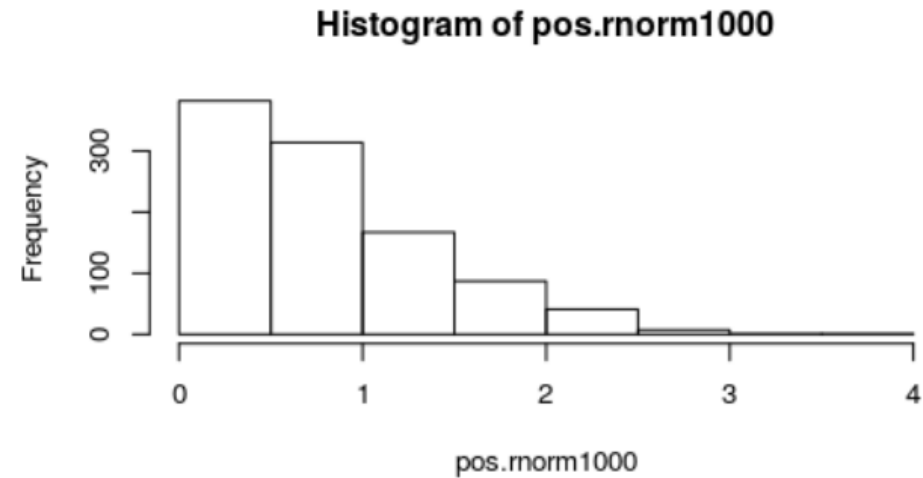
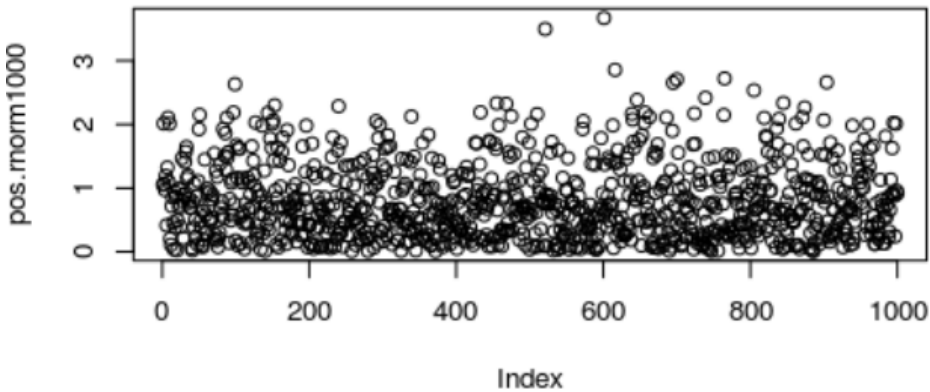
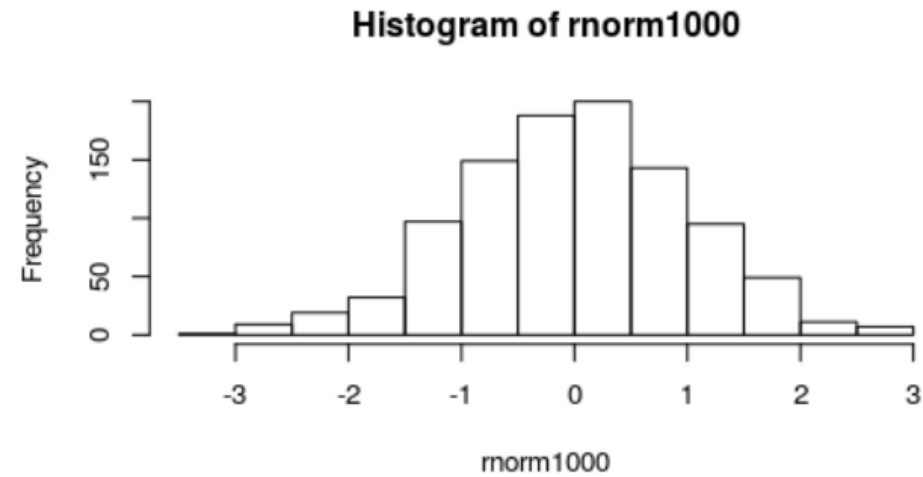
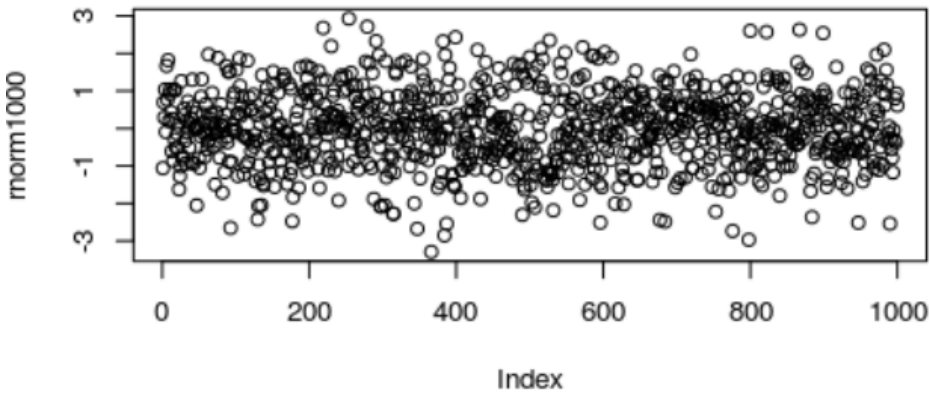


# Recap: figure array

In a 2x2 figure array we plot normal distributed random numbers, positive random numbers and their corresponding histograms

```
> rnorm1000 <- rnorm(1000)
> pos.rnorm1000 <- positive.rnd.numbers(1000)
> par(mfrow=c(2,2))
> plot(rnorm1000)
> hist(rnorm1000)
> plot(pos.rnorm1000)
> hist(pos.rnorm1000)
> par(mfrow=c(1,1))
```

# Recap: figure array



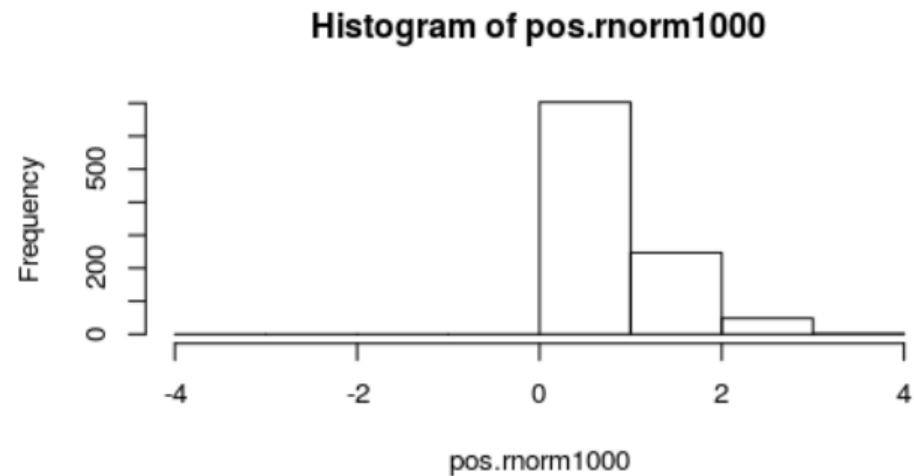
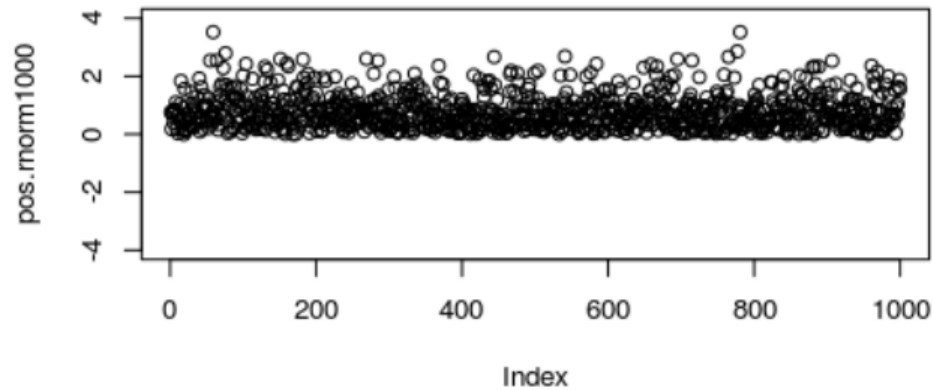
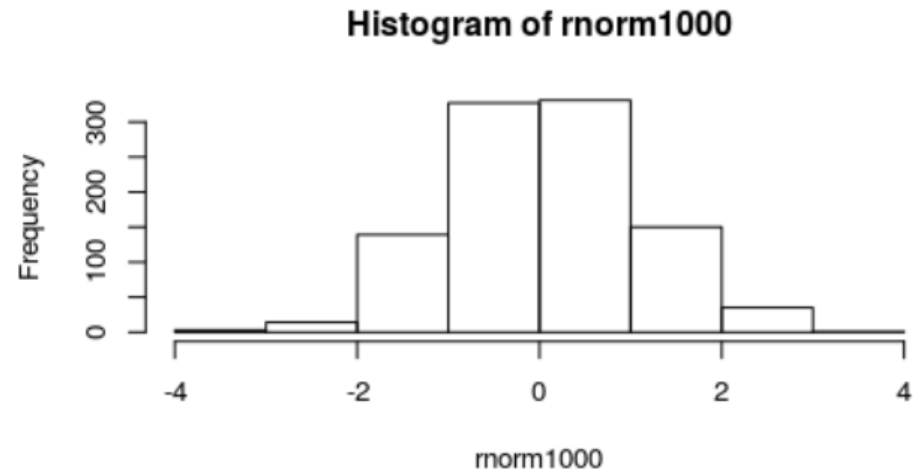
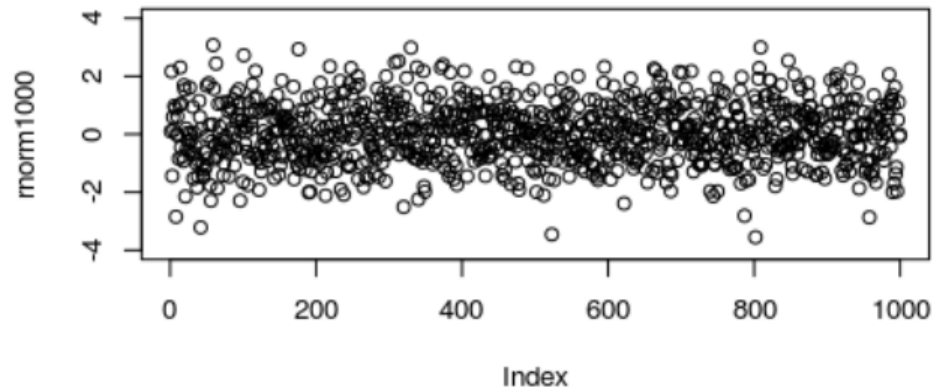
## Recap: figure array

- In the previous plot we might have difficulties to compare both distributions because axis limits and bins are chosen automatically

In order to avoid confusion, we better enforce identical axis limits and bins for both groups

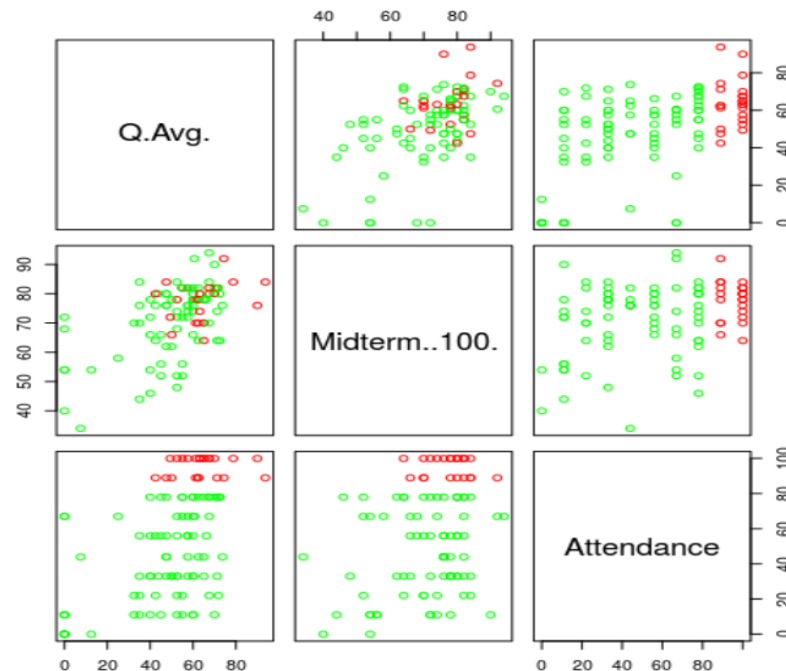
```
> par(mfrow=c(2,2))
> plot(rnorm1000, ylim=c(-4,4))
> hist(rnorm1000, breaks=seq(-4,4,1))
> plot(pos.rnorm1000, ylim=c(-4,4))
> hist(pos.rnorm1000, breaks=seq(-4,4,1))
> par(mfrow=c(1,1))
```

# Recap: figure array



# Figure array exercise: mid-semester analysis of grades, save from googledocs

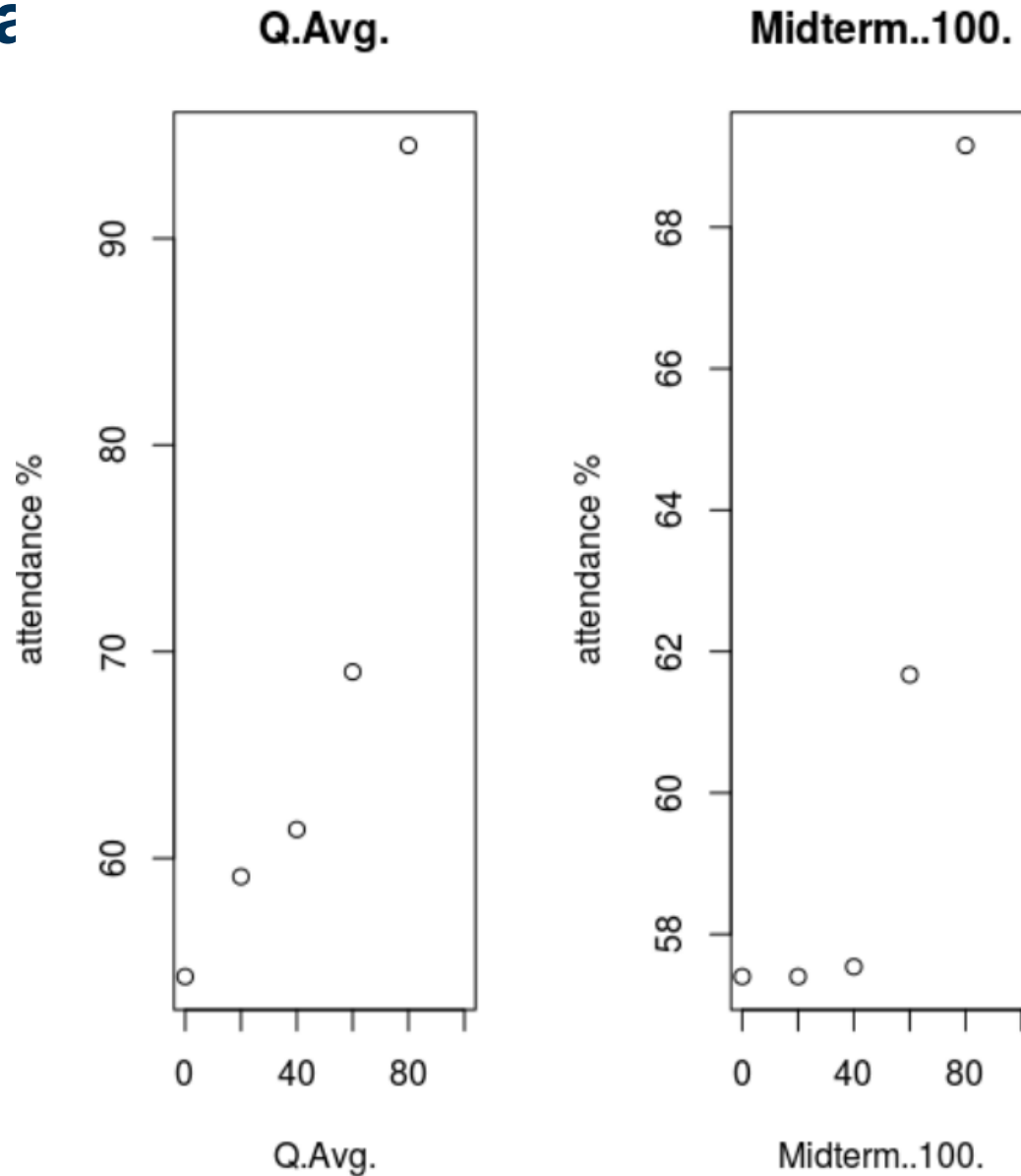
```
save google doc as csv file
grades <- read.table("grades.csv", sep="," ,header=T)
grades <- grades[1:length(grades$id)-1,]
summary(grades)
plot(grades[,c(6,7,10)], col=ifelse(grades$Attendance>80,
"red", "green"))
```



# Figure array exercise: mid-semester analysis of grades, grades average bins, v2

```
for each grade level (0%, 20%, etc), plot attendance
plot.new()
par(mfrow=c(1,2))
for (column in c(6:7)){ #cols for quiz and midterm
 means<-c()
 indices <- c()
 for (x in seq(0,100,20)){ #grade levels
 m = mean(grades$Attendance[grades[,column]>=x],
 na.rm=TRUE)
 means <- c(means,m)
 indices <- c(indices,x)
 }
 plot(indices,means,main=colnames(grades)[column],
 xlab=colnames(grades[column]),ylab='attendance %')
}
```

# Figure array exercise: mid-semester analysis of grades, gra



# Today

- Boxplot
- Stripchart
- Pie
- Examples

# Boxplot

- A so called boxplot shows us a graphical summary of a distribution
- We will create boxplots from random numbers

As a first step we create a data frame with normal distributed random numbers and positive random numbers

```
> rnumbers <- data.frame(rnorm1000=rnorm(1000),
pos.rnorm1000=positive.rnd.numbers(1000))
```

# Random numbers

```
> head(rnumbers)
```

|   | rnorm1000  | pos.rnorm1000 |
|---|------------|---------------|
| 1 | -1.7916345 | 0.3805948     |
| 2 | -0.6042055 | 0.2779461     |
| 3 | -0.2203400 | 1.2173147     |
| 4 | -1.5974419 | 0.2197444     |
| 5 | 1.4917653  | 1.6801998     |
| 6 | 0.0545331  | 0.2192972     |

```
> tail(rnumbers,n=3)
```

|      | rnorm1000  | pos.rnorm1000 |
|------|------------|---------------|
| 998  | -0.6980582 | 0.4687331     |
| 999  | -0.3358732 | 0.6524955     |
| 1000 | 1.0329650  | 0.1679040     |

# Random numbers

We learn the structure of an R object:

```
> str(rnumbers)
'data.frame': 1000 obs. of 2 variables:
 $ rnorm1000 : num -0.482 0.191 0.383 1.201
0.527 ...
 $ pos.rnorm1000: num 1.494 0.597 1.215 0.372 0.759 ...
```

We produce a summary of the data frame

```
> summary(rnumbers)
 rnorm1000 pos.rnorm1000
Min. : -3.292305 Min. : 0.000071
1st Qu.: -0.641645 1st Qu.: 0.300163
Median : -0.001719 Median : 0.673440
Mean : 0.002845 Mean : 0.795650
3rd Qu.: 0.677248 3rd Qu.: 1.147319
Max. : 2.987034 Max. : 3.889375
```

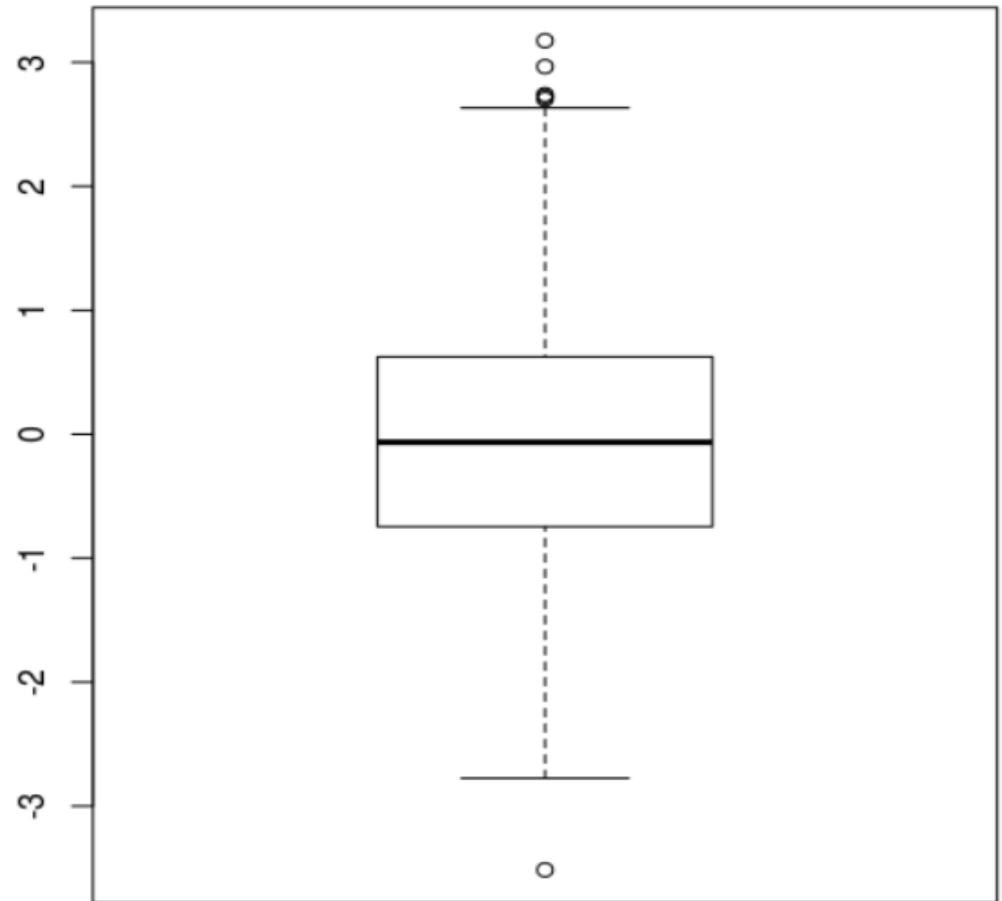
We observe min/max values, 1<sup>st</sup>/3<sup>rd</sup> quartile, median and mean of both distributions

# Boxplot

A “boxplot”, also known as “box-and-whiskers plot”, is a graphical summary of a distribution.

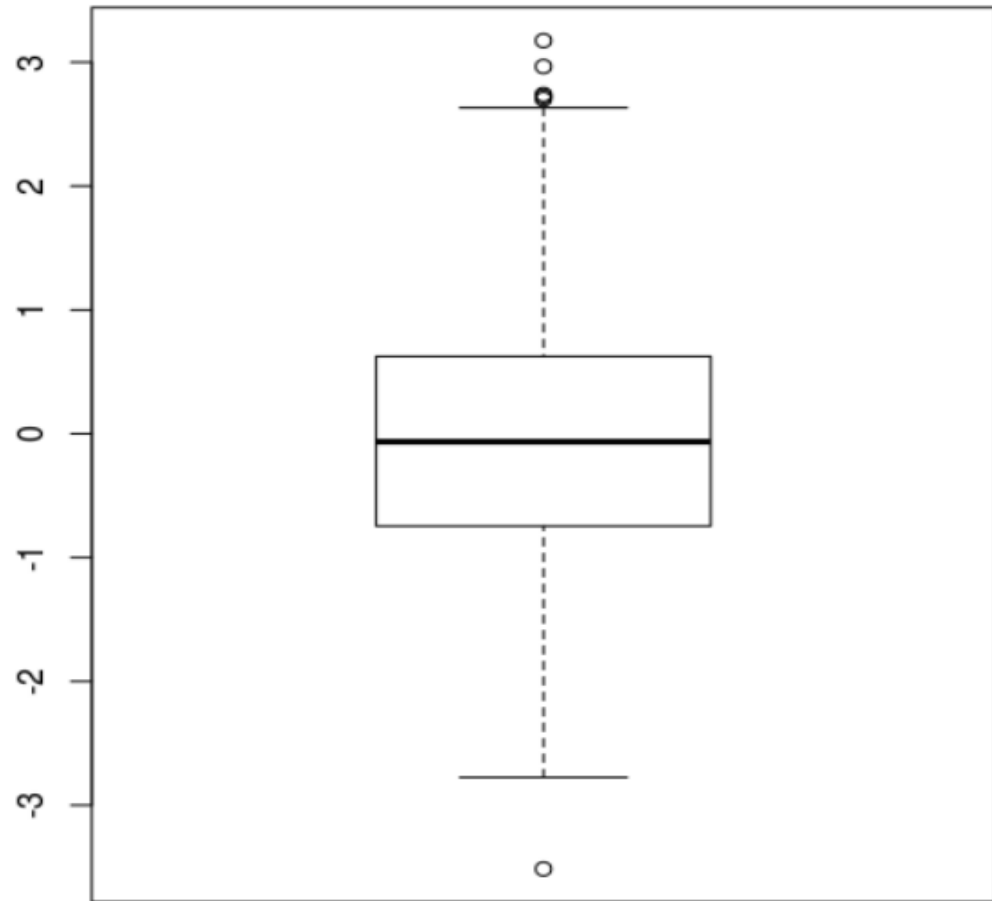
We differentiate between

- Box in the middle
- The lines (“whiskers”)
- Additional points



# Boxplot

- The box in the middle indicates first quartile, median, and third quartile
- The lines (“whiskers”) show the largest or smallest observation that falls within a distance of 1.5 times the box size from the nearest quartile
- If any observations fall farther away, the additional points are considered “extreme” values and are shown separately

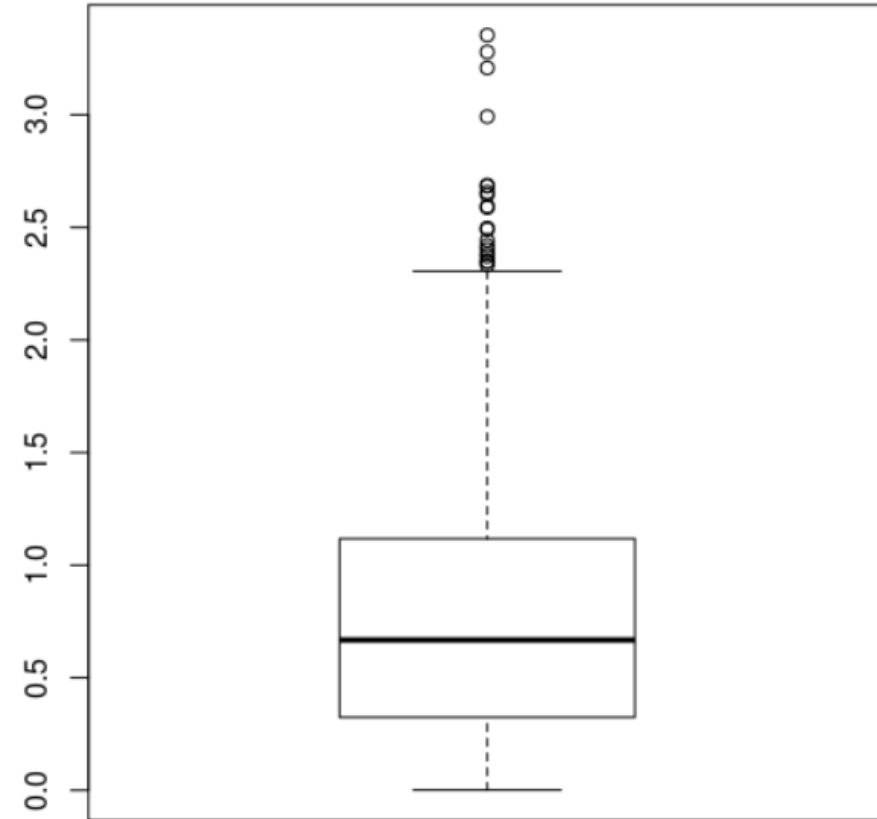
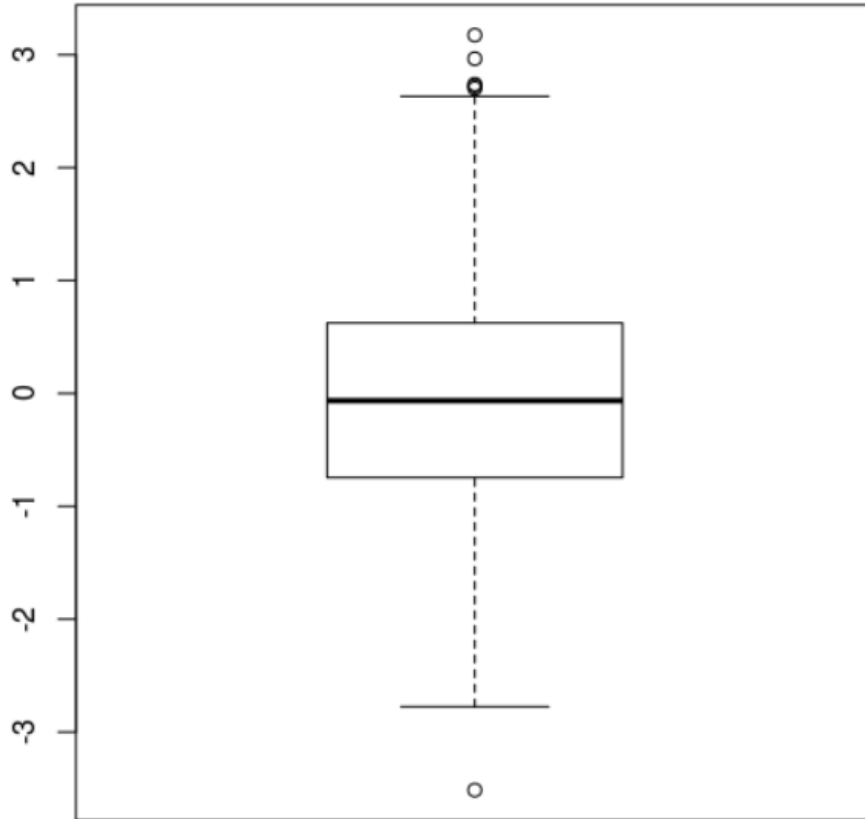


# Boxplot

Let's draw the boxplots of our two random distributions in a figure array

```
> par(mfrow=c(1,2))
> boxplot(rnumbers$rnorm1000)
> boxplot(rnumbers$pos.rnorm1000)
> par(mfrow=c(1,1))
```

# Boxplot



# Boxplot

- In the previous plot we observe differences between both boxplots
- Because of the (automatic chosen) different scales of the y-axis we need to be careful when comparing both plots with each other

We better enforce identical y-axis limits in both plots with ? to better identify group differences. How?

>

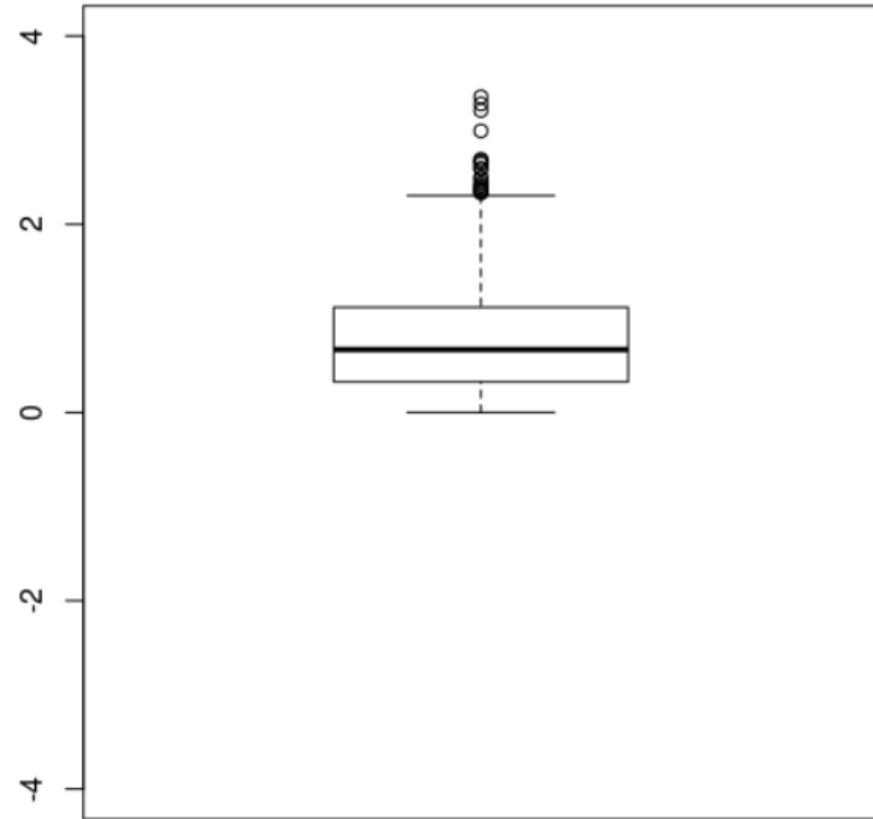
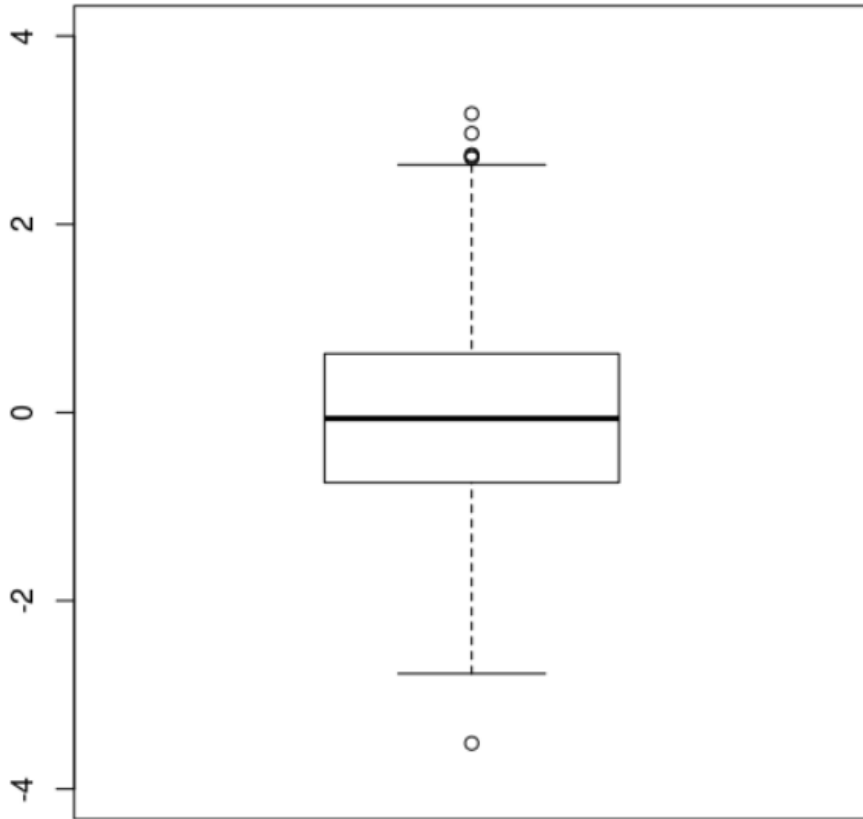
# Boxplot

- In the previous plot we observe differences between both boxplots
- Because of the (automatic chosen) different scales of the y-axis we need to be careful when comparing both plots with each other

We better enforce identical y-axis limits in both plots with `ylim` to better identify group differences

```
> par(mfrow=c(1,2))
> boxplot(rnumbers$rnorm1000, ylim=c(-4,4))
> boxplot(rnumbers$pos.rnorm1000, ylim=c(-4,4))
> par(mfrow=c(1,1))
```

# Boxplot



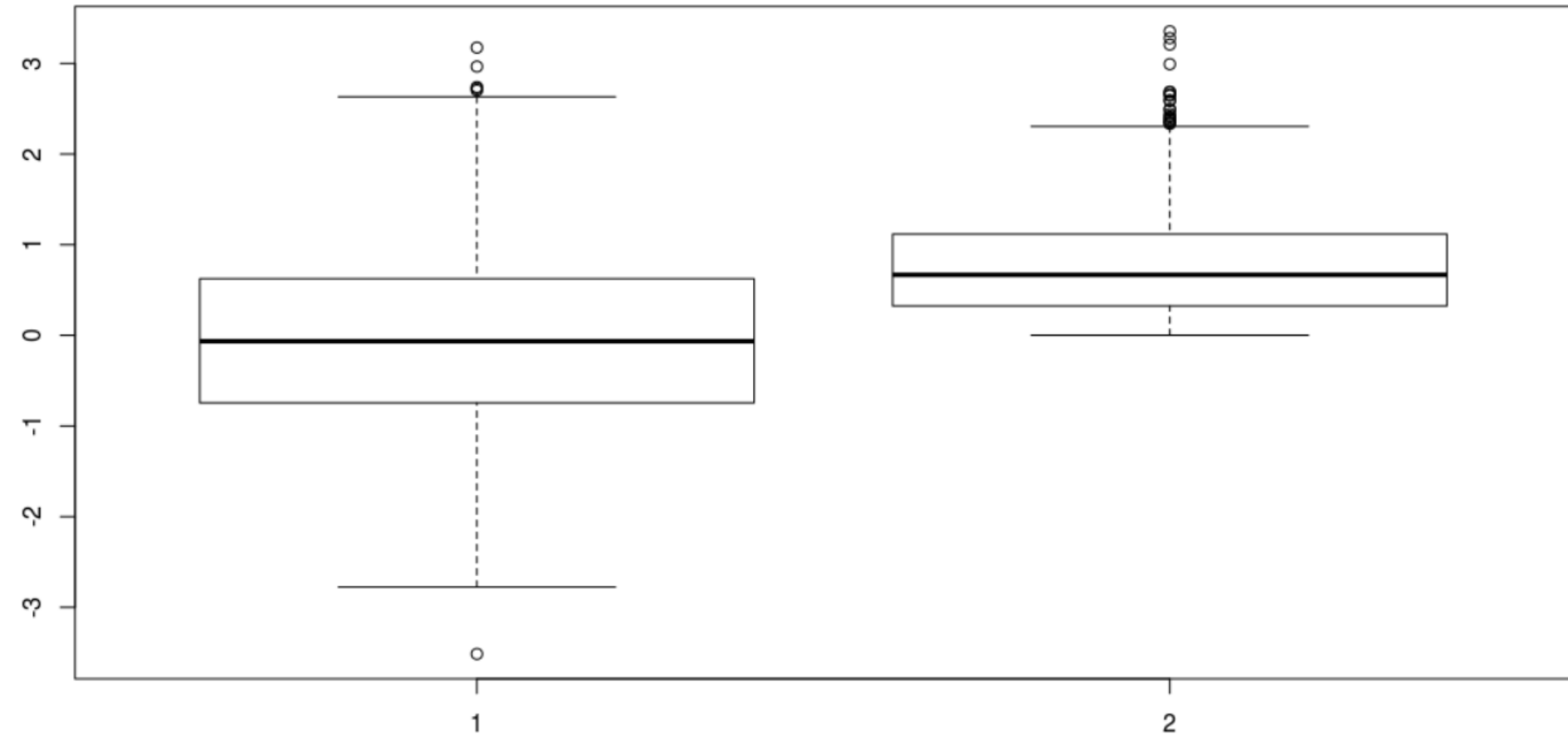
# Boxplot

- For boxplots, there is an alternative way to plot a set of boxplots in the same frame

With the alternative way there is no need to specify axis limits which is in particular useful when comparing more than two groups

```
> boxplot(rnumbers$rnorm1000, rnumbers$pos.rnorm1000)
```

# Boxplot



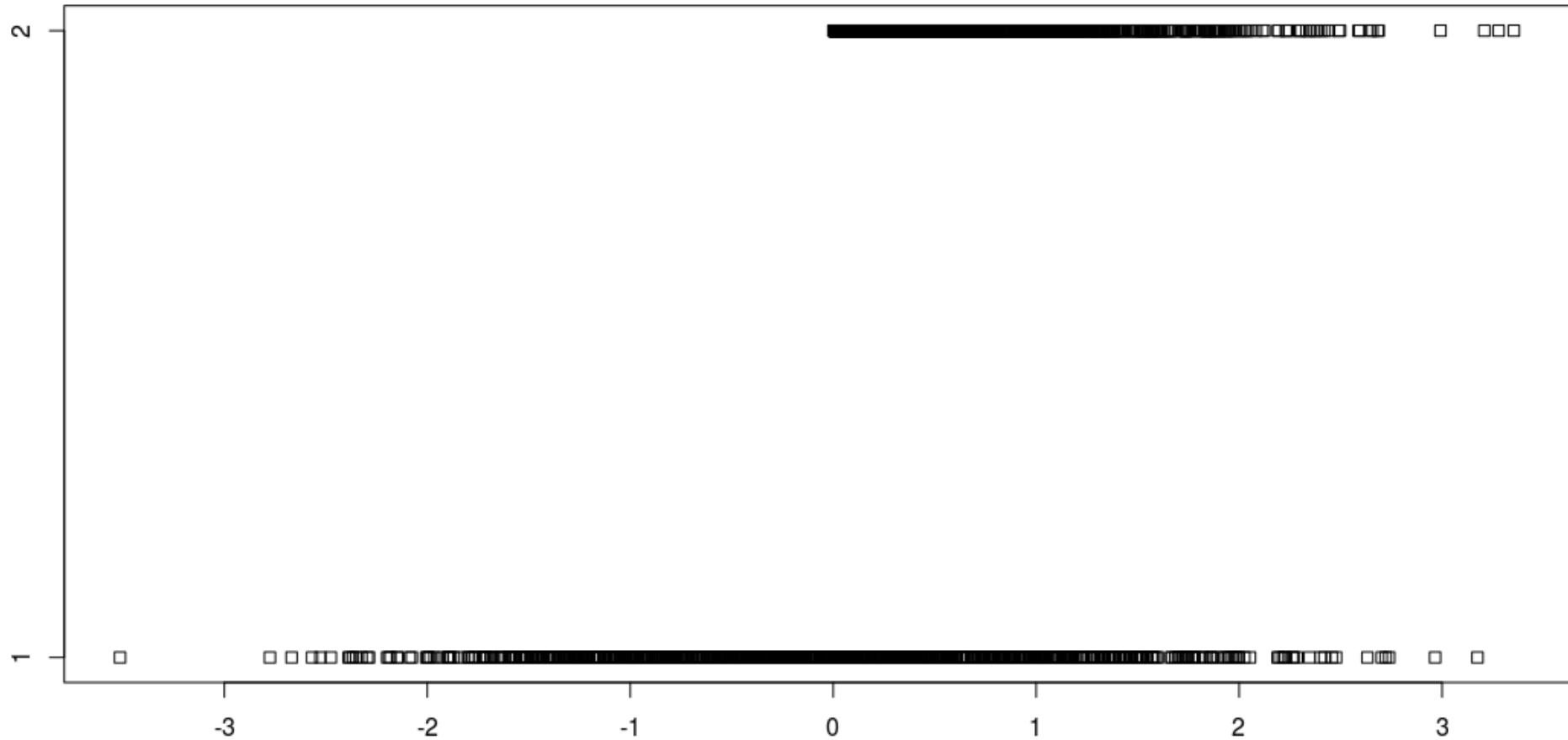
# Stripcharts

- Stripcharts produces one dimensional scatter plots (or dot plots) of our data
- Each data point is marked with a dot on a number line

Let's draw the stripchart of our two random distributions

```
> stripchart(list(rnumbers$rnorm1000,
rnumbers$pos.rnorm1000))
```

# Stripcharts



# Stripcharts

```
> help("stripchart")
stripchart(x, method = "overplot", jitter = 0.1, offset = 1/3,
 vertical = FALSE, group.names, add = FALSE,
 at = NULL, xlim = NULL, ylim = NULL,
 ylab = NULL, xlab = NULL, dlab = "", glab = "",
 log = "", pch = 0, col = par("fg"), cex = par("cex"),
 axes = TRUE, frame.plot = axes, ...)
```

**x**

the data from which the plots are to be produced. In the default method the data can be specified as a **single numeric vector**, or as **list of numeric vectors**, each corresponding to a component plot.

**method**

the method to be used to separate coincident points. The default method "overplot" causes such points to be overplotted, but it is also possible to specify "jitter" to jitter the points, or "stack" have coincident points stacked. The last method only **makes sense for very granular data**.

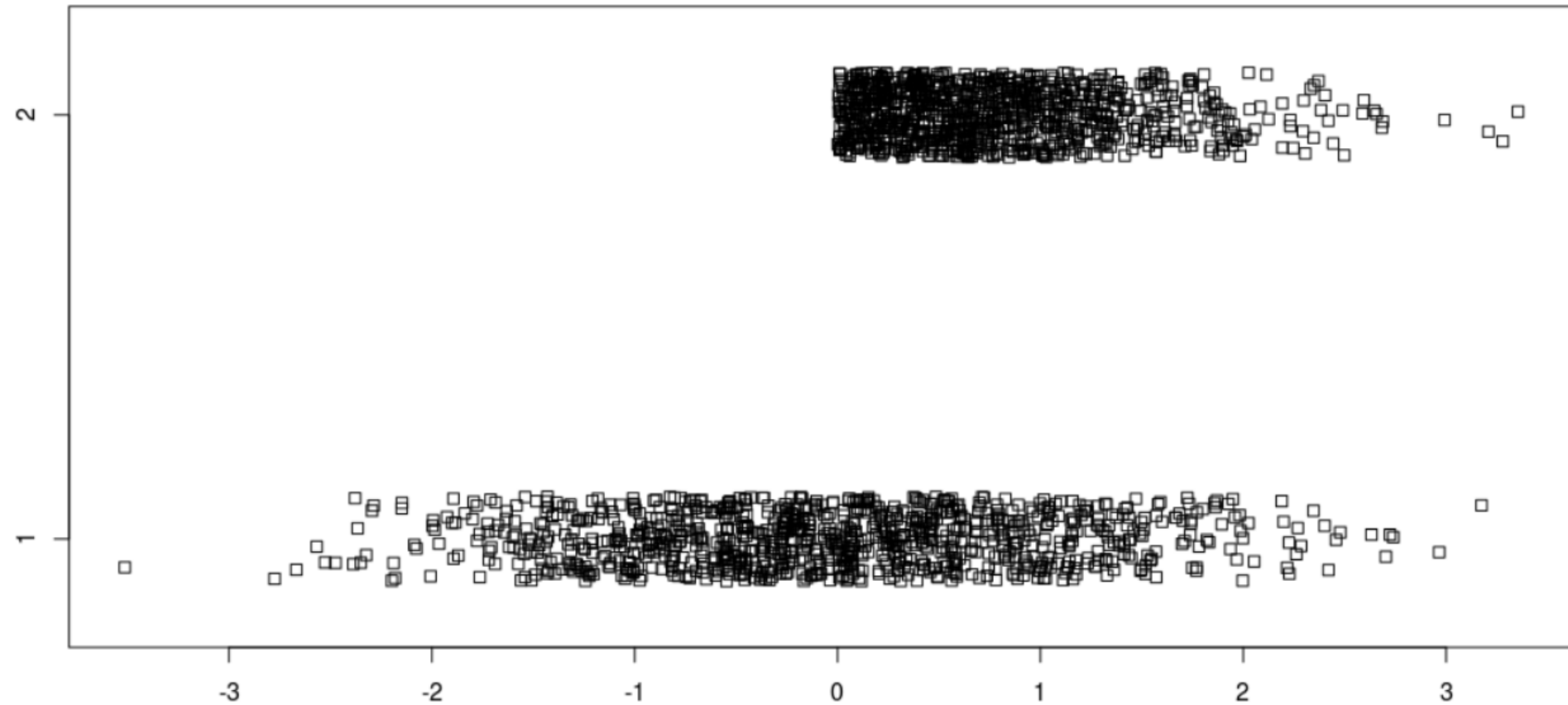
# Stripcharts

- From the previous plot we observe the differences between the two distributions
- The problem is that some points become invisible because they are overplotted

A helpful tool is the jitter method which offsets all points a random amount vertically

```
> stripchart(list(rnumbers$rnorm1000,
rnumbers$pos.rnorm1000), method="jitter")
```

# Stripcharts



# Stripcharts Quiz

- Let's play our plot
- Color the first set of points with **red** and the second set of points with **blue**
- Use default method overplot, but make the points smaller so more visible

```
> stripchart(list(rnumbers$rnorm1000,
rnumbers$pos.rnorm1000), ?)
```

# Pie charts

- Pie charts are an alternative to barplots

## Usage

```
pie(x, labels = names(x), edges = 200, radius = 0.8,
 clockwise = FALSE, init.angle = if(clockwise) 90 else 0,
 density = NULL, angle = 45, col = NULL, border = NULL,
 lty = NULL, main = NULL, ...)
```

## Arguments

|            |                                                                                                                                                                                                                                      |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x          | a vector of non-negative numerical quantities. The values in x are displayed as the areas of pie slices.                                                                                                                             |
| labels     | one or more expressions or character strings giving names for the slices. Other objects are coerced by <a href="#">as.graphicsAnnot</a> . For empty or NA (after coercion to character) labels, no label nor pointing line is drawn. |
| edges      | the circular outline of the pie is approximated by a polygon with this many edges.                                                                                                                                                   |
| radius     | the pie is drawn centered in a square box whose sides range from -1 to 1. If the character strings labeling the slices are long it may be necessary to use a smaller radius.                                                         |
| clockwise  | logical indicating if slices are drawn clockwise or counter clockwise (i.e., mathematically positive direction), the latter is default.                                                                                              |
| init.angle | number specifying the <i>starting angle</i> (in degrees) for the slices. Defaults to 0 (i.e., '3 o'clock') unless clockwise is true where init.angle defaults to 90 (degrees), (i.e., '12 o'clock').                                 |

# Pie charts

- Pie charts are an alternative to barplots
- We first start with the word list example
- Next we consider another example on caffeine consumption and marital status
- Next we consider an example on sales data

# Recap: word frequency in Wikipedia

- Since the for loop allowed us to iterate through large texts, we applied our function `findwords` to a Wikipedia article
- We selected the Wikipedia article about the R programming language
- We applied `findwords` and we created a barplot of the 10 most frequent words

# Recap: function `findwords`

```
finds locations of each word in file
findwords <- function(file, sort.by.freq = F)
{
 # fill word.vec from data in file
 word.vec <- scan(file, "")

 # initialize word list
 word.list <- list()

 # iterate through word vector
 for(i in 1:length(word.vec))
 {
 # store current word in variable word
 word <- word.vec[i]
 # add current word to word.list
 word.list[[word]] <- c(word.list[[word]], i)
 }
 # sort by word frequency or else sort alphabetically
 if(sort.by.freq) {return(word.list[order(sapply(word.list, length),
decreasing = T))]} else {return(word.list[sort(names(word.list))])}
}
```

# Recap: word frequency in Wikipedia

```
> word.list <- findwords("R_wikipedia.txt",
sort.by.freq=T)
Read 3395 items
```

```
> word.freq <- sapply(word.list, length)
```

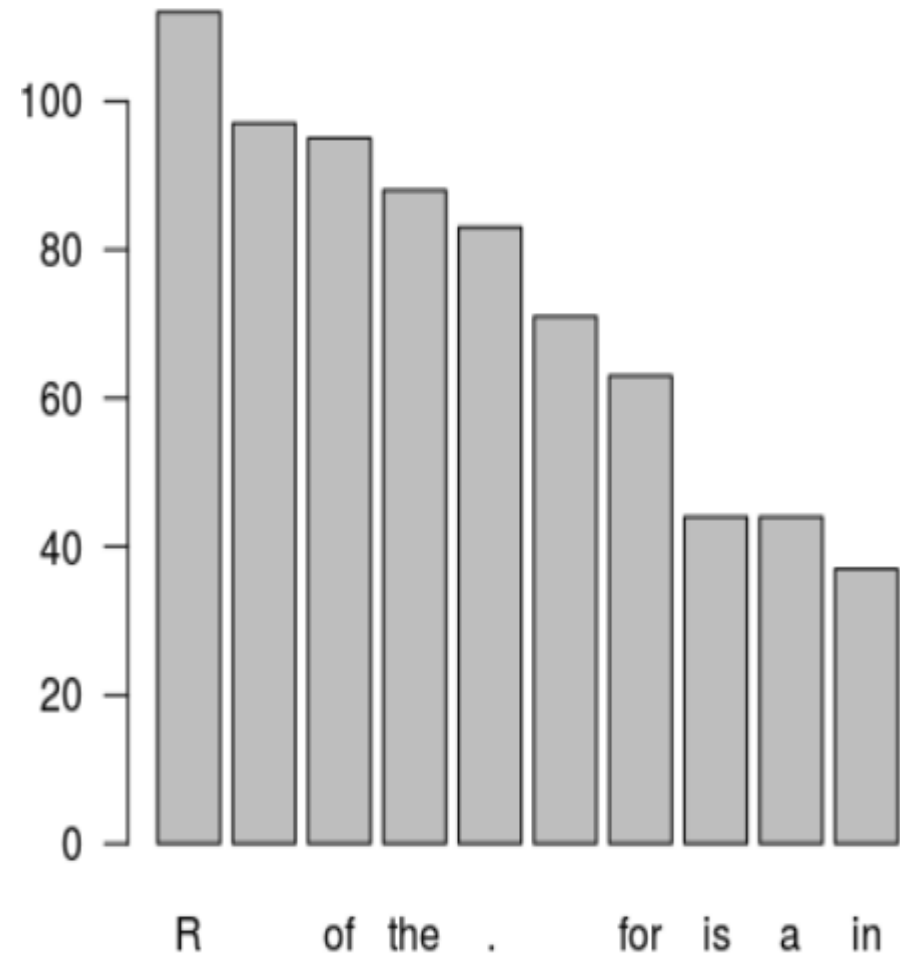
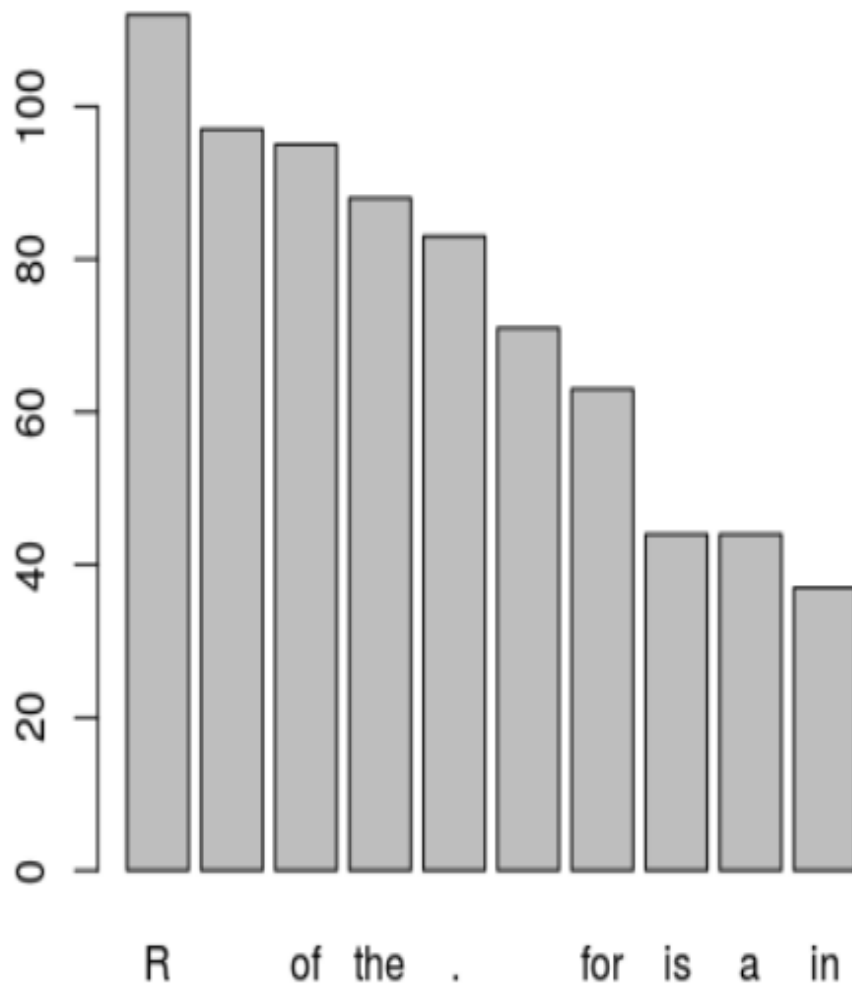
```
> word.freq[1:10]
```

|     |           |    |     |    |
|-----|-----------|----|-----|----|
| R   | and       | of | the | .  |
| 91  | 89        | 85 | 75  | 57 |
| for | Retrieved | is | a   | in |
| 51  | 49        | 35 | 35  | 32 |

```
> barplot(word.freq[1:10], las=0)
```

Rotate the labels on the y axis by adding "las = 1" as an argument. `las` can be 0, 1, 2 or 3.

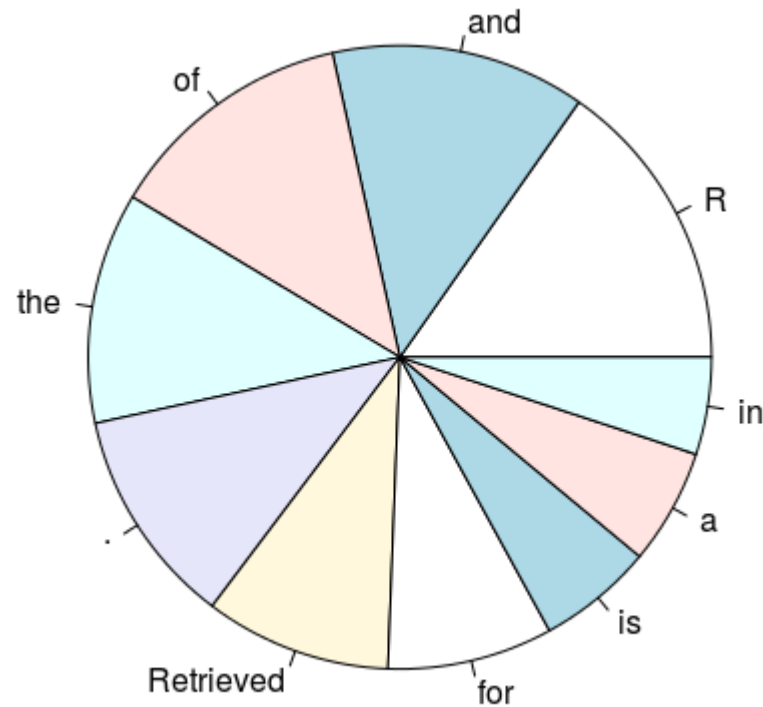
# Recap: word frequency in Wikipedia



# Pie chart of word frequency

We create a pie chart of word frequency with

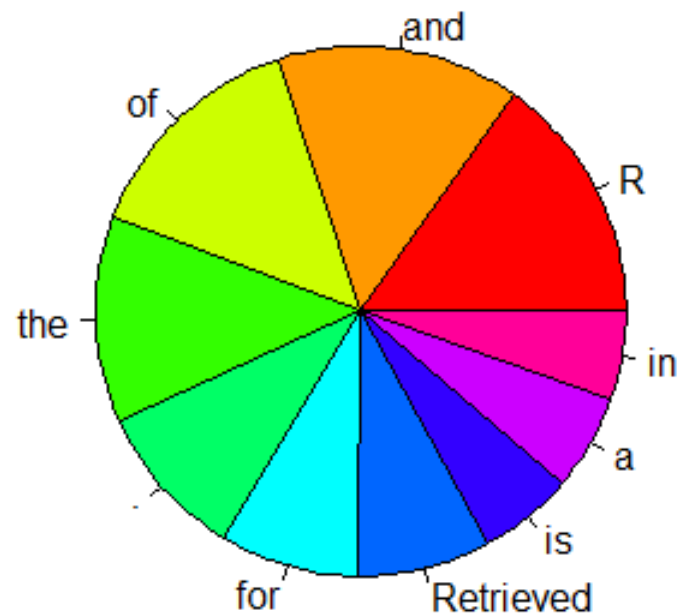
```
> pie(word.freq[1:10])
```



# Pie chart of word frequency

In order to prevent duplicate colors, we specify a rainbow color palette

```
> pie(word.freq[1:10], col=rainbow(10))
```



```
> pie(word.freq[1:3], col=c("red", "blue", "green"))
```

```
> pie(word.freq[1:8], col=rgb(c(1:8)/8, 1, 1))
```

# Caffeine consumption and marital status

- We create pie charts from data on caffeine consumption and marital status
- As a first step we create a matrix which contains data on caffeine consumption and marital status
- Next, we draw several pie charts

# Recap: matrix creation

- In R, a matrix is a vector with two additional attributes, the number of rows and number of columns

One of the ways to create a matrix is via the `matrix` function to obtain a matrix from a given data vector with `nrow` number of rows and `ncol` number of columns

```
> y <- matrix(c(1,2,3,4),nrow=2,ncol=2)
> y
```

|       | [, 1] | [, 2] |
|-------|-------|-------|
| [1, ] | 1     | 3     |
| [2, ] | 2     | 4     |

# Recap: matrix row names and column names

We can provide names for the rows and columns of a matrix

```
> y
 [,1] [,2]
[1,] 1 3
[2,] 2 4

> rownames(y) <- c("Row1", "Row2")
> colnames(y) <- c("Col1", "Col2")
```

```
> y
 Col1 Col2
Row1 1 3
Row2 2 4
```

# Caffeine consumption and marital status

```
> caff.marital
```

|              | 0   | 1-150 | 151-300 | >300 |
|--------------|-----|-------|---------|------|
| Married      | 652 | 1537  | 598     | 242  |
| Prev.married | 36  | 46    | 38      | 21   |
| Single       | 218 | 327   | 106     | 67   |

- Data consists of four levels of caffeine consumption
- Marital status has three categories
- The matrix represents the number of women in each category and level

# Caffeine consumption and marital status

We create our matrix as follows:

```
> caff.marital <-
matrix(c(652,1537,598,242,36,46,38,21,218,327,1
06,67),nrow=3,byrow=T)

> colnames(caff.marital) <- c("0","1-150","151-
300", ">300")

> rownames(caff.marital) <-
c("Married","Prev.married","Single")
```

# Recap: matrix indexing

We can access more than a single column/row/element at once

```
> z <- matrix(c(1:9), nrow=3)
```

```
> z
```

|       | [, 1] | [, 2] | [, 3] |
|-------|-------|-------|-------|
| [1, ] | 1     | 4     | 7     |
| [2, ] | 2     | 5     | 8     |
| [3, ] | 3     | 6     | 9     |

Select columns 2 and 3

```
> z[, 2:3]
```

|       | [, 1] | [, 2] |
|-------|-------|-------|
| [1, ] | 4     | 7     |
| [2, ] | 5     | 8     |
| [3, ] | 6     | 9     |

# Recap: matrix indexing

We can access more than a single column/row/element at once

```
> z <- matrix(c(1:9), nrow=3)
```

```
> z
```

|       | [, 1] | [, 2] | [, 3] |
|-------|-------|-------|-------|
| [1, ] | 1     | 4     | 7     |
| [2, ] | 2     | 5     | 8     |
| [3, ] | 3     | 6     | 9     |

Select columns 2 and 3

```
> z[, c(2, 3)]
```

|       | [, 1] | [, 2] |
|-------|-------|-------|
| [1, ] | 4     | 7     |
| [2, ] | 5     | 8     |
| [3, ] | 6     | 9     |

## Recap: matrix indexing

## Select first and second row

$$\mathbf{z} = [z_1, z_2, z_3, z_4, z_5, z_6, z_7]$$
$$[\cdot, 1] \quad [\cdot, 2] \quad [\cdot, 3]$$
$$[1, \quad 1 \quad 4 \quad 7]$$

|         |   |   |   |
|---------|---|---|---|
| $[2, ]$ | 2 | 5 | 8 |
|---------|---|---|---|

## Select third column of first and second row

$$Z = \begin{bmatrix} ? & ? & ? & ? & ? & ? & ? & ? \end{bmatrix}$$

[1] 7 8

# Recap: matrix indexing

Select first and second row

```
> z[c(1,2),]
 [,1] [,2] [,3]
[1,] 1 4 7
[2,] 2 5 8
```

Select third column of first and second row

```
> z[c(1,2), 3]
[1] 7 8
```

# Caffeine consumption and marital status

We can select a category from our matrix in different ways

```
> caff.marital
```

|              | 0   | 1-150 | 151-300 | >300 |
|--------------|-----|-------|---------|------|
| Married      | 652 | 1537  | 598     | 242  |
| Prev.married | 36  | 46    | 38      | 21   |
| Single       | 218 | 327   | 106     | 67   |

```
> caff.marital[1,]
```

|  | 0   | 1-150 | 151-300 | >300 |
|--|-----|-------|---------|------|
|  | 652 | 1537  | 598     | 242  |

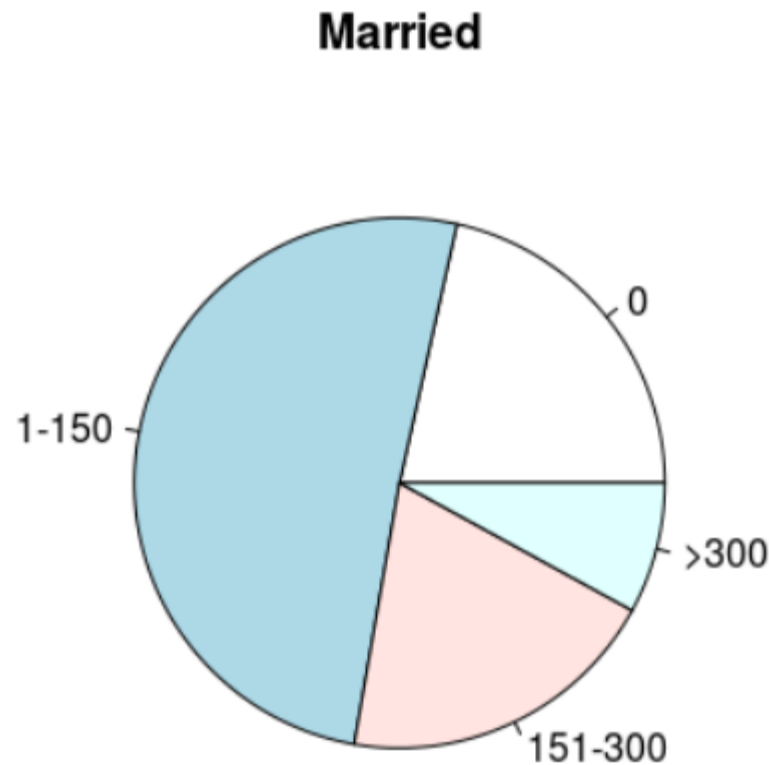
```
> caff.marital["Married",]
```

|  | 0   | 1-150 | 151-300 | >300 |
|--|-----|-------|---------|------|
|  | 652 | 1537  | 598     | 242  |

# Pie chart of caffeine consumption

We draw a single pie chart of the first category

```
> pie(caff.marital["Married",], main="Married")
```



# Pie chart of caffeine consumption

We use a figure array to draw the pie charts of all three categories into one plot

```
> par(mfrow=c(1,3))

> pie(caff.marital["Married",], main="Married")

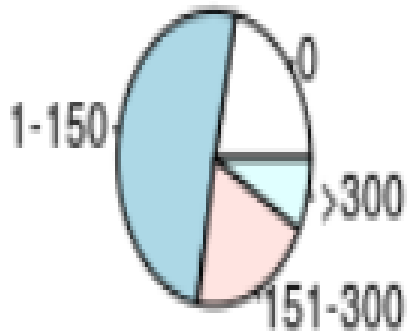
> pie(caff.marital["Prev.married",],
main="Previously married")

> pie(caff.marital["Single",], main="Single")

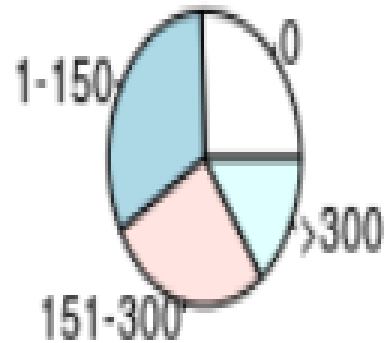
> par(mfrow=c(1,1))
```

# Pie chart of caffeine consumption

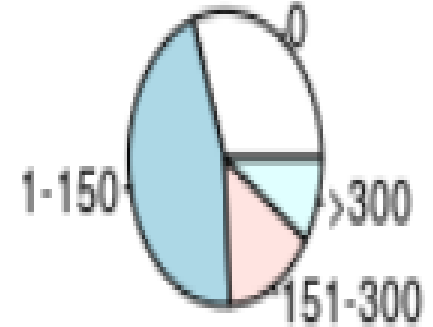
**Married**



**Previously married**



**Single**



# Sales data

Let's consider another example on sales data

```
> sales <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
```

```
> names(sales) <- c("Blueberry", "Cherry",
"Apple", "Boston Cream", "Other", "Vanilla
Cream")
```

```
> sales
```

| Blueberry    | Cherry | Apple         |
|--------------|--------|---------------|
| 0.12         | 0.30   | 0.26          |
| Boston Cream | Other  | Vanilla Cream |
| 0.16         | 0.04   | 0.12          |

# Sales data Quiz

Let's consider another example on sales data

```
> sales <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
```

```
> for (..) {
```

```
..
```

```
..
```

```
}
```

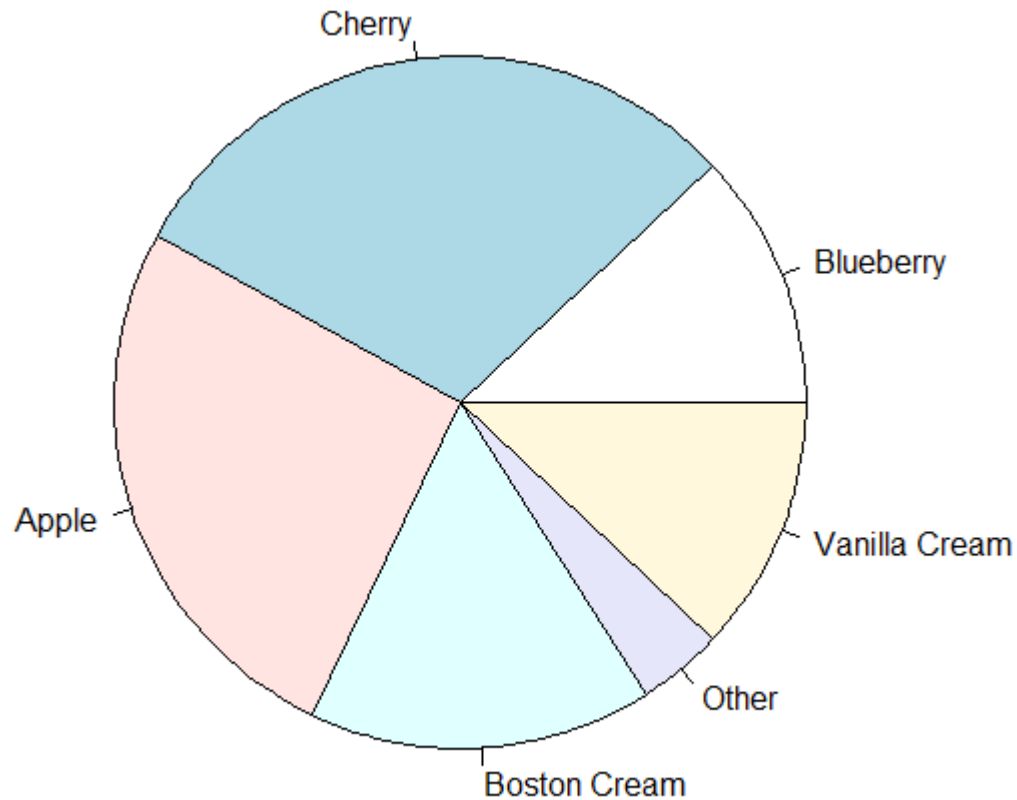
```
> sales
```

|           |           |           |
|-----------|-----------|-----------|
| 1st fruit | 2nd fruit | 3rd fruit |
| 0.12      | 0.30      | 0.26      |
| 4th fruit | 5th fruit | 6th fruit |
| 0.16      | 0.04      | 0.12      |

# Pie chart of sales data

Like before we create a default pie chart

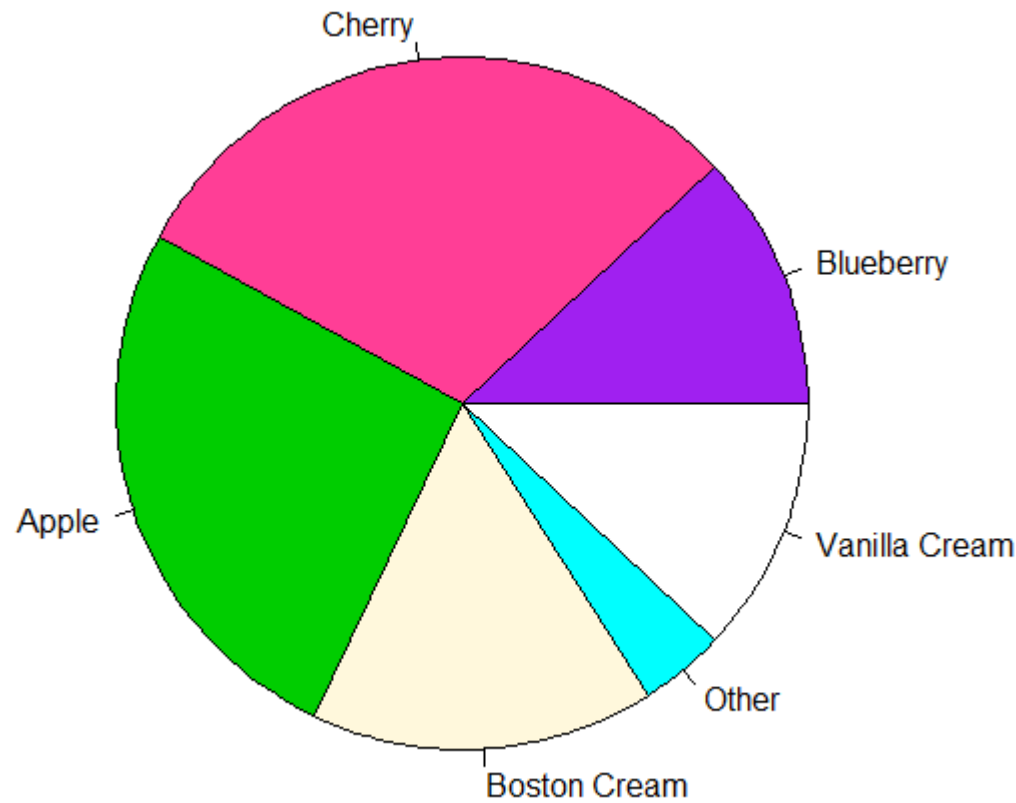
```
> pie(pie.sales)
```



# Pie chart of sales data

We can provide explicit color names

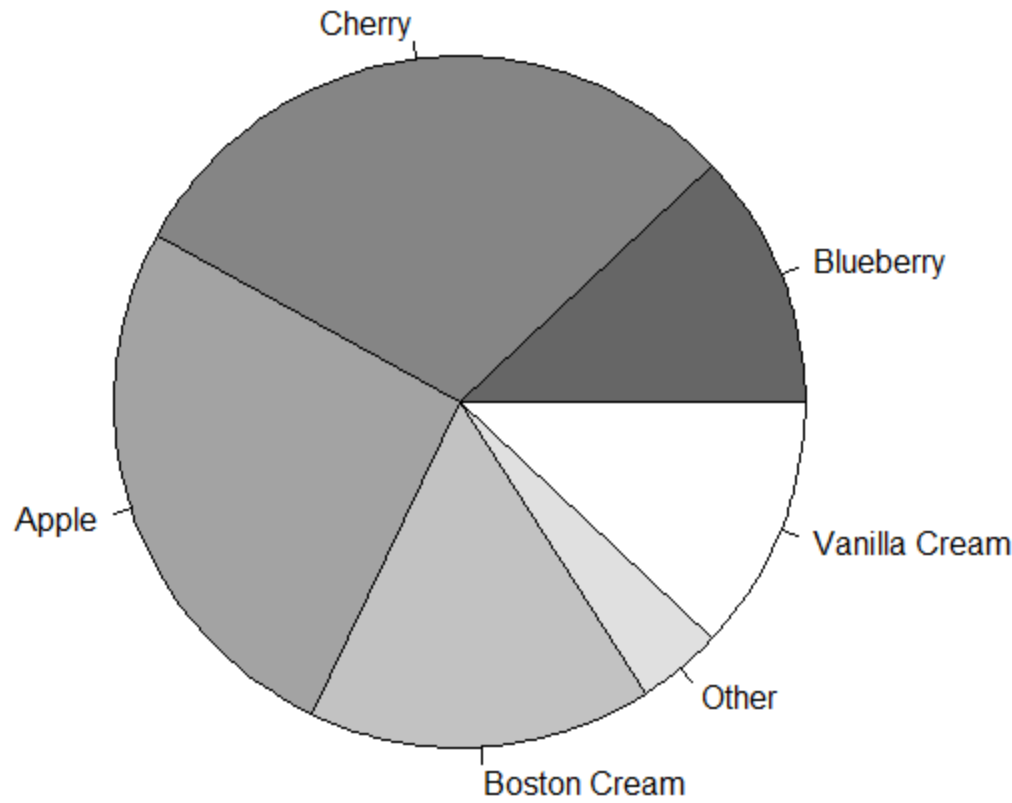
```
> pie(sales, col = c("purple", "violetred1",
"green3", "cornsilk", "cyan", "white"))
```



# Pie chart of sales data

We may want to have it in a gray as well

```
> pie(sales, col = gray(seq(0.4, 1.0, length = 6)))
```



# Homework



1. Use the data on body height and weight from previous homework. Create boxplots from body height, weight and BMI.
2. Create a pie chart of the 10 most frequent used words in your favorite Wikipedia article.
3. Create your own sales data matrix which consists of 10 products. Create pie charts in rainbow colors and in gray.