Load data from a file "experimentResult.Rdata".
Have a look on an object `experiment`. It contains results of an experiment: bacteria samples from three different locations were grown on Petri dishes in 4 different temperature settings.
The colonies were counted on 4 consecutive days, 3 times for each dish (colonies1, colonies2, colonies3) and number of colonies per 1 cm^2 was calculated. Unfortunately, not every dish was counted 3 times. Experiments lacking additional measurements are marked as `not recounted`.

1. Make a copy of experiment data frame, and work on it to not influence the original data you loaded.
2. Clean up the data. Column with entry number is unnecessary, as it doubles information from row number. Information about field should be next to day/temperature information.
3. Density of "colonies" was computed by dividing number of colonies by the dish surface. Therefore they should have at most 2 digits after "."
4. Change Fahrenheit into Celsius °C = (°F - 32) * 5/9
5. Take only rows with colonies counted 3 times.
6. Take only rows with incubation temperature greater than 0 °C and smaller than 38°C.
7. Calculate average of three colony counts and add it as a last column
8. Calculate mean of average colony count and sd by day of experiment.
9. Calculate mean of average colony count and sd by temperature.

```
load("experimentResult.Rdata")
ls( )
class(experiment)
head(experiment)
dim(experiment)

cpExperiment <- experiment
cpExperiment <- cpExperiment[,c(2:8)]

cpExperiment <- cpExperiment[  ,c("field", "inc.tempF",
"day","colonies", "colonies2", "colonies3", "recounted")]
cpExperiment$colonies3 <- round(cpExperiment$colonies3, 2)


cpExperiment[,3] <- (incTempC <- (cpExperiment$inc.tempF -
32)*5/9)
cpExperiment <- cpExperiment[cpExperiment$recounted == "y", ]

dim(cpExperiment)


cpExperiment[ cpExperiment$incTempC > 0 &
cpExperiment$incTempC < 38 ,  ]

cpExperiment <- cpExperiment[ (cpExperiment[ ,2] > 0) &
(cpExperiment[,2]< 38 ) & cpExperiment$recounted == "y",  ]

cpExperiment$mean <-apply(cpExperiment[
,c("colonies","colonies2","colonies3")], 1, mean)

tapply(cpExperiment$mean,cpExperiment$day,mean)
tapply(cpExperiment$mean,cpExperiment$day,sd)
```

Subsetting a dataframe and adding/changing values in columns may be done also with different syntax, with functions subset and transform.

```
subset(dataframe, subset=logical expression, select=columns to
select)

data(bacteria,package="MASS")
```

Only placebo samples:

```
subset(bacteria, trt=="placebo")
```

Only placebo samples, data from at least fourth week of treatment:

```
subset(bacteria,week>4&trt=="placebo")
```

Only placebo samples, data from 4-11 week of treatment:

```
subset(bacteria,week> 4 & week<11 & trt=="placebo")
```

Argument select specifies columns in the output:

```
subset(bacteria,week>4&trt=="placebo",select=3:5)
subset(bacteria,week>4&trt=="placebo",select=week:trt)
```

Transforming a data frame

```
transform(dataframe, columnname=new.values)

transform(bacteria, days=7*week)
```

Use subset and transform to rewrite your code for 2-7.

R has a very small set of buit-in constants. Look what you get with:

```
pi
letters
LETTERS
month.name
```

Other special values are: infinity (+,-)

```
>1/0
> -1/0
> Inf+1
> 0/Inf
```

Numerical result undefined: NaN

```
> Inf-Inf
> sqrt(-3)
```

Missing/not available values: NA
Each calculation with NA gives NA --> beware when using sum or other function

```
> 1+NA
> a<-1:4
> a[13]<-2
> a
 [1]  1  2  3  4 NA NA NA NA NA NA NA NA  2
> sum(a)
> max(a)
> ?sum
> sum(a,na.rm=T)
```

Graphics

Plotting commands are divided into three basic groups:
High-level plotting functions create a *new plot* on the graphics device, possibly with axes, labels, titles and so on.
Low-level plotting functions add more information to an existing plot, such as extra points, lines and labels.
Interactive graphics functions allow you interactively add information to, or extract information from, an existing plot, using a pointing device such as a mouse.
In addition, R maintains a list of graphical parameters, which can be manipulated to customize your plots.

`plot()` function. This is a generic function: the type of plot produced is dependent on the type or class of the first argument.
`plot(x, y)` produces a scatterplot of y against x
`plot(x)` a scatterplot of x against index of x

`plot(factor)` barplot
`plot(factor, y)` boxplot
`plot(data frame)` each variable against the others

More specific functions:
`barplot(vector), barplot(matrix)`
`boxplot(formula), boxplot(data frame)`
`hist(data, bins), hist(data, breaks)` It computes a histogram of the data. Results are returned as an object of class histogram and by default a histogram is plotted. Histogram may be added to an existing plot with argument add=TRUE.
Examples of function usage from R help may be seen with `example(function)`. Use it to see examples of `dotchart()` and `image()`.

Arguments for high-level plotting functions.

`add=TRUE`
: Forces the function to act as a low-level graphics function, superimposing the plot on the current plot (some functions only).

`axes=FALSE`
: Suppresses generation of axes—useful for adding your own custom axes with the axis() function. The default, axes=TRUE, means include axes.

`log="x"`
`log="y"`
`log="xy"`
: Causes the x, y or both axes to be logarithmic. This will work for many, but not all, types of plot.

`type=`
The `type=` argument controls the type of plot produced, as follows:
`type="p"` Plot individual points (the default)
`type="l"` Plot lines
`type="b"` Plot points connected by lines (both)
`type="o"` Plot points overlaid by lines
`type="h"` Plot vertical lines from points to the zero axis (high-density)
`type="s"`
`type="S"` Step-function plots. In the first form, the top of the vertical defines the point; in the second, the bottom.
`type="n"` No plotting at all. However axes are still drawn (by default) and the coordinate system is set up according to the data. Ideal for creating plots with subsequent low-level graphics functions.

`xlab=`*string*
`ylab=`*string* Axis labels for the x and y axes. Use these arguments to change the default labels, usually the names of the objects used in the call to the high-level plotting function.
`main=`*string* Figure title, placed at the top of the plot in a large font.
`sub=`*string* Sub-title, placed just below the x-axis in a smaller font.

Low-level plotting functions

May be used only when a plot is already iniciated with high-level plotting function.

```
points(x, y)
lines(x, y)
```
Adds points or connected lines to the current plot.

```
text(x, y, labels, ...)
```
Add text to a plot at points given by x, y.
Note: This function is often used in the sequence

```
> plot(x, y, type="n"); text(x, y, names)
```

The graphics parameter `type="n"` suppresses the points but sets up the axes, and the `text()` function supplies special characters, as specified by the character vector `names` for the points.

```
abline(a, b)
abline(h=y)
abline(v=x)
```

Adds a line of slope b and intercept a to the current plot. h=y may be used to specify y-coordinates for the heights of horizontal lines to go across a plot, and v=x similarly for the x-coordinates for vertical lines.

```
legend(x, y, legend, ...)
title(main, sub)
```

The `par()` function is used to access and modify the list of graphics parameters for the *current graphics device*.

`par()`
Without arguments, returns a list of all graphics parameters and their values for the current device.

`par(c("col", "lty"))`
With a character vector argument, returns only the named graphics parameters (here: default color and line type).

`par(col=4, lty=2)`
With named arguments sets the values of the named graphics parameters

Arguments may be also passed directly to the function call, for example cex in `text()`, pch in `points()` etc.

```
pch
col
lwd
lty
cex
```

Multiple figures by a window

`par(mfrow=c(2,3))` splits window into 2 rows, 3 columns. Those are then filled by rows.
`par(mfcol=c(2,3))` split is the same, but fill is done by columns

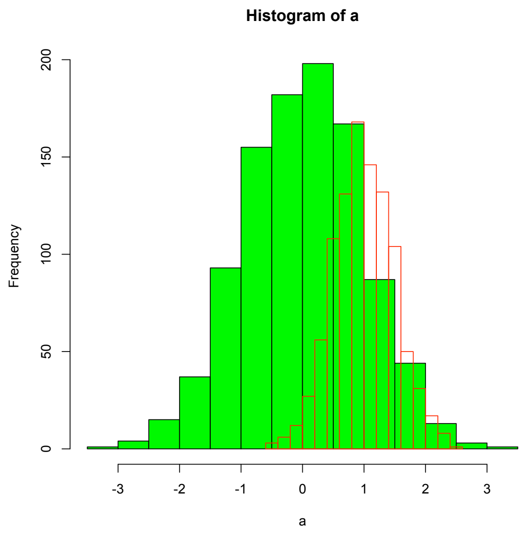
For more complicated cases of multiple figures per page is better to use `layout()` function.

`postscript()` For printing on PostScript printers, or creating PostScript graphics files.
`pdf()` Produces a PDF file
`png()` Produces a bitmap PNG file.
`jpeg()` Produces a bitmap JPEG file.

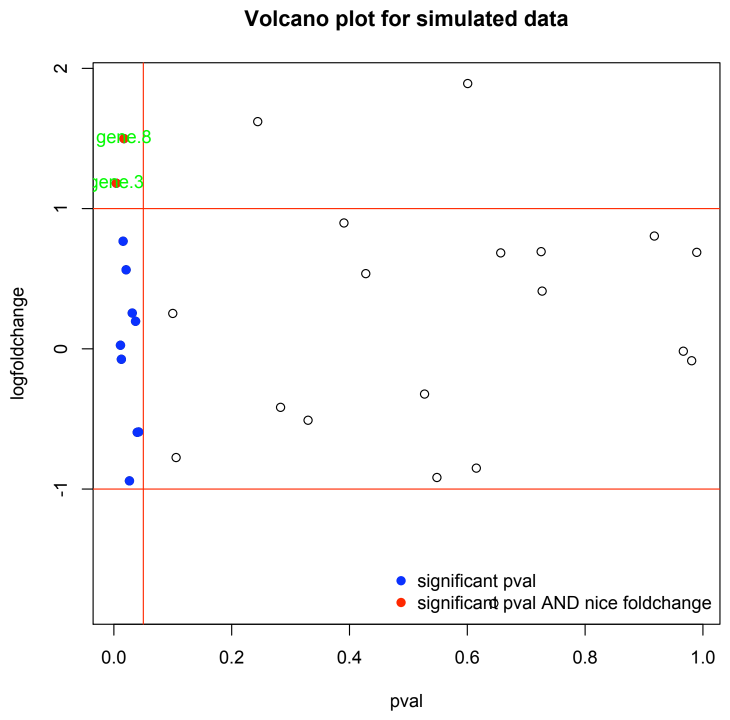
After drawing, the device driver should be closed with

```
 dev.off()
```

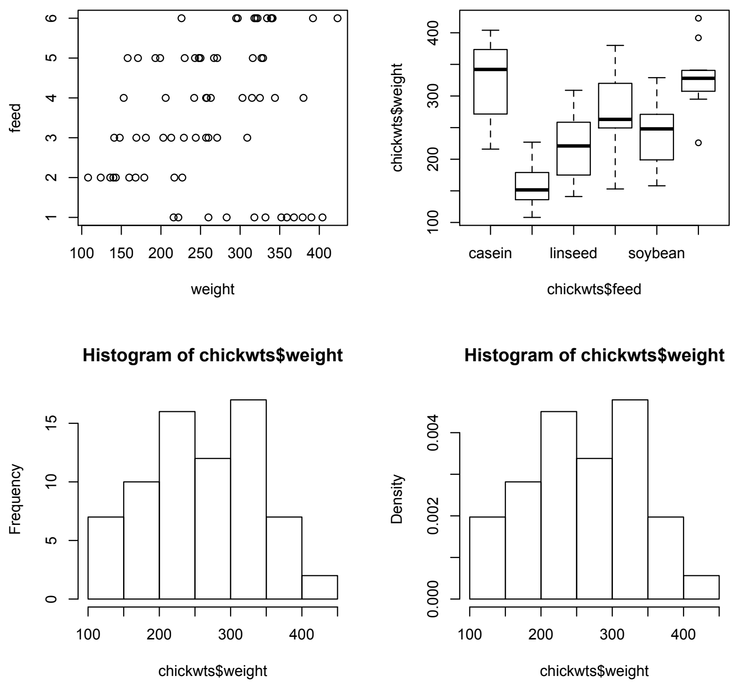
1. Drew 1000 times from a normal distribution with mean 0 and standard deviation of 1. Make a histogram of those data (in green).
   a. Now drew 1000 times from a normal distribution with mean 1 and standard deviation 0.5. Overlay histogram of those data on the previous histogram.
   b. Try to make more bins in the histogram.



2. Plot of gene expresssion data.
   a. Generate test data – a data frame with pvalues and log2 fold change for 30 genes.
      pvalues: 10 values from uniform distribution between 0 and 0.05, 20 values from uniform distribution between 0 and 1
      logfoldchange: 30 values from normal distribution with mean 0 and sd 1
      rownames: gene.x , where x is a number 1:30
   b. Plot foldchange versus pvalue.
   c. Add a red line to identify points with pvalue<0.05
   d. Add two lines to get at least two fold change
   e. Colour blue (just overlay with blue points) for points with pvalue<0.05
   f. Colour red (just overlay with red points) points with pvalue<0.05 and at least two foldchange
   g. Add names of interesting (red) genes.
   h. Add a legend



3. Split the plotting device into 4 fields. Draw 4 plots based on chickwts dataset:
    a. Scatterplot of weight against feed type
    b. Boxplot of weight agains feed type
    c. Histogram with 10 bins
    d. Histogram with 10 bins, with frequency instead of counts



```
1. a<-rnorm(1000)
   b<-rnorm(1000,mean= 1, sd=0.5)

hist(a)col="green")
hist(b,border="red",add=TRUE)
```

To get proper axes, we have to find out the highest counts from a and b.
To see the counts:

```
hist(b)$counts
To avoid hist plotting:
hist(b,plot="n")$counts
max<-max(hist(a,plot="n")$counts,hist(b)$counts,plot="n")
hist(a,col="green",ylim=c(0,max))
hist(b,border="red",add=TRUE)


2. testdata<-data.frame(pval=c(runif(10,0,0.05),runif(20,0,1)),
                 logfoldchange=rnorm(30),
                 row.names=paste("gene",1:30,sep="."))

plot(testdata)

abline(v=0.05,col="red")
abline(h=1,col="red")
abline(h=-1,col="red")
points(testdata[testdata$pval<0.05,],col="blue",pch=19)
points(testdata[testdata$pval<0.05,][abs(testdata$logfoldchang
e)>1,],col="red",pch=19)

Names of genes with pvalue <0.05
rownames(subset(testdata,pval<0.05))
Names of genes with at least two foldchange
rownames(subset(testdata,abs(logfoldchange))>1))
names of genes fulfilling both criteria:
genes<-rownames(subset(testdata,abs(logfoldchange)>1&
pval<0.05))

text(testdata[genes,],labels=genes, col="green")

legend(bty="n",pch=19,col=c("blue","red"),legend=c("significan
t pval","significant pval AND nice foldchange"),"bottomright")

3. par(mfrow=c(2,2))
plot(chickwts)
plot(chickwts$weight~chickwts$feed)
hist(chickwts$weight,breaks=10)
hist(chickwts$weight,breaks=10,freq=F)
```