Applied geostatistics
Exercise 1: Using the R Environment for Statistical Computing

D G Rossiter
University of Twente, Faculty of Geo-Information Science & Earth Observation (ITC)

June 26, 2010

Contents

1 Introduction **1**

2 R basics **2**

3 First interaction with R **4**
3.1 Answers 8

4 Loading additional packages **8**

5 Loading and examining an example dataset **10**
5.1 Answers 12

6 Descriptive statistics **12**
6.1 Summarizing one variable 12
6.2 Finding a specific record in a dataframe 14
6.3 Selecting part of a dataframe 15
6.4 Logical expressions 19
6.5 Answers 22

7 Exploratory graphics **23**
7.1 Univariate exploratory graphics 23
7.2 Answers for univariate exploratory graphics 30

Version 1.61. Copyright © 2007–2010 University of Twente, Faculty ITC All rights reserved. Reproduction and dissemination of the work as a whole (not parts) freely permitted if this original copyright notice is included. Sale or placement on a web site where payment must be made to access this document is strictly prohibited.
To adapt or translate please contact the author (http://www.itc.nl/personal/rossiter).

7.3 Bivariate exploratory graphics 31
7.4 Answers for bivariate exploratory graphics 33
7.5 Visualizing the transect 34
7.6 Answers for visualizing the transect 34

8 Linear modelling **35**
8.1 Answers for linear modelling 37

9 * User-defined functions **38**

10 Quitting R **39**

11 Self-test **41**

References **43**

Index of R concepts **45**

千里之行始于足下 (Qiān lǐ zhī xíng, shǐ yú zú xià)
– 《德经》, 第六十四章 ("Dé Jīng", dì 64 zhāng)
"A journey of a thousand miles begins at the spot underfoot"
– the Dé Jīng, Chapter 64

1 Introduction

In this exercise you will learn the basics of the statistical computing environment that we will use for geostatistical computing. This environment is the powerful, flexible, programmable and almost limitless **R environment for statistical computing and visualisation** [3, 8].

R is an immense program with almost unlimited possibilities; as a result it can be somewhat intimidating at first. Don't worry; with each lesson you will learn more and feel more comfortable.

After completing this first exercise in geostatistical computation you should be able to:

1. Start, interact with, and stop the R program;
2. Give commands to the R program and view the results;
3. Get on-line help for R commands.
4. Load an optional analysis package;
5. Load and examine a dataset provided with R;
6. Compute descriptive statistics;
7. Display some common exploratory graphics;
8. Compute and evaluate a simple linear model.

Beginning with the next exercise we will apply R to geostatistics.

The exercise is organized as a set of **tasks** followed by R code to accomplish the task and the resulting output. The output you see on your system should be the same (except when random numbers are generated). If not, go back and check your work.

Each task ends with a set of **questions** to check your understanding; **answers** and explanations of them are given at the end of each section. You can navigate between questions and answers with hyperlinks.

At the end of the guided exercise you will be given a **self-test task and questions** (§11), with neither code nor answers. You should be able to complete the task and answer the questions with the knowledge you have gained from the exercise. You should submit your answers to the instructor for grading and sample answers.

> **Note:** The code in these exercises was tested with Sweave [4, 5] on R version 2.6.1, `sp` package version 0.9-14, `gstat` package version 0.9-40, `spatial` package version 7.2-38, `lattice` package version 0.17-2, running on Mac OS X 10.4.11. So, the text and graphical output you see here was automatically

generated and incorporated into LaTeX by running actual code through R and its packages. Then the LaTeX document was compiled into the PDF version you are now reading. Your output may be slightly different on different versions and on different platforms.

2 R basics

Task 1 : Start R. •

How you do this is system-specific. On systems running Microsoft Windows, open the executable program `RGui.exe`.

After starting R, you will be looking at a **console** where you interact with R. Figure 1 shows the R for Windows graphical user interface (GUI) console.

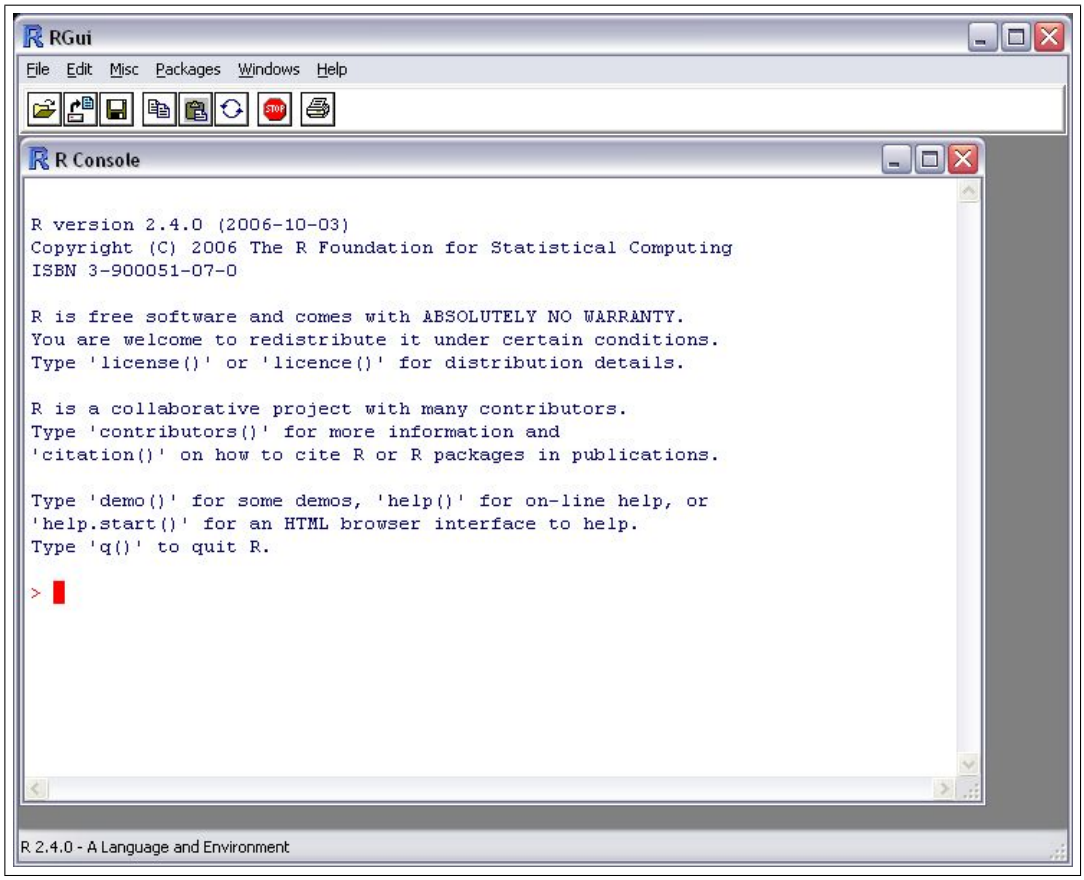


Figure 1: The R for Windows GUI console

At the top of the screen are the usual Windows menus; these have to do with system interaction and overall program control (for example, `File | Exit`) and not with statistical analysis.

Note: R is generally used with a **command line interface** (CLI), which just means that R waits for you to type commands, then executes them. The CLI is certainly intimidating for beginners, since you need to know the names and format of commands before you can get anything done. You learn with exercises like this, from the extensive help system, and from texts and technical notes. The CLI is preferred by power users (i.e. you, after a few lessons), because it allows complete control of calculations, and all options are available. The analyst knows exactly the steps in the analysis. The console log (all commands and results) can be saved to a text file for reporting or re-use. In addition, a CLI makes it easy to write and execute scripts.

You perform most actions in R by typing **commands** in response to a **command prompt**, which usually looks like this:

```
>
```

This `>` is a **prompt symbol** displayed by R, **not** typed by you. This is R's way of telling you it's waiting for you to enter a command.

Type your command and press the `Enter` or `Return` keys; R will **execute** (carry out) your command.

There are several ways to enter R commands:

1. Type them directly at the command prompt `>` as just explained;
2. Cut-and-paste from a document such as this one; make sure *not* to include the prompt `>` or continuation prompt `+` in the text to paste;
3. Cut-and-paste from a text file with R code; all the code in these exercises is provided on the CD in directory `Rcode`; these files have the name of the exercise or part of it, e.g. `ex11`, and extension `.R`. You can open these files in any plain-text editor or in a code editor such as `Tinn-R`;
4. Prepare scripts in a code editor and send them to R from within the editor environment; this depends on the particular code editor.

Sometimes the command will result in numerical output listed in the same console window under your command. Other command result in a graph displayed in a separate window. And some commands just cause R to do what you requested without any feedback.

R will accept the command once it is **syntactically complete**; in particular any parentheses must balance. If try to submit a command to R that is not a complete, R will prompt you to complete it with the **continuation prompt symbol**:

```
+
```

If you decide you don't really want to submit a partially-entered command, press the `Esc` key to cancel.

GUIs for R

Note: There are several groups developing GUI's for R; these are listed at http://www.sciviews.org/_rgui/. Probably the most used is the R Commander [2] written by John Fox; this is described at http://socserv.

`mcmaster.ca/jfox/Misc/Rcmdr/`. This is provided as an optional package, which must be installed and loaded, as explained in §4, below.

3 First interaction with R

At this point you should be looking at R's prompt in the console window:

```
>
```

R is waiting for your command. What can we do here? For a start, we can use R as an **interactive calculator**:

Task 2 : Compute the number of radians in one degree of a circle. •

```
> 2 * pi/360
```

```
[1] 0.017453
```

As this example shows, R has a few built-in constants, among them `pi` for the mathematical constant.[1]

However, we almost always use R **methods**, which are (roughly speaking) **commands** to R. Let's see a simple example of some useful methods and how to put them together.

Suppose we have laid out a transect of 1000 m length across a study area, and we want to sample at 10 random positions on this transect, to a precision of 10 cm. We need to determine the positions of the samples with respect to the start of the transect, to which we assign coördinate 0.

Q1 : *What is the coördinate of the other end of this transect?* *Jump to A1* •

Task 3 : Select a random sample of ten positions along the transect. Round these to the nearest 0.1 m, and present them in order from the beginning to the end of the transect. •

We will first do this one step at a time:

1. Select the random sample;
2. Round each position to one decimal place;
3. Sort the sample in ascending order

After this step-by-step approach, we will see how to do it in one command.

Task 4 : Draw a random sample of ten positions along the transect and print it. •

[1] The Euler constant e must be calculated with the `exp` method as `exp(1)`

A random sample from the uniform distribution (each value equally likely) is drawn with the `runif` method:

```
> runif(10, 0, 1000)

 [1] 428.59 812.95 874.35 336.92 809.75 410.57 730.90 189.42 775.98
[10] 825.80
```

The result is a 10-element **vector** of uniformly-distributed random numbers in the range [0...1000]; this result is printed below the command.

Q2 : *Your results will be different from the ones printed in this note; why?* *Jump to A2* •

This first example of an R method illustrates several important points:

1. R includes a large number of **methods**; in this case: `runif` to generate **r**andom numbers from the **unif**orm distribution;
2. R methods have **arguments** that specify the exact behaviour of the method. The `runif` method has three arguments:
   (a) The **number of random numbers** to produce (here `10`, i.e. the number of observations in the transect);
   (b) The **mimum value of the uniform range** (here `0`, i.e. the start of the transect);
   (c) The **maximum value of the uniform range** (here `1000`, i.e. the end of the transect);
3. R has a rich set of methods for **simulation** of **random processes**; in this case we simulated a uniform random process along a line.

On-line help

How do we know how many arguments each method has and their meaning? There is a printed manual, which lists each method with its arguments, function, usage, and examples; this is reproduced as an **on-line help system** within R. You can access this system with the `help` method, which can also be written as `?`.

Task 5 : Display the help for the `runif` method. •

```
> help(runif)
```

This can also be written as:

```
> ?runif
```

The help text for this method opens in another window or a browser window, depending on how your R is configured.

Q3 : *What are the three arguments to the `runif` method? Are they all required? Does the order matter?* *Jump to A3* •

The call to the `runif` method, above, drew the sample and printed it for us to view. But we can't do anything more with it until we **save the results of a method** in a **local variable** in the **workspace**. Then it is available to be used as an argument to other methods.

We save the results of any command using the `<-` (assignment) operator. This is written with two characters: `<` followed by `-`, with no space. This looks like an arrow pointing to the left; you can easily remember that the R command on the **right-hand side** of the assignment operator `<-` is "pointing at" the **left-hand side**, which is the name of the workspace object into which the results will be stored.

```
> sample <- runif(10, 0, 1000)
```

Listing the objects in the workspace

The sample has been stored in the workspace; we can check the contents of the workspace with the `ls` ("list") method with an empty argument:

```
> ls()

[1] "sample"
```

The `ls` method is an example of a method with only **optional** arguments; if no arguments are given, the **default** is to list the names of all the objects in the workspace.

So, we see the `sample` object, but what is its value? When R stores the results of a command in a workspace object it does not also print the results on the console. We can see the value of any workspace object with the `print` method:

```
> print(sample)

 [1] 380.464 838.184 190.798 501.144 871.743  53.883 584.027  13.835
 [9] 985.919 645.234
```

Q4 : *This sample is different from the previous one. Why?* *Jump to A4* •

Now we have the positions, but they are too precise for us to locate in the field; we are satisfied with 0.1 m precision.

Task 6 : Round each position along the transect to one decimal place, store the results in the same variable, and print them. •

We use the `round` method:

```
> sample <- round(sample, 1)
> sample

 [1] 380.5 838.2 190.8 501.1 871.7  53.9 584.0  13.8 985.9 645.2
```

The second line of this example shows that simply typing the object name implicitly calls the `print` method; this is then another way of showing the contents of an object in the workspace.

This command illustrates two other important feature of R:

- Many methods are **vectorized**: they can work on vectors (including matrices) as well as scalars. Here the `round` method is modifying the results of the `runif` method, which is a 10-element vector;
- An assignment can **overwrite** an object in the workspace; in this case the previous value of `sample` (unrounded) is **replaced** by the rounded value.

Now we have the positions to the right precision, but of course we want them in order so we can visit them in order as we walk along the transect:

Task 7 : Sort the positions along the transect and print them. •

We use the `sort` method:

```
> (sample <- sort(sample))

 [1]  13.8  53.9 190.8 380.5 501.1 584.0 645.2 838.2 871.7 985.9
```

Note that the sample is sorted in **ascending** order, i.e. from small to large.

Q5 : *How do you tell R to sort in **descending** order, i.e. from large to small?* *Jump to A5* •

This example shows another way of printing the output of an R command that stores its results in a workspace object: by enclosing the whole command, including the assignment in parenthesis `( ... )`; this forces another evaluation, which prints its results.

Now, here are the previous tasks combined into one and carried out by one R command:

Task 8 : Select a random sample of ten positions along the transect. Round these to the nearest 0.1 m, and present them in order from the beginning to the end of the transect. •

```
> sort(round(runif(10, 0, 1000), 1))

 [1]  25.8 129.8 227.3 295.2 350.9 618.1 735.6 748.6 831.9 969.3
```

Isn't that easier?

This example illustrates another important feature of R:

- Values **returned** by a method can be immediately used as an argument to another method. Here the results of `runif` is the vector to be rounded by the `round` method; and these are then used by the `sort` method. To understand a complex expression, read it from the **inside out**.

Removing objects from the workspace

We are done with the local variable `sample`, so we **remove** it from the workspace with the `rm` ("**rm**ove") method:

Task 9 : Remove the temporary variable `sample` from the workspace. •

We check the 'before' and 'after' contents of the workspace with the `ls` method:

```
> ls()

[1] "sample"

> rm(sample)
> ls()

character(0)
```

The result `character(0)` is R-speak for an empty vector of character strings, i.e. there are no names of objects to list.

3.1 Answers

A1 : *1000 m* *Return to Q1* •

A2 : *Random number generation gives a different result each time.*[2]. *Return to Q2* •

A3 : *There are three possible arguments: the number of samples* `n`*, the minimum value* `min` *and the maximum* `max`*. The last two are not required and* ***default*** *to* 0 *and* 1*, respectively. If arguments are* ***named*** *directly, they can be put in any order. If not, they have to follow the default order.* *Return to Q3* •

A4 : *Same answer as above: random number generation gives a different result each time.* *Return to Q4* •

A5 : *Call* `sort` *with the* ***optional argument*** `decreasing=TRUE`*, e.g.* `(sample <- sort(sample, decreasing=TRUE))`*.* ***Note****: This can be abbreviated to* `d=T`*.*

If you couldn't answer this question . . . did you look at the help text? *Return to Q5* •

4 Loading additional packages

When R starts, it loads a small set of **basic packages** which are necessary to support the S language, basic statistics, and base graphics.

[2] To start a simulation at the same point (e.g. for testing) use the `set.seed` method

Task 10 : Display the list of loaded packages. •

This is done with the `search` method with no arguments:

```
> search()

 [1] ".GlobalEnv"        "package:stats"     "package:graphics"
 [4] "package:grDevices" "package:utils"     "package:datasets"
 [7] "package:lattice"   "package:methods"   "Autoloads"
[10] "package:base"
```

There are a large number of **optional packages** for specific statistical procedures which must be loaded during each session before they can be used. Some of these are quite common, e.g. `lattice` for Trellis graphics [11]. Others are more specialised, e.g. for geostatistics, time-series analysis, neural networks, and non-linar models. We will use the `gstat` geostatistical package [7] extensively in this module.

Task 11 : Display the list of installed packages and find `gstat` in the list. •

This is done with the `library` method with no argument:

```
> library()
```

This method will open a window with a list of all available packages; scroll down to find `gstat`.

> **Note:** If `gstat` is not in the list, you must first **install** it from the on-line R package archive known as CRAN:
>
> 1. Make sure your computer is connected to the Internet;
> 2. Select menu item `Packages | Install Package(s) from CRAN ...`;
> 3. You will be asked to select a **mirror**; this is a computer near you that stores all the materials from CRAN so we don't depend on on server;
> 4. From the long list, select the package(s) you want, here `gstat` and `sp`, and confirm.
> 5. R will install them, and they are now ready to load and use.

> **Note:** There are many, many more packages available at CRAN; the `Packages | Install package(s)...` menu item in the Windows GUI will show the complete list.

Task 12 : Load the `gstat` package. •

To load an installed package, it is possible to use the `library` method, naming the package you want to install; however it is better to use the `require` method, which first checks whether the library is already loaded, and only if not, loads it.

```
> require(gstat)
```

```
Loading required package: sp
```

Notice that package `sp` (which provides classes and methods for spatial data) was loaded automatically; this is because `gstat` uses some methods from `sp`.

Task 13 : Confirm that `gstat` was loaded. •

The `search` method shows all loaded libraries[3]:

```
> search()

 [1] ".GlobalEnv"        "package:gstat"     "package:sp"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:lattice"
[10] "package:methods"   "Autoloads"         "package:base"
```

You should now see `gstat`, as well as `sp`, in the list of packages.

5 Loading and examining an example dataset

R comes with many example datasets (part of the default `datasets` package) and most add-in packages also include example datasets. These are used to test statistical methods, and are a good example of the 'open source' collaborative attitude to collective scientific knowledge exemplified by the R project.

> **Note:** Of course, there are many ways to get your own data into R; we will see some of them in later lessons.

Task 14 : Display the list of installed datasets. •

This is done with the `data` method with no argument:

```
> data()
```

This method will open a window with a list of all available datasets, organized by package. The dataset we will use for this exercises is provided with the `sp` package, which was just loaded along with `gstat`.

Task 15 : Scroll down to the list of datasets for the `sp` package. •

Q6 : *Three of the listed datasets refer to the Meuse river. What are the names of the datasets?* *Jump to A6* •

Datasets are also documented in the help system.

Task 16 : Display the help for the `meuse` dataset. •

[3] and attached data frames, see §6.1

```
> help(meuse)
```

Q7 : *According to the help text, where can you find more information about this dataset?* *Jump to A7* •

Task 17 : Load the Meuse data set. •

To load a dataset, use the `data` method, naming the dataset you want to load:

```
> data(meuse)
```

Task 18 : Verify that the objects from the `meuse` dataset have been loaded. •

Remember (§3), to list the objects in the workspace, we can use the `ls` method:

```
> ls()

[1] "meuse"
```

Task 19 : Examine the structure of the `meuse` object. •

To examine the structure, use the `str` method:

```
> str(meuse)

'data.frame':        155 obs. of  14 variables:
 $ x       : num  181072 181025 181165 181298 181307 ...
 $ y       : num  333611 333558 333537 333484 333330 ...
 $ cadmium: num  11.7 8.6 6.5 2.6 2.8 3 3.2 2.8 2.4 1.6 ...
 $ copper : num  85 81 68 81 48 61 31 29 37 24 ...
 $ lead    : num  299 277 199 116 117 137 132 150 133 80 ...
 $ zinc    : num  1022 1141 640 257 269 ...
 $ elev    : num  7.91 6.98 7.8 7.66 7.48 ...
 $ dist    : num  0.00136 0.01222 0.10303 0.19009 0.27709 ...
 $ om      : num  13.6 14 13 8 8.7 7.8 9.2 9.5 10.6 6.3 ...
 $ ffreq   : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
 $ soil    : Factor w/ 3 levels "1","2","3": 1 1 1 2 2 2 2 1 1 2 ...
 $ lime    : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 1 1 ...
 $ landuse: Factor w/ 15 levels "Aa","Ab","Ag",..: 4 4 4 11 4 11 4 2 2 15 ...
 $ dist.m : num  50 30 150 270 380 470 240 120 240 420 ...
```

The first line of output tells us that `meuse` is a **data frame**, which the class of object used to hold most data sets.

The following lines show the **fields** (attributes, variables); here we see:

- two **coördinates** (here named `x` and `y`);
- four **categorical** variables (fields `ffreq`, `soil`, `lime` and `landuse`);

- four **continuous** variables, representing element concentrations from soil samples measured in mg kg^{-1}: fields `cadmium`, `copper`, `zinc` and `lead`);
- another continuous variable for the organic matter concentration in weight percent (`om`);
- three more continuous variable for (1) the elevation in meters above a local base level (`elev`) and (2) the distance from the river, both absolute (`dist.m`) and normalized (`dist`).

Q8 : *How can you find the above information on-line?* *Jump to A8* •

Each field then has a a set of values, one per observation.

Q9 : *How many variables are there in this dataset? How many observations?* *Jump to A9* •

5.1 Answers

A6 : *(1) `meuse`, the Meuse river data set; (2) `meuse.grid`, a prediction grid for this area; (3) `meuse.riv`, an outline of the River Meuse in the area.* *Return to Q6* •

A7 : *The on-line help text cites the textbook by Burrough and McDonnell [1]; however (if you have this book you can check) it doesn't give too much more information. The dataset is fully described in the field report by Rikken and Van Rijn [9].* *Return to Q7* •

A8 : *From the information in the help text (`?meuse`) and the reference mentioned in the help text.* *Return to Q8* •

A9 : *14 variables (fields), each with 155 observations (cases).* *Return to Q9* •

6 Descriptive statistics

In this section we see how to do some simple data description with R.

6.1 Summarizing one variable

Let's first look at one attribute, the lead (Pb) content. This metal is a serious human health hazard. It can be inhaled as dust from disturbed soil or taken up by plants and ingested by humans and animals as they eat the plants. The critical value for Pb in agricultural soils, according to the Berlin Digital Environmental Atlas[4], is 600 mg kg^{-1} for agricultural fields: above this level

[4] http://www.stadtentwicklung.berlin.de/umwelt/umweltatlas/ed103103.htm

grain crops can not be grown for human consumption. Above 300 mg kg^{-1} crops must be tested; above 100 mg kg^{-1} consumption of leafy vegetables should be avoided. Levels below 200 mg kg^{-1} are required for sports fields or parks where soil may become bare from over-use. Natural levels in most unpolluted soils are on the order of 30 mg kg^{-1}.

Task 20 : Summarize the lead content of the 155 soil samples. •

For almost all R objects, the `summary` method will provide a simple summary:

```
> summary(meuse$lead)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   37.0    72.5   123.0   153.0   207.0   654.0
```

This example shows that fields in the data frame are commonly referred to by their name, using the syntax `frame$field`. You can think of the `$` sign as an extraction method: extract the named field, here `lead`, from the named frame.

Q10 : *In the expression* `meuse$lead`, *what is the name of the* ***data frame*** *and what is the name of the* ***field*** *(variable)?* *Jump to A10* •

Q11 : *What are the minimum and maximum lead concentrations in this sample?* *Jump to A11* •

There are many R methods to summarize a list of numbers; these have obvious names: `min`, `max`, `median`, `mean`, `range`, `var` (sample variance), `sd` (sample standard deviation), and `quantile`:

```
> min(meuse$lead)

[1] 37

> max(meuse$lead)

[1] 654

> median(meuse$lead)

[1] 123

> mean(meuse$lead)

[1] 153.36

> range(meuse$lead)

[1]  37 654

> var(meuse$lead)

[1] 12392
```

```
> sd(meuse$lead)

[1] 111.32

> quantile(meuse$lead)

   0%   25%   50%   75%  100%
 37.0  72.5 123.0 207.0 654.0

> quantile(meuse$lead, seq(0, 1, 0.1))

   0%   10%   20%   30%   40%   50%   60%   70%   80%   90%  100%
 37.0  50.0  65.8  77.2  87.0 123.0 148.0 182.6 226.4 290.4 654.0
```

The `quantile` method is typical of R methods: it has a default second argument for the quantiles to be computed, but this can be over-ridden if we want a different list; here we used the `seq` method to specify the vector `0, 0.1 ... 0.9, 1` to get the deciles.

Attaching a data frame to the search path

The only name visible to R at this point is `meuse`, i.e. the data frame. We can make the field names within the frame visible by **attaching** the data frame to the search path with the `attach` method.

Task 21 : Attach the `meuse` data frame to the search path. •

After using `attach`, we use `search` to check the search path:

```
> attach(meuse)
> search()

 [1] ".GlobalEnv"        "meuse"             "package:gstat"
 [4] "package:sp"        "package:stats"     "package:graphics"
 [7] "package:grDevices" "package:utils"     "package:datasets"
[10] "package:lattice"   "package:methods"   "Autoloads"
[13] "package:base"
```

The `meuse` data frame now appears in the search path; this means that R will look inside it for names it can't find in the workspace. So, we can name the field directly, rather than use the `data.frame$field` syntax:

```
> summary(lead)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   37.0    72.5   123.0   153.0   207.0   654.0

> summary(meuse$lead)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   37.0    72.5   123.0   153.0   207.0   654.0
```

6.2 Finding a specific record in a dataframe

Let's do some useful analysis:

Task 22 : Display the record for the most polluted observation. •

First we determine which of the 155 observations has the maximum value, with the `which.max` method[5]:

```
> which.max(lead)

[1] 54
```

Q12 : *Of the 155 rows of the data frame, representing the observations , which row contains the record for the most polluted observation?* *Jump to A12* •

Viewing a data frame as a matrix

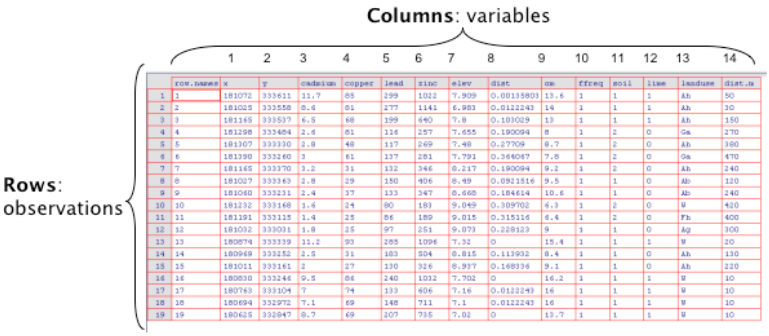
To display this record, we introduce another way of looking at R data frames: as a **two-dimensional matrix**, where the **rows** are the **cases** (observations) and the **columns** are the fields (variables). The matrix is referred to in standard mathematical notation with square brackets: `[ row, column ]`; this is the `[]` operator.

First, we look at the whole data frame in matrix form using the `fix` method, which is also used to edit values in a data frame.

Task 23 : Display the `meuse` data frame in an editing window. •

```
> fix(meuse)
```

The following figure shows the window opened by `fix`, and how the frame is a matrix of rows (observations) and columns (variables).



We don't want to edit this frame, just to view it.

Task 24 : Close the editing window. •

Under MS-Windows, select the GUI menu item `File | Close` or click the close button at the upper-right of the editing window.

6.3 Selecting part of a dataframe

Now we see how to use matrix notation to select parts of the data frame. We will see two notations: (1) specifying rows and columns directly, and (2)

[5] there is of course a `which.min` method

by logical conditions for rows.

First, one row:

```
> meuse[which.max(lead), ]
```

```
        x      y cadmium copper lead zinc elev      dist   om ffreq
55 179973 332255      12    117  654 1839  7.9 0.0054321 16.5     1
   soil lime landuse dist.m
55    1    1       W     10
```

In this expression `which.max(lead)` returns the value `54`, as we saw before; so this expression is equivalent to:

```
> meuse[54, ]
```

This shows us all fields (columns) of the 54th record, because the second (column) subscript (after the comma) is left blank.

The following figure highlights this one matrix row in the data editor:[6]

	row.names	x	y	cadmium	copper	lead	zinc	elev	dist	om	ffreq	soil	lime	landuse	dist.m
45	46	180355	332299	4.2	51	281	746	7.94	0.081222	5.1	1	2	0	Ah	100
46	47	180292	332157	4.3	50	294	746	6.36	0.190086	5.3	1	2	0	Am	200
47	48	180283	332014	3.1	38	211	464	7.78	0.287941	4.5	1	2	0	Ah	320
48	49	180282	331861	1.7	26	135	365	8.18	0.423826	4.9	1	2	0	Ah	480
49	50	180270	331707	1.7	24	112	282	9.42	0.554289	4.5	1	2	0	Bw	660
50	51	180199	331591	2.1	32	162	375	8.867	0.603225	5.5	1	2	0	Bw	690
51	52	180135	331552	1.7	24	94	222	8.292	0.614071	3.4	1	2	0	Ab	710
52	53	180237	332351	8.2	47	191	812	8.06	0.00135803	11.1	1	1	1	Ah	10
53	54	180103	332297	17	128	405	1548	7.98	0	12.3	1	1	1	W	10
54	55	179973	332255	12	117	654	1839	7.9	0.0054321	16.5	1	1	1	W	10
55	56	179826	332217	9.4	104	482	1528	7.74	0.0054321	13.9	1	1	1	W	10
56	57	179687	332161	8.2	76	276	933	7.552	0.0054321	8.1	1	1	1	W	20
57	58	179792	332035	2.6	36	180	432	7.76	0.146578	3.1	1	1	0	Fw	200
58	59	179902	332113	3.5	34	207	550	6.74	0.135684	5.8	1	1	0	Fw	140
59	60	180100	332213	10.9	90	541	1571	6.68	0.0703333	10.2	1	1	1	Fw	70
60	61	179604	332059	7.3	80	310	1190	7.4	0.0484831	12	1	1	1	W	20
61	62	179526	331936	9.4	78	210	907	7.44	0.0054321	14.1	1	1	1	W	10
62	63	179495	331770	8.3	77	158	761	7.36	0.0054321	14.5	1	1	1	Fw	10
63	64	179489	331633	7	65	141	659	7.2	0.0316663	14.8	1	1	1	W	20
64	65	179414	331494	6.8	66	144	643	7.22	0.0122243	13.3	1	1	1	Ah	10
65	66	179334	331366	7.4	72	181	801	7.36	0.0122243	15.2	1	1	1	W	20

Q13 : *How close is this observation to the river?* *Jump to A13* •

Note: You are probably confused about the number `55` that is printed at the left of the output, before the field values. This is the **row name** of the 54th record; this is a character string, not a number. We can get a clue to why the 54th record has row name `55` by listing all the row names of the `meuse` data frame with the `row.names` method:

```
> row.names(meuse)
```

```
  [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"
 [11] "11"  "12"  "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"
 [21] "21"  "22"  "23"  "24"  "25"  "26"  "27"  "28"  "29"  "30"
 [31] "31"  "32"  "33"  "34"  "35"  "37"  "38"  "39"  "40"  "41"
 [41] "42"  "43"  "44"  "45"  "46"  "47"  "48"  "49"  "50"  "51"
 [51] "52"  "53"  "54"  "55"  "56"  "57"  "58"  "59"  "60"  "61"
 [61] "62"  "63"  "64"  "65"  "66"  "67"  "69"  "75"  "76"  "79"
 [71] "80"  "81"  "82"  "83"  "84"  "85"  "86"  "87"  "88"  "89"
 [81] "90"  "123" "160" "163" "70"  "71"  "91"  "92"  "93"  "94"
 [91] "95"  "96"  "97"  "98"  "99"  "100" "101" "102" "103" "104"
[101] "105" "106" "108" "109" "110" "111" "112" "113" "114" "115"
```

[6] The highlighting was done "by hand" for the figure and is not a feature of the data editor.

```
[111] "116" "117" "118" "119" "120" "121" "122" "124" "125" "126"
[121] "127" "128" "129" "130" "131" "132" "133" "134" "135" "136"
[131] "161" "162" "137" "138" "140" "141" "142" "143" "144" "145"
[141] "146" "147" "148" "149" "150" "151" "152" "153" "154" "155"
[151] "156" "157" "158" "159" "164"
```

Here we can see that the names range from `"1"` to `"164"`; these are most likely the original sample numbers from the soil pollution study. Nine of the original samples are not in this data frame, e.g. original sample `"36"` is missing. This is explained in the help text for the `meuse.all` data set: "Eight samples were left out because they were not indicative for the metal content of the soil. They were taken in an old pit. One sample contains an outlying zinc value, which was also discarded for the meuse (155) data set."

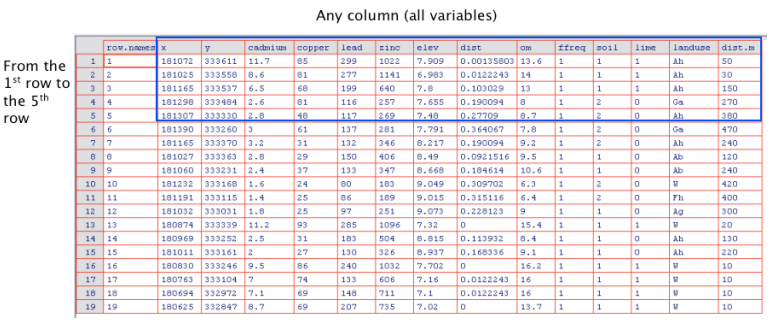
Let's see some other ways to specify individual or groups of rows and columns; the output should explain each format:

All columns of the first five rows:

```
> meuse[1:5, ]
```

```
       x      y cadmium copper lead zinc  elev      dist   om ffreq
1 181072 333611    11.7     85  299 1022 7.909 0.0013580 13.6     1
2 181025 333558     8.6     81  277 1141 6.983 0.0122243 14.0     1
3 181165 333537     6.5     68  199  640 7.800 0.1030290 13.0     1
4 181298 333484     2.6     81  116  257 7.655 0.1900940  8.0     1
5 181307 333330     2.8     48  117  269 7.480 0.2770900  8.7     1
  soil lime landuse dist.m
1    1    1      Ah     50
2    1    1      Ah     30
3    1    1      Ah    150
4    2    0      Ga    270
5    2    0      Ah    380
```

In the data editor this looks like:



	row.names	x	y	cadmium	copper	lead	zinc	elev	dist	om	ffreq	soil	lime	landuse	dist.m
1	1	181072	333611	11.7	85	299	1022	7.909	0.00135803	13.6	1	1	1	Ah	50
2	2	181025	333558	8.6	81	277	1141	6.983	0.0122243	14	1	1	1	Ah	30
3	3	181165	333537	6.5	68	199	640	7.8	0.103029	13	1	1	1	Ah	150
4	4	181298	333484	2.6	81	116	257	7.655	0.190094	8	1	2	0	Ga	270
5	5	181307	333330	2.8	48	117	269	7.48	0.27709	8.7	1	2	0	Ah	380
6	6	181390	333260	3	61	137	281	7.791	0.364067	7.8	1	2	0	Ga	470
7	7	181165	333370	3.2	31	132	346	8.217	0.190094	9.2	1	2	0	Ah	240
8	8	181027	333363	2.8	29	150	406	8.49	0.0921516	9.5	1	1	0	Ab	120
9	9	181060	333231	2.4	37	133	347	8.668	0.184614	10.6	1	1	0	Ab	240
10	10	181232	333168	1.6	24	80	183	9.049	0.309702	6.3	1	2	0	W	420
11	11	181191	333115	1.4	25	86	189	9.015	0.315116	6.4	1	2	0	Fh	400
12	12	181032	333031	1.8	25	97	251	9.073	0.228123	9	1	1	0	Ag	300
13	13	180874	333339	11.2	93	285	1096	7.32	0	15.4	1	1	1	W	20
14	14	180969	333252	2.5	31	183	504	8.815	0.113932	8.4	1	1	0	Ah	130
15	15	181011	333161	2	27	130	326	8.937	0.168336	9.1	1	1	0	Ah	220
16	16	180830	333246	9.5	86	240	1032	7.702	0	16.2	1	1	1	W	10
17	17	180763	333104	7	74	133	606	7.16	0.0122243	16	1	1	1	W	10
18	18	180694	332972	7.1	69	148	711	7.1	0.0122243	16	1	1	1	W	10
19	19	180625	332847	8.7	69	207	735	7.02	0	13.7	1	1	1	W	10

The first two columns of the first five rows:

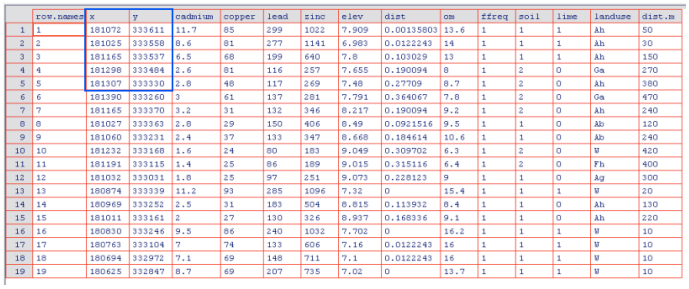
```
> meuse[1:5, 1:2]
```

```
       x      y
1 181072 333611
2 181025 333558
3 181165 333537
4 181298 333484
5 181307 333330
```

In the data editor this looks like:

From the 1st column to the 2nd column

From the 1st row to the 5th row



	row.names	x	y	cadmium	copper	lead	zinc	elev	dist	om	ffreq	soil	lime	landuse	dist.m
1	1	181072	333611	11.7	85	299	1022	7.909	0.00135803	13.6	1	1	1	Ah	50
2	2	181025	333558	8.6	81	277	1141	6.983	0.0122243	14	1	1	1	Ah	30
3	3	181165	333537	6.5	68	199	640	7.8	0.103029	13	1	1	1	Ah	150
4	4	181298	333484	2.6	81	116	257	7.655	0.190094	8	1	2	0	Ga	270
5	5	181307	333330	2.8	48	117	269	7.48	0.27709	8.7	1	2	0	Ah	380
6	6	181390	333260	3	61	137	281	7.791	0.364067	7.8	1	2	0	Ga	470
7	7	181165	333370	3.2	31	132	346	8.217	0.190094	9.2	1	2	0	Ah	240
8	8	181027	333363	2.8	29	150	406	8.49	0.0921516	9.5	1	1	0	Ah	120
9	9	181060	333231	2.4	37	133	347	8.668	0.184614	10.6	1	1	0	Ah	240
10	10	181232	333168	1.6	24	80	183	9.049	0.309702	6.3	1	2	0	W	420
11	11	181191	333115	1.4	25	86	189	9.015	0.315116	6.4	1	2	0	Fh	400
12	12	181032	333031	1.8	25	97	251	9.073	0.228123	9	1	1	0	Ag	300
13	13	180874	333339	11.2	93	285	1096	7.32	0	15.4	1	1	1	W	20
14	14	180969	333252	2.5	31	183	504	8.815	0.113932	8.4	1	1	0	Ah	130
15	15	181011	333161	2	27	130	326	8.937	0.168336	9.1	1	1	0	Ah	220
16	16	180830	333246	9.5	86	240	1032	7.702	0	16.2	1	1	1	W	10
17	17	180763	333104	7	74	133	606	7.16	0.0122243	16	1	1	1	W	10
18	18	180694	332972	7.1	69	148	711	7.1	0.0122243	16	1	1	1	W	10
19	19	180625	332847	8.7	69	207	735	7.02	0	13.7	1	1	1	W	10

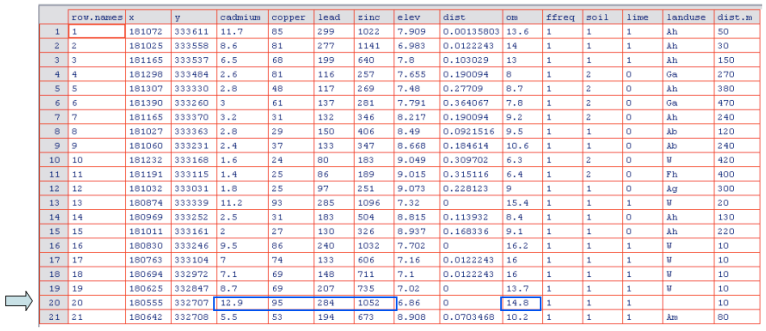
Columns 3 through 6 and 9 of the 20th observation:

```
> meuse[20, c(3:6, 9)]
```

```
   cadmium copper lead zinc   om
20    12.9     95  284 1052 14.8
```

In the data editor this looks like:

3 4 5 6 9



	row.names	x	y	cadmium	copper	lead	zinc	elev	dist	om	ffreq	soil	lime	landuse	dist.m
1	1	181072	333611	11.7	85	299	1022	7.909	0.00135803	13.6	1	1	1	Ah	50
2	2	181025	333558	8.6	81	277	1141	6.983	0.0122243	14	1	1	1	Ah	30
3	3	181165	333537	6.5	68	199	640	7.8	0.103029	13	1	1	1	Ah	150
4	4	181298	333484	2.6	81	116	257	7.655	0.190094	8	1	2	0	Ga	270
5	5	181307	333330	2.8	48	117	269	7.48	0.27709	8.7	1	2	0	Ah	380
6	6	181390	333260	3	61	137	281	7.791	0.364067	7.8	1	2	0	Ga	470
7	7	181165	333370	3.2	31	132	346	8.217	0.190094	9.2	1	2	0	Ah	240
8	8	181027	333363	2.8	29	150	406	8.49	0.0921516	9.5	1	1	0	Ah	120
9	9	181060	333231	2.4	37	133	347	8.668	0.184614	10.6	1	1	0	Ah	240
10	10	181232	333168	1.6	24	80	183	9.049	0.309702	6.3	1	2	0	W	420
11	11	181191	333115	1.4	25	86	189	9.015	0.315116	6.4	1	2	0	Fh	400
12	12	181032	333031	1.8	25	97	251	9.073	0.228123	9	1	1	0	Ag	300
13	13	180874	333339	11.2	93	285	1096	7.32	0	15.4	1	1	1	W	20
14	14	180969	333252	2.5	31	183	504	8.815	0.113932	8.4	1	1	0	Ah	130
15	15	181011	333161	2	27	130	326	8.937	0.168336	9.1	1	1	0	Ah	220
16	16	180830	333246	9.5	86	240	1032	7.702	0	16.2	1	1	1	W	10
17	17	180763	333104	7	74	133	606	7.16	0.0122243	16	1	1	1	W	10
18	18	180694	332972	7.1	69	148	711	7.1	0.0122243	16	1	1	1	W	10
19	19	180625	332847	8.7	69	207	735	7.02	0	13.7	1	1	1	W	10
20	20	180555	332707	12.9	95	284	1052	6.86	0	14.8	1	1	1		10
21	21	180642	332708	5.5	53	194	673	8.908	0.0703468	10.2	1	1	1	Am	80

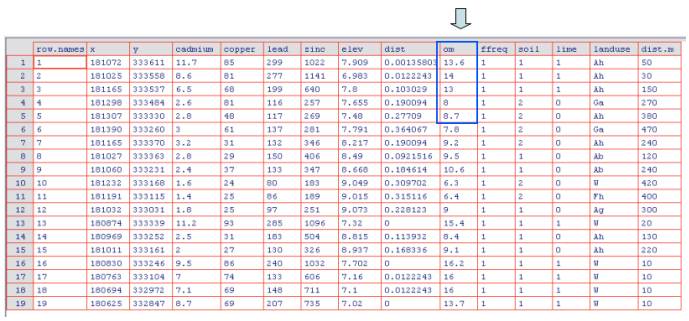
The organic matter content for the first five observations:

```
> meuse[1:5, "om"]
```

```
[1] 13.6 14.0 13.0  8.0  8.7
```

In the data editor this looks like:

From the 1st row to the 5th row



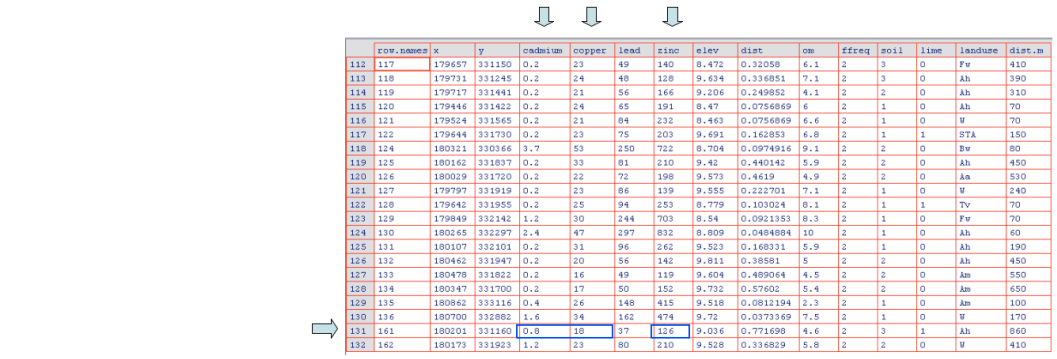
	row.names	x	y	cadmium	copper	lead	zinc	elev	dist	om	ffreq	soil	lime	landuse	dist.m
1	1	181072	333611	11.7	85	299	1022	7.909	0.00135803	13.6	1	1	1	Ah	50
2	2	181025	333558	8.6	81	277	1141	6.983	0.0122243	14	1	1	1	Ah	30
3	3	181165	333537	6.5	68	199	640	7.8	0.103029	13	1	1	1	Ah	150
4	4	181298	333484	2.6	81	116	257	7.655	0.190094	8	1	2	0	Ga	270
5	5	181307	333330	2.8	48	117	269	7.48	0.27709	8.7	1	2	0	Ah	380
6	6	181390	333260	3	61	137	281	7.791	0.364067	7.8	1	2	0	Ga	470
7	7	181165	333370	3.2	31	132	346	8.217	0.190094	9.2	1	2	0	Ah	240
8	8	181027	333363	2.8	29	150	406	8.49	0.0921516	9.5	1	1	0	Ah	120
9	9	181060	333231	2.4	37	133	347	8.668	0.184614	10.6	1	1	0	Ah	240
10	10	181232	333168	1.6	24	80	183	9.049	0.309702	6.3	1	2	0	W	420
11	11	181191	333115	1.4	25	86	189	9.015	0.315116	6.4	1	2	0	Fh	400
12	12	181032	333031	1.8	25	97	251	9.073	0.228123	9	1	1	0	Ag	300
13	13	180874	333339	11.2	93	285	1096	7.32	0	15.4	1	1	1	W	20
14	14	180969	333252	2.5	31	183	504	8.815	0.113932	8.4	1	1	0	Ah	130
15	15	181011	333161	2	27	130	326	8.937	0.168336	9.1	1	1	0	Ah	220
16	16	180830	333246	9.5	86	240	1032	7.702	0	16.2	1	1	1	W	10
17	17	180763	333104	7	74	133	606	7.16	0.0122243	16	1	1	1	W	10
18	18	180694	332972	7.1	69	148	711	7.1	0.0122243	16	1	1	1	W	10
19	19	180625	332847	8.7	69	207	735	7.02	0	13.7	1	1	1	W	10

The last example shows the contents of the other three metals, for the site that is least polluted with lead; this uses the `which.min` method:

```
> meuse[which.min(lead), c("cadmium", "copper", "zinc")]

    cadmium copper zinc
161     0.8     18  126
```

In the data editor this looks like:



	row.names	x	y	cadmium	copper	lead	zinc	elev	dist	om	ffreq	soil	lime	landuse	dist.m
112	117	179657	331150	0.2	23	49	140	8.472	0.32058	6.1	2	3	0	Fw	410
113	118	179731	331245	0.2	24	48	128	9.634	0.336851	7.1	2	3	0	Ah	390
114	119	179717	331441	0.2	21	56	166	9.206	0.249852	4.1	2	2	0	Ah	310
115	120	179446	331422	0.2	24	65	191	8.47	0.0756869	6	2	1	0	Ah	70
116	121	179524	331565	0.2	21	84	232	8.463	0.0756869	6.6	2	1	0	W	70
117	122	179644	331730	0.2	23	75	203	9.691	0.162853	6.8	2	1	1	STA	150
118	124	180321	330366	3.7	53	250	722	8.704	0.0974916	9.1	2	2	0	Bw	80
119	125	180162	331837	0.2	33	81	210	9.42	0.440142	5.9	2	2	0	Ah	450
120	126	180029	331720	0.2	22	72	198	9.573	0.4619	4.9	2	2	0	Aa	530
121	127	179797	331919	0.2	23	86	139	9.555	0.222701	7.1	2	1	0	W	240
122	128	179642	331955	0.2	25	94	253	8.779	0.103024	8.1	2	1	1	Tv	70
123	129	179849	332142	1.2	30	244	703	8.54	0.0921353	8.3	2	1	0	Fw	70
124	130	180265	332297	2.4	47	297	832	8.809	0.0484884	10	2	1	0	Ah	60
125	131	180107	332101	0.2	31	96	262	9.523	0.168331	5.9	2	1	0	Ah	190
126	132	180462	331947	0.2	20	56	142	9.811	0.38581	5	2	2	0	Ah	450
127	133	180478	331822	0.2	16	49	119	9.604	0.489064	4.5	2	2	0	Am	550
128	134	180347	331700	0.2	17	50	152	9.732	0.57602	5.4	2	2	0	Am	650
129	135	180862	333116	0.4	26	148	415	9.518	0.0812194	2.3	2	1	0	Am	100
130	136	180700	332882	1.6	34	162	474	9.72	0.0373369	7.5	2	1	0	W	170
131	161	180201	331160	0.8	18	37	126	9.036	0.771698	4.6	2	3	1	Ah	860
132	162	180173	331923	1.2	23	80	210	9.528	0.336829	5.8	2	2	0	W	410

6.4 Logical expressions

Now we move to another analysis, namely to determine the level of pollution in the whole sample. We use this analysis to illustrate the use of **logical expressions** in R. We begin with an example.

Task 25 : Determine the proportion of samples that are above the critical values of 600, 300, 200 and 100 mg kg^{-1}. •

First we do this one at a time, using the `sum` method to count the number of cases when a **logical expression** is `TRUE`.

Start with the highest threshold, 600 mg kg^{-1}:

```
> round(sum(lead > 600)/length(lead) * 100, 1)

[1] 0.6
```

The logical expression here is `(lead > 600)`. This is either `TRUE` or `FALSE` for each. When we `sum` these, the `TRUE` are counted as `1` and the `FALSE` as `0`. We compare this to the `length` of the vector to get the proportion:

> **Note:** R defines the usual logical operators for arithmetical comparison, i.e. `>`, `<`, `>=` (greater than or equal), `<=`, `==`, and `!=` (not equal to). Note that "equal to" must be written with two equals signs: `==`.

```
> lead > 600

  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [11] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [21] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [31] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [41] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [51] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
 [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [71] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
 [81] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [91] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[101] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[121] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[131] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[141] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[151] FALSE FALSE FALSE FALSE FALSE

> sum(lead > 600)

[1] 1

> length(lead)

[1] 155

> sum(lead > 600)/length(lead)

[1] 0.0064516
```

Then we multiplied by 100 to get a percentage and rounded to one decimal place.

Q14 : *How many of the 155 observations are above the 600 mg kg^{-1} threshold? What is this as a percentage?* Jump to A14 •

Now for the other thresholds:

```
> round(sum(lead > 300)/length(lead) * 100, 1)

[1] 8.4

> round(sum(lead > 200)/length(lead) * 100, 1)

[1] 27.1

> round(sum(lead > 100)/length(lead) * 100, 1)

[1] 54.8

> round(sum(lead > 30)/length(lead) * 100, 1)

[1] 100
```

Q15 : *What proportion of the observations are above the natural (background) level of 30 mg kg^{-1}? Above the safe limit for sports fields, 200 mg kg^{-1}?* Jump to A15 •

Using logical expressions to select records in a data frame Logical expressions provide another way to select rows (records, fields) from data frames, i.e. rows that meet some logical condition; this continues the discussion of §6.3.

Task 26 : Select the observations from the rarely-flooded soils. •

The on-line documentation in this case doesn't help so much; but reference to the original field report [9] reveals that in field `ffreq`, the rarely-flooded soils are indicated by code `3`. We use this in a **logical condition** in the row position of the matrix notation:

We build this up piece-by-piece so that you can see what is going on "behind the scenes"; this should give a better understanding when you attempt to write your own selection expressions.

First, we compute a vector with the same length as the number of rows (records) in the data frame, showing whether the row meets the condition:

```
> summary(ffreq)

 1  2  3
84 48 23

> ffreq == "3"

  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [11] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [21] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [31] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [41] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [51] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [71] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [81] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [91] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[101] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[121] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[131] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[141]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[151]  TRUE  TRUE  TRUE  TRUE  TRUE

> summary(ffreq == "3")

   Mode   FALSE    TRUE    NA's
logical     132      23       0
```

Q16 : *How many records meet the condition (rarely flooded)? How many do not? What is the meaning of the third category in the summary?* *Jump to A16* •

The logical vector of `T` and `F` values can then be used in the row position of the selection; only the rows where the logical vector is `T` will be selected:

```
> meuse.flood3 <- meuse[ffreq == "3", ]
> str(meuse.flood3)

'data.frame':        23 obs. of  14 variables:
 $ x      : num  180923 180467 179917 179822 179991 ...
 $ y      : num  332874 331694 331325 331242 331069 ...
```

```
$ cadmium: num  0.2 0.2 0.8 1 0.8 1.2 2 1.5 1.1 0.8 ...
$ copper : num  20 14 46 29 19 31 27 29 22 20 ...
$ lead   : num  80 49 42 48 41 73 146 95 72 51 ...
$ zinc   : num  220 133 141 158 129 206 451 296 189 154 ...
$ elev   : num  9.15 10.08 9.97 10.14 10.32 ...
$ dist   : num  0.228 0.598 0.446 0.397 0.581 ...
$ om     : num  4.4 4.4 4.5 5.2 4.6 6.9 7 5.4 6.2 5 ...
$ ffreq  : Factor w/ 3 levels "1","2","3": 3 3 3 3 3 3 3 3 3 3 ...
$ soil   : Factor w/ 3 levels "1","2","3": 1 2 2 2 3 1 1 1 1 1 ...
$ lime   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
$ landuse: Factor w/ 15 levels "Aa","Ab","Ag",..: 1 5 5 5 15 15 15 4 4 10 ...
$ dist.m : num  290 680 540 480 720 380 310 430 370 290 ...
```

Q17 : *How many records are in this subset? How many fields?* *Jump to A17* •

We can ask the same questions about the subset (i.e. only rarely-flooded soils) as we did about the whole set. We repeat the task and questions from §6.1:

Task 27 : Summarize the lead content of the soil samples from the rarely-flooded soils; compare these to the summary statistics for the whole sample. •

```
> summary(meuse.flood3$lead)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
     41      51      80     103     141     258

> summary(meuse$lead)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   37.0    72.5   123.0   153.0   207.0   654.0
```

Q18 : *How do the lead concentrations of the rarely-flooded soils compare the the lead concentrations of the full sample?* *Jump to A18* •

We are finished with the dataframe containing the subsample, so we remove it from the workspace with the `rm` method:

```
> rm(meuse.flood3)
```

6.5 Answers

A10 : *The data frame is* `meuse`*, the field is* `lead`. *Return to Q10* •

A11 : *Minimum* `37.0`*, maximum* `654.0`. *Return to Q11* •

A12 : *The 54th row; this is the vector* ***index*** *given as output by* `which.max`.

Return to Q12 •

A13 : *10 m (see field* `dist.m`*).* *Return to Q13* •

A14 : *1 observation; this is* `0.6%`*.* *Return to Q14* •

A15 : *All of them;* `27.1%`*.* *Return to Q15* •

A16 : *23 yes, 132 no. The third category, marked* `NA's`*, is the number of "not available", i.e. observations with missing values. In this case there are none.* *Return to Q16* •

A17 : *Twenty-three (23) records (observations); all 14 fields were selected, same as the original dataframe.* *Return to Q17* •

A18 : *Quite a bit lower.* *Return to Q18* •

7 Exploratory graphics

R provides a rich environment for statistical visualisation [6]. There are two graphics systems: the `base` system (in the `graphics` package, loaded by default when R starts) and the Trellis system (implemented in R by the `lattice` package); we will use the base graphics in this exercise.

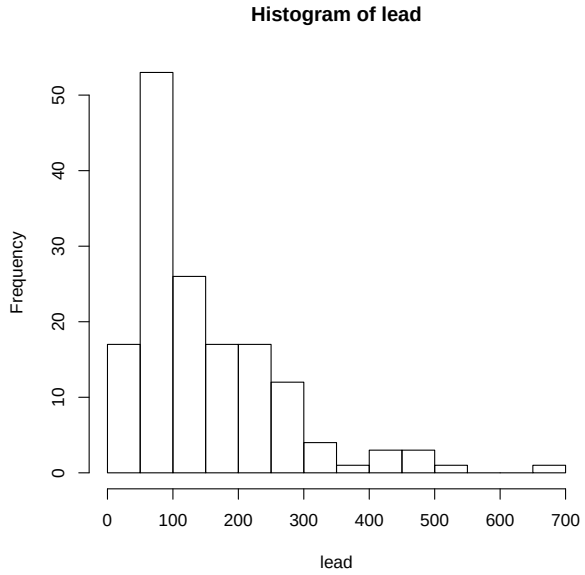
Before beginning a data analysis, it is helpful to **visualize** the dataset. R is an excellent environment for visualisation; it can produce simple plots but also plots of unlimited sophistication, information and beauty. We look first at single variables and then at the relation between two variables.

7.1 Univariate exploratory graphics

Task 28 : Visualise the **frequency distribution** of lead content in the soil samples with a frequency histogram. •

A histogram, displayed by the `hist` method, shows the distribution:

```
> hist(lead)
```



Q19 : *What are the two axes of the default histogram?* *Jump to A19* •

Q20 : *Describe the distribution: is it symmetric or skewed? What is the modal (most common) range of values? Does there appear to be one population or several?* *Jump to A20* •

Saving graphic output

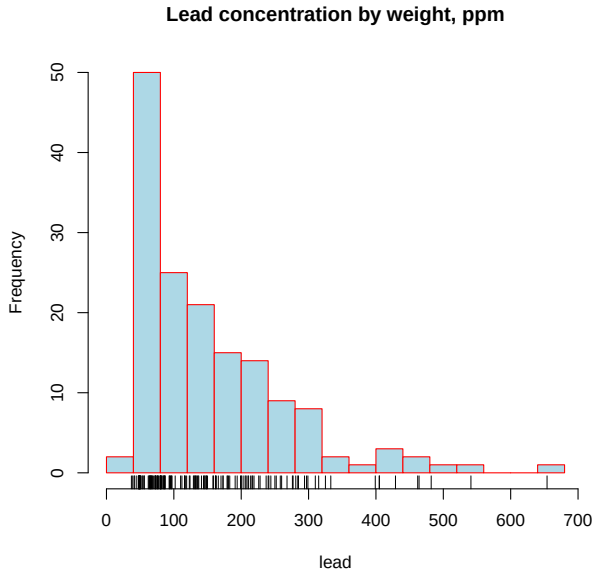
You can **save the graphics** window to any common graphics format. In the Windows GUI, bring the graphics window to the front (e.g. click on its title bar), select menu command `File | Save as ...` and then one of the formats.

All R graphics, including histograms, can be enhanced. Here we change the break points with the `breaks` argument, the colour of the bars with the `col` argument, the colour of the border with the `border` argument, and supply a title with the `main` argument; we then add a **rug** plot (with, what else?, the `rug` method) along the x-axis to show the actual observations.

> **Note:** R graphics are very customizable; for a good introduction to the possibilities, including how to specify colours, see §5 of Rossiter [10]. For now you should just follow along with the examples in this tutorial.

The `rug` method is an example of a graphics method that **adds** to an existing plot; whereas `hist` creates a **new** plot. Which does which? Consult the help.

```
> hist(lead, breaks = seq(0, 700, by = 40), col = "lightblue",
+     border = "red", main = "Lead concentration by weight, ppm")
> rug(lead)
```



Note the use of the `seq` ("sequence") method to make a list of break points. The `main=` argument is used to specify the main title; there is also a `sub=` argument for a subtitle.

> **Note:** To see the list of named colours, use the `colors` command with no argument: `colors()`. There are many other ways to specify colours; see Rossiter [10, §5.5] and `?colors`.

Q21 : *How do you find out about the various options for the histogram?*
Jump to A21 •

Task 29 : Visualise the **frequency distribution** of lead concentration with a stem plot. •

Stem-and-leaf plots

A **stem-and-leaf** plot, displayed by the `stem` method, shows the numerical values themselves, to some precision:

```
> stem(lead)

  The decimal point is 2 digit(s) to the right of the |

  0 | 4444
  0 | 55555555555555555566666677777777777777888888888888888899999999
  1 | 000001122222233333344444
  1 | 555555666667777888899
  2 | 000011111222334444
  2 | 55667888899
  3 | 001233
  3 |
  4 | 0113
  4 | 668
  5 | 4
  5 |
```

```
6 |
6 | 5
```

Q22 : *What is the advantage of the stem plot over the histogram?* *Jump to A22* •

The stem plot can be difficult to interpret if you are seeing it for the first time. There are two parts: the **stem** (to the left of the `|`) and the **leaf** (to the right). Notice the text that says `The decimal point is 2 digits to the right of the |`. So the first stem `0` and the first leaf |4| mean that the minimum value is approximately `040`, i.e. `40`.

Q23 : *How many observations have a value of approximately* `40`*?* *Jump to A23* •

We can check this with the `head` method applied to the sorted list:

```
> head(sort(lead), 10)

 [1] 37 39 41 42 45 48 48 48 48 48
```

Q24 : *What are the lead values of the observations that were reported as approximately* `40` *in the stem plot?* *Jump to A24* •

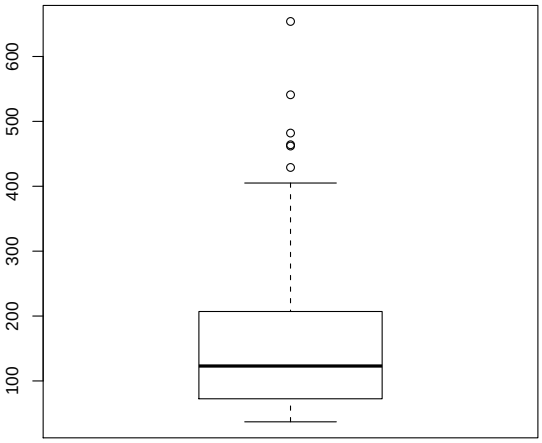
Q25 : *According to the stem plot, what is the approximate maximum lead value?* *Jump to A25* •

Boxplots

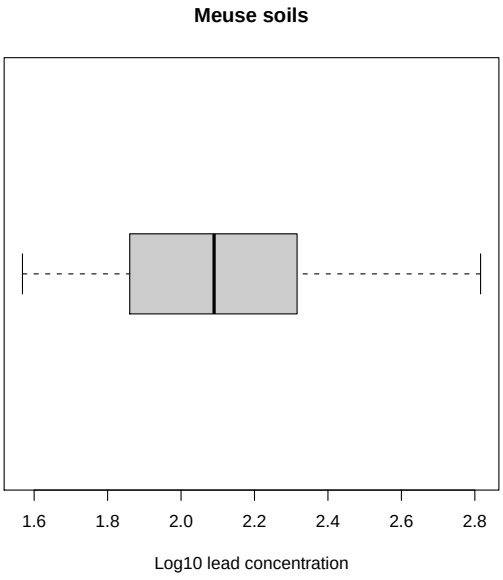
Another useful univariate plot is the **boxplot**, which gives a rough idea of the shape of a **unimodal** distribution.

Task 30 : Visualise the quantiles of distribution of the lead concentration with a box plot; repeat for the base-10 log of the lead concentration. •

```
> boxplot(lead)
```




```
> boxplot(log10(lead), horizontal = T, boxwex = 0.4, xlab = "Log10 lead concentration",
+     main = "Meuse soils", col = "gray80")
```



Optional: A classified boxplot The data displayed in a boxplot can be split by some classifying **factor**; this shows the distribution for each **level** of the factor separately, and allows us to see if the levels differ for the variable being plotted.

Q26 : *Look again at the structure of the* `meuse` *dataframe. Which variables are listed as classified factors?* *Jump to A26* •

It is suspected that floodwater is carrying the metals from upstream indus-

tries; so the classified factor `ffreq` (flood frequency) might well show some differences.

Task 31 : Display the different flood frequency classes. •

The classes of a factor can be listed with the `levels` method:

```
> levels(ffreq)
```

```
[1] "1" "2" "3"
```

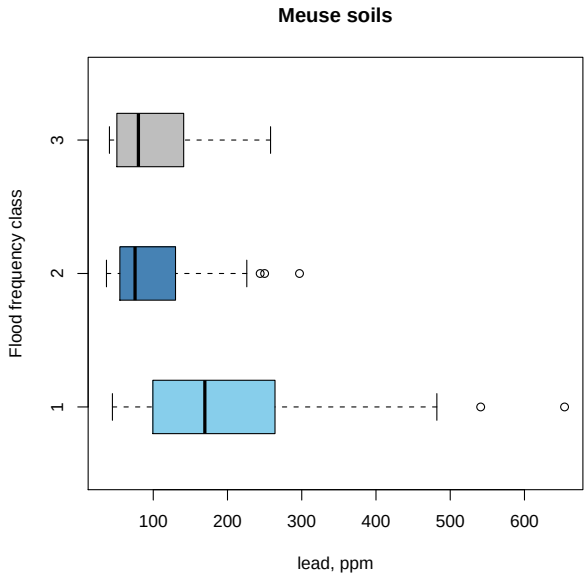
Q27 : *How many flood frequency classes are there in the study area? What are their names?* *Jump to A27* •

Task 32 : Display a boxplot of lead concentration, classified by flood frequency. •

To show the "dependency", we use the formula operator `~`, which can roughly be read as "is described by". So the expression `lead ~ ffreq` in the follwing command can be read "please show the lead content, separately for each flood frequency class".

We also take this opportunity to specify some optional arguments to make a nice graph.

```
> boxplot(lead ~ ffreq, horizontal = T, boxwex = 0.4, col = c("skyblue",
+     "steelblue", "gray"), main = "Meuse soils", ylab = "Flood frequency class",
+     xlab = "lead, ppm")
```



Q28 : *How does the distribution of lead concentration differ by flood fre-*

quency? *Jump to A28*

•

Optional: A sophisticated histogram This is a nice illustration of the power of R graphics. It is included here just to impress you with the possibilities.

Task 33 : Display a histogram of the lead concentration with break points every 50 mg kg^{-1}., with the count in each histogram bin printed on the appropriate bar. Shade the bars according to their count, in a colour ramp with low counts whiter and high counts redder. •

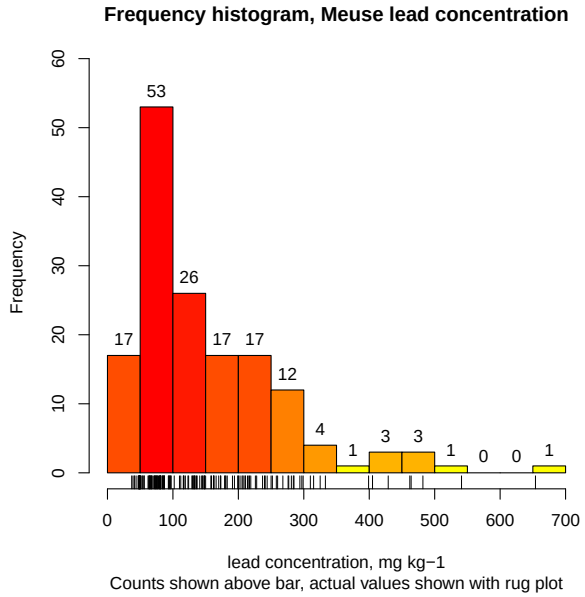
The solution to this task depends on the fact that the `hist` method not only plots a histogram graph, it can also **return** an object which can be **assigned** to an object in the workspace; we can then examine the object to find the counts, breakpoints etc.

We first compute the histogram but don't plot it (`plot=F` argument), then draw it with the `plot` command, specifying a colour ramp, which uses the computed counts, and a title. Then the `text` command adds text to the plot at `(x, y)` positions computed from the class mid-points and counts; the `pos=3` argument puts the text on top of the bar.

```
> h <- hist(lead, breaks = seq(0, 700, by = 50), plot = F)
> str(h)

List of 7
 $ breaks     : num [1:15] 0 50 100 150 200 250 300 350 400 450 ...
 $ counts     : int [1:14] 17 53 26 17 17 12 4 1 3 3 ...
 $ intensities: num [1:14] 0.00219 0.00684 0.00335 0.00219 0.00219 ...
 $ density    : num [1:14] 0.00219 0.00684 0.00335 0.00219 0.00219 ...
 $ mids       : num [1:14] 25 75 125 175 225 275 325 375 425 475 ...
 $ xname      : chr "lead"
 $ equidist   : logi TRUE
 - attr(*, "class")= chr "histogram"

> plot(h, col = heat.colors(length(h$mids))[length(h$count) -
+     rank(h$count) + 1], ylim = c(0, max(h$count) + 5),
+     main = "Frequency histogram, Meuse lead concentration",
+     sub = "Counts shown above bar, actual values shown with rug plot",
+     xlab = "lead concentration, mg kg-1")
> rug(lead)
> text(h$mids, h$count, h$count, pos = 3)
> rm(h)
```



Do you think you can reproduce this with Excel?

7.2 Answers for univariate exploratory graphics

A19 : *The horizontal axis is the value of the variable being summarized (in this case, lead concentration). It is divided into sections ("histogram bins") whose limits are shown by the vertical vars. The vertical axis is the count (frequency) of observations in each bin.* *Return to Q19* •

A20 : *The distribution is strongly right-skewed: high values are increasingly rare. The modal value is in the 50 to 100 mg kg^{-1} range. This is one population.* *Return to Q20* •

A21 : *From the help text:* `?hist`. *Return to Q21* •

A22 : *The stem plot shows the actual values (to some number of significant digits). This allows us to see if there is any pattern to the digits.* *Return to Q22* •

A23 : *4: there are four* `4` *digits following the* `0` *stem.* *Return to Q23* •

A24 : *37, 39, 41, 42; these were all rounded to 40.* *Return to Q24* •

A25 : *650 (actual value is 654).* *Return to Q25* •

A26 : `ffreq`, `soil`, `lime`, *and* `landuse`. *Return to Q26* •

A27 : *Three classes, named by the strings* `"1"`, `"2"`, *and* `"3"`. *The help text (*`?meuse`*) does not give any further information; but from the original report [9] we learn that these correspond to annual, 2–5 year, and infrequent flooding.* *Return to Q27* •

A28 : *The more frequently flooded, the more metal. This is especially true for the highest concentrations (extremely polluted sites).* *Return to Q28* •

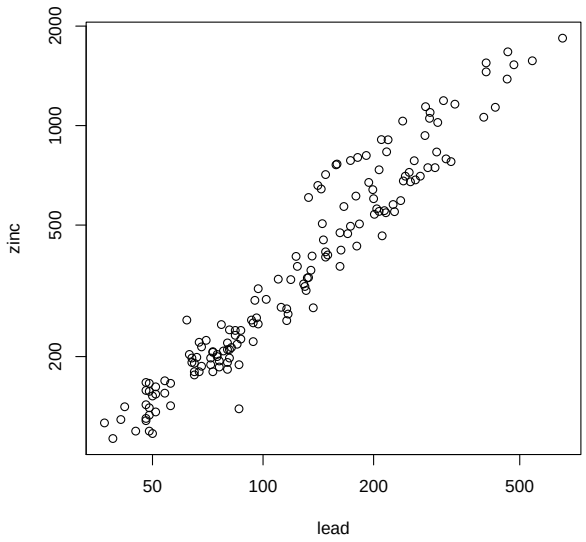
7.3 Bivariate exploratory graphics

When several variables have been collected, it is natural to compare them.

Task 34 : Display a **scatterplot** of lead vs. zinc concentation •

We again use `plot`, but in this case there are two variables, so a scatterplot is produced. That is, `plot` is an example of a **generic** method: its behaviour changes according to the **class** of object it is asked to work on,

```
> plot(lead, zinc, log = "xy")
```



Note that we specify the optional argument `log="xy"` to display both of these right-skewed variables on a logarithmic scale.

Q29 : *What is the relation between the two metals?* *Jump to A29* •

Interacting with plots

Some R graphics allow **interaction**: we first display the graph and then query it with the mouse.

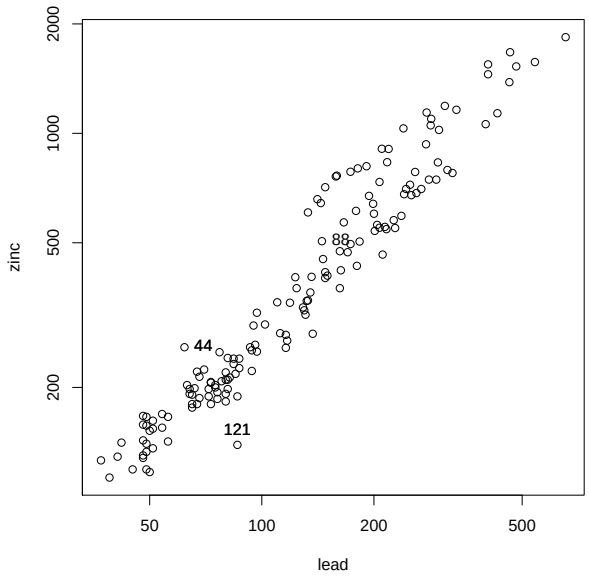
Task 35 : Identify the observations which do not fit the general pattern of a very close relation between lead and zinc. •

For this we use the `identify` method, specifying the same plot coördinates as the previous `plot` command (i.e. from the plot that is currently displayed):

```
> (pts <- identify(lead, zinc))
```

After `identify` is called, switch to the graphics window, **left-click** with the mouse on points to identify them, and **right-click** to exit. The plot will now show the row names of the selected points, and these row names are also shown in the console:

```
[1] 44  121
```



Q30 : *Which are the unusual observations?* *Jump to A30* •

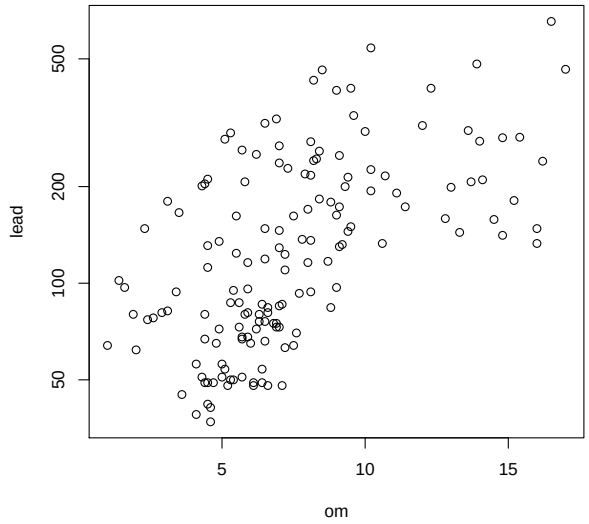
Remove the temporary variable:

```
> rm(pts)
```

Let's look at another bivariate relation, between the metal lead and the soil organic matter (SOM). Studies have often shown that SOM can bind metals, so that we would expect soils with high SOM to have high metal concentrations. Is this true?

Task 36 : Display a scatterplot of lead vs. SOM. •

```
> plot(lead ~ om, log = "y")
```



This example shows another way of specifying the variables in a scatterplot: with the **~ formula operator**, which we saw above in the classified boxplot and will meet many times again. It can be translated roughly as "(left side) depends on, or is explained by, (right side)". So in this case the lead content is explained by SOM, so by convention the dependent variable (lead) is shown on the vertical ("y") axis.

Note that since only lead is skewed, we only specify a log scale on its axis.

Q31 : *What is the relation between lead and SOM? Does this plot suggest that SOM could be used to predict lead content?* *Jump to A31* •

7.4 Answers for bivariate exploratory graphics

A29 : *They are strongly and positively related: the pollution level with lead is similar to that with zinc.* *Return to Q29* •

A30 : *There are no really far-out observations, but point 44 has lower lead than would be expected from its zinc content, and point 121 has much lower zinc than would be expected from its lead content. The other points fit the pattern very well.* *Return to Q30* •

A31 : *They are positively but weakly related: the points form a diffuse "cloud". This suggests that SOM would be a poor predictor of lead concentration.* *Return to Q31* •

7.5 Visualizing the transect

Back in §3 we computed sample points along a transect. We can visualize these with the `stripchart` method.

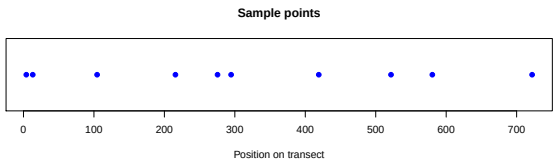
When we first ask R to display a plot, it automatically opens a **graphics window** of a default size and shape. Here we want a wide plot, so we open a second graphics window, defining its shape directly:

```
> windows(h = 3, w = 10)
```

The `windows` method opens a new graphics window under the MS-Windows operating system[7]. We specify the height (`h=`) as 3 inches[8] and the width (`w=`) as 10 inches.

Now we recompute the sample and display it in a stripchart, customizing the colour, printing character (using the `pch` graphics argument), its size (using the `cex` graphics argument) and labels:

```
> stripchart(runif(10, 0, 1000), pch = 20, cex = 1.5, col = "blue",
+     xlab = "Position on transect", main = "Sample points")
```



When we are done viewing this, we can remove the second graphics window with the `dev.off` method:

```
> dev.off()
```

This turns off the active graphics device, which in this case is the new window.

Q32 : *These ten points were sampled from the uniform distribution. Is the actual distribution of the ten points "uniform"? How can you explain the discrepancy?* *Jump to A32* •

7.6 Answers for visualizing the transect

A32 : *In common language, we think of "uniform" as meaning "equally distributed". This is certainly not the case here! In probability, the "uniform" distribution is the one where every value has an* ***equal chance*** *of being selected. So the first value selected has no influence on the the second, and so forth. There is no reason for the values to spread out evenly along the transect.*

[7] On other OS there are equivalent commands, e.g. `quartz` for Mac OS X and `x11` for Unix systems

[8] 1 inch = 2.54 cm

*If we do want equally-spaced sample points, that's a **systematic** sample, not a **random** sample.* *Return to Q32* •

8 Linear modelling

Modelling in R is a vast subject; here we just introduce the basic model formula syntax, model output, and diagnostic graphics.

In §7.3 we saw that Pb and Zn concentrations were closely related. We may wish to build a **linear model** to summarize this relation.

Task 37 : Build a linear model of Zn concentration as explained (modelled) by Pb concentration in the Meuse soil samples. •

Since both variables were strongly right-skewed, we should use the log-transform for both. We use the `log10` method for easy interpretation.

The model itself is computed with the `lm` "linear models" method, and summarized with the generic `summary` method. The **model formula** has the form:

```
LHS ~ RHS
```

where the `LHS` or "left-hand side" is the variable to be modelled, and the `RHS` or "right-hand side" is a formula with one or more explanatory variables. In this simple case we only specify one variable in the RHS.

```
> lm.zn.pb <- lm(log10(zinc) ~ log10(lead), data = meuse)
> summary(lm.zn.pb)

Call:
lm(formula = log10(zinc) ~ log10(lead), data = meuse)

Residuals:
    Min      1Q  Median      3Q     Max
-0.2527 -0.0528 -0.0182  0.0499  0.2092

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.3691     0.0469    7.87    6e-13 ***
log10(lead)   1.0476     0.0223   47.07   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0799 on 153 degrees of freedom
Multiple R-squared: 0.935,        Adjusted R-squared: 0.935
F-statistic: 2.22e+03 on 1 and 153 DF,  p-value: <2e-16
```

The summary shows the most important features of the model.

Q33 : *What are the largest positive and negative **residuals**, i.e. unexplained variation in a single observation?* *Jump to A33* •

Q34 : *How much log10 Zn is there predicted to be in the case that log10 Pb*

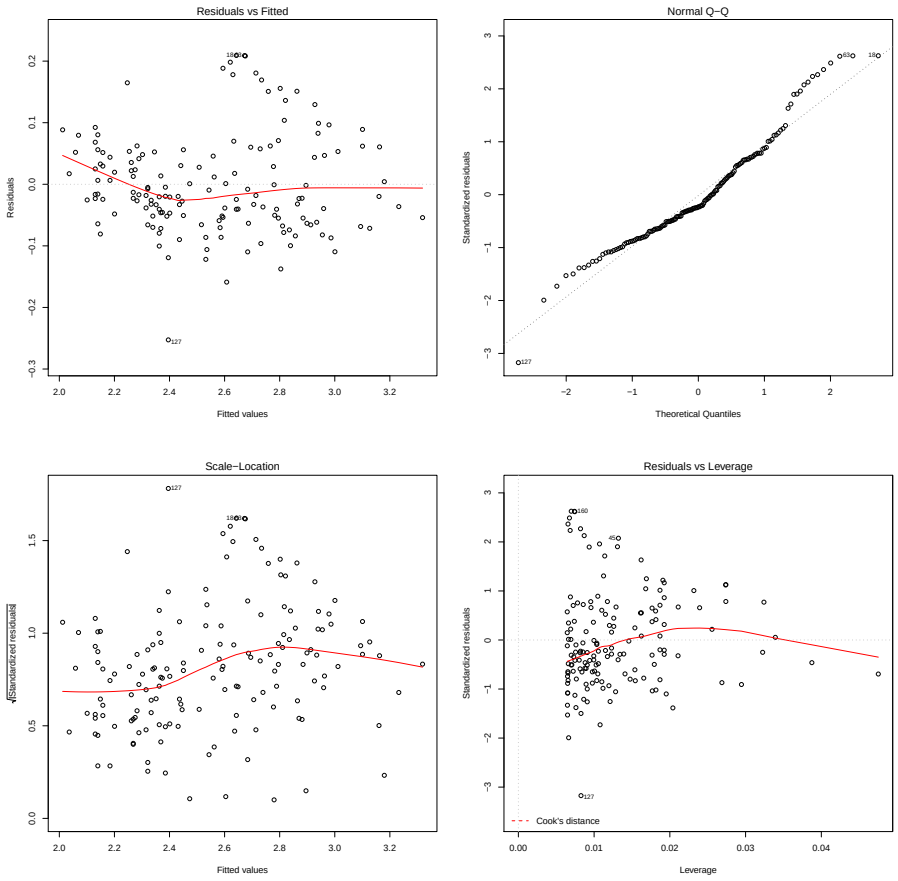
is zero? (i.e. Pb=1) (hint: see the `(Intercept)` *model coefficient).* *Jump to A34* •

Q35 : *For each log10 unit of Pb increase, what is the increase of log10 Zn (hint: see the* `log10(lead)` *model coefficient).* *Jump to A35* •

Q36 : *How much of the variance in Zn concentration is explained by the model, according to the adjusted R^2?* *Jump to A36* •

The results of a modelling exercise should always be checked to see if the modelling assumptions were met. Here we show some standard diagnostic plots provided by R with the generic `plot` method, which automatically calls the `plot.lm` method.

```
> par(mfrow = c(2, 2))
> plot(lm.zn.pb)
> par(mfrow = c(1, 1))
```



There are four standard plots:

1. **Residuals vs. fitted values**: there should be no pattern (e.g. "bow shape") or change in variability across the range ("heteroscedascity");

2. **Normal Q-Q plot of the residuals**: they should be normally-distributed (all on the 1:1 line, dense in the middle and increasingly-sparse for larger absolute residuals);
3. **Scale-location**: another way to look for heteroscedascity;
4. **Residuals vs. leverage**: shows which observations most influence the model fit.

In this case all the diagnostic plots look good; the only problem is a few poorly-fit observations, which are identified in the plots.

Q37 : *According to the diagnostics, which observations are poorly-fit? Which of these appears to be really outside the model?* *Jump to A37* •

8.1 Answers for linear modelling

A33 : *Residuals are (Actual-Predicted), so the largest* ***over****-prediction is* 0.2527*, largest* ***under****-prediction is* 0.2092 *log10 units.* *Return to Q33* •

A34 : *At zero log10 Pb there is predicted to be* 0.3691 *log10 units of Zn* *Return to Q34* •

A35 : *For each log10 unit of Pb increase, log10 of Zn increases by* 1.0476*, i.e. a bit more than 1:1.* *Return to Q35* •

A36 : *93.5%* *Return to Q36* •

A37 : *18, 63, and 127 are identified; 127 has an extreme standardized residual (< -3) and much higher predicted Zn than actual.* *Return to Q37* •

We can see this with the `fitted` and `residuals` model access methods:

```
> log10(meuse["127", "zinc"])

[1] 2.143

> fitted(lm.zn.pb)["127"]

   127
2.3957

> residuals(lm.zn.pb)["127"]

     127
-0.25268
```

We are finished with this model, so we remove it from the workspace with the `rm` method:

```
> rm(lm.zn.pb)
```

9 * User-defined functions

This **optional** section introduces user-defined functions, which are typical of R and very useful. However, if your head is already spinning from what we've already covered in this exercise, you can come back to this later. In that case skip ahead to the final section, §10.

If you have any programming experience, the way we solved the task in §6.4 probably bothered you, because we repeated the exact same command five times, just changing the threshold. Can R do this in a more systematic way? Of course!

Warning! If you have never done any computer programming, this will most likely be confusing the first time through. Read through and try to get the general idea; we will see many more examples in later exercises.

First, we define our own `function` (roughly speaking, our own command) with the `function` method:

```
> p.high <- function(level) {
+     round(100 * sum(lead > level)/length(lead), 1)
+ }
> ls()

[1] "meuse"  "p.high"
```

> **Note:** The above output shows the function being entered on several lines, as indicated by the continuation prompt `+`. You can also enter the command all on one line:

```
> p.high <- function(level) {round(100*sum(lead > level)/length(lead),1)}
```

Notice that `p.high` (the function we just defined) is now an object in the workspace, because we put it there with the `<-` assignment operator.

The function has one **argument** (the part that varies each time the function is called). Here it is named `level`. The rest of the function is the same command we repeated five times, but instead of directly naming the critical level (e.g. `600`) it uses the function argument `level`.

Now we've got our own function; let's use it. Make a list of the levels, using the `c` ("catenate", from Latin for "make a chain") method:

```
> lvls <- c(600, 300, 200, 100, 30)
```

Now build a list of the proportions, applying the `p.high` function to each element of the list `lvls` in turn, using the `for` operator.

```
> p <- NULL
> for (l in lvls) p <- c(p, p.high(l))
```

The `for` operator assigns each value in the list `lvls` to the loop variable `l`, which is then the argument to `p.high`.

So now we have `p`, with the percentages:

```
> p
```

```
[1]   0.6   8.4  27.1  54.8 100.0
```

We can make a nice table from the levels and percentages, using the `data.frame` ("make a data frame") method to make a temporary data frame, which will be printed out as a table with the data frame's field names, which we name as arguments to the `data.frame` method:

```
> (data.frame(level = lvls, percent.higher = p))
```

```
  level percent.higher
1   600            0.6
2   300            8.4
3   200           27.1
4   100           54.8
5    30          100.0
```

It's good practice to remove objects we no longer need from the workspace, using the `rm` method:

```
> rm(p, l, lvls, p.high)
> ls()
```

```
[1] "meuse"
```

Wasn't that fun?

10 Quitting R

When you stop an R session, R gives you the opportunity to save your current workspace in the file `.RData`. If you start R again in the same directory (e.g. by double-clicking on the `.RData` file in Windows Explorer), the workspace will be loaded again.

In preparation for this, it's good practice to remove objects that you no longer need from the workspace.

Task 38 : Remove un-needed objects from the workspace. •

In the present case, we have one data object, the Meuse dataset `meuse` and (if all the steps were followed correctly) no temporary variables:

```
> ls()
```

```
[1] "meuse"
```

The Meuse dataset is also on the search path:

```
> search()
```

```
 [1] ".GlobalEnv"        "meuse"             "package:gstat"
 [4] "package:sp"        "package:stats"     "package:graphics"
 [7] "package:grDevices" "package:utils"     "package:datasets"
[10] "package:lattice"   "package:methods"   "Autoloads"
[13] "package:base"
```

When we are working with an attached dataset, it is good practice to **remove** it from the search path with the `detach` method. In this way fields in the data frame are no longer visible, even though the object remains in the workspace:

```
> summary(lead)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   37.0    72.5   123.0   153.0   207.0   654.0

> detach(meuse)
> search()

 [1] ".GlobalEnv"        "package:gstat"     "package:sp"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:lattice"
[10] "package:methods"   "Autoloads"         "package:base"

> summary(lead)

Error: object "lead" not found
Error in summary(lead) : error in evaluating the argument 'object' in
selecting a method for function 'summary'
```

We can re-load the Meuse dataset any time, and any temporary variables were only used in this session. So they all can be deleted, using the `rm` ("remove") method.

```
> rm(meuse)
> ls()

character(0)
```

> **Note:** If you are **sure** you want to clear the entire workspace, you can use this form:
>
> `> rm(list = ls())`
>
> This makes a list of all the objects (using the `ls` method) and then passes it to the `rm` method, so that everything is erased.

Saving the console log

Before leaving R, or indeed at any time, we may well want to **save the console log**, that is, all the commands we issued and the results that R printed in the console window. We can then review this document, edit it to remove unimportant or erroneous commands and output, add our own commentary, review our processing steps etc.

Task 39 : Save the console log to a text file. •

In the Windows GUI, bring the console window to the front (e.g. click on its title bar) and select menu command `File | Save as ...`; the default file name is `lastsave.txt`. It is good practice to change this name to the current date, e.g. `19Feb07log.txt`.

Now we can leave R.

Task 40 : Stop R. •

We do this with the `q` ("quit") method; under MS-Windows you can select the `File | Exit` menu item; under Mac OS X the `R | Quit R` menu item or their keyboard shortcuts.

```
> q()
```

You are asked if you want to save workspace image. Generally you answer Yes; this creates or modifies the special file `.RData` in the current working directory.

Task 41 : Open the saved console log in a plain-text editor and review its contents. •

You may want to add some comments or edit out your mistakes.

11 Self-test

This section is a small self-test of how well you mastered this exercise. You should be able to complete the tasks and answer the questions with the knowledge you have gained from the exercise. Please **submit your answers (including graphical output) to the instructor** for grading and sample answers.

Task 1 : Load the example dataset `pcb` which is provided with the `gstat` package and summarize it. •

Q1 : *How many variables and observations does this dataset have?* •

Q2 : *What is the meaning of variable* `PCB138`*?* •

> **Note:** PCB 138 is an organic chemical, 2,2',3,4,4',5-Hexachlorbiphenol that is a seriously toxic compound to many forms of animal life. It was used in many industrial products, especially transformers. It often accumulates in marine muds.

Q3 : *What are the minimum, maximum, and median values of* `PCB138`*?* •

Task 2 : Make a histogram of `PCB138`. Save the plot as a JPG graphics file. •

Q4 : *Describe the shape of the distribution of* `PCB138`. •

Task 3 : Make a scatterplot of the PCB concentration vs. distance from the coast. Save the plot as a JPG graphics file. •

Q5 : *What is the relation, if any, between PCB concentration and distance from the coast?* •

Task 4 : Make a data frame with just the observations that are further than 100 m from the coast and summarize their PCB concentrations. •

Q6 : *How many observations are further than 100 m from the coast?* •

Q7 : *How do their PCB concentrations compare to the full data set?* •

Task 5 : Remove the two objects you created from the workspace. •

References

[1] P A Burrough and R A McDonnell. *Principles of geographical information systems*. Oxford University Press, Oxford, 1998. 12

[2] J Fox. The R Commander: a basic-statistics graphical user interface for R. *Journal of Statistical Software*, 14(9):42, 2005. URL http://www.jstatsoft.org/v14/i09/. 3

[3] R Ihaka and R Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996. 1

[4] F Leisch. *Sweave User's Manual*. TU Wein, Vienna (A), 2.7.1 edition, 2006. URL http://www.stat.uni-muenchen.de/~leisch/Sweave/. 1

[5] F Leisch. Sweave, part I: Mixing R and LaTeX. *R News*, 2(3):28–31, December 2002. URL http://CRAN.R-project.org/doc/Rnews/. 1

[6] P Murrell. *R Graphics*. Chapman & Hall/CRC, Boca Raton, FL, 2005. ISBN 1-584-88486-X. 23

[7] E J Pebesma. Multivariable geostatistics in S: the gstat package. *Computers & Geosciences*, 30(7):683–691, 2004. 9

[8] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004. URL http://www.R-project.org. ISBN 3-900051-07-0. 1

[9] M G J Rikken and R P G Van Rijn. Soil pollution with heavy metals - an inquiry into spatial variation, cost of mapping and the risk evaluation of copper, cadmium, lead and zinc in the floodplains of the Meuse west of Stein, the Netherlands. Doctoraalveldwerkverslag, Dept. of Physical Geography, Utrecht University, 1993. 12, 21, 31

[10] D G Rossiter. *Introduction to the R Project for Statistical Computing for use at ITC*. International Institute for Geo-information Science & Earth Observation (ITC), Enschede (NL), 3.7 edition, 2009. URL http://www.itc.nl/personal/rossiter/teach/R/RIntro_ITC.pdf. 24, 25

[11] Deepayan Sarkar. Lattice. *R News*, 2(2):19–23, June 2002. URL http://CRAN.R-project.org/doc/Rnews/. 9

Index of R Concepts

< operator, 19
<- operator, 6
<= operator, 19
== operator, 19
> operator, 19
>= operator, 19
[] operator, 15

attach, 14

border graphics argument, 24
breaks graphics argument, 24

c, 38
cex graphics argument, 34
col graphics argument, 24
colors, 25

data, 10, 11
data.frame, 39
datasets package, 10
detach, 40
dev.off, 34

exp, 4

fitted, 37
fix, 15
for operator, 38
function, 38

graphics package, 23
gstat package, 1, 9, 10, 41

head, 26
help, 5
hist, 23, 24, 29

identify, 32

lattice package, 1, 9, 23
length, 19
levels, 28
library, 9
lm, 35
log10, 35
ls, 6, 8, 11, 40

main graphics argument, 24
max, 13
mean, 13
median, 13
meuse dataset, 10, 11, 39
min, 13

pcb dataset, 41
pch graphics argument, 34
pi, 4
plot, 29, 31, 32, 36
plot.lm, 36
print, 6

q, 41
quantile, 13, 14
quartz, 34

range, 13
require, 9
residuals, 37
rm, 8, 22, 37, 39, 40
round, 6, 7
row.names, 16
rug, 24
runif, 5–7

sd, 13
search, 9, 10, 14
seq, 14, 25
set.seed, 8
sort, 7, 8
sp package, 1, 10
spatial package, 1
stem, 25
str, 11
stripchart, 34
sum, 19
summary, 13, 35

text, 29

var, 13

which.max, 15, 22
which.min, 15, 18
windows, 34

x11, 34