Demonstration: Warm Up with **R**: Programming Language for Data Analysis

Wen-wen Tung

5/24/2019

About **R**

R was designed to make data analysis and statistics easier for analysts. The **R**-language Definition (http://cran.r-project.org/doc/manuals/R-lang.html) defines what **R** code means and how it should work. The most popular implementation is the one from The **R** Project for Statistical Computing (http://www.r-project.org).

R is tailored for data analysis. Known strengths of **R** include:

- Free and open source, and available on every major platform.
- Vast number of packages for statistical modelling, machine learning, visualisation, and importing and manipulating data.
- Backed by research community that develops cutting edge methods. Large expert and general user communities therefore peer-support.
- Tools to communicate results in html or pdf reports, or to create interactive websites.
- A strong foundation in functional programming with many tools for the creation and manipulation of functions.
- Connect to high-performance programming languages such as C, Fortran, and C++.

Known challenges for **R**: Most users are not programmers. **R** trades speed for productivity. Already not a particularly fast programming language, poorly written codes can be very slow and hog memory.

This set of notes and the accompanying exercises provide an introductory tour to **R** basics, such as how to navigate its workspace, add a library/package, use its data structure, assign values to variables, perform simple calculation, and generate random samples.

---

1. Starting **R**

1.1 Setting **R** working directory

Use following command to set your workspace.

```
### Set your working directory
#setwd("/home/student/rwd")

### Find out which directory you are
getwd()
```

```
## [1] "/Users/wen-wen/Something/Class/TaiwanNTU2019/NTU_Exercises"
```

1.2 Getting help

Method 1, use question mark `?` or `help` if you know the name of the function.

```
?read.table
help(read.table)
```

Method 2, use `help.search` if you know the subject on which you want help (data input, for example) but not the exact name of the function.

```
help.search("data input")
```

Method 3, `find` function tells you what package something is in:

```
find("lowess")
```

```
## [1] "package:stats"
```

Method 4, `apropos` gives the names of all objects in the search list that match your (partial) inquiry:

```
apropos("lm")
```

```
##  [1] ".colMeans"       ".lm.fit"         "colMeans"
##  [4] "confint.lm"      "contr.helmert"   "dummy.coef.lm"
##  [7] "getAllMethods"   "glm"             "glm.control"
## [10] "glm.fit"         "KalmanForecast"  "KalmanLike"
## [13] "KalmanRun"       "KalmanSmooth"    "kappa.lm"
## [16] "lm"              "lm.fit"          "lm.influence"
## [19] "lm.wfit"         "model.matrix.lm" "nlm"
## [22] "nlminb"          "predict.glm"     "predict.lm"
## [25] "residuals.glm"   "residuals.lm"    "summary.glm"
## [28] "summary.lm"
```

Last but not least, `help.start()` starts the HTML interface to on-line help (using a web browser available at your machine).

1.3 Use **R** packages

Use a library already built-in the base package of R:

```
library(lattice)
```

Use `help` to discover the contents of library packages, for example:

```
library(help=lattice)
```

View the full list of the contents of a library using `objects` with `search()`:

```
objects(grep("lattice", search()))
```

```
##   [1] "as.factorOrShingle"          "as.shingle"
##   [3] "axis.default"                "banking"
##   [5] "barchart"                    "barley"
##   [7] "bwplot"                      "canonical.theme"
##   [9] "cloud"                       "col.whitebg"
##  [11] "contourplot"                 "current.column"
##  [13] "current.panel.limits"        "current.row"
##  [15] "densityplot"                 "diag.panel.splom"
##  [17] "do.breaks"                   "dotplot"
##  [19] "draw.colorkey"               "draw.key"
##  [21] "environmental"               "equal.count"
##  [23] "ethanol"                     "histogram"
##  [25] "is.shingle"                  "larrows"
##  [27] "lattice.getOption"           "lattice.options"
##  [29] "latticeParseFormula"         "level.colors"
##  [31] "levelplot"                   "llines"
##  [33] "lplot.xy"                    "lpoints"
##  [35] "lpolygon"                    "lrect"
##  [37] "lsegments"                   "ltext"
##  [39] "ltransform3dMatrix"          "ltransform3dto3d"
##  [41] "make.groups"                 "melanoma"
##  [43] "oneway"                      "packet.number"
##  [45] "packet.panel.default"        "panel.3dscatter"
##  [47] "panel.3dwire"                "panel.abline"
##  [49] "panel.arrows"                "panel.average"
##  [51] "panel.axis"                  "panel.barchart"
##  [53] "panel.brush.splom"           "panel.bwplot"
##  [55] "panel.cloud"                 "panel.contourplot"
##  [57] "panel.curve"                 "panel.densityplot"
##  [59] "panel.dotplot"               "panel.error"
##  [61] "panel.fill"                  "panel.grid"
##  [63] "panel.histogram"             "panel.identify"
##  [65] "panel.identify.cloud"        "panel.identify.qqmath"
##  [67] "panel.levelplot"             "panel.levelplot.raster"
##  [69] "panel.linejoin"              "panel.lines"
##  [71] "panel.link.splom"            "panel.lmline"
##  [73] "panel.loess"                 "panel.mathdensity"
##  [75] "panel.number"                "panel.pairs"
##  [77] "panel.parallel"              "panel.points"
##  [79] "panel.polygon"               "panel.qq"
##  [81] "panel.qqmath"                "panel.qqmathline"
##  [83] "panel.rect"                  "panel.refline"
##  [85] "panel.rug"                   "panel.segments"
##  [87] "panel.smoothScatter"         "panel.spline"

##  [89] "panel.splom"                 "panel.stripplot"
##  [91] "panel.superpose"             "panel.superpose.2"
##  [93] "panel.superpose.plain"       "panel.text"
##  [95] "panel.tmd.default"           "panel.tmd.qqmath"
##  [97] "panel.violin"                "panel.wireframe"
##  [99] "panel.xyplot"                "parallel"
## [101] "parallelplot"                "prepanel.default.bwplot"
## [103] "prepanel.default.cloud"      "prepanel.default.densityplot"
## [105] "prepanel.default.histogram"  "prepanel.default.levelplot"
## [107] "prepanel.default.parallel"   "prepanel.default.qq"
## [109] "prepanel.default.qqmath"     "prepanel.default.splom"
## [111] "prepanel.default.xyplot"     "prepanel.lmline"
## [113] "prepanel.loess"              "prepanel.qqmathline"
```

```
## [115] "prepanel.spline"              "prepanel.tmd.default"
## [117] "prepanel.tmd.qqmath"          "qq"
## [119] "qqmath"                       "rfs"
## [121] "Rows"                         "shingle"
## [123] "show.settings"                "simpleKey"
## [125] "simpleTheme"                  "singer"
## [127] "splom"                        "standard.theme"
## [129] "strip.custom"                 "strip.default"
## [131] "stripplot"                    "tmd"
## [133] "trellis.currentLayout"        "trellis.device"
## [135] "trellis.focus"                "trellis.grobname"
## [137] "trellis.last.object"          "trellis.panelArgs"
## [139] "trellis.par.get"              "trellis.par.set"
## [141] "trellis.switchFocus"          "trellis.unfocus"
## [143] "trellis.vpname"               "USMortality"
## [145] "USRegionalMortality"          "which.packet"
## [147] "wireframe"                    "xscale.components.default"
## [149] "xyplot"                       "xyplot.ts"
## [151] "yscale.components.default"
```

Attach the `barley` dataframe in the `lattice` library to your current workspace (global environment):

```
attach(barley)

### Optionally, you could take a look of (or edit it)
### Remove the # sign before using the command
# fix(barley)
```

Installing packages (you will need these)

```
### Remove the # sign before using the commands
# install.packages("ncdf4")
# install.packages("tidyverse")
# install.packages("maps")
# install.packages("fields")
# install.packages("gapminder")
# install.packages("mlbench")
# install.packages("HURDAT")
```

1.4 Keep the workspace clean

To see what variables you have created in the current session:

```
objects()
```

```
## character(0)
```

To see which packages and dataframes are currently attached:

```
search()
```

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

Note that The global environment `.GlobalEnv` , *aka* the user's workspace, is the first item on the search path. It can also be accessed by `globalenv()` .

Here, put a variable `kiwi` with value `1` into the global environment:

```
kiwi <- 1
```

See `kiwi` in the global environment. There are two methods:

```
objects(1)
```

```
## [1] "kiwi"
```

```
objects(globalenv())
```

```
## [1] "kiwi"
```

Examine variables within the dataframe `barley` and print one of its variables `year` .

```
objects(barley)
```

```
## [1] "site"    "variety" "year"    "yield"
```

```
year
```

```
##   [1] 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931
##  [15] 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931
##  [29] 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931
##  [43] 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931
##  [57] 1931 1931 1931 1931 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932
##  [71] 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932
##  [85] 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932
##  [99] 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932
## [113] 1932 1932 1932 1932 1932 1932 1932 1932
## Levels: 1932 1931
```

Remove `kiwi` from the global environment.

```
rm(kiwi)
```

Detach `barley` dataframe, therefore its variables are no longer accessible directly by name.

```
detach(barley)
search()
```

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

To get rid of everything in the user's workspace:

```
rm(list=ls())
objects(1)
```

```
## character(0)
```

2. A simple computing sample session

This section introduces to you some features of the **R** environment with examples. Practice at your own pace after the lecture. Many features will be unfamiliar at first. Some will be explained in the next section.

Some useful references for further reading:

- **R** Base Package Documentation (https://stat.ethz.ch/R-manual/R-devel/RHOME/library/base/html/00Index.html)
    - Arithmetic Operators (https://stat.ethz.ch/R-manual/R-devel/RHOME/library/base/html/Arithmetic.html)
    - Comparison Operators (https://stat.ethz.ch/R-manual/R-devel/RHOME/library/base/html/Comparison.html)
    - Logic Operators (https://stat.ethz.ch/R-manual/R-devel/RHOME/library/base/html/Logic.html)
    - Reserved Words in **R** (https://stat.ethz.ch/R-manual/R-devel/RHOME/library/base/html/Reserved.html)
    - Extract or Replace Parts of an **R** Object (https://stat.ethz.ch/R-manual/R-devel/RHOME/library/base/html/Extract.html)
- Lattice Graphics (https://stat.ethz.ch/R-manual/R-patched/library/lattice/html/Lattice.html)
- Tidyverse (https://www.tidyverse.org/)

2.1 Linear regression and nonparametric regression using random variables as examples

Generate two pseudo-random normal vectors of x- and y-coordinates.

```
x <- rnorm(50)
y <- rnorm(x)
```

Calculate basic statistics of x: mean, median, standard deviation, variance, correlation, or covariance.

```
mean(x)
```

```
## [1] 0.04131912
```

```
median(x)
```

```
## [1] -0.07504731
```

```
sd(x)
```

```
## [1] 0.8635204
```

```
var(x)
```

```
## [1] 0.7456675
```

To calculate correlation and covariance between x and y vectors:

```
cor(x,y)
```

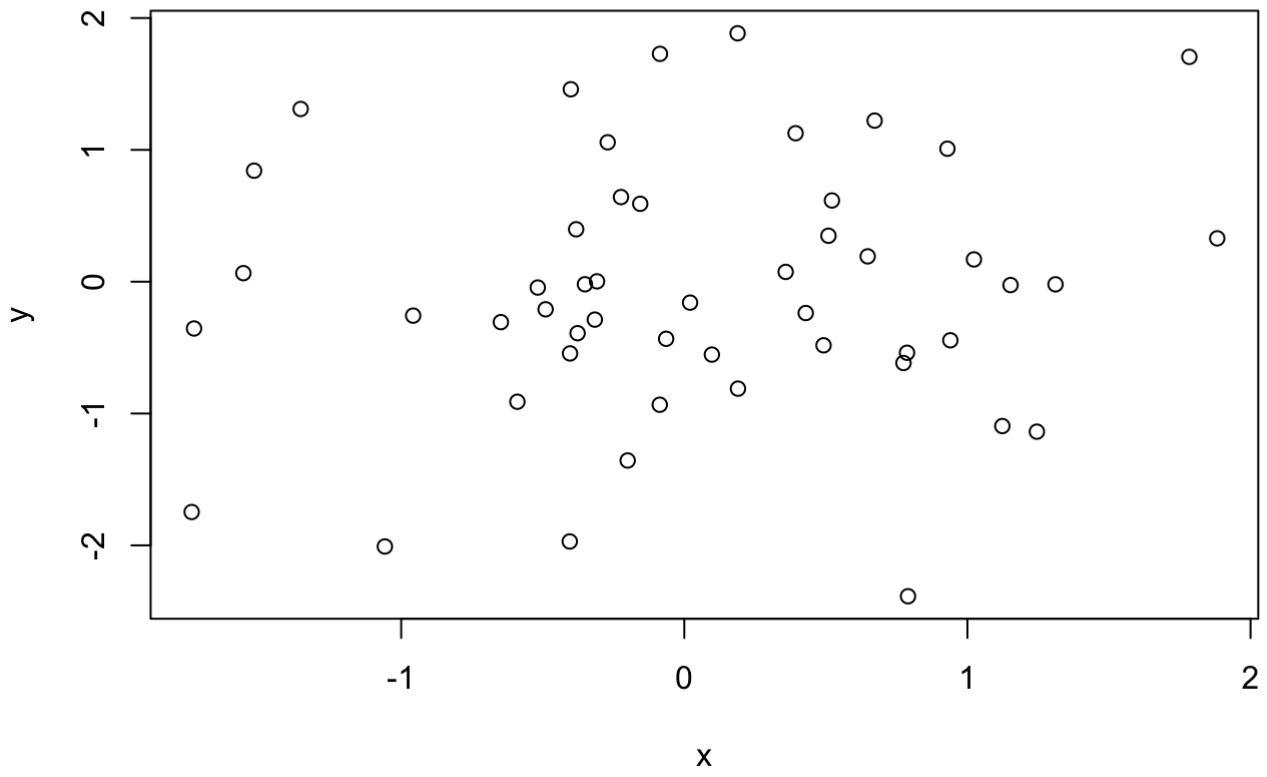
```
## [1] 0.1142728
```

```
cov(x,y)
```

```
## [1] 0.09576404
```

Plot the (x,y) points in the plane. A graphics window will appear automatically.

```
plot(x, y)
```



See which **R** objects are now in the **R** workspace.

```
ls()
```

```
## [1] "x" "y"
```

Remove objects no longer needed.

```
rm(x, y)
```

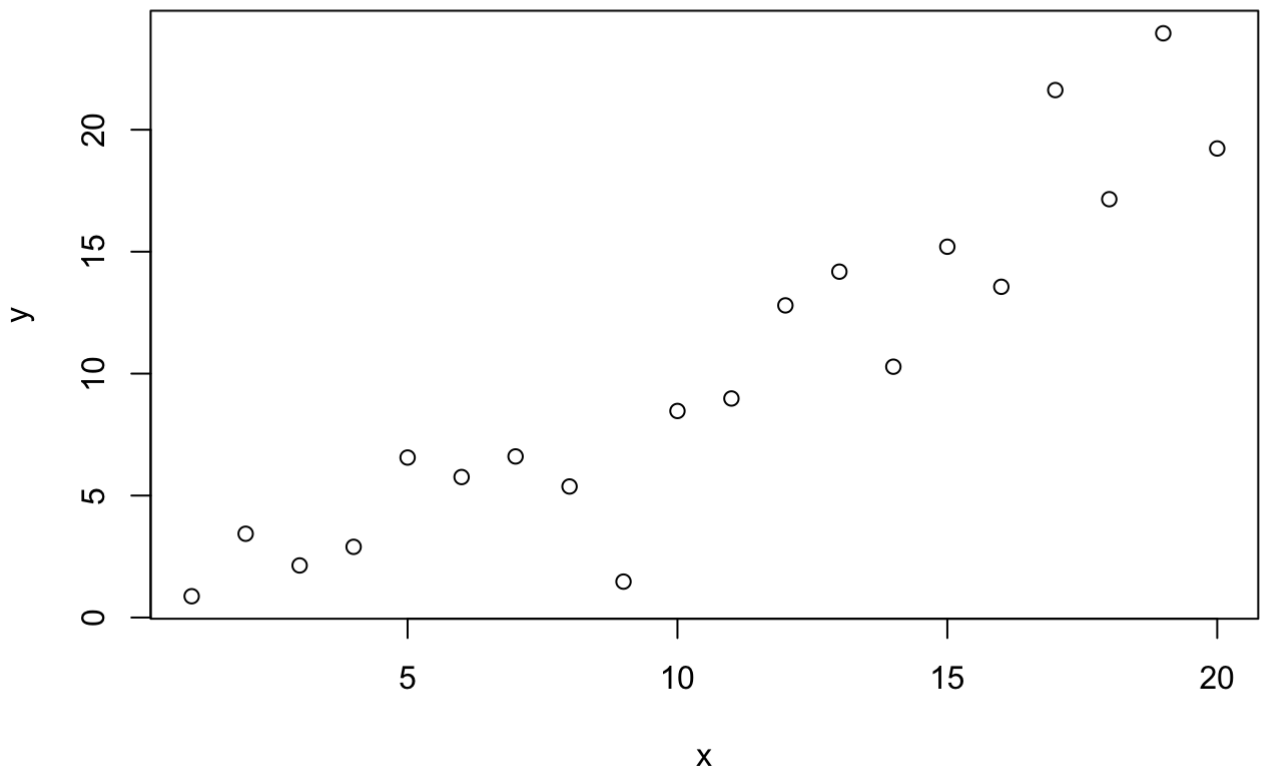
In the following, we create a synthesized dataset to demonstrate weighted regression. In this dataset, the common least squares assumption of constant standard deviation in the y-residuals is violated, a situation called heteroscedasticity.

Make a vector x = (1, 2, …, 20).

```
x <- 1:20
```

Assign a 'weight' vector of standard deviations, then calculate y. Plot the data.

```
w <- 1 + sqrt(x)/2
#### Note y = x + perturbations
set.seed(8)
y= x + rnorm(x)*w
plot(x,y)
```



Make a dataframe of two columns, x and y, and look at it.

```
dummy <- data.frame(x=x, y=y)
dummy
```

```
##     x          y
## 1   1  0.8731209
## 2   2  3.4346528
## 3   3  2.1351294
## 4   4  2.8983300
## 5   5  6.5589587
## 6   6  5.7599914
## 7   7  6.6044395
## 8   8  5.3725348
## 9   9  1.4723708
## 10 10  8.4689347
## 11 11  8.9802307
## 12 12 12.7978951
## 13 13 14.1810857
## 14 14 10.2837436
## 15 15 15.2034551
## 16 16 13.5608846
## 17 17 21.6255565
## 18 18 17.1522217
## 19 19 23.9543896
## 20 20 19.2319197
```

Fit a simple linear regression and look at the analysis. With y to the left of the tilde, we are modelling y dependent on x.

```
fm <- lm(y ~ x, data=dummy)
summary(fm)
```

```
##
## Call:
## lm(formula = y ~ x, data = dummy)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.9393 -1.1681  0.2576  1.2311  4.7708
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.2830     1.2831   -1.00    0.331
## x             1.0772     0.1071   10.06 8.19e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.762 on 18 degrees of freedom
## Multiple R-squared:  0.8489, Adjusted R-squared:  0.8405
## F-statistic: 101.1 on 1 and 18 DF,  p-value: 8.185e-09
```

Since we know the (non-constant) standard deviations, we can do a weighted regression.

```
fm1 <- lm(y ~ x, data=dummy, weight=1/w^2)
summary(fm1)
```

```
##
## Call:
## lm(formula = y ~ x, data = dummy, weights = 1/w^2)
##
## Weighted Residuals:
##      Min       1Q   Median       3Q      Max
## -2.82926 -0.46358 -0.04519  0.48093  1.67787
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.52122    0.94597  -0.551    0.588
## x            1.00742    0.09646  10.444 4.56e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.045 on 18 degrees of freedom
## Multiple R-squared:  0.8584, Adjusted R-squared:  0.8505
## F-statistic: 109.1 on 1 and 18 DF,  p-value: 4.557e-09
```

Make the columns in the dataframe visible as variables.

```
attach(dummy)
```

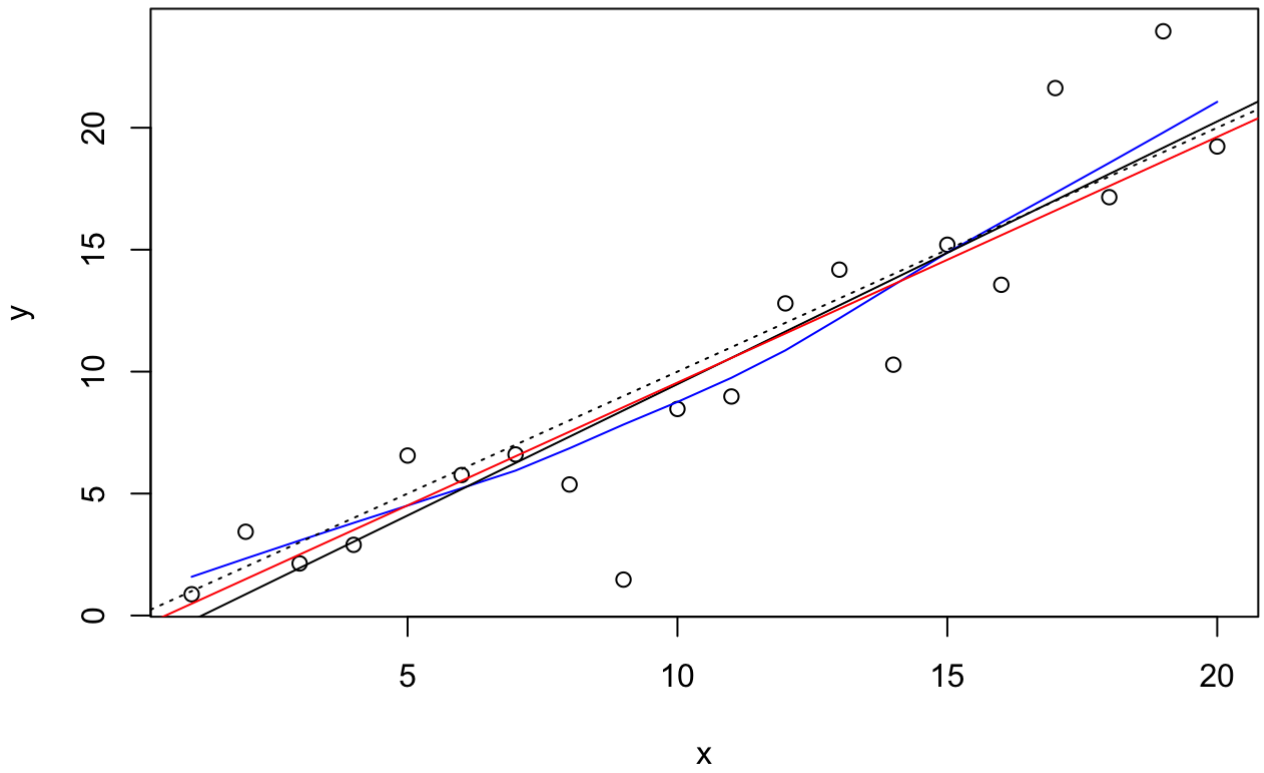
```
## The following objects are masked _by_ .GlobalEnv:
##
##     x, y
```

Make a nonparametric local regression function using the LOWESS smoother (locally weighted regression and smoothing scatterplots, by Cleveland 1979, 1981).

```
lrf <- lowess(x, y)
```

Plot the data points along with the regression lines.

```
plot(x, y)

#### Add in the local regression.
lines(x, lrf$y, col="blue")

#### The "true" regression line: (intercept 0, slope 1).
abline(0, 1, lty=3)

#### Unweighted regression line.
abline(coef(fm), col = "black")

#### Weighted regression line.
abline(coef(fm1), col = "red")
```

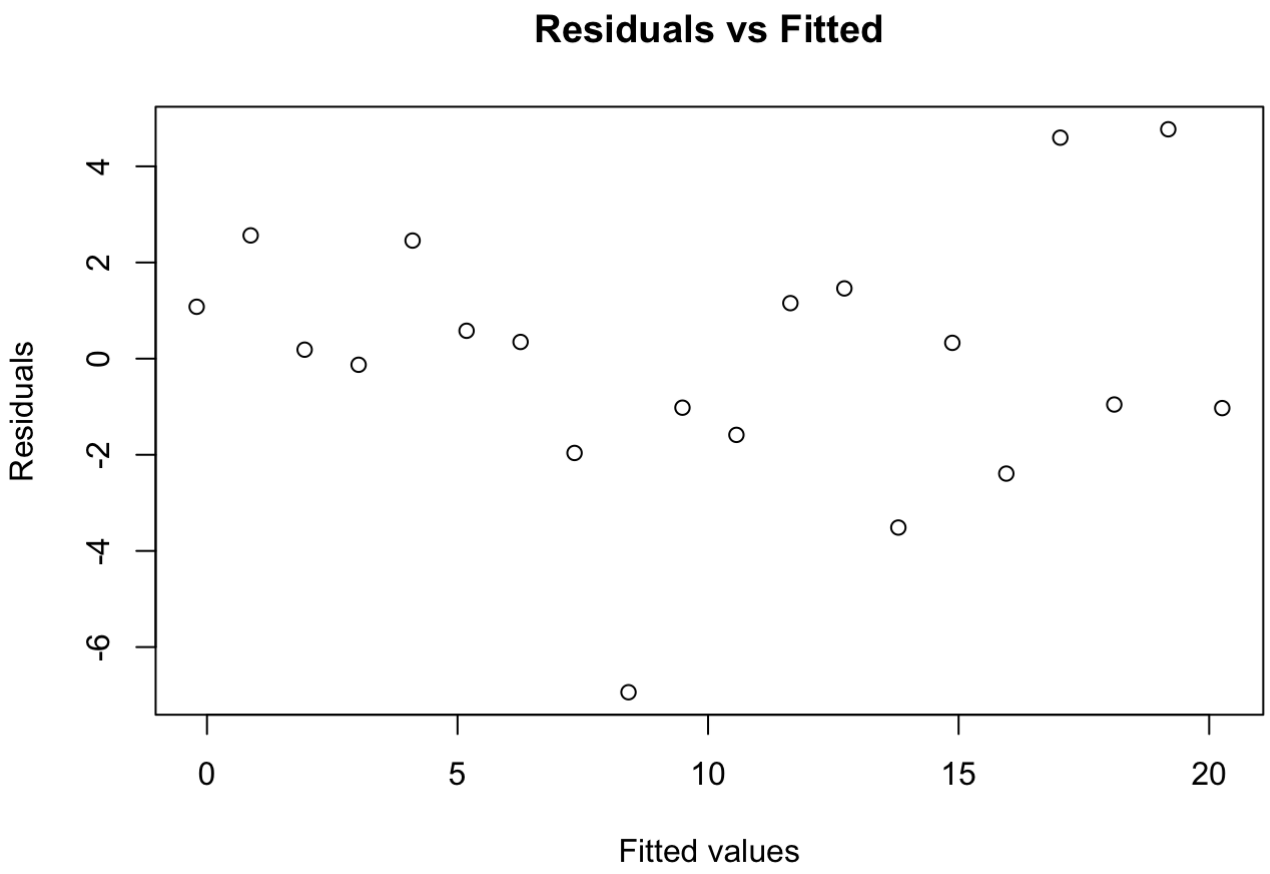


Remove dataframe from the search path.

```
detach()
```

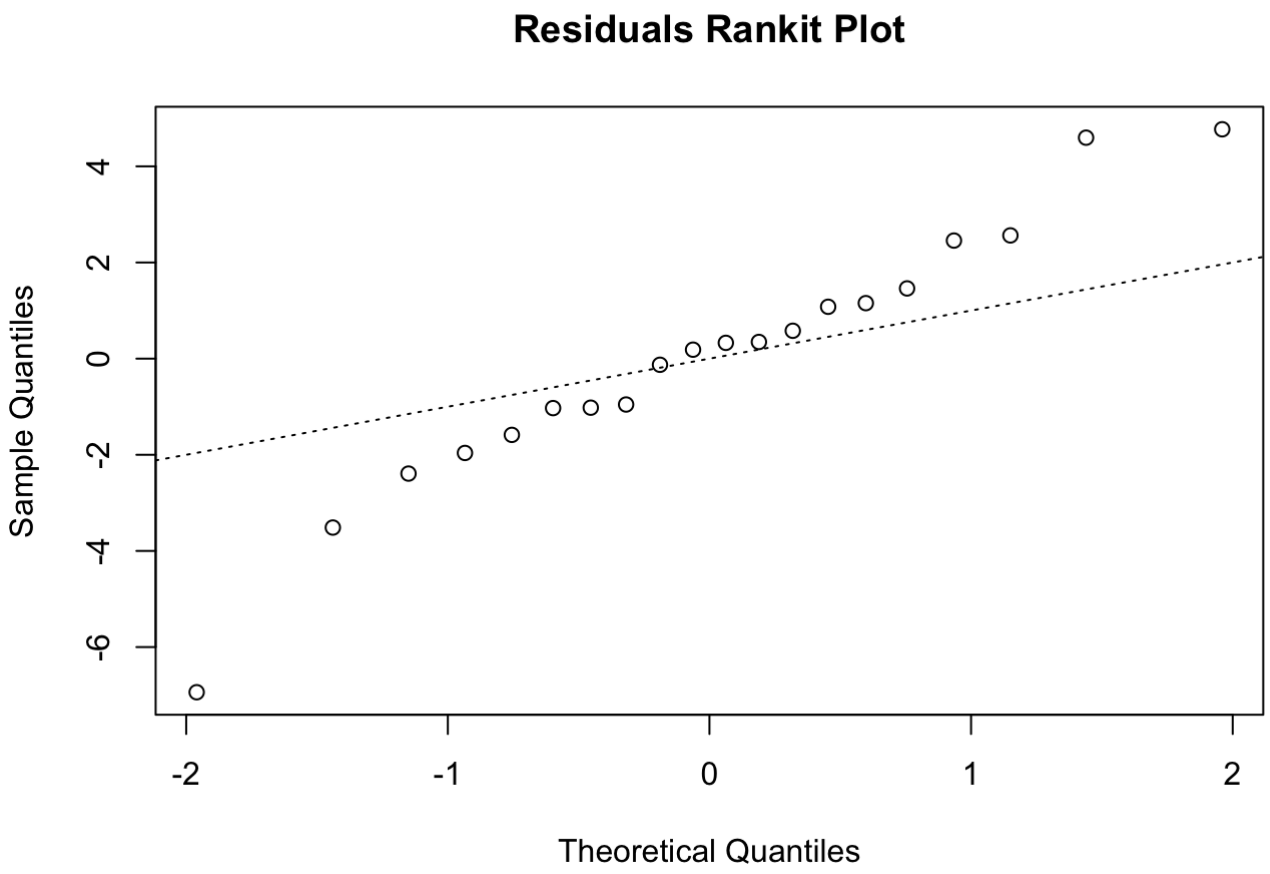
A standard regression diagnostic plot to check for heteroscedasticity (the variability of a variable is unequal across the range of values of a second variable that predicts it).

```
plot(fitted(fm), resid(fm),
     xlab="Fitted values",
     ylab="Residuals",
     main="Residuals vs Fitted")
```



A normal scores plot to check for the deviation of residual distribution from normality.

```
qqnorm(resid(fm), main="Residuals Rankit Plot")
#### If residual distribution is normal...
abline(0, 1, lty=3)
```



Clean up again.

```
rm(fm, fm1, lrf, x, y, w, dummy)
```

2.2 Explore data from Michelson's experiment

In this subsection we look at data from the classical experiment of Michelson to measure the speed of light in 1879. The data consists of five experiments (`Expt`), each consisting of 20 consecutive 'runs' (`Run`). The response is the speed of light measurement (`Speed`), suitably coded (km/sec, with 299000 subtracted). More details can be see with typing `?morley` in the **R** console.

Reference: A. A. Michelson (1882) Experimental determination of the velocity of light made at the United States Naval Academy, Annapolis. *Astronomic Papers* **1** 135–8. U.S. Nautical Almanac Office. (See Table 24.)

This dataset is available in the built-in `morley` data object, but we will read it to illustrate the `read.table` function.

First, get the path to the data file.

```
filepath <- system.file("data", "morley.tab" , package="datasets")
filepath
```

```
## [1] "/Library/Frameworks/R.framework/Resources/library/datasets/data/morley.tab"
```

Look at the file, an optional step.

```
file.show(filepath)
```

Read in the Michelson data as a dataframe, and look at it. There are five experiments (column `Expt`) and each has 20 runs (column `Run`) and `Speed` is the recorded speed of light.

```
mm <- read.table(filepath)
mm
```

```
##     Expt Run Speed
## 001    1   1   850
## 002    1   2   740
## 003    1   3   900
## 004    1   4  1070
## 005    1   5   930
## 006    1   6   850
## 007    1   7   950
## 008    1   8   980
## 009    1   9   980
## 010    1  10   880
## 011    1  11  1000
## 012    1  12   980
## 013    1  13   930
## 014    1  14   650
## 015    1  15   760
## 016    1  16   810
## 017    1  17  1000
## 018    1  18  1000
## 019    1  19   960
## 020    1  20   960
## 021    2   1   960
## 022    2   2   940
## 023    2   3   960
## 024    2   4   940
## 025    2   5   880
## 026    2   6   800
## 027    2   7   850
## 028    2   8   880
## 029    2   9   900
## 030    2  10   840
## 031    2  11   830
## 032    2  12   790
## 033    2  13   810
## 034    2  14   880
## 035    2  15   880
## 036    2  16   830
## 037    2  17   800
## 038    2  18   790
## 039    2  19   760
## 040    2  20   800
## 041    3   1   880
## 042    3   2   880
## 043    3   3   880
## 044    3   4   860
## 045    3   5   720
## 046    3   6   720
## 047    3   7   620
## 048    3   8   860
## 049    3   9   970
## 050    3  10   950
## 051    3  11   880
## 052    3  12   910
## 053    3  13   850
## 054    3  14   870
## 055    3  15   840
## 056    3  16   840
```

```
## 057    3  17   850
## 058    3  18   840
## 059    3  19   840
## 060    3  20   840
## 061    4   1   890
## 062    4   2   810
## 063    4   3   810
## 064    4   4   820
## 065    4   5   800
## 066    4   6   770
## 067    4   7   760
## 068    4   8   740
## 069    4   9   750
## 070    4  10   760
## 071    4  11   910
## 072    4  12   920
## 073    4  13   890
## 074    4  14   860
## 075    4  15   880
## 076    4  16   720
## 077    4  17   840
## 078    4  18   850
## 079    4  19   850
## 080    4  20   780
## 081    5   1   890
## 082    5   2   840
## 083    5   3   780
## 084    5   4   810
## 085    5   5   760
## 086    5   6   810
## 087    5   7   790
## 088    5   8   810
## 089    5   9   820
## 090    5  10   850
## 091    5  11   870
## 092    5  12   870
## 093    5  13   810
## 094    5  14   740
## 095    5  15   810
## 096    5  16   940
## 097    5  17   950
## 098    5  18   800
## 099    5  19   810
## 100    5  20   870
```

Change `Expt` and `Run` into factors.

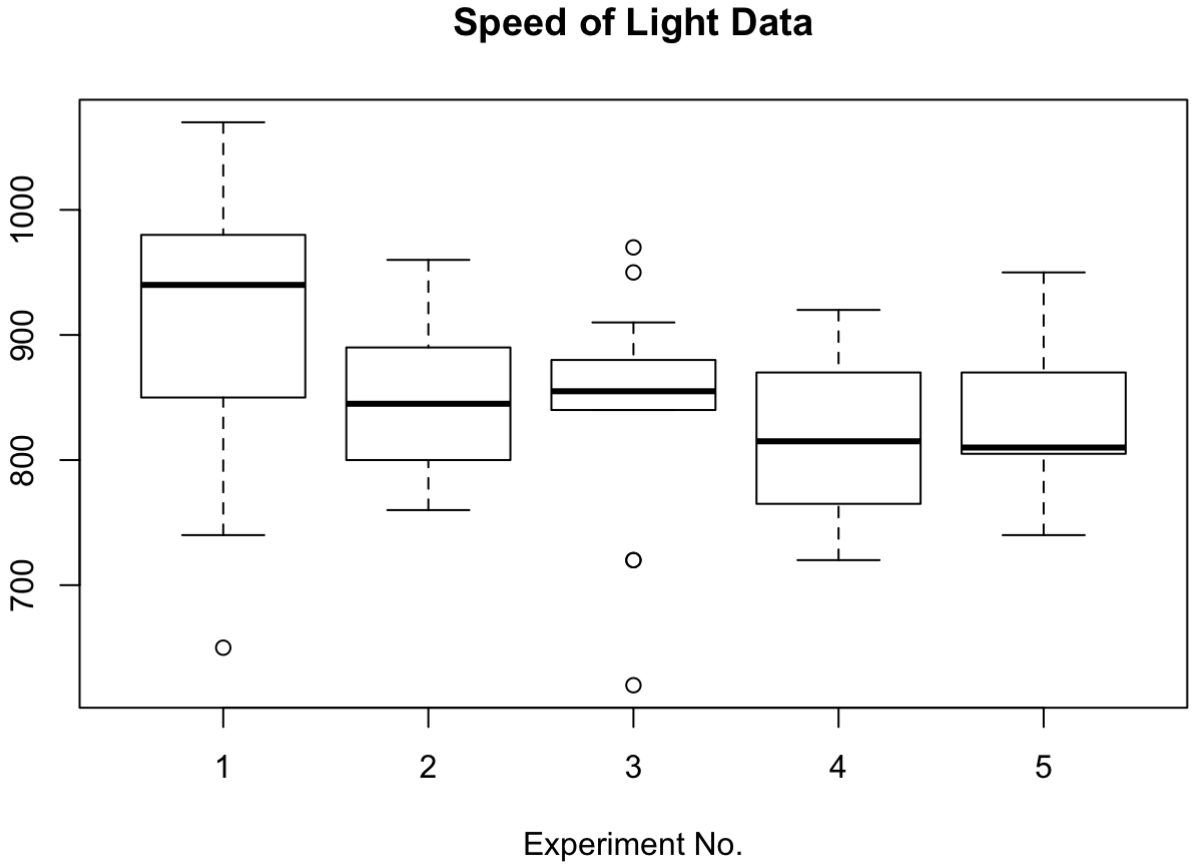
```
mm$Expt <- factor(mm$Expt)
mm$Run <- factor(mm$Run)
```

Make the dataframe visible at position 3 (the default).

```
attach(mm)
```

Compare the five experiments with simple boxplots.

```
plot(Expt, Speed, main="Speed of Light Data", xlab="Experiment No.")
```



Analyze as a randomized block, with ‘runs’ and ‘experiments’ as factors.

```
fm <- aov(Speed ~ Run + Expt, data=mm)
summary(fm)
```

```
##             Df Sum Sq Mean Sq F value  Pr(>F)
## Run         19 113344    5965   1.105 0.36321
## Expt         4  94514   23629   4.378 0.00307 **
## Residuals   76 410166    5397
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Fit the sub-model omitting ‘runs’, and compare using a formal analysis of variance.

```
fm0 <- update(fm, . ~ . - Run)
anova(fm0, fm)
```

```
## Analysis of Variance Table
##
## Model 1: Speed ~ Expt
## Model 2: Speed ~ Run + Expt
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     95 523510
## 2     76 410166 19    113344 1.1053 0.3632
```

Clean up before moving on

```
detach()
rm(fm, fm0)
```

2.3 Some **R** graphics

R has four graphics systems:

- Base graphics: Easiest to learn
- Grid graphics: Powerful set of modules for building other tools
- Lattice graphics: General purpose system based on grid graphics, powerful for detailed visualization of big data projects
- ggplot2: “The grammar of graphics”, fluid and professional visualization, but so far can not scale up for big data projects

2.3.1 More **R** Base graphics

We have seen some features of the base graphics system. Here are contour and image plots.

x is a vector of 50 equally spaced values in the interval [-pi, pi]. y is the same.

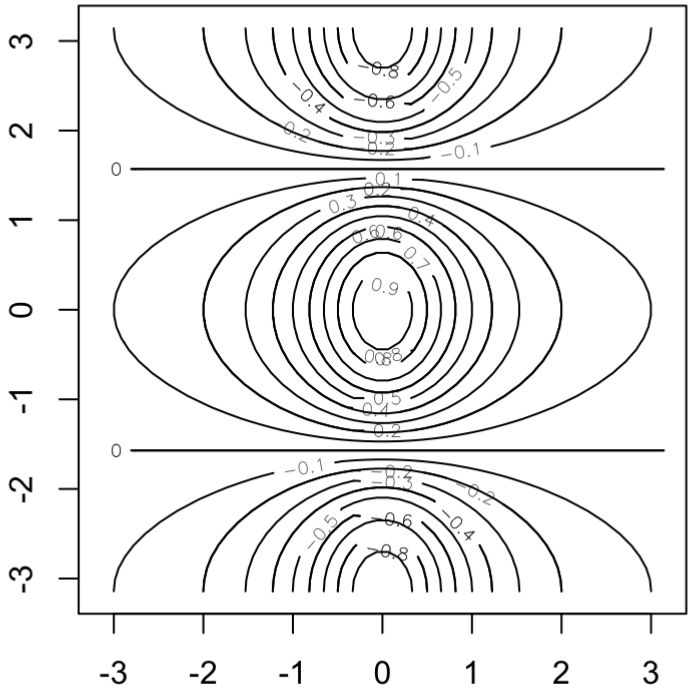
```
x <- seq(-pi, pi, len=50)
y <- x
```

f is a square matrix, with rows and columns indexed by x and y respectively, of values of the function cos(y)/(1 + x^2).

```
f <- outer(x, y, function(x, y) cos(y)/(1 + x^2))
```

Save the plotting parameters and set the plotting region to ‘square’.

```
oldpar <- par(no.readonly = TRUE)
par(pty="s")

#### Make a contour map of f; add in more lines for more detail.
contour(x, y, f)
contour(x, y, f, nlevels=15, add=TRUE)
```

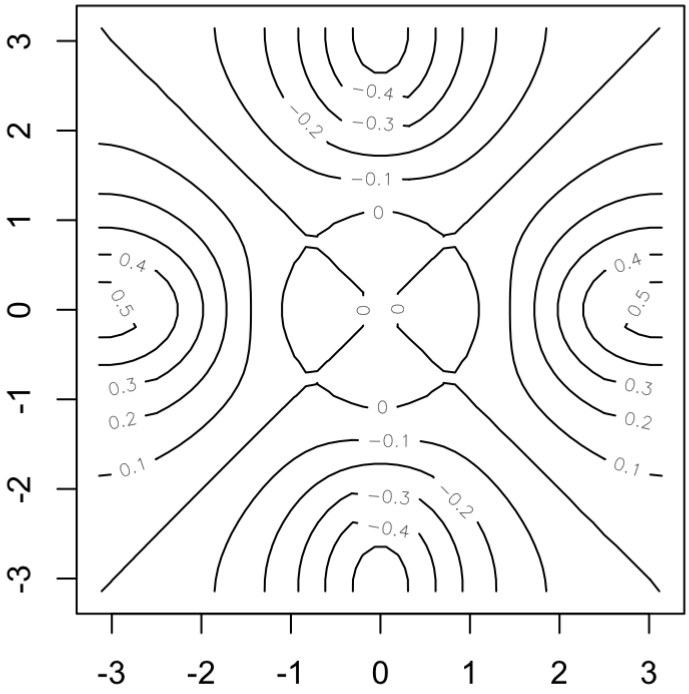


fa is the ‘asymmetric part’ of f. (t() is transpose).

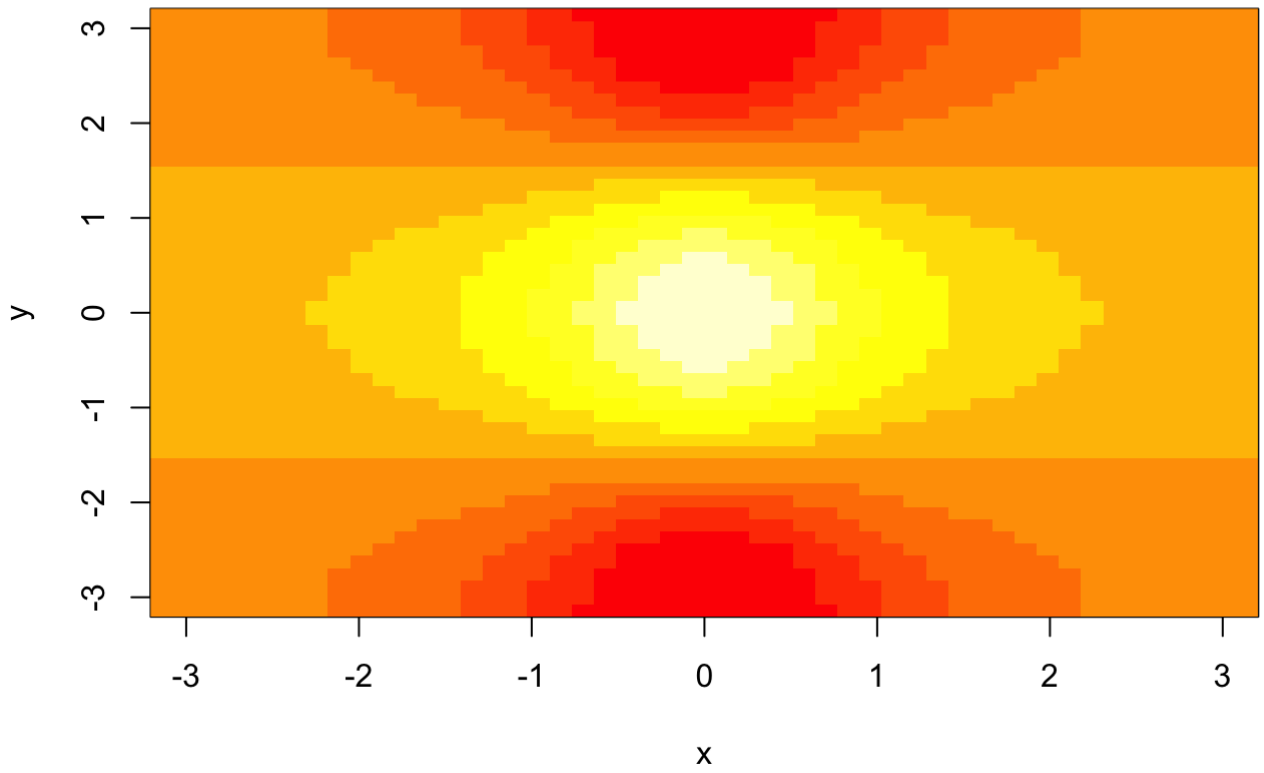
```
fa <- (f-t(f))/2
```

Make a contour plot,…

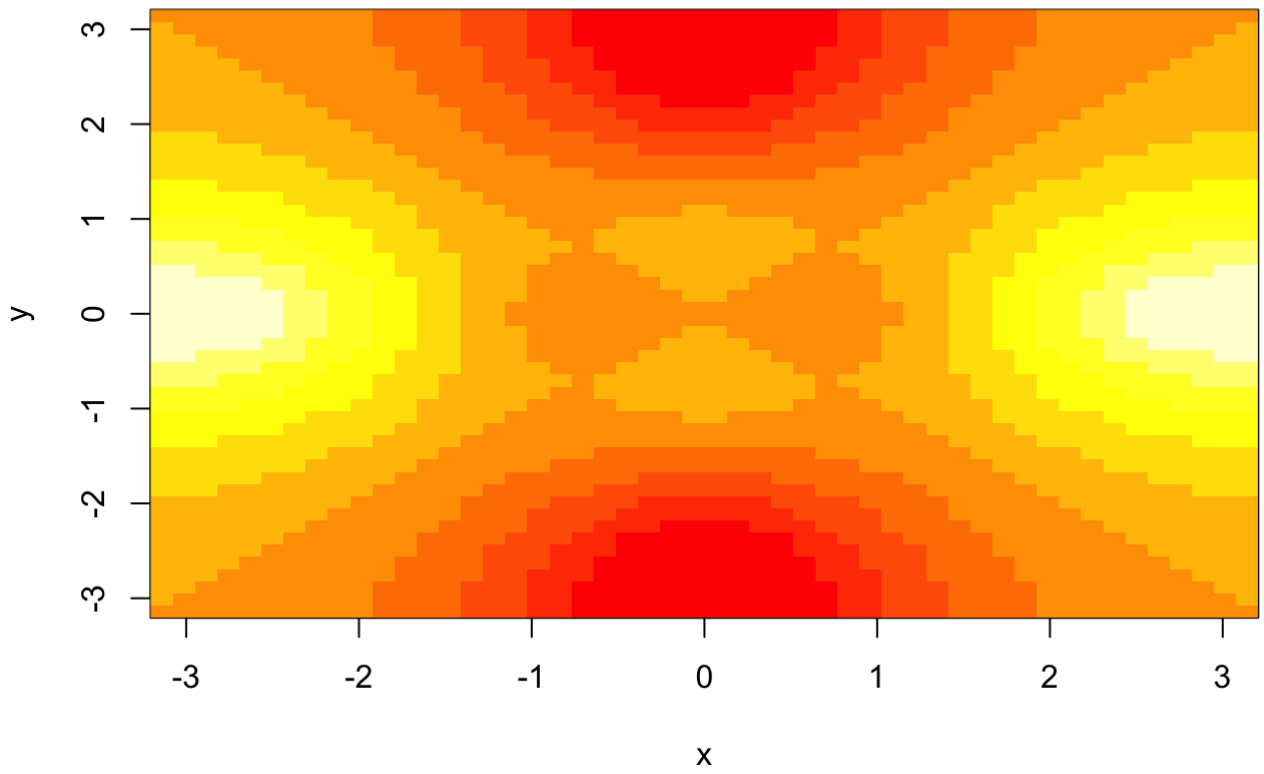
```
par(pty="s")
contour(x, y, fa, nlevels=15)
```




```
#### and restore the old graphics parameters.
par(oldpar)
#### Make some high density image plots,
image(x, y, f)
```




```
image(x, y, fa)
```



Clean up before moving on

```
objects(); rm(x, y, f, fa)
```

```
## [1] "f"        "fa"       "filepath" "mm"       "oldpar"   "x"
## [7] "y"
```

2.3.2 Quickstart visualization with `ggplot2`

We use the a subset of Gapminder (https://www.gapminder.org/data/) data as example. The R `gapminder` data spans from 1952 to 2007 and includes six variables: `country`, `continent`, `year`, `lifeExp` (life expectancy at birth), `pop` (total population), and `gdpPrecap` (per-capita Gross Domestic Product). The per-capital GDP is in the unit of international dollar, which is a hypothetical unit of currency that has the same purchasing power parity that the US dollar had in the United States in 2005.

Please check out the video How not to be ignorant about the world (https://www.gapminder.org/videos/how-not-to-be-ignorant-about-the-world/)

Take a look at the 2007 Gapminder data. Note how “heavy duty” tidyverse is.

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────────────────────
──────── tidyverse 1.2.1 ──
```

```
## ✓ ggplot2 2.2.1     ✓ purrr   0.2.4
## ✓ tibble  1.4.2     ✓ dplyr   0.7.5
## ✓ tidyr   0.8.1     ✓ stringr 1.4.0
## ✓ readr   1.1.1     ✓ forcats 0.3.0
```

```
## Warning: package 'stringr' was built under R version 3.4.4
```

```
## ── Conflicts ────────────────────────────────────────────────────────────────
─── tidyverse_conflicts() ──
## ✕ dplyr::filter() masks stats::filter()
## ✕ dplyr::lag()    masks stats::lag()
```

```
library(gapminder)

gapminder_2007 <- gapminder %>% filter(year == 2007)
```

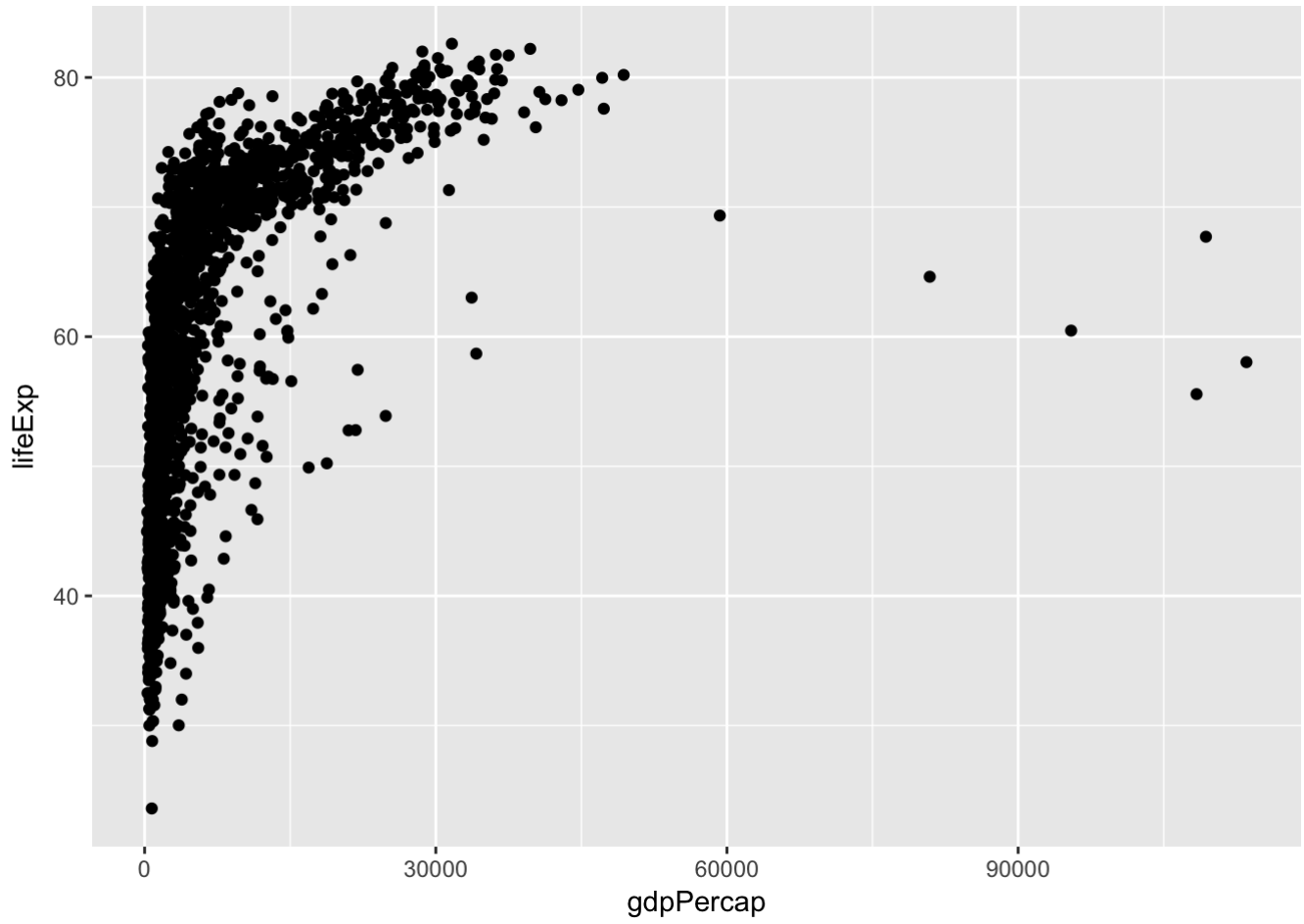
```
## Warning: package 'bindrcpp' was built under R version 3.4.4
```

```
gapminder_2007
```

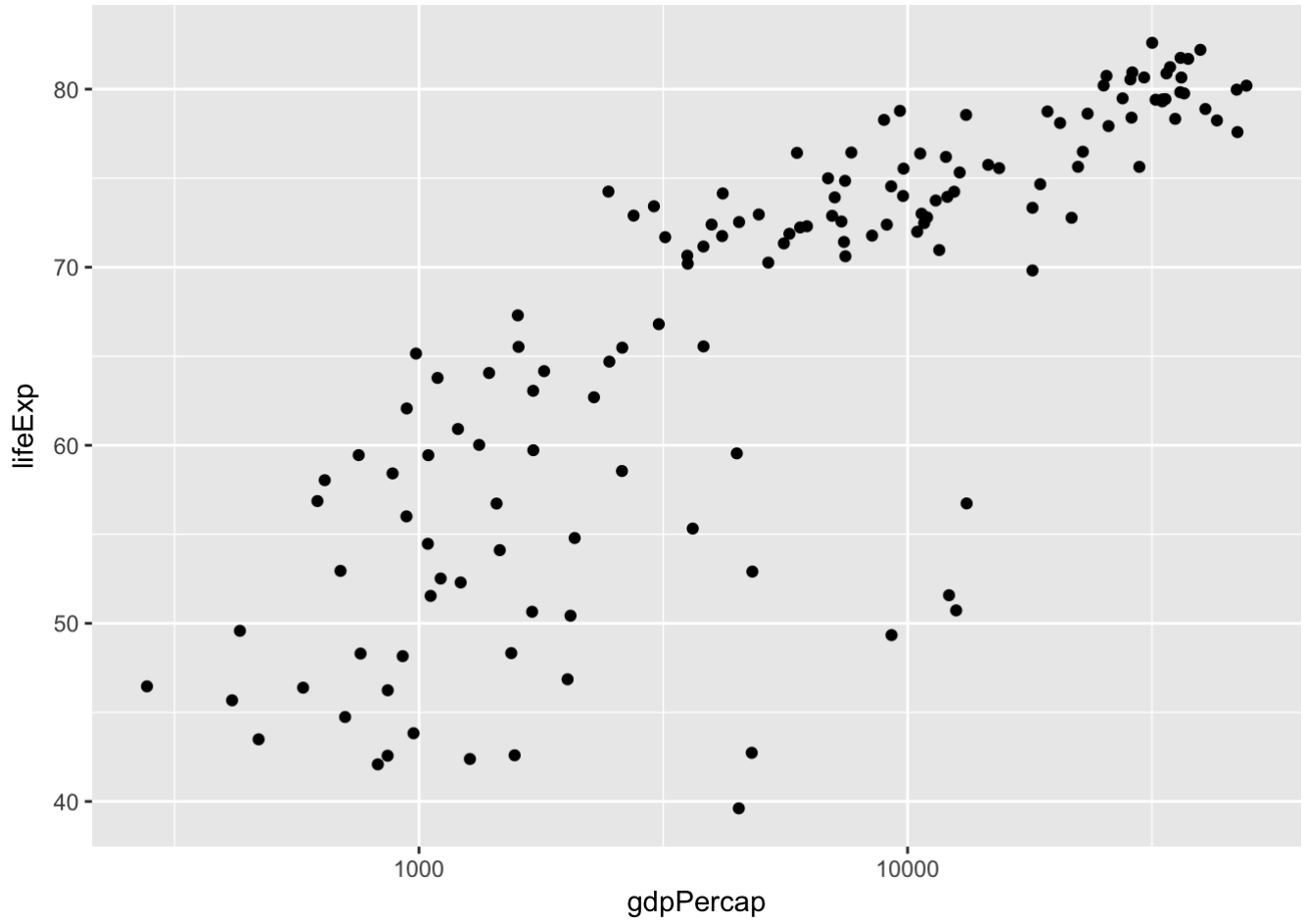
```
## # A tibble: 142 x 6
##    country     continent  year lifeExp       pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>     <int>     <dbl>
##  1 Afghanistan Asia       2007    43.8  31889923      975.
##  2 Albania     Europe     2007    76.4   3600523     5937.
##  3 Algeria     Africa     2007    72.3  33333216     6223.
##  4 Angola      Africa     2007    42.7  12420476     4797.
##  5 Argentina   Americas   2007    75.3  40301927    12779.
##  6 Australia   Oceania    2007    81.2  20434176    34435.
##  7 Austria     Europe     2007    79.8   8199783    36126.
##  8 Bahrain     Asia       2007    75.6    708573    29796.
##  9 Bangladesh  Asia       2007    64.1 150448339     1391.
## 10 Belgium     Europe     2007    79.4  10392226    33693.
## # ... with 132 more rows
```

Visualize with `ggplot2` using scatter plots in linear and log scale. Note the progressing complexity. The attachment of `ggplot2` is redundant with that of `tidyverse` previously. It demonstrates that packages within `tidyverse` can be loaded individually.

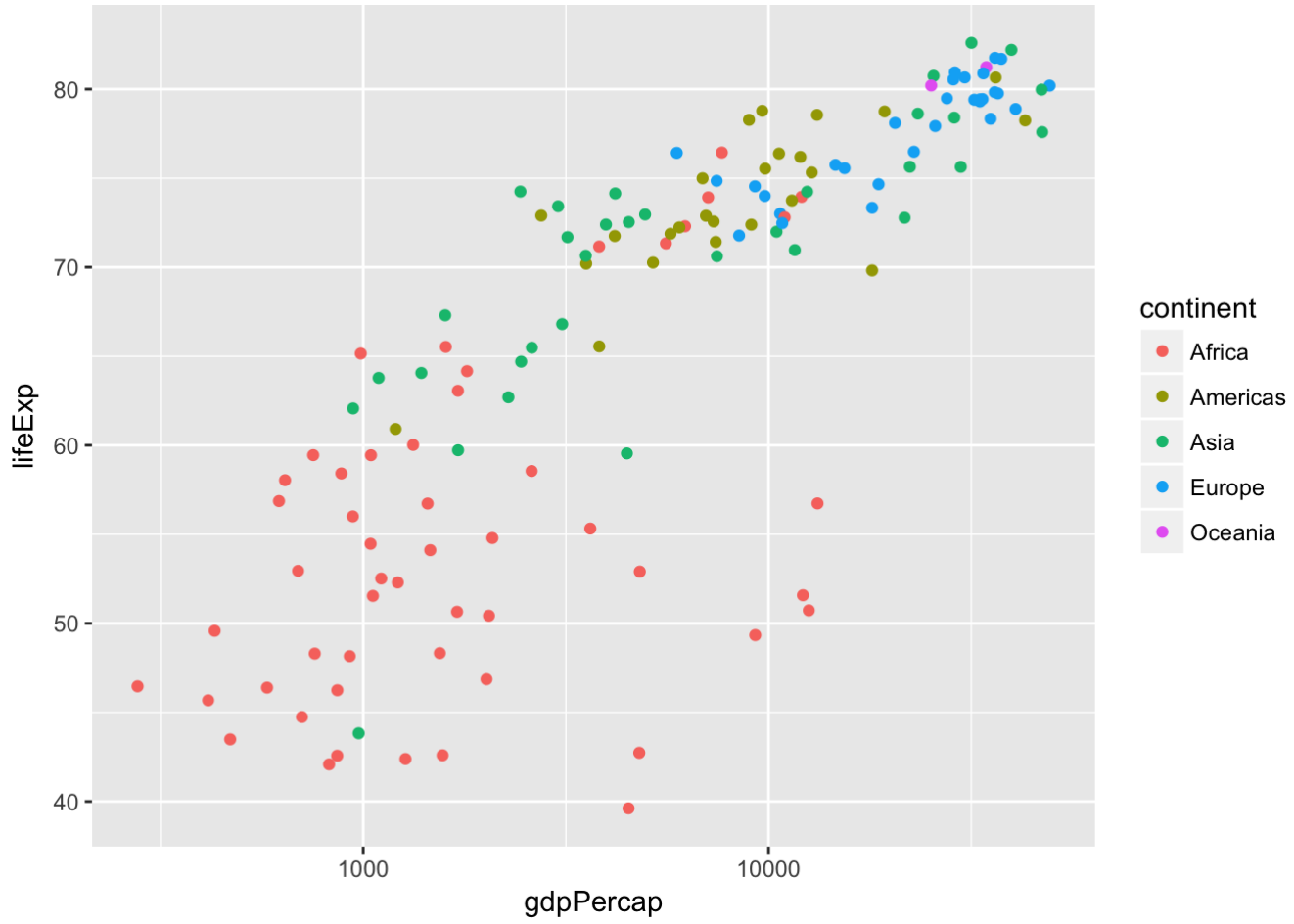
```
library(ggplot2)

ggplot(gapminder, aes(x = gdpPercap, y = lifeExp)) +
geom_point()
```



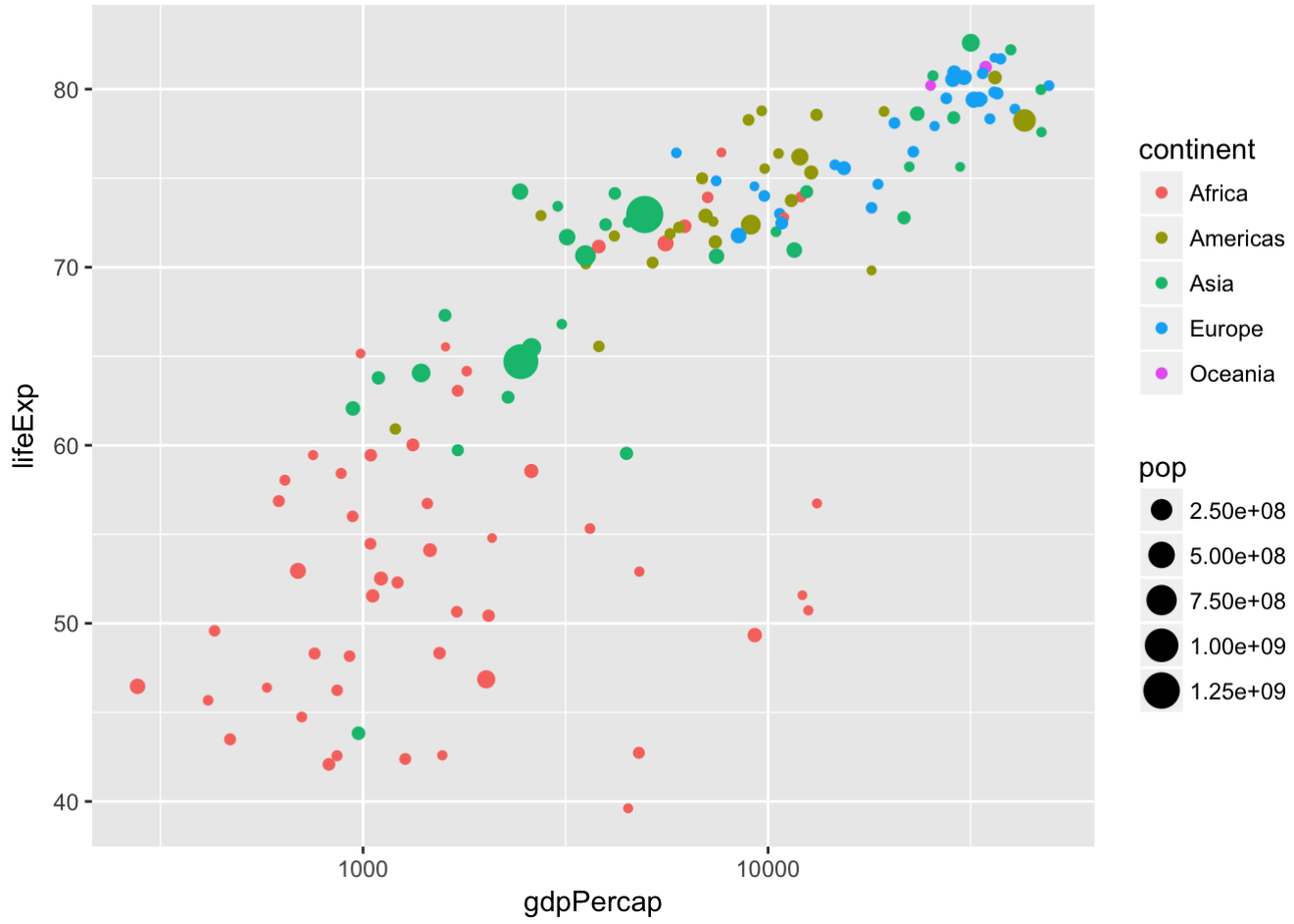

```
ggplot(gapminder_2007, aes(x = gdpPercap, y = lifeExp)) +
geom_point() +
scale_x_log10()
```




```
ggplot(gapminder_2007, aes(x = gdpPercap, y = lifeExp, color = continent)) +
geom_point() +
scale_x_log10()
```

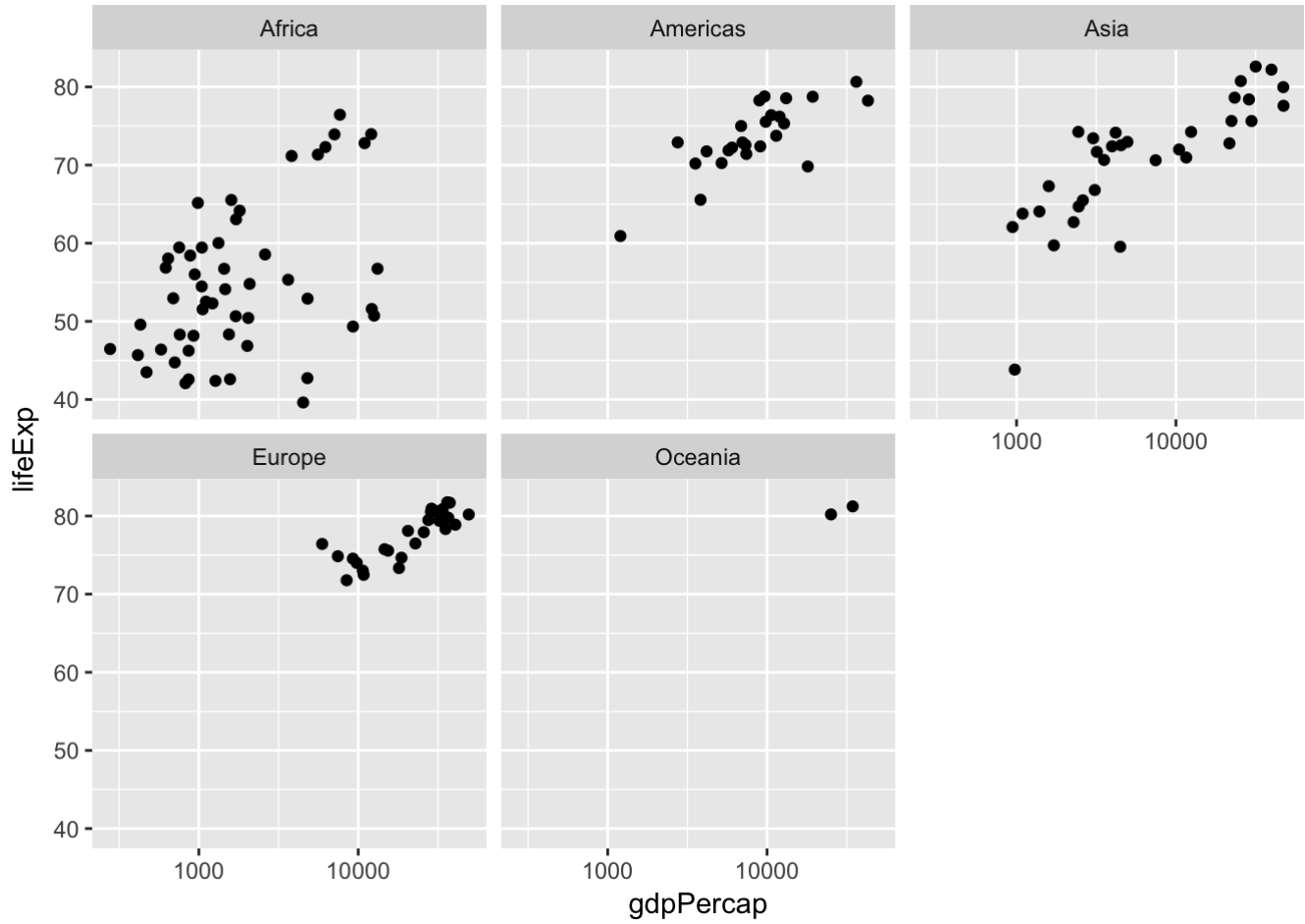



```
ggplot(gapminder_2007, aes(x = gdpPercap, y = lifeExp, color = continent,
size = pop)) +
geom_point() +
scale_x_log10()
```



Faceting (we will see more power in Lattice graphics soon)

```
ggplot(gapminder_2007, aes(x = gdpPercap, y = lifeExp)) +
geom_point() +
scale_x_log10() +
facet_wrap(~ continent)
```



Some clean up.

```
rm(list=ls())
objects()
```

```
## character(0)
```

```
detach("package:ggplot2")
detach("package:purrr")
detach("package:tibble")
detach("package:dplyr")
detach("package:tidyr")
detach("package:stringr")
detach("package:readr")
detach("package:forcats")
detach("package:tidyverse")
detach("package:gapminder")
```

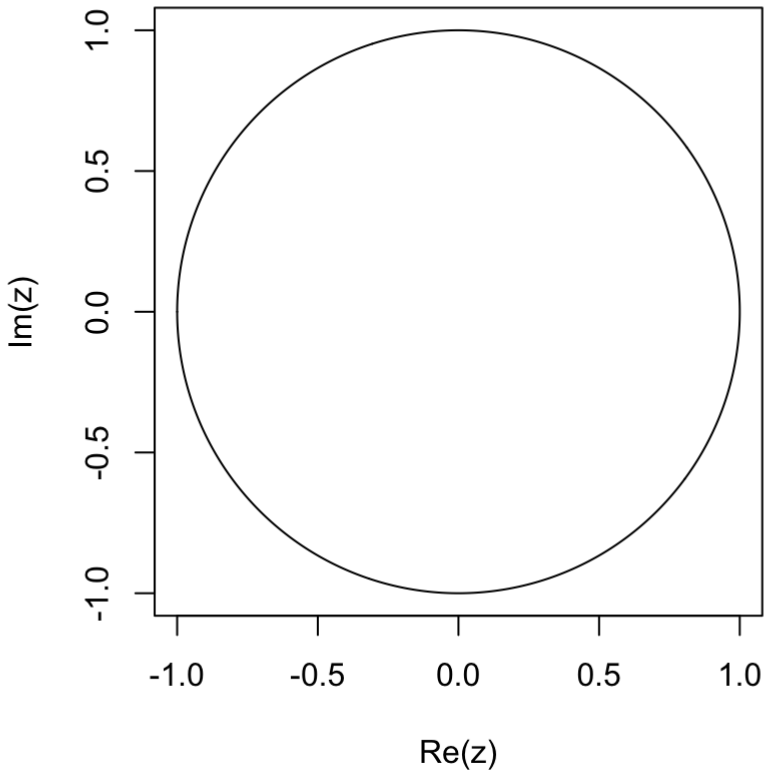
2.4 Complex arithmetic

1i is used for the complex number i.

```
th <- seq(-pi, pi, len=500)
z <- exp(1i*th)
```

Plotting complex arguments means plot imaginary versus real parts. This should be a circle.

```
par(pty="s")
plot(z, type="l")
```



Suppose we want to sample points within the unit circle…

One method would be to take complex numbers with standard normal real and imaginary parts.

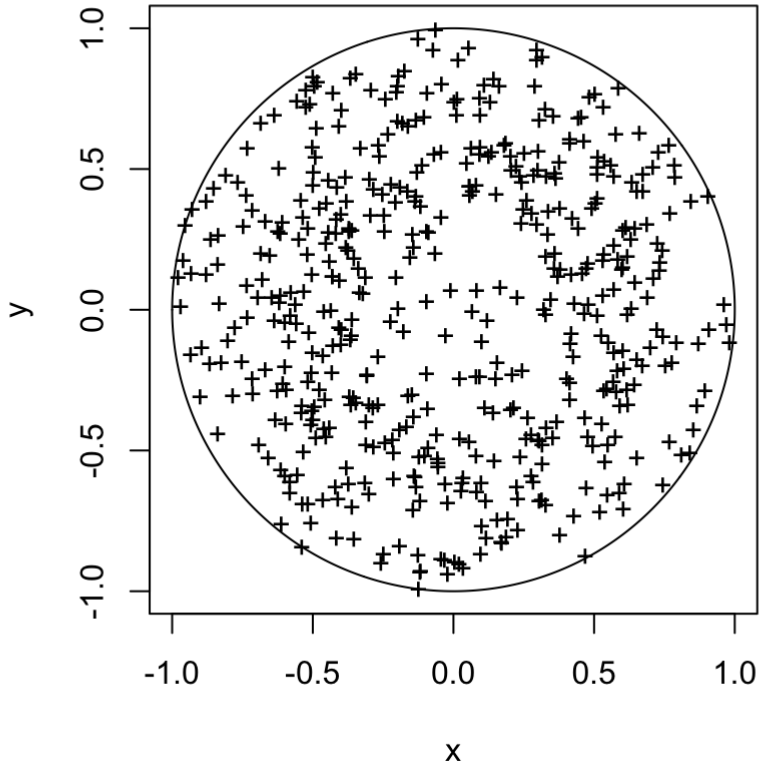
```
w <- rnorm(500) + rnorm(500)*1i
```

…and to map any outside the circle onto their reciprocal.

```
w <- ifelse(Mod(w) > 1, 1/w, w)
```

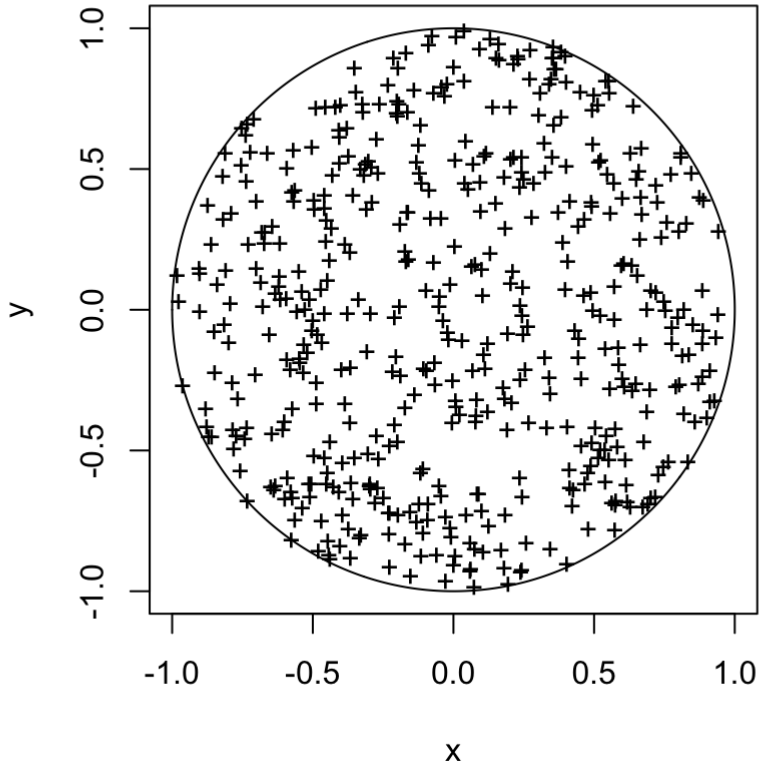
All points are inside the unit circle, but the distribution is not uniform.

```
par(pty="s")
plot(w, xlim=c(-1,1), ylim=c(-1,1), pch="+",xlab="x", ylab="y")
lines(z)
```



The second method uses the uniform distribution. The points should now look more evenly spaced over the disc.

```
w <- sqrt(runif(500))*exp(2*pi*runif(500)*1i)
par(pty="s")
plot(w, xlim=c(-1,1), ylim=c(-1,1), pch="+", xlab="x", ylab="y")
lines(z)
```



Clean up.

```
rm(th, w, z)
```

2.5 Distinguish rocks from mines from sonar data

Following is an example of using a classification model to distinguish rocks from mines in sonar data. The data have been taken from the UCI Repository of Machine Learning Databases (http://www.ics.uci.edu/~mlearn/MLRepository.html).

Reference: Gorman, R. P., and Sejnowski, T. J. (1988). “Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets”, *Neural Networks*, **Vol. 1**, pp. 75-89.

This is the data set in the above study of the classification of sonar signals using a neural network. The task is to train a network to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock.

Each pattern is a set of 60 numbers in the range 0.0 to 1.0. Each number represents the energy within a particular frequency band, integrated over a certain period of time. The label associated with each record contains the letter “R” if the object is a rock and “M” if it is a mine (metal cylinder).

The `Sonar` data is in the `mlbench` package.

```
library(mlbench)
data(Sonar)

# Take a quick look of the data
str(Sonar)
```

```
## 'data.frame':    208 obs. of  61 variables:
##  $ V1   : num  0.02 0.0453 0.0262 0.01 0.0762 0.0286 0.0317 0.0519 0.0223 0.0164 ...
##  $ V2   : num  0.0371 0.0523 0.0582 0.0171 0.0666 0.0453 0.0956 0.0548 0.0375 0.0173 ...
##  $ V3   : num  0.0428 0.0843 0.1099 0.0623 0.0481 ...
##  $ V4   : num  0.0207 0.0689 0.1083 0.0205 0.0394 ...
##  $ V5   : num  0.0954 0.1183 0.0974 0.0205 0.059 ...
##  $ V6   : num  0.0986 0.2583 0.228 0.0368 0.0649 ...
##  $ V7   : num  0.154 0.216 0.243 0.11 0.121 ...
##  $ V8   : num  0.16 0.348 0.377 0.128 0.247 ...
##  $ V9   : num  0.3109 0.3337 0.5598 0.0598 0.3564 ...
##  $ V10  : num  0.211 0.287 0.619 0.126 0.446 ...
##  $ V11  : num  0.1609 0.4918 0.6333 0.0881 0.4152 ...
##  $ V12  : num  0.158 0.655 0.706 0.199 0.395 ...
##  $ V13  : num  0.2238 0.6919 0.5544 0.0184 0.4256 ...
##  $ V14  : num  0.0645 0.7797 0.532 0.2261 0.4135 ...
##  $ V15  : num  0.066 0.746 0.648 0.173 0.453 ...
##  $ V16  : num  0.227 0.944 0.693 0.213 0.533 ...
##  $ V17  : num  0.31 1 0.6759 0.0693 0.7306 ...
##  $ V18  : num  0.3 0.887 0.755 0.228 0.619 ...
##  $ V19  : num  0.508 0.802 0.893 0.406 0.203 ...
##  $ V20  : num  0.48 0.782 0.862 0.397 0.464 ...
##  $ V21  : num  0.578 0.521 0.797 0.274 0.415 ...
##  $ V22  : num  0.507 0.405 0.674 0.369 0.429 ...
##  $ V23  : num  0.433 0.396 0.429 0.556 0.573 ...
##  $ V24  : num  0.555 0.391 0.365 0.485 0.54 ...
##  $ V25  : num  0.671 0.325 0.533 0.314 0.316 ...
##  $ V26  : num  0.641 0.32 0.241 0.533 0.229 ...
##  $ V27  : num  0.71 0.327 0.507 0.526 0.7 ...
##  $ V28  : num  0.808 0.277 0.853 0.252 1 ...
##  $ V29  : num  0.679 0.442 0.604 0.209 0.726 ...
##  $ V30  : num  0.386 0.203 0.851 0.356 0.472 ...
##  $ V31  : num  0.131 0.379 0.851 0.626 0.51 ...
##  $ V32  : num  0.26 0.295 0.504 0.734 0.546 ...
##  $ V33  : num  0.512 0.198 0.186 0.612 0.288 ...
##  $ V34  : num  0.7547 0.2341 0.2709 0.3497 0.0981 ...
##  $ V35  : num  0.854 0.131 0.423 0.395 0.195 ...
##  $ V36  : num  0.851 0.418 0.304 0.301 0.418 ...
##  $ V37  : num  0.669 0.384 0.612 0.541 0.46 ...
##  $ V38  : num  0.61 0.106 0.676 0.881 0.322 ...
##  $ V39  : num  0.494 0.184 0.537 0.986 0.283 ...
##  $ V40  : num  0.274 0.197 0.472 0.917 0.243 ...
##  $ V41  : num  0.051 0.167 0.465 0.612 0.198 ...
##  $ V42  : num  0.2834 0.0583 0.2587 0.5006 0.2444 ...
##  $ V43  : num  0.282 0.14 0.213 0.321 0.185 ...
##  $ V44  : num  0.4256 0.1628 0.2222 0.3202 0.0841 ...
##  $ V45  : num  0.2641 0.0621 0.2111 0.4295 0.0692 ...
##  $ V46  : num  0.1386 0.0203 0.0176 0.3654 0.0528 ...
##  $ V47  : num  0.1051 0.053 0.1348 0.2655 0.0357 ...
##  $ V48  : num  0.1343 0.0742 0.0744 0.1576 0.0085 ...
##  $ V49  : num  0.0383 0.0409 0.013 0.0681 0.023 0.0264 0.0507 0.0285 0.0777 0.0092 ...
##  $ V50  : num  0.0324 0.0061 0.0106 0.0294 0.0046 0.0081 0.0159 0.0178 0.0439 0.0198 ...
##  $ V51  : num  0.0232 0.0125 0.0033 0.0241 0.0156 0.0104 0.0195 0.0052 0.0061 0.0118 ...
##  $ V52  : num  0.0027 0.0084 0.0232 0.0121 0.0031 0.0045 0.0201 0.0081 0.0145 0.009 ...
##  $ V53  : num  0.0065 0.0089 0.0166 0.0036 0.0054 0.0014 0.0248 0.012 0.0128 0.0223 ...
##  $ V54  : num  0.0159 0.0048 0.0095 0.015 0.0105 0.0038 0.0131 0.0045 0.0145 0.0179 ...
##  $ V55  : num  0.0072 0.0094 0.018 0.0085 0.011 0.0013 0.007 0.0121 0.0058 0.0084 ...
##  $ V56  : num  0.0167 0.0191 0.0244 0.0073 0.0015 0.0089 0.0138 0.0097 0.0049 0.0068 ...
```

```
##  $ V57  : num  0.018 0.014 0.0316 0.005 0.0072 0.0057 0.0092 0.0085 0.0065 0.0032 ...
##  $ V58  : num  0.0084 0.0049 0.0164 0.0044 0.0048 0.0027 0.0143 0.0047 0.0093 0.0035 ...
##  $ V59  : num  0.009 0.0052 0.0095 0.004 0.0107 0.0051 0.0036 0.0048 0.0059 0.0056 ...
##  $ V60  : num  0.0032 0.0044 0.0078 0.0117 0.0094 0.0062 0.0103 0.0053 0.0022 0.004 ...
##  $ Class: Factor w/ 2 levels "M","R": 2 2 2 2 2 2 2 2 2 2 ...
```

We can shorten the data display to first five rows and columns, plus the last column.

```
Sonar[1:5, c(1:5, 61)]
```

```
##       V1     V2     V3     V4     V5 Class
## 1 0.0200 0.0371 0.0428 0.0207 0.0954     R
## 2 0.0453 0.0523 0.0843 0.0689 0.1183     R
## 3 0.0262 0.0582 0.1099 0.1083 0.0974     R
## 4 0.0100 0.0171 0.0623 0.0205 0.0205     R
## 5 0.0762 0.0666 0.0481 0.0394 0.0590     R
```

Split the data randomly into about 60% training set and 40% test set. Note how the function `nrow` is used.

```
# Suffle the rows randomly
newrows <- sample(nrow(Sonar))
Sonar <- Sonar[newrows, ]

# Take the first ~60% of rows
split <- round(nrow(Sonar) * .60)
trainS <- Sonar[1:split, ]
testS <- Sonar[(split + 1):nrow(Sonar), ]

# Confirm test set size, should be about 40%
nrow(testS) / nrow(Sonar)
```

```
## [1] 0.3990385
```

Fit a model to the training set using `glm`, then try to predict the test set.

```
model <- glm(Class ~ ., family = binomial(link = "logit"), trainS)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
p <- predict(model, testS, type = "response")
summary(p)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   0.000   0.371   1.000   1.000
```

Convert probabilities into classes and look at their frequencies in a table, then further compare predicted vs. actual classes in another table. The second table is a Confusion Matrix (https://en.wikipedia.org/wiki/Confusion_matrix).

```
p_class <- ifelse(p > .50, "M", "R")
table(p_class)
```

```
## p_class
##  M  R
## 31 52
```

```
table(p_class, testS[["Class"]])
```

```
##
## p_class  M  R
##       M  9 22
##       R 36 16
```

```
# accuracy...
sum(factor(p_class)==factor(testS[["Class"]]))/length(p)
```

```
## [1] 0.3012048
```

3. Citing R

To cite **R** in a paper:

```
citation()
```

```
##
## To cite R in publications use:
##
##   R Core Team (2017). R: A language and environment for
##   statistical computing. R Foundation for Statistical Computing,
##   Vienna, Austria. URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2017},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please
## cite it when using it for data analysis. See also
## 'citation("pkgname")' for citing R packages.
```

To cite a user-supplied package, lattice for example:

```
citation("lattice")
```

```
##
## To cite the lattice package in publications use:
##
##   Sarkar, Deepayan (2008) Lattice: Multivariate Data Visualization
##   with R. Springer, New York. ISBN 978-0-387-75968-5
##
## A BibTeX entry for LaTeX users is
##
##   @Book{,
##     title = {Lattice: Multivariate Data Visualization with R},
##     author = {Deepayan Sarkar},
##     publisher = {Springer},
##     address = {New York},
##     year = {2008},
##     note = {ISBN 978-0-387-75968-5},
##     url = {http://lmdvr.r-forge.r-project.org},
##   }
```