Datenanalyse mit R, Teil 1

by heise Developer • April 19, 2013 • original (http://www.heise.de/developer/artikel/Datenanalyse-mit-R-Teil-1-1845278.html?artikelseite=2)

Eine zentrale Datenstruktur in R ist der Vektor, also eine Auflistung gleichartiger Objekte, ähnlich den Vektoren in Java oder C++. Hier unterscheidet sich R von anderen statistischen Sprachen wie MATLAB, die Vektoren in ihrer mathematischen Bedeutung als Punkt in einem Vektorraum auffassen. Anders ist so vor allem die Interpretation der Operationen auf Vektoren: Der Befehl *length(x)* gibt die Anzahl der Elemente in *x* zurück – im obigen Beispiel also 1000. Ein Mathematiker würde hier die Länge eines Vektors in einem euklidischen Raum erwarten.

Operationen auf einem Vektor führt R grundsätzlich komponentenweise aus:

```
> a <- c(1,2,4,4,12,2,2,8,6,3)
> b <- c(3,5,2,4,2,24,3,2,2,2)
> a*b
[1]  3 10  8 16 24 48  6 16 12  6
```

Alle Elemente besitzen den gleichen Typ. Ein einzelnes oder mehrere Elemente werden über den [-Operator angesprochen (Details zum [-Operator erhält man mit dem Befehl *help("[")*). Hier ein paar Beispiele zum Ausprobieren.

Das zweite Element des Vektors:

```
> a[2]
[1] 2
```

Alle Elemente vom zweiten bis zum fünften:

```
> a[2:5]
[1] 1  2  4  4 12
```

Alle Elemente, die größer als 5 sind:

```
> a[a>5]
[1]  12  8  6
```

Die Operation *a*b* gibt nicht, wie man erwarten könnte, das Skalarprodukt der beiden Vektoren zurück, sondern multipliziert nur jeweils die einzelnen Komponenten. Soll ein Programm das Skalarprodukt berechnen, ist der Operator *%*%* zu verwenden.

Eine spezielle Form eines Vektors ist die sogenannte Sequenz. Sie beschreibt eine festgelegte Folge von Zahlen und wird mit der Funktion *seq* erstellt. Alle Zahlen zwischen –1 und 1 mit einer Schrittweite von 0,2 lassen sich beispielsweise durch die folgende Eingabe erzeugen:

```
> seq(-1,1,0.2)
 [1] -1.0 -0.8 -0.6 -0.4 -0.2  0.0  0.2  0.4  0.6  0.8  1.0
```

Sequenzen mit der Schrittweite 1 werden meist in einer etwas kompakteren Schreibweise abgekürzt:

```
> 1:5           #(gleichbedeutend mit seq(1,5,1))
[1] 1 2 3 4 5
```

Um statistische Analysen durchführen zu können, ist es nötig, Daten auch in tabellarischer Form darstellen zu können. Das wird in R mit einem sogenannten Data Frame erreicht, einer Auflistung von Tabellenspalten, die wiederum aus jeweils einem Vektor bestehen. Mit der Funktion *data.frame()* lässt sich ein solcher zusammensetzen:

```
> n <- c("Merkur","Venus","Erde","Mars")
> r <- c(2439, 6052, 6378, 3397)
> m <- c(0,0,1,2)
> planeten <- data.frame(name=n, aequator.radius=r, anzahl.monde=m)
> planeten
    name aequator.radius anzahl.monde
1 Merkur            2439            0
2  Venus            6052            0
3   Erde            6378            1
4   Mars            3397            2
```

Wie man in diesem Beispiel erkennt, können die Elementtypen der einzelnen Vektoren (Spalten) unterschiedlich sein. Hier gehören sie den Typen *character* und *numeric* an.

Ein weiterer, spezieller Datentyp ist der Faktor: Dabei handelt es sich um Aufzählungen von Variablen, die keinen zahlenmäßigen Wert besitzen, wie Farbe, Beruf oder Geschlecht. Sie lassen sich also nicht der Größe nach ordnen (man kann nicht sagen, ob die Farbe Rot kleiner ist als Grün). Faktoren ordnungsgemäß zu deklarieren ist wichtig, da sie in den statistischen Berechnungen und bei grafischen Darstellungen anders zu behandeln sind. Es gibt auch geordnete Faktoren, bei denen eine natürliche Reihenfolge der möglichen Werte festgelegt ist. Der Unterschied zu numerischen Größen ist, dass sich auf diesen Faktoren keine mathematischen Operationen ausführen lassen. Ein Beispiel sind Schuhgrößen: Sie lassen sich in eine sinnvolle Ordnung bringen, Addieren oder Multiplizieren ergibt hingegen kein brauchbares Ergebnis.

Das folgende Beispiel enthält eine bessere Version der zuvor erstellten Planetentabelle, die auch alle Funktionen in R richtig interpretieren können.

```
> n <- factor(c("Merkur","Venus","Erde","Mars"))
> r <- c(2439, 6052, 6378, 3397)
> m <- c(0,0,1,2)
> s <- ordered(c("klein", "groß", "groß", "mittel"),
>              levels=c("klein", "mittel", "groß"))
```

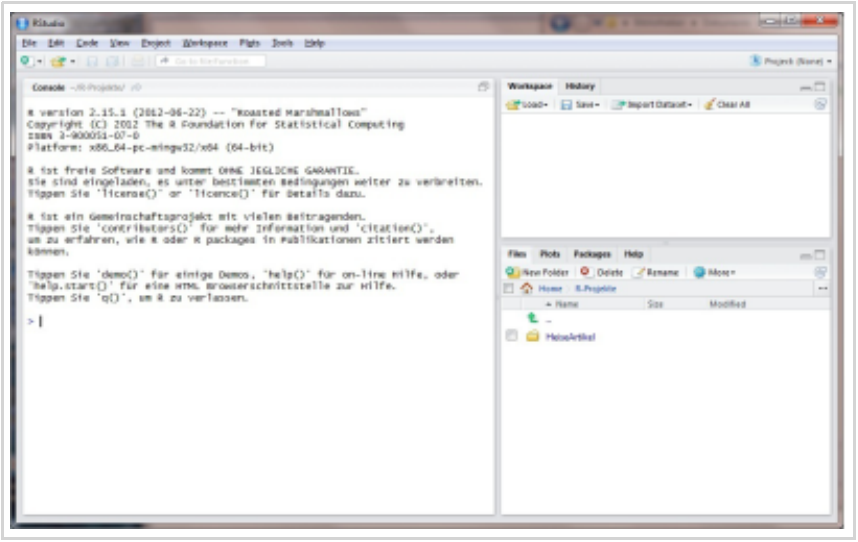
```
> planeten <- data.frame(name=n, aequator.radius=r, anzahl.monde=m,
>                        groessen.kategorie=s)
> planeten
    name aequator.radius anzahl.monde groessen.kategorie
1 Merkur            2439            0              klein
2  Venus            6052            0               groß
3   Erde            6378            1               groß
4   Mars            3397            2             mittel
```

Die allgemeinste und komplexeste Datenstruktur ist die Liste. Sie ist eine geordnete Aufzählung von Datenobjekten. Der wichtige Unterschied zwischen einer Liste und einem Vektor ist, dass Listen Objekte unterschiedlichen Typs enthalten können. Außerdem lassen sich die einzelnen Objekte benennen und die Elemente über ihre Position in der Liste oder ihre Namen referenzieren. Syntaktisch ist zu beachten, dass die einzelnen Elemente mit einer doppelten eckigen Klammer ([[]]) angesprochen werden:

```
> pers.info <- list(name="Friedrich", ehepartner="Maria", anz.kinder=3,
    + alter.kinder=c(4,7,9))
> pers.info[["ehepartner"]]
[1] "Maria"
> pers.info[[2]]
[1] "Maria"
> pers.info[["alter.kinder"]][2]
[1] 7
```

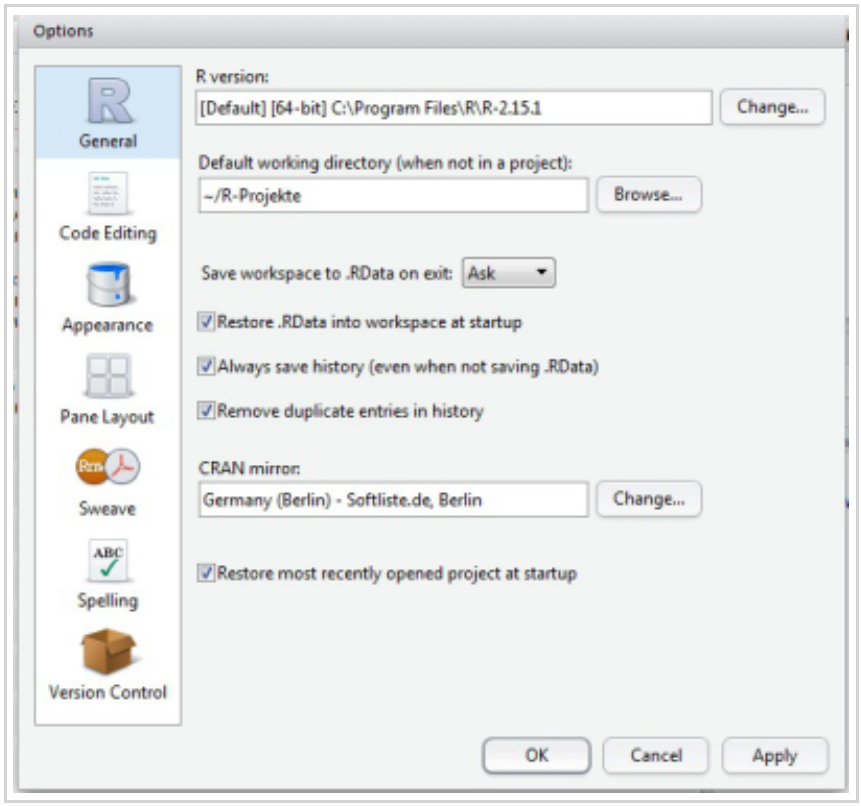
Da die Entwicklung von R-Skripten mit einem Editor und die Ausführung auf der Konsole recht mühsam sind, gibt es einige Entwicklungstools. Diese reichen von erweiterten Editoren mit Code-Highlighting (zum Beispiel Textpad) über Emacs Add-on Packages (ESS: "Emacs speaks Statistics") bis hin zu richtigen GUIs wie JGR (Java Gui for R). Bei der Windows-Version wird Rgui.exe mit installiert. Unter Linux startet man eine Tk/Tcl-Version einfach mit *R --gui=Tk*. Eine Übersicht gängiger GUI findet sich auf einer Projektsite (http://www.sciviews.org/_rgui/).

Eines der Werkzeuge, das einer modernen integrierten Entwicklungsplatform (IDE) am nächsten kommt, ist RStudio (http://www.rstudio.org). Abbildung 2 zeigt den Anfangsbildschirm nach dem ersten Start: Auf der linken Seite befindet sich die R-Console, in der eine R-Session läuft. Auf der rechten Seite oben ist der Workspace zu finden. Beim ersten Start ist er leer. Später sieht man dort alle aktiven Objekte (die man sich in der R-CKonsole mit *ls()* anzeigen lassen kann). Im unteren Bereich rechts sind ein File-Browser, ein Ausgabefenster für alle Plots, die verfügbaren R-Pakete und das Hilfe-Fenster.



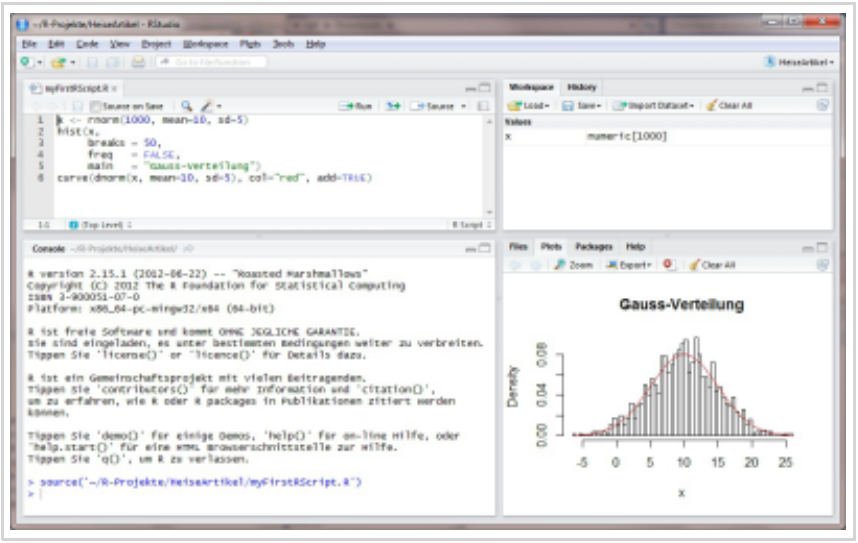
RStudio Startbildschirm (Abb. 2) (http://www.heise.de/developer/artikel/Datenanalyse-mit-R-Teil-1-1845278.html?artikelseite=2;view=zoom;zoom=2)

Es empfiehlt sich, nach dem ersten Start im Menü unter "Tools-Options..." das "Default working directory" anzupassen.



Options-Dialog von RStudio (Abb. 3) (http://www.heise.de/developer/artikel/Datenanalyse-mit-R-Teil-1-1845278.html?artikelseite=2;view=zoom;zoom=3)

Wird RStudio dann neu gestartet, sollte im File-Browser das zu Anfang angelegte Verzeichnis erscheinen. Oben rechts in der Symbolleiste, direkt unter dem Menü, befindet sich die Anzeige des aktuellen Projekts (none). Um ein erstes Projekt anzulegen, klickt man auf den Auswahlpfeil der Drop-down-Box und selektiert "New Project...". Im anschließenden Dialog wird "Create project from: Existing Directory" und das Verzeichnis ~\R-Projekte\HeiseArtikel ausgewählt. Im File-Browser sollte nun die Datei *HeiseArtikel.Rproj* sowie das zuvor erstellte Script *myFirstRScript.R* zu sehen sein. Durch einen Klick auf letzteres öffnet sich im linken oberen Bereich über der Konsole der Editor. Mit "Source" (oben rechts im Editor) lässt sich das Skript starten, und folgendes Bild erscheint:



RStudio nach Ausführung von myFirstRScript.R (Abb. 4) (http://www.heise.de/developer/artikel/Datenanalyse-mit-R-Teil-1-1845278.html?artikelseite=2;view=zoom;zoom=4)

Im Konsole-Fenster ist der *source()*-Befehl zu sehen, und im Grafikbereich erscheint ein Histogramm. Im "Workspace"-Fenster lässt sich erkennen, dass es jetzt ein Objekt (Vektor) x vom Typ *numeric* mit 1000 Werten gibt.
