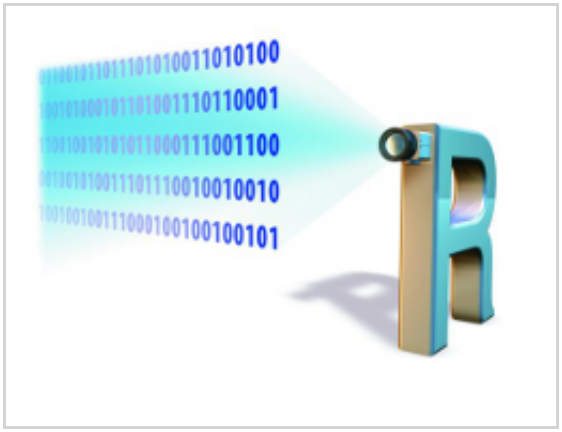
Datenanalyse mit R, Teil 1

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



Mitten im goldenen Zeitalter des Data Mining entwickeln und verwenden Experten überall statistische Methoden: sei es für die zielgerichtete Werbung, die Analyse der Finanzmärkte oder die Entwicklung neuer Medikamente. Zusätzlich befeuern dies die immer größeren Datenmengen, in denen man große Potenziale vermutet. Grund genug, sich die populärste Entwicklungsumgebung für statistische Berechnungen einmal etwas näher anzusehen: R.

R ist eine Open-Source-Entwicklungsumgebung für statistische Analysen, vergleichbar mit anderen statistischen Softwarepaketen wie MATLAB, dem SAS Enterprise Miner oder SPSS Statistics. Sie basiert auf einer eigenen Skriptsprache, die für mathematische Berechnungen optimiert ist. R erlaubt es, Datensätze aus viele Datenquellen

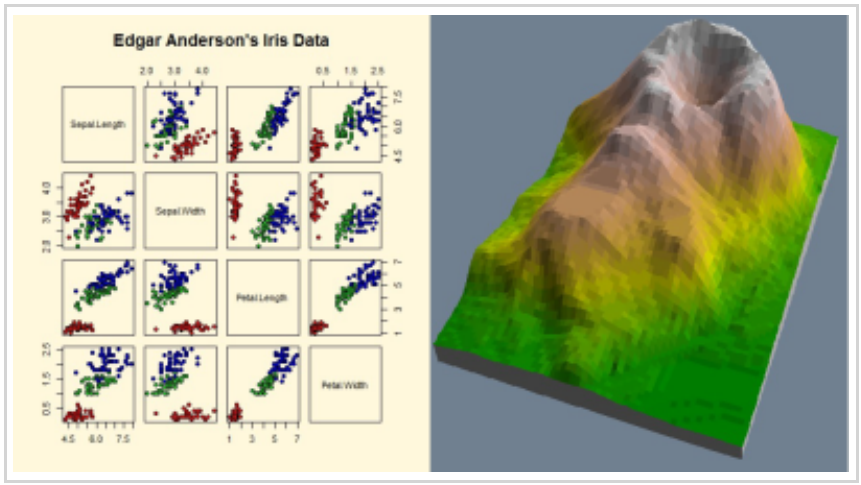
zu laden, diese zu transformieren und anschließend zu untersuchen. So gewonnene Einsichten sind wertvoll und lassen sich häufig zu Vorhersagemodelle weiterentwickeln. R bietet darüber hinaus eine Menge an domänenspezifischen Erweiterungen für besondere statistische Verfahren oder Visualisierungen.

In den letzten Jahren hat sich R zur populärsten Sprache für statistische Analysen entwickelt (siehe hierzu: 2011 Data Miner Survey (http://rexeranalytics.com/Data-Miner-Survey-Results-%202011.html) von Rexer Analytics). Allen voran nutzen sie Big-Data-Firmen wie Google und Facebook, die sich auch aktiv an der Weiterentwicklung beteiligen. In der Geschäftswelt, wo das für wissenschaftliche Zwecke entwickelte R immer mehr Verbreitung findet, hat sich der Begriff Predictive Analytics (PA) entwickelt: Statt Daten der Vergangenheit zu betrachten und zu analysieren wie noch bei Business Intelligence, sind Prognosen über zukünftige Entwicklungen Ziel von PA. Die Erkenntnisse reichen von der Entdeckung von Zusammenhängen und Trends (zum Beispiel im Kundenverhalten) über quantitative Prognosen auf der Basis statistischer Modelle bis hin zur Automatisierung von Entscheidungsprozessen.

Für R wird, wie für alle Programmiersprachen, eine Laufzeitumgebung benötigt. Diese ist für Linux, Mac OS X und Windows verfügbar und steht auf der Projektsite (http://cran.r-project.org/) bereit. Die Basisausführung mit einigen grundlegenden Statistikpaketen befindet sich samt detaillierter Installationsanleitung im Unterverzeichnis *base*. Folgende Schritt-für-Schritt-Anleitung beschränkt sich auf die Windows-Version, die Arbeitspunkte sollten allerdings analog für alle verfügbaren Binaries umzusetzen sein.

Die R-Konsole, eine grafische IDE wird später vorgestellt, lässt sich mit dem Aufruf von *R.exe* aus dem Installationsverzeichnis heraus starten. Bei erfolgreicher Eingabe sollten nun Informationen zur Versionsnummer und einige Hinweise zum Aufruf der Hilfe und Demonstrationsanwendungen auf dem Bildschirm erscheinen.

Durch das Eingeben von *demo(graphics)* und einer Bestätigung via RETURN-Taste lässt sich eine Demo der grafischen Ausgabemöglichkeiten starten (siehe Abb. 1). Ein Klick mit der Maus in das sich öffnende Fenster bewirkt, dass im Terminal der Code der erzeugten Plots erscheint. Zum Verlassen der R-Konsole ist der Befehl *q()* vorgesehen.



Zwei Beispiele aus demo(graphics) und demo(persp) (Abb. 1) (http://www.heise.de/developer/artikel/Datenanalyse-mit-R-Teil-1-1845278.html?view=zoom;zoom=1)

Das Workspace-Image ist das Speicherabbild des laufenden R-Prozesses. Beim Verlassen von R besteht die Möglichkeit, das Image in der Datei *.Rdata* zu speichern, um es beim nächsten Start automatisch wieder laden zu können.

Am besten beginnt man mit dem Anlegen eines Verzeichnisses für R-Projekte (zum Beispiel *~\R-Projekte\HeiseArtikel*) im eigenen *Home*-Verzeichnis, von wo aus R auch gestartet werden sollte. Da der Installer unter Windows das R-Installationsverzeichnis nicht in die *PATH*-Variable einträgt, ist die Session mit dem vollen Pfad zu starten:

```
"C:\Program Files\R\R-2.15.0\bin\x64\R.exe" --no-save
```

Der verwendete Parameter *--no-save* verhindert beim Verlassen von R mit *q()* das Speichern des Images. Für das hier betrachtete Beispiel sind als Erstes 1000 normal verteilte Zufallszahlen mit dem Mittelwert 10 und der Standardabweichung 5 zu generieren. Sie sollen danach in einem Vektor namens *random.numbers* unterkommen.

```
> random.numbers <- rnorm(1000, mean=10, sd=5)
```

Bereits an dieser Zeile erkennt man einige Eigenheiten von R: Beispielsweise können die Variablennamen einen Punkt enthalten, was in der R-Community auch weit verbreitet ist. Eine Bedeutung hat er allerdings nicht. Als Zuordnungs-Operator lässt sich <- oder = verwenden, wobei man sich mit zweiterem eher als "Newbie" zu erkennen gibt. Informationen zum Programmierstil sind in einem Styleguide (http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html) zu finden.

Den Vektor *random.numbers* hat R mit der letzten Anweisung also im Workspace angelegt. Mit *ls()* lassen sich die dort vorhandenen Objekte anschauen:

```
> ls()
[1] "random.numbers"
```

Der Befehl *str(object)* zeigt die Struktur eines Objekts:

```
> str(random.numbers)
 num [1:1000] 8.2 -1.68 17.71 11.88 11.54 ...
```

Am Output ist zu erkennen, dass *random.numbers* ein numerischer Vektor der Länge 1000 ist. Danach folgen die ersten Zufallszahlen.

Um die Session wiederaufrufbar zu machen, wird ein Skript benötigt, das sich mit jedem Text-Editor erstellen lässt. Zum Ausprobieren wird zunächst eine Datei *myFirstRScript.R* im zuvor angelegten Verzeichnis erstellt. Das R-Skript selbst sieht dann beispielsweise folgendermaßen aus:

```
x <- rnorm(1000, mean=10, sd=5)
hist(x,
     breaks = 50,
     freq   = FALSE,
     main   = "Gauss-Verteilung")
curve(dnorm(x, mean=10, sd=5), col="red", add=TRUE)
```

Von der R-Konsole aus lässt sich das Script mit

```
> source('myFirstRScript.R')
```

aufrufen.
