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=3)

Das folgende Beispiel soll in die wichtigsten Befehle und Methoden von R einführen:

Eine Firma verkauft Haushaltsgroßgeräte. Mit dem Erhalt der Ware bekommen die Kunden eine Rechnung, die sie innerhalb von 21 Tagen zahlen müssen. Um ein Skonto von drei Prozent zu erhalten, muss die Zahlung innerhalb von sieben Tagen eingehen. Da die Firma Ausgaben hat, ist sie sehr daran interessiert, ihre Liquidität pro Kalenderwoche für mindestens einen Monat im Voraus zu wissen. Bisher wurden die Einnahmen pro Woche lediglich abgeschätzt.

Das minimale Zeitfenster für einen Zahlungseingang war ein Tag, das maximale 21 Tage nach Rechnungserhalt. Die Firma existiert seit dem 1. Januar 2012 und hat insgesamt 2000 Geräte an 100 Kunden verkauft.

Das Ziel der Analyse ist es, möglichst viel über das Zahlungsverhalten der Kunden zu lernen und für neue Rechnungen das wahrscheinlichste Datum für den Zahlungseingang zu bestimmen. Als Grundlage dienen (generierte und stark vereinfachte) Daten der abgeschlossenen Verkaufsaufträge der Kunden, für die die Zahlungsdauer, also die Anzahl Tage von der Rechnungsstellung bis zum Zahlungseingang, bekannt ist.

Das Vorgehen im folgenden Beispiel ist in fünf Schritte unterteilt, die auch für echte Analyseprojekte in der Praxis typisch sind:

1. Vorbereitungen: Daten sind zu laden und in das richtige Datenformat zu bringen.
2. Erste Begutachtung der Daten: Mit verschiedenen Visualisierungsmöglichkeiten werden die Daten begutachtet

und es lassen sich eventuell erste Zusammenhänge finden.

3. Modellbildung: Aufgrund der ersten Begutachtung der Daten und praktischen Menschenverstands lässt sich ein Modell bilden, das im Laufe der weiteren Schritte überprüft werden kann und gegebenenfalls anzupassen ist.
4. Analyse des Modells: Mit interaktiven Visualisierungsmöglichkeiten und Clustering-Methoden lässt sich das Modell analysieren. Im vorliegenden Beispiel können die Kunden beispielsweise nach ihrem Zahlungsverhalten in Kategorien aufgeteilt werden.
5. Prognose und Test: Das Modell erlaubt es abzuschätzen, wann eine offene Rechnung voraussichtlich vom betreffenden Kunden bezahlt wird.

In der Datei *SalesOrders.csv* befinden sich Informationen von 100 Kunden, die zwischen dem 1. Januar 2012 und dem 30. Juni 2012 eine Lieferung und eine Rechnung bekommen haben. Zusätzlich ist dort die Dauer in Tagen bis zum Zahlungseingang vermerkt. Die Datei legt man am besten in einem Ordner *data* im Projektordner *HeiseArtikel* ab.

Die erste Aufgabe besteht darin, die beschriebenen Daten einzulesen. RStudio stellt hierfür eine recht einfache Möglichkeit zur Verfügung: Im oberen Bereich des Workspace-Fensters befindet sich das Icon "Import Dataset". Klickt man darauf, lässt sich die Quelle für den Import bestimmen. Im vorliegenden Fall ist "From Text File..." und dort die Datei *SalesOrder.csv* auszuwählen. Im danach erscheinenden Pop-up-Fenster lässt sich die Struktur der Datei erkennen. Außerdem schlägt der Import-Assistent vor, daraus einen Data Frame mit vier Spalten zu machen. Nach einem erfolgreichen Import befindet sich das neue Objekt *SalesOrders* im Speicher und lässt sich in der Konsole benutzen. Im Workspace-Fenster erkennt man, dass es 2000 Observations (Zeilen) mit 4 Variable (Spalten) umfasst. In der Konsole lässt sich nachlesen, dass zwei Befehle ausgeführt wurden:

```
> SalesOrders <- read.csv("C:/Users/…/R-Projekte/
     HeiseArtikel/data/SalesOrders.csv", sep=";")
> View(SalesOrders)
>
```

read.csv liest das File ein. Für nähere Informationen hierzu genügt ein Klick auf die Registerkarte "Help" im rechten unteren Teilfenster und die Eingabe von *read.csv* im Suchfeld. Der Befehl *View()* zeigt hingegen das Datenobjekt *SalesOrders* im Editor an. Dessen Struktur lässt sich mit dem Befehl *str()* näher betrachten:

```
> str(SalesOrders)
'data.frame':  2000 obs. of  4 variables:
 $ BuyerRef       : int  1088 1078 1075 1091 1004 1068 1011 1077
        1005 1017 ...
 $ DeliveryDate   : Factor w/ 182 levels "2012-01-01","2012-01-02",..:
        1 1 2 2 2 2 2 2 2 ...
 $ TotalPrice     : int  24693 22215 19340 20646 21981 27150 18148 23999 23608
        23007 ...
 $ PaymentDuration: int  15 20 7 6 7 7 6 11 24 26 ...
```

Es handelt sich also um ein *data.frame* mit den vier Variablen (Spalten) *BuyerRef*, *DeliveryDate*, *TotalPrice* und *PaymentDuration*. Eine einzelne Spalte (Variable) spricht man mit dem $-Operator an:

```
> SalesOrders$BuyerRef[1:5]
[1] 1088 1078 1075 1091 1004
```

Ein Blick auf die Struktur der Variablen *DeliveryDate* zeigt, dass es sich hier nicht wie erwartet um ein Datum, sondern einen Faktor (auch als "category" oder "enumerated type" bezeichnet) handelt:

```
> str(SalesOrders$DeliveryDate)
 Factor w/ 182 levels "2012-01-01","2012-01-02",..: 1 1 2 2 2 2 2 ..
```

Der Import-Assistent hat das Datum folglich nicht als solches erkannt und dessen Werte als Strings interpretiert.

Um zu erfahren, wie der Import-Assistent die CSV-Datei importiert hat, ist zunächst die Historie der Eingabebefehle zu durchsuchen. Hierfür kann man in der Konsole den Befehl *history()* eingeben oder in RStudio auf die Registerkarte History im oberen rechten Fenster klicken. Dort findet sich der Befehl *read.csv()*, dessen Online-Hilfe

sich unter *Help* in RStudio (oder mit dem Befehl *help("read.csv")* in der Console) aufrufen lässt. Wenn man sich die Liste der Parameter durchliest, findet man recht schnell den Parameter *stringsAsFactors*, der die Konvertierung steuert. Da dieser standardmäßig auf *TRUE* gesetzt ist, ist der Datenimport also erneut mit *stringsAsFactor=FALSE* durchzuführen.

Hierfür kann der Entwickler zunächst ein neues R-Skript im Projekt anlegen (Menu File-New-R Skript oder durch Klicken auf das +Icon, direkt unter dem Menü-Punkt File). Ein leeres Editor-Fenster mit dem Titel *Untiteled1** wird geöffnet. Als Nächstes ist im History-Fenster die Zeile mit dem *read.csv()*-Befehl zu markieren und auf *Icon To Source* zu klicken. Der Befehl wird in das aktive Editor-Fenster übernommen, wo er sich anpassen lässt. Ein weiterer Parameter ist von Nöten:

```
# load data from csv file
SalesOrders <- read.csv("data/SalesOrders.csv",
                        sep=";",
                        stringsAsFactors=FALSE)
```

Kommentare sind in R mit der Raute # zu kennzeichnen. Nun kann man die Datei im aktuellen Projekt unter dem Namen *preprocessData.R* speichern und das Script ausführen (Kommando Source am oberen rechten Rand des Editors).

Tipp: Durch das Markieren der Checkbox "Source on Save" wird das gesamte File beim Sichern in der Konsole ausgeführt. Durch Eingabe des *str()*-Befehls sollte folgendes Ergebnis erscheinen:

```
> str(SalesOrders)
'data.frame':    2000 obs. of  4 variables:
 $ BuyerRef       : int  1088 1078 1075 1091 1004 1068 1011 1077 ...
 $ DeliveryDate   : chr  "2012-01-01" "2012-01-01" "2012-01-02" ...
 $ TotalPrice     : int  24693 22215 19340 20646 21981 27150 ...
 $ PaymentDuration: int  15 20 7 6 7 7 6 11 24 26 ...
```

Die Variable *DeliveryDate* ist jetzt ein Vektor mit *Strings (characters)*, der sich in ein Datum konvertieren lässt. Hierfür gibt es unterschiedliche Befehle, fast alle beginnen mit *as.ZielTyp*. Zum Ausprobieren kann man einen Character-Vektor der Länge 1 anlegen und ihn dann mit *as.Date()* konvertieren:

```
> today <- "2012-07-02"
> str(today)
 chr "2012-07-02"
> d1 <- as.Date(today, format="%Y-%m-%d")
> str(d1)
 Date[1:1], format: "2012-07-02"
```

Für ein weiteres Experiment lassen sich die ersten zehn Einträge des Data Frame *SalesOrders* mit

```
> SO <- SalesOrders[1:10, ]
```

kopieren. Man beachte das Komma im [-Operator. Da es sich bei einem Data Frame immer um ein zweidimensionales Objekt handelt, benötigt der [-Operator zwei Auswahlkriterien: eines für die Zeilen (hier 1:10) und eines für die Spalten (da keine Selektion stattfindet, sind alle Spalten zu kopieren).

Anschließend kann man die komplette Spalte *DeliveryDate* mit dem Befehls *as.Date()* konvertieren und das Ergebnis wieder mit *str()* ansehen.

```
> SalesOrders$DeliveryDate <- as.Date(SalesOrders$DeliveryDate,
     format="%Y-%m-%d")
> str(SalesOrders$DeliveryDate)
 Date[1:1000], format: "2012-01-01" "2012-01-02" "2012-01-02"
     "2012-01-02" ...
```

Der Befehl *as.Date()* erhält einen kompletten Character-Vektor und liefert einen neuen Vektor des Typs *Date* zurück. Dieser wird wieder der Spalte *DeliveryDate* des Data Frame *SalesOrders* zugeordnet. Daran erkennt man deutlich den Vektor-orientierten Ansatz von R. Fast alle Befehle, die einen Vektor, ein *data.frame*, eine Matrix oder ein Array als Input haben, wenden eine Funktion auf die einzelnen Elemente einer Spalte (oder einer Zeile) an. In anderen Programmiersprachen würde man eher versuchen, einen Loop über den Vektor zu implementieren (siehe Exkurs: Analysemethoden und "Fallen" (http://www.heise.de/developer/artikel/Datenanalyse-mit-R-Teil-1-1845278.html?artikelseite=4)).

Die Sprache R erfreut sich zunehmend größerer Beliebtheit sowohl in akademischen Anwendungen als auch in Unternehmen. Sie bietet eine sehr mächtige Plattform zur Analyse und Visualisierung von Daten und ist frei verfügbar. R bietet einen Funktionsumfang an analytischen und statistischen Verfahren, der kaum einen Wunsch offen lässt. Achtet man auf einige Fallstricke, lässt sich das Potenzial, dass in den Datenmengen schlummert, effizient nutzen. Mehr erfährt man auch in einem zweiten Artikel (http://www.heise.de/developer/artikel/Datenanalyse-mit-R-Teil-2-1847554.html). (jul (mailto:jul@heise.de))

Dr. Ralf Ehret
arbeitete von 1991 bis 1998 am europäischen Teilchenforschungszentrum CERN an Datenanalysen, bevor er zur SAP AG wechselte. Dort beschäftigte er sich unter anderem mit dem Thema Predicitve Analytics für die In-Memory-Technik HANA.

Benjamin Graf
entwickelt als Data Scientist bei der SAP AG statistische Algorithmen und Prognosemodelle auf der HANA-Plattform. Vor seiner Tätigkeit bei SAP beschäftigte er sich an der Universität von Edinburgh mit automatischer Übersetzung und maschinellem Lernen.
