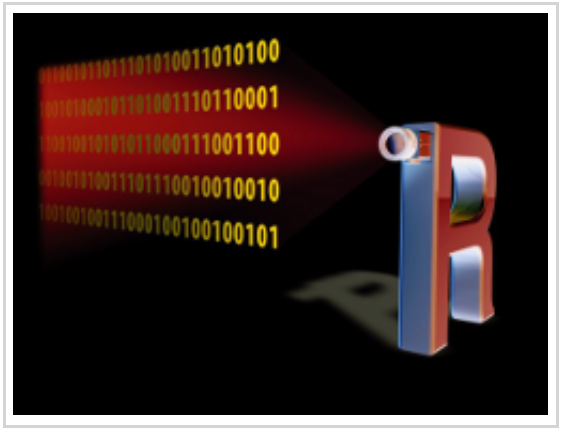
Datenanalyse mit R, Teil 2

by heise Developer • April 26, 2013 • original (http://www.heise.de/developer/artikel/Datenanalyse-mit-R-Teil-2-1847554.html)



Datensammeln allein nutzt nichts, man muss das Gesammelte auch auswerten können. Die Sprache R bietet hierzu zahlreiche analytische und statistische Möglichkeiten. Ob es sich nun um Biotechnologie, Finanzmarktanalysen oder das Kaufverhalten von Kunden handelt: In all diesen Gebieten entstehen ständig neue Data-Mining-Verfahren, die durch das Engagement der Open-Source-Community schnell ihren Weg in R finden. Um sie richtig nutzen zu können, wirft dieser Artikel einen Blick auf die Modellbildung, die Vorgehensweise bei der Analyse sowie Prognosen und Tests in der seit 1993 bestehenden Sprache.

Nachdem die Daten im ersten Teil (http://www.heise.de/developer/artikel/Datenanalyse-mit-R-Teil-1-1845278.html) eingelesen und konvertiert wurden, versucht man im nächsten Schritt, sie zu verstehen. Zunächst schaut man sich dazu die Verteilungen der einzelnen Attribute an. Um mehrere in einem Plot darzustellen, lassen sich die grafischen

Ausgabeparameter mit dem Befehl *par()* ändern. Es lohnt sich, ihn in der Hilfe etwas näher anzuschauen. Hiernach ist der Parameter *mfrow* dafür geeignet, den Ausgabebereich des Fensters in eine Anzahl von Zeilen und Spalten zu teilen. Um nicht immer den ganzen Namen des Data Frame und das mit $ gekennzeichnete Attribut zu tippen, gibt es den Befehl *attach(ObjectName)*, der auf den ersten Blick am ehesten mit einem *setNamespace* zu vergleichen ist. Aber Vorsicht: Eigentlich legt er nur eine Suchreihenfolge für Ausdrücke fest, die man sich über den Befehl *search()* anschauen kann. Um die Namen der Objekt-Attribute zu erfahren, ist *names(Object)* einzugeben:

```
> # Vorbereitungen
> par(mfrow=c(2,2))          # Teilt das Ausgabefenster in 4 Bereiche
> attach(SalesOrders)        # nie in Scripten verwenden!
> names(SalesOrders)         # Spaltennamen auslesen
[1] "BuyerRef"    "DeliveryDate"  "TotalPrice"   "PaymentDuration"
```

Die Verteilung der Attribute lässt sich durch Eingabe des *hist()*-Befehls in Histogrammen ansehen. Hierbei werden der Wertebereich eines Attributes automatisch in äquidistante Bereiche, sogenannte Bins, unterteilt und die Häufigkeiten der Werte pro Bin bestimmt.

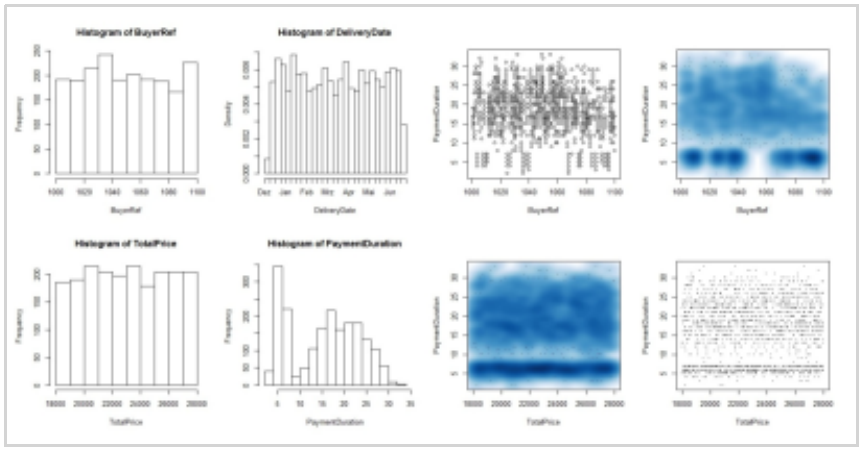
```
> # Eindimensionale Plots
> hist(BuyerRef)             # Histogram für erste Spalte
> hist(DeliveryDate, "weeks")# spezielle hist() Fkt für Date Klasse
> hist(TotalPrice)           # ...
> hist(PaymentDuration)      # ...
```

Ein weiteres Hilfsmittel zum Begutachten von Daten sind Scatter-Plots, die eine Abhängigkeit von zwei Attributen zeigen können. Deshalb nennt man sie auch Korrelationsplots.

```
> # 2-dimensionale Korrelationsplots
> plot(BuyerRef, PaymentDuration)
> smoothScatter(BuyerRef, PaymentDuration)
> smoothScatter(TotalPrice, PaymentDuration)
```

```
> plot(TotalPrice, PaymentDuration, cex=0.3)
```

Das Ergebnis ist in Abbildung 1 zu sehen:



Ergebnisplots zur Visualisierung (Abb. 1) (http://www.heise.de/developer/artikel/Datenanalyse-mit-R-Teil-2-1847554.html?view=zoom;zoom=1)

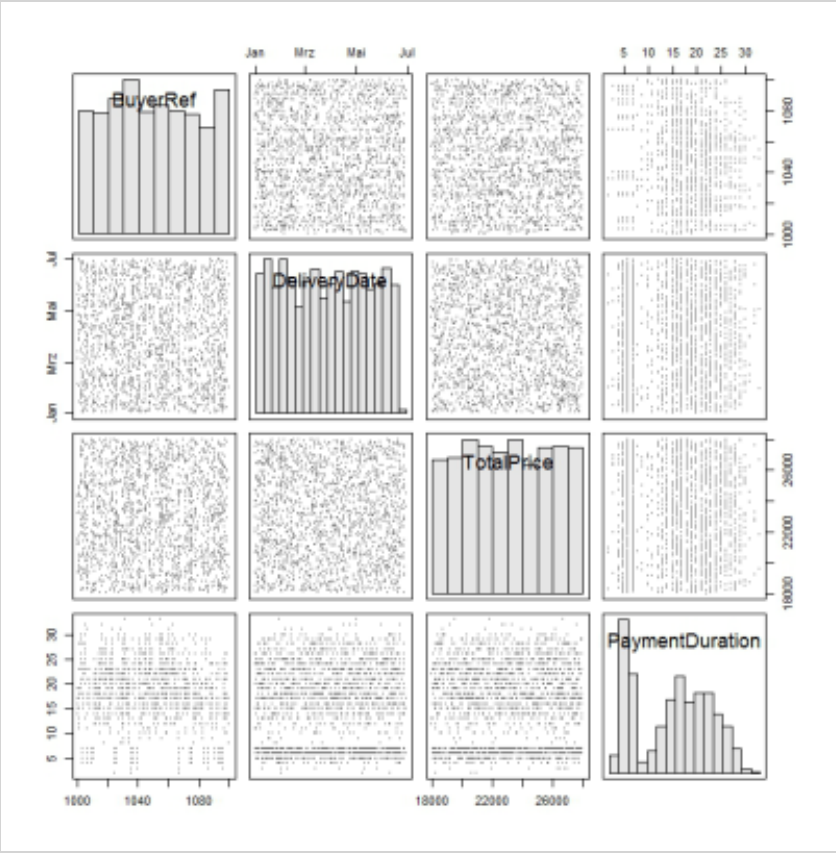
Auf der linken Seite der Abbildung sind die eindimensionalen Histogramme der einzelnen Attribute zu sehen. Während die ersten drei Histogramme kaum auffällig sind, erkennt man bei der *PaymentDuration* einen schmalen Peak bei etwa 5 und einen breiten Peak bei etwa 19 – das erste Anzeichen dafür, dass es ein unterschiedliches Zahlungsverhalten gibt.

Der rechte Teil von Abbildung 1 zeigt die Abhängigkeit des Attributs *PaymentDuration* von *BuyerRef* und *TotalPrice*. Für das Attribut *BuyerRef* wurde der Scatterplot mit zwei verschiedenen Methoden erzeugt: Das erste Diagramm zeigt den mit *plot()* entstandenen Standard-Scatterplot. Hierbei wird aus jeder Zeile der Daten ein Punkt des Werte-Paares (BuyerRef, PaymentDuration) erzeugt. Ein Nachteil von plot ist, dass sich bei einer großen Anzahl von Punkten kaum etwas erkennen lässt und alle einzeln gezeichnet werden müssen, was bei einer Milliarde Punkten etwas länger dauert. Die Methode *smoothScatter()* hingegen berechnet die Dichte der Punkte für unterschiedliche Bereiche und zeigt sie an. Enger besetzte Gebiete erscheinen dunkler. Durch die Verwendung der

Funktion *smoothScatter()* geht allerdings Information verloren. Eine weitere Möglichkeit zur Visualisierung ist, die Punktgröße im Standart-Plot mit dem Parameter *cex* in der *plot()*-Funktion zu reduzieren. Beim Attribut *TotalPrice* (letzter Plot) erkennt man jetzt deutlich, dass es sich bei *PaymentDuration* um eine Integer-Größe handelt.

Eine Möglichkeit, die obigen Daten in nur einem Bild darzustellen, ist die Verwendung der Funktion *scatterplotMatrix()* aus dem Paket car:

```
> scatterplotMatrix(SalesOrders, diagonal="histogram", cex=0.5,
+                   smooth=FALSE, reg.line=FALSE)
```



Visualisierung der Daten mit scatterplotMatrix() (Abb. 2) (http://www.heise.de/developer/artikel/Datenanalyse-mit-R-Teil-2-1847554.html?view=zoom;zoom=2)

Das Package car (Companion to Applied Regression) ist ein Zusatzpaket für Regressionsmodelle. Die Funktion *scatterplotMatrix()* ist die zentrale Plot-Methode des Pakets und führt im Hintergrund eigentlich noch Glättungsalgorythmen (density für die Diagonalelemente und LOESS für die Nichtdiagonalelemente) aus. Für das gezeigte Diagramm wurden diese Standardparameter ausgeschaltet beziehungsweise überschrieben.

Beim Erstellen eines statistischen Modells ist es wichtig, sich im Vorhinein darüber klar zu sein, welche Aufgabe man mit dem Modell lösen möchte. In vorliegenden Fall könnte sie folgendermaßen aussehen:

1. Erstelle aus den historischen Verkaufsaufträgen der Kunden und den einzelnen Zeitdifferenzen von Rechnungsausgang und Zahlungseingang (*PaymentDuration*) Kundenprofile.
2. Ordne die Kunden den Profilen zu.
3. Sage für einen neuen Verkaufsauftrag eines Kunden vorher, was der wahrscheinlichste Termin für den Geldeingang ist.

Zur Analyse stehen die vier Attribute aus unserem Datenobjekt *SalesOrders* zur Verfügung:

- BuyerRef, eine Kundennummer;
- DeliveryDate, das Auslieferungsdatum (und damit das Rechnungsdatum);
- TotalPrice, der Bruttopreis der bestellten Waren;
- PaymentDuration, die Anzahl Tage ab dem Auslieferungsdatum bis die Zahlung erfolgte.

In der Visualisierung der Daten ließ sich erkennen, dass es keine Auffälligkeiten für die Attribute *DeliverDate* und *TotalPrice* gibt, sondern nur in der *PaymentDuration*. So lässt sich allerdings noch nichts über den Zusammenhang zu den einzelnen Kunden aussagen. Um Daten auf Kundenebene zu aggregieren, besteht eine Möglichkeit darin, zunächst Zeitreihenobjekte für jeden Kunden zu erzeugen. Zeitreihen sind Wertepaare aus einer diskreten Messgröße und des Zeitpunkts der Messung. Diese können in konstanten Abständen (äquidistant) oder zufällig

verteilt sein. Im vorliegenden Fall entspricht die Zahlungsdauer (*PaymentDuration*) der Messung und das Datum der Rechnungsstellung (*DeliveryDate*) dem Zeitpunkt. Die Liste aller Werte-Zeit-Paare eines Kunden entspricht seiner Zahlungshistorie.
