Data cleanig in R

von

Stefan Thieß

Abstract:

1.Kapital: Principel of tidy data – TidyR-Paket
Nach der Lehre der Principel of tidy data bestehen folgende Anforderungen an eine Tabelle. 1. Jede Spalte enthält nur ein Attribut. 2. Eine Zeile pro Beobachtungsobjekt. 3. Jede Tabelle enthält nur eine Art von Beobachtungsobjekt. Das TidyR-Paket erhält Methoden zur Veränderung von Tabellen.

2.Kapitel: Typenumwandlung – lubridate-Paket
Werte in Tabellen sind ein Abbild der Wirklichkeit. Grundlegende Datentypen in R lassen sich mit der as.--Funktion umwandeln. Das lubridate-Paket hilft bei der Standardisierung von Kalender- und Uhrzeitangaben.

3.Kapitel: Zeichenketten verändern – StrinR-Paket
Normierung von Zeichenketten ist wichtig um Duplikate zu finden. Zeichenketten lassen sich durch die Methoden des StringR-Pakets verändern.

4.Kapitel: fehlende Werte – is.na-Funktion
Fehlende Werte werden in R mit einem NA dargestellt. Die is.na-Funktion findet fehlende Werte. Der Wiedergabetyp ist ein Wahrheitswert. In verschiedenen Kombinationen lassen sich so die fehlenden Daten finden und die entsprechenden Zeilen oder Spalten ,mit der complete.cases-Funktion, in der Tabelle löschen.

5.Kapitel: extreme Werte – editrules-Paket
Oft sind extreme Werte offensichtliche Fehler. Diese lassen sich durch das editrules-Paket finden. Mit der editset-Funktion lassen sich Bedingungen formulieren. Diese können auch genutzt werden um fehlende Werte zu ersetzen.

6. Kapitel: R und Datenbanken – RmySQL-Paket
Data-Warehouse beschreibt die Datenverdichtung in einem System mit unterschiedlichsten Quellen. Anforderung an R ist: Daten von unterschiedlichen Servern importieren, anschließend die Daten zu bereinigen und die bereinigten Daten an den Hauptserver exportieren. Mit RmySQL kann eine Verbindung zu MySQL-Servern hergestellt werden.

1.Kapitel

In dieser Arbeit befasse ich mich mit der Datenbereinigung in der Programmiersprache R. Grundsätzlich ist die Datenbereinigung (engl. data cleaning) ein Prozess innerhalb der Datenanalyse. Er bildet genau die Schnittstelle zwischen der Datensammlung und der Auswertung dieser.

Die Datenbereinigung ist grundsätzlich in 2 Schritte aufzuteilen.
Zuerst müssen die Daten standardisiert werden, danach müssen diese bereinigt werden.

Meist sind statistische Daten in 2-Dimensionalen Tabelle angeordnet. Über die Ordnung innerhalb einer Tabelle hat sich Bradley Wickham Gedanken gemacht und daraus seine „Principels of tidy data „ entwickelt. Auf Grundlage seiner Theorie hat dieser dann das Paket TidyR in R für uns bereitgestellt.

Aus Sicht von Wickham hat die perfekte Tabelle folgende Eigenschaften.
In der ersten Spalte sind die Beobachtungsobjekte. Also diejenigen Objekte, über welche die Tabelle Informationen bereithält. Die erste Zeile wiederum ist für die Bezeichnungen der Werte da. Der Wertebereich befindet sich darunter. Jede Spalte besteht aus einem Attribut, wie z.B Geschwindigkeit, Größe oder Gewicht. Jede Zeile besteht aus den Werten , die ein einzelnes Beobachtungsobjekt betreffen, also Max Mustermann , Geschwindigkeit 30km/h, Größe 1,80, Gewicht 73kg.

Aus dieser Vorstellung heraus formuliert Wickham die Anforderungen an „tidy data“

- Jede Spalte enthält genau ein Attribut.
- Jede Zeile beinhaltet nur Daten eines bestimmten Beobachtungsobjektes.
- Jede Tabelle enthält nur eine Art von Beobachtungsobjekt. [1]

Daraus folgt auch eine Negativabgrenzung – „dirty data“.
Besonders häufig treten folgende Eigenschaften von „dirty data“ auf:

- In der ersten Spalte, wo eigentlich die Spaltennamen sind, sind Werte enthalten.
- Variablen sind im Wertebereich.
- Mehrere Variablen sind in einer Spalte.
- Mehrere Beobachtungsobjekte sind in einer Tabelle.[2]

Um diese schmutzigen Daten zu reinigen, hat Wickham das Paket TidyR für die Sprache R geschrieben.
In dem Paket sind vor allem Funktionen zum verändern von Tabellen enthalten. Im nachfolgenden stelle ich die 4 Funktionen – gather, seperate, spread und unite – vor. Jede dieser Funktionen löst eines der Probleme von „dirty data“.

1 Wickham: Tidy Data .,Abschnitt 2.3
2 Wickham: Tidy Data. Abschnitt 3

Problem: Spaltennamen sind Werte

Die oberste Zeile einer „tidy-data“ sollte stets eine beschreibende Funktion für die darunter stehenden Werte haben. An der unten stehenden Tabelle „new3“ sind die Spaltennamen, also die oberste Zeile mit den Werten Kauffrau, Lehrerin und Floristin beschriftet. Diese beschreiben nicht die darunter liegenden Werte. Die darunter liegenden Werte sind einfache Wahrheitswerte, beschreiben also, ob das Beobachtungsobjekt , hier Lisa, Sarah und Karin, in Relation zu dem Spaltenname wahr oder falsch ist. Somit sind die Spaltennamen Werte und müssen in den Wertebereich , also unterhalb der ersten Spalte gebracht werden.
Die Funktion 'gather', zu deutsch versammeln, fügt alle Spaltennamen zu einer neuen Spalte im Wertebereich zusammen. Der Aufruf der Methode folgt dabei folgender Syntax. Zuerst ist die Methode 'gather' aufzurufen. Als Parameter ist an erster Stelle die Bezeichnung des Datensatzes zu übergeben. Danach folgt der Spaltenname für die Werte, die fälschlicherweise in der obersten Spalte sind (hier also Beruf). Danach ist der Name für die Werte einzugeben, die unterhalb der Werte sind, die als Spaltennamen deklariert wurden. Darauf sind dann diejenigen Spalten zu benennen, die nicht in Werte umgewandelt werden sollen(hier -Name). Diese Spaltennamen sind mit einem '-' zu versehen. Alles ist mit einem ',' voneinander zu trennen.

```
> New3
   Name Kauffrau Lehrerin Floristin
1  Lisa        1        0         0
2 Sarah        0        1         0
3 Karin        0        0         1

> tidyr::gather(New3,Beruf, Wahr, -Name)
   Name     Beruf Wahr
1  Lisa  Kauffrau    1
2 Sarah  Kauffrau    0
3 Karin  Kauffrau    0
4  Lisa  Lehrerin    0
5 Sarah  Lehrerin    1
6 Karin  Lehrerin    0
7  Lisa Floristin    0
8 Sarah Floristin    0
9 Karin Floristin    1
```

Problem: Variablen sind im Wertebereich

Der offensichtliche Fehler dieser Tabelle besteht in der Mehrfachnennung der Beobachtungsobjekte, hier Lisa und Sarah. Diese Mehrfachnennung resultiert aus der Abhängigkeit von den Kleidungsgegenstände zu ihrer Anzahl. Der Wertetyp ist hier nicht das Kleidungsstück, sondern das einzelne Kleidungsstück, die Jacke, Hose, Schuhe, da sich die Anzahl direkt auf die einzelnen Kleidungsstücke beziehen und nicht die Gesamtanzahl der Kleidungsstücke beschreiben.
Somit sind die Variablen Jacken, Hosen, Schuhe in die oberste Spalte zu schreiben und die dazugehörigen Werte (die Anzahl) in den darunterliegenden Wertebereich. Die hierfür vorliegende Funktion aus dem Paket TidyR ist die spread-Funktion, zu deutsch: Ausbreiten. Das Aufrufen der

Methode geht dabei wie folgt: Zuerst ist die Methode spread aufzurufen. In Klammern folgt nacheinander, der Datensatz der verändert werden soll, der Name der Spalte, deren „Werte“ zu Spaltennamen werden sollen (hier: Kleidungsgegenstände, es ist auch eine Mehrfachnennung möglich) , gefolgt von dem Namen der Spalten, die in dem Wertebereich bleiben sollen (hier: Anzahl).

```
> Pre5
   Name Kleidungsgegenstände Anzahl
1  Lisa                Jacken     12
2  Lisa                 Hosen      8
3  Lisa                Schuhe     27
4 Sarah                Jacken     24
5 Sarah                 Hosen     13
6 Sarah                Schuhe     21

> tidyr::spread(Pre5, Kleidungsgegenstände, Anzahl)
   Name Hosen Jacken Schuhe
1  Lisa     8     12     27
2 Sarah    13     24     21
```

Problem: mehrere Variablen sind in einer Spalte

Wenn mehrere Variablen und dadurch auch mehrere Werte in einer Spalte sind, ist die seperate-Funktion des TidyR-Paketes zu verwenden. Im Ergebnis werden so aus einer Spalte mehrere Spalten, je nachdem, wie viele verschiedene Wertetypen in der einen Spalte vorliegen. Im obigen Beispiel sind die Wertetypen Alter und Größe in einer Spalte zusammengefasst. Um diese zu entzweien gilt es folgende Syntax zu beachten. Wiederum ist die separate-Methode aufzurufen. In der Klammer sind folgende Parameter einzugeben. Zuerst der Name des Datensatzes ,welches verändert werden soll, gefolgt von derjenigen Spalte, welche geteilt werden soll. Danach sind die Spaltennamen zu nennen in die aufgeteilt werden soll. Hierbei ist die Spaltennamen in einen Vektor zu schreiben ('c(...)') . Hierauf folgt die Angabe des Symbols, durch sep=“(SYMBOL)“ , welches die Werte trennt.

```
> Pre4
   Name Alter.Größe     Beruf
1  Lisa      25/169  Kauffrau
2 Sarah      30/160  Lehrerin
3 Karin      51/172 Floristin

> tidyr::separate(Pre4,Alter.Größe,
c("Alter","Größe"), sep="/")
   Name Alter Größe     Beruf
1  Lisa    25   169  Kauffrau
2 Sarah    30   160  Lehrerin
3 Karin    51   172 Floristin
```

Problem: Mehrere Beobachtungsobjekte sind in einer Tabelle

Die Tabelle beinhaltet zwei Beobachtungseinheiten, einmal den Menschen mit den zu ihnen gehörenden Wertetypen Alter und Größe und dem Tier mit den dazugehörigen Wertetypen Tierart und Größe des Tieres. Da in einer Tabelle stets nur ein Beobachtungsobjekt sein soll, muss aus der vorliegenden Tabelle ,zwei Tabellen gemacht werden. Hierfür ist keine Methode aus dem TidyR-Paket notwendig. Die Tabelle wird einfach durch die Speicherung eines Teils der Tabelle in ein neues Objekt gelöst. Man gibt zuerst den Namen des neuen Objektes an, gefolgt von einem umgekehrten Pfeil ('<-') . Darauf ist der Name der ursprünglichen Tabelle anzugeben und dann sind in eckigen Klammern diejenigen Spalten einzugeben, die in das Objekt gespeichert werden sollen.

```
> Pre6
   Name Alter Größe Tier_Name   Tierart Größe_Tier
1  Lisa    25   169     Bello      Hund         40
2 Sarah    30   160     Felix     Katze         20
3 Karin    51   172  Flauschi Kaninchen         10

> Pre62←Pre6[1:3]

> Pre62
   Name Alter Größe
1  Lisa    25   169
2 Sarah    30   160
3 Karin    51   172
```

2.Kapitel

Typenumwandlung in R

Die Notwendigkeit sich über Typenumwandlung in einer Statistischen Tabelle Gedanken zu machen folgt aus dem Fakt, dass jeder Wert in einer Tabelle immer auch eine Abbildung eines Wertebereiches in der realen Welt ist. Die grundlegenden Typen in R sind:

- character: "…."
- numeric: 23.44, 120 ,NaN, Inf
- integer: 4L, 1223L
- factor: factor ("Hello"), factor (8)
- logical: TRUE, FALSE, NA

Die Konvertierung eines Datentyps in einen anderen Datentyp geschieht durch die as-Funktion. Zuerst ist die as-Funktion einzugeben, danach eine Punktnotation und danach der Gewünschte Datentyp, in den umgewandelt werden soll. Für eine Datenumwandlung in ein integer-Typ würde dies also wie folgt aussehen: 'as.integer'. [3]

3 De Jonge, van der Loo: An introduction to data cleaning with R. Abschnitt 2.3.1

Auch hierzu wieder ein kurzes Beispiel. In der Tabelle Pre7 ist die Spalte Besitzen_ein_Auto vom Datentyp integer. Es sind jedoch nur Nullen und Einsen, was darauf schließen lässt, dass der Autor der Tabelle hiermit Wahrheitswerte ausdrücken wollte. Um in einer Tabelle auf eine bestimmte Spalte zuzugreifen ,ist der Einsatz des Dollarzeichenoperators ($) notwendig. Vor dem Dollarzeichen ist der Name der Tabelle anzugeben und nach dem Dollarzeichen ist der Name derjenigen Spalte anzugeben, auf welche zugegriffen werden soll.

```
> Pre7
   Name Besitzen_ein_Auto
1  Lisa                 1
2 Sarah                 0
3 Karin                 1

> Pre7$Besitzen_ein_Auto←as.logical
(Pre7$Besitzen_ein_Auto)

> Pre7
   Name Besitzen_ein_Auto
1  Lisa              TRUE
2 Sarah             FALSE
3 Karin              TRUE
```

Ein besonderes Augenmerk möchte ich auf den Umgang mit Datums- und Uhrzeitangaben legen. In mannigfaltiger Form werden Datumsangaben heutzutage gespeichert. Bei banalen Dingen wie dem Ausleihen eines Buches, bis hin zum Speichern von Vertragsdaten, überall spielt das Datum eine entscheidende Rolle. Dabei variiert die Darstellung von Datumsinformationen von Region zu Region und Mensch zu Mensch. So wird in den USA der 12.Mai 2010, den wir auch als 12.05.2010 schreiben, als May 12, 2010 oder 05-12-10 geschrieben. Auch die Trennungssymbole kommen in den unterschiedlichsten Formen vor. Ebenso verhält es sich mit der Uhrzeit. Um bei Datums- und Uhrzeitangaben eine einheitlichen Datendarstellung zu haben wurde das lubridate-Paket für R bereitgestellt. Die Autoren des Paketes sind Garett Grolemund und Hadley Wickham.

Der Programmierer muss erkennen um welches Datumsformat es sich handelt. Das lubridate-Paket kann nur Datumsangaben vereinheitlichen, jedoch nicht automatisch erkennen. Eine Datumsangaben ist mit y für das Jahr, m für den Monat und d für den Tag anzugeben. Der Programmierer muss dies explizit angeben. Für die Uhrzeitangabe ist dies ebenso anzugeben. Auf den Unterstrich als Trennungsoperator zwischen Datums- und Uhrzeitangabe ist zu achten.

```
> lubridate::ymd_hms("2016/7/13 10.30.09")
[1] "2016-07-13 10:30:09 UTC"
```

Im folgenden Beispiel sind Geburtstage in unterschiedlichen Formaten gespeichert. Es besteht keine Einheitlichkeit der Daten. Um die Daten zu vereinheitlichen ist auf die Zeile Geburtstag mit dem Dollarzeichenoperator zuzugreifen und mit dem dmy-Operator des lubridate-Pakets die entsprechende Zeile zu vereinheitlichen.

```
> PreTime
   Name Alter     Geburtstag
1  Lisa    25       21.05.90
2 Sarah    30          7/3/84
3 Karin    51 16. April 1969
> PreTime$Geburtstag←lubridate::dmy
(PreTime$Geburtstag)

> PreTime
   Name Alter Geburtstag
1  Lisa    25 1990-05-21
2 Sarah    30 1984-03-07
3 Karin    51 1969-04-16
```

3.Kapitel

Normierung von Zeichenketten

Bei dem Eintippen von Zeichenketten eines Benutzers eines PC, ist dieser durch die Zeichen der Tastatur beschränkt. Jedoch kann es bei der Eingabe von derselben Information zu Inkonsistenzen kommen. So kann es bei der Eingabe eines Straßennamens zu einer Vielzahl von Abweichungen kommen. Die Schreibweise der Straßennamen kann unterschiedliche sein, die Verwendung von 'ss' statt 'ß' kann abweichen, die Groß und Kleinschreibung , insbesondere bei Hausnummern mit Zusatzbuchstaben kann abweichen und die Anzahl der Leerzeichen kann variieren. Dies alles macht für die Menschliche Wahrnehmung keinen großen Unterschied, für die Vergleichbarkeit von Daten auf dem Computer jedoch schon. Nur exakt gleiche Zeichenketten sind (ohne Vergleichsfunktion mit Ähnlichkeitsmetriken) als Duplikate erkennbar. Schon ein Leerzeichen zu viel zwischen Straßennamen und Hausnummer, führt zu zwei unterschiedlichen Werten. Deshalb ist eine Normierung von Zeichenketten von großer Bedeutung. Natürlich ist es modernen Vergleichsfunktionen mit Ähnlichkeitsmetriken möglich Duplikate zuerkennen, dennoch erhöht eine Normierung die Erfolgsaussichten dieser.

Zur Manipulation von Zeichen benutze ich das Stringr-Paket von Hadley Wickham. Dabei benutze ich folgende Funktionen:

- str_trim: Entfernt alle Leerzeichen vor dem ersten und/oder nach dem letzten Zeichen.
- str_replace_all: Entfernt ein bestimmtes Zeichen und ersetzt dieses durch ein anderes.
- str_pad: Fügt Zeichen an die Zeichenkette an.
- toupper: Macht alle Kleinbuchstaben zu Großbuchstaben.

An der folgenden selbst gestellten Aufgabe werde ich Beispielhaft die Anwendbarkeit der Funktionen erläutern.

Es sei die Tabelle Stringr, in der unter anderem die Firmen-ID von Mitarbeiter gespeichert wird. Eine Interne Richtlinie kann wie folgt lauten:

- Die Firmen-ID muss mit der Zeichenkette „NEW“ beginnen
- Die Firmen-ID muss insgesamt 7 Zeichen lang sein
- Alle Zeichen müssen hintereinander geschrieben werden

```
> StringR
   Name  Firmen.ID     Beruf
1  Lisa  oLd135     Kauffrau
2 Sarah NEW23   85  Lehrerin
3 Karin    new5821 Floristin
```

Schauen wir uns die Firmenwerte genauer an.

```
> StringR$Firmen.ID
[1] "oLd135   "  "NEW23   85" "   new5821"
```

Zu erkennen ist:

- viele unnötige Leerzeichen sind vor ,zwischen und hinter der Firmen-ID
- Buchstaben sind in Klein und Großschreibung
- Die Firmen-ID an der ersten Stelle hat weniger als die geforderten 7 Zeichen

Unter Einsatz der trim-Funktion werden die Leerzeichen vor dem ersten und nach dem letzten Zeichen gelöscht. Als Wert wird die zu ändernde Spalte der Tabelle übergeben. Ebenfalls möglich ist es , nur Leerzeichen vor dem ersten oder nach dem letzten Zeichen zu löschen. Hierbei muss als zusätzlicher Wert eine Variable c mit dem Wert „left“ oder „right“ übergeben werden.

```
> StringR$Firmen.ID←stringr::str_trim (StringR$Firmen.ID)
> StringR$Firmen.ID
[1] "oLd135"     "NEW23   85" "new5821"
```

In der replace_all-Funktion wird als erstes das Objekt übergeben, welches man verändern will, dann unter Angabe des Schlüsselwortes pattern den String-Wert, der zu ändern ist und dann unter dem Schlüsselwort repl den neue String-Wert. Im vorliegenden Beispiel werden so die Leerzeichen innerhalb der mittleren Firmen-ID entfernt.

```
> StringR$Firmen.ID←stringr::str_replace_all
(StringR$Firmen.ID, pattern = " ", repl = "")
> StringR$Firmen.ID
[1] "oLd135"  "NEW2385" "new5821"
```

Mit der pad-Funktion wird ein String auf seine Länge überprüft. Ist der String dabei kürzer als die mit dem Schlüsselwort width übergebene Zahl, so werden die fehlenden Zeichen durch die Zeichen erweitert, welche mit dem Schlüsselwort pad übergeben wird. side bestimmt dabei, von welcher Seite die neuen Zeichen angefügt werden sollen. Im Beispiel werden die fehlenden Zeichen mit der

0 von der rechten Seite aufgefüllt, soweit die Firmen-ID weniger als 7 Zeichen lang ist.

```
> StringR$Firmen.ID←stringr::str_pad (StringR$Firmen.ID,
width = 7, side = "right", pad = "0")
> StringR$Firmen.ID
[1] "oLd1350" "NEW2385" "new5821"
```

Nun wird die toupper-Funktion verwendet. Diese stammt nicht aus dem stringR-Paket, sondern gehört zu den Standardfunktionen von R. Alle Buchstaben werden zu Großbuchstaben.

```
> StringR$Firmen.ID←toupper(StringR$Firmen.ID)
> StringR$Firmen.ID
[1] "OLD1350" "NEW2385" "NEW5821"
```

Wiederum wird die replace_all-Funktion benutzt und alle "OLD" Strings durch "NEW" ersetzt.

```
> StringR$Firmen.ID←stringr::str_replace_all
(StringR$Firmen.ID,
pattern = "OLD", repl = "NEW")

> StringR
   Name Firmen.ID     Beruf
1  Lisa   NEW1350  Kauffrau
2 Sarah   NEW2385  Lehrerin
3 Karin   NEW5821 Floristin
```

4.Kapitel

Fehlende Werte

Fehlende Werte können in Tabellen in unterschiedlicher Form vorkommen. In R werden fehlende Werte mit NA gekennzeichnet. In weiteren Programmen können fehlende Werte auf mit #N/A (Excel), mit einem Punkt ("." in SAS) oder auch einfach mit einer Leerzeichen gekennzeichnet sein. Zunächst werde ich auf die Suche nach fehlenden Werten in R beschreiben. Darauf erkläre ich den Umgang mit fehlenden Werten.

Die Tabelle „mis“ mit der ich im vorliegenden arbeiten werde, ist klein und übersichtlich. Die meisten Tabellen im wissenschaftlichen und wirtschaftlichen Bereich bestehen meist aus mehreren hunderten oder tausenden von Spalten und Zeilen, sodass fehlende Werte nicht sofort erkannt werden können. Die verwendeten Funktionen sind bereits in R impliziert und bedürfen daher keine Installation von Paketen.

```
> mis
   Name Alter Größe
1  Lisa    25   169
2 Sarah    NA    NA
3 Karin    NA   172
```

Die is.na-Funktion gibt Wahrheitswerte wieder. Dabei wird für jedes NA der Wert TRUE zurückgegeben. Die Tabelle wird als gesamtes in Form von Vektoren für die einzelnen Zeilen wiedergegeben.

```
> is.na(mis)
      Name Alter Größe
[1,] FALSE FALSE FALSE
[2,] FALSE  TRUE  TRUE
[3,] FALSE  TRUE FALSE
```

Fügt man ein any an den oben genannten Code an, so wird überprüft ob überhaupt ein fehlender Wert vorkommt. TRUE wird zurückgegeben sobald mindestens ein Wert NA ist.

```
> any(is.na(mis))
[1] TRUE
```

Mit der sum-Funtion werden alle fehlenden Werte aufaddiert. Das Ergebnis ist dann die Summe der fehlenden Werte.

```
> sum(is.na(mis))
[1] 3
```

Die complete.cases-Funktion überprüft jeweils die gesamte Zeile. Ist eine Zeile vollständig, wird der Wahrheitswert TRUE zurückgegeben. Im vorliegenden Fall ist nur die erste Zeile (Lisa) vollständig.

```
> complete.cases(mis)
[1]  TRUE FALSE FALSE
```

Nun widmen wir uns dem Umgang mit fehlenden Werten. Zum einen können die Zeilen und die Spalten mit fehlenden Werten einfach aus der Tabelle gelöscht werden. Die na.omit-Funktion entfernt alle Zeilen, in der mindestens ein fehlender Wert vorkommt. Somit verbleibt als einzige Zeile die Erste.

```
> na.omit(mis)
  Name Alter Größe
1 Lisa    25   169
```

Um alle vollständigen Spalten wiederzugeben, greife ich direkt auf die Spalten mit den Eckige-Klammern-Operator zu und prüfe jede Spalte darauf, ob in den Spalten jeweils ein fehlende Wert vorkommt. Ausgegeben werden nur die Spalten, welche kein fehlenden Wert haben.

```
> mis2←mis[,complete.cases(mis)]
> mis2
[1] "Lisa"  "Sarah" "Karin"
```

Wenn die Tabelle Werte enthält, die miteinander in Relation stehen, so kann auch versucht werden, die fehlenden Werte zu ersetzen. Die Relation von Einnahmen, Ausgaben und Gewinn , welche Einnahmen minus Ausgaben gleich Gewinn ist, kann genutzt werden um fehlende Wert zu ersetzen. Mit dem Paket editrule von E. De Jonge and Mark van der Loo können Bedingungen erstellt werden. Die editset-Funktion erlaubt es allgemeine Bedingungen zu formulieren. Diese Bedingung kann als Objekt gespeichert werden und immer wieder verwendet werden. Im unten folgenden Beispiel speichere ich die Bedingung Einnahme minus Ausgaben gleich Gewinn in das Objekt Cor. Im Paket deducorrect ist die Funktion deduImpute , die es ermöglicht die Bedingung Cor auf die Tabelle data anzuwenden.

```
> data
  Firmenname Einnahmen Ausgaben Gewinn
1      FirmaA      1000       NA    600
2      FirmaB        NA      400    900
3      FirmaC       550      300     NA

> Cor←editrules::editset(expression
(Einnahmen - Ausgaben == Gewinn))
> Cor
Edit set:
num1 : Einnahmen == Ausgaben + Gewinn

> data_new<-deducorrect::deduImpute(Cor, data)
> data_new$corrected
  Firmenname Einnahmen Ausgaben Gewinn
1      FirmaA      1000      400    600
2      FirmaB      1300      400    900
3      FirmaC       550      300    250
```

5.Kapitel

Extreme Werte

Extreme Werte sind Werte, die vom Großteil der anderen Werte abweicht. Dabei möchte ich auf eine besondere Form der extremen Werte eingehen, den offensichtlichen Fehlern. Es ist zwischen 3 verschiedenen Arten von offensichtlichen Fehlern zu unterscheiden. So können zum einen Werte aufgrund von Erkenntnissen nicht plausibel sein. Ein Mensch der 234 Jahre alt ist, ist so ein Beispiel. Zum anderen können Werte schlicht weg keinen Sinn ergeben. Ein Menschen der -5 Jahre alt ist, kann es nicht geben. Auch können Werte in verschiedenen Einheiten vorliegen. So kann Größe in Meter, Zentimeter und Inches gemessen werden. Eine Vermischung dieser Einheiten in einer Spalte, kann in der Auswertung der Daten zu Fehlern führen.

Um nun die offensichtlichen Fehler aus der Tabelle editR zu entfernen, erstellen wir folgende Bedingungen:

- Werte des Attributs Alter dürfen nicht kleiner als 0 sein
- Werte des Attributs Alter dürfen nicht größer als 150 sein
- Werte des Attributs Größe müssen mindestens 40 sein
- Werte des Attributs Größe müssen kleiner gleich 275 sein

Diese Bedingungen wird einem Objekt in R zugewiesen. Mit der violatedEdits-Funktion aus dem editrules-Paket kann die Tabelle editR auf offensichtliche Fehler untersucht werden. Die Bedingungen sind nummeriert. So ist die Bedingung '0<= Alter' die Nummer 1 (num1). Diese Nummerierung findet sich im Ausgabewert der violatedEdits-Funktion wieder.

```
> editR
   Name Alter Größe
1  Lisa    25   350
2 Sarah   180    25
3 Karin   -51   172

> (R←editrules::editset(c("Alter>=0",
"Alter<=150","Größe>=40","Größe<=275")))

Edit set:
num1 : 0 <= Alter
num2 : Alter <= 150
num3 : 40 <= Größe
num4 : Größe <= 275
```

```
> editrules::violatedEdits(R,editR)
      edit
record  num1  num2  num3  num4
     1 FALSE FALSE FALSE  TRUE
     2 FALSE  TRUE  TRUE FALSE
     3  TRUE FALSE FALSE FALSE
```

6.Kapitel

Data-Warehouse und R

Abschließend wird die Frage behandelt, wie R und data cleaning in das moderne Konzept des Data-Warehouse eingebunden werden kann. Das Data-Warehouse Konzept beschreibt die „ Sammlung, Verdichtung und Selektion entscheidungsrelevanter Informationen“[4] Der Wunsch ist hierbei also, alle relevanten, geschäftlichen Informationen in einer Datenbank bereitstellen zu können, um dann bessere Entscheidungen für die Zukunft zu treffen. Die Schwierigkeit dabei ist, die Daten aus unterschiedlichsten Quellen zu vereinheitlichen und an die Hauptdatenbank zu übertragen. Für R konkret bedeutet dies, es muss möglich sein sich Daten aus Datenbanken zu importieren, die Daten zu bereinigen und im Anschluss in die Hauptdatenbank zu exportieren.

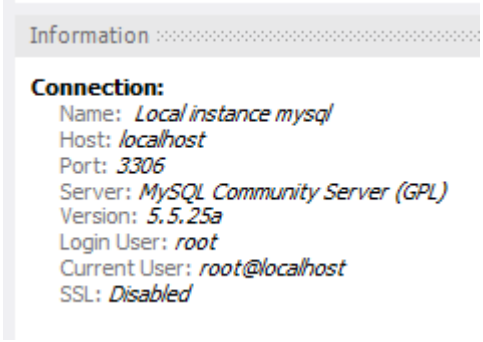
Ich habe mich im folgenden mit dem RMySQL-Paket beschäftigen, welches für die Verbindung zwischen MySQL-Servern und R sorgt.
Zuerst wird das Paket RmySQL installiert.
Die dbConnect-Funktion verbindet R mit der Datenbank. Folgende Informationen braucht die Funktion:

- user: Benutzername unter dem die mySQL Datenbank erstellt wird.
- password: Ist das Passwort für die Datenbank.
- dbname: Ist der Name der Datenbank, mit der kommuniziert werden soll
- host: Ist der Name des PCs, auf dem MySQL installiert ist.

R erstellt für die Verbindung ein Objekt. Hier hat es den Namen DataBase.

```
> install.packages("RmySQL")
> library(RMySQL)
```

4 http://enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/daten-wissen/Business-Intelligence/Data-Warehouse



(Abbild aus dem Programm MySQL Workbench 6.3 SE)

```
> DataBase = dbConnect(MySQL(),user = 'root', password =
'123', dbname = 'new_schema_test', host = 'localhost')
```

Ist die Verbindung mit dem Server zustande gekommen, kann man sich die Liste der Tabellen auf dem Server anzeigen lassen. Dies macht die dbListTables-Funktion aus dem mySQL-Paket. Als Parameter wird das Objekt der Verbindung übergeben. Die Tabelle selber lässt sich mit dbReadTable in R anzeigen. Dann kann die Tabelle verändert werden.

```
> dbListTables(DataBase)
[1] "datenpaket1" "datenpaket2"

> dbReadTable(DataBase, "datenpaket1")
   Name Alter     Beruf
1  Lisa    25  Kauffrau
2 Sarah    30  Lehrerin
3 Karin    51 Floristin

> dbReadTable(DataBase, "datenpaket2")
   Name Alter     Beruf
1  Lisa    25  Kauffrau
2 Sarah    30  Lehrerin
3 Karin    51 Floristin
```

Zusammenfassung

R hat alle Voraussetzungen, die man für data cleaning benötigt. Eine Vielzahl von Paketen macht es dem Benutzer möglich auf jedes Problem zu reagieren.R kann in moderne Softwaresysteme integriert werden und so die Datenqualität erhöhen.

Quellenverzeichnis

Edwin de Jong, Mark van der Loo: An introduction to data cleaning with R

Detlef Apel, Wolfgang Behme, Rüdiger Eberlein, Christian Merighi: Datenqualität erfolgreich steuern, Hanser Verlag

Andrie de Vries , Joris Meys: R für Dummis

Janet Valande: PHP und MySQL für Dummis

Bradley Wickham: Tidy Data. In Journal of Statistical Software, August 2014, Volume 59, Issu 10