O'REILLY®

US-Bestseller

R für Data Science

DATEN IMPORTIEREN, BEREINIGEN, UMFORMEN, MODELLIEREN UND VISUALISIEREN

Hadley Wickham &
Garrett Grolemund
Übersetzung von Frank Langenau

R für Data Science

Daten importieren, bereinigen, umformen, modellieren und visualisieren

Hadley Wickham und Garrett Grolemund

Deutsche Übersetzung von Frank Langenau

O'REILLY®

Hadley Wickham & Garrett Grolemund

Lektorat: Alexandra Follenius
Übersetzung: Frank Langeau
Korrektorat: Claudia Lötschert, www.richtiger-text.de
Satz: III-Satz, www.drei-satz.de
Herstellung: Susanne Bröckelmann
Umschlaggestaltung: Michael Oréal, www.oreal.de
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

Bibliografische Information Der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN:
Print 978-3-96009-050-2
PDF 978-3-96010-153-6
ePub 978-3-96010-154-3
mobi 978-3-96010-155-0

Dieses Buch erscheint in Kooperation mit O'Reilly Media, Inc. unter dem Imprint »O'REILLY«. O'REILLY ist ein Markenzeichen und eine eingetragene Marke von O'Reilly Media, Inc. und wird mit Einwilligung des Eigentümers verwendet.

1. Auflage 2018

5 4 3 2 1 0

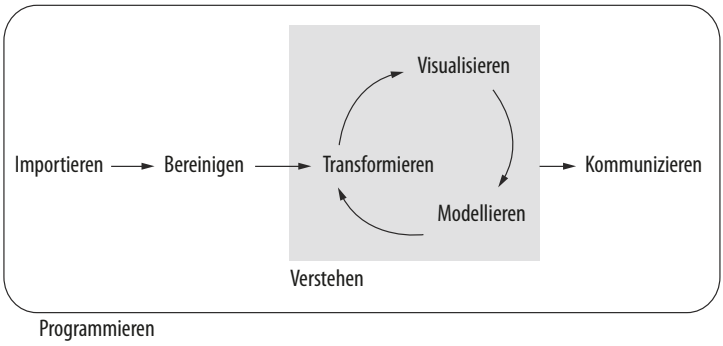
Inhalt

Vorwort

Data Science ist eine spannende Disziplin, mit der Sie Rohdaten in Verständnis, Erkenntnis und Wissen umwandeln können. Das Buch *R für Data Science* soll Ihnen die wichtigsten Tools für Data Science in R nahebringen. Nachdem Sie dieses Buch gelesen haben, verfügen Sie über das Handwerkszeug, um ein breites Spektrum anspruchsvoller Data-Science-Problemstellungen mithilfe der besten Bestandteile von R anzugehen.

Was Sie lernen werden

Data Science ist ein riesiges Feld, und Sie werden es wahrscheinlich nicht meistern, wenn Sie nur ein einziges Buch lesen. Mit diesem Buch können Sie sich eine solide Basis für die wichtigsten Tools erarbeiten. Unser Modell der erforderlichen Tools in einem typischen Data-Science-Projekt sieht etwa wie folgt aus:



Zuerst müssen Sie Daten in R *importieren*. In der Regel heißt das, dass Sie gespeicherte Daten aus einer Datei, Datenbank oder Web-API übernehmen und in einen Dataframe in R laden. Wenn Sie Ihre Daten nicht nach R übernehmen können, lässt sich keine Data Science damit anstellen!

Nachdem Sie die Daten importiert haben, empfiehlt es sich, sie zu *bereinigen* oder *aufzubereiten*. Dabei speichert man die Daten in einer konsistenten Form, die die Semantik des Datensets mit der Art und Weise der Speicherung abstimmt. Kurz

gesagt, wenn Ihre Daten bereinigt sind, ist jede Spalte eine Variable, und jede Zeile ist eine Beobachtung. Das Bereinigen von Daten ist wichtig, weil Sie sich wegen der konsistenten Struktur auf Fragen über die Daten konzentrieren können und nicht damit abmühen müssen, die Daten für verschiedene Funktionen in die richtige Form zu bekommen.

Wenn die Daten bereinigt sind, werden sie oftmals in einem ersten Schritt *transformiert*. Die Transformation schließt das Einengen auf interessierende Beobachtungen ein (wie zum Beispiel alle Personen in einer Stadt oder alle Daten aus dem letzten Jahr), das Erstellen neuer Variablen, die Funktionen von vorhandenen Variablen sind (wie das Berechnen der Geschwindigkeit aus Weg und Zeit), und das Berechnen einer Menge von Zusammenfassungsstatistiken (wie Anzahlen oder Mittelwerte). Bereinigen und Transformieren bezeichnet man zusammengenommen als *Aufbereiten* – im Englischen *Wrangling* genannt.

Nachdem Sie über bereinigte Daten mit den benötigten Variablen verfügen, gibt es zwei Hauptmodule der Wissensgenerierung: Visualisierung und Modellierung. Diese weisen viele Stärken und Schwächen auf, die komplementär zueinander sind, sodass jede reale Analyse diese Schritte mehrfach durchlaufen wird.

Visualisierung ist eine fundamentale menschliche Aktivität. Eine gute Visualisierung zeigt Ihnen Dinge, die Sie nicht erwartet haben, oder wirft neue Fragen über die Daten auf. Außerdem kann eine gute Visualisierung darauf hinweisen, dass Sie die falschen Fragen stellen oder andersartige Daten erfassen müssen. Visualisierungen können Sie überraschen, lassen sich aber nicht besonders gut normieren, weil ein Mensch sie interpretieren muss.

Modelle ergänzen die Visualisierung. Wenn Sie einmal Ihre Fragen ausreichend genau gestellt haben, können Sie sie mithilfe eines Modells beantworten. Modelle sind ein fundamentales mathematisches oder rechentechnisches Werkzeug, sodass sie sich im Allgemeinen gut skalieren lassen. Selbst wenn das nicht zutrifft, ist es normalerweise billiger, mehr Computer zu kaufen als mehr Gehirne! Doch jedes Modell geht von Annahmen aus, und seinem Wesen nach kann ein Modell seine eigenen Annahmen nicht beantworten. Ein Modell kann Sie also grundsätzlich nicht überraschen.

Der letzte Schritt bei Data Science ist die *Kommunikation*, ein absolut entscheidender Teil jedes Datenanalyseprojekts. Es spielt keine Rolle, wie gut Ihre Modelle und die Visualisierung Sie dazu gebracht haben, die Daten zu verstehen, sofern Sie nicht auch Ihre Ergebnisse für andere kommunizieren können.

Den Rahmen für alle diese Tools bildet die *Programmierung*. Als übergreifendes Tool haben Sie mit Programmierung in jedem Teil des Projekts zu tun. Dabei brauchen Sie kein Programmierexperte zu sein, um ein Data Scientist zu sein, doch wenn Sie mehr über Programmierung lernen, zahlt sich das aus, denn als besserer Programmierer können Sie häufig anfallende Aufgaben automatisieren und neue Probleme wesentlich leichter lösen. Diese Tools verwenden Sie in fast

jedem Data-Science-Projekt, doch für die meisten Projekte sind sie nicht ausreichend. Als Faustregel gilt ein 80-20-Verhältnis: Etwa 80% jedes Projekts können Sie angehen mit den Tools, die Sie in diesem Buch kennenlernen, doch um die restlichen 20% zu lösen, brauchen Sie weitere Tools. Das ganze Buch hindurch verweisen wir auf Quellen, wo Sie mehr zu den jeweiligen Themen lernen können.

Wie dieses Buch organisiert ist

Die vorhergehende Beschreibung der Tools der Data Science ist ungefähr in der Reihenfolge angeordnet, in der Sie sie in einer Analyse einsetzen (auch wenn Sie diese Schritte natürlich mehrfach durchlaufen). Unserer Erfahrung nach ist dies aber nicht die beste Methode, sich mit diesen Tools bekanntzumachen:

- Mit Datenerfassung und Bereinigen zu beginnen, ist suboptimal, weil Sie 80% der Zeit mit langweiligen Routinearbeiten zubringen und die anderen 20% der Zeit bizarre und frustrierende Aufgaben lösen müssen. Das ist ein schlechter Ausgangspunkt, um sich ein neues Thema anzueignen! Stattdessen beginnen wir mit Visualisierung und Transformation von Daten, die bereits importiert und bereinigt wurden. Wenn Sie später also Ihre eigenen Daten erfassen und bereinigen, bleibt Ihre Motivation hoch, weil Sie wissen, dass der Aufwand gerechtfertigt ist.
- Manche Themen lassen sich am besten mit anderen Tools erläutern. Zum Beispiel glauben wir, dass es einfacher zu verstehen ist, wie Modelle arbeiten, wenn Sie bereits mit Visualisierung, Bereinigen von Daten und Programmierung vertraut sind.
- Programmiertools an sich sind nicht unbedingt interessant, doch sie erlauben es, beträchtlich anspruchsvollere Probleme anzugehen. Im Mittelteil des Buchs geben wir eine Auswahl von Programmiertools an, und Sie werden dann erfahren, wie sie sich mit den Data-Science-Tools kombinieren lassen, um interessante Modellierungsprobleme in den Griff zu bekommen.

In jedem Kapitel versuchen wir, uns an ein ähnliches Muster zu halten: Wir beginnen mit einigen motivierenden Beispielen, sodass Sie ein Gesamtbild bekommen, und tauchen dann ab in die Details. Jeder Abschnitt des Buchs ist gepaart mit Übungen, an denen Sie Ihre erworbenen Kenntnisse überprüfen können. Es mag verlockend sein, die Übungen zu überspringen, doch es gibt keine bessere Möglichkeit zu lernen, als sich an echten Problemen zu versuchen.

Was Sie hier nicht lernen werden

Es gibt weitere wichtige Themen, die dieses Buch aber nicht behandelt. Unserer Ansicht nach ist es wichtig, unerbittlich auf das Wesentliche konzentriert zu bleiben, sodass Sie möglichst schnell selbst aktiv werden können. Das heißt, dass dieses Buch nicht jedes wichtige Thema behandeln kann.

Big Data

Dieses Buch konzentriert sich stolz auf kleine, speicherresidente Datenmengen. Das ist der richtige Ort für den Einstieg, weil Sie große Datenmengen (sprich: Big Data) erst dann beherrschen, wenn Sie genügend Erfahrungen mit kleineren Datenmengen gesammelt haben. Die Tools, die Sie in diesem Buch kennenlernen, verarbeiten problemlos Hunderte MB Daten, und mit etwas Sorgfalt können Sie sie normalerweise auch für 1 bis 2 GB Daten verwenden. Wenn Sie regelmäßig mit größeren Datenmengen (sagen wir 10 bis 100 GB) arbeiten, sollten Sie mehr über `data.table` lernen (*http://bit.ly/Rdatatable*). Dieses Buch geht nicht auf `data.table` ein, weil es eine recht spartanische Benutzeroberfläche hat, die es schwieriger macht zu lernen, weil es weniger linguistische Anhaltspunkte gibt. Doch wenn Sie mit großen Datenmengen arbeiten, gleicht der Leistungsgewinn den zusätzlich erforderlichen Lernaufwand bei Weitem aus.

Wenn Ihre Daten noch größer als diese sind, sollten Sie sorgfältig untersuchen, ob Ihr Big-Data-Problem eigentlich ein verkapptes Problem mit geringem Datenumfang ist. Während die vollständigen Daten sehr umfangreich sein können, haben die Daten, die für die Beantwortung einer spezifischen Frage erforderlich sind, oftmals nur einen geringen Umfang. Vielleicht finden Sie eine Teilmenge, eine Stichprobe oder eine Zusammenfassung, die in den Hauptspeicher passt und es trotzdem noch erlaubt, die Frage zu beantworten, an der Sie interessiert sind. Die Herausforderung ist hier, den geringst möglichen Datenumfang zu finden, was oftmals eine ganze Menge Iterationen verlangt. Eine andere Möglichkeit ist, dass Ihr Problem mit Big Data eigentlich aus sehr vielen Problemen mit kleinen Datenmengen besteht. Jedes einzelne Problem passt vielleicht in den Hauptspeicher, doch Sie haben Millionen von ihnen. Zum Beispiel könnte es sein, dass Sie ein Modell an jede Person in Ihrer Datenbank anpassen möchten. Die Aufgabe wäre trivial, wenn Sie lediglich zehn oder hundert Personen gespeichert hätten, doch stattdessen haben Sie eine Million. Erfreulicherweise ist jedes einzelne Problem unabhängig von den anderen (eine Einrichtung, die manchmal als hochgradig parallel bezeichnet wird), sodass Sie ein System (wie Hadoop oder Spark) brauchen, mit dem Sie verschiedene Datenmengen an verschiedene Computer zur Verarbeitung schicken können. Nachdem Sie herausgefunden haben, wie die Frage für eine einzelne Teilmenge mithilfe der Tools, die in diesem Buch beschrieben werden, zu beantworten ist, lernen Sie neue Tools wie zum Beispiel `sparklyr`, `rhipe` und `ddr` kennen, um das Problem für die vollständige Datenmenge zu lösen.

Python, Julia und Konsorten

In diesem Buch lernen Sie weder etwas über Python, Julia noch irgendeine andere Programmiersprache, die für Data Science geeignet ist. Das hat nichts damit zu tun, dass diese Tools unserer Ansicht nach schlecht sind. Ganz und gar nicht! Und

in der Praxis verwenden die meisten Data-Science-Teams eine Mischung von Sprachen, oftmals mindestens R und Python.

Allerdings sind wir überzeugt davon, dass es am besten ist, immer nur ein Tool auf einmal beherrschen zu lernen. Sie werden schneller besser, wenn Sie tief in die Materie eindringen, als wenn Sie sich mit Vielem nur oberflächlich beschäftigen. Das heißt nicht, dass Sie ausschließlich eine Sache kennen sollten, sondern lediglich, dass Sie im Allgemeinen schneller lernen, wenn Sie jeweils nur bei einem Thema bleiben. Streben Sie auf jeden Fall danach, in Ihrer Laufbahn Neues zu lernen, doch stellen Sie sicher, dass Sie ein solides Verständnis haben, bevor Sie sich dem nächsten interessanten Komplex zuwenden.

R ist unserer Ansicht nach ein großartiger Ausgangspunkt für Ihre Data-Science-Tour, weil es eine Umgebung ist, die von Grund auf für die Unterstützung von Data Science konzipiert wurde. R ist nicht einfach eine Programmiersprache, sondern eine interaktive Umgebung für die Beschäftigung mit Data Science. Um Interaktion zu unterstützen, ist R eine wesentlich flexiblere Sprache als viele ihrer Peers. Diese Flexibilität hat auch ihre Nachteile, doch der große Vorteil ist, wie leicht sich zugeschnittene Grammatiken für spezifische Teile des Data-Science-Prozesses entwickeln lassen. Diese Minisprachen helfen Ihnen dabei, über Probleme als Data Scientist nachzudenken, während die flüssige Interaktion zwischen Ihrem Gehirn und dem Computer unterstützt wird.

Nicht rechteckige Daten

Dieses Buch konzentriert sich ausschließlich auf »rechteckige« Daten: Sammlungen von Werten, die jeweils mit einer Variablen und einer Beobachtung verbunden sind. Es gibt Unmengen von Datensätzen, die von Haus aus nicht in dieses Paradigma passen – unter anderem Bilder, Klänge, Baumstrukturen und Text. Doch rechteckige Datenstrukturen kommen in der Wissenschaft und der Industrie äußerst häufig vor, und wir glauben, dass sie ein großartiger Ausgangspunkt für Ihre Data-Science-Tour sind.

Hypothesenbestätigung

Die Datenanalyse kann man in zwei Lager einteilen: Hypothesengenerierung und Hypothesenbestätigung (manchmal auch bestätigende Analyse genannt). Der Schwerpunkt dieses Buchs liegt eindeutig auf der Hypothesengenerierung oder Datenexploration. Hier blicken wir intensiv auf die Daten und generieren – in Kombination mit Ihrer Sachkenntnis – viele interessante Hypothesen, die bei der Erklärung helfen, warum sich die Daten gerade auf diese Art und Weise verhalten. Diese Hypothesen bewerten Sie formlos, wobei Sie Ihre Skepsis nutzen, um den Daten in verschiedener Art und Weise verborgene Informationen zu entlocken.

Das Gegenstück zur Hypothesengenerierung ist die Hypothesenbestätigung. Diese ist aus zwei Gründen schwierig:

- Man benötigt ein genaues mathematisches Modell, um falsifizierbare Vorhersagen generieren zu können. Hierzu sind oftmals erhebliche statistische Fachkenntnisse erforderlich.
- Eine Beobachtung lässt sich nur einmal nutzen, um eine Hypothese zu bestätigen. Sobald man sie mehrmals verwendet, kommt man zur explorativen Datenanalyse zurück. Um eine Hypothesenbestätigung durchzuführen, müssen Sie also Ihren Analyseplan »vorab registrieren« (im Voraus ausarbeiten) und nicht von ihm abweichen, selbst wenn Sie bereits die Daten gesehen haben. In Teil IV sprechen wir über einige Strategien, die dieses Vorgehen erleichtern.

Es ist üblich, die Modellierung als Tool für die Hypothesenbestätigung und die Visualisierung als Tool für die Hypothesengenerierung aufzufassen. Doch diese Zweiteilung ist falsch: Modelle dienen oft zur Erkundung, und mit etwas Sorgfalt können Sie Visualisierung für die Bestätigung nutzen. Der Unterschied zeigt sich vor allem darin, wie oft man jede Beobachtung betrachtet: Wenn Sie sie nur einmal ansehen, handelt es sich um eine Bestätigung, und wenn Sie sie mehrfach ansehen, ist es eine Exploration.

Voraussetzungen

Wir haben einige Annahmen darüber getroffen, was Sie bereits wissen, um möglichst gut von diesem Buch profitieren zu können. Allgemein sollten Sie gut mit Zahlenmaterial umgehen können, und es ist hilfreich, wenn Sie bereits etwas Programmiererfahrung mitbringen. Falls Sie noch nie programmiert haben, dürfte Ihnen das Buch *Hands-On Programming with R* von Garrett eine nützliche Begleitung zu diesem Buch sein.

Um den Code in diesem Buch auszuführen, brauchen Sie vier Dinge: R, RStudio, eine Auswahl von R-Paketen – das sogenannte *Tidyverse* – und eine Handvoll anderer Pakete. Pakete sind die grundlegenden Einheiten von reproduzierbarem R-Code. Sie umfassen wiederverwendbare Funktionen, die Dokumentation, die sie beschreibt, und Beispieldaten.

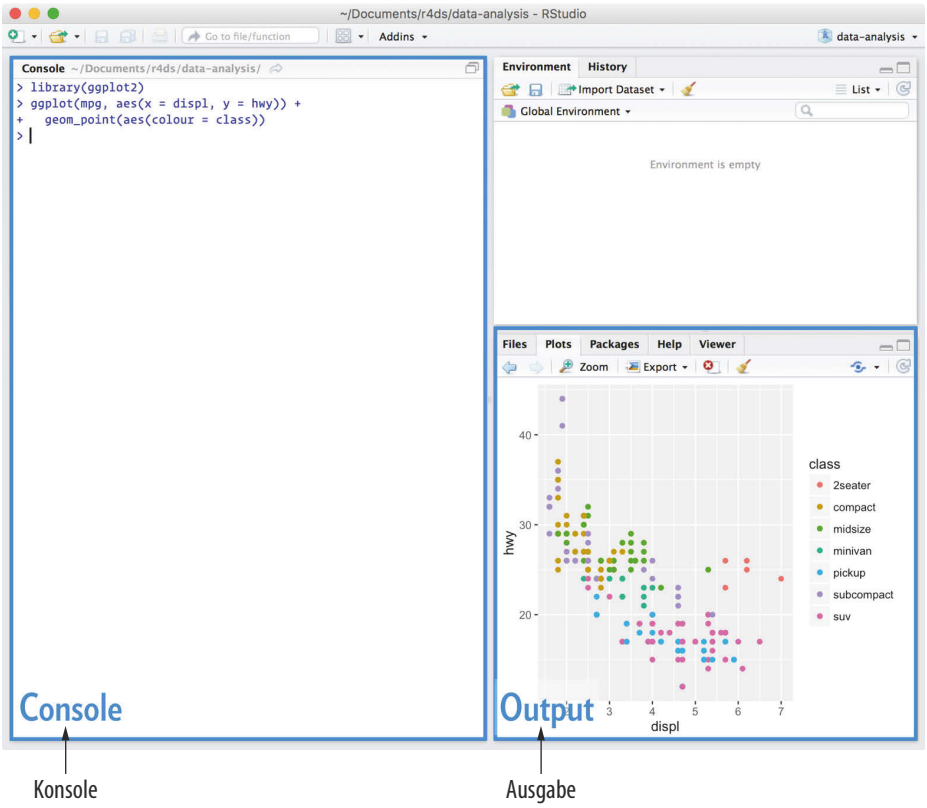
R

Um R herunterzuladen, besuchen Sie die Site von CRAN, was für Comprehensive R Archive Network (ein Netzwerk mit einem umfassenden Archiv von Modulen für die Programmiersprache R) steht. CRAN besteht aus zahlreichen Mirror-Servern, die weltweit verteilt sind und die Aufgabe haben, R und R-Pakete zu verteilen. Versuchen Sie gar nicht erst, einen Mirror in Ihrer Nähe zu suchen. Verwenden Sie stattdessen den Cloud-Mirror *https://cloud.r-project.org*, der Ihnen diese Arbeit abnimmt.

Eine neue Hauptversion von R erscheint einmal im Jahr, und jedes Jahr gibt es zwei bis drei kleinere Versionen. Es empfiehlt sich, die Programme regelmäßig zu aktualisieren. Ein Upgrade kann etwas mühsam sein, vor allem bei Hauptversionen, bei denen es erforderlich ist, alle Pakete erneut zu installieren. Doch ganz unprofessionell ist es, die Upgrades auf die lange Bank zu schieben.

RStudio

RStudio ist eine integrierte Entwicklungsumgebung (Integrated Development Environment, IDE) für die R-Programmierung. Sie können RStudio von *http://www.rstudio.com/download* herunterladen und installieren. Die Software wird mehrmals im Jahr aktualisiert. Ist eine neue Version verfügbar, teilt Ihnen RStudio dies mit. Es empfiehlt sich, die Upgrades regelmäßig zu installieren, um von den neuesten und besten Features zu profitieren. Für dieses Buch ist mindestens die Version RStudio 1.0.0 erforderlich[1]. Wenn Sie RStudio starten, sehen Sie in der Benutzeroberfläche zwei Schlüsselbereiche:



1 Die Übersetzung basiert auf der Version 1.0.143.

Fürs Erste müssen Sie nur wissen, dass Sie R-Code in den Konsolenbereich eintippen und *Eingabe* drücken, um ihn auszuführen. Mehr lernen Sie im weiteren Verlauf dieses Buchs.

Das Tidyverse

Des Weiteren müssen Sie einige R-Pakete installieren. Ein R-*Paket* ist eine Sammlung von Funktionen, Daten und Dokumentationen, die die Fähigkeiten des grundlegenden R erweitern. Pakete zu verwenden, ist der Schlüssel für erfolgreiches Arbeiten mit R. Die meisten Pakete, die Sie in diesem Buch kennenlernen, gehören zum sogenannten Tidyverse. Die Pakete im Tidyverse sind nach einer gemeinsamen Philosophie von Daten und R-Programmierung konzipiert und von Haus aus für eine Zusammenarbeit ausgelegt.

Das vollständige Tidyverse lässt sich mit einer einzigen Codezeile installieren:

```
install.packages("tidyverse")
```

Tippen Sie auf Ihrem Computer diese Codezeile in die Konsole ein und drücken Sie *Eingabe*, um die Anweisung auszuführen. R lädt die Pakete von CRAN herunter und installiert sie auf Ihrem Computer. Sollten Probleme bei der Installation auftreten, prüfen Sie zunächst, ob eine ordnungsgemäße Verbindung zum Internet besteht und dass die Site *https://cloud.r-project.org/* weder durch eine Firewall noch durch einen Proxy blockiert ist.

Die Funktionen, Objekte und Hilfedateien in einem Paket können Sie erst dann verwenden, wenn Sie sie mit `library()` geladen haben. Nachdem ein Paket installiert ist, können Sie es mit der Funktion `library()` laden:

```
library(tidyverse)
#> Loading tidyverse: ggplot2
#> Loading tidyverse: tibble
#> Loading tidyverse: tidyr
#> Loading tidyverse: readr
#> Loading tidyverse: purrr
#> Loading tidyverse: dplyr
#> Conflicts with tidy packages ----------------------------------
#> filter(): dplyr, stats
#> lag():    dplyr, stats
```

Hieraus geht hervor, dass Tidyverse die Pakete **ggplot2**, **tibble**, **tidyr**, **readr**, **purrr** und **dplyr** lädt. Diese Pakete gelten als *Kern* des Tidyverse, weil Sie sie in fast jeder Analyse verwenden.

Pakete im Tidyverse ändern sich ziemlich häufig. Mit dem Befehl `tidyverse_update()` können Sie sich die verfügbaren Updates ansehen und optional installieren.

Andere Pakete

Es gibt viele andere ausgezeichnete Pakete, die nicht Bestandteil des Tidyverse sind, weil sie Probleme in einem anderen Bereich lösen oder für einen anderen Satz von zugrunde liegenden Prinzipien konzipiert sind. Das macht sie weder besser noch schlechter, nur anders. Mit anderen Worten ist die Ergänzung zum Tidyverse nicht das Messyverse, sondern viele andere Universes von zusammenhängenden Paketen. Wenn Sie mehr Data-Science-Projekte mit R angehen, lernen Sie neue Pakete kennen und neue Methoden, über die Daten nachzudenken.

In diesem Buch verwenden wir drei Datenpakete, die nicht im Tidyverse enthalten sind:

```
install.packages(c("nycflights13", "gapminder", "Lahman"))
```

Diese Pakete stellen Daten zu Fluglinien, Weltentwicklung und Baseball bereit, an denen wir wichtige Konzepte der Data Science veranschaulichen.

R-Code ausführen

Der letzte Abschnitt hat einige Beispiele für die Ausführung von R-Code gezeigt. Der Code im Buch sieht folgendermaßen aus:

```
1 + 2
#> [1] 3
```

Wenn Sie denselben Code in der Konsole auf Ihrem Computer ausführen, sieht er so aus:

```
> 1 + 2
[1] 3
```

Die beiden Codefragmente unterscheiden sich vor allem in zwei Punkten. In Ihrer Konsole tippen Sie den Code nach dem Zeichen > ein. Dieses Zeichen ist das sogenannte *Prompt*, das wir aber im Buch nicht wiedergeben. Die Ausgaben haben wir im Buch mit #> auskommentiert, während sie in Ihrer Konsole direkt nach dem Code erscheinen. Wenn Sie mit einer elektronischen Version des Buchs arbeiten, können Sie aufgrund dieser beiden Unterschiede den Code ganz einfach aus dem Buch in die Konsole kopieren.

Das gesamte Buch hindurch halten wir uns an einen einheitlichen Satz von Konventionen in Bezug auf den Code:

- Funktionen werden in einer Listingschrift wiedergegeben und mit Klammern versehen, wie zum Beispiel `sum()` oder `mean()`.
- Andere R-Objekte (wie zum Beispiel Daten oder Funktionsargumente) stehen in Listingschrift ohne die nachfolgenden Klammern, wie zum Beispiel `flights` oder `x`.

- Wenn wir klar machen wollen, aus welchem Paket ein Objekt stammt, schreiben wir den Paketnamen gefolgt von zwei Doppelpunkten, zum Beispiel `dplyr::mutate()` oder `nycflights13::flights`. Dies ist ebenfalls gültiger R-Code.
- Wird nur auf den Paketnamen Bezug genommen, erscheint der Name wie bei **ggplot2** in Fettschrift.

Hilfe erhalten und mehr lernen

Dieses Buch ist keine Insel, es gibt nicht nur eine einzige Quelle, die es Ihnen erlaubt, R zu beherrschen. Wenn Sie die im Buch beschriebenen Techniken auf Ihre eigenen Daten anwenden, werden schnell Fragen auftauchen, die ich in diesem Buch nicht beantwortet habe. Dieser Abschnitt gibt einige Tipps, wie Sie sich Hilfe holen und wie Sie mehr lernen können.

Falls Sie irgendwo steckenbleiben, beginnen Sie mit Google. Normalerweise genügt es, in einer Suche ein »R« hinzuzufügen, um sie auf relevante Ergebnisse einzuschränken: Wenn die Suche nichts Richtiges gebracht hat, bedeutet das oftmals, dass keine R-spezifischen Ergebnisse verfügbar sind. Google ist insbesondere nützlich für Fehlermeldungen. Wenn Sie eine Fehlermeldung erhalten und diese nicht interpretieren können, sollten Sie diese Meldung per Google suchen! Die Chancen stehen gut, dass jemand in der Vergangenheit über die gleiche Fehlermeldung gestolpert ist und es irgendwo im Web Hilfe gibt. (Wenn die Fehlermeldung nicht in Englisch erscheint, führen Sie `Sys.setenv(LANGUAGE = "en")` aus und starten den Code noch einmal; die Wahrscheinlichkeit, Hilfe für englische Fehlermeldungen zu finden, ist wesentlich höher.)

Probieren Sie Stackoverflow (*http://stackoverflow.com*), wenn Ihnen Google nicht helfen konnte. Investieren Sie zunächst etwas Zeit, um nach einer vorhandenen Antwort zu suchen; schließen Sie `[R]` in Ihre Anfrage ein, um die Suche auf Fragen und Antworten zu beschränken, die mit R zu tun haben. Wenn Sie nichts Brauchbares finden, bereiten Sie ein minimales und reproduzierbares Beispiel oder **Reprex** vor. Ein gutes Reprex erleichtert es anderen Leuten, Ihnen zu helfen, und oftmals finden Sie die Ursache für das Problem selbst heraus, während Sie das Beispiel formulieren.

Drei Dinge müssen Sie in Ihrem Beispiel angeben, damit es reproduzierbar wird: erforderliche Pakete, Daten und Code:

- *Pakete* sollten am Anfang des Skripts geladen werden, sodass man leicht sehen kann, welche Pakete für das Beispiel erforderlich sind. Das ist auch ein passender Zeitpunkt, um zu überprüfen, ob man die neueste Version jedes Pakets verwendet; es ist denkbar, dass Sie einen Bug entdeckt haben, der inzwischen

beseitigt ist, seit Sie das Paket installiert haben. Für Pakete im Tidyverse lässt sich das ganz einfach mit `tidyverse_update()` prüfen.

- Um *Daten* in eine Anfrage einzubinden, ist es am einfachsten, mit `dput()` den R-Code zu generieren, der sie neu erzeugt. Um zum Beispiel den Datensatz `mtcars` in R erneut zu erzeugen, führe ich die folgenden Schritte aus:
  1. In R `dput(mtcars)` ausführen.
  2. Die Ausgabe kopieren.
  3. In meinem reproduzierbaren Skript `mtcars <-` eintippen und dann einfügen.

  Probieren Sie, die kleinste Teilmenge Ihrer Daten zu finden, bei der das Problem immer noch auftritt.
- Investieren Sie etwas Zeit, um sicherzustellen, dass Ihr *Code* für andere Personen leicht zu lesen ist:
  - Achten Sie darauf, Leerzeichen zu verwenden und prägnante, aber informative Variablennamen zu vergeben.
  - Weisen Sie mit Kommentaren auf die Stelle hin, wo Ihr Problem auftritt.
  - Entfernen Sie möglichst alles, was nichts mit dem Problem zu tun hat.

  Je kürzer der Code ist, desto verständlicher ist er und desto einfacher lässt er sich korrigieren.

Vergewissern Sie sich abschließend, dass Sie tatsächlich ein reproduzierbares Beispiel erstellt haben, indem Sie eine neue R-Sitzung starten, das Skript kopieren und in die Sitzung einfügen.

Des Weiteren sollten Sie sich darauf vorbereiten, Probleme zu lösen, bevor sie auftreten. Wenn Sie jeden Tag ein wenig Zeit investieren, um R zu lernen, macht sich das auf lange Sicht bezahlt. So können Sie unter anderem im RStudio-Blog (*https://blog.rstudio.org*) verfolgen, was Hadley, Garrett und andere Leute bei RStudio tun. Hier finden Sie Ankündigungen zu neuen Paketen, neue IDE-Features und persönliche Schulungen. Außerdem können Sie Hadley (@hadleywickham (*https://twitter.com/hadleywickham*)) oder Garrett (@statgarrett (*https://twitter.com/statgarrett*) auf Twitter oder @rstudiotips (*https://twitter.com/rstudiotips*) folgen, um sich über neue Features in der IDE zu informieren.

Wenn Sie sich umfassend in der R-Community informieren wollen, sollten Sie *http://www.r-bloggers.com* lesen: Hier werden mehr als 500 Blogs über R aus der ganzen Welt zusammengefasst. Als aktiver Twitter-Benutzer folgen Sie dem Hashtag *#rstats*. Twitter ist eines der wichtigsten Tools, die Hadley nutzt, um mit neuen Entwicklungen in der Community Schritt zu halten.

Danksagungen

Dieses Buch ist nicht einfach das Produkt von Hadley und Garrett, sondern das Ergebnis vieler Konversationen (persönlich und online), die wir mit unzähligen Leuten in der R-Community geführt haben. Es gibt einige Personen, denen wir besonders danken möchten, weil sie viele Stunden damit zugebracht haben, unsere einfältigen Fragen zu beantworten und uns zu helfen, besser über Data Science nachzudenken:

- Jenny Bryan und Lionel Henry für viele hilfreiche Diskussionen in Bezug auf das Arbeiten mit Listen und Listenspalten.
- Die drei Kapitel zum Workflow wurden (mit Genehmigung) adaptiert aus »R basics, workspace and working directory, RStudio projects« (*http://bit.ly/Rbasicsworkflow*) von Jenny Bryan.
- Genevera Allen für Diskussionen über Modelle, Modellierung, die statistische Lernperspektive und den Unterschied zwischen Hypothesengenerierung und Hypothesenbestätigung.
- Yihui Xie für seine Arbeit zum Paket **bookdown** (*https://github.com/rstudio/bookdown*) und für sein unermüdliches Reagieren auf meine Feature-Anfragen.
- Bill Behrman dafür, dass er das gesamte Buch aufmerksam gelesen und in seinem Data-Science-Kurs in Stanford getestet hat.
- Der #rstats-Twitter-Community, die alle Entwürfe der Kapitel rezensiert und tonnenweise nützliches Feedback zurückgeliefert hat.
- Tal Galili für die Erweiterung seines Pakets **dendextend**, um einen Abschnitt zum Clustern zu untermauern, der es in den endgültigen Entwurf nicht geschafft hat.

Dieses Buch wurde öffentlich geschrieben und viele Leute haben mit Pull-Requests dazu beigetragen, kleinere Probleme zu beseitigen. Ein besonderer Dank geht an alle, die über GitHub mitgewirkt haben (in alphabetischer Reihenfolge genannt): adi pradhan, Ahmed ElGabbas, Ajay Deonarine, @Alex, Andrew Landgraf, @batpigandme, @behrman, Ben Marwick, Bill Behrman, Brandon Greenwell, Brett Klamer, Christian G. Warden, Christian Mongeau, Colin Gillespie, Cooper Morris, Curtis Alexander, Daniel Gromer, David Clark, Derwin McGeary, Devin Pastoor, Dylan Cashman, Earl Brown, Eric Watt, Etienne B. Racine, Flemming Villalona, Gregory Jefferis, @harrismcgehee, Hengni Cai, Ian Lyttle, Ian Sealy, Jakub Nowosad, Jennifer (Jenny) Bryan, @jennybc, Jeroen Janssens, Jim Hester, @jjchern, Joanne Jang, John Sears, Jon Calder, Jonathan Page, @jonathanflint, Julia Stewart Lowndes, Julian During, Justinas Petuchovas, Kara Woo, @kdpsingh, Kenny Darrell, Kirill Sevastyanenko, @koalabearski, @KyleHumphrey, Lawrence Wu, Matthew Sedaghatfar, Mine Cetinkaya-Rundel, @MJMarshall, Mustafa Ascha, @nate-d-olson, Nelson Areal, Nick Clark, @nickelas, @nwaff, @OaCantona,

Patrick Kennedy, Peter Hurford, Rademeyer Vermaak, Radu Grosu, @rlzijdeman, Robert Schuessler, @robinlovelace, @robinsones, S'busiso Mkhondwane, @seamus-mckinsey, @seanpwilliams, Shannon Ellis, @shoili, @sibusiso16, @spirgel, Steve Mortimer, @svenski, Terence Teo, Thomas Klebel, TJ Mahr, Tom Prior, Will Beasley, Yihui Xie.

Onlineversion

Unter *http://r4ds.had.co.nz* ist eine Onlineversion dieses Buchs (in Englisch) verfügbar. Dort finden Sie auch Ergänzungen, Änderungen und Erweiterungen, die zwischen den gedruckten Ausgaben des Buchs vorgenommen wurden. Den Quellcode für das Buch finden Sie unter *https://github.com/hadley/r4ds*. Das Buch wird von *https://bookdown.org* unterstützt, was es erleichtert, R-Markdown-Dateien in HTML, PDF und EPUB zu konvertieren.

Dieses Buch wurde wie folgt erstellt:

```
devtools::session_info(c("tidyverse"))
#> Session info -------------------------------------------------
#>  setting  value
#>  version  R version 3.3.1 (2016-06-21)
#>  system   x86_64, darwin13.4.0
#>  ui       X11
#>  language (EN)
#>  collate  en_US.UTF-8
#>  tz       America/Los_Angeles
#>  date     2016-10-10
#> Packages -----------------------------------------------------
#>  package      * version    date       source
#>  assertthat     0.1        2013-12-06 CRAN (R 3.3.0)
#>  BH             1.60.0-2   2016-05-07 CRAN (R 3.3.0)
#>  broom          0.4.1      2016-06-24 CRAN (R 3.3.0)
#>  colorspace     1.2-6      2015-03-11 CRAN (R 3.3.0)
#>  curl           2.1        2016-09-22 CRAN (R 3.3.0)
#>  DBI            0.5-1      2016-09-10 CRAN (R 3.3.0)
#>  dichromat      2.0-0      2013-01-24 CRAN (R 3.3.0)
#>  digest         0.6.10     2016-08-02 CRAN (R 3.3.0)
#>  dplyr        * 0.5.0      2016-06-24 CRAN (R 3.3.0)
#>  forcats        0.1.1      2016-09-16 CRAN (R 3.3.0)
#>  foreign        0.8-67     2016-09-13 CRAN (R 3.3.0)
#>  ggplot2      * 2.1.0.9001 2016-10-06 local
#>  gtable         0.2.0      2016-02-26 CRAN (R 3.3.0)
#>  haven          1.0.0      2016-09-30 local
#>  hms            0.2-1      2016-07-28 CRAN (R 3.3.1)
#>  httr           1.2.1      2016-07-03 cran (@1.2.1)
#>  jsonlite       1.1        2016-09-14 CRAN (R 3.3.0)
#>  labeling       0.3        2014-08-23 CRAN (R 3.3.0)
#>  lattice        0.20-34    2016-09-06 CRAN (R 3.3.0)
#>  lazyeval       0.2.0      2016-06-12 CRAN (R 3.3.0)
#>  lubridate      1.6.0      2016-09-13 CRAN (R 3.3.0)
#>  magrittr       1.5        2014-11-22 CRAN (R 3.3.0)
```

```
#>  MASS           7.3-45     2016-04-21 CRAN (R 3.3.1)
#>  mime           0.5        2016-07-07 cran (@0.5)
#>  mnormt         1.5-4      2016-03-09 CRAN (R 3.3.0)
#>  modelr         0.1.0      2016-08-31 CRAN (R 3.3.0)
#>  munsell        0.4.3      2016-02-13 CRAN (R 3.3.0)
#>  nlme           3.1-128    2016-05-10 CRAN (R 3.3.1)
#>  openssl        0.9.4      2016-05-25 cran (@0.9.4)
#>  plyr           1.8.4      2016-06-08 cran (@1.8.4)
#>  psych          1.6.9      2016-09-17 CRAN (R 3.3.0)
#>  purrr        * 0.2.2      2016-06-18 CRAN (R 3.3.0)
#>  R6             2.1.3      2016-08-19 CRAN (R 3.3.0)
#>  RColorBrewer   1.1-2      2014-12-07 CRAN (R 3.3.0)
#>  Rcpp           0.12.7     2016-09-05 CRAN (R 3.3.0)
#>  readr        * 1.0.0      2016-08-03 CRAN (R 3.3.0)
#>  readxl         0.1.1      2016-03-28 CRAN (R 3.3.0)
#>  reshape2       1.4.1      2014-12-06 CRAN (R 3.3.0)
#>  rvest          0.3.2      2016-06-17 CRAN (R 3.3.0)
#>  scales         0.4.0.9003 2016-10-06 local
#>  selectr        0.3-0      2016-08-30 CRAN (R 3.3.0)
#>  stringi        1.1.2      2016-10-01 CRAN (R 3.3.1)
#>  stringr        1.1.0      2016-08-19 cran (@1.1.0)
#>  tibble       * 1.2        2016-08-26 CRAN (R 3.3.0)
#>  tidyr        * 0.6.0      2016-08-12 CRAN (R 3.3.0)
#>  tidyverse    * 1.0.0      2016-09-09 CRAN (R 3.3.0)
#>  xml2           1.0.0.9001 2016-09-30 local
```

Konventionen in diesem Buch

In diesem Buch werden folgende typografische Konventionen verwendet:

Kursiv

: Kennzeichnet neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateierweiterungen.

Fett

: Kennzeichnet die Namen von R-Paketen.

`Schreibmaschinenschrift`

: Wird verwendet für Programmlistings und innerhalb von Absätzen, um auf Programmelemente zu verweisen, zum Beispiel Namen von Variablen und Funktionen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter.

`Schreibmaschinenschrift fett`

: Zeigt Befehle oder anderen Text, den der Benutzer buchstäblich so eingeben soll.

`Schreibmaschinenschrift kursiv`

Zeigt Text, der durch Werte ersetzt werden soll, die der Benutzer eingibt oder die sich aus dem Kontext ergeben.

Dieses Symbol kennzeichnet einen Tipp oder Vorschlag.

Die Codebeispiele verwenden

Den Quellcode können Sie von *https://github.com/hadley/r4ds* herunterladen.

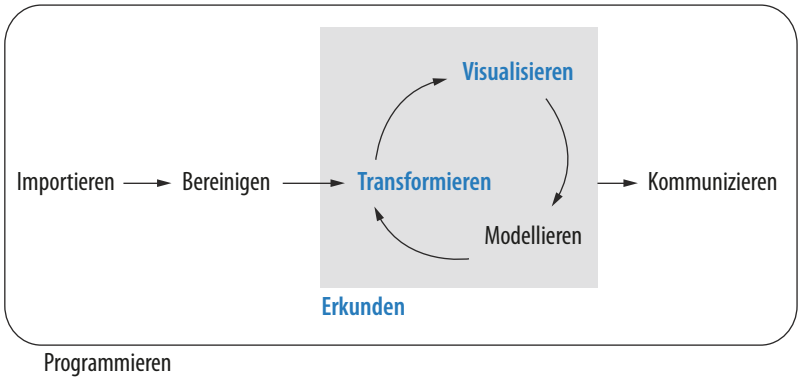
Dieses Buch soll Ihnen bei Ihrer Arbeit helfen. Im Allgemeinen dürfen Sie den Code, der in diesem Buch angegeben ist, in Ihren Programmen und Dokumentationen verwenden. Sie brauchen uns in dieser Hinsicht nicht um Erlaubnis zu fragen, es sei denn, Sie reproduzieren einen erheblichen Teil des Codes. Wenn Sie zum Beispiel ein Programm schreiben, das mehrere Codeblöcke aus diesem Buch enthält, brauchen Sie keine Erlaubnis. Wollen Sie dagegen eine CD-ROM mit Beispielen von O'Reilly-Büchern verkaufen oder vertreiben, brauchen Sie die entsprechenden Genehmigungen. Wenn Sie auf eine Frage antworten und dabei dieses Buch zitieren und Beispielcode angeben, ist keine Erlaubnis notwendig. Möchten Sie aber erhebliche Teile des Beispielcodes aus diesem Buch in die Dokumentation Ihres Produkts einfügen, müssen Sie eine Genehmigung einholen.

Wir schätzen eine Literaturangabe, fordern sie aber nicht. Normalerweise enthält eine Literaturangabe Titel, Autor, Herausgeber und ISBN, zum Beispiel: »*R für Data Science* von Hadley Wickham und Garrett Grolemund (O'Reilly). Copyright 2017 Garrett Grolemund, Hadley Wickham, 978-3-96009-050-2.«

Sollten Sie im Zweifel sein, ob die Verwendung der Codebeispiele außerhalb der fairen Verwendung oder der oben angegebenen Erlaubnis liegt, können Sie uns gern unter *kontakt@oreilly.de* kontaktieren.

TEIL I
Erkunden

Ziel des ersten Teils dieses Buchs ist es, mit den grundlegenden Tools der *explorativen Datenanalyse* möglichst schnell auf Touren zu kommen. Das Wesen der explorativen Datenanalyse besteht darin, Daten zu begutachten, schnelle Hypothesen zu generieren, diese ad hoc zu testen und diese Schritte immerfort zu wiederholen. Ziel der explorativen Datenanalyse ist es, viele aussichtsreiche Annahmen zu generieren, die man später in größerer Tiefe untersuchen kann.



In diesem Teil des Buchs lernen Sie einige nützliche Tools kennen, die sich unmittelbar bezahlt machen:

- Visualisierung ist ein hervorragender Ausgangspunkt in Bezug auf die R-Programmierung, weil der Nutzen klar auf der Hand liegt: Es lassen sich ansprechende und informative Diagramme erstellen, die beim Verständnis der Daten hilfreich sind. In Kapitel 1 steigen Sie ein in die Visualisierung. Dabei lernen Sie die grundlegende Struktur eines **ggplot2**-Diagramms und leistungsfähige Techniken kennen, um Daten in Diagramme zu überführen.
- Da Visualisierung allein in der Regel nicht genügt, macht Sie Kapitel 3 vertraut mit den Schlüsselbegriffen, mit denen Sie wichtige Variablen auswählen,

Schlüsselbeobachtungen ausfiltern, neue Variablen erzeugen und Zusammenfassungen berechnen können.

- Schließlich kombinieren Sie in Kapitel 5 die Visualisierung und Transformation mit Ihrer Neugier und Skepsis, um interessante Fragen über Daten zu stellen und zu beantworten.

Modellierung ist ein wichtiger Teil des Sondierungsprozesses, doch Sie besitzen noch nicht die Fähigkeiten, um sie effektiv zu lernen oder anzuwenden. Wir kommen darauf in Teil IV zurück, nachdem Sie mit mehr Tools zur Datenaufbereitung und Programmierung ausgestattet sind.

In diese drei Kapitel, die Ihnen die Tools der Datenanalyse näherbringen, sind drei Kapitel eingeschachtelt, die sich auf Ihren R-Workflow konzentrieren. In den Kapiteln 2, 4 und 6 lernen Sie bewährte Praktiken kennen, um R-Code zu schreiben und zu organisieren. Damit werden Sie auf lange Sicht erfolgreich bleiben, weil Sie die Tools an die Hand bekommen, um organisiert zu bleiben, wenn Sie reale Projekte angehen müssen.

KAPITEL 1

Datenvisualisierung mit ggplot2

Einführung

> *The simple graph has brought more information to the data analyst's mind than any other device.*
>
> *– John Tukey*

In diesem Kapitel erfahren Sie, wie Sie Ihre Daten mit **ggplot2** visualisieren. R verfügt über mehrere Systeme, um Diagramme zu erzeugen, doch **ggplot2** ist eines der elegantesten und flexibelsten. Das Paket **ggplot2** implementiert die *Grammar of Graphics*, ein einheitliches System, um Grafiken zu beschreiben und zu erstellen. Mit **ggplot2** beschleunigen Sie Ihre Arbeit – Sie brauchen nur ein System zu lernen und können es in vielen Situationen anwenden.

Wenn Sie mehr über das theoretische Fundament von **ggplot2** erfahren möchten, sollten Sie sich »A Layered Grammar of Graphics« (*http://vita.had.co.nz/papers/layered-grammar.pdf*) zu Gemüte führen.

Voraussetzungen

Im Mittelpunkt dieses Kapitels steht **ggplot2**, eine Kernkomponente des Tidyverse. Um auf die Datensets, Hilfeseiten und Funktionen, die wir in diesem Kapitel verwenden, zugreifen zu können, laden Sie das Tidyverse mit folgendem Code:

```
library(tidyverse)
#> Loading tidyverse: ggplot2
#> Loading tidyverse: tibble
#> Loading tidyverse: tidyr
#> Loading tidyverse: readr
#> Loading tidyverse: purrr
#> Loading tidyverse: dplyr
#> Conflicts with tidy packages ----------------------------------
#> filter(): dplyr, stats
#> lag():    dplyr, stats
```

Diese eine Codezeile lädt den Kern des Tidyverse, das heißt Pakete, die Sie in fast jeder Datenanalyse verwenden werden. Außerdem erfahren Sie hier, welche Funktionen des Tidyverse mit Funktionen in Basis-R (oder anderen Paketen, die Sie vielleicht geladen haben) kollidieren könnten.

Wenn Sie diesen Code ausführen und die Fehlermeldung »there is no package called 'tidyverse'« (Kein Paket mit dem Namen `tidyverse` vorhanden) ernten, müssen Sie es zuerst installieren und `library()` noch einmal ausführen:

```
install.packages("tidyverse")
library(tidyverse)
```

Ein Paket müssen Sie nur einmal installieren, doch Sie müssen es jedes Mal erneut laden, wenn Sie eine neue Sitzung beginnen.

Wenn Sie ausdrücklich angeben müssen, woher eine Funktion (oder ein Datenset) stammt, verwenden Sie das spezielle Format `package::function()`. So sagen Sie mit `ggplot2::ggplot()` explizit, dass die Funktion `ggplot()` aus dem Paket **ggplot2** verwendet wird.

Erste Schritte

Anhand der ersten Grafik soll nun eine Frage beantwortet werden: Verbrauchen Autos mit großen Motoren mehr Sprit als Autos mit kleinen Motoren? Wahrscheinlich kennen Sie schon die Antwort, doch versuchen Sie einmal, Ihre Antwort genau zu formulieren. Wie sieht die Beziehung zwischen Hubraum und Kraftstoffeffizienz aus? Ist sie positiv? Negativ? Linear? Nichtlinear?

Der Dataframe mpg

Ihre Antwort können Sie mit dem Dataframe `mpg` testen, der Bestandteil von **ggplot2** ist (aka `ggplot2::mpg`). Ein Dataframe ist eine matrixförmige Anordnung von Variablen (in den Spalten) und Beobachtungen (in den Zeilen). Der Dataframe `mpg` enthält Beobachtungen, die die US-amerikanische Umweltschutzbehörde zu 38 Automodellen erhoben hat:

```
mpg
#> # A tibble: 234 × 11
#>   manufacturer model displ  year   cyl      trans   drv
#>          <chr> <chr> <dbl> <int> <int>      <chr> <chr>
#> 1         audi    a4   1.8  1999     4   auto(l5)     f
#> 2         audi    a4   1.8  1999     4 manual(m5)     f
#> 3         audi    a4   2.0  2008     4 manual(m6)     f
#> 4         audi    a4   2.0  2008     4   auto(av)     f
#> 5         audi    a4   2.8  1999     6   auto(l5)     f
#> 6         audi    a4   2.8  1999     6 manual(m5)     f
#> # ... with 228 more rows, and 4 more variables:
#> # cty <int>, hwy <int>, fl <chr>, class <chr>
```

Zu den Variablen in mpg gehören:

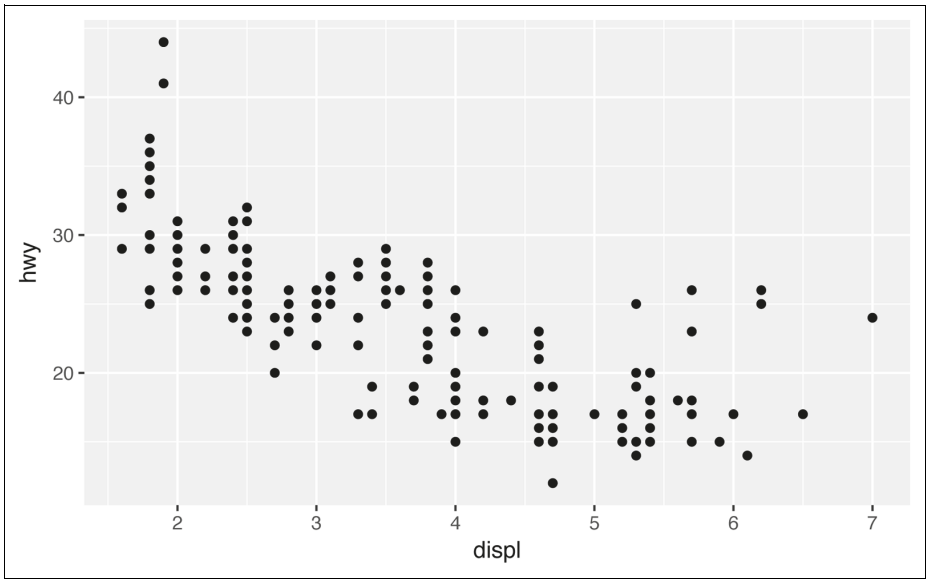
- displ, der Hubraum eines Autos in Litern.
- hwy, die Kraftstoffeffizienz eines Autos auf der Autobahn in Meilen pro Gallone (mpg, daher der Name des Dataframes). Ein Auto mit geringer Kraftstoffeffizienz verbraucht mehr Kraftstoff als ein Auto mit hoher Kraftstoffeffizienz, wenn beide die gleiche Strecke zurücklegen.

Um mehr über mpg zu erfahren, können Sie die dazugehörige Hilfeseite mit dem Befehl ?mpg öffnen.

Einen ggplot erzeugen

Mit dem folgenden Code können Sie mpg als Diagramm darstellen, wobei displ auf der x-Achse und hwy auf der y-Achse aufgetragen werden:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```



Das Diagramm zeigt eine negative Beziehung zwischen Hubraum (displ) und Kraftstoffeffizienz (hwy). Mit anderen Worten: Autos mit großen Motoren verbrauchen mehr Kraftstoff. Hat dieses Ergebnis Ihre Hypothese über Kraftstoffeffizienz und Hubraum bestätigt oder widerlegt?

Mit **ggplot2** beginnen Sie ein Diagramm mit der Funktion ggplot(). Die Funktion ggplot() erzeugt ein Koordinatensystem, dem Sie Ebenen hinzufügen können. Das erste Argument von ggplot() ist das Datenset, das im Diagramm zu verwen-

den ist. So erzeugt `ggplot(data = mpg)` ein leeres Diagramm, doch das nutzt uns gar nichts, und ich verzichte hier auf eine Abbildung.

Das Diagramm vervollständigen Sie, indem Sie eine oder mehrere Ebenen zu `ggplot()` hinzufügen. Die Funktion `geom_point()` fügt eine Ebene mit Punkten in das Diagramm ein, was ein Streudiagramm ergibt. Das Paket **ggplot2** bringt viele geometrische Funktionen mit, die jeweils einen anderen Ebenentyp zu einem Diagramm hinzufügen. Im Verlauf dieses Kapitels lernen Sie noch eine ganze Menge solcher Funktionen kennen.

Jede geometrische Funktion in **ggplot2** übernimmt ein Argument `mapping`. Es definiert, wie die Variablen Ihres Datensets auf visuelle Eigenschaften abgebildet werden. Das Argument `mapping` wird immer zusammen mit `aes()` angegeben. Die Argumente `x` und `y` von `aes()` legen fest, welche Variablen auf die x- und y-Achsen abzubilden sind. Das Paket **ggplot2** sucht nach der zugeordneten Variablen im Argument `data`, in diesem Fall `mpg`.

Eine Grafikvorlage

Diesen Code wollen wir in eine wiederverwendbare Vorlage – auch Template genannt – überführen, um Grafiken mit **ggplot2** zu erstellen. Möchten Sie dann eine Grafik erzeugen, ersetzen Sie die in spitze Klammern gefassten Abschnitte im folgenden Code durch ein Datenset, eine geometrische Funktion bzw. eine Auflistung von Zuordnungen:

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

Der Rest dieses Kapitels zeigt, wie Sie diese Vorlage fertigstellen und erweitern, um verschiedene Arten von Diagrammen zu erzeugen. Den Anfang macht die `<MAPPINGS>`-Komponente.

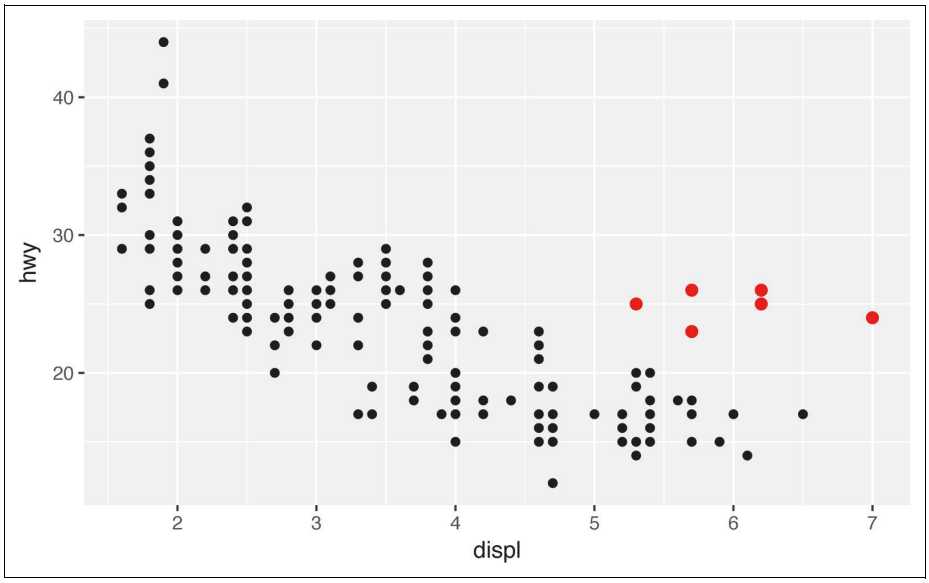
Übungen

1. Führen Sie `ggplot(data = mpg)` aus. Was sehen Sie?
2. Wie viele Zeilen sind in `mtcars` enthalten? Wie viele Spalten?
3. Was beschreibt die Variable `drv`? Lesen Sie die Hilfe für `?mpg`, um es herauszufinden.
4. Erstellen Sie ein Streudiagramm von `hwy` über `cyl`.
5. Was passiert, wenn Sie ein Streudiagramm von `class` über `drv` erstellen? Warum ist das Diagramm nicht aussagekräftig?

Zuordnungen visueller Eigenschaften

> *The greatest value of a picture is when it forces us to notice what we never expected to see.*
>
> *– John Tukey*

Im folgenden Diagramm scheint eine Gruppe von Punkten (die rot hervorgehobenen) aus dem linearen Trend herauszufallen. Diese Autos haben einen höheren Kraftstoffverbrauch, als Sie möglicherweise erwarten. Wie können Sie das für diese Autos erklären?



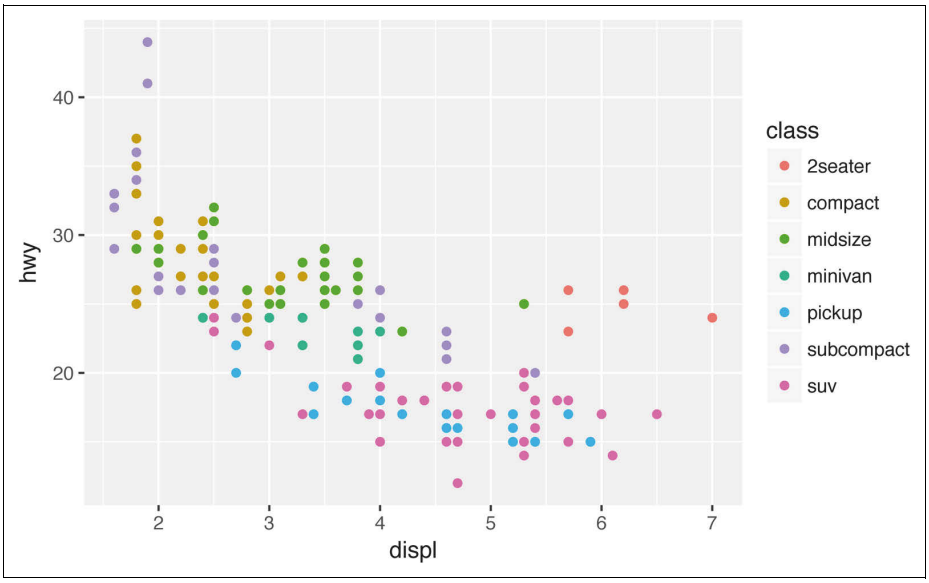
Als Hypothese nehmen wir an, dass es sich um Hybridfahrzeuge handelt. Um diese Hypothese zu testen, sehen wir uns den `class`-Wert für jedes Auto an. Die Variable `class` des Datensets `mpg` klassifiziert die Autos in Gruppen wie zum Beispiel kompakt, mittelgroß und SUV. Wenn die abseits gelegenen Punkte für Hybridfahrzeuge stehen, sollten diese als Kleinwagen oder vielleicht als Kleinstwagen klassifiziert werden (wobei zu bedenken ist, dass die Daten erfasst wurden, bevor Hybrid-Lkw und -SUVs populär wurden).

Nun können Sie eine dritte Variable, wie `class`, zu einem zweidimensionalen Streudiagramm hinzufügen, indem Sie sie einer *visuellen Eigenschaft* (im Sprachgebrauch von R als *aesthetic* bezeichnet) von Objekten in Ihrem Diagramm zuordnen. Zu den visuellen Eigenschaften gehören unter anderem die Größe, die Form und die Farbe der Punkte. Einen Punkt (wie den als Nächstes gezeigten) können Sie auf verschiedene Arten anzeigen, indem Sie die Werte seiner visuellen Eigenschaften ändern. Da wir mit dem Begriff »Wert« bereits Daten beschreiben, neh-

men wir das Wort »Ebene«, um visuelle Eigenschaften zu beschreiben. Hier ändern wir die Ebenen von Größe, Form und Farbe eines Punkts, um den Punkt klein, dreieckig oder blau zu machen:

Informationen über Ihre Daten können Sie vermitteln, indem Sie die visuellen Eigenschaften in Ihrem Diagramm auf Variablen in Ihrem Datenset abbilden. Zum Beispiel können Sie die Farben Ihrer Punkte der Variablen `class` zuordnen, um die Klasse jedes Fahrzeugs deutlich zu machen:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



(Wenn Sie wie Hadley britisches Englisch bevorzugen, dürfen Sie auch `colour` anstelle von `color` schreiben.)

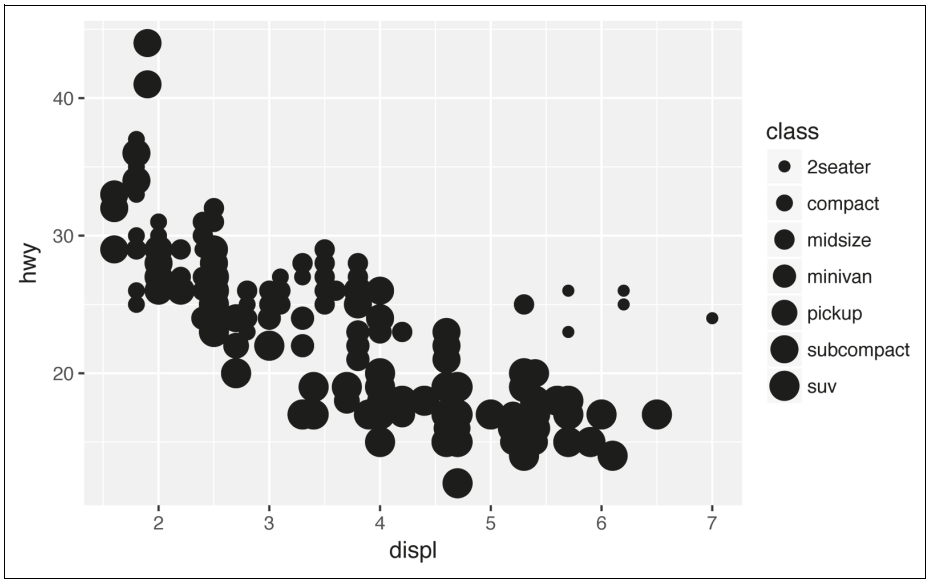
Um eine visuelle Eigenschaft auf eine Variable abzubilden, ordnen Sie den Namen der visuellen Eigenschaft dem Namen der Variablen innerhalb von `aes()` zu. Das

Paket **ggplot2** weist automatisch eine eindeutige Ebene der visuellen Eigenschaft (hier eine eindeutige Farbe) jedem eindeutigen Wert der Variablen zu, was man als *Skalierung* bezeichnet. Außerdem fügt **ggplot2** eine Legende hinzu, die erläutert, welche Ebenen welchen Werten entsprechen.

Die Farben verraten, dass viele der außergewöhnlichen Punkte zu zweisitzigen Autos gehören. Diese Autos scheinen keine Hybridfahrzeuge zu sein, und tatsächlich handelt es sich um Sportwagen! Aber Sportwagen haben genau wie SUVs und Pick-up-Trucks große Motoren, jedoch im Gegensatz zu denen kleine Karosserien wie mittlere und Kleinwagen, was ihre Reichweite verbessert. Im Nachhinein gesehen dürfte es sich bei diesen Autos kaum um Hybridfahrzeuge gehandelt haben, da sie mit großen Motoren ausgestattet sind.

Im vorherigen Beispiel haben wir `class` auf das visuelle Attribut `color` abgebildet, doch wir hätten `class` in der gleichen Weise auch auf das visuelle Attribut `size` abbilden können. In diesem Fall würde die genaue Größe jedes Punkts seine Klassenzugehörigkeit offenbaren. Hier erhalten wir aber eine *Warnung*, weil die Zuordnung einer ungeordneten Variablen (`class`) zu einer geordneten visuellen Eigenschaft (`size`) keine gute Idee ist:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, size = class))
#> Warning: Using size for a discrete variable is not advised.
```

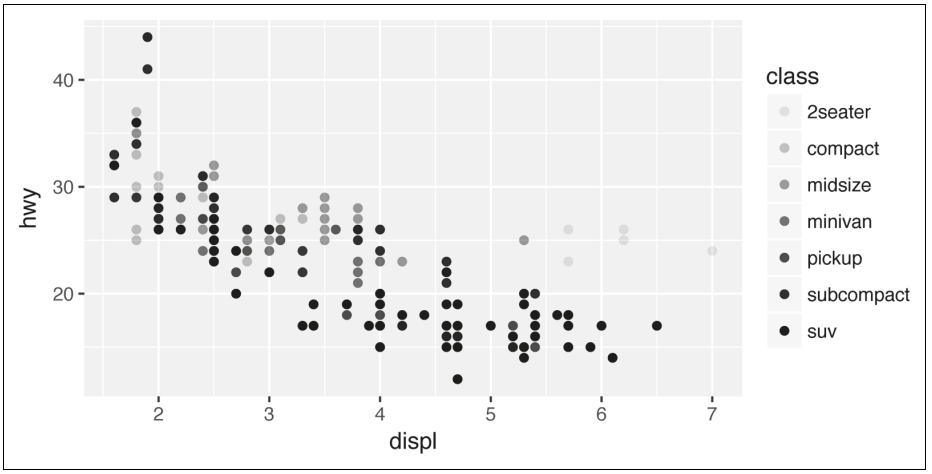


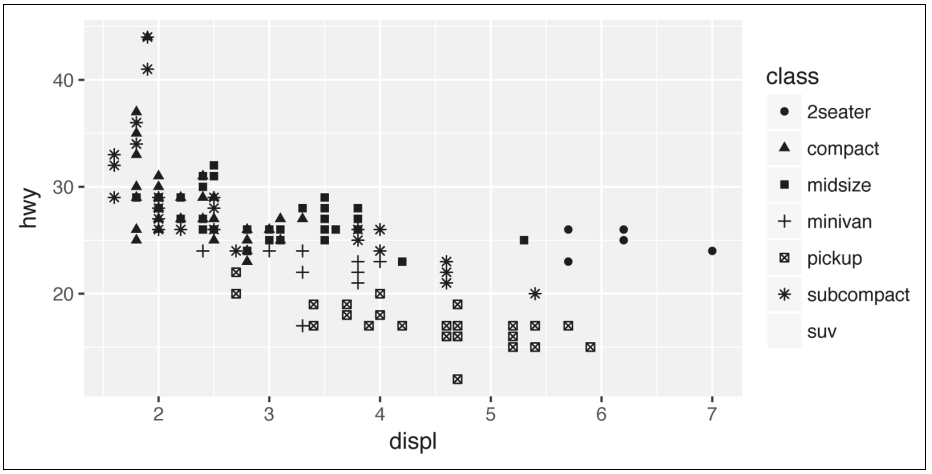
Oder wir hätten `class` dem visuellen Attribut `alpha`, das die Transparenz der Punkte steuert, oder der Form (`shape`) der Punkte zuordnen können:

```
# Top
ggplot(data = mpg) +
```

```
  geom_point(mapping = aes(x = displ, y = hwy, alpha = class))

# Bottom
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, shape = class))
```





Was ist mit den SUVs passiert? **ggplot2** verwendet nur sechs Formen gleichzeitig. Standardmäßig erscheinen zusätzliche Gruppen nicht im Diagramm, wenn Sie diese Ästhetik verwenden.

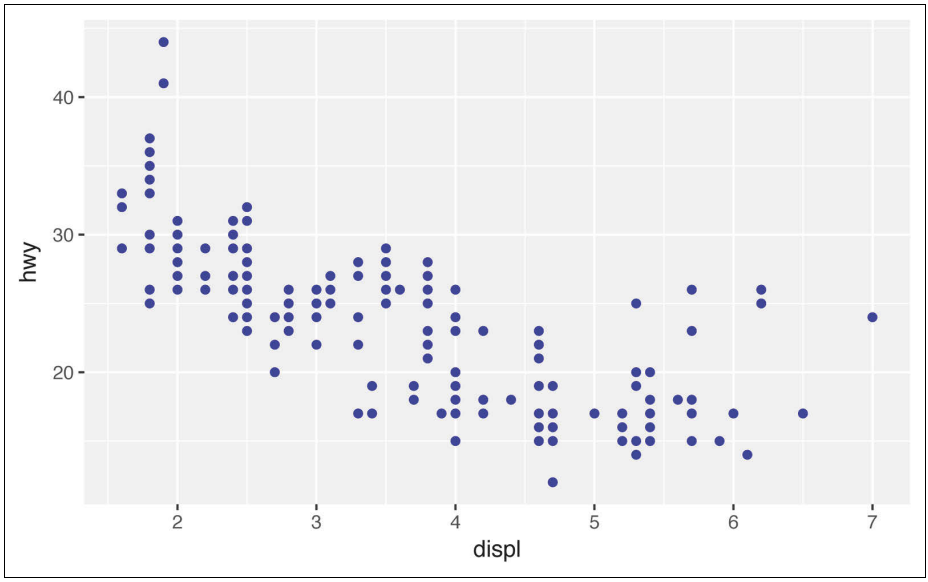
Für jede visuelle Eigenschaft, die Sie verwenden, ordnet `aes()` den Namen der visuellen Eigenschaft einer anzuzeigenden Variablen zu. Die `aes()`-Funktion sammelt die visuellen Zuordnungen, die von einer Ebene verwendet werden, und übergibt sie an das Zuordnungsargument der Ebene. Die Syntax liefert eine nützliche Erkenntnis über `x` und `y`: Die x- und y-Positionen eines Punkts sind selbst visu-

elle Eigenschaften, die Sie auf die Variablen abbilden können, um Informationen über die Daten anzuzeigen.

Nachdem Sie eine visuelle Eigenschaft zugeordnet haben, kümmert sich **ggplot2** um den Rest. Das Paket wählt eine sinnvolle Skala aus, die mit der visuellen Eigenschaft zu verwenden ist, und konstruiert eine Legende, die die Zuordnung zwischen Ebenen und Werten erläutert. Für die visuellen Elemente `x` und `y` erzeugt **ggplot2** keine Legende, erzeugt aber eine Achsenlinie mit Teilstrichen und einer Beschriftung. Die Achsenlinie fungiert als Legende; sie erklärt die Zuordnung zwischen Positionen und Werten.

Die visuellen Eigenschaften Ihrer geometrischen Attribute können Sie auch manuell *festlegen*. Zum Beispiel lassen sich alle Punkte im Diagramm blau darstellen:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), color = "blue")
```



Hier vermittelt die Farbe keine Informationen über eine Variable, sondern ändert nur das Aussehen des Plots. Um eine visuelle Eigenschaft manuell festzulegen, setzen Sie die visuelle Eigenschaft nach dem Namen als Argument Ihrer `geom`-Funktion, das heißt *außerhalb* von `aes()`. Wählen Sie einen Wert aus, der für diese visuelle Eigenschaft sinnvoll ist:

- den Namen einer Farbe als Zeichenfolge
- die Größe eines Punkts in mm
- die Form eines Punkts als Zahl entsprechend Abbildung 1-1

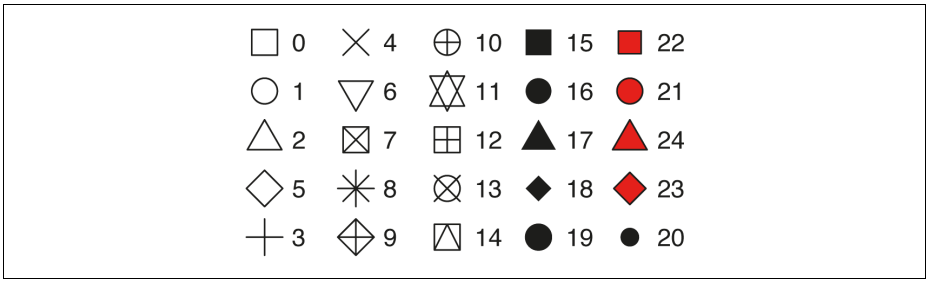


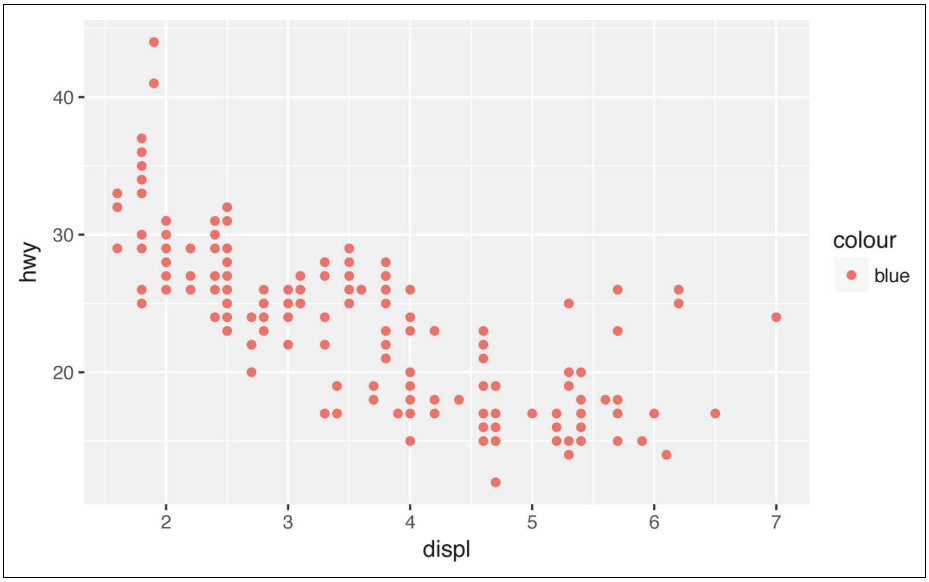
Abbildung 1-1: R bringt 25 integrierte Formen mit, die nach Nummern identifiziert werden.

Scheinbar gibt es bei den Formen gemäß Abbildung 1-1 einige Duplikate: Zum Beispiel sind den Werten 0, 15 und 22 Quadrate zugeordnet. Die Unterscheidung ergibt sich aus der Wechselwirkung der visuellen Eigenschaften `color` und `fill`. Der Rand der hohlen Formen (0 bis 14) wird durch `color` bestimmt, die gedeckten Formen (15 bis 18) sind mit der Farbe `color` gefüllt, und die ausgefüllten Formen (21 bis 24) besitzen einen Rand aus `color` und sind gefüllt mit `fill`.

Übungen

1. Was ist falsch an diesem Code? Warum sind die Punkte nicht blau?

```
ggplot(data = mpg) +
  geom_point(
mapping = aes(x = displ, y = hwy, color = "blue")
```



2. Welche Variablen in `mpg` sind kategorial? Welche Variablen sind kontinuierlich? (Hinweis: Geben Sie **`?mpg`** ein, um die Dokumentation für das Datenset zu lesen.) Wie können Sie diese Informationen sehen, wenn Sie `mpg` ausführen?
3. Bilden Sie eine kontinuierliche Variable auf `color`, `size` und `shape` ab. Wie unterschiedlich verhalten sich diese visuellen Eigenschaften bei kategorialen und kontinuierlichen Variablen?
4. Was passiert, wenn Sie dieselbe Variable zu mehreren visuellen Eigenschaften zuordnen?
5. Was bewirkt die visuelle Eigenschaft `stroke`? Mit welchen Formen funktioniert sie? (Hinweis: Verwenden Sie `?geom_point`.)
6. Was passiert, wenn Sie eine visuelle Eigenschaft auf etwas anderes als einen Variablennamen abbilden, beispielsweise `aes(color = displ < 5)`?

Häufige Probleme

Sobald Sie R-Code ausführen, werden Sie wahrscheinlich auch auf Probleme treffen. Keine Sorge – das passiert jedem. Ich schreibe R-Code seit Jahren, und jeden Tag schreibe ich auch mal Code, der nicht funktioniert!

Vergleichen Sie zunächst aufmerksam den Code, den Sie ausführen, mit dem Code, wie er im Buch abgedruckt ist. R ist äußerst pingelig, und bereits ein falsch gesetztes Zeichen kann schon alles verderben. Achten Sie darauf, dass jede öffnende Klammer (mit einer schließenden Klammer) und jedes Anführungszeichen " mit einem anderen Anführungszeichen " gepaart wird. Es kommt auch vor, dass Sie den Code starten und gar nichts passiert. Sehen Sie dann im linken Teil der Konsole nach: Wenn dort ein Pluszeichen (+) erscheint, geht R davon aus, dass Sie einen Ausdruck noch nicht vollständig eingegeben haben, und wartet darauf, dass Sie die Eingabe beenden. In diesem Fall ist es normalerweise einfach, von Grund auf neu zu beginnen, indem Sie *Esc* drücken, um die Verarbeitung des aktuellen Befehls abzubrechen.

Beim Erzeugen von **ggplot2**-Grafiken treten häufig Probleme auf, wenn das Pluszeichen an der falschen Stelle steht: Es muss am Ende der Zeile und nicht am Anfang stehen! Achten Sie also darauf, dass Sie nicht versehentlich Code wie diesen schreiben:

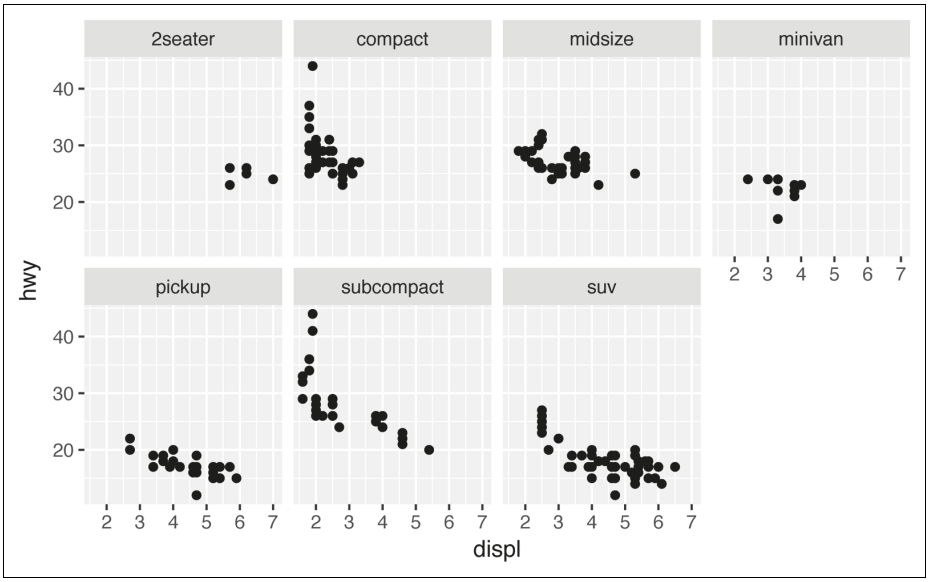
```
ggplot(data = mpg)
+ geom_point(mapping = aes(x = displ, y = hwy))
```

Sollten Sie trotzdem steckenbleiben, probieren Sie die Hilfe aus. Zu jeder R-Funktion erhalten Sie Hilfe, wenn Sie in der Konsole **`?Funktionsname`** eingeben oder in RStudio den Funktionsnamen auswählen und *F1* drücken. Sollte die Hilfe nicht allzu hilfreich erscheinen – keine Bange –, gehen Sie stattdessen zu den Beispielen weiter und suchen nach Code, der dem entspricht, was Sie zu tun versuchen.

Wenn das auch nicht hilft, lesen Sie zumindest die Fehlermeldung aufmerksam. Manchmal liegt die Antwort hier vergraben! Doch wenn Sie gerade in R einsteigen, kann die Antwort zwar in der Fehlermeldung liegen, doch Sie wissen noch nicht, wie sie zu interpretieren ist. Ein anderes großartiges Werkzeug ist Google: Suchen Sie nach dem Text der Fehlermeldung, da höchstwahrscheinlich schon jemand anderes das gleiche Problem hatte und vielleicht online Hilfe bekommen hat.

Facetten

Visuelle Eigenschaften sind eine Möglichkeit, zusätzliche Variablen hinzuzufügen. Bei einer anderen Methode, die sich insbesondere für kategoriale Variablen anbietet, unterteilt man das Diagramm in *Facetten*. Das sind Teildiagramme, die jeweils eine Teilmenge der Daten anzeigen.



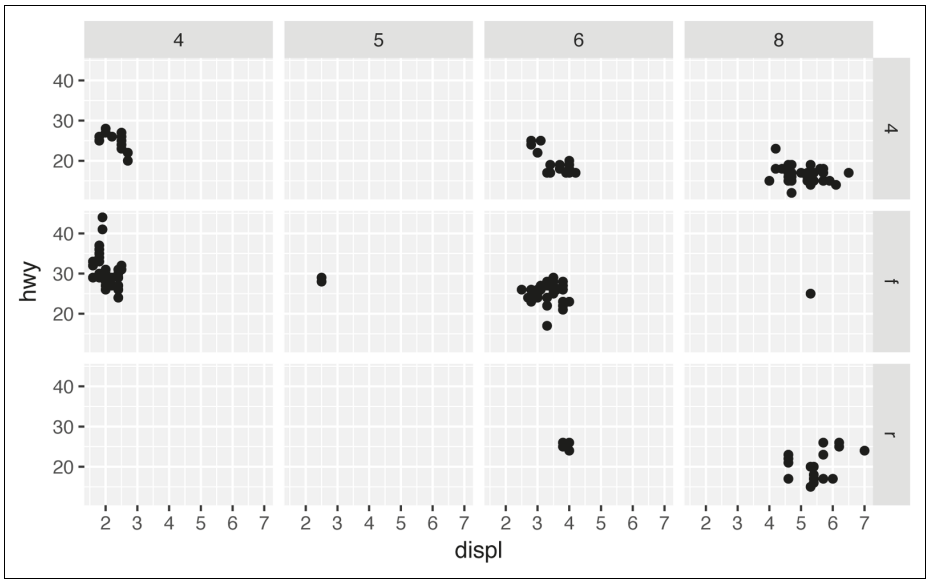
Um ein Diagramm nach einer einzelnen Variablen in Facetten zu unterteilen, verwenden Sie `facet_wrap()`. Das erste Argument von `facet_wrap()` sollte eine Formel sein, die Sie mit einer Tilde (~), gefolgt von einem Variablennamen erzeugen (hier ist »Formel« der Name einer Datenstruktur in R und kein Synonym für »Gleichung«). Die Variable, die Sie an `facet_wrap()` übergeben, sollte diskret sein:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow = 2)
```

Um ein Diagramm nach der Kombination von zwei Variablen in Facetten zu unterteilen, fügen Sie `facet_grid()` in den Diagrammaufruf ein. Das erste Argu-

ment von `facet_grid()` ist ebenfalls eine Formel. Diesmal sollte die Formel zwei Variablennamen enthalten, die durch eine Tilde getrennt sind:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(drv ~ cyl)
```



Wenn Sie die Zeilen- oder Spaltendimension lieber nicht in Facetten teilen möchten, schreiben Sie einen Punkt anstelle eines Variablennamens, zum Beispiel `+ facet_grid(. ~ cyl)`.

Übungen

1. Was passiert, wenn Sie Facetten nach einer kontinuierlichen Variablen erzeugen wollen?
2. Was bedeuten die leeren Zellen in einem Diagramm mit `facet_grid(drv ~ cyl)`? In welchem Verhältnis stehen sie zu diesem Diagramm?

   ```
   ggplot(data = mpg) +
     geom_point(mapping = aes(x = drv, y = cyl))
   ```

3. Welche Diagramme erzeugt der folgende Code? Was bewirkt der Punkt?

   ```
   ggplot(data = mpg) +
     geom_point(mapping = aes(x = displ, y = hwy)) +
     facet_grid(drv ~ .)

   ggplot(data = mpg) +
     geom_point(mapping = aes(x = displ, y = hwy)) +
     facet_grid(. ~ cyl)
   ```

4. Nehmen Sie das erste in Facetten geteilte Diagramm dieses Abschnitts:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow = 2)
```

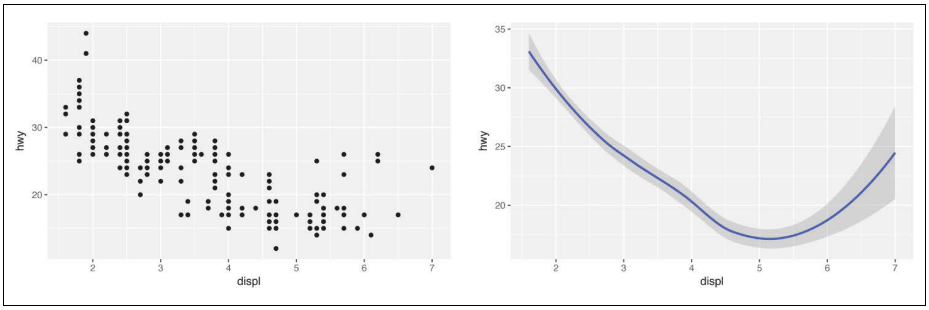
   Welche Vorteile bringt eine Facettendarstellung gegenüber der visuellen Eigenschaft `color`? Was sind die Nachteile? Wie könnte sich die Bilanz ändern, wenn man ein größeres Datenset verwendet?

5. Lesen Sie `?facet_wrap`. Was gibt `nrow` an? Was bedeutet `ncol`? Welche anderen Optionen steuern das Layout der einzelnen Panels? Weshalb hat `facet_grid()` nicht die Variablen `nrow` und `ncol`?

6. Wenn ein Diagramm mit `facet_grid()` erstellt wird, sollte man normalerweise die Variable mit mehr eindeutigen Stufen in die Spalten stellen. Warum?

Geometrische Objekte

Inwieweit ähneln sich diese beiden Diagramme?



Beide Diagramme enthalten dieselbe Variable `x` und dieselbe Variable `y`, und beide beschreiben dieselben Daten. Aber die Diagramme sind nicht identisch. Die einzelnen Diagramme verwenden verschiedene visuelle Objekte, um die Daten darzustellen. In der **ggplot2**-Syntax sagen wir, dass sie verschiedene *geoms* verwenden.

Ein `geom` ist das geometrische Objekt, das ein Diagramm verwendet, um Daten darzustellen. Oftmals werden Diagramme nach dem Typ des `geom` beschrieben, den das Diagramm verwendet. Zum Beispiel verwenden Balkendiagramme `geom_bar`, Liniendiagramme `geom_line`, Boxplots verwenden `geom_boxplot` usw. Streudiagramme durchbrechen diesen Trend, sie verwenden `geom_point`. Wie man in den obigen Diagrammen sehen kann, lassen sich dieselben Daten mit verschiedenen `geom`-Objekten darstellen. Das linke Diagramm verwendet das `geom_point`, das rechte Diagramm das `geom_smooth`, eine glatte Linie, die an die Daten angeglichen ist.

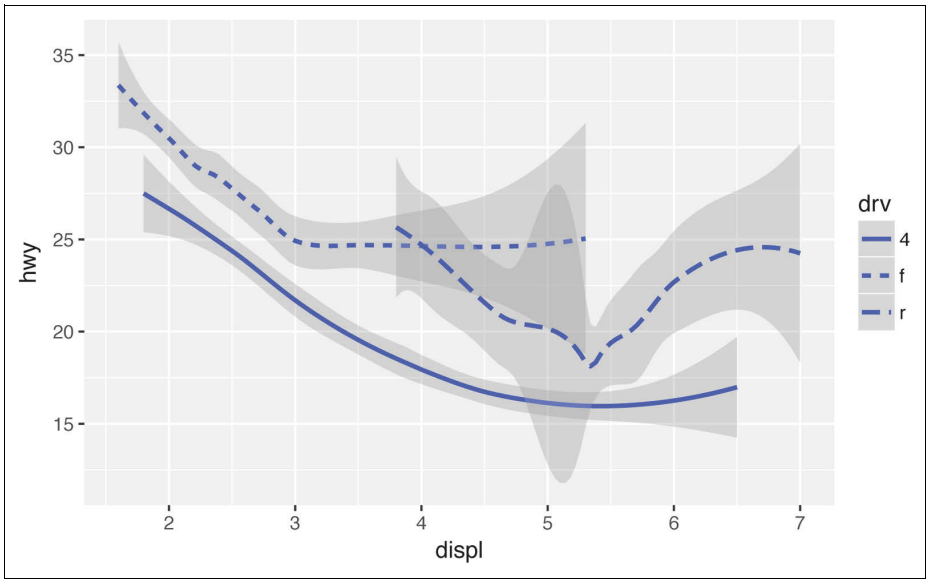
Um das geom in einem Diagramm zu ändern, wechseln Sie die geom-Funktion aus, die Sie ggplot() hinzugefügt haben. Zum Beispiel können Sie für die obigen Diagramme den folgenden Code verwenden:

```
# left
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))

# right
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
```

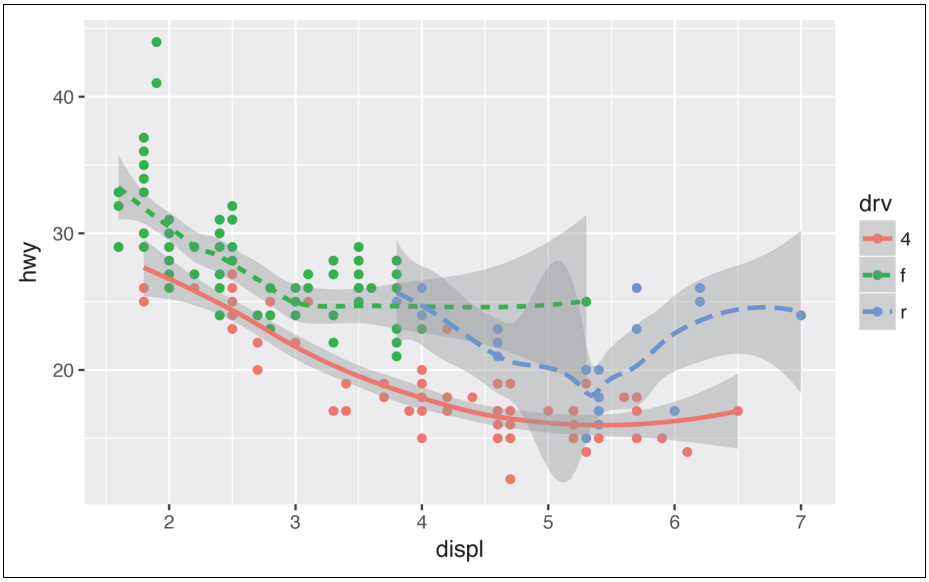
Jede geom-Funktion in **ggplot2** übernimmt ein Argument mapping. Allerdings funktioniert nicht jede visuelle Eigenschaft mit jedem geometrischen Objekt. Man kann zwar die Form eines Punkts festlegen, jedoch nicht die »Form« einer Linie. Andererseits *könnte* man den Linientyp einer Linie festlegen. Mit geom_smooth() wird für jeden eindeutigen Wert der Variablen, den Sie dem Linientyp zuordnen, eine andere Linie mit einem anderen Linientyp gezeichnet:

```
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy, linetype = drv))
```



Hier trennt geom_smooth() die Fahrzeugdaten in drei Linien, und zwar basierend auf dem Wert drv, der die Antriebstechnik eines Fahrzeugs beschreibt. Eine Linie beschreibt alle Punkte mit dem Wert 4, eine andere Linie alle Punkte mit dem Wert f und eine dritte Linie alle Punkte mit dem Wert r. Hierbei steht die 4 für Vierradantrieb, f für Frontantrieb (front-wheel drive) und r für Heckantrieb (rear-wheel drive).

Um dies klarer zu machen, legen wir die Linien über die Rohdaten und färben sie entsprechend dem Wert von `drv`.



Beachten Sie, dass dieses Diagramm zwei geometrische Objekte in derselben Grafik enthält! Falls Ihnen das zu aufregend ist, schnallen Sie sich an! Im nächsten Abschnitt lernen Sie, wie Sie mehrere geometrische Objekte im selben Diagramm unterbringen.

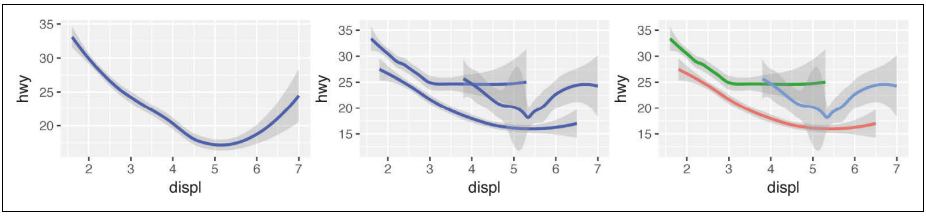
Allein im Paket **ggplot2** sind über 30 `geom`-Objekte enthalten, Erweiterungspakete bieten sogar noch mehr (siehe *http://exts.ggplot2.tidyverse.org* mit Beispielen). Für einen umfassenden Überblick am besten geeignet ist der **ggplot2**-Spickzettel, den Sie unter *https://www.rstudio.com/resources/cheatsheets* finden. Wollen Sie mehr über ein bestimmtes geometrisches Objekt lernen, rufen Sie die Hilfe beispielsweise mit `?geom_smooth` auf.

Viele `geom`-Objekte wie zum Beispiel `geom_smooth()` verwenden ein einzelnes geometrisches Objekt, um mehrere Datenzeilen anzuzeigen. Für diese `geom`-Objekte können Sie die visuelle Eigenschaft `group` auf eine kategoriale Variable festlegen, um mehrere Objekte zu zeichnen. **ggplot2** zeichnet ein separates Objekt für jeden eindeutigen Wert der Gruppierungsvariablen. Praktisch gruppiert **ggplot2** automatisch die Daten für diese `geom`-Objekte, wenn Sie eine visuelle Eigenschaft auf eine diskrete Variable abbilden (wie im `linetype`-Beispiel). Es ist komfortabel, sich auf dieses Feature zu verlassen, weil die visuelle Eigenschaft `group` an sich den `geom`-Objekten weder eine Legende noch Unterscheidungsmerkmale hinzufügt:

```
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
```

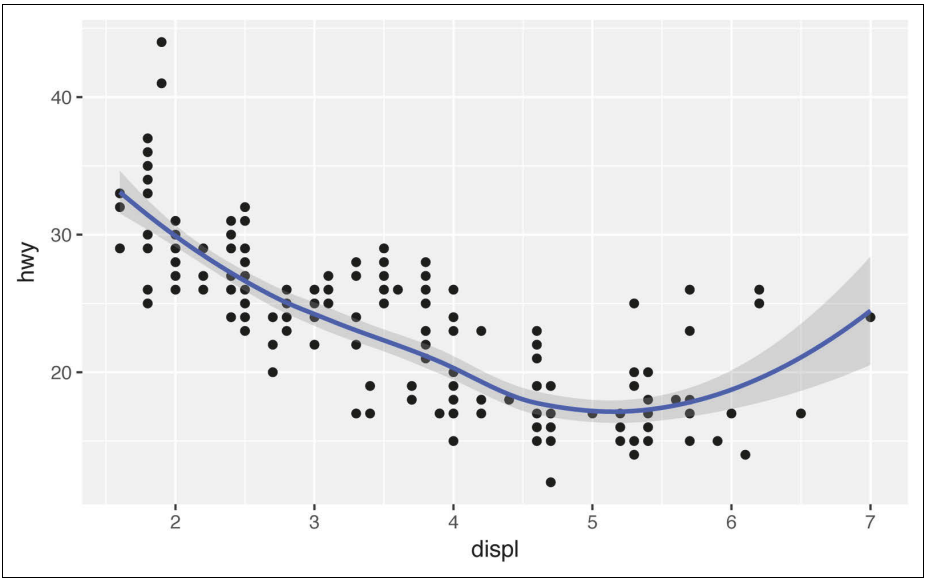
```
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy, group = drv))

ggplot(data = mpg) +
  geom_smooth(
    mapping = aes(x = displ, y = hwy, color = drv),
    show.legend = FALSE
  )
```



Um mehrere geometrische Objekte im selben Diagramm anzuzeigen, fügen Sie mehrere geom-Funktionen in `ggplot()` ein:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
```



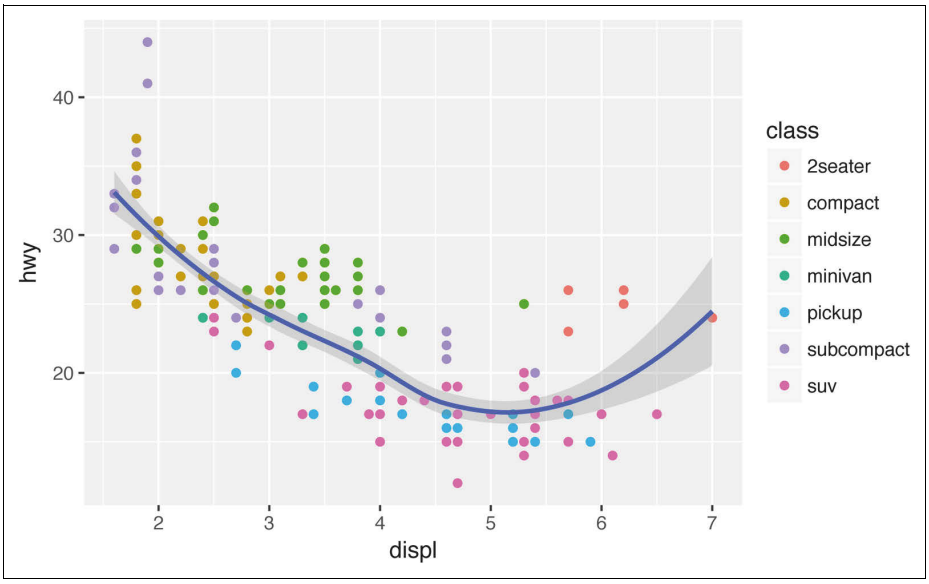
Allerdings bringt das doppelten Code mit sich. Nehmen Sie an, Sie möchten die y-Achse ändern, um `cty` anstelle von `hwy` anzuzeigen. Dann müssten Sie die Variable an zwei Stellen ändern, wobei Sie vielleicht vergessen, die eine zu aktualisieren. Derartige Wiederholungen lassen sich vermeiden, wenn Sie einen Satz von Zuordnungen an `ggplot()` übergeben. **ggplot2** behandelt diese Zuordnungen als globale Zuordnungen, die für jedes geometrische Objekt im Diagramm gelten. Mit ande-

ren Worten produziert der folgende Code das gleiche Diagramm wie der vorherige Code:

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth()
```

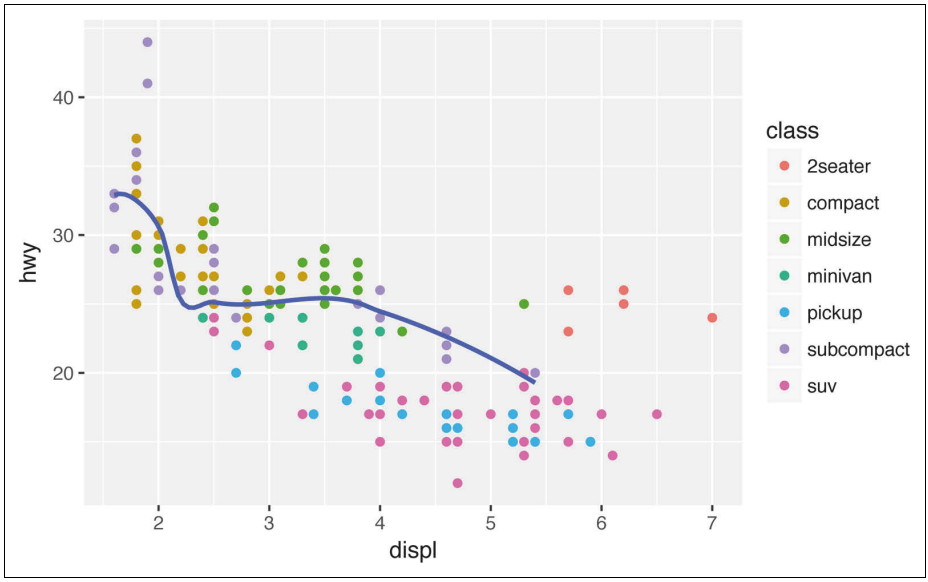
Wenn Sie Zuordnungen in einer geom-Funktion unterbringen, behandelt **ggplot2** sie als lokale Zuordnungen für die Ebene. Ich verwende diese Zuordnungen, um die globalen Zuordnungen *nur für diese Ebene* zu erweitern oder zu überschreiben. Dadurch ist es möglich, verschiedene visuelle Eigenschaften in verschiedenen Ebenen anzuzeigen:

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth()
```



Nach dem gleichen Konzept können Sie verschiedene `data` für jede Ebene spezifizieren. Hier zeigt unsere glatte Linie lediglich eine Teilmenge aus dem `mpg`-Datenset an, die Fahrzeugkategorie `subcompact` (Kleinstwagen). Das lokale `data`-Argument in `geom_smooth()` überschreibt das globale `data`-Argument in `ggplot()` nur für diese Ebene:

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth(
    data = filter(mpg, class == "subcompact"),
    se = FALSE
  )
```



(Wie die Funktion `filter()` arbeitet, lernen Sie im nächsten Kapitel. Fürs Erste müssen Sie nur wissen, dass dieser Befehl nur die Fahrzeugkategorie `subcompact` auswählt.)

Übungen

1. Welches `geom`-Objekt würden Sie verwenden, um ein Liniendiagramm zu zeichnen? Einen Boxplot? Ein Histogramm? Ein Flächendiagramm?
2. Führen Sie den folgenden Code gedanklich aus und sagen Sie voraus, wie die Ausgabe aussehen wird. Führen Sie dann den Code in R aus und überprüfen Sie Ihre Voraussagen:

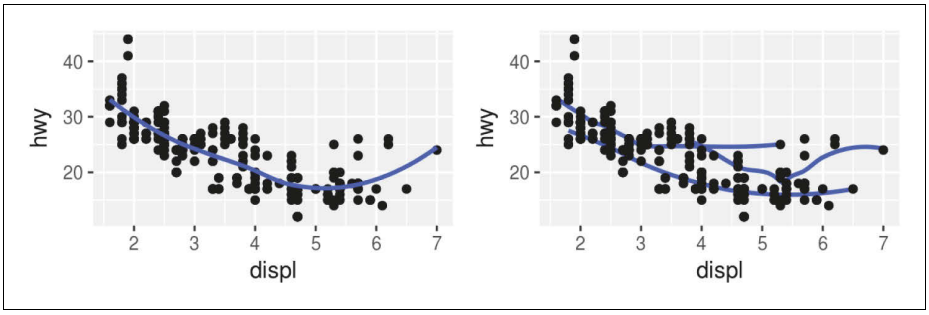
```
ggplot(
  data = mpg,
  mapping = aes(x = displ, y = hwy, color = drv)
) +
  geom_point() +
  geom_smooth(se = FALSE)
```

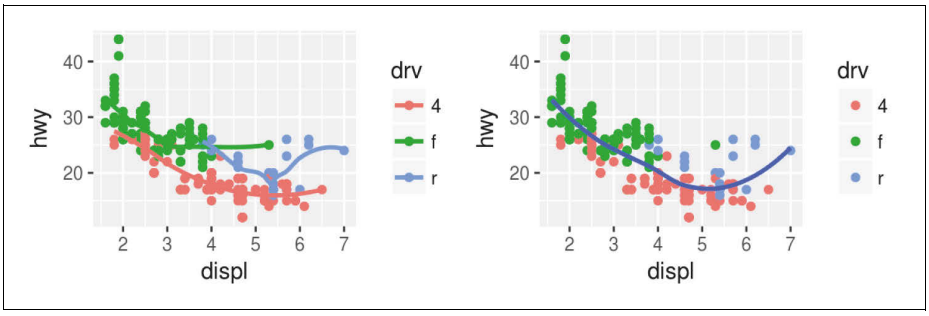
3. Was bewirkt der Ausdruck `show.legend = FALSE`? Was passiert, wenn Sie ihn entfernen? Warum habe ich ihn weiter vorn im Kapitel verwendet?
4. Was bewirkt das Argument `se` im Aufruf von `geom_smooth()`?
5. Sehen die beiden folgenden Diagramme unterschiedlich aus? Warum bzw. warum nicht?

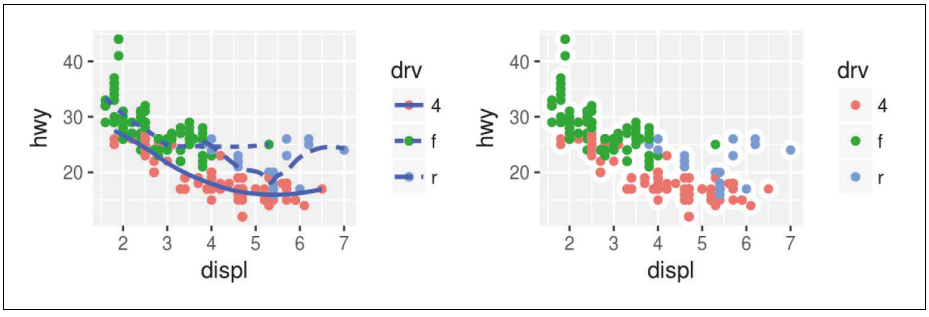
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth()

ggplot() +
  geom_point(
    data = mpg,
    mapping = aes(x = displ, y = hwy)
  ) +
  geom_smooth(
    data = mpg,
    mapping = aes(x = displ, y = hwy)
  )
```

6. Erstellen Sie den erforderlichen R-Code, um die folgenden Diagramme zu erzeugen:



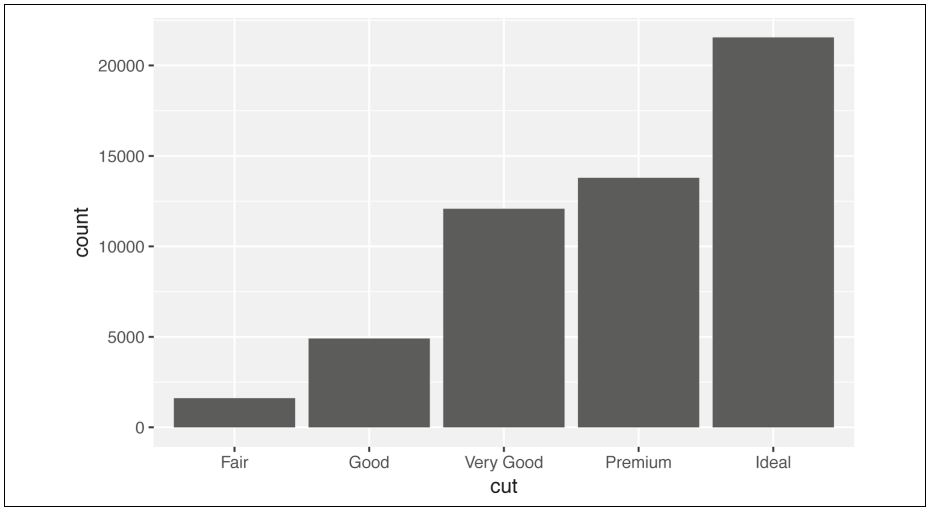




Statistische Transformationen

Als Nächstes wenden wir uns Balkendiagrammen zu. Solche Diagramme sehen einfach aus, doch sie sind unter anderem deshalb interessant, weil sie raffinierte Zusammenhänge aufdecken können. Sehen Sie sich ein einfaches Balkendiagramm an, wie es mit `geom_bar()` gezeichnet wurde. Das folgende Diagramm zeigt die Gesamtanzahl von Diamanten im Datenset `diamonds` an, gruppiert nach `cut`. Das Datenset `diamonds` ist Bestandteil von **ggplot2** und enthält zu etwa 54.000 Diamanten Informationen wie Preis (`price`), Karat (`carat`), Farbe (`color`), Reinheit (`clarity`) und Schliff (`cut`). Das Diagramm zeigt, dass es mehr Diamanten mit hochwertigen Schliffen gibt als solche mit minderwertigen Schliffen:

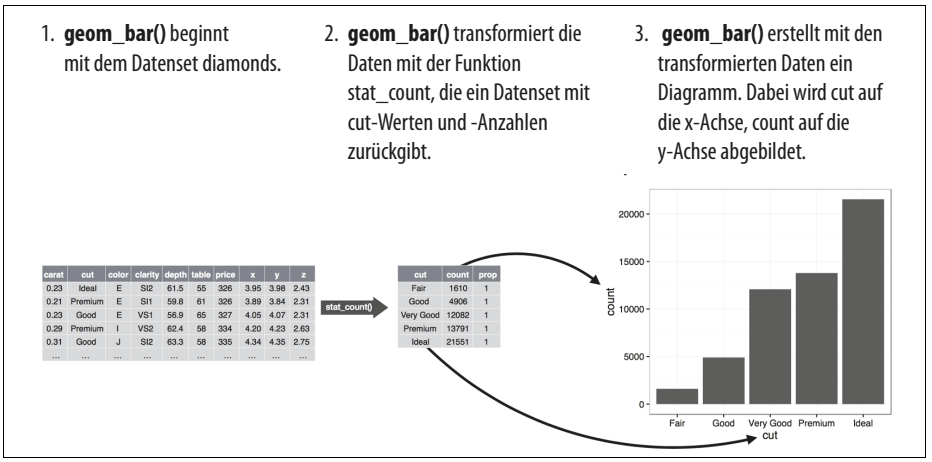
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut))
```



Auf der x-Achse zeigt das Diagramm den Schliff (`cut`), einer Variablen von `diamonds`. Auf der y-Achse wird die Anzahl (`count`) dargestellt, wobei aber `count` keine Variable in `diamonds` ist! Woher stammt nun `count`? Viele Diagramme, wie zum Beispiel Streudiagramme, stellen die Rohdaten des Datensets dar. Andere Diagramme, wie zum Beispiel Balkendiagramme, berechnen neue Werte, die dann im Diagramm wiedergegeben werden.

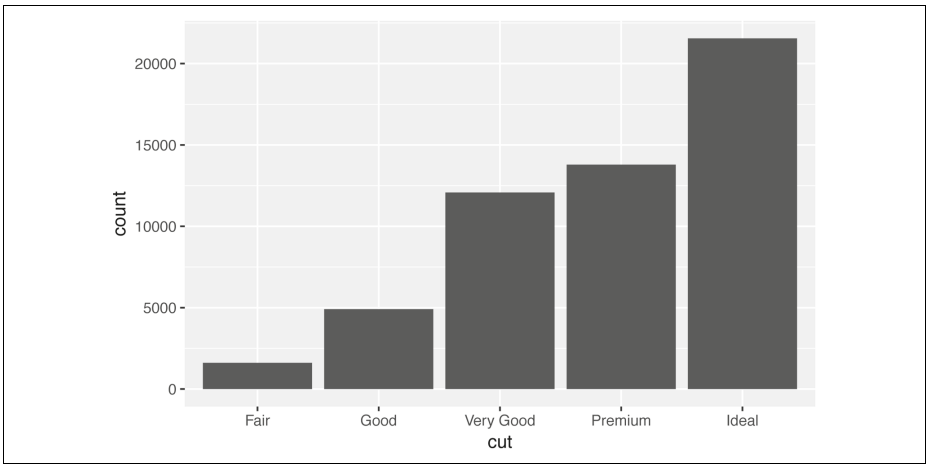
- Balkendiagramme, Histogramme und Häufigkeitspolygone kategorisieren die Daten und stellen dann die Anzahlen in den Kategorien als Diagramm dar, das heißt die Anzahl der Punkte, die in jede Kategorie fallen.
- Glättungsfunktionen passen ein Modell an die Daten an und stellen dann die Vorhersagen aus dem Modell grafisch dar.
- Boxplots berechnen eine robuste Zusammenfassung der Verteilung und zeigen einen speziell formatierten Kasten an.

Der Algorithmus, mit dem die neuen Werte für eine Grafik berechnet werden, heißt `stat`, was kurz für statistische Transformation steht. Die folgende Abbildung beschreibt, wie dieser Vorgang mit `geom_bar()` funktioniert:



Um zu erfahren, welcher `stat` ein `geom` verwendet, sehen Sie sich den Standardwert für das Argument `stat` an. Zum Beispiel zeigt `?geom_bar`, dass für `stat` der Standardwert `count` eingestellt ist. Das heißt, `geom_bar()` verwendet `stat_count()`. Die Funktion `stat_count()` ist auf derselben Seite wie `geom_bar()` dokumentiert, und wenn Sie nach unten scrollen, finden Sie einen Abschnitt »Computed variables« (berechnete Variablen). Hier sehen Sie, dass die Funktion zwei neue Variablen berechnet: `count` und `prop`. Im Allgemeinen können Sie `geom`-Funktionen und `stat`-Funktionen synonym verwenden. Zum Beispiel können Sie das obige Diagramm mit `stat_count()` anstelle von `geom_bar()` neu erstellen:

```
ggplot(data = diamonds) +
  stat_count(mapping = aes(x = cut))
```



Das funktioniert, weil jedes geometrische Objekt eine standardmäßige statistische Transformation hat und jede statistische Transformation ein standardmäßiges geometrisches Objekt. Das heißt, Sie können in der Regel `geom`-Objekte verwenden, ohne sich um die zugrunde liegende statistische Transformation kümmern zu müssen. Es gibt drei Gründe, warum Sie eine statistische Transformation explizit verwenden müssen:

- Sie wollen die standardmäßige statistische Transformation überschreiben. Im folgenden Code habe ich die `stat` in `geom_bar()` von `count` (dem Standardwert) in `identity` geändert. Dadurch kann ich die Höhe der Balken auf die Rohdaten einer y-Variablen abbilden. Wenn Menschen, die nur gelegentlich damit zu tun haben, über Balkendiagramme sprechen, meinen sie manchmal leider den Typ des Balkendiagramms, bei dem die Höhe des Balkens bereits in den Daten präsent ist, oder das vorherige Balkendiagramm, in dem sich die Höhe des Balkens aus der Anzahl der (gezählten) Zeilen ergibt.

```
demo <- tribble(
  ~a,      ~b,
  "bar_1", 20,
  "bar_2", 30,
  "bar_3", 40
)

ggplot(data = demo) +
  geom_bar(
    mapping = aes(x = a, y = b), stat = "identity"
  )
```