Bio & Business Data Science

Nicolas Meseth

Inhalt

Vorwort

Part I

Fall: Campusbier

Einführung

Die Campusbier-Fallstudie ist in dem Sinne eine besondere Fallstudie im Rahmen dieses Buches, als dass sie als Einführung in das Basishandwerkszeug mit R dient. Wenn die Zeit für nur eine Fallstudie ausreicht und man Anfänger mit der Datenanalyse mit R ist, dann ist diese Fallstudie die richtige. Sie steht deshalb auch am Anfang des Buches und dient in allen meinen Modulen zur Datenanalyse mit R als Gegenstand der Einführung.

Das Campusbier-Projekt beschäftigt sich mit der Vermarktung des hochschuleigenen Bieres, das 2009 von Studierenden auf der Versuchsbrauanlage der Hochschule Osnabrück entwickelt wurde. Im Jahr 2019 wurde das Bier erstmals in größerer Menge gebraut und in Flaschen abgefüllt, so dass es der Osnabrücker Bevölkerung zugänglich gemacht werden konnte. Zuvor war das Bier nur intern in Fässern erhältlich, die etwa für Veranstaltungen am Campus der Hochschule verwendet wurden. Die erste öffentliche Verkaufsrunde im Mai/Juni 2019 war ein voller Erfolg, was zu einer dauerhaften Verfügbarkeit des Bieres über den Onlie-Shop campusbier.de führte. Seitdem sind in vielen Projekten weitere Aktionen und Produkte hinzugekommen, deren Verkäufe sich im vorliegenden Datensatz wiederspiegeln.

Der Datensatz

Der Datensatz besteht aus den Informationen zu den knapp 3000 Bestellungen, die in den Jahren 2019 bis 2022 über den Online-Shop aufgegeben wurden. Neben den Metadaten einer Bestellung, wie der Bestellzeitpunkt, der Kunde oder die Zahlungsart, gibt es auch die Informationen über die gekauften Produkte, also den Warenkorb jeder Bestellung. Diese Daten liegen als CSV-Exporte aus dem E-Commerce-System Shopify vor. Der Datensatz besteht aus den beiden Dateien `orders.csv` und `line_items.csv`. Um dem Datenschutz Rechnung zu tragen, wurden sämtliche persönliche Daten der Kunden entfernt. Jeder Kunde ist nur über eine technische Nummer identifizierbar, die in jeder Bestellung angegeben ist. Haben mehrere Bestellungen die gleiche Kundennummer, so stammen diese alle vom selben Kunden.

Ladet euch am besten jetzt die beiden Dateien in euer Arbeitsverzeichnis herunter. Hier die Links zum Download (rechte Maustaste, “Link speichern unter”):

- orders.csv
- line_items.csv

1 CSV-Dateien einlesen

Im ersten Schritt jeder Datenanalyse müssen wir unserem Computer den Datensatz zur Verfügung stellen. Wir sprechen dabei auch vom *Laden* des Datesatzes. Dabei sagen wir dem Computer, wo die Daten zu finden sind und dass er sie für den schnelleren Zugriff in seinen Arbeitsspeicher holen soll.

Daten liegen in den meisten Fällen in Form von Dateien vor. In manchen Fällen sind sie auch in einer Datenbank gespeichert. Im Fall einer Datei kann ein Datensatz in unterschiedlichen *Formaten* darin gespeichert werden. Ein gängiges Format ist das CSV-Format, das auf einfachen Textdateien basiert.

1.1 Das CSV-Format

1.1.1 CSV als weit verbreitetes Format

Für die Speicherung von Daten bieten sich textuelle Formate an, weil sie auf jedem Betriebsystem mit einem einfachen Texteditor betrachtet und bearbeitet werden können. Das ermöglicht das einfache Teilen von Daten und somit die Zusammenarbeit. Auch für den Datenaustausch zwischen verschiedenen Informationssystemen wird häufig ein textbasiertes Format verwendet, um spezifisches Formate der jeweiligen Hersteller, wie etwa proprieäte Datenbanken, zu überbrücken. Deshalb bieten die meisten Informationssysteme Schnittstellen für den Export und Import von Textdateien an. Speziell das CSV-Format ist hier sehr beliebt, aus guten Gründen:

- Die Verwendung von einfachen Textdateien erlaubt die Speicherung und Verarbeitung auf unterschiedlichen Umgebungen wie Windows, macOS oder Linux.
- Das Format ist einfach zu verstehen und auch für Menschen lesbar.
- CSV ist ein offenes Format, d. h. es gibt keine Organisation, die daran die Rechte besitzt und es kann daher von jeder Software verwendet werden. Es gab lange nicht einmal eine offizielle Spezifikation des Formats. Mittlerweile gibt es eine Spezifikation als offizieller MIME Type.

Auch das E-Commerce-System Shopify, aus dem die vorligenden Verkaufsdaten stammen, bietet eine Möglichkeit zum Exportieren von Textdateien im sogenannten CSV-Format an.

1.1.2 CSV eignet sich für strukturierte Daten

CSV steht für *Comma Separated Values* und beschreibt ein Format, um *strukturierte Daten* in einer Textdatei abzuspeichern. Ihr erkennt eine Textdatei im CSV-Format an der Endung `.csv`.

Das CSV-Format basiert auf einfachen Textkodierungen, häufig im UTF-8 oder ASCII-Kodierungssystem (letzeres immer seltener wegen der geringen Anzahl verfügbarer Zeichen), die mit fast jedem Werkzeug und Editor gelesen und bearbeitet werden können. Zusätzlich bildet das CSV-Format eine tabellarische Struktur ab, bei dem die Daten in Zeilen und Spalten getrennt werden. Alle darauf folgenden Zeilen stellen Beobachtungen oder Datensätze dar, deren Variablenwerte mit dem gleichen Trennzeichen abgegrenzt werden.

Das CSV-Format speichert strukturierte Daten in einer tabellarischen Form, ähnlich wie in Spreadsheets. Die erste Zeile einer CSV-Datei ist üblicherweise der sogenannte Header (Kopfzeile) und beinhaltet die Spaltennamen mit Kommata oder Semikolon (Trennzeichen) voneinander getrennt. Jede weitere Zeile stellt eine Beobachtung (Englisch: *observation* oder *case*) oder auch Datensatz (Englisch: *record*) dar. Jeder Datensatz enthält für die im Header definierten Attribute (oder Spalten) einen Wert, die durch das gleiche Trennzeichen voneinander getrennt sind. Es muss nicht jeder Spaltenwert existieren. Sollte ein Wert für eine Beobachtung nicht vorhanden sein, so wird einfach nach dem Komma nichts eingetragen und es folgen zwei Kommata nacheinander. In R werden fehlende Werte mit `NA` gekennzeichnet.

Die Verwendung des Komma als Trennzeichen in CSV-Dateien ist keineswegs verbindlich, auch wenn es Bestandteil des Namens ist. Generell kann jedes Symbol verwendet werden. Häufige Alternativen sind das Semikolon, Leerzeichen oder ein Tabstop. Letzteres wird oft mit der eigenen Endung `.tsv` für *Tab Separated Values* gespeichert.

1.1.3 Beispiel einer CSV-Datei

Der Auschnitt unten zeigt die ersten vier Zeilen der `orders.csv`. Die erste Zeile enthält die Namen der hier gezeigten vier Spalten (der Datensatz hat mehr Spalten, das ist nur ein Auszug), die mit einem Komma voneinander getrennt sind. Darunter folgen drei beispielhafte Datensätze:

```
id,order_id,name,order_number,app_id,created_at
1130007101519,B1014,1014,580111,2019-05-24T14:59:16+02:00
1130014965839,B1015,1015,580111,2019-05-24T15:09:08+02:00
1130026958927,B1016,1016,580111,2019-05-24T15:22:41+02:00
...
```

1.2 CSV-Daten laden mit `{readr}`

1.2.1 Aus einer lokalen Datei

Für das Laden Datensätzen aus CSV-Dateien bietet das Tidyverse ein eigenes Paket namens `{readr}` an. Dieses wird automatisch mit dem `{tidyverse}`-Paket mitgeladen. Das Paket bietet für CSV-Dateien, bei denen das Komma als Trennzeichen verwendet wird, die Funktion `read_csv` an:

```
orders <- read_csv("./data/orders.csv")
```

Auch der R-Basisumfang bietet eine ähnliche Funktion für genau diesen Anwendungsfall an. Diese heisst `read.csv`, man achte hier auf das Detail: Statt eines Unterstrichs wird bei der R-Basisfunktion ein Punkt zwischen den beiden Wörtern `read` und `csv` verwendet. Wenn ihr mit dem Tidyverse und mit Tibbles arbeitet (wie in diesem Buch durchgängig), dann achtet darauf immer die `{readr}`-Funktion `read_csv` zu verwenden, weil nur diese die Daten als Tibble zurückgibt und zudem noch ein paar nützliche Zusatzfunktionen bietet.

1.2.2 Daten von einem Webserver laden

Die CSV-Datei muss nicht lokal auf dem eigenen Rechner vorliegen, sondern kann mit `read_csv` über das HTTP-Protkoll direkt von einem Webserver im Internet abgerufen werden. Der Code unten lädt die tagesaktuelle Version des Covid-19-Datensatzes, der auf den Servern von Our World in Data gehostet wird:

```
covid <- read_csv("https://covid.ourworldindata.org/data/owid-covid-data.csv")
```

```
Rows: 219868 Columns: 67
-- Column specification --------------------------------------------------------
Delimiter: ","
chr   (4): iso_code, continent, location, tests_units
dbl  (62): total_cases, new_cases, new_cases_smoothed, total_deaths, new_dea...
date  (1): date

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

2 Datensätze erkunden

2.1 Sprechende Tibbles

Alleine beim Aufruf seines Namens gibt ein Tibble in der Konsole viele seiner Informationen preis:

```
orders
```

```
# A tibble: 2,874 x 68
     order_id name  order~1 app_id created_at          updated_at          test
        <dbl> <chr>   <dbl>  <dbl> <dttm>              <dttm>              <lgl>
 1    1.13e12 B1014    1014 580111 2019-05-24 12:59:16 2019-06-19 13:23:26 FALSE
 2    1.13e12 B1015    1015 580111 2019-05-24 13:09:08 2019-06-21 14:40:07 FALSE
 3    1.13e12 B1016    1016 580111 2019-05-24 13:22:41 2019-06-21 12:35:23 FALSE
 4    1.13e12 B1017    1017 580111 2019-05-24 13:27:43 2019-06-21 14:27:18 FALSE
 5    1.13e12 B1018    1018 580111 2019-05-24 13:36:46 2019-06-21 12:11:57 FALSE
 6    1.13e12 B1019    1019 580111 2019-05-24 13:44:41 2019-06-21 14:37:21 FALSE
 7    1.13e12 B1020    1020 580111 2019-05-24 13:49:21 2019-06-21 12:25:16 FALSE
 8    1.13e12 B1021    1021 580111 2019-05-24 13:59:57 2019-06-21 11:49:47 FALSE
 9    1.13e12 B1022    1022 580111 2019-05-24 14:43:53 2019-06-19 14:12:38 FALSE
10    1.13e12 B1023    1023 580111 2019-05-24 14:48:16 2019-06-21 15:54:24 FALSE
# ... with 2,864 more rows, 61 more variables: current_subtotal_price <dbl>,
#   current_total_price <dbl>, current_total_discounts <dbl>,
#   current_total_duties_set <dbl>, total_discounts <dbl>,
#   total_line_items_price <dbl>, total_outstanding <dbl>, total_price <dbl>,
#   total_tax <dbl>, total_tip_received <dbl>, taxes_included <lgl>,
#   discount_codes <chr>, financial_status <chr>, fulfillment_status <chr>,
#   source_name <chr>, landing_site <chr>, landing_site_ref <chr>, ...
```

Neben den ersten paar Zeilen als Vorschau gibt ein Tibble auch die Gesamtzahl an Zeilen und Spalten aus. Hier sind es 2874 Zeilen und 68 Spalten. Darunter folgt eine mit Kommata getrennte Auflistung der Spaltennamen und ihren Datentypen. Diese Liste wird aber nach wenigen Zeilen abgebrochen, um die Konsole nicht mit Text zu überladen.

Versucht das einmal: Ladet die CSV-Datei statt mit `read_csv` mit der Funktion `read.csv` aus dem Basis-R. Gebt jetzt den Namen des Dataframes in die Konsole ein und drückt Enter. Was ist der Unterschied bei der Ausgabe? Was gefällt euch besser?

2.2 Spalten und deren Datentypen

```
spec(orders)
```

```
cols(
  order_id = col_double(),
  name = col_character(),
  order_number = col_double(),
  app_id = col_double(),
  created_at = col_datetime(format = ""),
  updated_at = col_datetime(format = ""),
  test = col_logical(),
  current_subtotal_price = col_double(),
  current_total_price = col_double(),
  current_total_discounts = col_double(),
  current_total_duties_set = col_double(),
  total_discounts = col_double(),
  total_line_items_price = col_double(),
  total_outstanding = col_double(),
  total_price = col_double(),
  total_tax = col_double(),
  total_tip_received = col_double(),
  taxes_included = col_logical(),
  discount_codes = col_character(),
  financial_status = col_character(),
  fulfillment_status = col_character(),
  source_name = col_character(),
  landing_site = col_character(),
  landing_site_ref = col_character(),
  location_id = col_double(),
  note = col_character(),
  tags = col_character(),
  processed_at = col_datetime(format = ""),
  processing_method = col_character(),
  payment_details_gateway = col_character(),
```

```
  payment_details_credit_card_company = col_character(),
  customer_id = col_double(),
  customer_accepts_marketing = col_double(),
  customer_accepts_marketing_updated_at = col_datetime(format = ""),
  customer_marketing_opt_in_level = col_character(),
  customer_sms_marketing_consent = col_logical(),
  customer_created_at = col_datetime(format = ""),
  customer_updated_at = col_datetime(format = ""),
  customer_gender = col_character(),
  customer_is_hsos = col_double(),
  customer_state = col_character(),
  customer_orders_count = col_double(),
  customer_total_spent = col_double(),
  customer_last_order_id = col_double(),
  customer_note = col_character(),
  customer_verified_email = col_double(),
  customer_tax_exempt = col_double(),
  customer_tags = col_character(),
  customer_last_order_name = col_character(),
  campaign_tag = col_character(),
  shipping_address_city = col_character(),
  shipping_address_zip = col_double(),
  shipping_address_country = col_character(),
  shipping_address_latitude = col_double(),
  shipping_address_longitude = col_double(),
  billing_address_city = col_character(),
  billing_address_zip = col_double(),
  billing_address_country = col_character(),
  billing_address_company = col_character(),
  billing_address_latitude = col_double(),
  billing_address_longitude = col_double(),
  client_details_browser_ip = col_character(),
  client_details_browser_height = col_double(),
  client_details_browser_width = col_double(),
  client_details_user_agent = col_character(),
  cancel_reason = col_character(),
  cancelled_at = col_datetime(format = ""),
  closed_at = col_datetime(format = "")
)
```

```
col_double()
```

```
<collector_double>
```

2.3 Häufigkeiten schnell erfassen

Mit dem bereits bekannten `{janitor}`-Paket erhalten wir eine Funktion, um für nominal skalierte Merkmale schnell die Häufigkeiten, sowohl absolut als auch prozentual, zu ermitteln:

```
library(janitor)

orders %>%
  tabyl(payment_details_gateway)
```

```
payment_details_gateway    n      percent valid_percent
                 manual  194 0.0675017397    0.06752523
                 paypal 1790 0.6228253305    0.62304212
       shopify_payments  889 0.3093249826    0.30943265
                   <NA>    1 0.0003479471            NA
```

Wenn wir eine zweite Variable hinzufügen, so erstellt `tabyl` eine Kreuztabelle mit den absoluten Häufigkeiten der jeweiligen Kombinationen:

```
orders %>%
  tabyl(payment_details_gateway, payment_details_credit_card_company)
```

```
payment_details_gateway American Express Mastercard Visa  NA_
                 manual                0          0    0  194
                 paypal                0          1    3 1786
       shopify_payments               14        372  303  200
                   <NA>                0          0    0    1
```

Wir können so erkennen, dass für PayPal-Zahlungen nur sehr selten ein Kreditkartenanbieter hinterlegt ist. Bei den Shopify-Payments hingegen ist das in den meisten Bestellungen der Fall. Dabei liegt Mastercard knapp vor Visa, American Express ist eher die Ausnahme.

> Es wäre doch interessant zu wissen, warum genau eine Bestellung keine Angabe zur Zahlungsart besitzt. Prüft doch mal nach, welche das ist und versucht die Frage zu beantworten.

3 Spalten auswählen

4 Lange und breite Daten

Wie bereits in Abschnitt Fall 2: Covid gesehen, können Daten in unterschiedlicher Form vorliegen.

Quellen