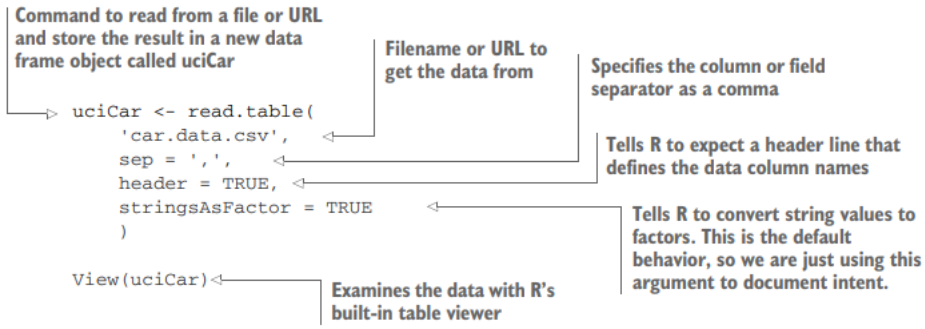
```
#https://raw.githubusercontent.com/WinVector/PDSwR2/main/UCICar/car.data.csv

#https://raw.githubusercontent.com/charlesmutai/statlog/main/creditdata.csv
#https://raw.githubusercontent.com/WinVector/PDSwR2/main/Statlog/mapping.R
# https://raw.githubusercontent.com/WinVector/PDSwR2/main/Statlog/german.data
```

LOADING WELL-STRUCTURED DATA

Loading data of this type into R is a one-liner: we use the R command utils::read .table() and we're done.




```
uciCar <- read.table(
'https://raw.githubusercontent.com/WinVector/PDSwR2/main/UCICar/car.data.csv',
sep = ',',
header = TRUE,
stringsAsFactor = TRUE
)
uciCar
```

```
uciCar <- read.table('https://raw.githubusercontent.com/WinVector/PDSwR2/main/UCICar/car.data.csv', sep=',', header=TRUE, stringsAsFactor= TR
uciCar
```

A data.frame: 1728 × 7

buying	maint	doors	persons	lug_boot	safety	rating
<fct>	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>
vhigh	vhigh	2	2	small	low	unacc
vhigh	vhigh	2	2	small	med	unacc
vhigh	vhigh	2	2	small	high	unacc
vhigh	vhigh	2	2	med	low	unacc
vhigh	vhigh	2	2	med	med	unacc
vhigh	vhigh	2	2	med	high	unacc
vhigh	vhigh	2	2	big	low	unacc
vhigh	vhigh	2	2	big	med	unacc
vhigh	vhigh	2	2	big	high	unacc
vhigh	vhigh	2	4	small	low	unacc
vhigh	vhigh	2	4	small	med	unacc
vhigh	vhigh	2	4	small	high	unacc
vhigh	vhigh	2	4	med	low	unacc
vhigh	vhigh	2	4	med	med	unacc
vhigh	vhigh	2	4	med	high	unacc
vhigh	vhigh	2	4	big	low	unacc
vhigh	vhigh	2	4	big	med	unacc
vhigh	vhigh	2	4	big	high	unacc
vhigh	vhigh	2	more	small	low	unacc
vhigh	vhigh	2	more	small	med	unacc
vhigh	vhigh	2	more	small	high	unacc
vhigh	vhigh	2	more	med	low	unacc
vhigh	vhigh	2	more	med	med	unacc
vhigh	vhigh	2	more	med	high	unacc
vhigh	vhigh	2	more	big	low	unacc
vhigh	vhigh	2	more	big	med	unacc
vhigh	vhigh	2	more	big	high	unacc
vhigh	vhigh	3	2	small	low	unacc
vhigh	vhigh	3	2	small	med	unacc
vhigh	vhigh	3	2	small	high	unacc
⋮	⋮	⋮	⋮	⋮	⋮	⋮
low	low	4	more	big	low	unacc
low	low	4	more	big	med	good
low	low	4	more	big	high	vgood
low	low	5more	2	small	low	unacc
low	low	5more	2	small	med	unacc
low	low	5more	2	small	high	unacc
low	low	5more	2	med	low	unacc
low	low	5more	2	med	med	unacc
low	low	5more	2	med	high	unacc

```
head(uciCar)
```

A data.frame: 6 × 7

	buying	maint	doors	persons	lug_boot	safety	rating
	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>
1	vhigh	vhigh	2	2	small	low	unacc
2	vhigh	vhigh	2	2	small	med	unacc
3	vhigh	vhigh	2	2	small	high	unacc
4	vhigh	vhigh	2	4	small	low	unacc
5	vhigh	vhigh	2	4	small	med	unacc
6	vhigh	vhigh	2	4	small	high	unacc

```
class(uciCar)
```

'data.frame'

```
typeof(uciCar)
```

'list'

```
dim(uciCar)
```

1728 · 7

```
str(uciCar)
```

```
'data.frame':   1728 obs. of  7 variables:
 $ buying  : Factor w/ 4 levels "high","low","med",..: 4 4 4 4 4 4 4 4 4 4 4 4 ...
 $ maint   : Factor w/ 4 levels "high","low","med",..: 4 4 4 4 4 4 4 4 4 4 4 4 ...
 $ doors   : Factor w/ 4 levels "2","3","4","5more": 1 1 1 1 1 1 1 1 1 1 1 1 ...
 $ persons : Factor w/ 3 levels "2","4","more": 1 1 1 1 1 1 1 1 1 2 2 2 ...
 $ lug_boot: Factor w/ 3 levels "big","med","small": 3 3 3 2 2 2 1 1 1 3 3 3 ...
 $ safety  : Factor w/ 3 levels "high","low","med": 2 3 1 2 3 1 2 3 1 2 3 1 ...
 $ rating  : Factor w/ 4 levels "acc","good","unacc",..: 3 3 3 3 3 3 3 3 3 3 3 3 ...
```

```
print(uciCar)
```

```
1714    low   low 5more       4      med    low  unacc
1715    low   low 5more       4      med    med   good
1716    low   low 5more       4      med   high  vgood
1717    low   low 5more       4      big    low  unacc
1718    low   low 5more       4      big    med   good
1719    low   low 5more       4      big   high  vgood
1720    low   low 5more    more    small    low  unacc
1721    low   low 5more    more    small    med    acc
1722    low   low 5more    more    small   high   good
1723    low   low 5more    more      med    low  unacc
1724    low   low 5more    more      med    med   good
1725    low   low 5more    more      med   high  vgood
1726    low   low 5more    more      big    low  unacc
1727    low   low 5more    more      big    med   good
1728    low   low 5more    more      big   high  vgood
```

```
View(uciCar)
```

A data.frame: 1728 × 7

buying	maint	doors	persons	lug_boot	safety	rating
<fct>	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>
vhigh	vhigh	2	2	small	low	unacc
vhigh	vhigh	2	2	small	med	unacc
vhigh	vhigh	2	2	small	high	unacc
vhigh	vhigh	2	2	med	low	unacc
vhigh	vhigh	2	2	med	med	unacc
vhigh	vhigh	2	2	med	high	unacc
vhigh	vhigh	2	2	big	low	unacc
vhigh	vhigh	2	2	big	med	unacc
vhigh	vhigh	2	2	big	high	unacc
vhigh	vhigh	2	4	small	low	unacc
vhigh	vhigh	2	4	small	med	unacc
vhigh	vhigh	2	4	small	high	unacc
vhigh	vhigh	2	4	med	low	unacc
vhigh	vhigh	2	4	med	med	unacc
vhigh	vhigh	2	4	med	high	unacc
vhigh	vhigh	2	4	big	low	unacc
vhigh	vhigh	2	4	big	med	unacc
vhigh	vhigh	2	4	big	high	unacc
vhigh	vhigh	2	more	small	low	unacc
vhigh	vhigh	2	more	small	med	unacc
vhigh	vhigh	2	more	small	high	unacc
vhigh	vhigh	2	more	med	low	unacc
vhigh	vhigh	2	more	med	med	unacc
vhigh	vhigh	2	more	med	high	unacc
vhigh	vhigh	2	more	big	low	unacc

EXAMINING OUR DATA

- class()—Tells you what kind of R object you have. In our case, class(uciCar) tells us the object uciCar is of class data.frame. Class is an object-oriented concept, which describes how an object is going to behave. R also has a (less useful)
- typeof() command, which reveals how the object's storage is implemented.
- dim()—For data frames, this command shows how many rows and columns are in the data.
- head()—Shows the top few rows (or "head") of the data. Example: head(uciCar).
- help()—Provides the documentation for a class. In particular, try help(class (uciCar)).
- str()—Gives you the structure for an object. Try str(uciCar).
- summary()—Provides a summary of almost any R object. summary(uciCar) shows us a lot about the distribution of the UCI car data.
- print()—Prints all the data. Note: for large datasets, this can take a very long time and is something you want to avoid.
- View()—Displays the data in a simple spreadsheet-like grid viewer.

low low 5more 2 small med unacc

low low 5more 2 med low unacc

WORKING WITH OTHER DATA FORMATS

.csv is not the only common data file format you'll encounter. Other formats include .tsv (tab-separated values), pipe-separated (vertical bar) files, Microsoft Excel workbooks, JSON data, and XML. R's built-in read.table() command can be made to read most separated value formats. Many of the deeper data formats have corresponding R packages:

- CSV/TSV/FWF—The package reader (http://readr.tidyverse.org) supplies tools for reading "separated data" such as comma-separated values (CSV), tabseparated values (TSV), and fixed-width files (FWF).

- SQL—https://CRAN.R-project.org/package=DBI
- XLS/XLSX—http://readxl.tidyverse.org
- .RData/.RDS—R has binary data formats (which can avoid complications of parsing, quoting, escaping, and loss of precision in reading and writing numeric or floating-point data as text). The .RData format is for saving sets of objects and object names, and is used through the save()/load() commands. The .RDS format is for saving single objects (without saving the original object name) and is used through the saveRDS()/readRDS() commands. For ad hoc work, .RData is more convenient (as it can save the entire R workspace), but for reusable work, the .RDS format is to be preferred as it makes saving and restoring a bit more explicit. To save multiple objects in .RDS format, we suggest using a named list.
- JSON—https://CRAN.R-project.org/package=rjson
- XML—https://CRAN.R-project.org/package=XML
- MongoDB—https://CRAN.R-project.org/package=mongolite

low low 5more more small med acc

TRANSFORMING DATA IN R

Data often needs a bit of transformation before it makes sense. In order to decrypt troublesome data, you need what's called the schema documentation or a data dictionary. In this case, the included dataset description says the data is 20 input columns followed by one result column. In this example, there's no header in the data file. The column definitions and the meaning of the cryptic A-* codes are all in the accompanying data documentation.

low low 5more more big med good

Let's start by loading the raw data into R. Start a copy of R or RStudio and type in the commands in the following listing.

Loading the credit dataset

```
d <- read.table('https://raw.githubusercontent.com/charlesmutai/statlog/main/creditdata.csv', sep=' ',
stringsAsFactors = FALSE, header = FALSE)
head(d)
```

```
d <- read.table('https://raw.githubusercontent.com/charlesmutai/statlog/main/creditdata.csv', sep=',', stringsAsFactor=FALSE, header=FALSE)
head(d)
```

	V1	V2	V3	V4	'
	<int>	<chr>	<chr>	<chr>	<ch
1	NA	Status_of_existing_checking_account	Duration_in_month	Credit_history	Purpo
2	1	... < 0 DM	6	critical account/other credits existing (not at this bank)	radio/televisior
3	2	0 < = ... < 200 DM	48	existing credits paid back duly till now	radio/televisior
4	3	no checking account	12	critical account/other credits existing (not at this bank)	education
5	4	... < 0 DM	42	existing credits paid back duly till now	furniture/equipme
6	5	... < 0 DM	24	delay in paying off in the past	car (new)

```
table(d$Purpose, d$Good_Loan)
```

```
< table of extent 0 x 0 >
```

As there was no column header in the file, our data.frame d will have useless column names of the form V#.

We can change the column names to something meaningful with the c() command, as shown in the following listing

Setting column names

```
d <- read.table('german.data',
                sep  =  " ",
                stringsAsFactors  =  FALSE, header  =  FALSE)
```

```
d <- read.table('https://raw.githubusercontent.com/WinVector/PDSwR2/main/Statlog/german.data', sep=' ', stringsAsFactor=FALSE, header=FALSE)
head(d)
```

A data.frame: 6 × 21

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	⋯	V12	V13
	<chr>	<int>	<chr>	<chr>	<int>	<chr>	<chr>	<int>	<chr>	<chr>	⋯	<chr>	<int>
1	A11	6	A34	A43	1169	A65	A75	4	A93	A101	⋯	A121	67
2	A12	48	A32	A43	5951	A61	A73	2	A92	A101	⋯	A121	22
3	A14	12	A34	A46	2096	A61	A74	2	A93	A101	⋯	A121	49
4	A11	42	A32	A42	7882	A61	A74	2	A93	A103	⋯	A122	45
5	A11	24	A33	A40	4870	A61	A73	3	A93	A101	⋯	A124	53
6	A14	36	A32	A46	9055	A65	A73	2	A93	A101	⋯	A124	35

```
colnames(d) <- c('Status_of_existing_checking_account', 'Duration_in_month',
                 'Credit_history', 'Purpose', 'Credit_amount', 'Savings_accou
     nt_bonds',
                 'Present_employment_since',
                 'Installment_rate_in_percentage_of_disposable_income',
                 'Personal_status_and_sex', 'Other_debtors_guarantors',
                 'Present_residence_since', 'Property', 'Age_in_years',
                 'Other_installment_plans', 'Housing',
                 'Number_of_existing_credits_at_this_bank', 'Job',
                 'Number_of_people_being_liable_to_provide_maintenance_for',
                 'Telephone', 'foreign_worker', 'Good_Loan')
str(d)
```

The c() command is R's method to construct a vector.12 We copied the column names directly from the dataset documentation. By assigning our vector of names into the data frame's colnames(), we've reset the data frame's column names to something sensible.

Transforming the car data

```
source("https://raw.githubusercontent.com/WinVector/PDSwR2/main/Statlog/mapping.R")
for(ci in colnames(d)) {
if(is.character(d[[ci]])) {
d[[ci]] <- as.factor(mapping[d[[ci]]])
}
}
```

```
source("https://raw.githubusercontent.com/WinVector/PDSwR2/main/Statlog/mapping.R")
for(ci in colnames(d)) {
if(is.character(d[[ci]])) {
d[[ci]] <- as.factor(mapping[d[[ci]]])
}
}
```

Examining our new data

We can now easily examine the purpose of the first three loans with the command print(d[1:3,'Purpose']). We can look at the distribution of loan purpose with summary(d$Purpose). This summary is why we converted the values into factors, as summary() does not report much for

string/character types, though we could also use table(d dollarsign Purpose, useNA = "always") directly on character types. We can also start to investigate the relation of loan type to loan outcome, as shown in the following listing.

Summary of Good_Loan and Purpose

```
d <- read.csv("https://raw.githubusercontent.com/charlesmutai/statlog/main/creditdata.csv")
table(d$Purpose, d$Good_Loan)
```

```
summary(d$Purpose, d$Good_Loan)
```

```
     Length  Class   Mode
          0   NULL   NULL
```

Double-click (or enter) to edit

EXPLORING DATA

> 1. Using summary statistics to explore data

Example: Suppose your goal is to build a model to predict which of your customers don't have health insurance. You've collected a dataset of customers whose health insurance status you know. You've also identified some customer properties that you believe help predict the probability of insurance coverage: age, employment status, income, information about residence and vehicles, and so on.

You've put all your data into a single data frame called customer_data that you've input into R.1 Now you're ready to start building the model to identify the customers you're interested in. It's tempting to dive right into the modeling step without looking very hard at the dataset first, especially when you have a lot of data. Resist the temptation. No dataset is perfect: you'll be missing information about some of your customers, and you'll have incorrect data about others. Some data fields will be dirty and inconsistent. If you don't take the time to examine the data before you start to model, you may find yourself redoing your work repeatedly as you discover bad data fields or variables that need to be transformed before modeling. In the worst case, you'll build a model that returns incorrect predictions—and you won't be sure why.

GET TO KNOW YOUR DATA BEFORE MODELING

By addressing data issues early, you can save yourself some unnecessary work, and a lot of headaches! You'd also like to get a sense of who your customers are:

- Are they young, middle-aged, or seniors?
- How affluent are they?
- Where do they live?

  Knowing the answers to these questions can help you build a better model, because you'll have a more specific idea of what information most accurately predicts the probability of insurance coverage. In this lesson, we'll demonstrate some ways to get to know your data, and discuss some of the potential issues that you're looking for as you explore. Data exploration uses a combination of summary statistics—means and medians, variances, and counts— and visualization, or graphs of the data. You can spot some problems just by using summary statistics; other problems are easier to find visually.

Using summary statistics to spot problems

In R, you'll typically use the summary() command to take your first look at the data. The goal is to understand whether you have the kind of customer information that can potentially help you predict health insurance coverage, and whether the data is of good enough quality to be informative

```
customer_data <- read.csv("https://raw.githubusercontent.com/charlesmutai/kdjha/main/custdata.csv")
summary(customer_data)
```

```
Customer_data <- read.csv('https://raw.githubusercontent.com/charlesmutai/kdjha/main/custdata.csv')
head(Customer_data)
```

A data.frame: 6 × 13

	X	custid	sex	is_employed	income	marital_status	health_ins	housi
	<int>	<chr>	<chr>	<lgl>	<dbl>	<chr>	<lgl>	
1	7	000006646_03	Male	TRUE	22000	Never married	TRUE	Hoı free
2	8	000007827_01	Female	NA	23200	Divorced/Separated	TRUE	
3	9	000008359_04	Female	TRUE	21000	Never married	TRUE	Hoı mortg
4	10	000008529_01	Female	NA	37770	Widowed	TRUE	Hoı free
5	11	000008744_02	Male	TRUE	39000	Divorced/Separated	TRUE	

```
str(Customer_data)
```

```
'data.frame':   73262 obs. of  13 variables:
 $ X             : int  7 8 9 10 11 15 17 19 20 21 ...
 $ custid        : chr  "000006646_03" "000007827_01" "000008359_04" "000008529_01" ...
 $ sex           : chr  "Male" "Female" "Female" "Female" ...
 $ is_employed   : logi  TRUE NA TRUE NA TRUE NA ...
 $ income        : num  22000 23200 21000 37770 39000 ...
 $ marital_status: chr  "Never married" "Divorced/Separated" "Never married" "Widowed" ...
 $ health_ins    : logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
 $ housing_type  : chr  "Homeowner free and clear" "Rented" "Homeowner with mortgage/loan" "Homeowner free and clear" ...
 $ recent_move   : logi  FALSE TRUE FALSE FALSE FALSE FALSE ...
 $ num_vehicles  : int  0 0 2 1 2 2 2 2 5 3 ...
 $ age           : int  24 82 31 93 67 76 26 73 27 54 ...
 $ state_of_res  : chr  "Alabama" "Alabama" "Alabama" "Alabama" ...
 $ gas_usage     : int  210 3 40 120 3 200 3 50 3 20 ...
```

```
summary(Customer_data)
```

```
       X             custid              sex            is_employed
 Min.   :     7   Length:73262       Length:73262       Mode :logical
 1st Qu.: 24906   Class :character   Class :character   FALSE:2351
 Median : 49848   Mode  :character   Mode  :character   TRUE :45137
 Mean   : 49914                                        NA's :25774
 3rd Qu.: 74795
 Max.   :100000

     income        marital_status     health_ins      housing_type
 Min.   :  -6900   Length:73262       Mode :logical   Length:73262
 1st Qu.:  10700   Class :character   FALSE:7307      Class :character
 Median :  26200   Mode  :character   TRUE :65955     Mode  :character
 Mean   :  41764
 3rd Qu.:  51700
 Max.   :1257000

 recent_move      num_vehicles        age         state_of_res
 Mode :logical   Min.   :0.000   Min.   :  0.00   Length:73262
 FALSE:62418     1st Qu.:1.000   1st Qu.: 34.00   Class :character
 TRUE :9123      Median :2.000   Median : 48.00   Mode  :character
 NA's :1721      Mean   :2.066   Mean   : 49.16
                 3rd Qu.:3.000   3rd Qu.: 62.00
                 Max.   :6.000   Max.   :120.00
                 NA's   :1720
   gas_usage
 Min.   :  1.00
 1st Qu.:  3.00
 Median : 10.00
 Mean   : 41.17
 3rd Qu.: 60.00
 Max.   :570.00
 NA's   :1720
```

```
Customer_data$is_employed
```

```
TRUE · <NA> · TRUE · <NA> · TRUE · <NA> · TRUE · <NA> · TRUE · TRUE · <NA> · TRUE · <NA> · <NA> ·
<NA> · <NA> · TRUE · <NA> · TRUE · <NA> · TRUE · TRUE · <NA> · TRUE · <NA> · <NA> · TRUE · <NA> ·
TRUE · TRUE · <NA> · TRUE · TRUE · <NA> · <NA> · TRUE · <NA> · TRUE · <NA> · TRUE · TRUE · <NA> ·
TRUE · TRUE · <NA> · TRUE · TRUE · <NA> · TRUE · TRUE · <NA> · <NA> · <NA> · TRUE · TRUE · <NA> ·
TRUE · TRUE · <NA> · <NA> · <NA> · <NA> · TRUE · TRUE · TRUE · <NA> · <NA> · TRUE · TRUE · TRUE ·
<NA> · TRUE · <NA> · TRUE · TRUE · TRUE · TRUE · <NA> · TRUE · <NA> · <NA> · <NA> · TRUE · TRUE ·
```

```
Customer_data$income
```

```
22000 · 23200 · 21000 · 37770 · 39000 · 11100 · 25800 · 34600 · 25000 · 31200 · 0 · 40000 · 0 · 39270 ·
3000 · 38400 · 20200 · 8800 · 36000 · 22000 · 20000 · 25000 · 9700 · 120000 · 0 · 0 · 30000 · 19100 · 6000 ·
40000 · 5200 · 23000 · 5000 · 17700 · 0 · 140000 · 4200 · 6600 · 18890 · 45000 · 50000 · 12000 · 38000 ·
80000 · 8500 · 6e+05 · 60000 · 0 · 45000 · 52100 · 18800 · 66000 · 14400 · 78000 · 6400 · 12000 · 138000 ·
20000 · 58000 · 7800 · 2030 · 0 · 30000 · 82600 · 5000 · 46400 · 33900 · 35000 · 17000 · 60000 · 19800 ·
18000 · 19800 · 1e+05 · 71000 · 7200 · 18000 · 63300 · 4000 · 0 · 61200 · 12000 · 6800 · 17000 · 35310 ·
32300 · 50000 · 6000 · 25000 · 90000 · 18000 · 50000 · 48000 · 12000 · 8800 · 8400 · 40000 · 7200 ·
18600 · 0 · 57000 · 20240 · 2500 · 33000 · 24000 · 75000 · 125000 · 24000 · 42600 · 62000 · 120000 ·
55000 · 14000 · 72400 · 0 · 28000 · 23000 · 22800 · 20900 · 2500 · 56070 · 106000 · 32000 · 26900 ·
25000 · 39000 · 4000 · 70000 · 12700 · 13200 · 5100 · 170000 · 60000 · 12000 · 34000 · 0 · 16900 · 7200 ·
266000 · 70000 · 33850 · 142000 · 500 · 9000 · 0 · 33600 · 75000 · 84404 · 48450 · 71000 · 78000 · 2000 ·
112800 · 37700 · 25000 · 33400 · 39000 · 18300 · 25600 · 22100 · 0 · 0 · 38000 · 25000 · 70000 · 20000 ·
65000 · 25000 · 0 · 82000 · 10000 · 52000 · 36000 · 65000 · 19600 · 53000 · 23200 · 7500 · 192250 ·
85000 · 8400 · 0 · 8000 · 38000 · 52000 · 10200 · 28000 · 50000 · 36810 · 39900 · 540 · 35000 · 14000 ·
15000 · 0 · 90000 · 35000 · 8500 · 0 · 55000 · ··· · 78000 · 0 · 25000 · 37120 · 28200 · 38000 · 33000 · 2300 ·
47400 · 0 · 150000 · 78000 · 58000 · 40000 · 46000 · 18000 · 16000 · 48000 · 2000 · 30000 · 17100 ·
42000 · 13430 · 5500 · 50000 · 75000 · 20000 · 39500 · 52000 · 66000 · 14600 · 0 · 10100 · 2200 · 78000 ·
```

```
summary(Customer_data$income)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -6900   10700   26200   41764   51700 1257000
```

```
summary(Customer_data$age)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00   34.00   48.00   49.16   62.00  120.00
```

Double-click (or enter) to edit

- The variable is_employed is missing for about a third of the data. The variable income has negative values, which are potentially invalid.
- About 90% of the customers have health insurance
- The variables housing_type, recent_move, num_vehicles, and gas_usage are each missing 1720 or 1721 values.
- The average value of the variable age seems plausible, but the minimum and maximum values seem unlikely. The variable state_of_res is a categorical variable; summary() reports how many customers are in each state (for the first few states).

The summary() command on a data frame reports a variety of summary statistics on the numerical columns of the data frame, and count statistics on any categorical columns (if the categorical columns have already been read in as factors1). As you see, the summary of the data helps you quickly spot potential problems, like missing data or unlikely values. You also get a rough idea of how categorical data is distributed. Let's go into more detail about the typical problems that you can spot using the summary.

Typical problems revealed by data summaries

At this stage, you're looking for several common issues:

- Missing values
- Invalid values and outliers
- Data ranges that are too wide or too narrow
- The units of the data

MISSING VALUES

A few missing values may not really be a problem, but if a particular data field is largely unpopulated, it shouldn't be used as an input without some repair. In R, for example, many modeling algorithms will, by default, quietly drop rows with missing values. As you see in the following listing, all the missing values in the is_employed variable could cause R to quietly ignore more than a third of the data

Will the variable is_employed be useful for modeling?

- The variable is_employed is missing for more than a third of the data. Why? Is employment status unknown? Did the company start collecting employment data only recently? Does NA mean "not in the active workforce" (for example, students or stay-at-home parents)?
- The variables housing_type, recent_move, num_vehicles, and gas_usage are missing relatively few values— about 2% of the data. It's probably safe to just drop the rows that are missing values, especially if the missing values are all in the same 1720 rows.

If a particular data field is largely unpopulated, it's worth trying to determine why; sometimes the fact that a value is missing is informative in and of itself. For example, why is the is_employed variable missing so many values? There are many possible reasons. Whatever the reason for missing data, you must decide on the most appropriate action. Do you include a variable with missing values in your model, or not? If you decide to include it, do you drop all the rows where this field is missing, or do you convert the missing values to 0 or to an additional category? We'll discuss ways to treat missing data in topic 4. In this example, you might decide to drop the data rows where you're missing data about housing or vehicles, since there aren't many of them. You probably don't want to throw out the data where you're missing employment information, since employment status is probably highly predictive of having health insurance; you might instead treat the NAs as a third employment category. You will likely encounter missing values when model scoring, so you should deal with them during model training.

INVALID VALUES AND OUTLIERS

Even when a column or variable isn't missing any values, you still want to check that the values that you do have make sense. Do you have any invalid values or outliers? Examples of invalid values include negative values in what should be a non-negative numeric data field (like age or income) or text where you expect numbers. Outliers are data points that fall well out of the range of where you expect the data to be. Can you spot the outliers and invalid values in the next listing?

Examples of invalid values and outliers

```
1    summary(customer_data$income)
```

Negative values for income could indicate bad data. They might also have a special meaning, like "amount of debt." Either way, you should check how prevalent the issue is, and decide what to do. Do you drop the data with negative income? Do you convert negative values to zero?

```
summary(customer_data$age)
```

Customers of age zero, or customers of an age greater than about 110, are outliers. They fall out of the range of expected customer values. Outliers could be data input errors. They could be special sentinel values: zero might mean "age unknown" or "refuse to state." And some of your customers might be especially long-lived.

Often, invalid values are simply bad data input. A negative number in a field like age, however, could be a sentinel value to designate "unknown." Outliers might also be data errors or sentinel values. Or they might be valid but unusual data points—people do occasionally live past 100. As with missing values, you must decide the most appropriate action: drop the data field, drop the data points where this field is bad, or convert the bad data to a useful value. For example, even if you feel certain outliers are valid data, you might still want to omit them from model construction, if the outliers interfere with the model-fitting process. Generally, the goal of modeling is to make good predictions on typical cases, and a model that is highly skewed to predict a rare case correctly may not always be the best model overall

DATA RANGE

You also want to pay attention to how much the values in the data vary. If you believe that age or income helps to predict the probability of health insurance coverage, then you should make sure there is enough variation in the age and income of your customers for you to see the relationships. Let's look at income again, in the next listing. Is the data range wide? Is it narrow?

```
summary(customer_data$income)
```

Even ignoring negative income, the income variable ranges from zero to over a million dollars. That's pretty wide (though typical for income). Data that ranges over several orders of magnitude like this can be a problem for some modeling methods. We'll talk about mitigating data range issues when we talk about logarithmic transformations in topic 4. Data can be too narrow, too. Suppose all your customers are between the ages of 50 and 55. It's a good bet that age range wouldn't be a very good predictor of the probability of health insurance coverage for that population, since it doesn't vary much at all

One factor that determines apparent data range is the unit of measurement. To take a nontechnical example, we measure the ages of babies and toddlers in weeks or in months, because developmental changes happen at that time scale for very young children. Suppose we measured babies' ages in years. It might appear numerically that there isn't much difference between a one-year-old and a two-year-old. In reality, there's a dramatic difference, as any parent can tell you! Units can present potential issues in a dataset for another reason, as well.

UNITS

Does the income data in listing 3.5 represent hourly wages, or yearly wages in units of $1000? As a matter of fact, it's yearly wages in units of 1000, but what if it were hourly wages? You might not notice the error during the modeling stage, but down the line someone will start inputting hourly wage data into the model and get back bad predictions in return

```
IncomeK = customer_data$income/1000
summary(IncomeK)
```

The variable IncomeK is defined as IncomeK = customer_data $income/1000. But suppose you didn't know that. Looking only at the summary, 1000." the values could plausibly be interpreted to mean either "hourly wage" or "yearly income in units of

- Are time intervals measured in days, hours, minutes, or milliseconds?
- Are speeds in kilometers per second, miles per hour, or knots?
- Are monetary amounts in dollars, thousands of dollars, or 1/100 of a penny (a customary practice in finance, where calculations are often done in fixed-point arithmetic)?
- This is actually something that you'll catch by checking data definitions in data dictionaries or documentation, rather than in the summary statistics;
- the difference between hourly wage data and annual salary in units of $1000 may not look that obvious at a casual glance.
- But it's still something to keep in mind while looking over the value ranges of your variables, because often you can spot when measurements are in unexpected units.
- Automobile speeds in knots look a lot different than they do in miles per hour.

Spotting problems using graphics and visualization

As you've seen, you can spot plenty of problems just by looking over the data summaries. For other properties of the data, pictures are better than text.

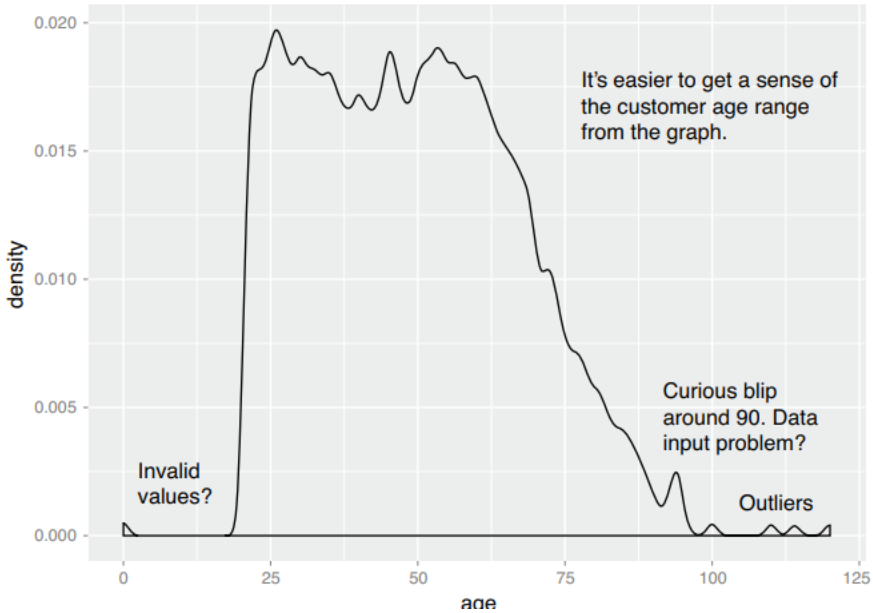
Figure below shows a plot of how customer ages are distributed. We'll talk about what the y-axis of the graph means later; for now, just know that the height of the graph corresponds to how many customers in the population are of that age. As you can see, information like the peak age of distribution, the range of the data, and the presence of outliers is easier to absorb visually than it is to determine textually. The use of graphics to examine data is called visualization.

A graphic should display as much information as it can, with the lowest possible cognitive strain to the viewer. Strive for clarity. Make the data stand out. Specific tips for increasing clarity include these:

- Avoid too many superimposed elements, such as too many curves in the same graphing space.
- Find the right aspect ratio and scaling to properly bring out the details of the data.
- Avoid having the data all skewed to one side or the other of your graph.

Visualization is an iterative process. Its purpose is to answer questions about the data.

```
summary(custdata$age)
```



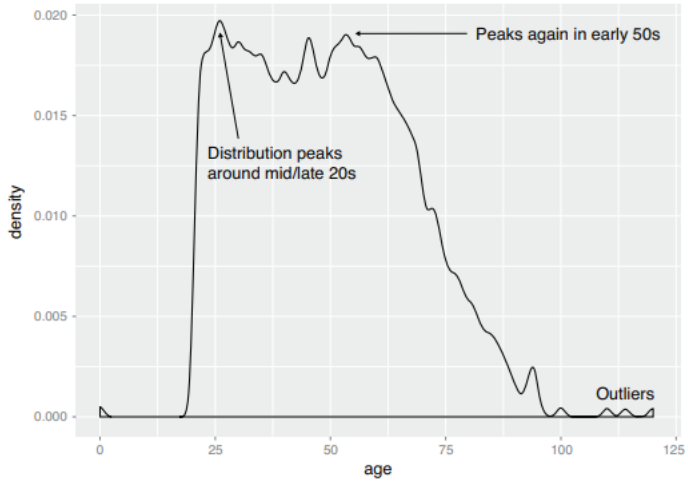
Visually checking distributions for a single variable

In this section we will look at

- Histograms
- Density plots
- Bar charts
- Dot plots

The visualizations in this section help you answer questions like these:

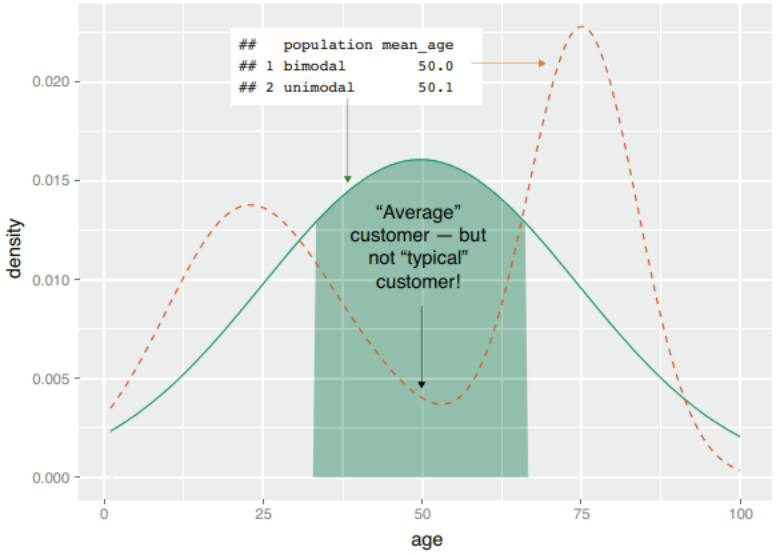
- What is the peak value of the distribution?
- How many peaks are there in the distribution (unimodality versus bimodality)?
- How normal (or lognormal) is the data? We'll discuss normal and lognormal distributions in later.
- How much does the data vary? Is it concentrated in a certain interval or in a certain category?



One of the things that's easy to grasp visually is the shape of the data distribution. The graph in figure above is somewhat flattish between the ages of about 25 and about 60, falling off slowly after 60. However, even within this range, there seems to be a peak at around the late-20s to early 30s range, and another in the early 50s. This data has multiple peaks: it is not unimodal.

Unimodality is a property you want to check in your data. Why? Because (roughly speaking) a unimodal distribution corresponds to one population of subjects. For the solid curve in figure 3.4, the mean customer age is about 50, and 50% of the customers are between 34 and 64 (the first and third quartiles, shown shaded). So you can say that a "typical" customer is middle-aged and probably possesses many of the demographic qualities of a middle-aged person—though, of course, you have to verify that with your actual customer information. The dashed curve in figure 3.4 shows what can happen when you have two peaks, or a bimodal distribution. (A distribution with more than two peaks is multimodal.) This set of customers has about the same mean age as the customers represented by the solid curve—but a 50-year-old is hardly a "typical" customer! This (admittedly exaggerated)

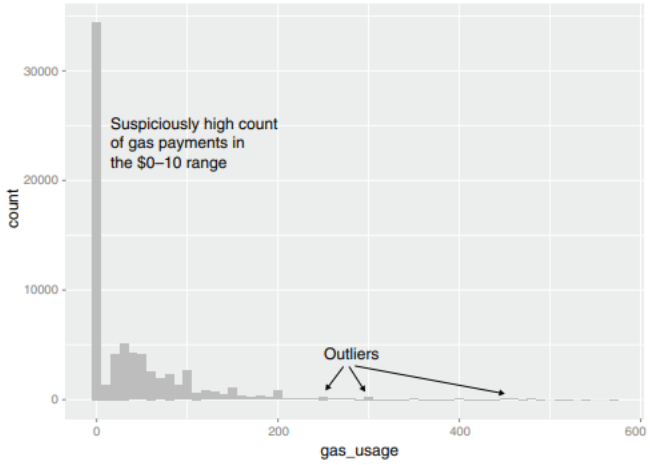
example corresponds to two populations of customers: a fairly young population mostly in their teens to late twenties, and an older population mostly in their 70s. These two populations probably have very different behavior patterns, and if you want to model whether a customer probably has health insurance or not, it wouldn't be a bad idea to model the two populations separately.



The histogram and the density plot are two visualizations that help you quickly examine the distribution of a numerical variable. Figures 3.1 and 3.3 are density plots. Whether you use histograms or density plots is largely a matter of taste. We tend to prefer density plots, but histograms are easier to explain to less quantitatively-minded audiences.

HISTOGRAMS

A basic histogram bins a variable into fixed-width buckets and returns the number of data points that fall into each bucket as a height. For example, suppose you wanted a sense of how much your customers pay in monthly gas heating bills. You could group the gas bill amounts in intervals of 10 :0–10, $10–20$,20–30, and so on. Customers at a boundary go into the higher bucket: people who pay around $20a month go into the 20–30$ bucket. For each bucket, you then count how many customers are in that bucket. The resulting histogram is shown in figure below




```
library(ggplot2)
ggplot(customer_data, aes(x=gas_usage)) +
geom_histogram(binwidth=10, fill="gray")
```

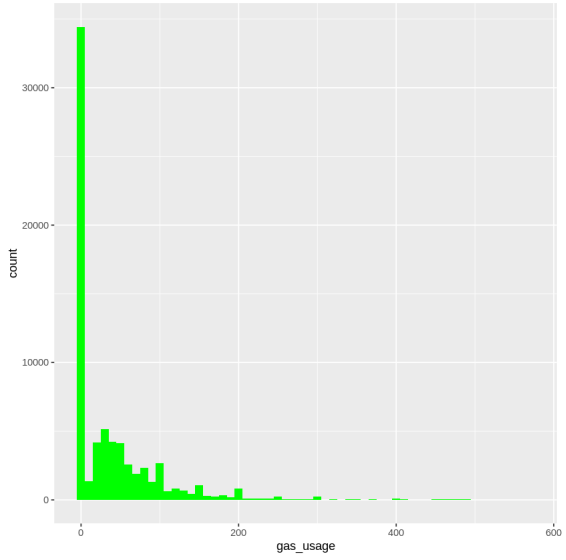
```
installed.packages('ggplot2')
```

A matrix:

Package	LibPath	Version	Priority	Depends	Imports	LinkingTo	Suggests	Enhances	Li

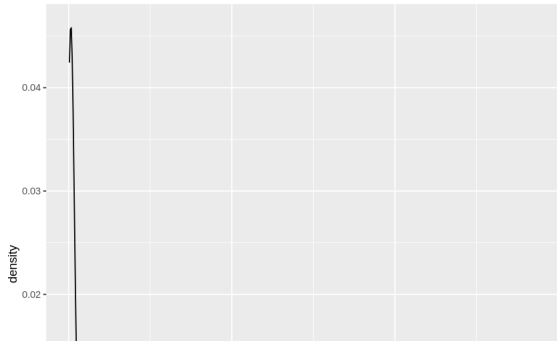
```
library(ggplot2)
ggplot(Customer_data, aes(x=gas_usage)) +
geom_histogram(binwidth=10, fill='green')
```

```
Warning message:
“Removed 1720 rows containing non-finite values (`stat_bin()`).”
```




```
library(scales)
ggplot(Customer_data, aes(x=gas_usage)) + geom_density() +
scale_x_continuous(labels=dollar)
```

```
Warning message:
“Removed 1720 rows containing non-finite values (`stat_density()`).”
```



The binwidth parameter tells the geom_histogram call how to make bins of $10 intervals (default is datarange/30). The fill parameter specifies the color of the histogram bars (default: black).

With the proper binwidth, histograms visually highlight where the data is concentrated, and point out the presence of potential outliers and anomalies. In figure, for example, you see that some outlier customers have much larger gas bills than is typical, so you may possibly want to drop those customers from any analysis that uses gas heating bills as an input. You also see an unusually high concentration of people who pay $0–10/month in gas. This could mean that most of your customers don't have gas heating, but on further investigation you notice this in the data dictionary (table)

Table 3.1 Data dictionary entry for `gas_usage`

Value	Definition
NA	Unknown or not applicable
001	Included in rent or condo fee
002	Included in electricity payment
003	No charge or gas not used
004–999	$4 to $999 (rounded and top-coded)

In other words, the values in the gas_usage column are a mixture of numerical values and symbolic codes encoded as numbers. The values 001, 002, and 003 are sentinel values, and to treat them as numerical values could potentially lead to incorrect conclusions in your analysis. One possible solution in this case is to convert the numeric values 1-3 into NA, and add additional Boolean variables to indicate the possible cases (included in rent/condo fee, and so on). The primary disadvantage of histograms is that you must decide ahead of time how wide the buckets are. If the buckets are too wide, you can lose information about the shape of the distribution. If the buckets are too narrow, the histogram can look too noisy to read easily. An alternative visualization is the density plot.

DENSITY PLOTS

You can think of a density plot as a continuous histogram of a variable, except the area under the density plot is rescaled to equal one. A point on a density plot corresponds to the fraction of data (or the percentage of data, divided by 100) that takes on a particular value. This fraction is usually very small. When you look at a density plot, you're more interested in the overall shape of the curve than in the actual values on the y-axis. You've seen the density plot of age; figure 3.6 shows the density plot of income.

The scales package brings in the dollar scale notation.

Sets the x-axis labels to dollars

```
library(scales)
ggplot(customer_data, aes(x=income)) + geom_density() +
scale_x_continuous(labels=dollar)
```

When the data range is very wide and the mass of the distribution is heavily concentrated to one side, like the distribution in figure 3.6, it's difficult to see the details of its shape. For instance, it's hard to tell the exact value where the income distribution has its peak. If the data is non-negative, then one way to bring out more detail is to plot the distribution on a logarithmic scale, as shown in figure 3.7. This is equivalent to plotting the density plot of log10(income)

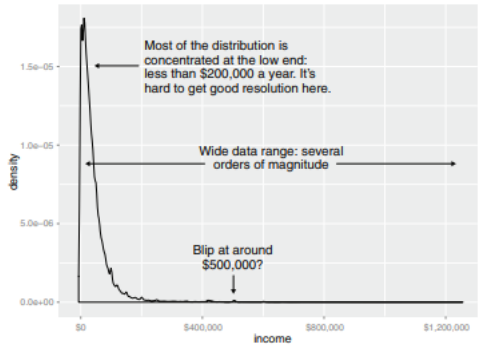


Figure 3.6 Density plots show where data is concentrated.

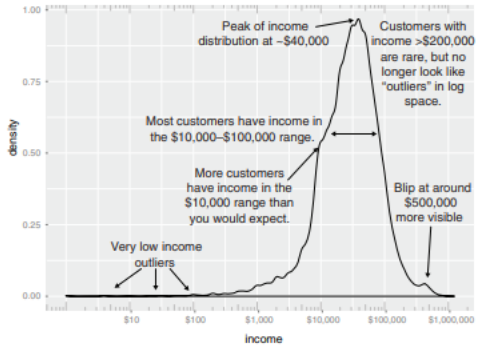


Figure 3.7 The density plot of income on a log10 scale highlights details of the income distribution that are harder to see in a regular density plot.

Creating a log-scaled density plot

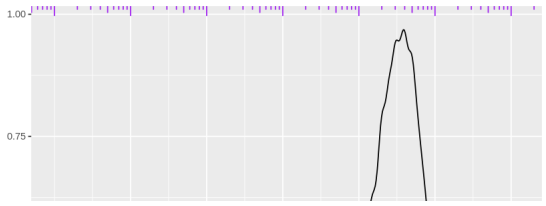
```
ggplot(customer_data, aes(x=income)) +
geom_density() +
scale_x_log10(breaks = c(10, 100, 1000, 10000, 100000, 1000000),
 labels=dollar) +
annotation_logticks(sides="bt", color="gray")
```

labels=dollar: Sets the x-axis to be in log10 scale, with manually set tick points and labels as dollars

color="gray":Adds log-scaled tick marks to the top and bottom of the graph

```
ggplot(Customer_data, aes(x=income)) +geom_density() +
scale_x_log10(breaks=c(10,100,1000,10000,100000,1000000),
labels=dollar) +
annotation_logticks(sides='bt', color= 'purple')
```

```
Warning message in self$trans$transform(x):
“NaNs produced”
Warning message:
“Transformation introduced infinite values in continuous x-axis”
Warning message:
“Removed 6856 rows containing non-finite values (`stat_density()`).”
```



Warning in self$trans$transform(x): NaNs produced

Warning: Transformation introduced infinite values in continuous x-axis

Warning: Removed 6856 rows containing non-finite values (stat_density).

This tells you that ggplot2 ignored the zero- and negative-valued rows (since log(0) = Infinity), and that there were 6856 such rows. Keep that in mind when evaluating the graph.

When should you use a logarithmic scale ?

You should use a logarithmic scale when percent change, or change in orders of magnitude, is more important than changes in absolute units. You should also use a log scale to better visualize data that is heavily skewed. For example, in income data, a difference in income of $5,000 means something very different in a population where the incomes tend to fall in the tens of thousands of dollars than it does in populations where income falls in the hundreds of thousands or millions of dollars. In other words, what constitutes a “significant difference” depends on the order of magnitude of the incomes you’re looking at. Similarly, in a population like that in figure 3.7, a few people with very high income will cause the majority of the data to be compressed into a relatively small area of the graph. For both those reasons, plotting the income distribution on a logarithmic scale is a good idea.

In log space, income is distributed as something that looks like a “normalish” distribution, as will be discussed in appendix B. It’s not exactly a normal distribution (in fact, it appears to be at least two normal distributions mixed together).

BAR CHARTS AND DOTPLOTS

A bar chart is a histogram for discrete data: it records the frequency of every value of a categorical variable. Figure 3.8 shows the distribution of marital status in your customer dataset. If you believe that marital status helps predict the probability of health insurance coverage, then you want to check that you have enough customers with different marital statuses to help you discover the relationship between being married (or not) and having health insurance.

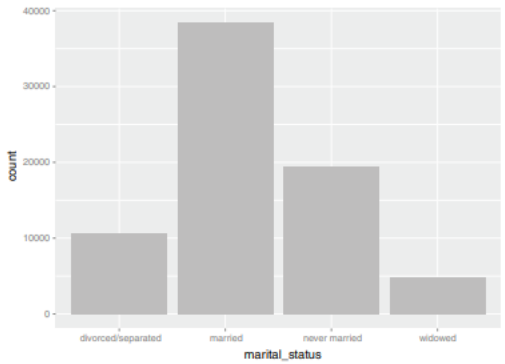
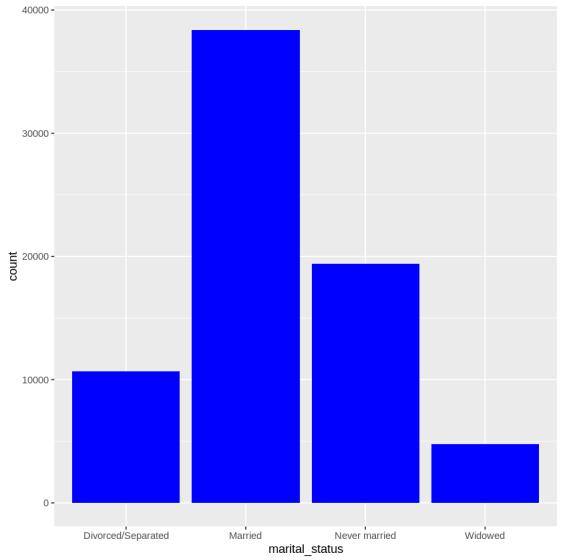


Figure 3.8 Bar charts show the distribution of categorical variables.

The ggplot2 command to produce figure 3.8 uses geom_bar:

```
ggplot(customer_data, aes(x=marital_status)) + geom_bar(fill="gray")
```

```
ggplot(Customer_data, aes(x=marital_status)) + geom_bar(fill='blue')
```

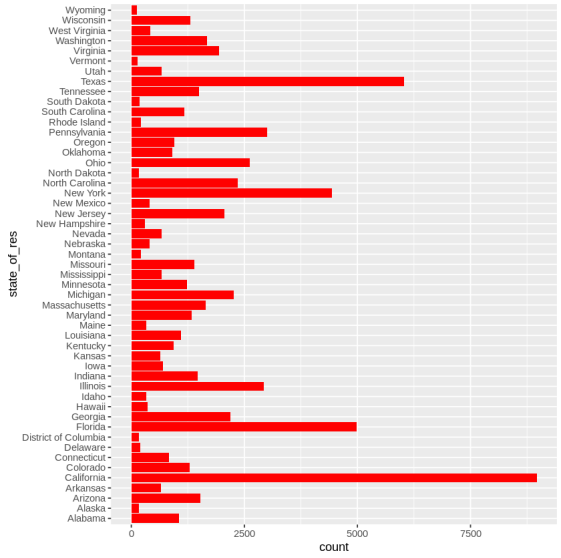


This graph doesn't really show any more information than summary(customer_ data$marital.stat) would show, but some people find the graph easier to absorb than the text. Bar charts are most useful when the number of possible values is fairly large, like state of residence. In this situation, we often find that a horizontal graph like that shown in figure 3.9 is more legible than a vertical graph.

The ggplot2 command to produce figure 3.9 is shown in the next listing.

```
ggplot(customer_data, aes(x=state_of_res)) +
geom_bar(fill="gray") +
coord_flip()
```

```
ggplot(Customer_data, aes(x=state_of_res)) +
geom_bar(fill='red') +
coord_flip()
```



- coord_flip(): Flips the x and y axes: state_of_res is now on the y-axis
- geom_bar(fill="gray") +: Plots bar chart as before: state_of_res is on x-axis, count is on y-axis

recommends that the data in a bar chart or dot plot be sorted, to more efficiently extract insight from the data. This is shown in figure 3.10. Now it is easy to see in which states the most customers—or the fewest—live.

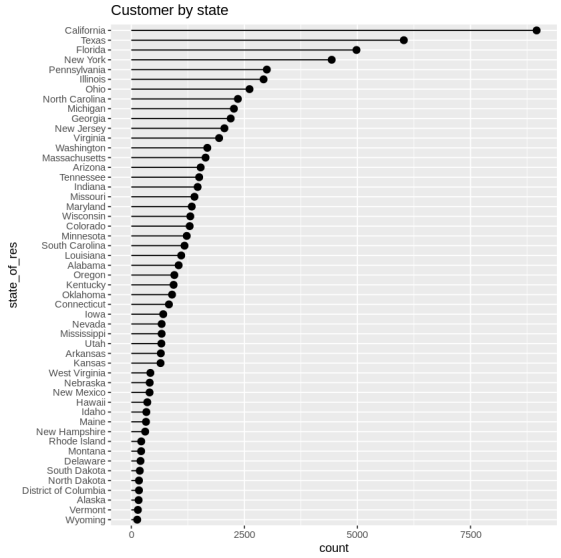
Listing 3.10 Producing a dot plot with sorted categories

```
#install.packages('WVPlots')
library(WVPlots)
ClevelandDotPlot(customer_data, "state_of_res",
sort = 1, title="Customers by state") +
coord_flip()
```

```
install.packages('WVPlots')
```

```
    Installing package into ‘/usr/local/lib/R/site-library’
    (as ‘lib’ is unspecified)
```

```
library(WVPlots)
ClevelandDotPlot(Customer_data, 'state_of_res',
sort =1, title='Customer by state') +
coord_flip()
```



- “sort = 1” sorts the categories in increasing order (most frequent last).
- ClevelandDotPlot(customer_data, "state_of_res",:Plots the state_of_res column of the customer_data data frame

summary of the visualizations that we’ve discussed so far:

Table 3.2 Visualizations for one variable

Graph type	Uses	Examples
Histogram or density plot	Examine data range Check number of modes Check if distribution is normal/ lognormal Check for anomalies and outliers	Examine the distribution of customer age to get the typical customer age range Examine the distribution of customer income to get typical income range
Bar chart or dot plot	Compare frequencies of the values of a categorical variable	Count the number of customers from different states of residence to determine which states have the largest or smallest customer base

Visually checking relationships between two variables

In addition to examining variables in isolation, you'll often want to look at the relationship between two variables. For example, you might want to answer questions like these:

- Is there a relationship between the two inputs age and income in my data?
- If so, what kind of relationship, and how strong?
- Is there a relationship between the input marital status and the output health insurance? How strong

You'll precisely quantify these relationships during the modeling phase, but exploring them now gives you a feel for the data and helps you determine which variables are the best candidates to include in a model

This part explores the following visualizations: *Line plots and scatter plots for comparing two continuous variables

- Smoothing curves and hexbin plots for comparing two continuous variables at high volume
- Different types of bar charts for comparing two discrete variables
- Variations on histograms and density plots for comparing a continuous and discrete variable

consider the relationship between two continuous variables.

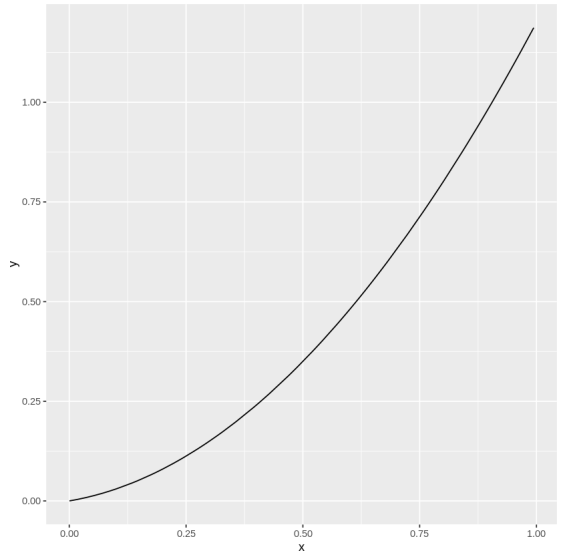
LINE PLOTS

Line plots work best when the relationship between two variables is relatively clean: each x value has a unique (or nearly unique) y value, as in figure 3.11. You plot figure 3.11 with geom_line.

Listing 3.11 Producing a line plot

```
x <- runif(100)
y <- x^2 + 0.2*x
ggplot(data.frame(x=x,y=y), aes(x=x,y=y)) + geom_line()
```


```
x <- runif(100)
y <- x^2 + 0.2*x
ggplot(data.frame(x=x,y=y), aes(x=x,y=y)) + geom_line()
```

SCATTER PLOTS AND SMOOTHING CURVES

You'd expect there to be a relationship between age and health insurance, and also a relationship between income and health insurance. But what is the relationship between age and income? If they track each other perfectly, then you might not want to use both variables in a model for health insurance. The appropriate summary statistic is the correlation, which we compute on a safe subset of our data

Listing 3.12 Examining the correlation between age and income

```
customer_data2 <- subset(customer_data,
0 < age & age < 100 &
0 < income & income < 200000)
```

- 0 < income & income < 200000): Only consider a subset of data with reasonable age and income values.

```
customer_data2 <- subset(Customer_data,
0 < age & age < 100 &
0< income & income <200000)
```

```
cor(customer_data2$age, customer_data2$income)
```

```
cor(customer_data2 $ age, customer_data2 $ income)
```

```
0.00576669679009915
```

The correlation is positive, as you might expect, but nearly zero, meaning there is apparently not much relation between age and income. A visualization gives you more insight into what's going on than a single number can. Let's try a scatter plot first (figure 3.12). Because our dataset has over 64,000 rows, which is too large for a legible scatterplot, we will sample the dataset down before plotting.

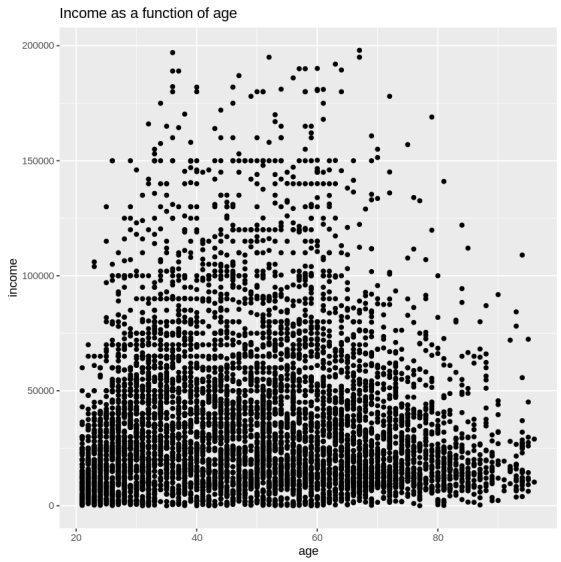
Listing 3.13 Creating a scatterplot of age and incom

```
set.seed(245566)
customer_data_samp <-
dplyr::sample_frac(customer_data2, size=0.1, replace=FALSE)
```

```
set.seed(245566)
customer_data_samp <-
dplyr:: sample_frac(customer_data2, size= 0.1, replace=FALSE)
```

```
ggplot(customer_data_samp, aes(x=age, y=income)) +
geom_point() +
ggtitle("Income as a function of age")
```

```
ggplot(customer_data_samp, aes(x=age, y= income)) +
geom_point() +
ggtitle('Income as a function of age')
```

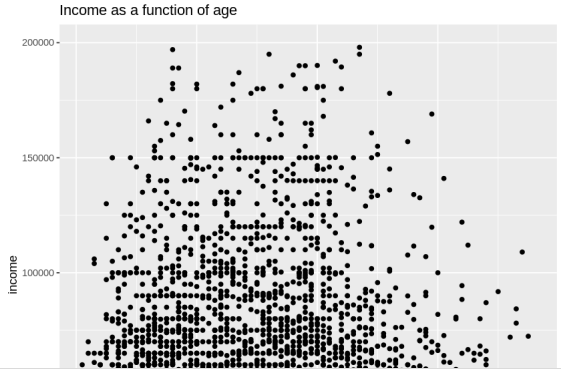


The relationship between age and income isn't easy to see. You can try to make the relationship clearer by also plotting a smoothing curve through the data, as shown in figure 3.13. The smoothing curve makes it easier to see that in this population, income tends to increase with age from a person's twenties until their mid-thirties, after which income increases at a slower, almost flat, rate until about a person's mid-fifties. Past the midfifties, income tends to decrease with age

```
ggplot(customer_data_samp, aes(x=age, y=income)) +
geom_point() + geom_smooth() +
ggtitle("Income as a function of age")
```

```
ggplot(customer_data_samp, aes(x=age, y=income)) +
geom_point() + geom_smooth() +
ggtitle('Income as a function of age')
```

```
`geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```



✓ 0s completed at 3:34 PM