A M.O.D.E.R.N. D.I.V.E. into the Language of Science

Modern Open-source Data-analysis Encouraging Reproducibility and iNtuitive Data-Inspired Visualization Efforts

Chester Ismay and Albert Y. Kim

2016-07-24

Contents

Chapter 1

Prerequisites

This book was written using the **bookdown** R package from Yihui Xie. In order to follow along and run the code in this book on your own, you'll need to have access to R (and preferably RStudio). You can find more information on both of these with a simple Google search for "R" and for "RStudio".

We will keep a running list of R packages you will need to have installed to complete the analysis as well here in the `needed_pkgs` character vector. You can check if you have all of the needed packages installed by running all of the lines below. The last line will install them (i.e., download their needed files from the internet to your hard drive).

You can run the `library` function on them to load them into your current analysis. Prior to each analysis where a package is needed you will see the corresponding `library` function.

```
needed_pkgs <- c("nycflights13", "dplyr", "ggplot2", "knitr", "devtools")
new.pkgs <- needed_pkgs[!(needed_pkgs %in% installed.packages())]

if (length(new.pkgs)) {
  install.packages(new.pkgs, repos = "http://cran.rstudio.com")
}
```

1.1 Colophon

The source of the book is available at and was built with versions of R packages given here:

```
devtools::session_info(needed_pkgs)
```

```
## Session info -------------------------------------------------------------

##  setting  value
##  version  R version 3.3.0 (2016-05-03)
##  system   x86_64, darwin13.4.0
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  tz       America/Los_Angeles
##  date     2016-07-24

## 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)
##  colorspace     1.2-6    2015-03-11 CRAN (R 3.3.0)
##  curl           0.9.7    2016-04-10 CRAN (R 3.3.0)
##  DBI            0.4-1    2016-05-08 CRAN (R 3.3.0)
##  devtools       1.12.0   2016-06-24 CRAN (R 3.3.0)
##  dichromat      2.0-0    2013-01-24 CRAN (R 3.3.0)
##  digest         0.6.9    2016-01-08 CRAN (R 3.3.0)
##  dplyr        * 0.5.0    2016-06-24 CRAN (R 3.3.0)
##  evaluate       0.9      2016-04-29 CRAN (R 3.3.0)
##  formatR        1.4      2016-05-09 CRAN (R 3.3.0)
##  ggplot2        2.1.0    2016-03-01 CRAN (R 3.3.0)
##  git2r          0.15.0   2016-05-11 CRAN (R 3.3.0)
##  gtable         0.2.0    2016-02-26 CRAN (R 3.3.0)
##  highr          0.6      2016-05-09 CRAN (R 3.3.0)
##  httr           1.2.1    2016-07-03 CRAN (R 3.3.0)
##  jsonlite       1.0      2016-07-01 cran (@1.0)
##  knitr        * 1.13     2016-05-09 CRAN (R 3.3.0)
##  labeling       0.3      2014-08-23 CRAN (R 3.3.0)
##  lazyeval       0.2.0    2016-06-12 CRAN (R 3.3.0)
##  magrittr       1.5      2014-11-22 CRAN (R 3.3.0)
##  markdown       0.7.7    2015-04-22 CRAN (R 3.3.0)
##  MASS           7.3-45   2016-04-21 CRAN (R 3.3.0)
##  memoise        1.0.0    2016-01-29 CRAN (R 3.3.0)
##  mime           0.5      2016-07-07 CRAN (R 3.3.0)
##  munsell        0.4.3    2016-02-13 CRAN (R 3.3.0)
##  nycflights13 * 0.2.0    2016-04-30 CRAN (R 3.3.0)
##  openssl        0.9.4    2016-05-25 CRAN (R 3.3.0)
##  plyr           1.8.4    2016-06-08 CRAN (R 3.3.0)
##  R6             2.1.2    2016-01-26 CRAN (R 3.3.0)
##  RColorBrewer   1.1-2    2014-12-07 CRAN (R 3.3.0)
##  Rcpp           0.12.6   2016-07-19 CRAN (R 3.3.0)
##  reshape2       1.4.1    2014-12-06 CRAN (R 3.3.0)
##  rstudioapi     0.6      2016-06-27 CRAN (R 3.3.0)
##  scales         0.4.0    2016-02-26 CRAN (R 3.3.0)
##  stringi        1.1.1    2016-05-27 CRAN (R 3.3.0)
##  stringr        1.0.0    2015-04-30 CRAN (R 3.3.0)
##  tibble         1.1      2016-07-04 CRAN (R 3.3.0)
##  whisker        0.3-2    2013-04-28 CRAN (R 3.3.0)
##  withr          1.0.2    2016-06-20 CRAN (R 3.3.0)
##  yaml           2.1.13   2014-06-12 CRAN (R 3.3.0)
```

Chapter 2

Introduction

2.1 Preamble

- Coggle Diagrams
- Lean on visualizations as much as possible first to introduce summary measures
- We will focus on the triad: computational, data, and inferential thinking.

2.2 Data/science pipeline

2.3 Reproducibility

2.4 Who is this book for? (Target)

Students taking a traditional intro stats class in a small college environment using RStudio preferably RStudio Server.

We assume no prerequisites: no calculus and no prior programming experience.

2.5 Algorithmic Thinking

Despite what you may think, computers are stupid. You need to explicitly tell it everything it needs to do; if you make even a slight mistake, it will cry.

To think about further

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 2. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter **??**.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 2.1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 2.1.

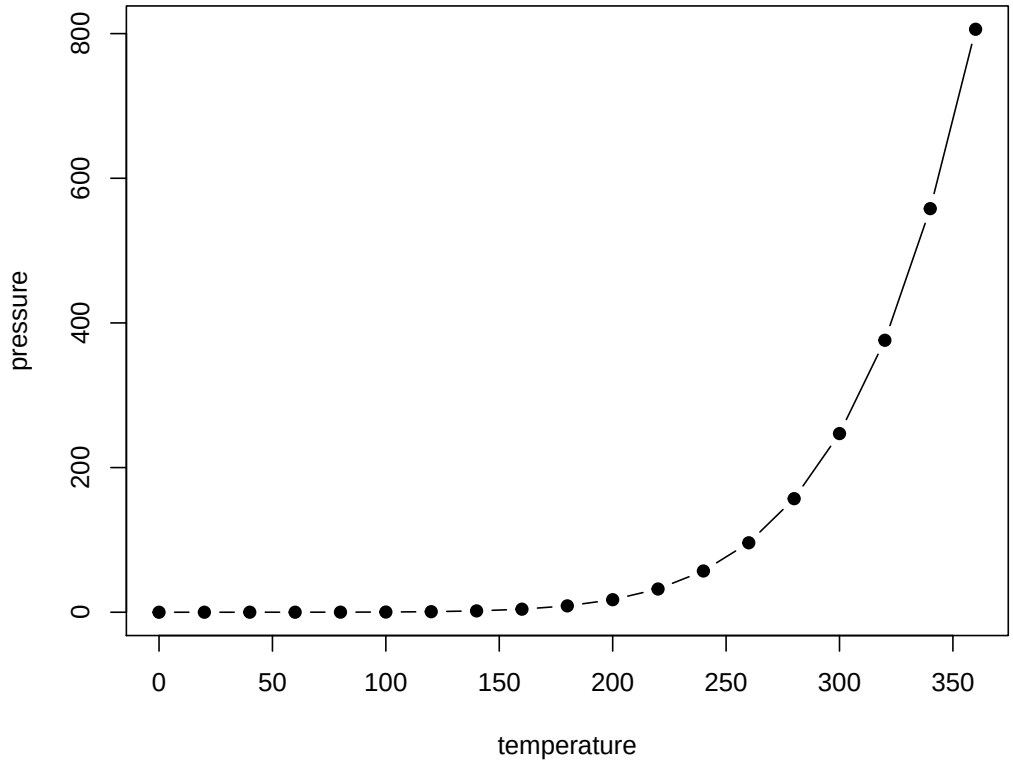

Figure 2.1: Here is a nice figure!

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

Table 2.1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

Chapter 3

Tidy data

nycflights13 example

- External link: based on normal forms

Need to give big picture question here and set up how this chapter ties in to chapters to come.

You have surely heard the word "tidy" in your life:

- "Tidy up your room!"
- "Please write your homework in a tidy way so that it is easier to grade and provide feedback."
- Marie Kondo's best-selling book *The Life-Changing Magic of Tidying Up: The Japanese Art of Decluttering and Organizing*
- "I am not by any stretch of the imagination a tidy person, and the piles of unread books on the coffee table and by my bed have a plaintive, pleading quality to me - 'Read me, please!' " - Linda Grant

So what does it mean for your data to be **tidy**? Put simply: it means that your data is organized. But it's more than just that. It means that your data follows the same standard format making it easy for others to find elements of your data, to manipulate and transform your data, and for our purposes continuing with the common theme: it makes it easier to visualize your data and the relationships between different variables in your data.

3.1 Definition

We will follow Hadley Wickham's definition of **tidy data** here (Wickham, 2014):

> A dataset is a collection of values, usually either numbers (if quantitative) or strings (if qualitative). Values are organised in two ways. Every value belongs to a variable and an observation. A variable contains all values that measure the same underlying attribute (like height, temperature, duration) across units. An observation contains all values measured on the same unit (like a person, or a day, or a race) across attributes.
>
> Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types. In **tidy data**:
>
> 1. Each variable forms a column.
> 2. Each observation forms a row.
> 3. Each type of observational unit forms a table.

Reading over this definition, you can begin to think about data sets that won't follow this nice format.

3.1.1 Learning check

- Give an example dataset that doesn't follow this format.
  - What features of this dataset might make it difficult to visualize?
  - How could the dataset be tweaked to make it **tidy**?

3.2 The `nycflights13` dataset

We likely have all flown on airplanes or know someone that has. Air travel has become an ever-present aspect of our daily lives. If you live in or are visiting a relatively large city and you walk around that city's airport, you see gates showing flight information from many different airlines. And you will frequently see that some flights are delayed because of a variety of conditions. Are there ways that we can avoid having to deal with these flight delays?

We'd all like to arrive at our destinations on time whenever possible. (Unless you secretly love hanging out at airports. If you are one of these people, pretend for the moment that you are very much anticipating being at your final destination.) Hadley Wickham (herein just referred to as "Hadley") created a dataset containing information about departing flights from the New York City area in 2013 (Wickham, 2016). We will begin by loading in this dataset and getting an idea of the structure of the dataset:

```
library(nycflights13)
data(flights)
```

The `library` function here loads the R package `nycflights13` into the current R environment in which you are working. (Note that you'll get an error if you try to load this package in and it hasn't been installed. Check Chapter 2 to make sure the package has been downloaded to your computer.) The next line of code `data(flights)` loads in the `flights` dataset that is stored in the `nycflights13` package.

This dataset and most others presented in this book will be in the `data.frame` format in R. Data frames are ways to look at collections of variables that are tightly coupled together. We next begin with a couple useful R functions to get a sense for what the `flights` dataset looks like:

```
head(flights)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     1     1      517            515         2      830
## 2  2013     1     1      533            529         4      850
## 3  2013     1     1      542            540         2      923
## 4  2013     1     1      544            545        -1     1004
## 5  2013     1     1      554            600        -6      812
## 6  2013     1     1      554            558        -4      740
## # ... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <time>
```

3.2.1 Learning check

- What does the `head` function give us? Why might it be a useful function to run on a dataset you have been presented with?
- What do you think the `tail` function would give us for the `flights` dataset?

- What does any *ONE* row in this dataset refer to? A. Data on an airline B. Data on a flight C. Data on an airport D. Data on multiple flights

We see that the `head` function gives us the first six rows (by default) for this dataset. This can give us an idea of what to expect our dataset to look like. For example, we see the different **variables** listed in the columns and we see that there are different types of variables. Some of the variables like `distance`, `day`, and `arr_delay` are what we will call **quantitative** variables. These variables vary in a numerical way. Other variables here are **categorical**.

Note that if you look in the leftmost portion near the `##` of the R output, you will see a column of numbers. These are the row numbers of the dataset. If you glance across a row with the same number, say row 5, you can get an idea of what each row correspond to. In other words, this will allow you to identify what object is being referred to in a given row. This is often called the **observational unit**. The **observational unit** in this example is an individual flight departing New York City in 2013.

Note: Frequently the first thing you should do when given a dataset is to

- identify the observation unit,
- specify the variables, and
- give the types of variables you are presented with.

```
str(flights)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    336776 obs. of  19 variables:
##  $ year          : int  2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
##  $ month         : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ day           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ dep_time      : int  517 533 542 544 554 554 555 557 557 558 ...
##  $ sched_dep_time: int  515 529 540 545 600 558 600 600 600 600 ...
##  $ dep_delay     : num  2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
##  $ arr_time      : int  830 850 923 1004 812 740 913 709 838 753 ...
##  $ sched_arr_time: int  819 830 850 1022 837 728 854 723 846 745 ...
##  $ arr_delay     : num  11 20 33 -18 -25 12 19 -14 -8 8 ...
##  $ carrier       : chr  "UA" "UA" "AA" "B6" ...
##  $ flight        : int  1545 1714 1141 725 461 1696 507 5708 79 301 ...
##  $ tailnum       : chr  "N14228" "N24211" "N619AA" "N804JB" ...
##  $ origin        : chr  "EWR" "LGA" "JFK" "JFK" ...
##  $ dest          : chr  "IAH" "IAH" "MIA" "BQN" ...
##  $ air_time      : num  227 227 160 183 116 150 158 53 140 138 ...
##  $ distance      : num  1400 1416 1089 1576 762 ...
##  $ hour          : num  5 5 5 5 6 5 6 6 6 6 ...
##  $ minute        : num  15 29 40 45 0 58 0 0 0 0 ...
##  $ time_hour     : POSIXct, format: "2013-01-01 05:00:00" "2013-01-01 05:00:00" ...
```

3.2.2 Learning check

- What are some examples in this data set of **categorical** variables? What makes them different than **quantitative** variables?
- What does `int`, `num`, and `chr` mean in the output above?
- How many different columns are in this dataset?
- How many different rows are in this dataset?

Another way to view the properties of a dataset is to use the `str` function ("str" is short for "structure"). This will give you the first few entries of each variable in a row after the variable. In addition, the type of the variable is given immediately after the `:` following each variable's name. Here, `int` and `num` refer to quantitative variables. In contrast, `chr` refers to categorical variables. One more type of variable is given

here with the `time_hour` variable: **POSIXct**. As you may suspect, this variable corresponds to a specific date and time of day.

Another nice feature of R is the help system. You can get help in R by simply entering a question mark before the name of a function or an object and you will be presented with a page showing the documentation. Note that this output help file is omitted here but can be accessed here on page 3 of the PDF document.

```
?flights
```

Another aspect of tidy data is a description of what each variable in the dataset represents. This helps others to understand what your variable names mean and what they correspond to. If we look at the output of `?flights`, we can see that a description of each variable by name is given.

3.3 How is `flights` tidy?

We see that `flights` has a rectangular shape with each row corresponding to a different flight and each column corresponding to a characteristic of that flight. This matches exactly with how Hadley defined tidy data:

1. Each variable forms a column.
2. Each observation forms a row.

But what about the third property?

3. Each type of observational unit forms a table.

We identified earlier that the observational unit in the `flights` dataset is an individual flight. And we have shown that this dataset consists of 336776 flights with 19 variables. In other words, some rows of this dataset don't refer to a measurement on an airline or on an airport. They specifically refer to characteristics/measurements on a given **flight** from New York City in 2013.

By contrast, also included in the `nycflights13` package are datasets with different observational units (Wickham, 2016):

- `weather`: hourly meterological data for each airport
- `planes`: construction information about each plane
- `airports`: airport names and locations
- `airlines`: translation between two letter carrier codes and names

You may have been asking yourself what `carrier` refers to in the `str(flights)` output above. The `airlines` dataset provides a description of this with each airline being the observational unit:

```
data(airlines)
airlines
```

```
## # A tibble: 16 x 2
##    carrier                        name
##      <chr>                       <chr>
## 1       9E           Endeavor Air Inc.
## 2       AA      American Airlines Inc.
## 3       AS        Alaska Airlines Inc.
## 4       B6             JetBlue Airways
## 5       DL        Delta Air Lines Inc.
## 6       EV    ExpressJet Airlines Inc.
## 7       F9      Frontier Airlines Inc.
## 8       FL AirTran Airways Corporation
## 9       HA      Hawaiian Airlines Inc.
## 10      MQ                  Envoy Air
```

```
## 11      OO       SkyWest Airlines Inc.
## 12      UA       United Air Lines Inc.
## 13      US             US Airways Inc.
## 14      VX              Virgin America
## 15      WN      Southwest Airlines Co.
## 16      YV          Mesa Airlines Inc.
```

```
library(knitr)
kable(airlines)
```

carrier	name
9E	Endeavor Air Inc.
AA	American Airlines Inc.
AS	Alaska Airlines Inc.
B6	JetBlue Airways
DL	Delta Air Lines Inc.
EV	ExpressJet Airlines Inc.
F9	Frontier Airlines Inc.
FL	AirTran Airways Corporation
HA	Hawaiian Airlines Inc.
MQ	Envoy Air
OO	SkyWest Airlines Inc.
UA	United Air Lines Inc.
US	US Airways Inc.
VX	Virgin America
WN	Southwest Airlines Co.
YV	Mesa Airlines Inc.

Note that R by default will print out the object when only its name is given as we have done here with `airlines`. If we'd prefer to print a data frame out in a clean format we can use the `kable` function in the `knitr` R package.

3.3.1 Review questions

1. What are common characteristics of "tidy" datasets?
2. What makes them useful for organizing data?
3. How many variables are presented in the table below? What does each row correspond to? (**Hint:** You may not be able to answer both of these questions immediately but take your best guess.)
4. What would the code `kable(head(flights))` produce?

students	faculty
4	2
6	3

4. The confusion you may have encountered in Question 3 is a common one those that work with data are commonly presented with. This dataset is not tidy. Actually, the dataset in Question 3 has three variables not the two that were presented. Make a guess as to what these variables are and present a tidy data set instead of this untidy one given in Question 3.

5. The actual data presented in Question 3 is given below in tidy data format:

role	Sociology?	Type of School
student	TRUE	Public
student	TRUE	Public
student	TRUE	Public
student	TRUE	Public
student	FALSE	Public
student	FALSE	Public
student	FALSE	Private
student	FALSE	Private
student	FALSE	Private
student	FALSE	Private
faculty	TRUE	Public
faculty	TRUE	Public
faculty	FALSE	Public
faculty	FALSE	Private
faculty	FALSE	Private

- What does each row correspond to?
- What are the different variables in this data frame?
- The `Sociology?` variable is known as a logical variable. What types of values does a logical variable take on?

3.4 What's to come?

In Chapter 4, we will further explore the distribution of the variable in this data frame that refers to departure delays for flights. We'll be interested in understanding how this variable varies in relation to the values of other variables in the dataset. We will see that visualization is often a powerful tool in helping us see what is going on in a dataset. It will be a useful way to expand on the `str` function we have seen here for tidy data.

Last updated:

```
## [1] "Sunday, July 24, 2016 19:15:35 PDT"
```

Chapter 4

Visualizing Data

In Chapter 3, we discussed the importance of datasets being **tidy**. You will see in examples here why having a tidy dataset helps us immensely with plotting our data. We will focus on using Hadley's `ggplot2` package in doing so, which was developed to work specifically on datasets that are **tidy**. It provides an easy way to customize your plots and is based on data visualization theory given in *The Grammar of Graphics* (Wilkinson, 2005).

4.1 Five Named Graphs. FNG

- Focussing on `dep_delay` here and how it relates with other variables
- May need to introduce some data manipulation here with `dplyr`
- Restrict types of plots:
  - histogram: as long as we avoid density plots
  - boxplot: have to set x=1 aes. categorical variables when univariate
    - Hadley suggests that a univariate boxplot shouldn't be used and prefers a histogram https://groups.google.com/forum/#!topic/ggplot2/N4qRSmqMZxI
    - Boxplots do make sense when comparing distributions across levels of another variable though
  - barplot: issue with categorical variables
    - May want to also discuss dot plots for many levels of categorical variable
  - scatterplot: jitter/alpha/color
  - linegraph: useful for time series / not useful if no ordering to x-values
  - faceting: show why better than stacked bargraphs in via patterns across levels
  - mosaicplot [I actually prefer side-by-side bar graphs/facetted bargraphs to stacked/mosaic so this might be moot?]

4.1.1 Review questions

- Have a variety of bad plots with data for the readers and have readers create better plots with `ggplot2`

Chapter 5

Manipulating Data

We describe our methods in this chapter.

Chapter 6

Inference

Topics

- (random) Sampling: representativeness/generalizability/bias
-

Bibliography

Wickham, H. (2014). Tidy data. *Journal of Statistical Software*, Volume 59(Issue 10).

Wickham, H. (2016). *nycflights13: Flights that Departed NYC in 2013*. R package version 0.2.0.

Wilkinson, L. (2005). *The Grammar of Graphics (Statistics and Computing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.