Introduction to R for data analysis

Peter Carbonetto

Research Computing Center and the Dept. of Human Genetics
University of Chicago

2. Aims of workshop

1. Get hands-on experience with the basic elements of data analysis in R.
2. Understand how to import data from a CSV file into an R data frame.
3. Use standard tools to summarize & manipulate data frames.
4. Learn how to install & use R packages.
5. Use ggplot2 to plot data.
6. Learn through “live coding”.

3. Our goal: Analyze Divvy data from 2016 & 2017

- Investigate bike sharing trends in Chicago.
- We will use data made available by Divvy:
  - ▷ www.divvybikes.com/system-data
- We will import and inspect the data, and take steps to prepare the data for analysis and plotting.
- Once we have carefully prepared the data, creating visualizations is (relatively) little effort.

4. The programmatic approach

- Data analysis usually involves *iterative refinement* and *repetition*.
- The *programmatic approach* to data analysis will allow you to...
  - ▷ Automate your analysis.
  - ▷ Quickly create a new analysis from existing code.
  - ▷ Expand capabilities with R packages.

5. It's your choice

Your may choose to...

- Use R on your computer.
- Use RStudio on your computer.
- Use RStudio Cloud.
- Follow what I do on the projector.

6. Software we will use today

1. **R** and/or **RStudio**.
2. R packages **readr**, **ggplot2** & **cowplot**.

7. Outline of workshop

1. Initial setup.
2. Analysis of Divvy station data.
3. Analysis of Divvy trip data.
4. Combining the data.

8. Initial setup

- Set up RStudio Cloud (optional).
- Workshop packet.
- Reading what I type.
- Pace, questions (e.g., keyboard shortcuts).
- Help.

9. Set up RStudio Cloud (optional)

- Go to **https://rstudio.cloud**.
- If necessary, log in or create an account.
- In Your Workspace, select **New Project > New Project From Git Repo**.
- Enter this URL:

`https://github.com/rcc-uchicago/R-intro-divvy`

10. Download or “clone” git repository

If not using RStudio Cloud, download the workshop packet to your computer.

- Go to **http://github.com/rcc-uchicago/R-intro-divvy**
- To download, click the green “**Clone or download**” button.

Or, if you have **git**, run this command:

```
git clone https://github.com/rcc-uchicago/
  R-intro-divvy.git
```

(Note the URL in the git command should not contain any spaces.)

- If necessary, uncompress the ZIP file.
- If necessary, rename folder to **R-intro-divvy**.

11. What's included in the workshop packet

- **slides.pdf**: These slides.
- **slides.Rmd**: R Markdown source used to generate these slides. *You may open this file in RStudio or your favourite editor.*
- **read_trip_data.R**: Some R code used in the examples.
- **Divvy_Stations_2017_Q3Q4.csv:** Divvy station data.
- **Divvy_Trips_2019_Q4.csv.gz:** 2019 Divvy trip data.

12. Set up your R environment

- Launch R or RStudio.

13. Load code for the hands-on exercises

Open R Markdown source file, **slides.Rmd**.

- In RStudio, select **File > Open File**.
- Alternatively, use your favourite text editor.

14. Run `sessionInfo()`

Check the version of R that you are using:

```
sessionInfo()
```

15. Clear your workspace

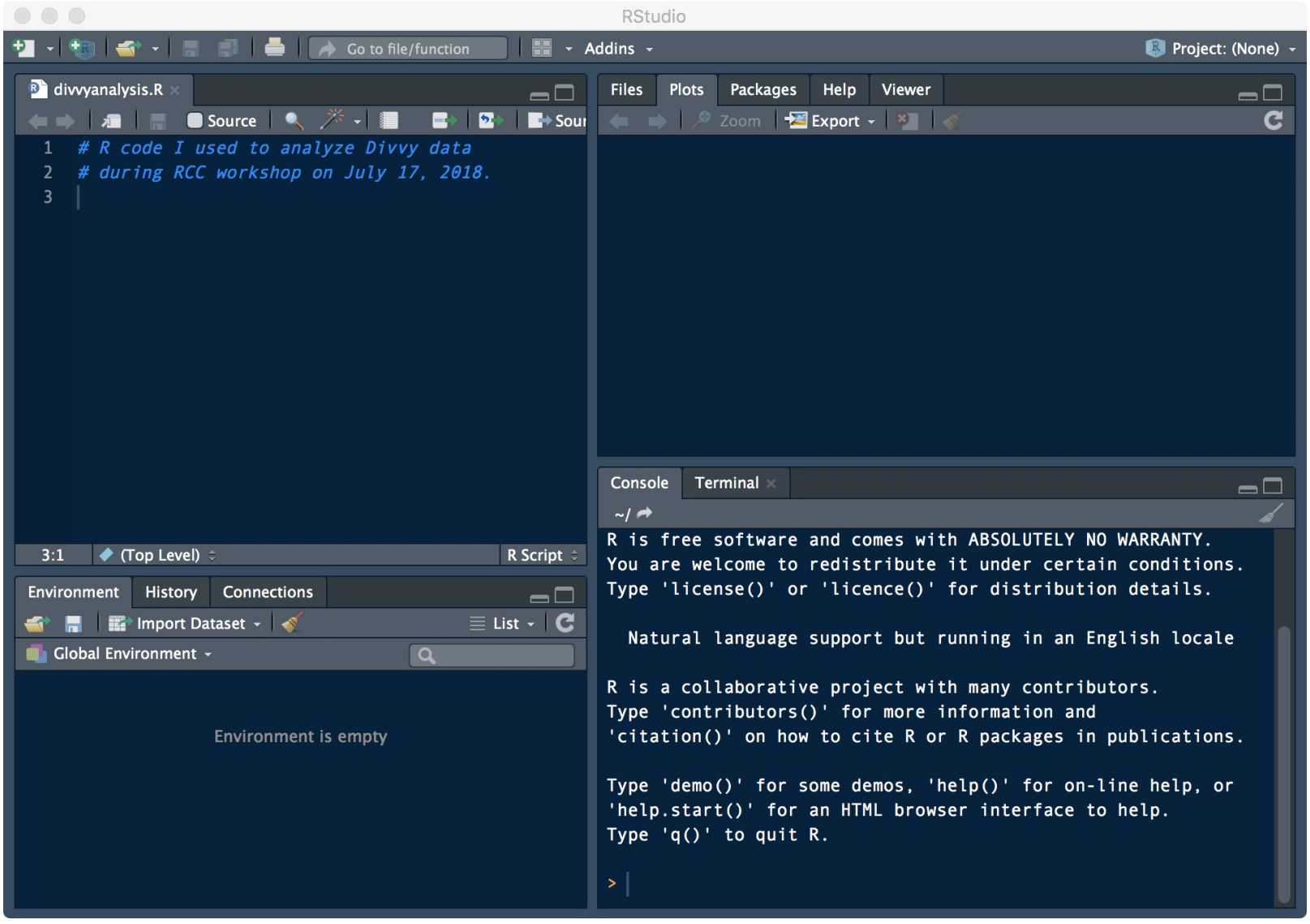
The R environment is where all variables (and functions) are stored and accessed. It is Best Practice to start with an empty environment, or "workspace". Run this:

```
ls()
```

If this outputs names of objects, it means your environment is not empty and you should restart R with a clean environment. Do either:

- `rm(list = ls())`.
- Or, in RStudio, **Session > Clear Workspace**.

16. The Console is where the action is



17. Set your working directory to “R-intro-divvy”

Check that you have the right working directory:

```
list.files()
```

You should see the tutorial files. If you don’t, change your working directory:

- In R, use the `setwd()` function.
- In RStudio, select **Session > Set Working Directory > Choose Directory...‘**

If you have changed your working directory, double-check that you have the right working directory before continuing.

18. Outline of workshop

1. Initial setup.
2. **Analysis of Divvy station data.**
3. Analysis of Divvy trip data.
4. Combining the data.

19. View CSV file

Open the CSV file **Divvy_Stations_2017_Q3Q4.csv** in RStudio or your favourite text editor (e.g., Notepad in Windows, TextEdit on Mac).

20. Import station data into R

Read the station data into a “data frame”:

```
stations<-read.csv("Divvy_Stations_2017_Q3Q4.csv",
                   stringsAsFactors = FALSE)
```

This will create a new object, “stations”, in your environment:

```
ls()
```

It is a “data frame” object:

```
class(stations)
```

21. Inspect the station data

Check that the data were read correctly, and inspect the table:

```
nrow(stations)
ncol(stations)
head(stations)
tail(stations)
summary(stations)
str(stations)
```

Inspect the data further:

```
sapply(stations,class)
object.size(stations)
```

22. Take a closer look at the “dpcapacity” column

Make a copy of the “dpcapacity” column:

```
x <- stations$dpcapacity
```

Run a few commands to take a closer look at the “dpcapacity” column:

```
class(x)
length(x)
summary(x)
min(x)
max(x)
mean(x)
median(x)
quantile(x,0.5)
table(x)
```

23. Data subviews

When you are working with large data sets, you need a strategy to inspect manageable subsets of the data. Here are some examples of printing subsets of the data:

```
head(stations,n = 4)
tail(stations,n = 4)
```

More examples:

```
stations[1:4,]
stations$name[1:4]
stations[1:4,2]
stations[1:4,"name"]
```

24. Data subviews

Yet more examples:

```
stations[1:4,c(2,3,6)]
stations[1:4,c("name","city","dpcapacity")]
```

25. Conditional subviews

One powerful way to inspect subsets is by condition. For example, to view all the stations with more than 40 docks, do

```
subset(stations,dpcapacity > 40)
```

or, equivalently,

```
stations[stations$dpcapacity > 40,]
```

It is interesting that a couple of the Divvy bike stations have no docks. What are these stations?

```
# Add code here.
```

26. Conditional subviews

Once you have generated a subview that you want to explore further, you can create a new data set from a subview, e.g.,

```
largest_stations<-subset(stations,dpcapacity > 40)
```

This object is also a data frame:

```
class(largest_stations)
```

27. Ordering the stations by number of docks

Here's way to access data on the smallest and largest stations:

```
rows <- order(stations$dpcapacity)
stations2 <- stations[rows,]
head(stations2)
tail(stations2)
```

28. Take a closer look at the “city” column

Above, we examined numeric data. Now let’s take a close look at another type of data.

```
x <- stations$city
class(x)
summary(x)
```

The summary is not very useful here! The key is to convert to a “factor”:

```
x <- factor(stations$city)
class(x)
summary(x)
```

29. Fixing the “city” column

Let’s fix the problem of two “Chicago” categories. First, select the offending rows in the table:

```
rows <- which(stations$city == "Chicago ")
```

Fix the “city” column by *overwriting* the selected rows:

```
stations[rows,"city"] <- "Chicago"
x <- factor(stations$city)
summary(x)
```

The “city” column is more useful if it is a factor, so let’s modify this column *inside* the data frame:

```
stations$city <- factor(stations$city)
summary(stations$city)
```

30. Save your code & session state

It is important to periodically save your code and results.
To save your workspace, go to **Session > Save Workspace As...** in RStudio, or run this code:

```
save.image("divvy_analysis.RData")
```

Later, to restore your environment in a new session, select **Session > Load Workspace...** in RStudio, or run this code:

```
load("divvy_analysis.RData")
```

31. Main concepts covered so far

- The R workspace & working directory.
- Read a data frame from a text (CSV) file.
- Tools to inspect a data frame.
- Tools to manipulate a data frame.
- Subviews and conditional subviews.
- Ordering rows of a data frame.
- Factors.
- Saving your results.

32. Outline of workshop

1. Initial setup.
2. Analysis of Divvy station data.
3. **Analysis of Divvy trip data.**
4. Combining the data.

33. Import the Divvy trip data into R

Previously, we used `read.csv` to import station data into R. Let's now use `read.csv` to load the trip data from the 4th quarter of 2017:

```
trips <-
  read.csv("Divvy_Trips_2017_Q4.csv",
           stringsAsFactors = FALSE)
```

You may find that this command look longer to run than before. The trips data frame is much larger:

```
nrow(trips)
ncol(trips)
object.size(trips)
```

34. Import Divvy trip data using `readr` (optional)

Install the **readr** package from CRAN:

```
install.packages("readr")
```

Load the package functions into your R environment:

```
library(readr)
```

Let's use the `read_csv` function from this package:

```
trips <- read_csv("Divvy_Trips_2017_Q4.csv")
```

35. Import Divvy trip data using `readr` (optional)

The `read_csv` output is *not* a data frame—it is a "tibble".

```
class(trips)
```

Usually, I convert it to a data frame:

```
class(trips) <- "data.frame"
```

36. A first glance at the trips data

Let's use some of the same commands we used earlier to quickly get an overview of the trip data:

```
nrow(trips)
ncol(trips)
head(trips)
summary(trips)
```

37. Convert “gender” to a factor

Let’s begin by converting the “gender” column to a factor:

```
trips$gender <- factor(trips$gender)
summary(trips$gender)
levels(trips$gender)
```

38. Missing data

- In R, “missing data” should be assigned the special value `NA` (“not available”).
- Many functions in R will correctly handle missing data as long as they are encoded as `NA`.
- The `read_csv` function from the `readr` package was “smart” enough to figure out that blank entries in the CSV file should be converted to `NA`.

39. Convert “station” columns to factors

Likewise, the “from station” column is also more useful as a factor:

```
summary(trips$from_station_name)
trips$from_station_name <-
  factor(trips$from_station_name)
summary(trips$from_station_name)
```

40. A note about dates & times

- `summary(trips$start_time)` and `summary(trips$end_time)` are also not informative.
- Processing dates & times is more complicated.
- See `help(strptime)` and the **lubridate** package.

41. Combining trip data from two quarters

Up to this point, we have only examined at the trip data from one quarter of one year. Suppose we wanted to analyze the trip data from both the 3rd and 4th quarters of 2017. How would we do that? First, we have to import both tables.

```
tripsQ4 <- trips
tripsQ3 <- read_csv("Divvy_Trips_2017_Q3.csv")
class(tripsQ3) <- "data.frame"
```

We can *merge* the two tables *by row*—that is, we put one table on top of the other.

```
trips <- rbind(tripsQ4,tripsQ3)
```

42. Combining trip data from two quarters

Let's double-check the result:

```
dim(tripsQ3)
dim(tripsQ4)
dim(trips)
head(trips)
```

43. Preparing data is tedious

Data preparation is sometimes >90% of the effort!

- *Many analysis mistakes are due to poor data preparation.*

Common issues include:

- Formatting mistakes in CSV file.
- Converting table columns to the appropriate data type.
- Inconsistent data entry (e.g., additional spaces).
- Missing data.

44. Moving beyond data preparation

- So far, we have illustrated a few of the challenges of working with large tabular data sets (“data frames”).
- In order to proceed to fun stuff, I’ve automated the data preparation steps by writing an R *function* to do this.

45. Outline of workshop

1. Initial setup.
2. Load & prepare the Divvy station data.
3. Load & prepare the Divvy trip data.
4. **Create a map of the Divvy stations.**
5. Create plots comparing bike sharing in 2016 & 2017.

46. Refresh your environment

We will begin a new analysis, so let's refresh our environment:

```
rm(list = ls())
```

Or, in RStudio, go to **Session > Restart R**.

47. Import the 2016 & 2017 Divvy data

Load function `read.divvy.data` that automates the reading and processing of all the downloaded Divvy data:

```
source("readdivvydata.R")
```

48. Import the 2016 & 2017 Divvy data

The `read.divvy.data` takes the name of the station file to import, and the names of trip data files to import & merge.

```
stationfile <- "Divvy_Stations_2017_Q3Q4.csv"
tripfiles <- list.files(pattern="Divvy_Trips_*")
```

49. Import the 2016 & 2017 Divvy data

Load and process the station data:

```
stations <- read.station.data(stationfile)
```

Load and process the trip data:

```
trips <- read.trip.data(tripfiles,stations)
```

Loading the trip data may take a minute to run (or longer if you have not installed the readr package).

50. Inspect the 2016 & 2017 Divvy data

```
head(stations)
summary(stations)
nrow(trips)
head(trips)
summary(trips)
```

51. Our first plot: a map of the Divvy stations

We will use the **ggplot2** package. It is a powerful (though not immediately intuitive) set of plotting functions that extend the base plotting functions in R.

```
install.packages("ggplot2")
```

I also recommend the **cowplot** package, an extension to ggplot2 developed by Claus Wilke at UT Austin.

```
install.packages("cowplot")
```

Load the ggplot2 and cowplot functions:

```
library(ggplot2)
library(cowplot)
```

52. Plot station longitude vs. latitude

The “stations” data frame gives the geographic co-ordinates (latitude & longitude) for each station. With ggplot, we can create a station map from the “stations” data frame in only a few lines of code:

```
a <- aes(x = longitude,y = latitude)
p <- ggplot(stations,a)
print(p)
g  <- geom_point()
p <- ggplot_add(g,p)
print(p)
```

53. Adjusting the plot

ggplot has many options for changing the look of the plot elements:

```
g <- geom_point(shape = 21,fill = "limegreen",
                color = "white",size = 3)
p <- ggplot_add(g,p)
print(p)
```

54. Use color to highlight the largest stations

To do this, map the “dpcapacity” column to colour in the plot:

```
a <- aes(x    = longitude,
         y    = latitude,
         fill = dpcapacity)
p <- ggplot(stations,a)
g <- geom_point(shape = 21,color = "white",
                size = 3)
p <- ggplot_add(g,p)
print(p)
```

55. Use color to highlight the largest stations

The colour scale is not great, so let's improve it:

```
g <- scale_fill_gradient2(low = "dodgerblue",
  mid = "darkblue",high = "red",midpoint = 25)
p <- ggplot_add(g,p)
print(p)
```

56. Scale points by number of departures

We need to add a new column to the “stations” data frame containing the total number departures. It can be calculated from the “trips” data frame:

```
counts <- table(trips$from_station_name)
```

Because we used `read.divvy.data`, station counts should be the same order as the stations. Let’s verify this:

```
all(names(counts) == stations$name)
```

57. Scale stations by the number of departures

Add the trip counts to the “stations” data frame:

```
stations$departures <- as.vector(counts)
head(stations)
```

Now we can include the “departures” column in a plot:

```
a <- aes(x    = longitude,
         y    = latitude,
         size = sqrt(departures))
p <- ggplot(stations,a)
```

Add points to your plot:

```
# Add code here
```

58. Save & share your plot

For exploratory analyses, GIF and PNG are great formats because the files are easy to attach to emails or webpages:

```
ggsave("station_map.png",p,dpi = 100)
```

For print or publication, save in a vector graphics format:

```
ggsave("station_map.pdf",p)
```

59. Save your code & session state

This is a good time to save your session.

```
save.image("divvyanalysis.RData")
```

60. Compare 2016 & 2017 biking activity

Earlier, we saw that that were more trips in 2017 than 2016. Which stations experienced the largest increase in trips? To explore this question, we take the following steps:

1. Count trips separately for 2016 and 2017.
2. Add these counts to the “stations” data frame, similar to before.

61. Count trips separately for 2016 & 2017

We will use `subset` followed by `table` to do this:

```
d1 <- subset(trips,start.year == 2016)
d2 <- subset(trips,start.year == 2017)
x1 <- table(d1$from_station_name)
x2 <- table(d2$from_station_name)
```

62. Add counts to the stations data frame

Remember we need to convert the “table” output to a vector:

```
stations$dep.2016 <- as.vector(x1)
stations$dep.2017 <- as.vector(x2)
head(stations)
```

63. Scatterplot of trips by station (2016 vs. 2017)

As before, now that we have prepared a data frame, creating the plot should be straightforward.

```
a <- aes(x = dep.2016,y = dep.2017)
p <- ggplot(stations,a)
g <- geom_point()
p <- ggplot_add(g,p)
print(p)
```

64. Scatterplot of trips by station

It is difficult to tell which stations had more trips in 2017—it would be helpful to compare against the diagonal (x = y) line.

```
g2 <- geom_abline(slope = 1,color = "orange",
                  linetype = "dashed")
p  <- ggplot_add(g2,p)
print(p)
```

65. Save your code & session state

Save your final results for safekeeping.

```
save.image("divvyanalysis.RData")
```

66. ggplot: Take home points

- Creating sophisticated plots requires relatively little effort *provided the data are in the right form.*
- All plots in ggplot2 require these three elements:
  1. A data frame.
  2. An "aesthetic mapping" that declares how columns are mapped to plot features (axes, shapes, colors, *etc.*).
  3. A "geom", short for "geometric object," that specifies the type of plot.
- All plots are created by *adding layers.*

67. Why data analysis in R?

- In R, a spreadsheet (“data frame”) is an *object* that can be inspected, manipulated and summarized with code.
- Therefore, we can write an R script to *automate* the data preparation and analysis steps.

68. Parting thoughts

1. Always record your analysis steps in a file so you can reproduce them later.
2. Keep track of which packages (and the versions) you used with `sessionInfo()`.
3. Use "R Markdown" to document your analyses.
4. Use packages—don't reinvent the wheel.
5. Email `help@rcc.uchicago.edu` for advice on using R on the RCC cluster.
6. See the **workflowr** package for simplifying organizing & sharing of data analyses; e.g., **stephenslab.github.io/wflow-divvy**.
7. Thank you!