# 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”—this includes learning from our mistakes!

# 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
- Much of the effort will be spent importing the data, inspecting the data, and preparing the data for analysis.
- Once we have carefully prepared the data, creating visualizations is (relatively) easy.

# 4. The programmatic approach

- Data analyses are usually *iterative* and *repetitive*. Therefore, you need to develop skills to *efficiently* create & refine your analyses.
- The *programmatic approach* to data analysis will allow us to...
  - ▷ Reuse our code to quickly create sophisticated analyses—*automation is your friend*!
  - ▷ Expand capabilities by installing new R packages.

# 5. It's your choice

Your may choose to...

- Use R on your laptop.
- Use RStudio on your laptop.
- Use R or RStudio on the RCC cluster.
- Pair up with your neighbour.
- 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. 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.

# 8. Initial setup

- WiFi.
- Power outlets.
- Computer clutter.
- Workshop packet.
- Reading what I type.
- Pace, questions (e.g., keyboard shortcuts).
- Help.

# 9. Download or “clone” git repository

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**.

# 10. What's included in the workshop packet

- **slides.pdf**: These slides.
- **slides.Rmd**: R Markdown source used to generate these slides. *Copy & paste code from this file.*
- **functions.R**: Some R code used in the hands-on examples.

# 11. Set up your R environment

- Launch R or RStudio.

# 12. Run `sessionInfo()`

Check the version of R that you are using:

```
sessionInfo()
```

# 13. Clear your R environment

The R environment is where all variables (and functions) are stored and accessed. You should start with an empty environment. 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:

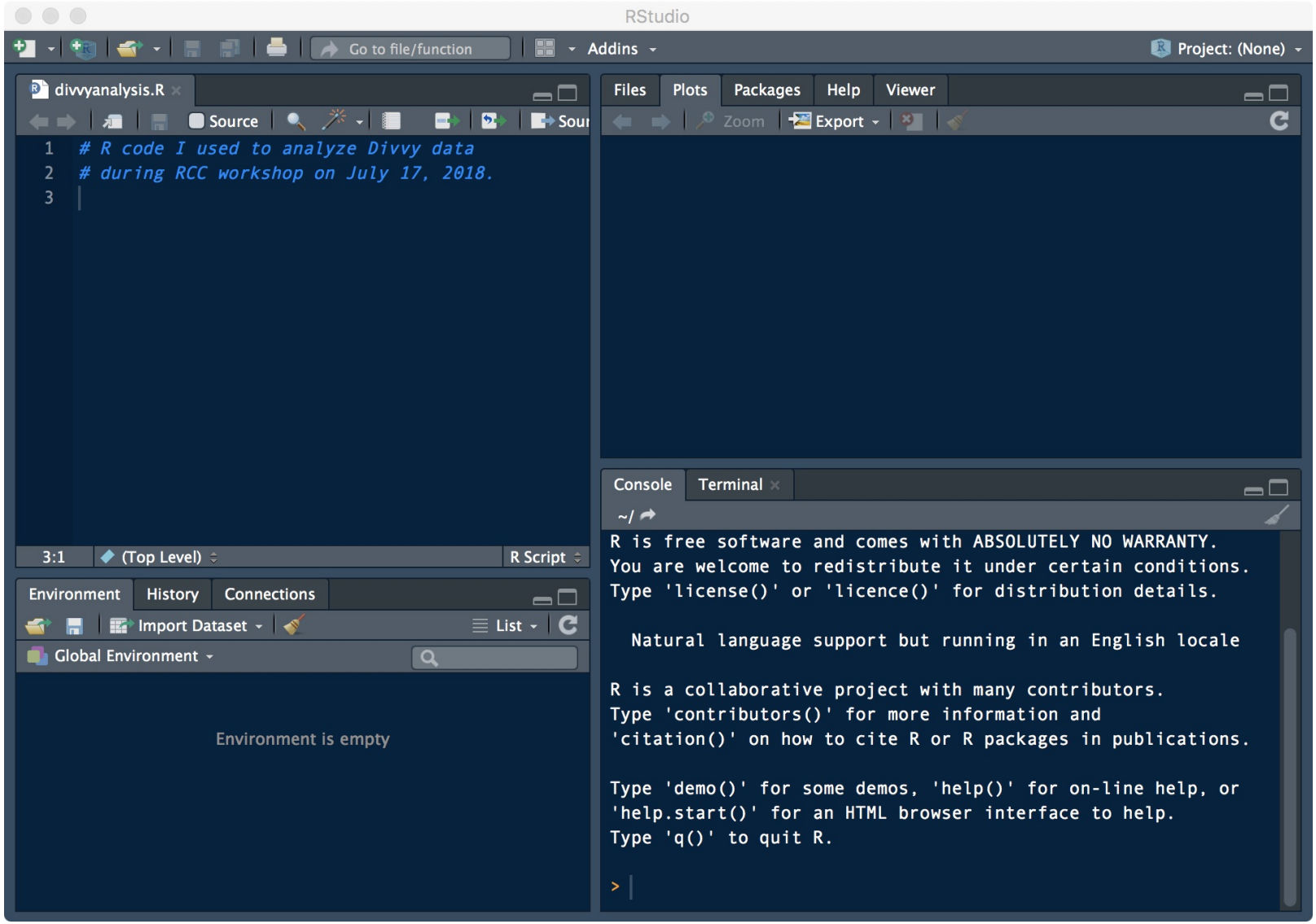
- `rm(list = ls())`.
- Or, in RStudio, **Session > Clear Workspace**.

# 14. 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.

# 15. The Console is the “brains” of RStudio



# 16. Download the Divvy data

- Disk space required: at least **2 GB**.
- Download the 2016 & 2017 data files from here:
  - ▷ **www.divvybikes.com/system-data**
- Download them to the **R-intro-divvy** folder.
- You should have 4 ZIP files:

```
Divvy_Trips_2016_Q1Q2.zip
Divvy_Trips_2016_Q3Q4.zip
Divvy_Trips_2017_Q1Q2.zip
Divvy_Trips_2017_Q3Q4.zip
```

- Decompress (“unzip”) all of these files.

# 17. Check that you have all the files

After unzipping, you should have **15** CSV files. Check this:

```
list.files(pattern = "*.csv")
```

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

Check that you have the right working directory:

```
getwd() # Should end with "R-intro-divvy".
```

If you don’t, change your working directory:

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

Before continuing, double-check that you have the right working directory.

# 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.

# 20. Take a quick look at CSV file

Open the CSV file **Divvy_Stations_2017_Q1Q2.csv** in RStudio, or in your favourite text editor (e.g., Notepad in Windows, TextEdit on Mac).

# 21. Import the station data into R

Load 2017 third & fourth quarter station data into an R “data frame”:

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

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

```
ls()
```

It is a “data frame” object:

```
class(stations)
```

# 22. Inspect the station data

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

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

Inspect the data in more detail:

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

# 23. Take a closer look at the “dpcapacity” column

Create a new object containing only the “dpcapacity” column:

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

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

```
class(x)
length(x)
summary(x)
table(x)
```

# 24. More on selecting rows & columns

Select first 4 rows of “name” column:

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

Select first 4 rows and multiple columns:

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

Getting the row and column names:

```
colnames(stations)
rownames(stations)
```

# 25. Take an even closer look at “dpcapacity”

It is interesting that a few of the Divvy bike stations are much larger than the others, whereas others have no docks. Where are these stations?

```
subset(stations,dpcapacity == 0)
subset(stations,dpcapacity >= 40)
```

Alternatively, we can sort the table rows, then inspect the top and bottom rows:

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

# 26. Take a closer look at the “city” column

Previously, we inspected *numeric* data. Next’s, let’s look at an example of non-numeric data.

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

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

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

# 27. Fixing the “city” column

Let’s fix the problem we found earlier. First, select the offending rows of the table:

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

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

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

The “city” column is more useful if it is a factor. Let’s modify this column inside the data frame:

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

# 28. What is a “factor”? (optional)

Factors are often very useful in data analyses. Let’s take a deeper look at what a factor *is*.

```
x <- stations$city
attributes(x)
unclass(x)
```

From the `unclass(x)` call, we see that a factor is really just an integer with values 1, 2, 3, *etc*., with which each integer value is associated with a *label* (e.g., “Chicago”, “Evanston”).

# 29. Save your code & session state

It is important to periodically save:

1. your code,
2. the state of your R environment.

To save your environment, go to **Session > Save Workspace As...** in RStudio, or run this code:

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

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

```
load("divvyanalysis.RData")
```

# 30. Main concepts covered so far

- The R environment & working directory.
- Read a data frame from a text (CSV) file.
- Tools to inspect a data frame.
- Tools to manipulate a data frame.
- Selecting rows & columns.
- Ordering rows of a data frame.
- Factors = categorical variables.
- Save state of R environment.

# Outline of workshop

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

# 32. 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)
```

# 33. Import Divvy trip data using `readr`

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")
```

# 34. Import Divvy trip data using `readr`

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

```
class(trips)
```

Typically, I convert it to a data frame:

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

# 35. 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)
```

# 36. Convert “gender” to a factor

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

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

# 37. “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`.

# 38. Convert “station” columns to factors

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)
```

# 39. 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.

# 40. 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.
- Entry inconsistencies (e.g., additional spaces).
- Missing data.

multiple files—this usually creates more headaches!

# 41. 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.

# 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.

# 43. 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**.

# 44. Import the 2016 & 2017 Divvy data

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

```
source("functions.R")
```

# 45. 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_*")
```

# 46. Import the 2016 & 2017 Divvy data

Load and process the trip and station data:

```
divvy <- read.divvy.data(stationfile,tripfiles)
```

This may take a minute to run, or longer if you have not installed the readr package.

# 27. Inspect the 2016 & 2017 Divvy data

The output is a “list” object, in which each list element is a data frame. Let’s extract the data frames from the list:

```
names(divvy)
stations <- divvy$stations
trips    <- divvy$trips
rm(divvy)
head(stations)
head(trips)
nrow(trips)
```

# Out first ggplot: a map of the Divvy stations

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

# 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:

```
aes1 <- aes(x = longitude,y = latitude)
p    <- ggplot(stations,aes1)
print(p)
out <- geom_point()
p2  <- ggplot_add(out,p)
print(p2)
```

*What geographic features of Chicago are recognizable from this plot?*

# Adjusting the plot

Let's make a few adjustments to the plot:

```
out <- geom_point(shape = 21,fill = "darkblue",
                  color = "white",size = 3)
p3  <- ggplot_add(out,p)
print(p3)
```

# Plotting contours instead of points

We can reuse our existing code, replacing the `geom_point` with a `geom_density_2d`, to create a very different plot:

```
out  <- geom_density_2d()
p4   <- ggplot_add(out,p)
print(p4)
```

# Use colors to highlight the largest stations

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

```
aes2 <- aes(x     = longitude,
            y     = latitude,
            color = dpcapacity)
p   <- ggplot(stations,aes2)
out <- geom_point()
p <- ggplot_add(out,p)
print(p)
```

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

```
out <- scale_color_gradient2(low = "white",
  mid = "skyblue",high = "red",midpoint = 25)
p <- ggplot_add(out,p)
print(p)
```

*Where are the largest Divvy stations?*

# Scale stations by the number of departures

Next, let's add an additional piece of information to this visualization:

- Number of departures at each station, should (?) roughly correspond to population density.

To do this, we need to add a new column to the "stations" data frame containing the total number departures, which is calculated from the "trips" data frame:

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

Because we carefully prepared the data frame in `read.divvy.data`, station counts should be the same order as the stations. We can check this:

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

# Scale stations by the number of departures

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

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

Let’s use this column in our new plot:

```
aes3 <- aes(x    = longitude,
            y    = latitude,
            size = sqrt(departures))
p    <- ggplot(stations,aes3)
out  <- geom_point(shape = 21,fill = "blue",
                   color = "white")
p    <- ggplot_add(out,p)
print(p)
```

# How to save and 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)
```

# Save your code & session state

This is a good time to save your session.

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

# Compare 2017 biking activity against 2016

Earlier, we observed an increase in trips from 2016 to 2017. Which stations experienced the largest increase?

- To examine this, we need to count trips separately for 2016 and 2017.
- Then we add these counts to the "stations" data frame.

We will use the `subset` and `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)
stations$dep.2016 <- as.vector(x1)
stations$dep.2017 <- as.vector(x2)
head(stations)
```

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

As before, now that we have prepared a data frame, plotting with ggplot is relatively straightforward:

```
aes3 <- aes(x = dep.2016,y = dep.2017)
p    <- ggplot(stations,aes3)
out  <- geom_point(shape = 20,size = 2)
p    <- ggplot_add(out,p)
print(p)
```

It is difficult to tell which stations had more trips in 2017—we need to compare against the x = y line.

```
out2 <- geom_abline(slope = 1,color = "skyblue",
                    linetype = "dashed")
p    <- ggplot_add(out2,p)
print(p)
```

*One station stands out because it has had a much larger increase in trips than other stations. What is this station?*

# Save your code & session state

Save your final results for safekeeping.

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

# 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.*

# 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 scripts to *automate* our data analyses.

# 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 packages—don't reinvent the wheel.
4. Email `help@rcc.uchicago.edu` for advice on using R on the RCC cluster.
5. Use "R Markdown" to document your analyses.
6. See the **workflowr** package for simplifying organizing & sharing of data analyses; e.g., **stephenslab.github.io/wflow-divvy**.
7. Thank you!