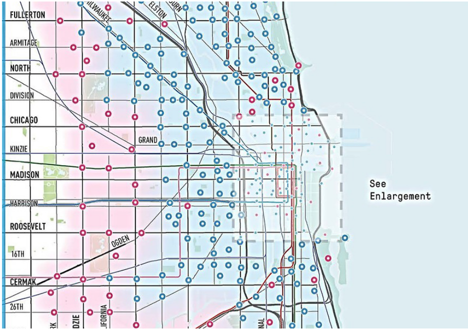
Introduction to R for data analysis

Peter Carbonetto

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



Aims of workshop

1. Work through a basic 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 create plots from data frames.
6. Learn through “live coding”—this includes learning from our mistakes!

Our goal: Analyze Divvy data from 2016 & 2017

- Investigate bike sharing trends in Chicago.
- We will use data made available from 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.

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.

Note: If you use the RCC cluster I'm assuming you know how to set up an interactive computing session with appropriate amount of compute time and memory, load R or RStudio, and display graphics (e.g., using ThinLinc).

Software we will use today

1. **R**
2. R packages **readr**, **ggplot2** & **cowplot**.
3. **RStudio** (optional).

Note: I'm assuming you have already installed R and/or RStudio on your laptop, or you are using the RCC cluster.

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.

Initial setup

- WiFi
- Power outlets
- Pace, questions (e.g., keyboard shortcuts).
- Help.

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.)

- **Note:** If you are using the RCC cluster, also download workshop packet on the cluster.

What's included in the workshop packet

1. **analysis**: directory where the code and data for our analyses will be stored.
2. **slides.pdf**: the workshop slides.
   - ▷ This PDF is useful for copying & pasting code from the slides. (Although I have found that you should only copy one line at a time.)
   - ▷ The PDF is included in the workshop packet.
   - ▷ You can also view the PDF by clicking the "slides.pdf" item on the GitHub webpage.

Set up your R environment

- Launch R or RStudio.
- *We will run all the code from the* ***"analysis"*** *folder*.
- To change your working directory:
  - ▷ In R, use `setwd()` function.
  - ▷ In RStudio, select **Session > Set Working Directory > Choose Directory. . .** '

Before continuing, check that you have the right working directory:

```
getwd()  # Should end with "analysis".
```

Run `sessionInfo()`

Check the version of R that you are using:

```
sessionInfo()
```

If you are using an older version of R (version 3.3 or less), I strongly recommend upgrading to the latest version. *Some of the examples may not work in older versions of R.*

Check your R environment

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

```r
ls()
```

If you see names of objects listed, it means your environment is not empty, and you should restart R with a clean environment.

- Do `rm(list = ls())`.
- Or, in RStudio, go to **Session > Restart R**.

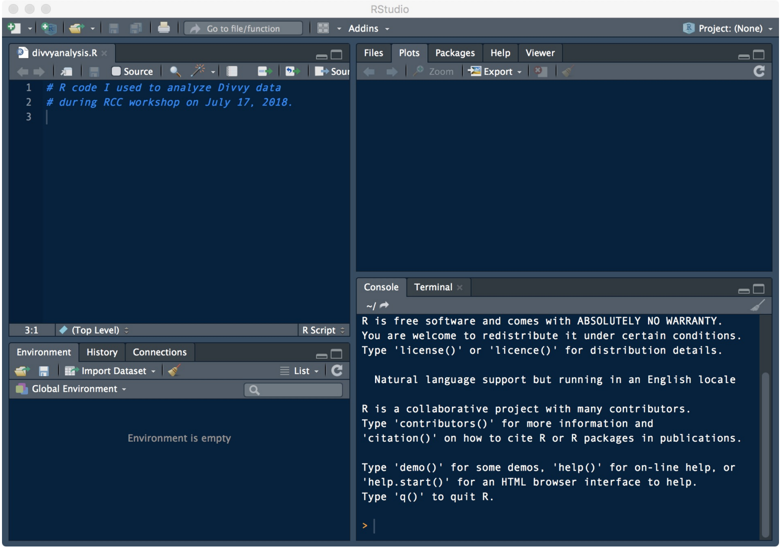
Creating a file to keep track of your analysis code

- In RStudio, select **File > R Script**.
- Alternatively, use your favourite editor.
- Add some comments to the title to remind yourself what this file is for, e.g.,

```
# Some of the R code I wrote during the RCC
# workshop on August 16, 2018.
```

- Save the file in the **"analysis"** folder. Name the file whatever you'd like (e.g., `divvyanalysis.R`).

The Console is the "brains" of RStudio



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 **"analysis"** 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.

Check that you have all the files

After unzipping, you should have **15** CSV files.

```
Sys.glob("*.csv")
```

If you don't see all 15 CSV files, you have not successfully downloaded and/or unzipped all the files in the "data" 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.

Eyeball the station data in the 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).
- CSV is a simple and commonly used data format.
- It is easily read into R, and read by humans.
- Each line stores an item (station).
- The first line is a special line called the “header”.
- Entries in each line (table row) are separated by commas.

Import the station data into R

Load the most up-to-date 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)
```

What does “read.csv” do, and what is a “data frame”? R has detailed documentation:

```
help(read.csv)
help(data.frame)
```

Inspect the station data

Run these commands to start inspecting the station data:

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

Inspect the data in more detail:

```
sapply(stations,class)
object.size(stations)
print(object.size(stations),units = "Kb")
```

What do we learn about the station data from running these commands? Does this reveal any issues with the data?

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)
```

Did we gain any additional insight from running these commands?

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)
```

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)
```

How were the rows originally ordered in `stations`*?*

Take a closer look at the “city” column

Above 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)
```

Did you discover an issue with the data from running these commands?

Improving 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, so let’s convert the column directly inside the data frame:

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

What is a "factor"?

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").

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")
```

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.
- Manipulate a data frame.
- Factors = categorical variables.
- Selecting rows & columns.
- Order rows of a data frame.
- 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.

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. Consider that the trips data is much larger:

```
nrow(trips)
ncol(trips)
print(object.size(trips),units = "Mb")
```

This gives an opportunity to demonstrate a faster method implemented in a *package*.

Import Divvy trip data using `readr` (optional)

Install the **readr** package from CRAN:

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

Load the functions from the package into your R environment:

```
library(readr)
```

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

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

Note: `read_csv` is similar to `read.csv`, but not the same.

- *How much faster is `read_csv`?*

Import Divvy trip data using `readr` (optional)

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"
```

- For more on tibbles, see:
  - ▷ `http://r4ds.had.co.nz`
- The **readr** package has many other features not covered here.
- Another fast method is `fread` from the **data.table** package.

More on packages in R

“Vignettes” are a great way to learn about a package:

```
vignette(package = "readr")
vignette("readr")
```

- CRAN is the official package source:
  - ▷ `https://cran.r-project.org`.
- Other good places to find packages:
  - ▷ Bioconductor
  - ▷ GitHub.
- *What packages are already installed?* `rownames(installed.packages())`
- *Where do the packages live?* `.libPaths()`
- *How to learn more about a package?* `help(package=readr)`

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)
```

Unfortunately, the summary command isn't particularly informative for many of the columns.

- *What columns should we convert to factors?*

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)
```

We observe that many gender entries are missing.

“Missing” data

- In R, “missing data” should always be assigned the special value `NA` (“not available” or “not assigned”).
- 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 is “smart” enough to figure out that blank entries in the CSV file should be converted to `NA`.

Convert “station” columns to factors

It is also useful to convert the “from station” column to a factor:

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

The summary is now more informative.

A note about dates & times

- `summary(trips)` is also not useful for the dates & times.
- Processing dates & times is more complicated.
- See `help(strptime)` and the **lubridate** package.

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.
- Many other examples of Poor Practices in recording data.

(And we haven't yet dealt with merging data from multiple files—this usually creates more headaches!)

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 a *function* in R to import and merge all the Divvy data into a single data frame.

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.

Clean up your environment

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

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

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

Import the 2016 & 2017 Divvy data

So far, we have only analyzed the trip data from the 4th quarter of 2017.

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

I wrote a function `read.divvy.data` to automate the reading and processing of all the downloaded Divvy data. It reads all the CSV files, then merges them into two data frames: one for the stations, and one for the trips.

Import the 2016 & 2017 Divvy data

Choose which station and trip files to import:

```
stnfile   <- "Divvy_Stations_2017_Q3Q4.csv"
tripfiles <- Sys.glob("Divvy_Trips*.csv")
```

Variables `stnfile` and `tripfiles` contains the names of the files to be importedl they do not actually contain any data.

Import the 2016 & 2017 Divvy data

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

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

Note: If your computer does not have enough memory to load all the trip data, use only the Q1 trip files instead:

```
tripfiles <- Sys.glob("Divvy_Trips*Q1.csv")
```

What `read.divvy.data` does:

- Reads the Divvy station data from the CSV file.
- Reads the Divvy trip data from the CSV files.
- Combines the Divvy trip data into a single data frame.
- Takes additional steps to prepare the data.

Import the 2016 & 2017 Divvy data

The output is a “list” containing two data frames. 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)
```

- *Were more trips taken in 2016 or 2017?*
- *Which columns were converted to factors?*
- *What oddities do you notice from the summary?*

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!