

# 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](http://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.
- Arranging Zoom and RStudio on the screen.
- 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

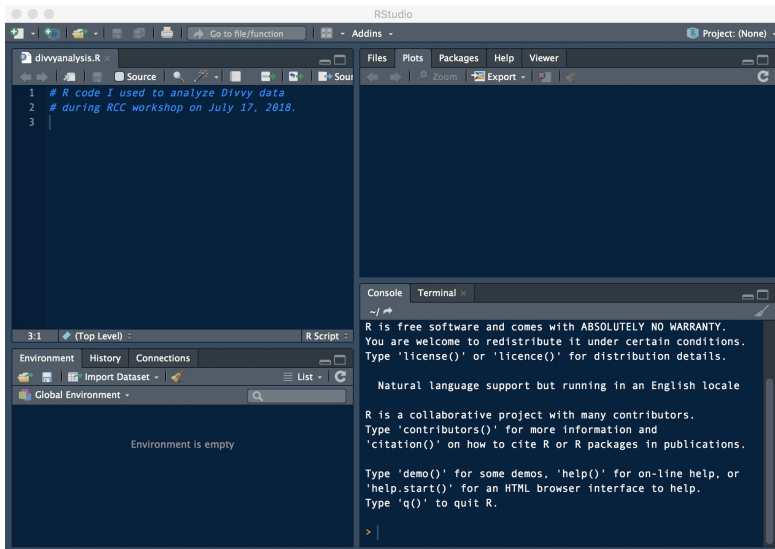
The R environment is where all variables (and functions) are stored and accessed. It is Best Practice to start with an empty environment, or “workspace”. Run this:

```
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. Create a map of the Divvy stations

A scatterplot can be created from two numeric vectors very simply using the “plot” function. Let’s see what happens if we plot the geographic co-ordinates (latitude & longitude) of the stations.

```
plot(stations$longitude, stations$latitude,  
      pch = 20)
```

The plot function has many, many options. We will not explore these options here. Let’s add color to the plot according to the “city” column:

```
plot(stations$longitude, stations$latitude,  
      col = stations$city, pch = 20)
```

## 31. Create stations map using ggplot2

Now let's recreate the stations map using **ggplot2**. It is a powerful (though not immediately intuitive) plotting interface. First, install ggplot2 if you have not already done so. (We will also use cowplot, an extension to ggplot2.)

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

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

As you can see, the ggplot2 code is a bit more complicated:

```
library(ggplot2)
```

```
p1 <- ggplot(stations,  
             aes(x = longitude,  
                 y = latitude,  
                 color = city)) +
```

```
  geom_point()
```

```
print(p1)
```

What is better about the new plot?

## 32. More on ggplot2

- All plots in ggplot2 require these three elements:
  1. A data frame.
  2. An “aesthetic mapping” that maps columns 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*.
- ggplot2 has an excellent website where you can learn more:  
`ggplot2.tidyverse.org`



## 33. A better stations map

Not satisfied with this plot, I experimented with the `geom_point` settings to improve the plot a bit:

```
p2 <- ggplot(stations,
             aes(x = longitude,
                 y = latitude,
                 fill = city)) +
  geom_point(shape = 21, size = 2, color = "white")
print(p2)
```

## 34. A better stations map

The default colours in ggplot2 are not great; they can be overridden here with the “scale” function `scale_fill_manual`. Also, I’m a big fan of the cowplot theme:

```
library(cowplot)
p2 <- p2 +
  scale_fill_manual(values = c("dodgerblue",
                                "darkorange",
                                "darkblue")) +
  theme_cowplot(font_size = 10)
print(p2)
```

We have only touched the surface of what ggplot2 can do. Observe that adjustments to the plot are made by adding “layers”. This is one of the distinctive features of ggplot2.

## 35. Save & share your plot

Let's save this last plot as a file that can be shared with others.

```
ggsave("stations.png", p2, dpi = 200)  
ggsave("stations.pdf", p2)
```

## 36. Save your results

It is important to periodically save your code and results. (Remember there is no “undo” command in R!) 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")
```

## 37. 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.
- Creating a plot using “plot”.
- Creating a plot using ggplot2.
- Saving your results.

## 38. Outline of workshop

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

## 39. Automating the data preparation

Now we will analyze the trip data. This is a much larger set of data. Since data preparation can be tedious, I've simplified the preparation of the trip data for you by writing a *script* to do this. Notice that the script loads the **readr** package. So you will need to install this package if you haven't already done so.

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

To run the script, simply type:

```
source("read_trip_data.R")
```

You will find that `read.csv` is horribly slow for large CSV files; for this reason, I used `read_csv` from the **readr** package. I recommend both the **readr** and **data.table** packages for importing large data sets.

## 40. 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)
```

What types of data are stored in this table?



## 41. Missing data

- In R, “missing data” has a special value, `NA` (short for “not available” or “not assigned”). Most data types allow NAs.
- 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`.

## 42. Plot number of trips by day

The days of the year are numbered:

```
summary(trips$dayofyear)
```

Let's count the number of trips by day. This is easily done if we first convert the "dayofyear" column to a factor:

```
trips$dayofyear <- factor(trips$dayofyear)
counts <- table(trips$dayofyear)
```

Do you see a trend in the number of trips by day? This trend may be more obvious if we plot the counts:

```
counts <- as.numeric(counts)
plot(274:365, counts, pch = 20)
```

One "trick" I used here was to first convert the output from `table` to a (numeric) vector, since `plot` expects numeric vectors as input.

## 43. Plot number of trips by day using ggplot2

To recreate the same plot using ggplot2, we first need to construct a data frame for plotting:

```
pdat <- data.frame(day = 274:365, counts = counts)
```

Now that we have a data frame for plotting, the ggplot2 code is rather straightforward:

```
p3 <- ggplot(pdat, aes(x = day, y = counts)) +  
  geom_point() +  
  theme_cowplot()
```

## 44. Where are the most frequently used Divvy stations?

Here we will create a plot that uses both the stations and trip data: a map of the stations, in which the area of each station is proportional to the number of trips from that station. To create this plot, we will have to *combine* the stations data and trip data. The number of trips by station is easily counted because we have converted the “from\_station\_name” to a factor:

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

The problem is that the stations in the trips table are not quite the same as the stations in the stations table:

```
length(stations$name)
```

```
length(names(counts))
```

```
length(intersect(stations$name, names(counts)))
```

The key is to select the elements of `counts` by name:

```
stations$trips <- as.numeric(counts[stations$name])
```

## 45. Where are the most frequently used Divvy stations?

With this new “trips” column in the stations data frame, creating the plot using ggplot2 is now mostly straightforward:

```
p4 <- ggplot(stations,
              aes(x = longitude,
                  y = latitude,
                  size = sqrt(trips))) +
  geom_point(shape = 21, color = "white",
             fill = "black") +
  theme_cowplot()
print(p4)
```

Notice that I scaled the size of the points by the *square root* of the number of trips so that the *area* of the points proportional to the number of trips.

## 46. Save your results

Save the results of your analysis.

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

## 47. Parting thoughts

1. Always keep track of your analysis code in a file (ideally, in a script that can be run).
2. Use “R Markdown” to document your analyses.
3. Use packages—don’t reinvent the wheel.
4. Email `help@rcc.uchicago.edu` for advice on using R on the RCC cluster. See also `https://github.com/rcc-uchicago/R-large-scale`.
5. I recommend the **workflowr** package for streamlining your data analyses and making them more reproducible, and easier to share:  
`https://github.com/jdblischak/workflowr`
6. Thank you!