02 - R and RStudio

ST 597 | Spring 2017
University of Alabama

02-Rintro.pdf

Intro

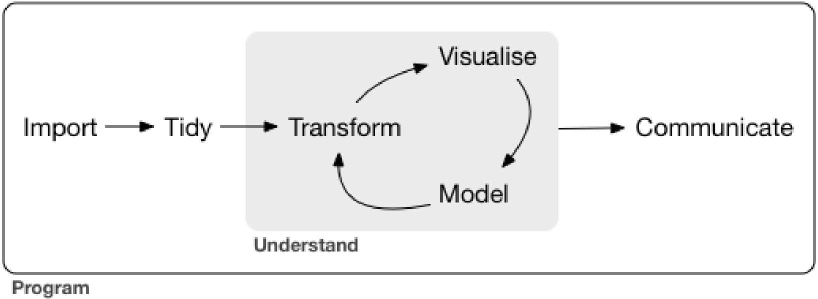
Installing R and RStudio

If you haven't already done so (on personal machines), install R and RStudio now:

- R (http://cran.r-project.org/)
- R Studio (http://www.rstudio.com/products/rstudio/download/)

And start up RStudio.

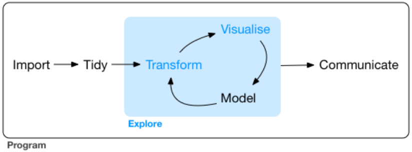
The Data Analytics Process



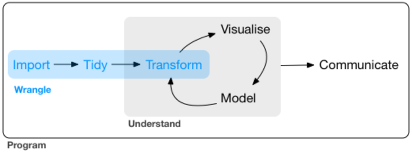
http://r4ds/diagrams/data-science.png/

Details

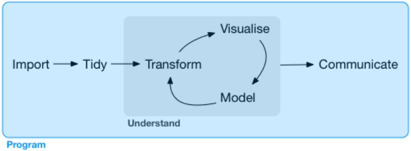
Explore



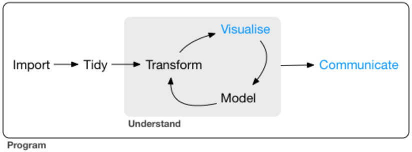
Wrangle



Program



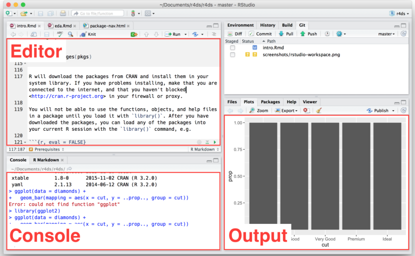
Communicate



http://r4ds/diagrams/

RStudio

RStudio IDE



~~~
ges(pkgs)

R will download the packages from CRAN and install them in your
system library. If you have problems installing, make that you are
connected to the internet, and that you haven't blocked
<http://cran.r-project.org> in your firewall or proxy.

You will not be able to use the functions, objects, and help files
in a package until you load it with `library()`. After you have
downloaded the packages, you can load any of the packages into
your current R session with the `library()` command, e.g.

```{r, eval = FALSE}
~~~

```
 xtable        1.8-0      2015-11-02 CRAN (R 3.2.0)
 yaml          2.1.13     2014-06-12 CRAN (R 3.2.0)
> ggplot(data = diamonds) +
+   geom_bar(mapping = aes(x = cut, y = ..prop.., group = cut))
Error: could not find function "ggplot"
> library(ggplot2)
> ggplot(data = diamonds) +
+   geom_bar(mapping = aes(x = cut, y = ..prop.., group = cut))
```

Staged	Status	Path
□	M	intro.Rmd
□	? ?	screenshots/rstudio-workspace.png

http://raw.githubusercontent.com/hadley/r4ds/master/screenshots/rstudio-layout.png

The RStudio IDE provides four “panes”. There are two primary panes:

- Console: Where you run “live” R code.
- Source: The editor where you can write scripts to save (for reproducibility).

The two other panes will show:

- Plots
- Help: Documentation for R functions
- Environment: the R objects you have created (also called *Workspace*)
- History: list of all the R code that is run in the console.
- . . . (many other things)

Customizing the Rstudio IDE

The RStudio IDE can be customized:

- `Tools -> Global Options ...`

Description of the options can be found here: http://support.rstudio.com/hc/en-us/articles/200549016-Customizing-RStudio

Under General:

- *Uncheck “Restore .RData into workspace at startup”*
- *Save workspace to .RData on exit to* ***Never***

R Projects

- It's good practice to keep all your files associated with a project in one place (data, scripts, figures, reports, etc.).
- RStudio facilitates this with **Projects**
  - Each Project has its own working directory, workspace, history, and source documents

R Project Details

- When a new project is created, RStudio:
  - Creates a project file (with an .Rproj extension) within the project directory. This file contains various project options and can also be used as a shortcut for opening the project directly from the filesystem.
  - Creates a hidden directory (named .Rproj.user) where project-specific temporary files (e.g. auto-saved source documents, window-state, etc.) are stored.
  - Loads the project into RStudio and display its name in the Projects toolbar (which is located on the far right side of the main toolbar).

RStudio documentation for Projects: `http://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects`

Your Turn #1 : Create a R Project

Create a new R Project for this class by clicking on dropdown at top right section of RStudio.

- It gives you the option to start a new directory (i.e., folder)
- Avoid using spaces in the project name (e.g., ST597)
- I usually create projects in google drive or dropbox so I can access the files from multiple computers
  - For computer lab, use X: drive

Using RStudio: Console Pane

Go to the console pane and let's do some math.

```
5+6-1
#> [1] 10
```

Save the results as an *object* named x

```
x = 5+6-1
```

To see the value of x, just enter x at the prompt

```
x
#> [1] 10
```

Note: Most resources for R will use <- (the two symbols < and -) instead of = to *assign* x the numeric value of 5+6-1.

R Variables

Make another object `y` and add it to `x`

```
y = 90
x + y
#> [1] 100
```

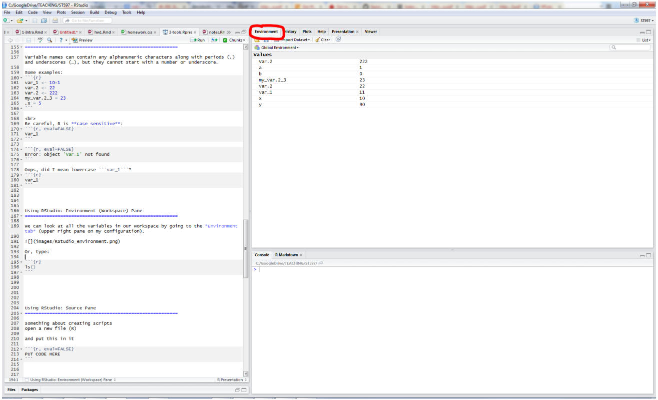
We can assign multiple variables to the same value

```
a = b = 0
```

```
a
#> [1] 0
b
#> [1] 0
```

Using RStudio: Environment (Workspace) Tab

We can look at all the variables in our workspace by going to the *Environment tab* (upper right pane on my configuration).



~~~
=================================================================

Variable names can contain any alphanumeric characters along with periods (.)
and underscores (_), but they cannot start with a number or underscore.

Some examples:
```{r}
var_1 <- 10+1
var.2 <- 22
var.2 <- 222
my_var.2_3 = 23
.x = 5
~~~


Be careful, R is **case sensitive**:
```{r, eval=FALSE}
Var_1
```

```{r, eval=FALSE}
Error: object 'Var_1' not found
```

Oops, did I mean lowercase ```var_1```?
```{r}
var_1
```

Using RStudio: Environment (Workspace) Pane
===

We can look at all the variables in our workspace by going to the *Environment
tab* (upper right pane on my configuration).

![](images/RStudio_environment.png)

Or, type:

```{r}
ls()
```

Using RStudio: Source Pane
===

something about creating scripts
open a new file (R)

and put this in it

```{r, eval=FALSE}
PUT CODE HERE
```
```

Values

| | |
|---|---|
| Var.2 | 222 |
| a | 1 |
| b | 0 |
| my_var.2_3 | 23 |
| var.2 | 22 |
| var_1 | 11 |
| x | 10 |
| y | 90 |

```
>
```

Or, type `ls()` for a list in the console:

```
ls()
#> [1] "a"   "b"   "course_url" "x"   "y"
```
```

R Packages

- Contributed **R Packages** are what makes R so great.
- An R package can contain: R functions, data, help pages, vignettes, non-R code (e.g., C++, Fortran)
- The Base R distribution actually consists of 14 packages
- There are 15 Recommended packages that come shipped with all binary distributions.
- And over 12,000 additional packages
- We will use several packages for this class; good thing they are so simple to use!

Using R Packages

It takes two steps to use the functions and data in an R package

1. Install the package
   - i.e. download the package to your computer
   - this only needs to be done one time
   - `install.packages()`
2. Load the package
   - i.e. tell R to look for the package functions and/or data
   - this needs to be done every time R is started (and you want to use the package)
   - `library()`

R Package Set-up (Lab Computers)

These steps are only necessary for Lab Computers:

1. Create sub-directory `RPackages` on your X: drive.
   - i.e. Create new folder in X: drive called *RPackages*
   - `X:/RPackages`
2. **TBD**

R Package Installation

1. *Install* the package on your computer
   - `Tools -> install.packages...`
   - Or, in the console type: `install.packages(pkgnames)`
   - Packages only need to be installed one time on a computer; do not *re*-install
2. Then, *load* into workspace to have access to all functions, datasets, and help files
   - Click on *Packages* tab and check boxes
   - Or, type `library(pkgname)` or `require(pkgname)`
3. Packages can be *updated* to ensure latest functionality and bug fixes
   - `Tools -> Check for Package Updates...`
   - Or, in console `update.packages()`
   - This just re-installs and writes over the old package

If you don't have root permission, then use the `lib=` argument.

Installing and Loading Packages

Your Turn #2

1. **Install** the package `tidyverse`
2. **Load** the packages into the workspace
3. Did you get any warnings? Make a note of these.
4. Ensure you have loaded it correctly:
   - Type `?mpg` in the console to see the help documentation for the data `mpg` from the `ggplot2` package.
   - Type `?ggplot` in the console to see the help documentation for the function `ggplot()`

Note on `tidyverse` package

- The `tidyverse` package is really just a wrapper to load several related R packages
  - `ggplot2` for graphics
  - `dplyr` for data manipulation
  - `tidyr` for getting data into tidy form
  - `readr` for loading in data
  - `tibble` for improved data frames
  - `purrr` for functional programming
- This provides a nice shortcut to load all of these packages with `library(tidyverse)` instead of each separately:

```r
#- the hard way
library(ggplot2)
library(dplyr)
library(tidyr)
library(readr)
library(tibble)
library(purrr)
```

Function conflicts

- Sometime you will come across functions from different packages that have the same name
    - For example, `filter` from package:dplyr and `filter` from package:stats
- If both packages are loaded, the function in the package that was loaded *last* will be invoked when calling the function.
- The other functions are said to be *masked*.
    - E.g., loading dplyr:

```
Attaching package: 'dplyr'
The following object is masked from 'package:stats':
    filter, lag
```

- If you want a specific function, add the package name separated by two colons

```
?filter
?stats::filter
?dplyr::filter
```

Note on using `library()`

- Packages only need to be *installed* (`install.packages()`) one time on your computer
- But packages need to be *loaded* (`library()`) every time you start a new R session

Using RStudio: Source Pane

- The source pane can save you lots of pain.
- This is where you will do most of your work.
- By executing commands from within the source editor rather than the console it is much easier to reproduce sequences of commands as well as package them for re-use as a function.
- Scripts can be saved for later use or sharing.

RStudio documentation: http://support.rstudio.com/hc/en-us/articles/200484448-Editing-and-Executing-Code

Your Turn #3

1. Create a new R script
    - `File -> New File -> R Script`
2. Copy and paste the following code (to make a scatter plot) into the new R script

```
#- Load the fuel economy data
library(tidyverse)  # note: mpg data is from ggplot2 package
data(mpg)           # loads the data (not necessary, but helpful
                    # to specify)

#- Make plot
ggplot(data=mpg) +
  geom_point(aes(x=displ, y=hwy))

#- Save plot
ggsave("mpg.pdf")

#- Save data
write_csv(mpg, path="mpg.csv")
```

Your Turn #4

3. Run the code in the console (Highlight all code and `Ctrl+Enter`)
4. Open the plot (mpg.pdf) in a pdf viewer and open the data (mpg.csv) in a spreadsheet program
   - where did you find these files?
5. Add the following properties to `geom_point()` and re-run:
   - Map the color of the points to the class (`color=class`)
   - Map the size of the points to the number of cylinders (hint: `size=cyl`)

Scripts for interactive analysis and reproducibility

- Working in the source pane instead of the console will save you time as you interact with the data.
- For example, you now have the code to produce a nice scatter plot with control for point size and colors.
- Working with a script will help with Reproducible Data Analysis
- Dangers of Point and Click Approach
- The # symbol marks a comment. The rest of the line is commented (not read by R).

```
y = 10        # set y equal to 10
y = 5         # set y equal to 5
# y = 1       # set y equal to 1  (Note: this will not be run by R)
y
#> [1] 5
```

Your Turn #5

Save your plot script in the project directory.

1. Create a subdirectory `R` to keep all your R scripts.
2. Use the extension (.R) for R scripts
   - For example: `mpg-plot.R`
3. Save `mpg-plot.R` in the `R` subdirectory
4. (Optional) Create subdirectories `data` and `figures`. Modify the script to add the components to the correct subdirectory
   - `ggsave("figures/mpg.pdf")`
   - `write_csv(mpg, path="data/mpg.csv")`

History

- RStudio keeps track of everything entered into the console in the **History** tab (top right pane in my config)
- Here you can send lines of code to the console or source
- When working in the console, you can also use `Up-arrow` to scroll through recent commands
- Or type the first few characters of your command and use `Ctrl+Up-arrow`
  - Example: Type `gg`, then `Ctrl+Up-arrow` to see a list of your recent commands that started with "gg"
- It is a good idea to save anything from the history that you may need again in a script.
- If you are working under an R Project, then your history should save automatically and be available next time to start up that project.

RStudio documentation: `http://support.rstudio.com/hc/en-us/articles/200526217-Command-History`

RStudio Keyboard Shortcuts

- You can improve your productivity by learning keyboard shortcuts
- In editor:
  - `Ctrl+Enter`: send code to console
    - (`Command+Enter` on Mac)
  - `Ctrl+2`: move cursor to console
  - `Ctrl+a`: select all
- In console
  - `Up_arrow`: retrieve previous command
  - `Ctrl+up arrow`: search commands
  - `Ctrl+1`: move cursor to editor
- Tab complete
  - start typing a variable or function name and then `Tab`
  - For functions, enter function name then parenthesis “(” then `Tab` and it will show you possible function arguments.

```
mean( + Tab
```

  - We will explore this more when we introduce functions

RStudio Help Pages

- Check out `Help` tab
- RStudio Main Help Page
- cheat sheets
- RStudio IDE
- Keyboard Shortcuts
  - Or `Alt+Shift+K`
- Getting R Help

Using R

R Resources

There is no shortage of free resources for learning R.

The official reference list is here: http://cran.r-project.org/other-docs.html

- Look for options that are more recent. E.g.,
  - Base R Cheatsheet
  - http://cran.r-project.org/doc/contrib/Baggott-refcard-v2.pdf
  - http://cran.r-project.org/doc/contrib/Torfs+Brauer-Short-R-Intro.pdf

Bank Discrimination?

Did a bank discriminatorily pay higher starting salaries to men than to women?

Let's examine some data of beginning salaries for entry-level clerical employees hired by the bank between 1969 and 1977.

The data can be found on the webpage at: `http://mdporter.github.io/ST597/data/salary.csv`

Load the Starting Salary Data into R

We can read this into R several ways:

1. From Rstudio: `Tools -> Import Dataset -> From Web URL ...`
2. Download file and use `Tools -> Import Dataset -> From Text File ...`
3. Use command line (reproducible option - save in script)

```
library(tidyverse)
url = 'http://mdporter.github.io/ST597/data/salary.csv'  # use quot
salary = read_csv(url)          # name the data: salary
```

View the data

1. RStudio Viewer: Go to `Environment` tab, and click on spreadsheet symbol next to `salary`.
2. Or, type `View(salary)` (with uppercase V)

   Notice, we can also just type the data name `salary` into the console

```
salary
#> # A tibble: 93 × 2
#>    Salary    Sex
#>     <int>  <chr>
#> 1     3900 Female
#> 2     4020 Female
#> 3     4290 Female
#> 4     4380 Female
#> 5     4380 Female
#> 6     4380 Female
#> 7     4380 Female
#> 8     4380 Female
#> 9     4440 Female
#> 10    4500 Female
#> # ... with 83 more rows
```

Quick view of the data

The function `glimpse()` provides a brief view of the data

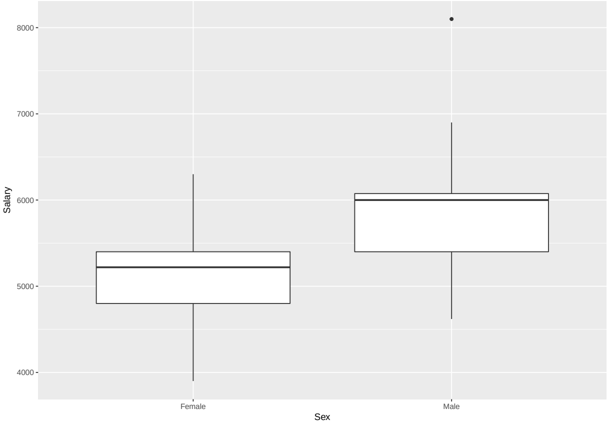
```
glimpse(salary)
#> Observations: 93
#> Variables: 2
#> $ Salary <int> 3900, 4020, 4290, 4380, 4380, 4380, 4380, 4380, 4440
#> $ Sex    <chr> "Female", "Female", "Female", "Female", "Female", "F
```

The function `summary()` gives an overall summary

```
summary(salary)
#>      Salary          Sex           
#>  Min.   :3900   Length:93         
#>  1st Qu.:4980   Class :character  
#>  Median :5400   Mode  :character  
#>  Mean   :5420                     
#>  3rd Qu.:6000                     
#>  Max.   :8100                     
```

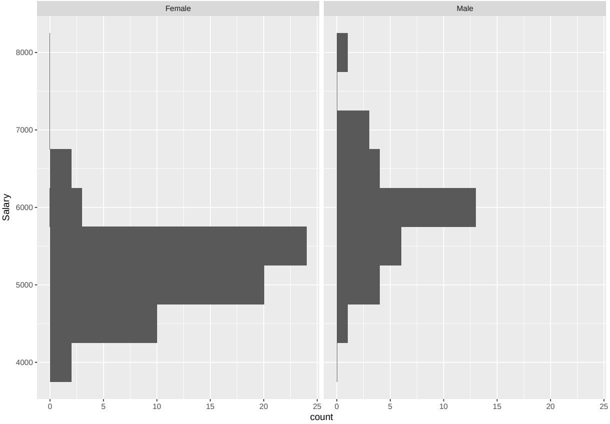
Make Boxplots

```
ggplot(data=salary) +
  geom_boxplot(aes(x=Sex, y=Salary))
```



Make Histograms

```
ggplot(data=salary) +
  geom_histogram(aes(x=Salary), binwidth=500) +
  facet_wrap(~Sex) + coord_flip()
```



Get Average Salary by Sex

```
salary %>%                           # start with salary data
  group_by(Sex) %>%                  # group or split by `Sex` column
  summarize(avg=mean(Salary)) # get the mean of `Salary` column
#> # A tibble: 2 × 2
#>      Sex      avg
#>    <chr>    <dbl>
#> 1 Female 5138.852
#> 2   Male 5956.875
                                     #  for each group
```

Comparing summary statistics

The average male salary was $`818.02` **larger** than the average female salary.

Can we conclude that there is gender discrimination?

Comparing summary statistics

The average male salary was \$818.02 **larger** than the average female salary.

Can we conclude that there is gender discrimination?

Your Turn #7

1. Would you feel any different if the reported difference had less or more digits? E.g.,
   - \$818
   - \$818.022541
   - around \$800
2. Is this an experimental or observational study?