Lecture 5: Meet R

ECON 480 - Econometrics - Fall 2018

Ryan Safner

September 12, 2018

Writing and Saving R Code

Objects in R

Data Frames

Quick Data Analysis Example

HOOD COLLEGE

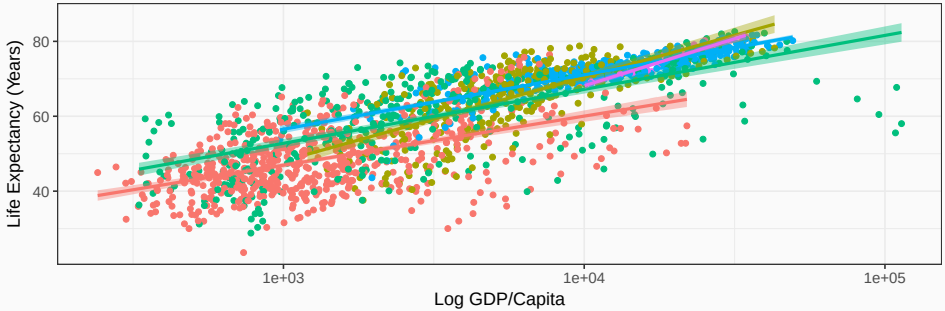
Excel, Stata, R

Excel	Stata	R
MS Office License	Expensive License	Free!
Proprietary	Proprietary	Open Source
Default, used in finance	Primary for economists	Largest use by data scientists
Not reproducible	Reproducible (.do files)	Reproducible (.R files)
Can't incorporate into docs	Can't incorporate into docs	Can run in chunks within docs
Very limited extensions	Many packages	Most packages written for R first
Point-and-click	Point-and-click or command line	Almost exclusively command line
Limited formulas	Just one command per task	Many alternative commands

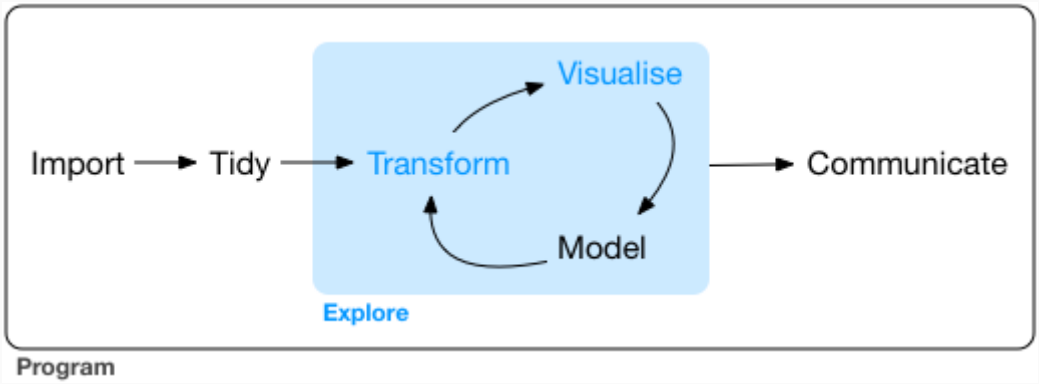
HOOD COLLEGE

Stata and Excel Can't Do This (Right in the Slides)

```
ggplot(data = gapminder, aes(x = gdpPercap,
    y = lifeExp, color = continent, fill= continent))+
    geom_point()+geom_smooth(method = "lm") +
    scale_x_log10()+ylab("Life Expectancy (Years)")+
    xlab("Log GDP/Capita")
```

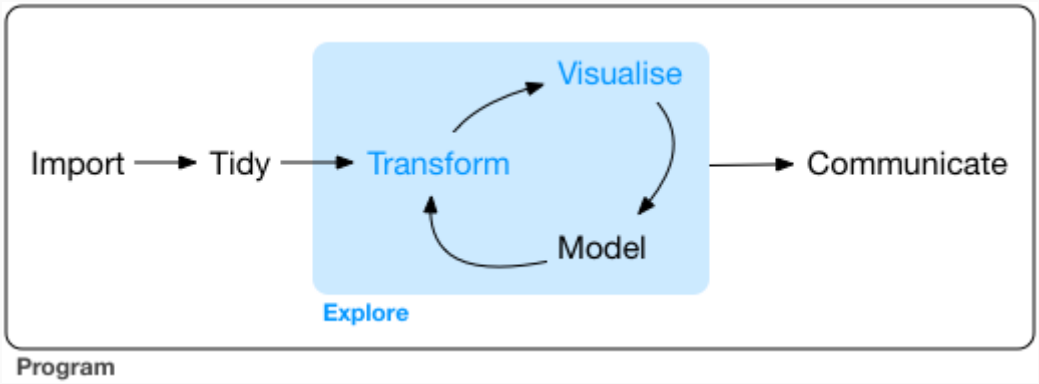


HOOD COLLEGE



Workflow

- Need software that can import, tidy, analyze, plot, and present data



Workflow

- Need software that can import, tidy, analyze, plot, and present data
- R can do all of this (with *R Markdown*, all in the same document)

- You are literally learning a new language, complete with grammar and syntax rules, and specific vocabulary

- You are literally learning a new language, complete with grammar and syntax rules, and specific vocabulary
- R like all command line programming is *very literal*, a single typo or misplaced comma will lead to a different outcome than you intended, or fail completely

HOOD COLLEGE

"There's an implied contract between you and R: it will do the tedious computation for you, but in return, you must be completely precise in your instructions. Typos matter. Case matters."

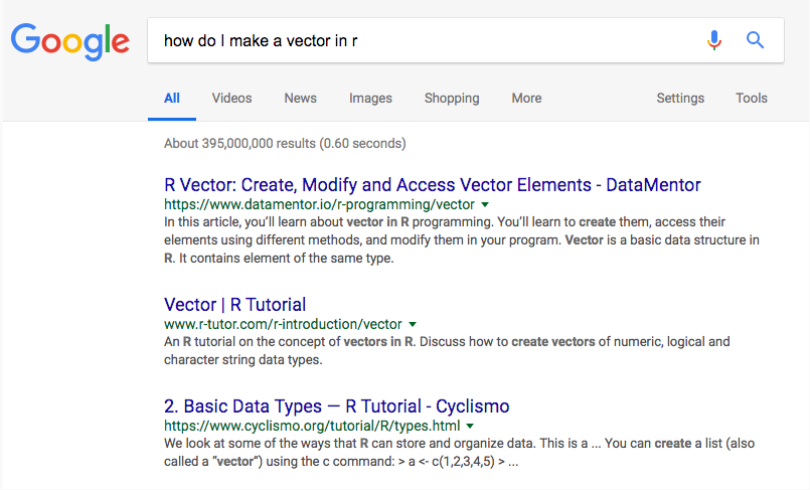
R for Data Science

Hadley Wickham

Chief Scientist, R Studio

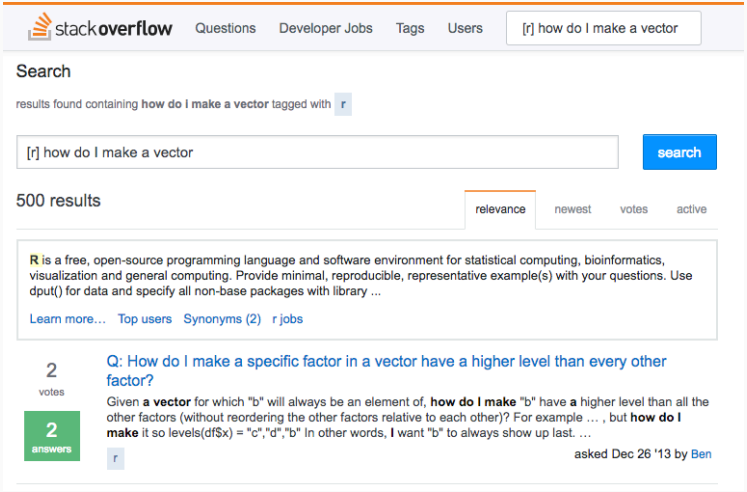
HOOD COLLEGE

Say Hello to My Little Friend

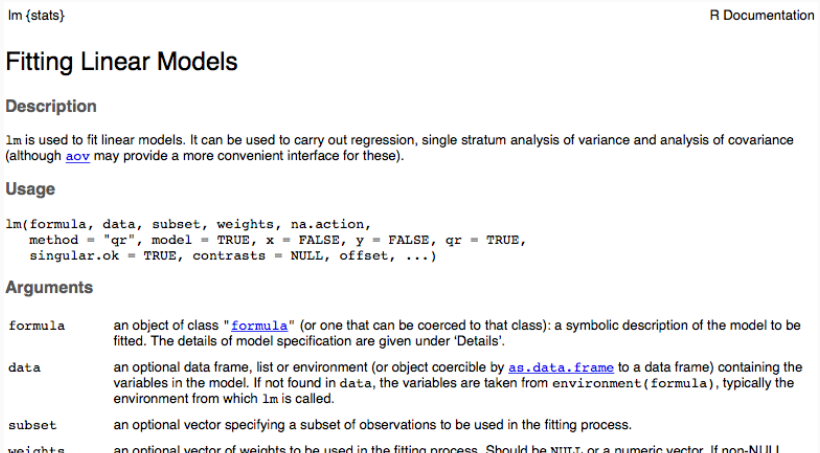


HOOD COLLEGE

Say Hello to Your New BEST Friend



```
#type help("functionname") to get documentation on the command
help("lm")
```



lm {stats} R Documentation

Fitting Linear Models

Description

`lm` is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although aov may provide a more convenient interface for these).

Usage

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

Arguments

`formula`	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`data`	an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `lm` is called.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`weights`	an optional vector of weights to be used in the fitting process. Should be NULL or a numeric vector. If non-NULL

Writing and Saving R Code

Naming Conventions

- Object names cannot start with a digit or contain a space or comma

HOOD COLLEGE

Naming Conventions

- Object names cannot start with a digit or contain a space or comma
- FOR THE LOVE OF GOD AVOID SPACES IN GENERAL

HOOD COLLEGE

Naming Conventions

- Object names cannot start with a digit or contain a space or comma
- FOR THE LOVE OF GOD AVOID SPACES IN GENERAL
  - You've seen webpages intended to be called `"my webpage in html"` turned into `http://my%20webpage%20in%20html.html`

HOOD COLLEGE

Naming Conventions

- Object names cannot start with a digit or contain a space or comma
- FOR THE LOVE OF GOD AVOID SPACES IN GENERAL
  - You've seen webpages intended to be called `"my webpage in html"` turned into `http://my%20webpage%20in%20html.html`
  - Consider both your R objects and your files and folder names on your computer...(`/School/ECON_480_Econometrics/Homeworks_and_Data/`)

HOOD COLLEGE

Naming Conventions

- Object names cannot start with a digit or contain a space or comma
- FOR THE LOVE OF GOD AVOID SPACES IN GENERAL
  - You've seen webpages intended to be called `"my webpage in html"` turned into `http://my%20webpage%20in%20html.html`
  - Consider both your R objects and your files and folder names on your computer...(`/School/ECON_480_Econometrics/Homeworks_and_Data/`)
- It will be wise to adopt some consistent standard for demarcating names:

```
i.use.snake.case
otherPeopleUseCamelCase
some_people_use_underscores
And_aFew.People_RENOUNCEconvention
```

HOOD COLLEGE

Working Directory

- *R* assumes a default (often inconvenient) working directory on your computer

HOOD COLLEGE

Working Directory

- *R* assumes a default (often inconvenient) working directory on your computer
- Find out where *R* currently is pulling or saving files to/from with **getwd()**

HOOD COLLEGE

Working Directory

- *R* assumes a default (often inconvenient) working directory on your computer
- Find out where *R* currently is pulling or saving files to/from with **getwd()**
- Change the working directory to wherever you plan on keeping your related data and documents with **setwd("filepath")**

HOOD COLLEGE

Working Directory

- *R* assumes a default (often inconvenient) working directory on your computer
- Find out where *R* currently is pulling or saving files to/from with **getwd()**
- Change the working directory to wherever you plan on keeping your related data and documents with **setwd("filepath")**
    - **filepath** refers to the location of the folder on your computer, e.g. `~/Dropbox/Teaching/Hood College/ECON 480 - Econometrics/Data`

HOOD COLLEGE

- *R* assumes a default (often inconvenient) working directory on your computer
- Find out where *R* currently is pulling or saving files to/from with **getwd()**
- Change the working directory to wherever you plan on keeping your related data and documents with **setwd("filepath")**
    - **filepath** refers to the location of the folder on your computer, e.g. `~/Dropbox/Teaching/Hood College/ECON 480 - Econometrics/Data`
    - OS-specific to Windows vs. Mac vs. Linux

HOOD COLLEGE

Writing and Saving R Code: Tips

- Comment, comment, comment!

```
# Run regression of y on x, save as reg1
reg1<-lm(y~x, data=data) #runs regression
summary(reg1$coefficients) #prints coefficients
```

HOOD COLLEGE

Writing and Saving R Code: Tips

- Comment, comment, comment!
- Use the hashtag # to start a comment (R ignores everything on that line after the hashtag)

```
# Run regression of y on x, save as reg1
reg1<-lm(y~x, data=data) #runs regression
summary(reg1$coefficients) #prints coefficients
```

HOOD COLLEGE

Writing and Saving R Code: Tips

- Comment, comment, comment!
- Use the hashtag # to start a comment (R ignores everything on that line after the hashtag)
- Can be made its own line or at the end of lines

```
# Run regression of y on x, save as reg1
reg1<-lm(y~x, data=data) #runs regression
summary(reg1$coefficients) #prints coefficients
```

HOOD COLLEGE

Writing and Saving R Code: Tips

- Comment, comment, comment!
- Use the hashtag # to start a comment (R ignores everything on that line after the hashtag)
- Can be made its own line or at the end of lines

```
# Run regression of y on x, save as reg1
reg1<-lm(y~x, data=data) #runs regression
summary(reg1$coefficients) #prints coefficients
```

- Save often!

HOOD COLLEGE

Writing and Saving R Code: Tips

- Comment, comment, comment!
- Use the hashtag # to start a comment (R ignores everything on that line after the hashtag)
- Can be made its own line or at the end of lines

```
# Run regression of y on x, save as reg1
reg1<-lm(y~x, data=data) #runs regression
summary(reg1$coefficients) #prints coefficients
```

- Save often!
  - Better yet, ask me about version control and GitHub (later)

HOOD COLLEGE

1. Command line/Console: writing each command by itself and copying down the result as needed

1. Command line/Console: writing each command by itself and copying down the result as needed
    - Great for testing individual commands to see what happens

1. Command line/Console: writing each command by itself and copying down the result as needed
    - Great for testing individual commands to see what happens
    - Not reproducible! Not saved! NOT RECOMMENDED!

1. Command line/Console: writing each command by itself and copying down the result as needed
    - Great for testing individual commands to see what happens
    - Not reproducible! Not saved! NOT RECOMMENDED!
2. .R files: A sequence of commands (and hopefully comments) saved as a script, the entire script is run all at once

HOOD COLLEGE

1. Command line/Console: writing each command by itself and copying down the result as needed
    - Great for testing individual commands to see what happens
    - Not reproducible! Not saved! NOT RECOMMENDED!
2. .R files: A sequence of commands (and hopefully comments) saved as a script, the entire script is run all at once
    - Can test individual commands in command line and then put good commands in *.R* file

HOOD COLLEGE

1. Command line/Console: writing each command by itself and copying down the result as needed
    - Great for testing individual commands to see what happens
    - Not reproducible! Not saved! NOT RECOMMENDED!
2. .R files: A sequence of commands (and hopefully comments) saved as a script, the entire script is run all at once
    - Can test individual commands in command line and then put good commands in *.R* file
    - Equivalent to a *.do* file for Stata

HOOD COLLEGE

1. Command line/Console: writing each command by itself and copying down the result as needed
    - Great for testing individual commands to see what happens
    - Not reproducible! Not saved! NOT RECOMMENDED!
2. .R files: A sequence of commands (and hopefully comments) saved as a script, the entire script is run all at once
    - Can test individual commands in command line and then put good commands in *.R* file
    - Equivalent to a *.do* file for Stata
    - Reproducible, saved, commented

HOOD COLLEGE

3. R Markdown (*.Rmd*) files: A plain text document written in *R Markdown* language

3. R Markdown (*.Rmd*) files: A plain text document written in *R Markdown* language
    - Allows for individual chunks of *R* code to be run individually (great for testing one command instead of all at once)

3. R Markdown (*.Rmd*) files: A plain text document written in *R Markdown* language
    - Allows for individual chunks of *R* code to be run individually (great for testing one command instead of all at once)
    - Reproducible, saved, commented as if a normal document

HOOD COLLEGE

3. R Markdown (*.Rmd*) files: A plain text document written in *R Markdown* language
    - Allows for individual chunks of *R* code to be run individually (great for testing one command instead of all at once)
    - Reproducible, saved, commented as if a normal document
    - Can write an entire document (text, equations, R commands, figures, tables, etc) with one file!

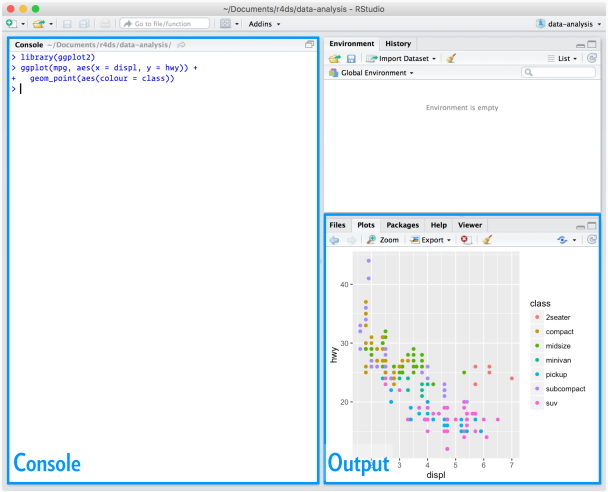
HOOD COLLEGE

3. R Markdown (*.Rmd*) files: A plain text document written in *R Markdown* language
    - Allows for individual chunks of *R* code to be run individually (great for testing one command instead of all at once)
    - Reproducible, saved, commented as if a normal document
    - Can write an entire document (text, equations, R commands, figures, tables, etc) with one file!
    - Can export to html, MS Word, Beamer, etc!

HOOD COLLEGE

3. R Markdown (*.Rmd*) files: A plain text document written in *R Markdown* language
    - Allows for individual chunks of *R* code to be run individually (great for testing one command instead of all at once)
    - Reproducible, saved, commented as if a normal document
    - Can write an entire document (text, equations, R commands, figures, tables, etc) with one file!
    - Can export to html, MS Word, Beamer, etc!
    - Markdown is a language that is intuitive, simple, human- and machine-readable

HOOD COLLEGE



Rstudio Windows

Agenda for Today and for the Future

- **Today**: practice using the Command Line/Console in *R*

HOOD COLLEGE

Agenda for Today and for the Future

- **Today**: practice using the Command Line/Console in *R*
  - Learn different commands and results relevant to data analysis

HOOD COLLEGE

Agenda for Today and for the Future

- **Today**: practice using the Command Line/Console in *R*
  - Learn different commands and results relevant to data analysis
  - By the end, saving a script as a *.R* file

HOOD COLLEGE

Agenda for Today and for the Future

- **Today**: practice using the Command Line/Console in *R*
  - Learn different commands and results relevant to data analysis
  - By the end, saving a script as a *.R* file
- **Later**: *R Markdown* and the benefits to plain text, literate programming, and workflow management

HOOD COLLEGE

- First, ecognize that R can be used as a simple calculator

```
2+2
```

```
## [1] 4
```

```
sqrt(100/4)
```

```
## [1] 5
```

```
log(5)
```

```
## [1] 1.609438
```

- First, ecognize that R can be used as a simple calculator

```
2+2
```

```
## [1] 4
```

```
sqrt(100/4)
```

```
## [1] 5
```

```
log(5)
```

```
## [1] 1.609438
```

- This will be helpful later when we want to transform variables

Using Packages

- Packages are extensions designed by users

HOOD COLLEGE

Using Packages

- Packages are extensions designed by users
- Remember, R is open source, packages are usually published first on Github

HOOD COLLEGE

Using Packages

- Packages are extensions designed by users
- Remember, R is open source, packages are usually published first on Github
- Official packages distributed and documented through CRAN

HOOD COLLEGE

Using Packages

- Packages are extensions designed by users
- Remember, R is open source, packages are usually published first on Github
- Official packages distributed and documented through CRAN
- To use a (previously-installed) package (note the “”):

```
library(”packagename”)
```

HOOD COLLEGE

Using Packages

- Packages are extensions designed by users
- Remember, R is open source, packages are usually published first on Github
- Official packages distributed and documented through CRAN
- To use a (previously-installed) package (note the “”):

```
library("packagename")
```

- If you do not have a package, they are easy to install with (yes, note the plural “s”)

```
install.packages("packagename")
```

HOOD COLLEGE

- Here is a list of the most important packages you will probably use for things relevant to econometrics

Package name	Use
`ggplot2`	Rendering beautiful graphics (scatterplots, histograms, etc)
`stargazer`	Rendering professioanl looking regression output tables
`dplyr`	Manipulating data much more intuitively
`sandwich`	More tools for regression, particularly robust SE's
`tidyverse`	An epic *meta*package of `ggplot2, dplyr` and other popular packages

- Here is a list of the most important packages you will probably use for things relevant to econometrics

Package name	Use
`ggplot2`	Rendering beautiful graphics (scatterplots, histograms, etc)
`stargazer`	Rendering professioanl looking regression output tables
`dplyr`	Manipulating data much more intuitively
`sandwich`	More tools for regression, particularly robust SE's
`tidyverse`	An epic *meta*package of `ggplot2, dplyr` and other popular packages

- Google each of these for more documentation

HOOD COLLEGE

- Here is a list of the most important packages you will probably use for things relevant to econometrics

Package name	Use
`ggplot2`	Rendering beautiful graphics (scatterplots, histograms, etc)
`stargazer`	Rendering professioanl looking regression output tables
`dplyr`	Manipulating data much more intuitively
`sandwich`	More tools for regression, particularly robust SE's
`tidyverse`	An epic *meta*package of `ggplot2, dplyr` and other popular packages

- Google each of these for more documentation
- We will explore each of these in more detail later

Objects in R

Vectors

- The simplest data structure in R is a vector, simply a collection of objects

HOOD COLLEGE

Vectors

- The simplest data structure in R is a vector, simply a collection of objects
- To construct a vector, use the "combine/concatenate" function "**c()**"

HOOD COLLEGE

Vectors

- The simplest data structure in R is a vector, simply a collection of objects
- To construct a vector, use the "combine/concatenate" function "**c()**"
- **Example**: a vector of the numbers 1 through 5

HOOD COLLEGE

Vectors

- The simplest data structure in R is a vector, simply a collection of objects
- To construct a vector, use the "combine/concatenate" function "**c()**"
- **Example**: a vector of the numbers 1 through 5

```
c(1,2,3,4,5)
```

```
## [1] 1 2 3 4 5
```

- We can also build vectors via generating series

HOOD COLLEGE

Vectors

- The simplest data structure in R is a vector, simply a collection of objects
- To construct a vector, use the "combine/concatenate" function "**c()**"
- **Example**: a vector of the numbers 1 through 5

```
c(1,2,3,4,5)
```

```
## [1] 1 2 3 4 5
```

- We can also build vectors via generating series

HOOD COLLEGE

Vectors

- The simplest data structure in R is a vector, simply a collection of objects
- To construct a vector, use the "combine/concatenate" function "**c()**"
- **Example**: a vector of the numbers 1 through 5

```
c(1,2,3,4,5)
```

```
## [1] 1 2 3 4 5
```

- We can also build vectors via generating series

```
1:5
```

```
## [1] 1 2 3 4 5
```

HOOD COLLEGE

Vectors II

- We can perform mathematical operations on a vector as a whole:

```
sum(1:5)
```

```
## [1] 15
```

```
mean(1:5)
```

```
## [1] 3
```

HOOD COLLEGE

Storing Objects

- R is an object-oriented programming language

HOOD COLLEGE

Storing Objects

- R is an object-oriented programming language
- We will almost always store data in **objects** and run **functions** on objects

HOOD COLLEGE

Storing Objects

- R is an object-oriented programming language
- We will almost always store data in **objects** and run **functions** on objects
    - Objects are assigned values: `objectname<-value`

HOOD COLLEGE

Storing Objects

- R is an object-oriented programming language
- We will almost always store data in **objects** and run **functions** on objects
  - Objects are assigned values: `objectname<-value`
  - Functions take the form: `functionname(objectname)`

HOOD COLLEGE

Storing Objects

- R is an object-oriented programming language
- We will almost always store data in **objects** and run **functions** on objects
  - Objects are assigned values: `objectname<-value`
  - Functions take the form: `functionname(objectname)`
- Storing a vector as an object called "*x*" using the assignment operator "`<-`"

HOOD COLLEGE

Storing Objects

- R is an object-oriented programming language
- We will almost always store data in **objects** and run **functions** on objects
  - Objects are assigned values: `objectname<-value`
  - Functions take the form: `functionname(objectname)`
- Storing a vector as an object called "*x*" using the assignment operator "`<-`"

```
x<-c(1,2,3,4,5)
```

- To inspect an object, we simply call it up by name

HOOD COLLEGE

Storing Objects

- R is an object-oriented programming language
- We will almost always store data in **objects** and run **functions** on objects
  - Objects are assigned values: `objectname<-value`
  - Functions take the form: `functionname(objectname)`
- Storing a vector as an object called "*x*" using the assignment operator "`<-`"

```
x<-c(1,2,3,4,5)
```

- To inspect an object, we simply call it up by name

HOOD COLLEGE

Storing Objects

- R is an object-oriented programming language
- We will almost always store data in **objects** and run **functions** on objects
  - Objects are assigned values: `objectname<-value`
  - Functions take the form: `functionname(objectname)`
- Storing a vector as an object called "*x*" using the assignment operator "`<-`"

```
x<-c(1,2,3,4,5)
```

- To inspect an object, we simply call it up by name

```
x
```

```
## [1] 1 2 3 4 5
```

HOOD COLLEGE

Storing Objects: Text

- Vectors need not contain only numbers:

Storing Objects: Text

- Vectors need not contain only numbers:

```
list<-c("red", "blue", "purple")
```

- Note above, *strings* or *characters* of text require quotation marks around them

HOOD COLLEGE

- Vectors need not contain only numbers:

```
list<-c("red", "blue", "purple")
```

- Note above, *strings* or *characters* of text require quotation marks around them
- We can inspect the object via calling it by its name, or more formally, telling *R* to **print()** the contents of the object

- Vectors need not contain only numbers:

```
list<-c("red", "blue", "purple")
```

- Note above, *strings* or *characters* of text require quotation marks around them
- We can inspect the object via calling it by its name, or more formally, telling *R* to **print()** the contents of the object

```
list
```

```
## [1] "red"    "blue"   "purple"
```

```
print(list)
```

```
## [1] "red"    "blue"   "purple"
```

- Vectors **must** contain the same type of elements (e.g. numerical or text)

```
mixed<-c(5, pi, TRUE, 4.3, "cabbage")
class(mixed)
```

```
## [1] "character"
```

- Vectors **must** contain the same type of elements (e.g. numerical or text)
- Technically this refers to **atomic vectors** (nearly all vectors are atomic)

```
mixed<-c(5, pi, TRUE, 4.3, "cabbage")
class(mixed)
```

```
## [1] "character"
```

- Vectors **must** contain the same type of elements (e.g. numerical or text)
- Technically this refers to **atomic vectors** (nearly all vectors are atomic)
- There can be a non-atomic vector, known as a **list** (see below)

```
mixed<-c(5, pi, TRUE, 4.3, "cabbage")
class(mixed)
```

```
## [1] "character"
```

- Vectors **must** contain the same type of elements (e.g. numerical or text)
- Technically this refers to **atomic vectors** (nearly all vectors are atomic)
- There can be a non-atomic vector, known as a **list** (see below)
- Vectors with "mixed" types will convert all elements to the lowest-common denominator, e.g. character

```
mixed<-c(5, pi, TRUE, 4.3, "cabbage")
class(mixed)
```

```
## [1] "character"
```

- Vectors **must** contain the same type of elements (e.g. numerical or text)
- Technically this refers to **atomic vectors** (nearly all vectors are atomic)
- There can be a non-atomic vector, known as a **list** (see below)
- Vectors with "mixed" types will convert all elements to the lowest-common denominator, e.g. character
- You can always check the type of vector using **class()**

```
mixed<-c(5, pi, TRUE, 4.3, "cabbage")
class(mixed)
```

```
## [1] "character"
```

HOOD COLLEGE

- Numeric (aka “double”), as it sounds, can perform mathematical operations

```
numeric<-c(1,2,3,4,5)
```

- Numeric (aka “double”), as it sounds, can perform mathematical operations

```
numeric<-c(1,2,3,4,5)
```

- Logical is a series of binary elements that can either be TRUE or FALSE

```
logical<-c(TRUE,FALSE,FALSE,TRUE)
```

HOOD COLLEGE

- Numeric (aka “double”), as it sounds, can perform mathematical operations

```
numeric<-c(1,2,3,4,5)
```

- Logical is a series of binary elements that can either be TRUE or FALSE

```
logical<-c(TRUE,FALSE,FALSE,TRUE)
```

- Character is a string of text: letters, numbers, and symbols, cannot perform mathematical operations

```
character<-c("one","two","7","orange")
```

Factors

- Factor is a special type of character variable, often used to indicate membership in one of several possible categories, called **levels** (e.g. for plotting, or conditional statistics and data work)

```
students<-factor(c("freshman", "senior", "senior", "junior", "freshman",
                   "sophomore", "freshman"))
levels(students) #extract unique levels
```

```
## [1] "freshman"  "junior"    "senior"    "sophomore"
```

```
nlevels(students) #count the number of levels
```

```
## [1] 4
```

HOOD COLLEGE

```
table(students) #tabulate number of values for each level
```

```
## students
##  freshman    junior    senior sophomore
##         3         1         2         1
```

- Again, check the type of data with **class()**

```
class(x)
```

```
## [1] "numeric"
```

```
x<-as.character(x)
class(x)
```

```
## [1] "character"
```

```
x<-as.numeric(x)
```

- Again, check the type of data with **class()**
- Change the type with **as.classname()**

```
class(x)
```

```
## [1] "numeric"
```

```
x<-as.character(x)
class(x)
```

```
## [1] "character"
```

```
x<-as.numeric(x)
```

- Again, check the type of data with **class()**
- Change the type with **as.classname()**
  - Note you can't turn characters into numeric (if there's no numbers), but you can turn numeric into characters

```
class(x)
```

```
## [1] "numeric"
```

```
x<-as.character(x)
class(x)
```

```
## [1] "character"
```

```
x<-as.numeric(x)
```

- Use [square brackets] to isolate elements of a vector for commands

```
print(list) #Our original list
```

```
## [1] "red"    "blue"   "purple"
```

```
list[1] #Inspect first element
```

```
## [1] "red"
```

```
list[c(1,3)] #Inspect first and third elements
```

```
## [1] "red"    "purple"
```

HOOD COLLEGE

- Use **[square brackets]** to isolate elements of a vector for commands

```
list[2]<-"orange" #Change second element
print(list) #Our new list
```

```
## [1] "red"    "orange" "purple"
```

- Keeping our first vector *x*, let's create another object *y*

```
y<-10
```

- Keeping our first vector *x*, let's create another object *y*

```
y<-10
```

- Let's create a third object *z*, which is the product of *x* and *y*

```
z<-x*y
z
```

```
## [1] 10 20 30 40 50
```

Redefining Objects

- Objects and variables maintain their value until they are changed. We can redefine x as 6 simply with another command to define x.

```
x
```

```
## [1] 1 2 3 4 5
```

```
x<-6
x
```

```
## [1] 6
```

Functions

- Most of R: `functionname(objectname)`

HOOD COLLEGE

Functions

- Most of R: `functionname(objectname)`
- A function produces a (hopefully useful) output based on the input that it recieves.

HOOD COLLEGE

Functions

- Most of R: `functionname(objectname)`
- A function produces a (hopefully useful) output based on the input that it recieves.
- Functions can be recognized by the parentheses () at the end of their names.

HOOD COLLEGE

Functions

- Most of R: `functionname(objectname)`
- A function produces a (hopefully useful) output based on the input that it recieves.
- Functions can be recognized by the parentheses () at the end of their names.
- The `c()` command that produces a vector, was an example of a function.

HOOD COLLEGE

- Mathematical/Statistical Functions

Function	Use	Example
`sum()`	Takes the sum of an object	`sum(1:10)`
`mean()`	Takes the arithmetic mean of an object	`mean(1:10)`
`rnorm()`	Takes a number of draws (e.g. 5) from a normal distribution	`rnorm(5)`
`round()`	Rounds an object (e.g. x) to (e.g. 2) decimal places	`round(x,2)`

HOOD COLLEGE

- Mathematical/Statistical Functions

Function	Use	Example
`sum()`	Takes the sum of an object	`sum(1:10)`
`mean()`	Takes the arithmetic mean of an object	`mean(1:10)`
`rnorm()`	Takes a number of draws (e.g. 5) from a normal distribution	`rnorm(5)`
`round()`	Rounds an object (e.g. x) to (e.g. 2) decimal places	`round(x,2)`

- Note functions can have functions in their arguments, e.g. `round(rnorm(5),2)`

HOOD COLLEGE

Data Frames

- The most important object in R is a data frame, used for statistics, plots, regressions, etc

```
gapminder
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
```

- The most important object in R is a data frame, used for statistics, plots, regressions, etc
  - "Rectangular" data (i.e. "spreadsheet"): rows are observations, columns are variables

```
gapminder
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
```

- The most important object in R is a data frame, used for statistics, plots, regressions, etc
  - "Rectangular" data (i.e. "spreadsheet"): rows are observations, columns are variables
  - Can hold variables of different classes

```
gapminder
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
```

- The most important object in R is a data frame, used for statistics, plots, regressions, etc
  - "Rectangular" data (i.e. "spreadsheet"): rows are observations, columns are variables
  - Can hold variables of different classes
  - All vectors (columns) must have the same length!

```
gapminder
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
```

- We often import existing data into a dataframe (see importing data below)

```
v1<-c(10,20,30,45,60) # A list of integers
v2<-LETTERS[1:5] # The first 5 letters
v3<-round(rnorm(5),2) #5 random draws from normal distr, rounded to 2 decimal pl
v4<-c("Apples","Oranges","Bananas","Kiwi","Watermelon") #Fruits
mydf<-data.frame(v1,v2,v3,v4) #make dataframe called mydf
```

HOOD COLLEGE

- We often import existing data into a dataframe (see importing data below)
- We could construct a data frame from scratch using **data.frame()**

```
v1<-c(10,20,30,45,60) # A list of integers
v2<-LETTERS[1:5] # The first 5 letters
v3<-round(rnorm(5),2) #5 random draws from normal distr, rounded to 2 decimal pl
v4<-c("Apples","Oranges","Bananas","Kiwi","Watermelon") #Fruits
mydf<-data.frame(v1,v2,v3,v4) #make dataframe called mydf
```

HOOD COLLEGE

- We often import existing data into a dataframe (see importing data below)
- We could construct a data frame from scratch using **data.frame()**
- Suppose we have 4 different vectors, **v1**, **v2**, **v3**, and **v4**

```
v1<-c(10,20,30,45,60) # A list of integers
v2<-LETTERS[1:5] # The first 5 letters
v3<-round(rnorm(5),2) #5 random draws from normal distr, rounded to 2 decimal pl
v4<-c("Apples","Oranges","Bananas","Kiwi","Watermelon") #Fruits
mydf<-data.frame(v1,v2,v3,v4) #make dataframe called mydf
```

HOOD COLLEGE

- Check the structure of a data frame with `str()`

```
str(mydf) #examine structure
```

```
## 'data.frame':    5 obs. of  4 variables:
##  $ v1: num  10 20 30 45 60
##  $ v2: Factor w/ 5 levels "A","B","C","D",..: 1 2 3 4 5
##  $ v3: num  0.58 0.19 0.1 -0.14 0.81
##  $ v4: Factor w/ 5 levels "Apples","Bananas",..: 1 4 2 3 5
```

```
class(mydf) #check it's a dataframe
```

```
## [1] "data.frame"
```

HOOD COLLEGE

- Note instead of making the vectors first and then combining them, we could have done it all at once:

```
mydf<-data.frame(v1=c(10,20,30,45,60),
                 v2=LETTERS[1:5],
                 v3=round(rnorm(5),2),
                 v4=c("Apples","Oranges","Bananas","Kiwi","Watermelon"))
```

HOOD COLLEGE

- We will use the `gapminder` dataset as a quick example

```
library("gapminder")
str(gapminder)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1704 obs. of  6 variables:
##  $ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3
##  $ year     : int  1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
##  $ lifeExp  : num  28.8 30.3 32 34 36.1 ...
##  $ pop      : int  8425333 9240934 10267083 11537966 13079460 14880372 12881818
##  $ gdpPercap: num  779 821 853 836 740 ...
```

- We will use the `gapminder` dataset as a quick example
- `str()` will give us a sense of the structure

```
library("gapminder")
str(gapminder)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1704 obs. of  6 variables:
##  $ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3
##  $ year     : int  1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
##  $ lifeExp  : num  28.8 30.3 32 34 36.1 ...
##  $ pop      : int  8425333 9240934 10267083 11537966 13079460 14880372 12881818
##  $ gdpPercap: num  779 821 853 836 740 ...
```

- **head()** will show us the top 6 rows (observations)

```
head(gapminder)
```

```
## # A tibble: 6 x 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

HOOD COLLEGE

- summary() will give us a summary statistics of each variable (columns)

```
summary(gapminder)
```

```
##         country        continent        year         lifeExp
##  Afghanistan:  12   Africa  :624   Min.   :1952   Min.   :23.60
##  Albania    :  12   Americas:300   1st Qu.:1966   1st Qu.:48.20
##  Algeria    :  12   Asia    :396   Median :1980   Median :60.71
##  Angola     :  12   Europe  :360   Mean   :1980   Mean   :59.47
##  Argentina  :  12   Oceania : 24   3rd Qu.:1993   3rd Qu.:70.85
##  Australia  :  12                  Max.   :2007   Max.   :82.60
##  (Other)    :1632
##       pop              gdpPercap
##  Min.   :6.001e+04   Min.   :   241.2
```

Data Frames: Selecting Columns (Variables)

- Each variable is stored as a part of a data frame that can be called with the $ sign

Data Frames: Selecting Columns (Variables)

- Each variable is stored as a part of a data frame that can be called with the $ sign
  - e.g. with the Diamonds data, price can be called with `Diamonds$price`:

```
summary(gapminder$gdpPercap)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
##     241.2    1202.1    3531.8    7215.3    9325.5 113523.1
```

```
mean(gapminder$gdpPercap)
```

```
## [1] 7215.327
```

HOOD COLLEGE

Quick Data Analysis Example

- Simple commands can help us learn about a dataset quickly

Function	Result
`min(distr)`	Find minimum value
`max(distr)`	Find maximum value
`range(distr)`	Find the range
`sort(distr)`	Sort values of distribution from smallest to largest
`sort(distr)[1]`	Find first value when sorted (equvalient to finding min)
`sort(distr, decreasing=TRUE)`	Sort from largest to smallest
`median(distr)`	Find the median
`mean(distr)`	Find the mean
`var(distr)`	Find the variance
`sd(distr)`	Find the standard deviation

Function	Result
`table(distr)`	Gives frequency table of categorical variable values
`fivenum(distr)`	Five number summary (min, q1, median, q3, max)
`summary(distr)`	Gives min, q1, median, mean, q3, max
`quantile(distr, 0.32)`	Find specific (e.g. 32nd) percentile
`summary(factor(distr))`	Lists all unique values in distr
`sum(distr)`	Takes the sum of all values in distr

HOOD COLLEGE

```
summary(gapminder$gdpPercap)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##    241.2   1202.1   3531.8   7215.3   9325.5 113523.1
```

```
mean(gapminder$pop)
```

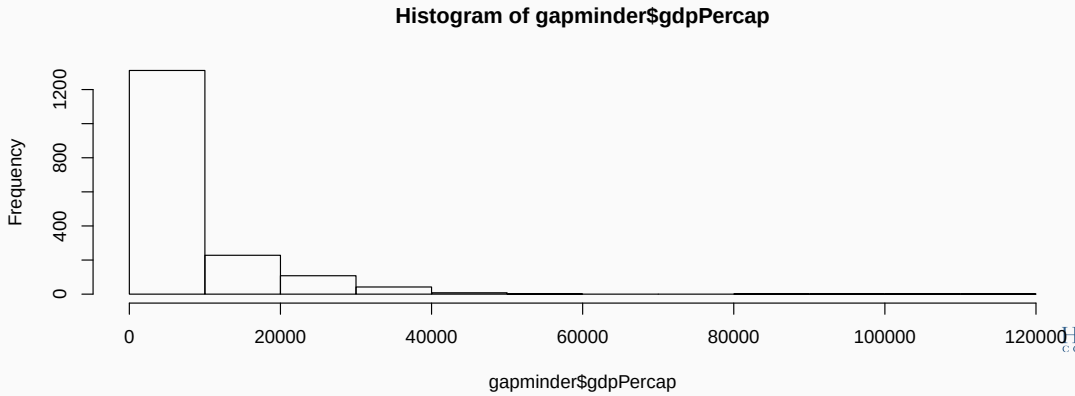
```
## [1] 29601212
```

```
table(gapminder$continent)
```

```
##
##   Africa Americas     Asia   Europe  Oceania
##      624      300      396      360       24
```

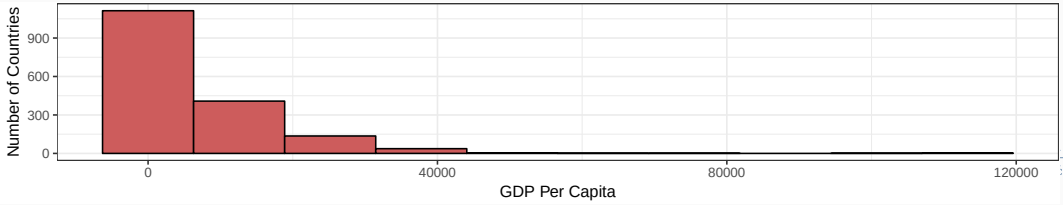
- Base R is very powerful and intuitive to plot, but not very sexy

```
hist(gapminder$gdpPercap)
```



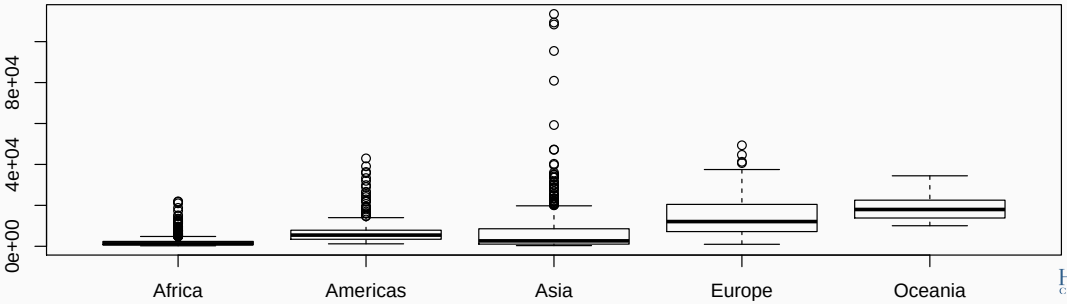
- This is where packages (like **ggplot2**) come in, but we'll have to learn later

```
library("ggplot2")
ggplot(gapminder, aes(x=gdpPercap))+
  geom_histogram(bins=10, color="black",fill="indianred")+
  xlab("GDP Per Capita")+ylab("Number of Countries")+
  theme_bw()
```



- Boxplots

```
boxplot(gdpPercap~continent,data=gapminder)
```



```
library("ggplot2")
library("gapminder")
ggplot(gapminder, aes(x=continent,y=gdpPercap ,fill=continent))+
    geom_boxplot()+ theme_bw()
```