CIS 8392
Topics in Big Data Analytics

#Intro to R

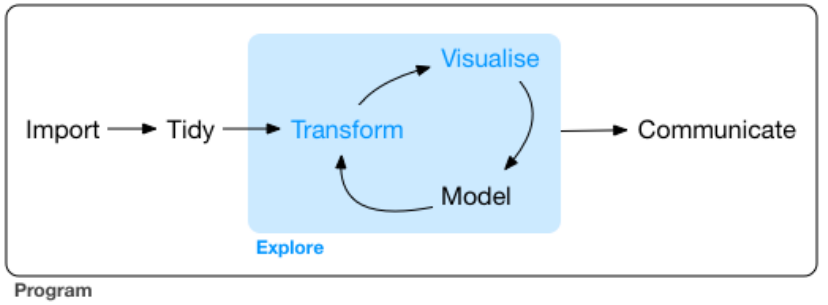
Yu-Kai Lin

About me (Yu-Kai Lin)

- Who am I?
- Where did I come from?
- What are my skills and expertise?
- What is my teaching philosophy?

About this course

- All about **DOING** data science, with an end-to-end workflow in big data analytics
  - This Is America's Hottest Job (Bloomberg, 5/18/18)



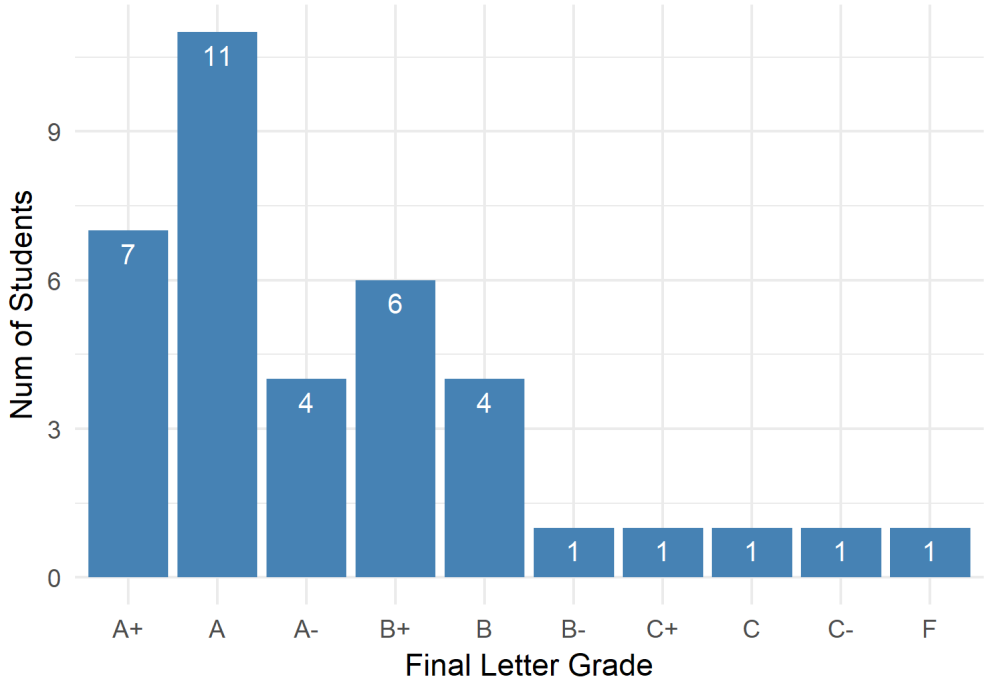
- It's **fast pacing** so be prepared to spend time on reading and practicing, even outside the classroom

- We emphasize **learning by doing**: in-class exercises, assignments, and project

Syllabus

- Course Information
- Course Mechanics
  - Typically 3 segments in each session/meeting
- Course Objectives
- Recommended Textbooks (4)
- Technology and Software Requirements (R + RStudio)
- Course Outline
- Student Evaluation (*review the grading expectations carefully*)
  - Assignments: 60%
  - Course Project: 30%
  - Participation: 10%

Final grades from previous semester



The B's and C's are due to only one reason: **late submissions**.

Agenda

In this segment, we will quickly go through the following basic topics:

1. Your own virtual machine for this course
2. Set up R and RStudio
3. Understand the layout and functionality of RStudio
4. Experiment with basic R expressions and data types

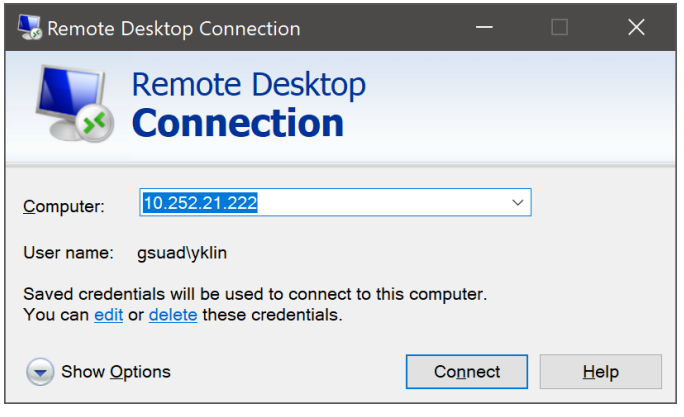
Most of you are likely to have some experience in these. I will quickly go through them to refresh your memory, and lay the groundwork for more advanced topics.

[**Acknowledgements**] The materials in the following slides are based on the source(s) below:

- R for Data Science by Garrett Grolemund and Hadley Wickham

Virtual machine (VM)

- A VM has been assigned to you. It has all the software, R packages, and large datasets pre-installed.
- The VM will be your primary environment in learning the topics and techniques taught in this course.
- If you are off campus, you need to connect to GSU VPN before you can log into your VM.
- Use **Remote Desktop Connection** to log into your VM. Windows has Remote Desktop Connection pre-installed. For Mac users, please download "Microsoft Remote Desktop" from the App Store.



Important notes about the VM

1. You should never shutdown or restart the VM. Just close the Remote Desktop Connection whenever you are done with the VM.

2. There is a limited storage space on the VM. You should not install any software (e.g., Microsoft Office) or add any data that are unrelated to this course to the VM.

Download and install R

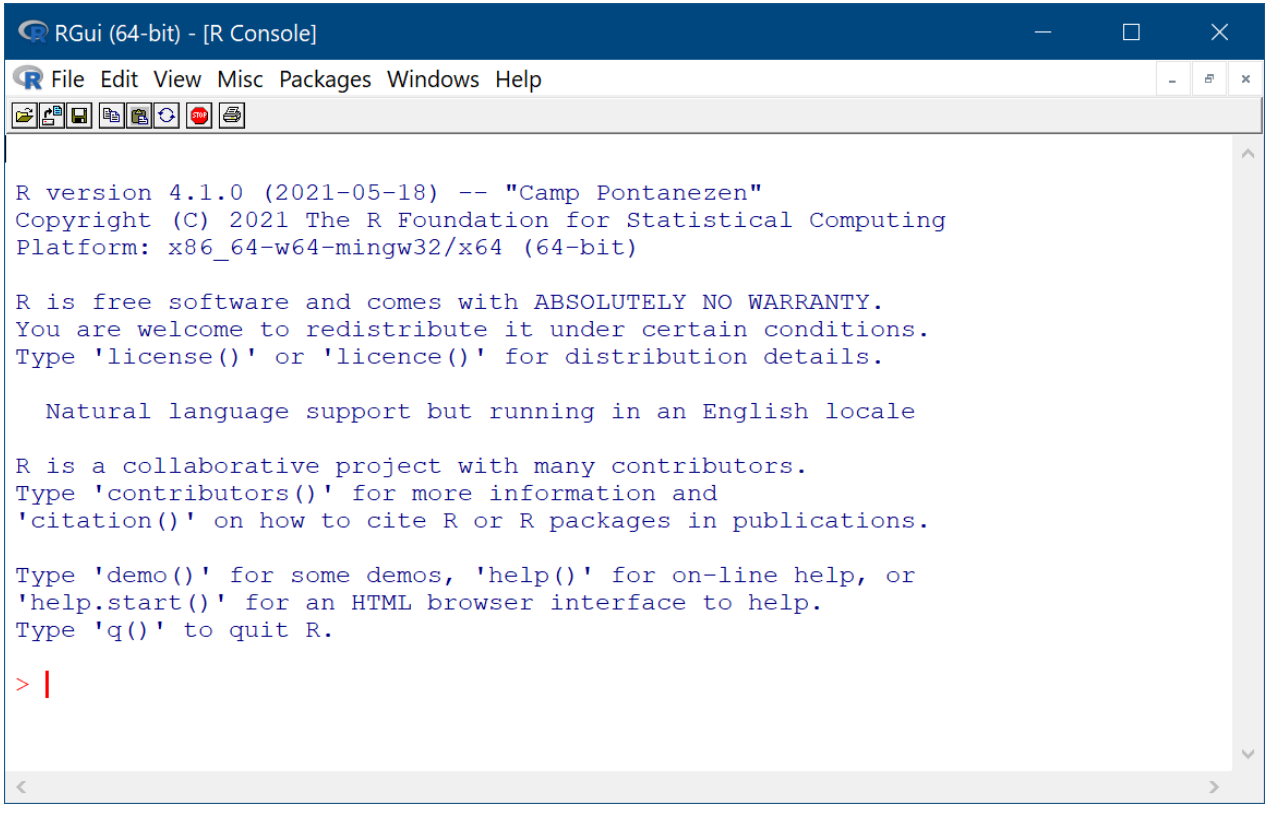
1. Visit CRAN: http://cran.r-project.org
    - CRAN = Comprehensive R Archive Network
2. Click a link on the right to download R for your system (Linux, Mac or Windows)
3. Install R (it is safe to accept the default setting and keep clicking "Next")

Step by step installation guides from YouTube:

- Mac: https://www.youtube.com/watch?v=uxuuWXU-7UQ
- Windows: https://www.youtube.com/watch?v=Ohnk9hcxf9M

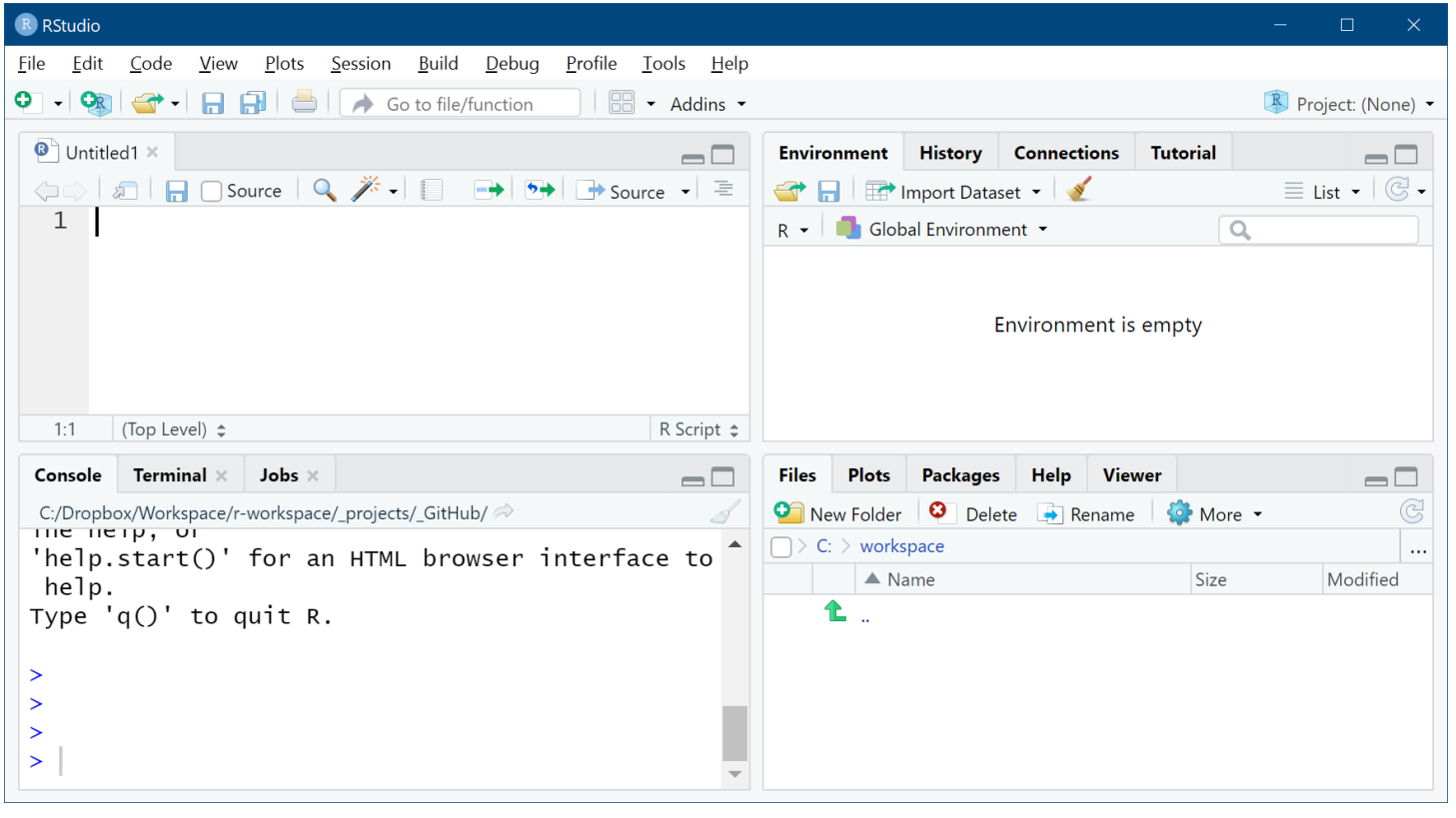
RGui

RGui is an interactive R environment that comes with R installation, but it is very basic and not so user-friendly.



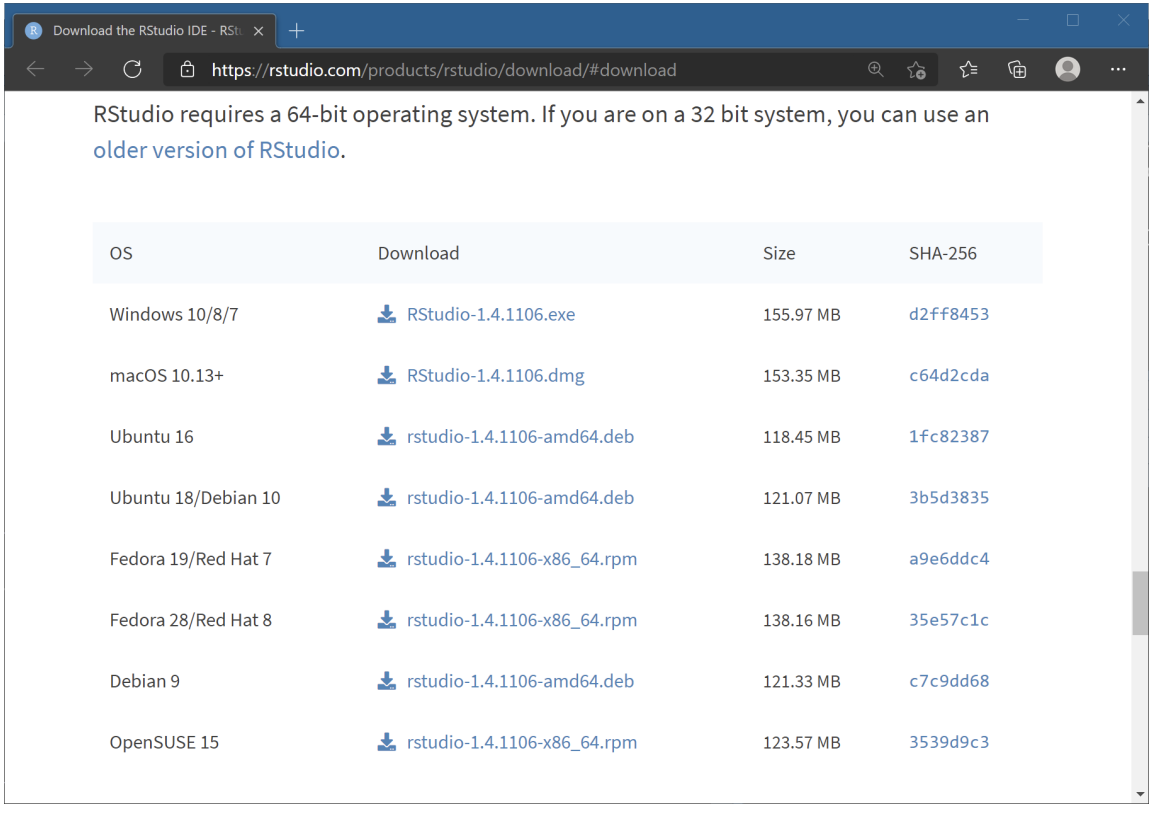
RStudio

RStudio is a development environment for R, and provides many advanced features to improve efficiency and ease of use for R users.

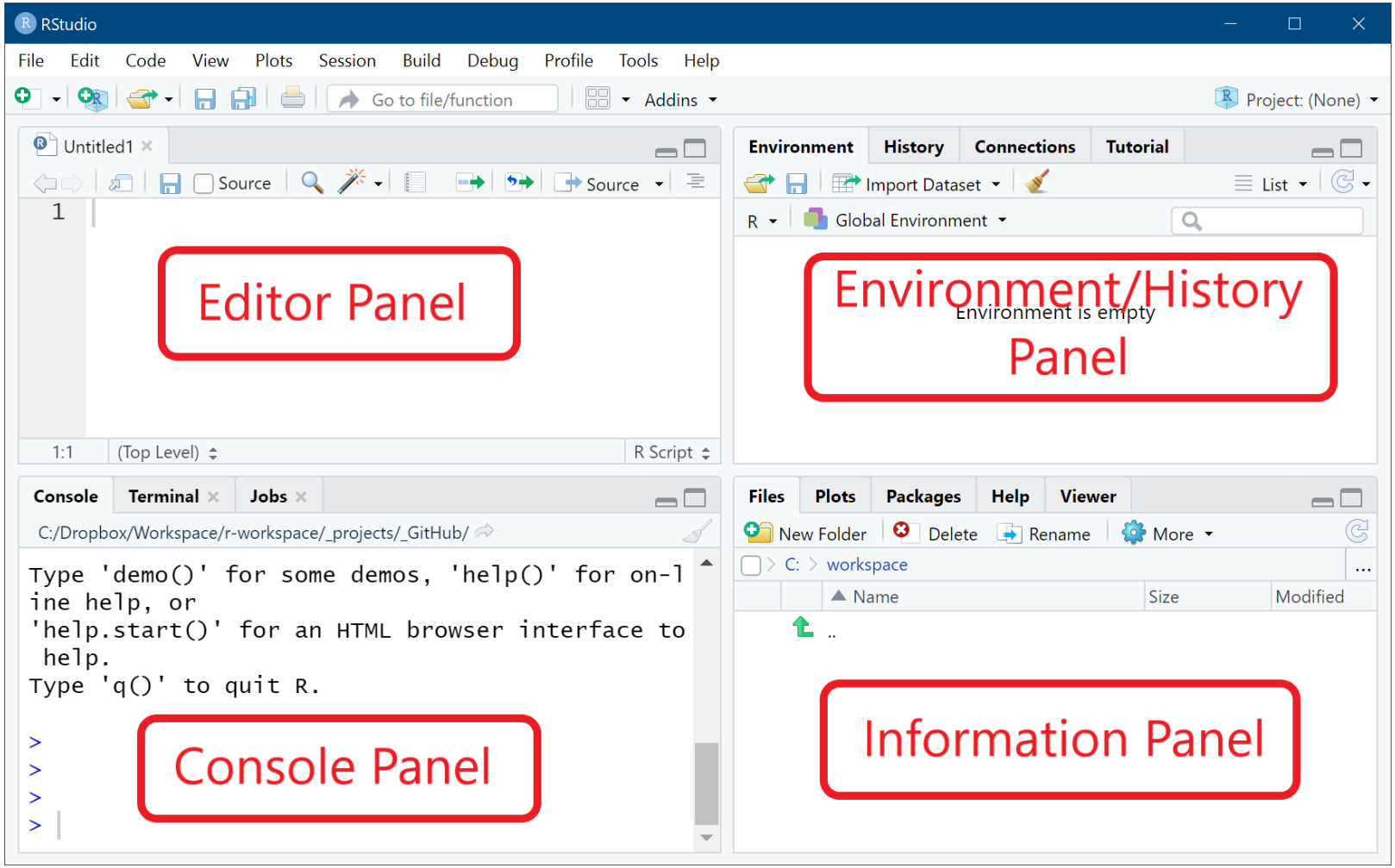


Downloading and installing RStudio

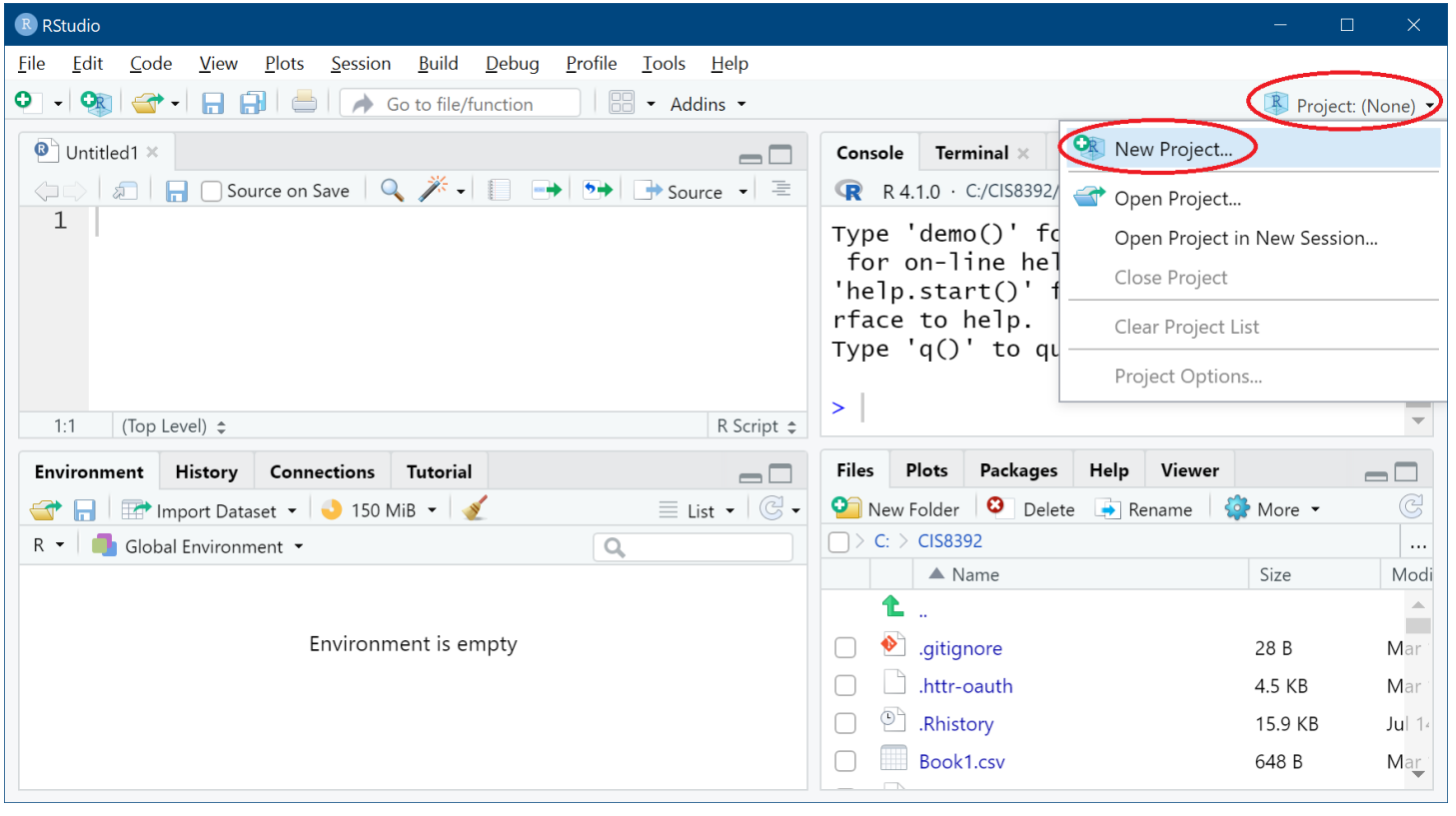
1. Visit https://rstudio.com/products/rstudio/download/#download
2. Under *All Installers*, choose one that fits your system (Linux, Mac or Windows)



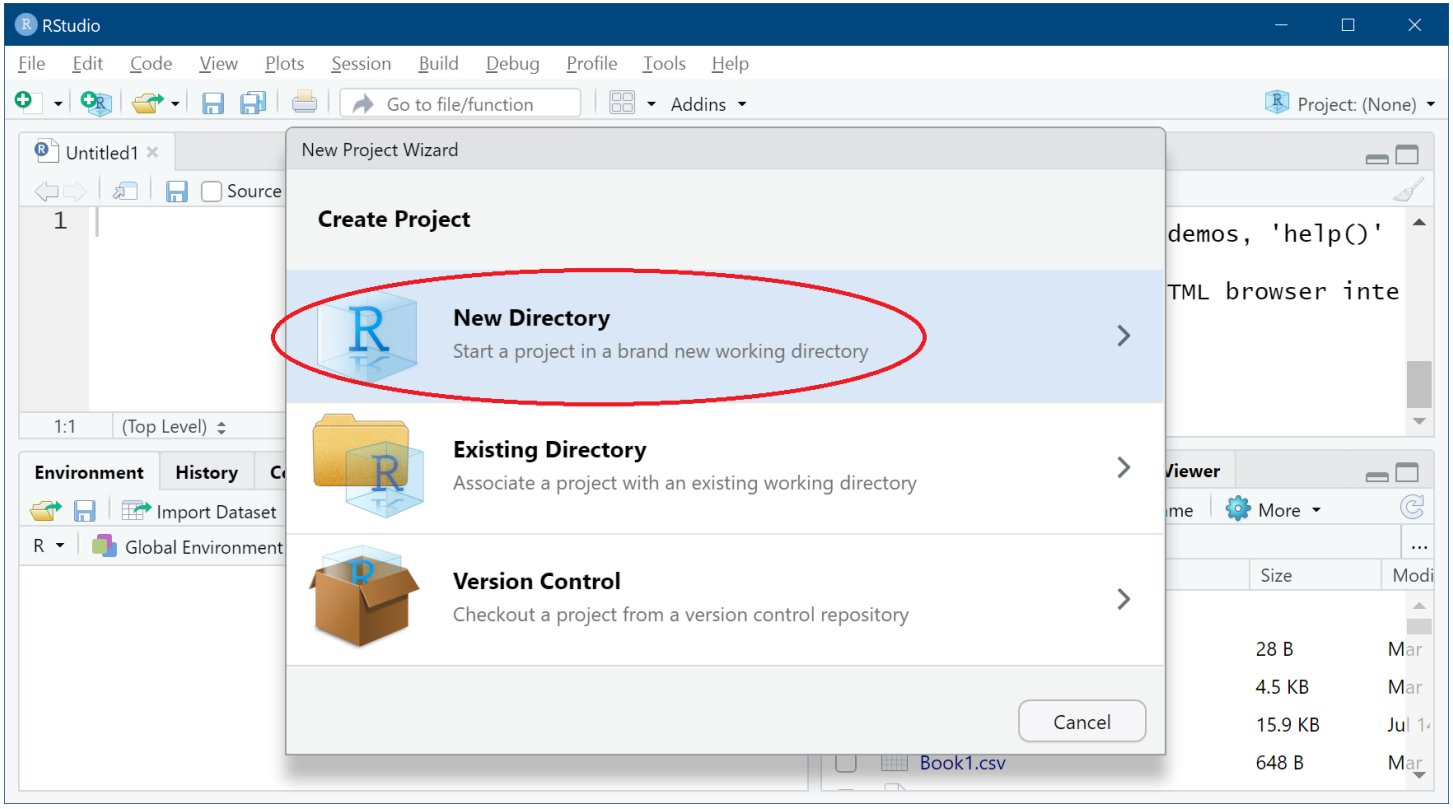
Getting started with RStudio



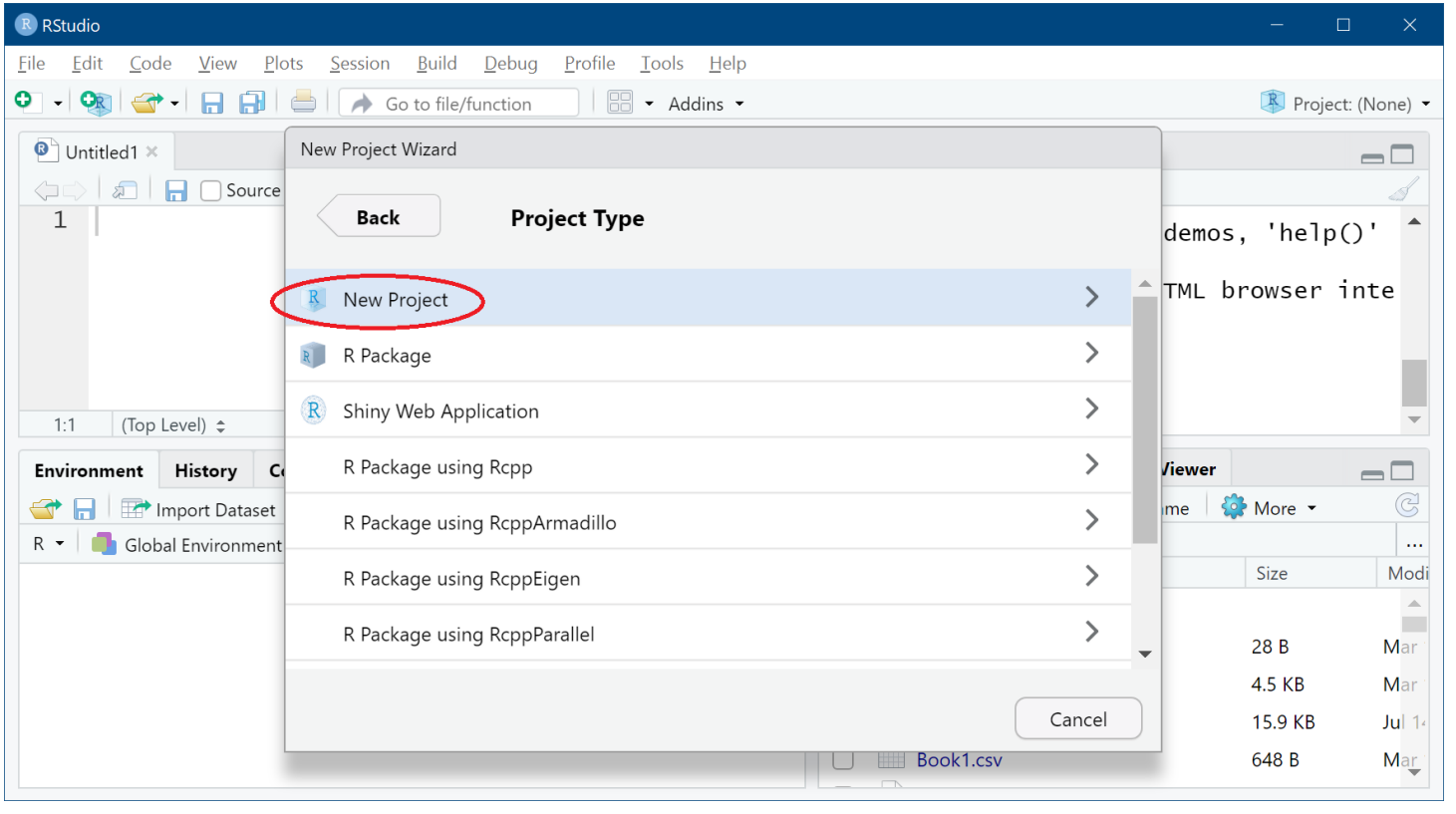
RStudio: set up a project for this course



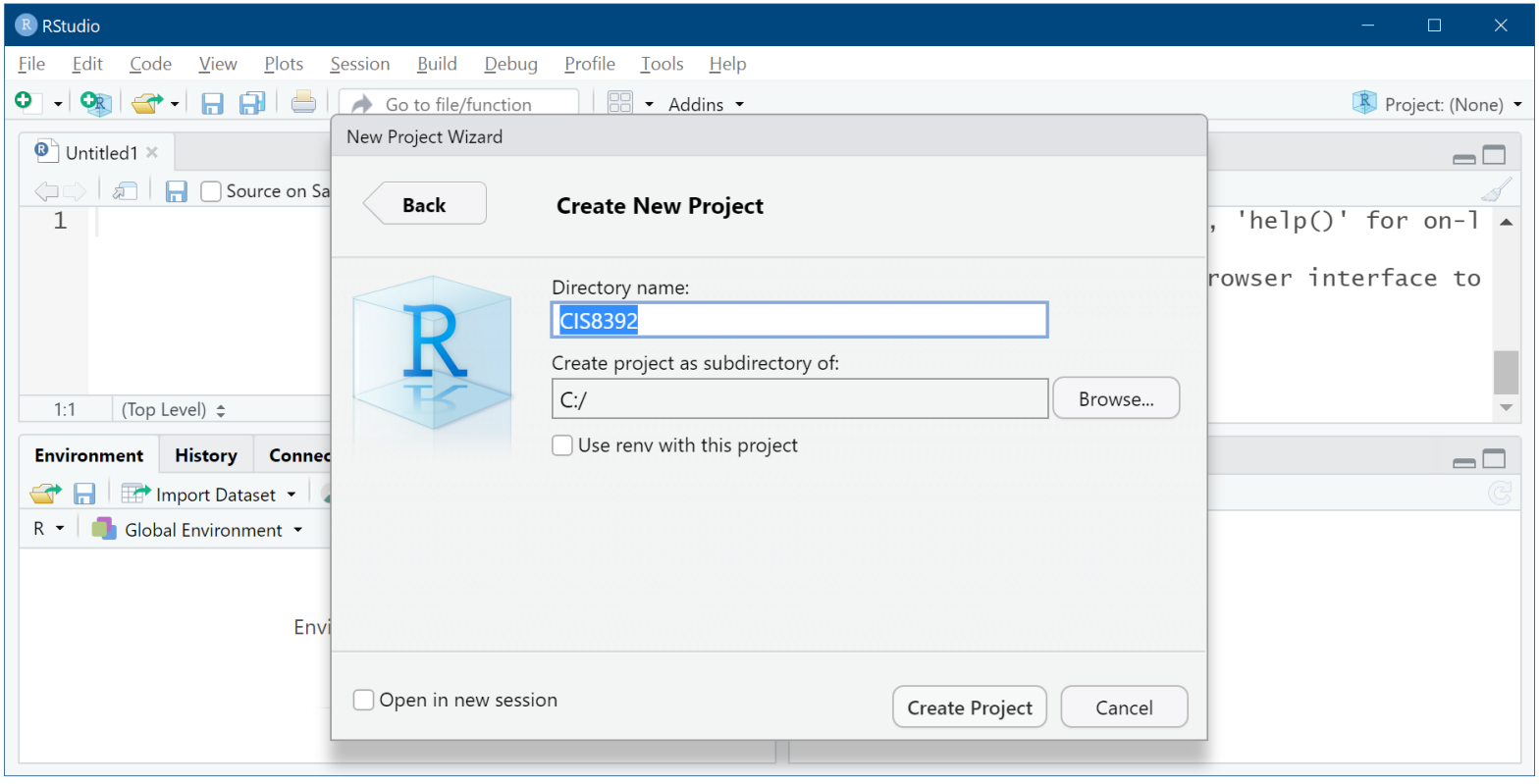
Create project in a new directory:



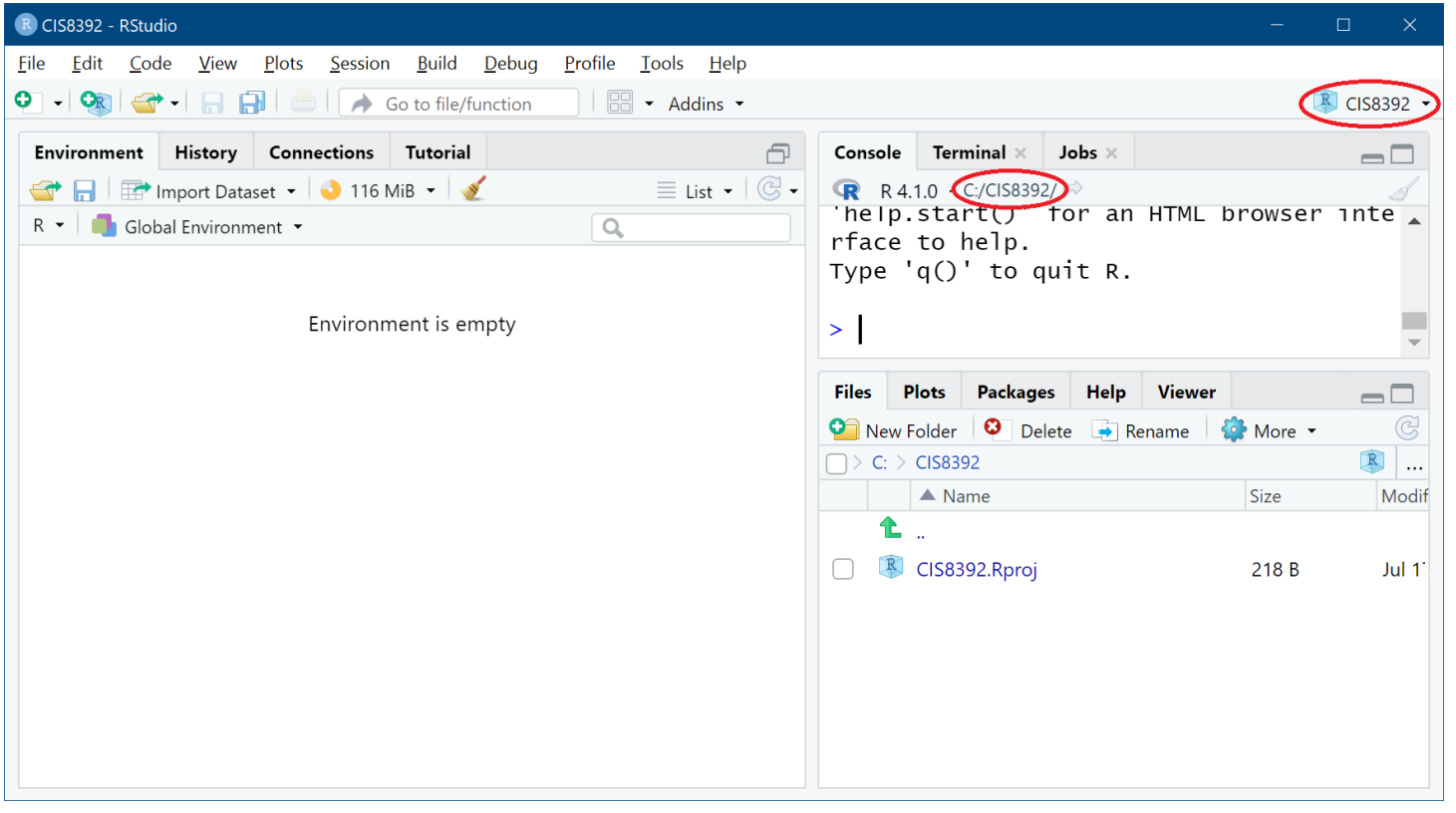
Choose "New Project"



1. Directory name: **CIS8392**
2. Create project as sub-directory of: Anywhere you like

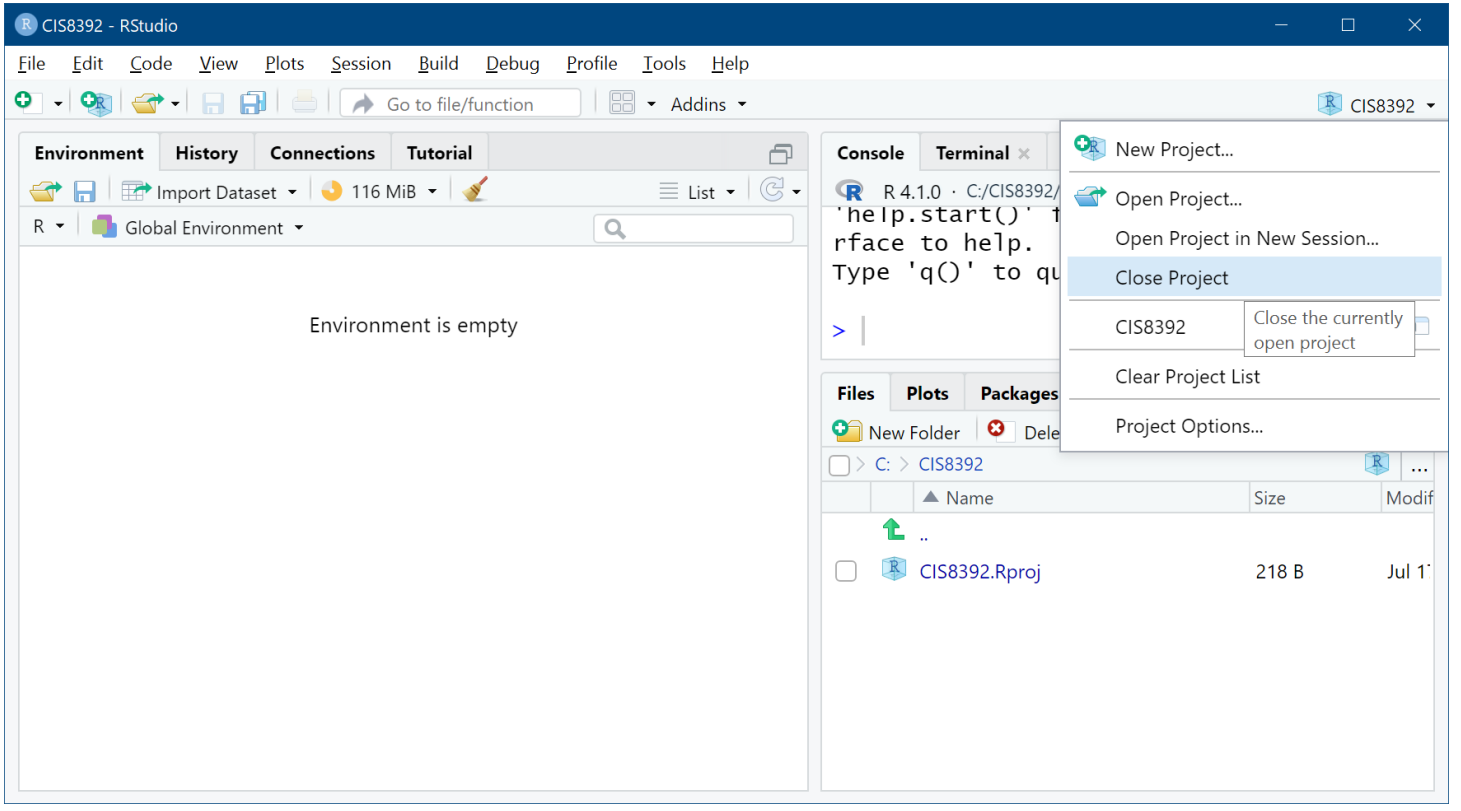


Once created, you can see from RStudio that you are using the **CIS8392** project and the location of the project directory.



You could close the project if you want, **but don't close it now!**

In our lectures, I always assume that you are using this course-specific project in RStudio.



Getting Help

There will be many occasions where you want to learn more about a built-in command or function. Type `help(function_name)` or `?function_name` to get more information. For example:

```
help(factorial)
?factorial
```

Use two question marks to search the whole help database, especially when you don't know exactly the function name. For example,

```
??read
```

Basic data types in R

- Numeric
- Character strings (text)
- Logical
- Factor

Numeric

Any number, no quotes.

Appropriate for math.

```
1 + 1
200000
sqrt(9)
```

```
class(0.3)  # "class" is function that shows the data type of an input
```

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

Character

Any symbols surrounded by single quotes (') or double quotes (")

```
class("Big Data Analytics")
```

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

```
nchar('Big Data Analytics')
```

```
## [1] 18
```

```
toupper("Big Data Analytics")
```

```
## [1] "BIG DATA ANALYTICS"
```

```
paste("Big", "Data", "Analytics", sep="_")
```

```
## [1] "Big_Data_Analytics"
```

Logical

Logical values are either TRUE or FALSE (Note: they are uppercase).

```
2 + 3 == 5   # use '==' to check whether two values are equal
```

```
## [1] TRUE
```

```
3 < 2
```

```
## [1] FALSE
```

```
TRUE == T    # use T as a short hand for TRUE; F for FALSE
```

```
## [1] TRUE
```

Factor

R's form of categorical data. Saved as an integer with a set of labels (e.g. levels)

```
states <- factor(c("FL", "GA", "AZ"))
states
```

```
## [1] FL GA AZ
## Levels: AZ FL GA
```

```
class(states)
```

```
## [1] "factor"
```

```
levels(states)
```

```
## [1] "AZ" "FL" "GA"
```

Detect / convert data types

`is.`*XYZ*`()` functions return Boolean for whether the argument is of type *XYZ*

`as.`*XYZ*`()` (tries to) "cast" its argument to type *XYZ* --- to translate it sensibly into a *XYZ*-type value

```
is.numeric(7)
```

```
## [1] TRUE
```

```
is.character(7)
```

```
## [1] FALSE
```

```
is.character("7")
```

```
## [1] TRUE
```

```
as.character(5/2)
```

```
## [1] "2.5"
```

```
as.numeric(as.character(5/2))
```

```
## [1] 2.5
```

```
2 * as.numeric(as.character(5/2))
```

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

Create variables

We can give names to data objects; these give us **variables**

Variables are created with the **assignment operator**, `<-` or `=`

Be careful that R is a case sensitive language. FOO, Foo, and foo are different!

```
x = 2      # use the equal sign to assign value
y <- 3     # you can also use an arrow to assign value
x          # print the value of a variable by typing its name
```

```
## [1] 2
```

```
x * y
```

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

```
x <- 8
x
```

```
## [1] 8
```

Variable names

Variable names cannot begin with numbers. Wise to avoid special characters, except for period (`.`) and underline (`_`)

Example of valid names:

- a
- b
- FOO
- my_var
- .day

Example of invalid names:

- 1
- 2nd
- ^mean
- !bad
- $

Improve readability of your code

- A command can spread across multiple lines. This can often improve readability.

```
x = paste("How", "are", "you?",   # what happens if you only enter this line
          sep=" ")
x
```

```
## [1] "How are you?"
```

- We can put multiple commands in the same line, but they need to be separated by a semicolon (;)

```
a = 1; b = 2
a + b
```

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

Your turn

1. Create variables `f_name` and `l_name` with values equal to your own first/last names
2. Get the number of characters in `f_name` and `l_name` and save them to `length_f_name` and `length_l_name` respectively
3. Use the `paste()` function to get your whole name
4. `length_f_name` multiplied by `length_l_name`
5. `length_f_name` divided by `length_l_name`
6. Show if `length_f_name` is greater than `length_l_name`

Advanced data types in R

- *Vector*: a set of values, all of the same data type
- *List*: a set of values, potentially with different data types
- *Matrix*: special 2D numerical structure
- *Array*: multi-dimantional structure of the same data type
- *Data frame*: like an Excel sheet or a database table

Vector

A **vector** is a sequence of values, all of the same type

```
x <- c(1, 3, 7, 15)   # c stands for "combine"
x
```

```
## [1]  1  3  7 15
```

```
is.vector(x)
```

```
## [1] TRUE
```

```
length(x)             # find the number of elements in a vector
```

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

```r
seq(from=1, to=10)          # sequence
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
1:10                        # sequence shorthand
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
seq(from=1, to=10, by=2)    # sequence
```

```
## [1] 1 3 5 7 9
```

```r
rep(7, times=3)             # repeat
```

```
## [1] 7 7 7
```

Name a vector

```
vec <- c(10, 20, 7, 13) # assigning a vector to a variable
vec
```

```
## [1] 10 20  7 13
```

```
names(vec) <- c("value1", "value2", "value3", "value4")
vec
```

```
## value1 value2 value3 value4
##     10     20      7     13
```

```
vec <- c("value1"=10, "value2"=20, "value3"=7, "value4"=13) #same result
vec <- c(value1=10, value2=20, value3=7, value4=13)         #same result
```

Combining vectors

```
vec1 <- c(1, 3, 5)
vec2 <- c(11, 13, 15)
c(vec1, vec2, c(21, 23, 25))
```

```
## [1]  1  3  5 11 13 15 21 23 25
```

Vector arithmetics

Vector computations are performed **element-wise**

```
earnings <- c(10, 20, 30, 40)
expenses <- c(5, 25, 25, 10)
5 * earnings
```

```
## [1]  50 100 150 200
```

```
earnings - expenses
```

```
## [1]  5 -5  5 30
```

```
earnings * c(1, 2, 3, 4)
```

```
## [1]  10  40  90 160
```

Recycling

Recycling repeat elements in shorter vector when combined with longer

```
u <- c(10, 20)
v <- c(1, 2, 3, 4, 5)
u + v  # the shorter vector will be recycled to match the longer vector
```

```
## [1] 11 22 13 24 15
```

Under the hood:

u + v

= c(10, 20) + c(1, 2, 3, 4, 5)

= c(10, 20, **10, 20, 10**) + c(1, 2, 3, 4, 5) *# recycling*

= c(10+1, 20+2, 10+3, 20+4, 10+5) *# element-wise operation*

= c(11, 22, 13, 24, 15)

Test if a vector has a specific value

```
x <- c(10, 20, 30)
```

Does `x` has 20?

```
20 %in% x
```

```
## [1] TRUE
```

Does `x` has 40?

```
40 %in% x
```

```
## [1] FALSE
```

```
basket <- c("apple", "banana")
```

Does `basket` has apple?

```
"apple" %in% basket
```

```
## [1] TRUE
```

Does `basket` has cheese?

```
"cheese" %in% basket
```

```
## [1] FALSE
```

Missing values: NA

In real world, your data may contain missing values. In R, we use `NA` (upper case) to represent a missing value.

```
vec = c(1, 4, NA, 2)
vec
```

```
## [1]  1  4 NA  2
```

```
sum(vec)
```

```
## [1] NA
```

```
max(vec)
```

```
## [1] NA
```

`NA` creates problems for most numerical functions.

For example, we cannot add `NA` to other numbers.

To apply these numerical functions on data with `NA`s, we simply just remove `NA`s from the calculation. That is,

```
sum(vec, na.rm = T)
```

```
## [1] 7
```

```
max(vec, na.rm = T)
```

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

Vector indexing

You can retrieve elements from a vector by specifying the indexes of the elements. This operation is also known as `subsetting`.

```
vec <- c("value1"=10, "value2"=20, "value3"=30, "value4"=40)
vec[1] # get the element at index 1
```

```
## value1
##     10
```

```
vec["value3"] # get the element whose name matches the string
```

```
## value3
##     30
```

You can provide more than just one index.

```
vec[1:3] # specify a vector of indexes
```

```
## value1 value2 value3
##     10     20     30
```

```
vec[c(3, 2, 1, 4)] # return with the specified order
```

```
## value3 value2 value1 value4
##     30     20     10     40
```

```
vec[c("value4", "value4")]
```

```
## value4 value4
##     40     40
```

List

List is also a container, but it can accommodate items of different data types.

```
x <- list("Bob", c(100,80,90))
x #whenever you see [[1]], [[2]], ..., the object is a list
```

```
## [[1]]
## [1] "Bob"
##
## [[2]]
## [1] 100  80  90
```

Just like vectors, you can give each element a name:

```
x <- list(name="Bob", grades=c(100,80,90))
x #whenever you see $xxx, $yyy, ... the object is a list
```

```
## $name
## [1] "Bob"
##
## $grades
## [1] 100  80  90
```

List indexing

```
x[2]          # get the second elment as a list
```

```
## $grades
## [1] 100  80  90
```

```
x["grades"]    # get the elment named "grades" as a list
```

```
## $grades
## [1] 100  80  90
```

```
y1 = x[2]
class(y1)
```

```
## [1] "list"
```

```
x[[2]]         # get the second elment as a vector
```

```
## [1] 100  80  90
```

```
y2 = x[[2]]
class(y2)
```

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

```
x[["grades"]]  # get the elment named "grades" as a vector
```

```
## [1] 100  80  90
```

```
x$grades       # most readable
```

```
## [1] 100  80  90
```

Your turn

```
## $name
## [1] "Alice"  "Bob"    "Claire" "Daniel"
##
## $female
## [1]  TRUE FALSE  TRUE FALSE
##
## $age
## [1] 20 25 30 35
```

```
## [1] "Bob"
```

1. Create the above **list**
2. Get the name "Bob" *from* the list
    - Hint: Get the name vector from the list and then get the second element in the vector

Matrix

A matrix is a collection of data elements arranged in a two-dimensional rectangular layout.

```
A <- matrix(
  1:6,                # the data elements
  nrow=2,             # number of rows
  ncol=3,             # number of columns
  byrow = TRUE)       # fill matrix by rows
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

Array

An array is a collection of values (same type) organized into multiple dimentions.

```
(x <- array(1:24, dim=c(3,4,2)))
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]   13   16   19   22
## [2,]   14   17   20   23
## [3,]   15   18   21   24
```

```
x[2,1,2]
```

```
## [1] 14
```

```
x[2,1,]
```

```
## [1]  2 14
```

```
x[2,,]
```

```
##      [,1] [,2]
## [1,]    2   14
## [2,]    5   17
## [3,]    8   20
## [4,]   11   23
```

Data frame

A data frame is a set of vectors of equal length. Consider data frame as an Excel sheet or a database table.

```
course <- c("CIS8392", "CIS8010", "CIS8050", "CIS1234")
num_of_students <- c(20, 10, 40, 30)
analytics_course <- c(TRUE, TRUE, TRUE, FALSE)
df <- data.frame(course, n_students=num_of_students, analytics_course,
                 stringsAsFactors=F)
df # notice the column names and row names
```

```
##    course n_students analytics_course
## 1 CIS8392         20             TRUE
## 2 CIS8010         10             TRUE
## 3 CIS8050         40             TRUE
## 4 CIS1234         30            FALSE
```

Useful functions for data frames

```
ncol(df) # number of columns
```

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

```
nrow(df) # number of rows
```

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

```
colnames(df) # get column names
```

```
## [1] "course"           "n_students"       "analytics_course"
```

```
rownames(df) # get row names
```

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

Change column and row names

```
df2 <- df # create a copy of df, and name it as "df2"
colnames(df2) <- c("col1", "col2", "col3") # assign column names
colnames(df2)
```

```
## [1] "col1" "col2" "col3"
```

```
rownames(df2) <- c("row1", "row2", "row3", "row4") # assign row names
rownames(df2)
```

```
## [1] "row1" "row2" "row3" "row4"
```

```
df
```

```
##    course n_students analytics_course
## 1 CIS8392         20             TRUE
## 2 CIS8010         10             TRUE
## 3 CIS8050         40             TRUE
## 4 CIS1234         30            FALSE
```

```
df2
```

```
##         col1 col2  col3
## row1 CIS8392   20  TRUE
## row2 CIS8010   10  TRUE
## row3 CIS8050   40  TRUE
## row4 CIS1234   30 FALSE
```

Get values from a column

There are many ways you can get values out of a column.

- The most readable way: `dataframe_name$column_name`

```
df$course
```

```
## [1] "CIS8392" "CIS8010" "CIS8050" "CIS1234"
```

```
df$n_students
```

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

Get values from rows

```
df
```

```
##    course n_students analytics_course
## 1 CIS8392         20             TRUE
## 2 CIS8010         10             TRUE
## 3 CIS8050         40             TRUE
## 4 CIS1234         30            FALSE
```

```
df[2,]  # row 2
```

```
##    course n_students analytics_course
## 2 CIS8010         10             TRUE
```

```
df[c(1,3),]  # rows 1 & 3
```

```
##    course n_students analytics_course
## 1 CIS8392         20             TRUE
## 3 CIS8050         40             TRUE
```

Specify rows and columns

```
df[2,1]             # row 2, column 1
```

```
## [1] "CIS8010"
```

```
df[c(3,4),c(1,2)]    # rows 3 & 4, columns 1 & 2
```

```
##    course n_students
## 3 CIS8050         40
## 4 CIS1234         30
```

```
df["2","n_students"] # "2": row name, "n_students": column name
```

```
## [1] 10
```

Your turn

```
##          name female age
## row_1   Alice   TRUE  20
## row_2     Bob  FALSE  25
## row_3 Claire   TRUE  30
## row_4 Daniel  FALSE  35
```

1. Create the above **data frame** (don't forget the column/row names!)

2. Obtain the mean of the age column from the data frame