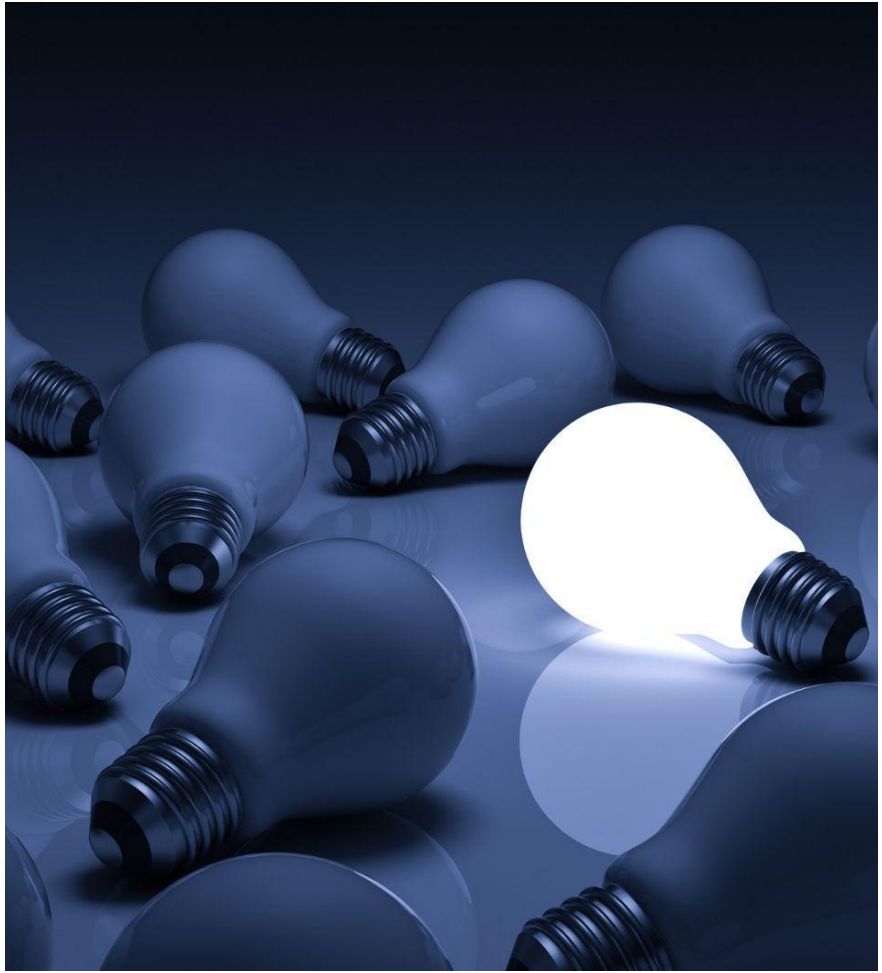
R for Data Cleaning

What is R?
A scripting language

Why use R?
To reproduce &/or share workflows.



Reference: Courtesy of: J. Bayalis. (c) Field Museum of Natural History.
https://mm.fieldmuseum.org/136227d2-e642-4952-b993-3fe7c434cdc2 (accessed on 29 Oct 2021)

Agenda

Introduction to R	01
R-Studio Setup	02
Script Setup	03
Importing and Viewing Data	04
Reshaping Data	05
Exporting Data	06
Extras	07

Introduction to R

What is this thing called “R”?

R is both:

- a scripting language
- software to R-scripts

R-Studio is:

- an IDE — “integrated development environment” (a program that makes coding more convenient)

R & **R-Studio** are useful for:

- making reproducible data-handling workflows
- sharing those workflows (as scripts & documentation)

What Can/Can't R Do?

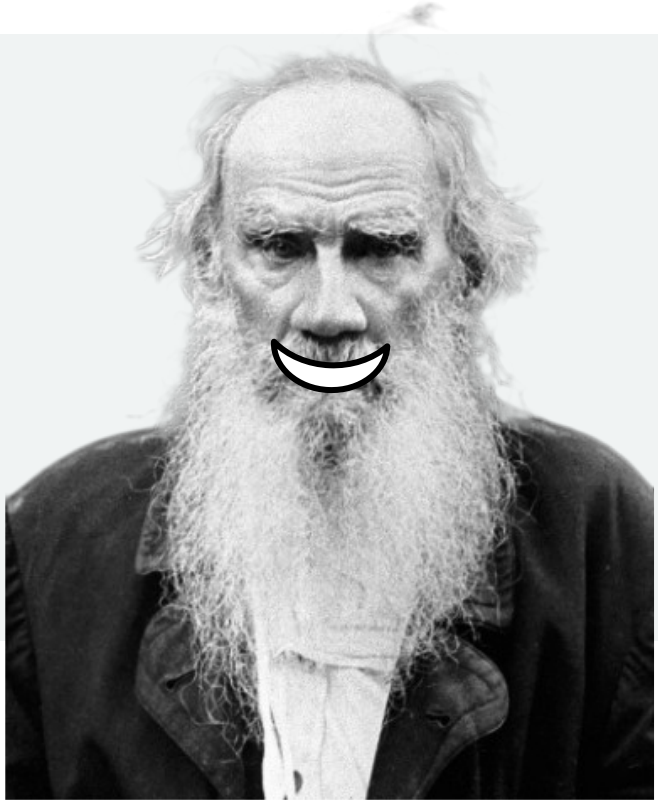
	R
Formulas	☑
Charts & Graphs	☑
Auto-formats fields	Some functions
Allow creation of lookup lists	☑
Allow character sets besides Latin1 (e.g., UTF-8)	☑
Track changes after closing	☑
Maintain integrity between rows and columns	☑
Enforce referential integrity between different sheets	Some assembly required

Useful Philosophy/ies

“All tidy data is alike; all messy data is messy in its own way.” – Hadley Wickham/Leo

Tidy data structure:
(based on Codd 1990/normalization)

- 1 row = 1 observation (record)
- 1 column = 1 variable (attribute)
- 1 value = 1 observation of 1 variable
- 1 table = 1 type of observational unit [1 record-type]
    - e.g.— 1 table for locality records; 1 table taxon records

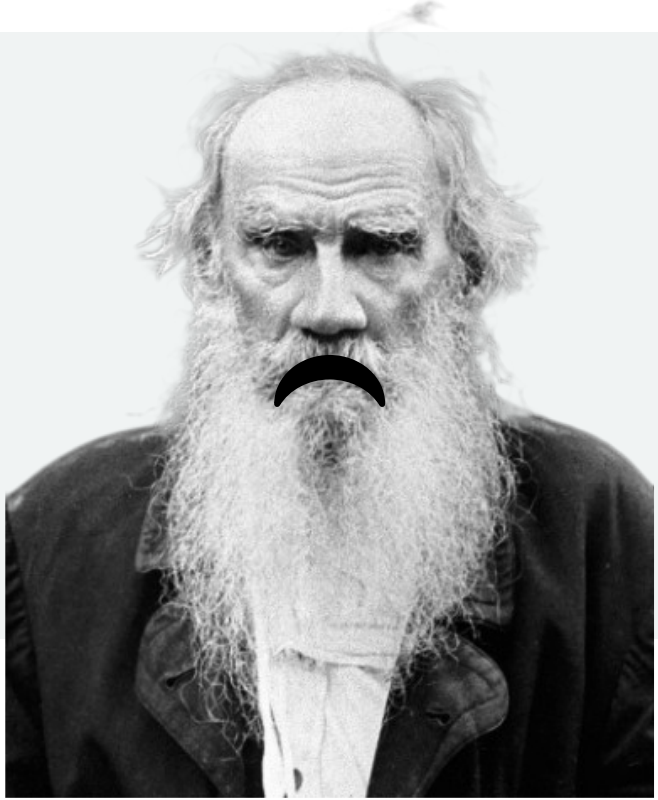


Useful Philosophy/ies

"All tidy data is alike; all messy data is messy in its own way." – Hadley Wickham/Leo

Messy data structure

- 1 row != 1 observation (record)
- 1 column!= 1 variable (attribute)
- 1 value != 1 observation of 1 variable
- 1 table != 1 type of observational unit [1 record-type]
    - e.g.— 1 table for locality records; 1 table taxon records



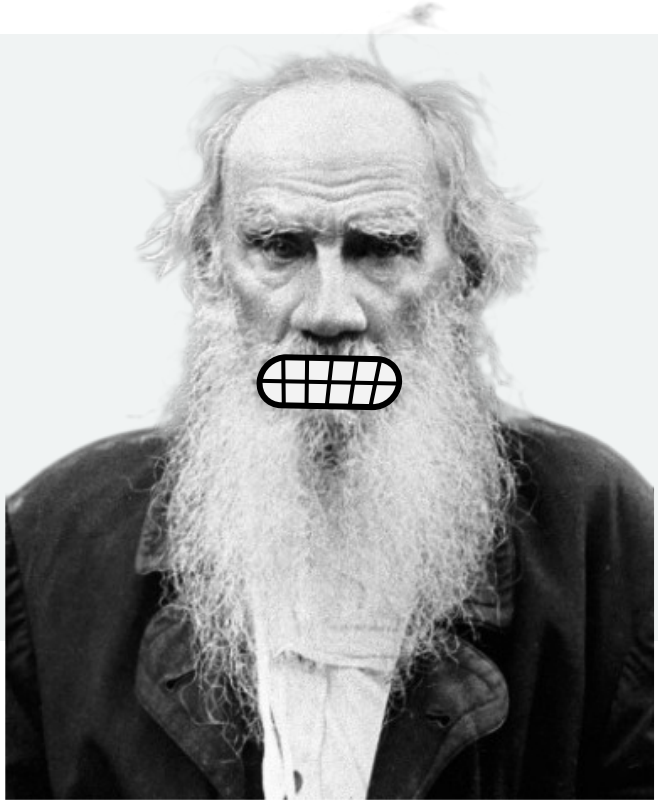
Useful Philosophy/ies

“All tidy data is alike; all messy data is messy in its own way.” – Hadley Wickham/Leo

Messy data structure

- 1 row != 1 observation (record)
- 1 column != 1 variable (attribute)
- 1 value != 1 observation of 1 variable
- 1 table != 1 type of observational unit [1 record-type]
  - e.g.-- 1 table for locality records; 1 table taxon records

But Remember! **Useful data**
“Data cleaning is improving quality of data to make them ‘fit for use’”



Reshaping Data

Gathering and Spreading

The two actions are opposites:

1. Gathering: Adds rows to consolidate columns

1

Cat #	ID-1	ID-2	ID-3
FM 15	P. jagori	P. minor	P. vampyrus
FM 36	R. inops	R. rufus	R. arcuatus

Cat #	ID-#	ID
FM 15	ID-1	P. jagori
FM 15	ID-2	P. minor
FM 15	ID-3	P. vampyrus
FM 36	ID-1	R. inops
FM 36	ID-2	R. rufus
FM 36	ID-3	R. arcuatus

Reshaping Data

Gathering and Spreading

The two actions are opposites:

1. Gathering: Adds rows to consolidate columns
2. Spreading: Adds columns to consolidate rows

1

Cat #	ID-1	ID-2	ID-3
FM 15	P. jagori	P. minor	P. vampyrus
FM 36	R. inops	R. rufus	R. arcuatus

Cat #	ID-#	ID
FM 15	ID-1	P. jagori
FM 15	ID-2	P. minor
FM 15	ID-3	P. vampyrus
FM 36	ID-1	R. inops
FM 36	ID-2	R. rufus
FM 36	ID-3	R. arcuatus

Obj	Meas.	Value
jar	height	36
jar	width	21
jar	depth	5
vase	height	20
vase	width	6
vase	depth	4

Obj	height	width	depth
jar	36	21	5
vase	20	6	4

2

Reshaping Data

Split-Apply-Combine is a philosophy/phrase commonly referenced with R users

- Break down code and datasets into only the pieces relevant for each step
- Similar to “faceting”, “querying” … other database platforms have other fancy names for it (map reduce?)

Gathering & spreading are similar to what you can do with Pivot Tables (in Excel/similar spreadsheet-software)

– **Except** Pivot Tables tend to be unstable (especially with larger/more complex/memory-intensive datasets)

\+ R stays stable

R-Studio Setup

R-Studio – Some Help…

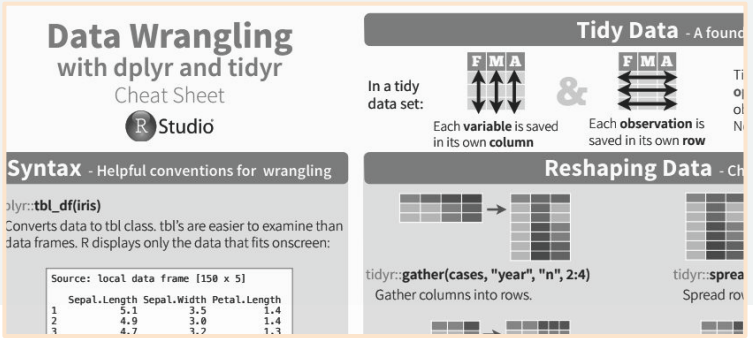
You can trust **CRAN** — the “Comprehensive R Archive Network.”

Installation

- *1st* – Install R
  - cran.r-project.org
- *2nd* – Install R-Studio
  - rstudio.com/products/rstudio/download

Cheat Sheet:

- Overview of what a package can do.
  - rstudio.com/resources/cheatsheets

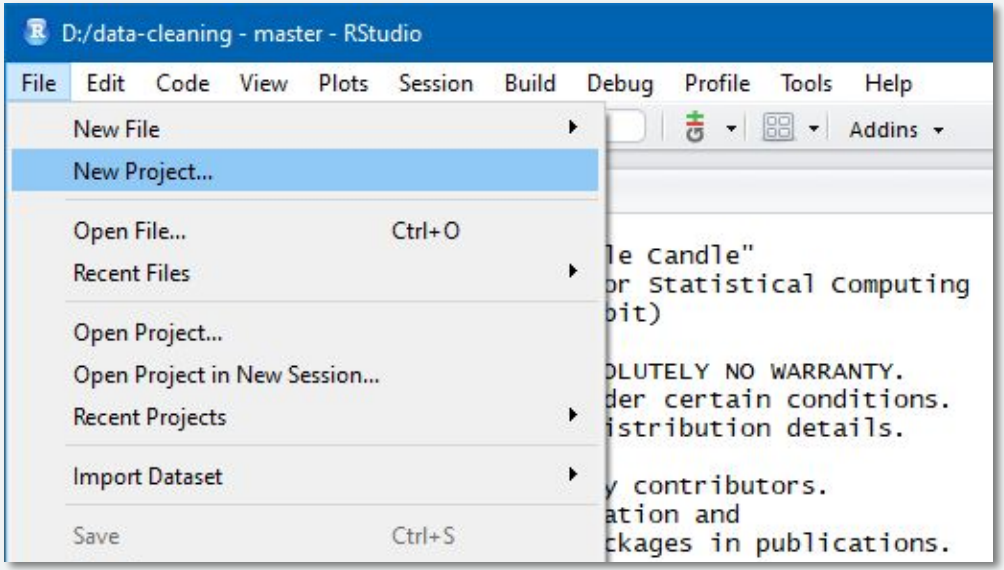


R-Studio Setup

0) Double click the R-Studio icon to start the program.

Make a new project:

1) Go to File → New Project



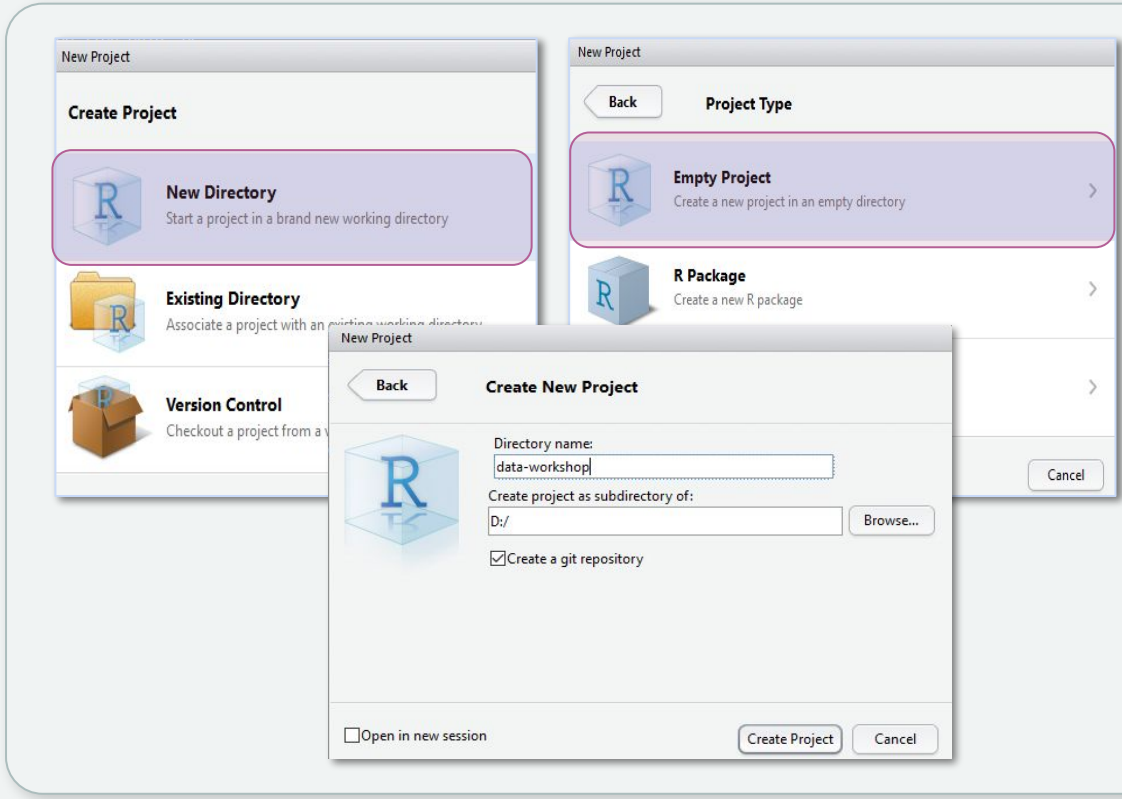
R-Studio Setup

2) Choose “New directory”
   then choose “Empty project”

3) Name the new project directory **`Test`**, and save it to your **`Desktop`**.

After today, never save on Desktop!

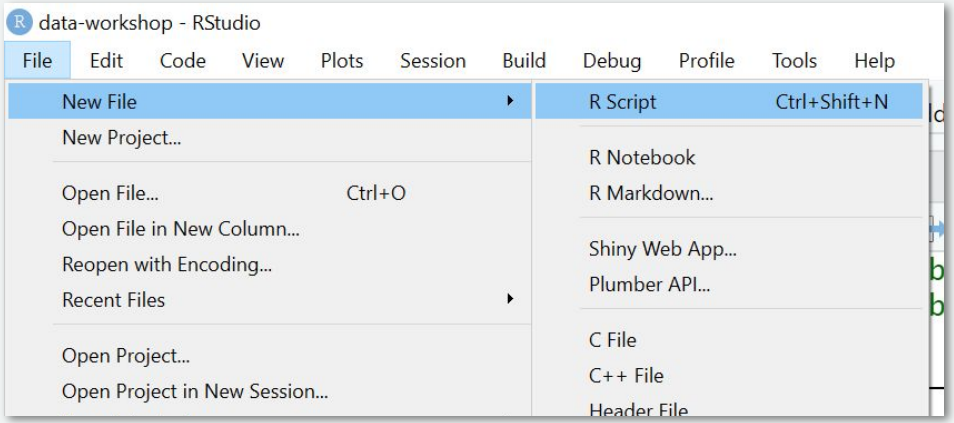
4) Click “Create Project”



Opening a Script

5) In RStudio,

   Go to File → New File → R-Script

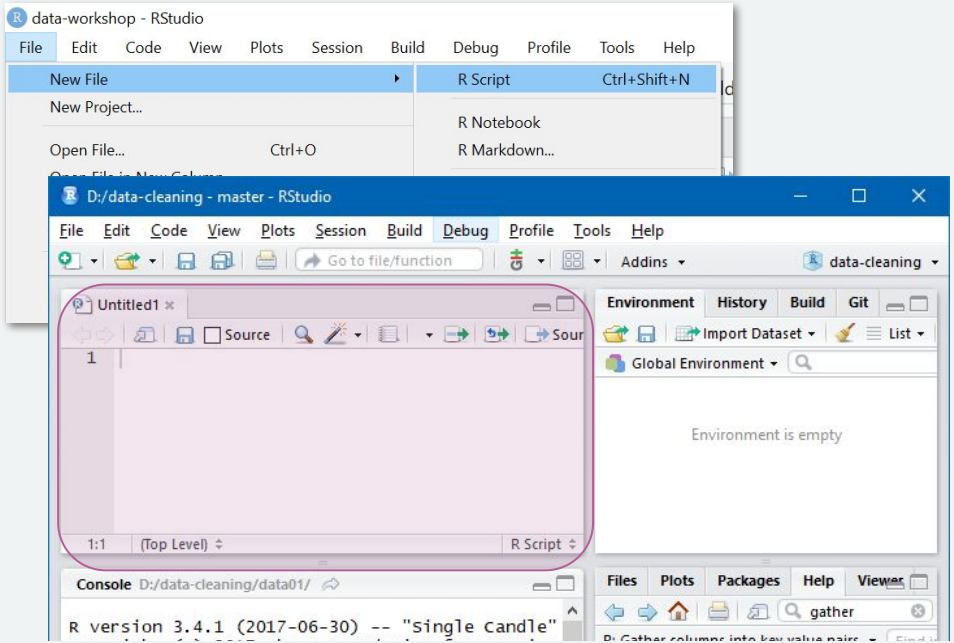


Opening a Script

5) In RStudio,

Go to File → New File → R-Script

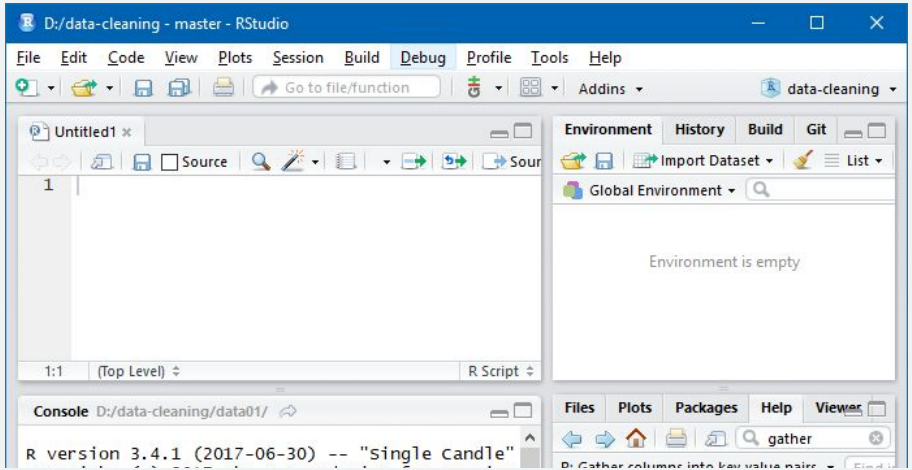
The new R script will open in the Source Editor pane (upper left):



Opening a Script

My eyes are on fire! is there a way to change Script Editor background from white to a darker color?

Go to **Tools** -> **Global Options** -> **Appearance** -> choose a theme.
("**Merbivore Soft**" is easy on the eyes)



R-Studio Interface

Source Editor pane

1. = edit/save scripts

Console pane

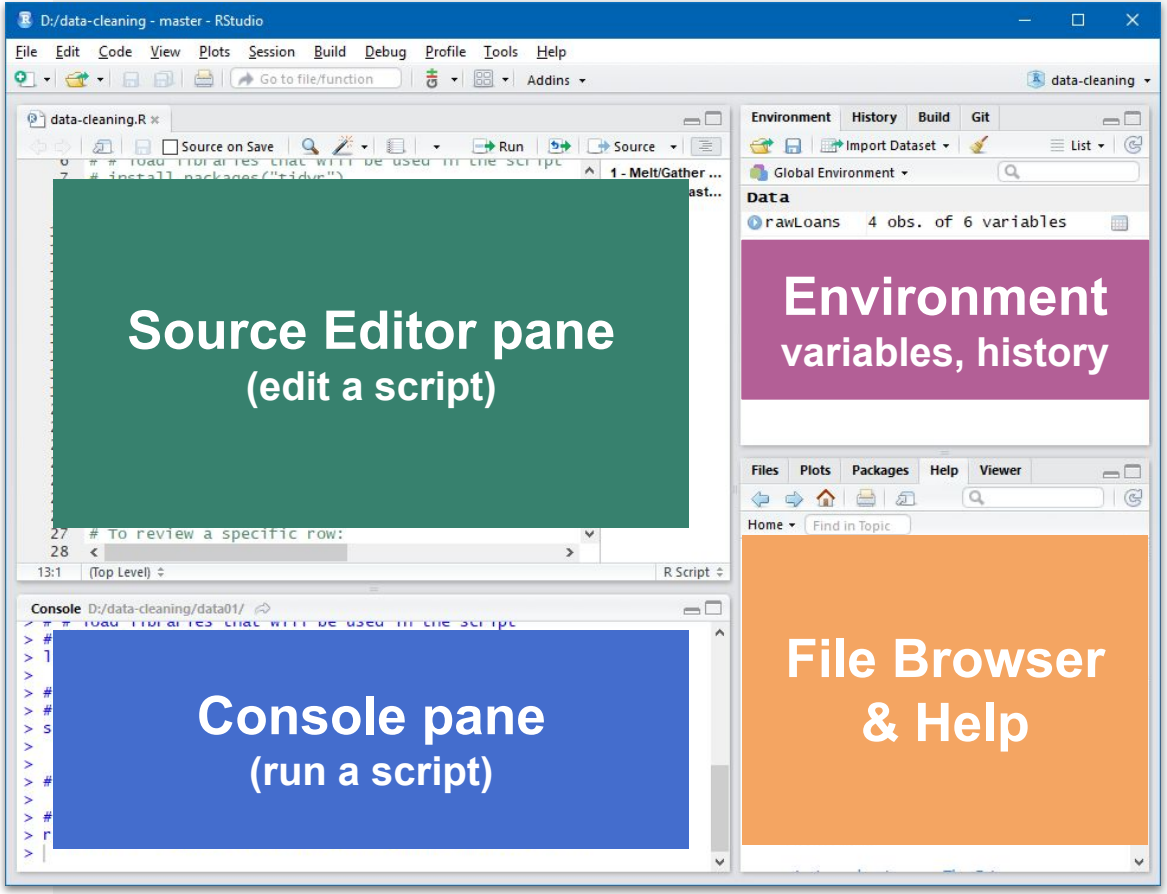
2. = run scripts

Environment

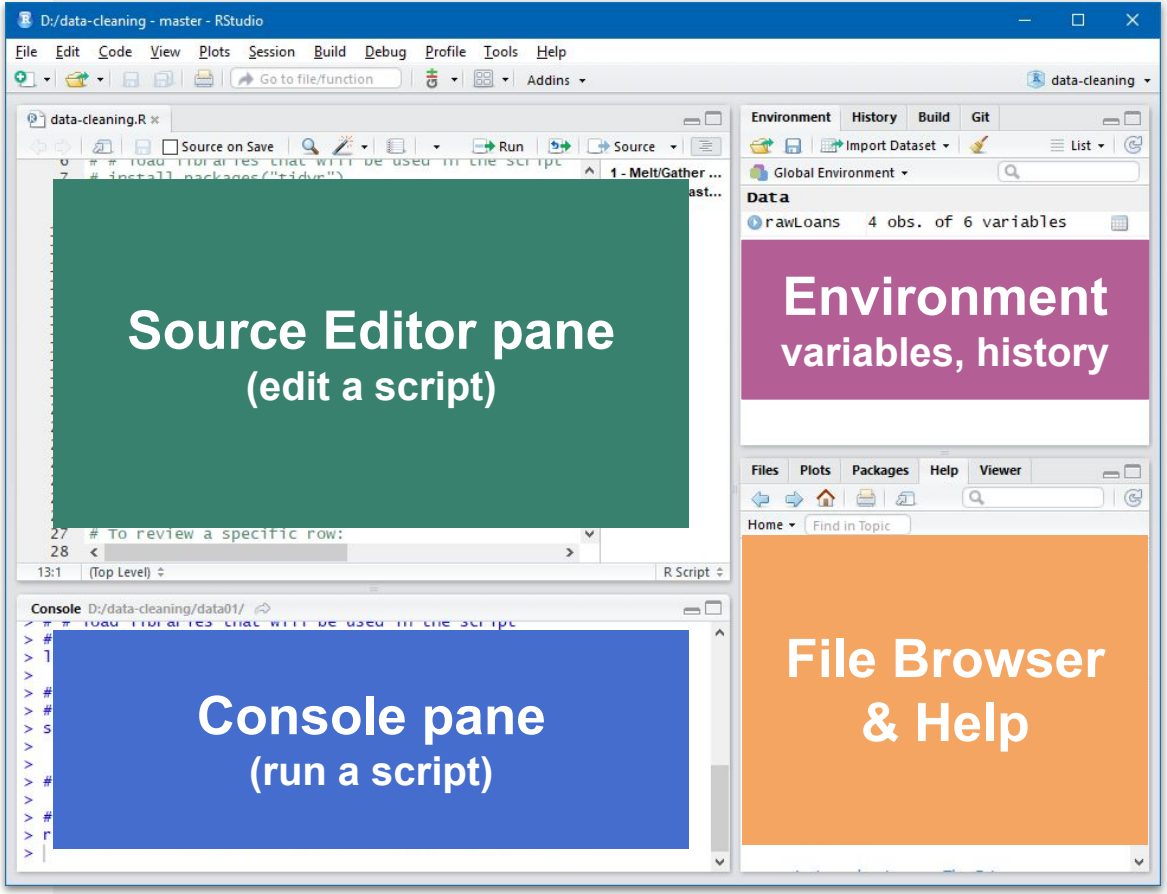
3. = see objects

Help = help!

4. - Google...
5. - Stackoverflow...



R-Studio Interface



More details here:

1. https://github.com/rstudio/cheatsheets/blob/master/source/pdfs/rstudio-IDE-cheatsheet.pdf.

Mapping and Planning

Do your mapping & planning steps!

Specifically:

- Map - which fields map directly to where they go?
- Plan - which fields need to be split/combined/reshaped?

Find your functions & their corresponding packages:

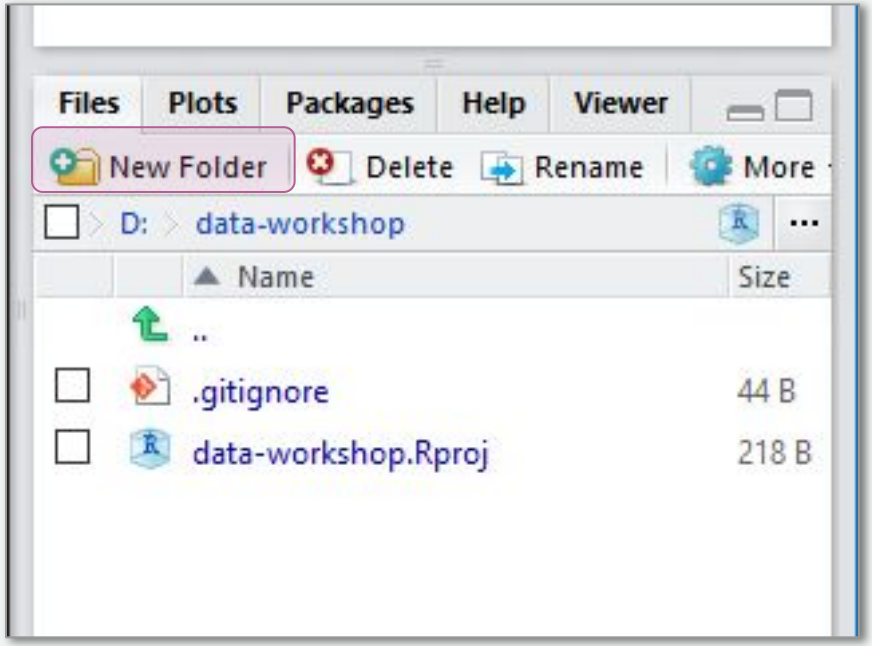
- Google “R how to [sort by one column, add rows, etc]”
- In R Studio, search by name for packages/functions in the Help panel.

A little more R-Studio Setup

In the Help/Files pane (lower right),

- 6) click the “Files” tab:
  - add a folder named **data01raw**
  - and a folder named **data02processed**

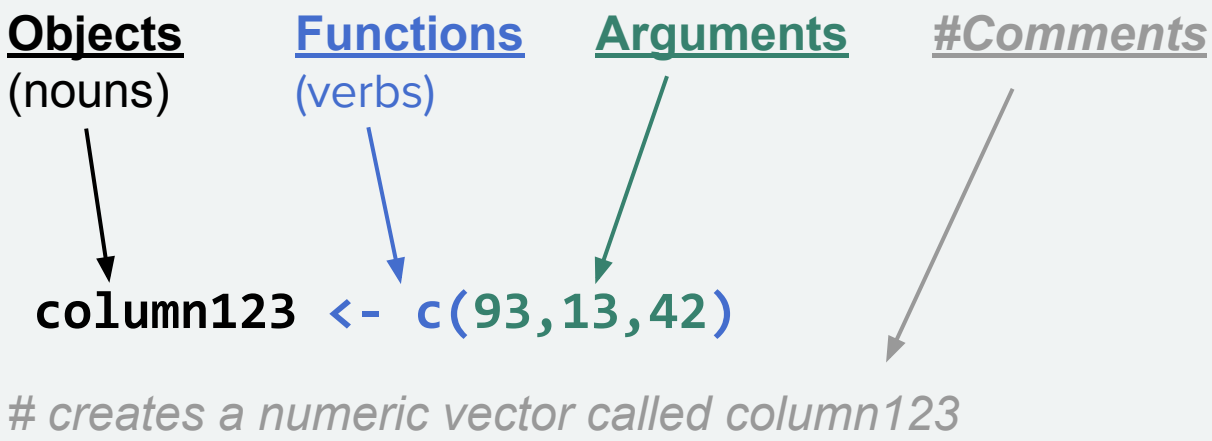
- 7) Outside of RStudio, download **WorkshopData.csv**, and move the file to the **data01raw** folder inside your project directory..



Script Setup

Script Setup (& some R syntax)

R-scripts are text files with a ".R" extension, and [hopefully] working code inside:



Arguments and their options modify what a function does

Script Setup (& some R syntax)

R-scripts are text files with a ".R" extension, and [hopefully] working code inside:

Objects [nouns]: `column123`

- data structures: **Vectors, Data Frames** *(also: Lists, Data tables...)*
- data types: Numeric, Character, Factor, Integer, Logical, NA

Functions [verbs]: `c()`

```
# The function "c()" combines values into a vector
```

- There are many built-in functions in the "base" package.
- More can be added by installing extra packages.

Script Setup (& some R syntax)

R-scripts are text files with a ".R" extension, and [hopefully] working code inside:

Defining some terms:

- A data frame is a table
- A vector is a field/column
- "A package is a collection of R functions, data and compiled code"
- "The location where the packages are stored is called the library"

Comment out lines (or parts of lines) with "#"

Tips on making objects:

- Be consistent with:
  - meaningful & concise names, e.g.: "day_1" (not "first_day_of_the_month" or "djm1")
  - formatting & spacing

Reference:
https://stackoverflow.com/questions/26900228/what-is-the-difference-between-a-library-and-a-package-in-r#:~:text=In%20R%2C%20a%20package%20is%20a%20collection%20of,and%20it%20will%20be%20stored%20in%20your%20library.

Script Setup (& some R syntax)

R-scripts are text files with a ".R" extension, and [hopefully] working code inside:

Formatting in R is very forgiving -- spaces and line breaks won't affect how functions are executed

- BUT be careful -- easy to get messy...

Consider following one of these style guides:

- http://adv-r.had.co.nz/Style.html
- https://google.github.io/styleguide/Rguide.xml
- http://jef.works/R-style-guide/

Basics

Step 2

The first 4 lines of your script should always be comments (#):

1. The script's purpose
2. R version
3. Your EMu user name
4. The date

```
1  # Workshop script for cleaning messy data.
2  # R version 3.4.0 (2017-04-21)
3  # KWebbink
4  # 16-Aug-2017
5
6
7
8
```

Basics

Step 3

Install and load “`tidyr`” package.

```
# Workshop script for cleaning messy data.
# R version 3.4.0 (2017-04-21)
# KWebbink
# 16-Aug-2017

install.packages("tidyr")
library("tidyr")

```

Basics

Step 4

Set the working directory with `setwd()`.

- On Windows systems, a standard DOS file path begins with volume or drive letter, **`C:`** for the local disk.
- In macOS, you may omit the local disk or use **`~`**, which is shorthand for the home directory.

```
# Workshop script for cleaning messy data.
# R version 3.4.0 (2017-04-21)
# KWebbink
# 16-Aug-2017

install.packages("tidyr")
library("tidyr")

# Input data-files will go here:
setwd("C:/path/to/project/data")
```

Basics

Step 5

Save your script by pressing `Ctrl+S`.

- It will appear as a ".R" text file inside your project directory.

```
1  # Workshop script for cleaning messy data.
2  # R version 3.4.0 (2017-04-21)
3  # KWebbink
4  # 16-Aug-2017
5
6  install.packages("tidyr")
7  library("tidyr")
8
9  # Input data-files will go here:
10 setwd("C:/path/to/project/data")
```

FIELD MUSEUM

Executing Commands

Step 6

Run/Execute parts of your script by:

- Highlighting the text
- And pressing `Ctrl+Enter` (or in macOS, `⌘+Return`)

Executed code will show in the Console pane (lower left).

When R is ready to run more code, a new "`>`" prompt will appear:

```
1  # Workshop script for cleaning messy data.
2  # R version 3.4.0 (2017-04-21)
3  # KWebbink
4  # 16-Aug-2017
5
6  install.packages("tidyr")
7  library("tidyr")
8
9  # Input data-files will go here:
10 setwd("C:/path/to/project/data")
```

```
> # Workshop script for cleaning messy data.
>
> install.packages("tidyr")
> library("tidyr")
>
> # Input/Output data-files will go here:
> setwd("C:/path/to/project/data")
>
```

Executing Commands

Step 7

In the Console pane, check your working directory by entering `getwd()` and hitting **Enter**.

```
> install.packages(“tidyr”)
> library(“tidyr”)
>
> # Input/Output data-files will go here:
> setwd(“C:/path/to/project/data”)
> getwd()
[1] “C:/path/to/project/directory”
>
```

Importing and Viewing Data

Importing Data Into R

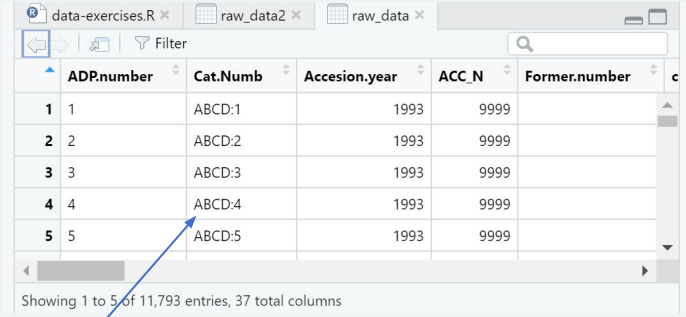
1) Import `WorkshopDataset.csv` with the function `read.csv()`

2) Run line 12-13 (by highlighting them & pressing `Ctrl+Enter`)

…The **Environment** pane will show information about the objects.

3) Click `raw_data` for a summary:

```
9  # Input/Output data-files will go here:
10 setwd("C:/path/to/project/data")
11
12 raw_data <- read.csv("WorkshopDataset.csv",
13                       stringsAsFactors = F)
```



Importing Data Into R

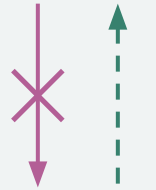
4. **NOTE:** In R, the data frame `rawLoans`…
   - **is a copy** of `01loans.csv` held in your computer's RAM.
   - **is not** a live view of the file "`01loans.csv`"

editing rawLoans … is NOT … editing 01loans.csv

data-cleaning.R × | rawLoans ×

Filter

	Loan	Obj1	Obj2	Obj3	Obj4	Obj5
1	1	PF 123	PF 234	PF 345	PF 456	
2	2	PP 111	PP 112			
3	3	P 987	P 876	P 765	P 654	P 543
4	4	PR 100				



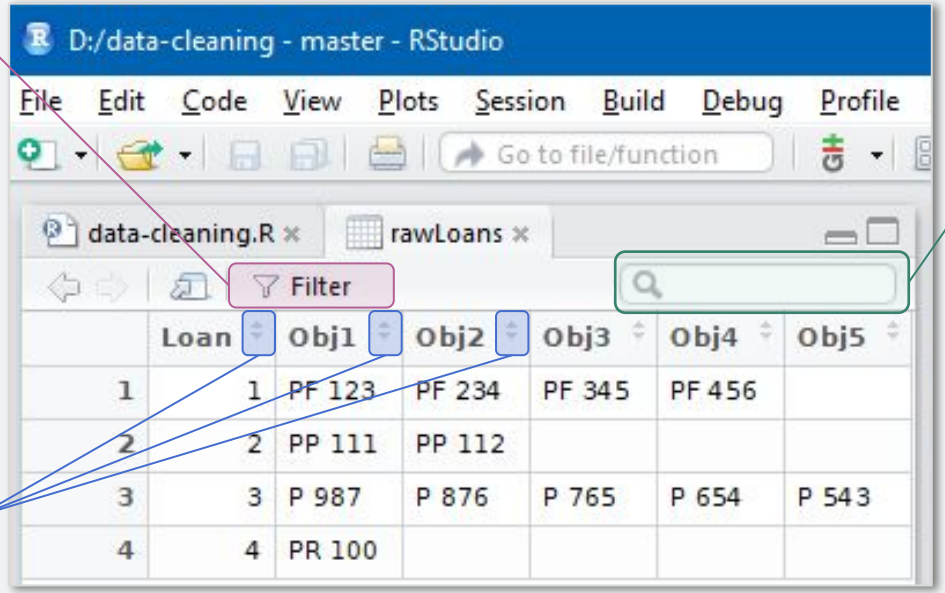
`01loans.csv`

Summarizing in the dataframe preview

Note: Sorting/filtering the data preview does **not** edit the actual dataframe.

Click **Filter** to filter by values in a single column.

Click the **sort-icons** to sort by a column.



Enter a value in the search bar to find records with that value in any field.

Reshaping Data

Reshaping data with `gather()` & `spread()`

`gather()` is similar to what often needs to be done with wide datasets, where multiple columns exist for the same attribute/type of value, and they need to be gathered into a single column.

`spread()` is similar to what often needs to be done with long datasets, where records have multiple rows to allow multiple values of the same attribute

- e.g., multiple rows for one catalog record's giant list of multi-value notes.

gather()

1. Gather (or “melt”) `rawLoans` to make 1 column for “`Loan #`”, & 1 column for “`Objects`”.

```
7  setwd(“C:/path/to/project/data”)
8
9  rawLoans <- read.csv(“01loans.csv”)
10
11 tidyLoans <- gather(rawLoans,
12                      key = “ObjKey”,
13                      value = “Objects”,
14                      2:6)
```

gather()

1. Gather (or “melt”) `rawLoans` to make 1 column for “`Loan #`”, & 1 column for “`Objects`”.

`gather()` takes some arguments:

- `key` = a new single column for the old column-names.
- `value` = a new single column for the corresponding old column-values.
- `x:y` = a range of selected columns (by name or index #) to gather.

```
7   setwd(“C:/path/to/project/data”)
8
9   rawLoans <- read.csv(“01loans.csv”)
10
11  tidyLoans <- gather(rawLoans,
12                      key = “ObjKey”,
13                      value = “Objects”,
14                      2:6)
```

FIELD MUSEUM

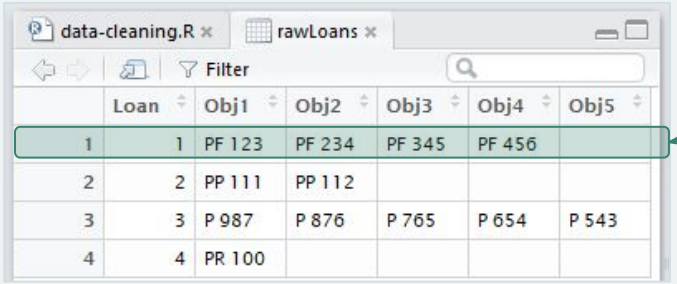
gather()

1. Gather (or “melt”) `rawLoans` to make 1 column for “`Loan #`”, & 1 column for “`Objects`”.
2. Run/execute lines **11-14** by:
   - Highlighting the lines.
   - And pressing `Ctrl+Enter`

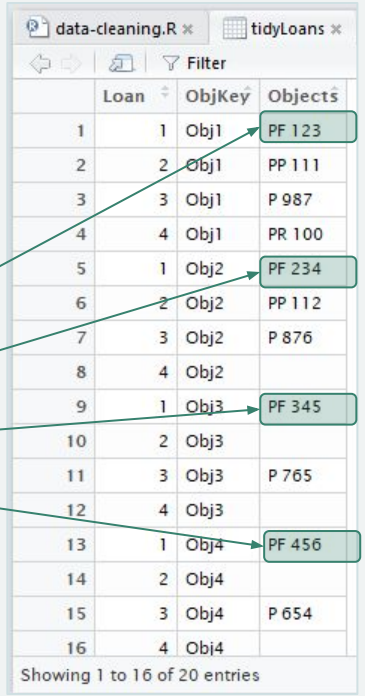
```
7  setwd(“C:/path/to/project/data”)
8
9  rawLoans <- read.csv(“01loans.csv”)
10
11 tidyLoans <- gather(rawLoans,
12                     key = “ObjKey”,
13                     value = “Objects”,
14                     2:6)
```

gather()

```
11  tidyLoans <- gather(rawLoans,
12                      key = "ObjKey",
13                      value = "Objects",
14                      2:6)
```



	Loan	Obj1	Obj2	Obj3	Obj4	Obj5
1	1	PF 123	PF 234	PF 345	PF 456	
2	2	PP 111	PP 112			
3	3	P 987	P 876	P 765	P 654	P 543
4	4	PR 100				



	Loan	ObjKey	Objects
1	1	Obj1	PF 123
2	2	Obj1	PP 111
3	3	Obj1	P 987
4	4	Obj1	PR 100
5	1	Obj2	PF 234
6	2	Obj2	PP 112
7	3	Obj2	P 876
8	4	Obj2	
9	1	Obj3	PF 345
10	2	Obj3	
11	3	Obj3	P 765
12	4	Obj3	
13	1	Obj4	PF 456
14	2	Obj4	
15	3	Obj4	P 654
16	4	Obj4	

Showing 1 to 16 of 20 entries

gather()

```
11  tidyLoans <- gather(rawLoans,
12                      key = "ObjKey",
13                      value = "Objects",
14                      2:6)
```

data-cleaning.R × rawLoans ×

Filter

	Loan	Obj1	Obj2	Obj3	Obj4	Obj5
1	1	PF 123	PF 234	PF 345	PF 456	
2	2	PP 111	PP 112			
3	3	P 987	P 876	P 765	P 654	P 543
4	4	PR 100				

data-cleaning.R × tidyLoans ×

Filter

	Loan	ObjKey	Objects
1	1	Obj1	PF 123
2	2	Obj1	PP 111
3	3	Obj1	P 987
4	4	Obj1	PR 100
5	1	Obj2	PF 234
6	2	Obj2	PP 112
7	3	Obj2	P 876
8	4	Obj2	
9	1	Obj3	PF 345
10	2	Obj3	
11	3	Obj3	P 765
12	4	Obj3	
13	1	Obj4	PF 456
14	2	Obj4	
15	3	Obj4	P 654
16	4	Obj4	

Showing 1 to 16 of 20 entries

subsetting [ ]

1. Gather (or “melt”) `rawLoans` to make 1 column for “`Loan #`”, & 1 column for “`Objects`”.

If we don’t want the `ObjKey` column in the new dataframe:

2. Subset `tidyLoans` with lines 16-17 (`tidyLoans[row,column]`)

```
11  tidyLoans <- gather(rawLoans,
12                      key = “ObjKey”,
13                      2:6)
14
15
16   # Drop the ObjKey column by subsetting:
17   tidyLoans2 <- tidyLoans [,2]
```

subsetting [ ]

In dataframes: rows, columns, and values can be referenced by subsetting:

by **numeric index**

- `tidyLoans[1,3]`
  - # returns the value in row 1, column 3
- `tidyLoans[1,]`
  - # returns row 1 (all values)
- `tidyLoans[,-3]`
  - # removes column 3 (all values)

by **$name**

- `tidyLoans$Loan`
  - # returns all values in “Loan” column
- `tidyLoans$Loan[2]`
  - # returns 2nd value in “Loan” column

by **logical condition**

- `tidyLoans[tidyLoans$Loan==2,]`
  - # returns rows where “Loan” = “2”

Exporting Data

Exporting Data

1. Export tidyLoans2 with the function `write.csv()`.
2. Run/execute lines **18-19** by:
   - Highlighting the lines.
   - And pressing `Ctrl+Enter`

```
15  # Drop the ObjKey column by subsetting:
16  tidyLoans2 <- tidyLoans[,-2]
17
18  write.csv(tidyLoans2,
19            file = “tidy.csv”)
```

Exporting Data

1. Export tidyLoans2 with the function `write.csv()`.
2. Run/execute lines **18-19** by:
   - Highlighting the lines.
   - And pressing `Ctrl+Enter`

```
15  # Drop the ObjKey column by subsetting:
16  tidyLoans2 <- tidyLoans[,-2]
17
18  write.csv(tidyLoans2,
19            file = “tidy.csv”)
```

3. Open up “tidy.csv” in Excel…

Exporting Data

1. Export tidyLoans2 with the function `write.csv()`.
2. Run/execute lines **18-19** by:
   - Highlighting the lines.
   - And pressing `Ctrl+Enter`

```
15  # Drop the ObjKey column by subsetting:
16  tidyLoans2 <- tidyLoans[,-2]
17
18  write.csv(tidyLoans2,
19            file = “tidy.csv”,
19            row.names = F)
```

3. Open up “tidy.csv” in Excel…
   - Is it mangled?
   - Try setting the argument “`row.names`” to “`FALSE`”:

Extras

Some other functions to try…

Get summaries of columns

```
summary(rawLoans)
```

```
> summary(rawData)
Cat..Numb.          University                  Collector
UWP:122471: 2   Univ of Guatemala:760     Vargas I   : 207
UWP:157339: 2                             Betancur J : 203
UWP:100217: 1                             Callejas R : 48
```

Some other functions to try…

Use regular expressions:

```
gsub("\\s+",
     " ",
     rawLoans$Loans)
```

```
# replace multiple spaces…
# …with a single space…
# …in this column.
```

Some other functions to try…

Run an R script in R-Studio:

```
setwd(“path/to/script-folder”)
source(“script.R”)
```

```
# output objects will show
in Environment pane
```

Git

And now for something
[slightly] different:

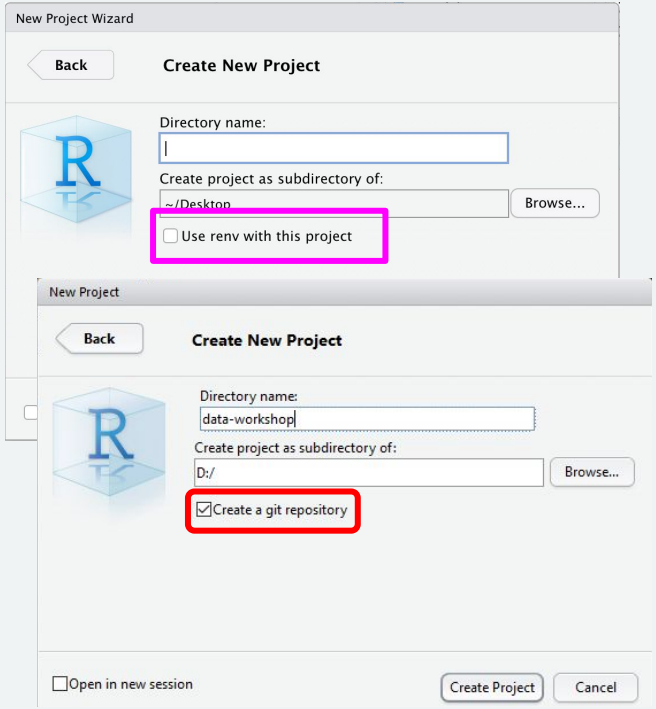
168 lines (135 sloc) | 8.2 KB

```
1
2           GIT - the stupid content tracker
3
4   "git" can mean anything, depending on your mood.
5
6    - random three-letter combination that is pronounceable, and not
7      actually used by any common UNIX command.  The fact that it is a
8      mispronounciation of "get" may or may not be relevant.
9    - stupid. contemptible and despicable. simple. Take your pick from the
10     dictionary of slang.
11   - "global information tracker": you're in a good mood, and it actually
12     works for you. Angels sing, and a light suddenly fills the room.
13   - "goddamn idiotic truckload of sh*t": when it breaks
```

Git

What is this thing called git?

- **Git:** a way to share and track changes in code (or any digital content stored in a file or directory) control
  - If (when) you break your code, you can use git to retrieve (pull) a previously saved (pushed) version.
- **Github.com** & **Bitbucket.com**
  - Online code repositories that use git.
  - Users can share (fork/branch) code.



R-Studio – More Help…

Installation

- *1st* – Install R
  - cran.r-project.org
- *2nd* – Install R-Studio
  - rstudio.com/products/rstudio/download

For more help:

- R-Studio's built-in help
- www.rOpenSci.org
- RStudio.com/resources/cheatsheets

Other R Courses/Resources:

- **Roger Peng** / Johns Hopkins
  - class: r-programming
  - blog: SimplyStatistics.org
- **Data Carpentry** / R for Ecology
  - datacarpentry.org/R-ecology-lesson/

Thank you!

Sharon Grant
sgrant@fieldmuseum.org
https://www.fieldmuseum.org/

Janeen Jones
jjones@fieldmuseum.org
https://www.fieldmuseum.org/

Kate Webbink
kwebbink@fieldmusuem.org
https://www.fieldmuseum.org/

Abigail McArthur-Self
amcarthur-self@fieldmusuem.org
https://www.fieldmuseum.org/

Alexis Ramirez
aramirez@fieldmusuem.org
https://www.fieldmuseum.org/

This project was made possible in part by the **Institute of Museum and Library Services** Grant ME-249136-OMS-21 | **IMLS.gov**