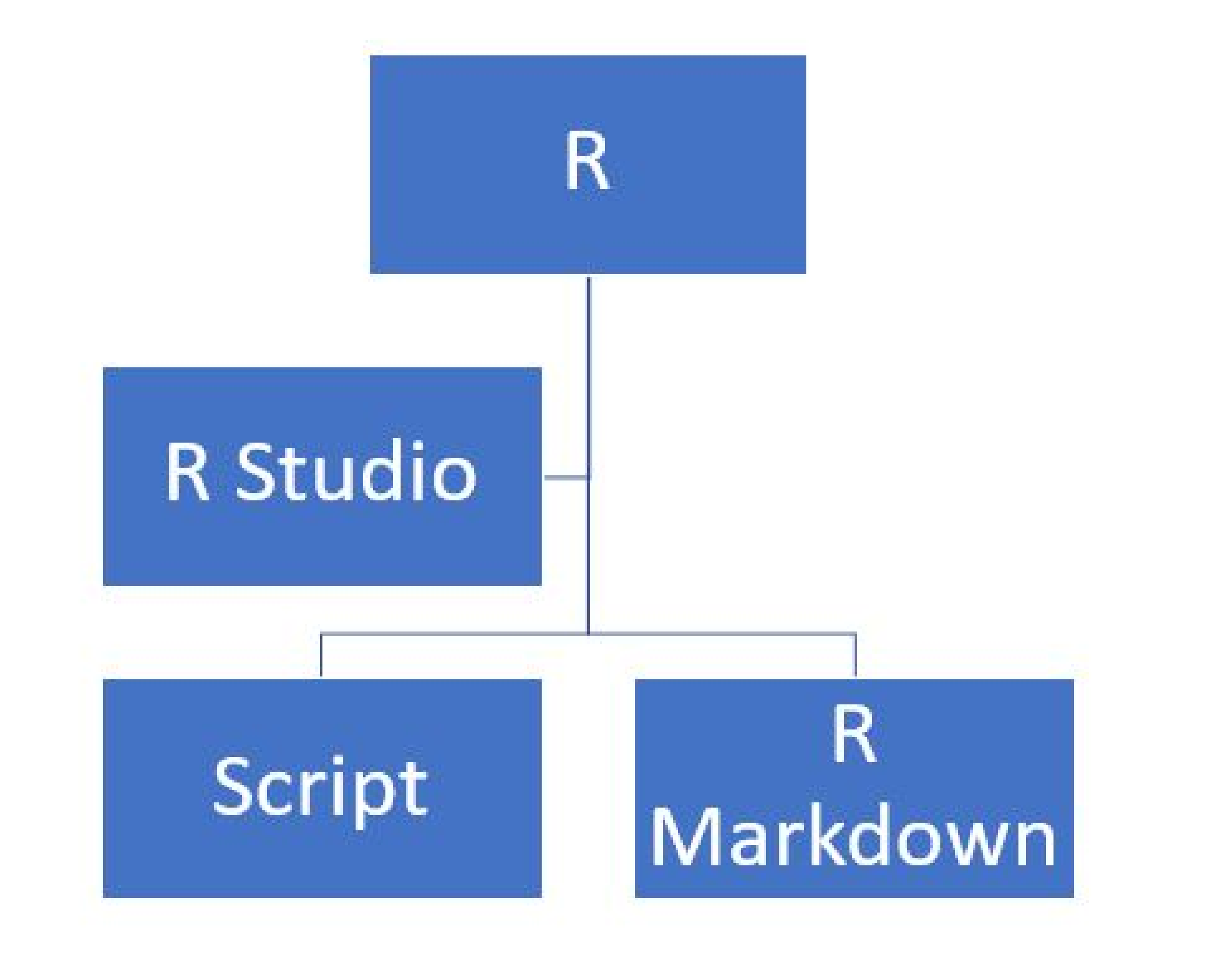
Tidyverse I

Math Camp

2023

R Studio

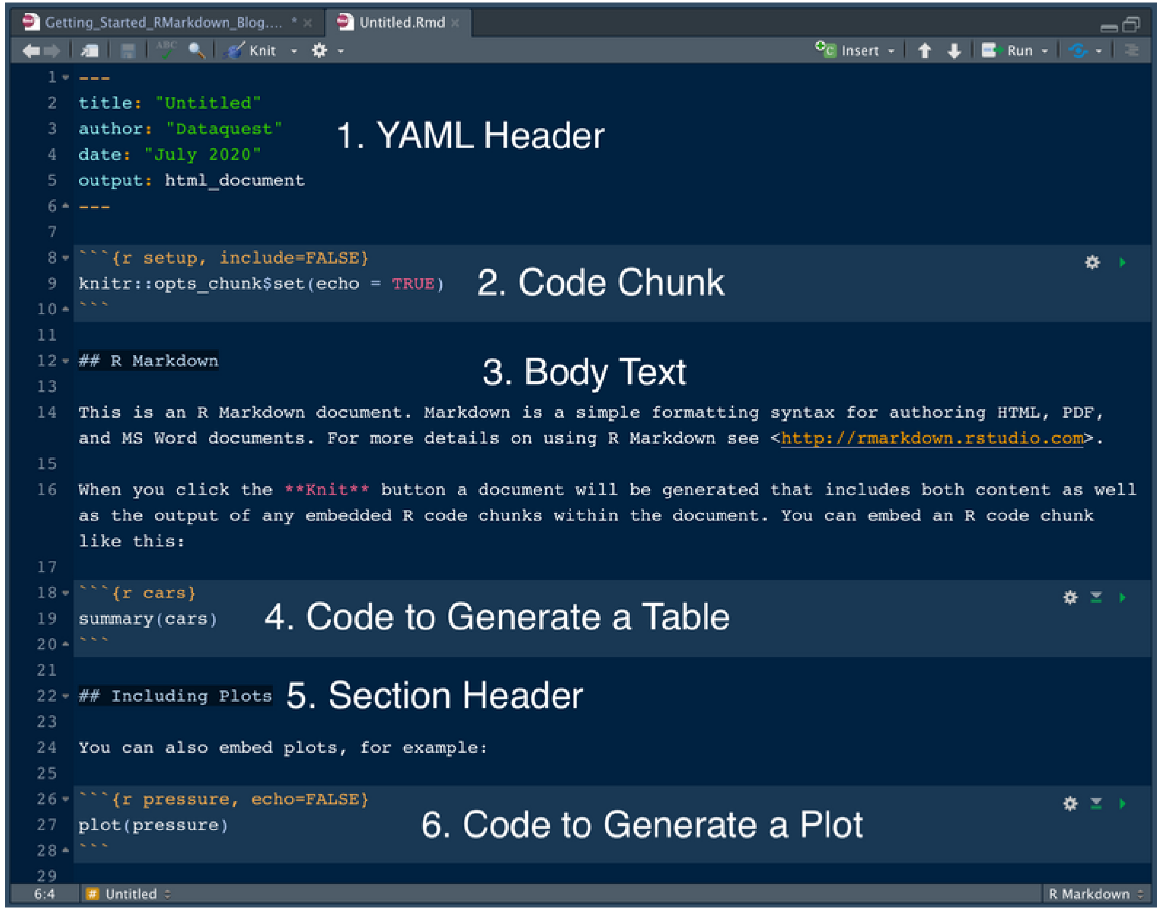


R Markdown

Within R Studio, we can create a type of document called Markdown.

- Neat way of combining code + text
- Used for final documents/presentation of results/assignments
- Has a visual editor that makes it look like a normal word processor

R Markdown


Getting_Started_RMarkdown_Blog....
Untitled.Rmd
Knit
Insert
Run
---
title: "Untitled"
author: "Dataquest"
date: "July 2020"
output: html_document
---
1. YAML Header
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
2. Code Chunk
R Markdown
3. Body Text
This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:
```{r cars}
summary(cars)
```
4. Code to Generate a Table
Including Plots
5. Section Header
You can also embed plots, for example:
```{r pressure, echo=FALSE}
plot(pressure)
```
6. Code to Generate a Plot
6:4
Untitled
R Markdown

R Markdown

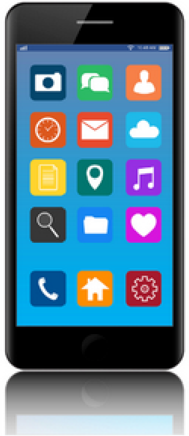
You can "knit" the document to:

- html
- Word
- pdf

For this last one, you need some version of Latex:

```
install.packages('tinytex')
```

Packages/Libraries

R: A new phone	R Packages: Apps you can download
	

On phone	On R
Download app	install.packages("")
Open app	library()

Loading libraries/packages

If you load a library without installing it first, you will get this error:

```r
library(BioConductor)
```

```
## Error in library(BioConductor): there is no package called 'BioConductor'
```

If you try to run a command from a package without loading it first, you will get this error:

```r
stargazer(mtcars)
```

```
## Error in stargazer(mtcars): could not find function "stargazer"
```

Tidyverse

A combination of packages that are useful for: importing, cleaning and transforming, processing and analyzing, visualizing.

Tidyverse

When you install (and load) tidyverse, you're installing (and loading) all of these packages. You can, instead, load the individual packages you'll use.

Notice the message you get:

```
> library(tidyverse)
-- Attaching packages --------------------------------------- tidyverse 1.3.0 --
v ggplot2 3.3.2     v purrr   0.3.4
v tibble  3.0.3     v dplyr   1.0.2
v tidyr   1.1.0     v stringr 1.4.0
v readr   1.3.1     v forcats 0.5.0
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
>
```

Reading Data into R

- Do not use the Graphical User Interface
    - It runs the code in the console
    - This means the data won't import next time
- Syntax: `objectname <- function("filename.extension", arguments)`
- e.g. `survey_data <- read_csv("2020_presidential_survey_August.csv)`

Reading Data into R

- Function depends on data type:
  - .csv - `read_csv("filename.csv")` (this requires `library(readr)`)
  - .xlsx - `read_excel("filename.xlsx")` (this requires `library(readxl)`)
  - .dta - `read_dta("filename.dta")` (this requires `library(haven)`)

Where is the file?

- This is crucial
- R uses relative filepaths
- In an .RMD file, the location of the file is the working directory
- if the dataset is in the same directory, just give its name
- if it's in a subdirectory (say `data`) write it
  `read_csv("data/file.csv")`

Using Projects

- Easy way of staying organized
- No need to refer to file paths
- All the files are contained within the Project

Dplyr for Data Manipulation

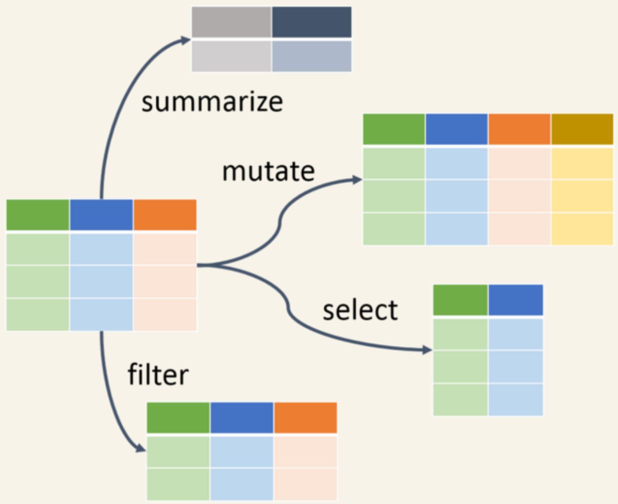
1) select

2) filter

3) mutate

4) summariz(s)e

5) group_by



Dplyr: Pipes %>%

- Read `%>%` as "then"
- It sends the result of one function to another function
- It lets us perform multiple operations at once, without creating an object for each one of them.
- `%>%` increases the readability of your code

- Basic Syntax: `function(object, arguments)`

- piped syntax: `object %>% function(arguments)`

Chicago Neighborhoods

Let's use the data on Chicago neighborhoods.

1- Note its file extension

2- Note its location

3- Write the appropriate command

```
chicago <- read_csv("Census Data_selected_2008-2012.csv")
```

Exploring data

Once you have uploaded a dataset, it is important to first become familiarized with it.

Some useful commands: `glimpse`, `names`, `head`, `View`

```
glimpse(chicago)
```

Renaming

Notice the variable names are quite messy. Let's fix this with `rename`.

- syntax: `rename(newvariable = oldvariable)`

```
chicago %>% (neighborhood = COMMUNITY.AREA.NAME)
```

Let's check if this worked:

```
names(chicago)
```

Rename

- Our code above just outputs a dataframe, but it's not saved anywhere.
- Our original object hasn't changed! We need to overwrite it:

```
chicago <- chicago %>% rename(neighborhood = COMMUNITY.AREA.NAME)
```

PRACTICE

Rename the following variables:

1) PERCENT.HOUSEHOLDS.BELOW.POVERTY into "poverty"

2) PERCENT.AGED.25+.WITHOUT.HIGH.SCHOOL.DIPLOMA into "diploma".

3) COMMUNITY.AREA.NAME into "neighborhood".

Select

Sometimes you will be working with large datasets that have hundreds of variables. But what if you want to work with only some of these variables?

- We can create a dataset that has only the variables we want by using the `select` function

```
small_chicago <- chicago %>%
  select(poverty, diploma, neighborhood)
```

Select

- select by name:
  - `select(poverty, diploma)`
- select by position:
  - `select(c(1, 3, 10))`
- select by range:
  - `select(neighborhood:poverty)` or `select(1:3)`
- drop variables with `-`
  - `select(-diploma)`
- "select helpers" that make subsetting variables very easy
  - `contains()`, `starts_with()`, `ends_with()`

PRACTICE

Create a data set with all the original variables except "community area number"

ANSWERS

```
chicago <- chicago %>% select(-"Community.Area.Number")

names(chicago)
```

Filter

Instead of picking out whole columns (i.e. variables), we can pick out **observations** that comply with given condition(s).

We usually do this to perform operations only on a subset of cases

- syntax: `data %>% filter(variable CONDITION)`

Filter and R's Logical Operators

Operator	Description
<	less than
<=	less than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x & y	x AND y
x | y	x OR y
isTRUE(x)	test if X is TRUE

Filter

Create a subset of the data only with neighborhoods were poverty is about 20%

```
pov20 <- chicago %>% filter(poverty > 20)
```

You can combine multiple conditions too:

```
chicago %>%
  filter(poverty > 20 & diploma > 40 & neighborhood != "Hermosa")
```

PRACTICE

1) Create a new dataset that only has North Side neighborhoods and where per capita income is above $40,000.

2) How many neighborhoods are left?

ANSWERS

```
chicago_new <- chicago %>% filter(ZONE == "North" & PER.CAPITA.INCOME >40000)
```

%in%

Can come in very handy when filtering.

```
chicago %>% filter(ZONE == "North" | ZONE=="South")
```

Is somewhat easier to write as:

```
chicago %>% filter(ZONE %in% c("North", "South"))
```

Slice

- Filtering observations based on their *position*.
- Useful to see, for instance, cases at the top, bottom, etc.

```
chicago %>% slice(1:5)
```

But note that if your dataset is manipulated somehow, this might change!

Slice

There are other very useful slices:

- slice_head(n=)
- slice_tail(prop=0.2)
- slice_min/slice_max(variable, n=)
- slice_sample(prop=0.5)

```
chicago %>% slice_sample(prop=0.2)
```

PRACTICE

Create a dataset that has the 10 neighborhoods with the highest levels of crowded housing

ANSWER

```
chicago_crowded <- chicago %>% slice_max(PERCENT.OF.HOUSING.CROWDED, n = 10)
```

Arrange

- Arranges observations in ascending or descending order
- Useful for viewing, esp. outputs of other commands
- syntax: data %>% arrange(name of variable)
- syntax: data %>% arrange(desc(name of variable))

```
chicago <- chicago %>% arrange(diploma)
```

Mutate

- The `mutate` function allows us to create new variables
- these can be based on some transformation of an existing variable (or not)
- syntax: data %>% mutate(new variable name = operation)

```
chicago %>% mutate(newvar = 1)
```

Mutate

Let's say we want to convert the poverty numbers into a proportion.

```
chicago <- chicago %>% mutate(pov_prop = poverty/100)
```

Notice how we first assign a new variable name and then tell R how we want that variable to be created.

PRACTICE

Imagine that you are creating a simple additive index of 3 indicators: per capita income, percent of housing crowded, and percent without diploma.

Create a new variable with the sum of these values and divide it by 3.

ANSWERS

```
chicago <- chicago %>% mutate(index = (PER.CAPITA.INCOME + PERCENT.OF.HOUSING.CROWDED + diploma)/3)
```

mutate, but (simple) conditional

- When want the new variable to have one value in some cases, but another value otherwise
- if only two conditions, you can use `ifelse`. If more, `case_when`
- both can be very useful to recode NAs

mutate + ifelse

```
chicago <- chicago %>% mutate(pov_high = ifelse(poverty >=30, "High", "Low"))
```

- mutate(pov_high = create a new variable called "pov_high"
- ifelse(poverty >30, "High"...) = if poverty (the original variable) is more than than 30, assign it a value of "high" in the new variable pov_high
- ifelse(...,"Low") = when it is not less than 30, assing it a value of "low" in the new variable pov_high

mutate + case_when

Illustration By Allison Horst

mutate + case_when

- syntax: `mutate(newvar = case_when(condition1 ~ value1, condition2 ~ value2))`

```
chicago %>% mutate(pov_high = case_when(
  poverty >= 30 ~ "High",
  poverty <30 ~ "Low"
))
```

mutate + case_when

Can combine more conditions!

```
chicago %>% mutate(pov = case_when(
  poverty >= 30 ~ "High",
  poverty >=15 & poverty< 30 ~ "Med",
  poverty < 15 ~ "Low"
))
```

PRACTICE

Create a new variable called "region" that groups neighborhoods into three categories:

1. North + Northwest = "North"
2. West + Southwest = "West"
3. South + Far South = "South"

ANSWER

```
chicago <- chicago %>% mutate(region = case_when(
  ZONE == "North" | ZONE=="Northwest" ~ "North",
  ZONE == "West" | ZONE=="Southwest" ~ "West",
  ZONE == "South" | ZONE=="Far South" ~ "South"
))
```

Summariz(s)e

`summarise` reduces observations to a single value based on functions

- syntax: `summarize(name = function(variable))`

```
chicago %>%
  summarize(avg_pov = mean(poverty))
```

Summariz(s)e

Let's add other descriptive statistics:

```
chicago %>% summarize(mean_pov = mean(poverty),
                      sd_pov = sd(poverty),
                      max_pov = max(poverty))
```

Summarize

Note what happens if there are NAs in your column:

```
chicago %>% summarize(mean_hardship  = mean(HARDSHIP.INDEX))
```

Once you've explored your NAs and you want to bypass this:

```
chicago %>% summarize(mean_hardship  = mean(HARDSHIP.INDEX, na.rm = TRUE))
```

Count

Quick frequency table with a much nicer output than the base R `table` function

- syntax: count(name of variable)

```
chicago %>% count(ZONE)
```

NAs!

Datasets are almost never complete, and dealing with NAs is a crucial part of data wrangling and analysis.

The result above tells us there are 3 rows that do not data for ZONE. We can use our `filter` command to see which cases these are:

```
chicago %>% filter(is.na(ZONE))
```

NAs!

If we want to include "Loop" and "Near South Side" in our analysis by Zone, we should assign them some value.

```
chicago <- chicago %>% mutate(region = case_when(
  neighborhood == "Loop" ~ "North",
  neighborhood == "Near South Side" ~  "South",
  TRUE ~ region
))
```

The last line means that all the remaining cases that do not comply with either of the above conditions retain their original values.

NAs

We may also choose to drop the "CHICAGO" row if we are not interested in having it in our analysis.

How would we do this?

```
chicago <- chicago %>% filter(neighborhood!="CHICAGO")
```

Grouping

- Many times you'll want to analyze data across groups (regions, gender, age group, political party, etc.). We can use the `group_by()` function for this

- syntax: `group_by(variable) %>%`

```
chicago %>%
  group_by(ZONE) %>%
  summarize(avg_pov = mean(poverty))
```

Pipes are very useful here!

PRACTICE

What is the zone with the highest average per capita income?

Hint: use `arrange` after summarizing to view the output in ascending order

ANSWERS

```
chicago %>% group_by(ZONE) %>%
  summarize(avg_income = mean(PER.CAPITA.INCOME)) %>% arrange(desc(avg_income))
```