R Workshop #1

Ryan Safner

11/27/2018

Data Wrangling

Subsetting Data

Dealing with Missing Data

Merging Datasets

Tidy Data

Managing Your Workflow

Advanced: Automate All The Things!

Data Wrangling

Data Types

- A good time to refresh the different types of data

Data Types

- A good time to refresh the different types of data
    1. `Numeric`

Data Types

- A good time to refresh the different types of data
    1. `Numeric`
        - Called "`double`" if non-integer (i.e. has a decimal point)

Data Types

- A good time to refresh the different types of data

  1. `Numeric`

     - Called “**`double`**” if non-integer (i.e. has a decimal point)
     - Integer if an integer

Data Types

- A good time to refresh the different types of data
  1. **`Numeric`**
     - Called “**`double`**” if non-integer (i.e. has a decimal point)
     - Integer if an integer
  2. **`Character`**: any string of text (needs “quotes”)

Data Types

- A good time to refresh the different types of data
  1. `Numeric`
     - Called "`double`" if non-integer (i.e. has a decimal point)
     - Integer if an integer
  2. `Character`: any string of text (needs "quotes")
  3. `Factor`: a type of character or integer that signifies membership in a category (categories are called "`levels`")

Data Types

- A good time to refresh the different types of data
    1. `Numeric`
        - Called "**`double`**" if non-integer (i.e. has a decimal point)
        - Integer if an integer
    2. `Character`: any string of text (needs "quotes")
    3. `Factor`: a type of character or integer that signifies membership in a category (categories are called "`levels`")
        - Ordered factor if order/rank of levels matter (i.e. `1>2>3>4` or `freshman<sophomore<junior<senior`)

Data Types

- A good time to refresh the different types of data
  1. `Numeric`
     - Called "`double`" if non-integer (i.e. has a decimal point)
     - Integer if an integer
  2. `Character`: any string of text (needs "quotes")
  3. `Factor`: a type of character or integer that signifies membership in a category (categories are called "`levels`")
     - Ordered factor if order/rank of levels matter (i.e. `1>2>3>4` or `freshman<sophomore<junior<senior`)
  4. `Logical`: binary variable with values either `TRUE` or `FALSE`

Data Types

- A good time to refresh the different types of data

1. `Numeric`
    - Called “`double`” if non-integer (i.e. has a decimal point)
    - Integer if an integer
2. `Character`: any string of text (needs “quotes”)
3. `Factor`: a type of character or integer that signifies membership in a category (categories are called “`levels`”)
    - Ordered factor if order/rank of levels matter (i.e. `1>2>3>4` or `freshman<sophomore<junior<senior`)
4. `Logical`: binary variable with values either `TRUE` or `FALSE`
    - Not really applicable for our purposes

Data Types

- A good time to refresh the different types of data
    1. `Numeric`
        - Called "**`double`**" if non-integer (i.e. has a decimal point)
        - Integer if an integer
    2. `Character`: any string of text (needs "quotes")
    3. `Factor`: a type of character or integer that signifies membership in a category (categories are called "`levels`")
        - Ordered factor if order/rank of levels matter (i.e. `1>2>3>4` or `freshman<sophomore<junior<senior`)
    4. `Logical`: binary variable with values either `TRUE` or `FALSE`
        - Not really applicable for our purposes
    5. `Date/time`: a point in time (Year, Month, Day, Hour, Min, Sec, etc)

Data Types

- A good time to refresh the different types of data
  1. `Numeric`
     - Called “`double`” if non-integer (i.e. has a decimal point)
     - Integer if an integer
  2. `Character`: any string of text (needs “quotes”)
  3. `Factor`: a type of character or integer that signifies membership in a category (categories are called “`levels`”)
     - Ordered factor if order/rank of levels matter (i.e. `1>2>3>4` or `freshman<sophomore<junior<senior`)
  4. `Logical`: binary variable with values either `TRUE` or `FALSE`
     - Not really applicable for our purposes
  5. `Date/time`: a point in time (Year, Month, Day, Hour, Min, Sec, etc)
     - Try to avoid if possible, if needed, ask me!

Data Wrangling: Why Wrangle?

- **Multiple Sources of Data**: may want to merge multiple spreadsheets into one `data.frame`

Data Wrangling: Why Wrangle?

- **Multiple Sources of Data**: may want to merge multiple spreadsheets into one `data.frame`
- **Un-tidy Data**: data might come in a spreadsheet in a form that is not friendly to plotting, regression, etc

Data Wrangling: Why Wrangle?

- **Multiple Sources of Data**: may want to merge multiple spreadsheets into one `data.frame`
- **Un-tidy Data**: data might come in a spreadsheet in a form that is not friendly to plotting, regression, etc
- **Getting a Better Look**: a lot of data analysis happens before the plots and regressions, need to know *what data we have* and how it looks like

Data Wrangling: Why Wrangle?

- **Multiple Sources of Data**: may want to merge multiple spreadsheets into one `data.frame`
- **Un-tidy Data**: data might come in a spreadsheet in a form that is not friendly to plotting, regression, etc
- **Getting a Better Look**: a lot of data analysis happens before the plots and regressions, need to know *what data we have* and how it looks like
    - May need to rescale, transform, or create new variables

Data Wrangling: Why Wrangle?

- **Multiple Sources of Data**: may want to merge multiple spreadsheets into one `data.frame`
- **Un-tidy Data**: data might come in a spreadsheet in a form that is not friendly to plotting, regression, etc
- **Getting a Better Look**: a lot of data analysis happens before the plots and regressions, need to know *what data we have* and how it looks like
    - May need to rescale, transform, or create new variables
    - May want to **subset** or look at data **conditionally** for patterns, comparing groups, eliminate outliers, etc

Data Wrangling: Why Wrangle?

- This is all “stuff under the hood”

Data Wrangling: Why Wrangle?

- This is all “stuff under the hood”
  - Most analysis requires this but you never see it in the final paper or figures

Data Wrangling: Why Wrangle?

- This is all “stuff under the hood”
    - Most analysis requires this but you never see it in the final paper or figures
- Generous researchers take raw, messy data and offer a copy of the final, cleaned, dataset along with their process of how they wrangled it

Subsetting Data

Subsetting Data: Matrix Analogy

- `data.frame` is a type of `matrix`: each cell is indexed by its `[row #, column #]`

```
m<-matrix(c("a","b","c","d","e","f"),nrow=2)
m
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "c"  "e"
## [2,] "b"  "d"  "f"
```

Subsetting Data: Matrix Analogy

- `data.frame` is a type of `matrix`: each cell is indexed by its `[row #, column #]`

```
m<-matrix(c("a","b","c","d","e","f"),nrow=2)
m
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "c"  "e"
## [2,] "b"  "d"  "f"
```

- Subset a **specific row**:

```
m[2,]
```

```
## [1] "b" "d" "f"
```

Subsetting Data: Matrix Analogy

- data.frame is a type of matrix: each cell is indexed by its [row #, column #]

```
m<-matrix(c(”a”,”b”,”c”,”d”,”e”,”f”),nrow=2)
m
```

```
##      [,1] [,2] [,3]
## [1,] ”a”  ”c”  ”e”
## [2,] ”b”  ”d”  ”f”
```

- Subset a **specific row**:

```
m[2,]
```

```
## [1] ”b” ”d” ”f”
```

- Subset a **specific column**:

```
m[,3]
```

```
## [1] ”e” ”f”
```

Subsetting Data: Matrix Analogy

- `data.frame` is a type of `matrix`: each cell is indexed by its `[row #, column #]`

```
m<-matrix(c(”a”,”b”,”c”,”d”,”e”,”f”),nrow=2)
m
```

```
##      [,1] [,2] [,3]
## [1,] ”a”  ”c”  ”e”
## [2,] ”b”  ”d”  ”f”
```

- Subset a **specific row**:

```
m[2,]
```

```
## [1] ”b” ”d” ”f”
```

- Subset a **specific column**:

```
m[,3]
```

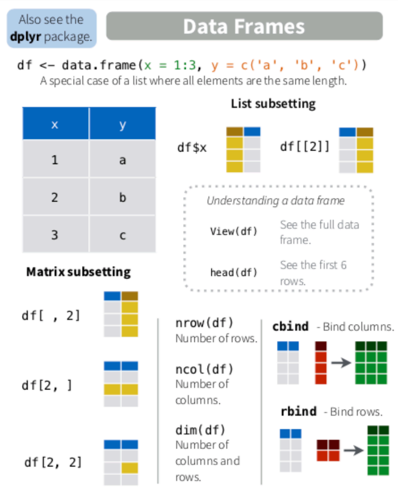
```
## [1] ”e” ”f”
```

- Subset a **specific element**:

```
m[2,3]
```

```
## [1] ”f”
```

Subsetting Data



Also see the **dplyr** package.

Data Frames

```
df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))
```

A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting

`df$x`

`df[[2]]`

Understanding a data frame

`View(df)` See the full data frame.

`head(df)` See the first 6 rows.

Matrix subsetting

`df[ , 2]`

`df[2, ]`

`df[2, 2]`

`nrow(df)` Number of rows.

`ncol(df)` Number of columns.

`dim(df)` Number of columns and rows.

cbind - Bind columns.

rbind - Bind rows.

Subsetting Data: Data Frames

- We can do the same thing for `data.frame`s:

```
df<-data.frame(Nums=c(1,2,3,4,5),
               Lets=c(”a”,”b”,”c”,”d”,”e”))
df
```

```
##   Nums Lets
## 1    1    a
## 2    2    b
## 3    3    c
## 4    4    d
## 5    5    e
```

Subsetting Data: Data Frames

- Can also subset a `data.frame` by position:

```
df
```

```
##   Nums Lets
## 1    1    a
## 2    2    b
## 3    3    c
## 4    4    d
## 5    5    e
```

Subsetting Data: Data Frames

- Can also subset a `data.frame` by position:

```
df
```

```
##   Nums Lets
## 1    1    a
## 2    2    b
## 3    3    c
## 4    4    d
## 5    5    e
```

- Subset a **specific row (observation)**:

```
df[2,]
```

```
##   Nums Lets
## 2    2    b
```

Subsetting Data: Data Frames

- Can also subset a `data.frame` by position:

```
df
```

```
##   Nums Lets
## 1    1    a
## 2    2    b
## 3    3    c
## 4    4    d
## 5    5    e
```

- Subset a **specific row (observation)**:

```
df[2,]
```

```
##   Nums Lets
## 2    2    b
```

- Subset a **specific column (variable)**:

```
df[,2]
```

```
## [1] ”a” ”b” ”c” ”d” ”e”
```

Subsetting Data: Data Frames

- Can also subset a `data.frame` by position:

```
df
```

```
##   Nums Lets
## 1    1    a
## 2    2    b
## 3    3    c
## 4    4    d
## 5    5    e
```

- Subset a **specific row (observation)**:

```
df[2,]
```

```
##   Nums Lets
## 2    2    b
```

- Subset a **specific column (variable)**:

```
df[,2]
```

```
## [1] "a" "b" "c" "d" "e"
```

- Subset a **specific value**:

```
df[2,2]
```

```
## [1] "b"
```

Subsetting Data: Data Frames II

- The nice thing about data frames is that instead of remembering the order of columns, we have the **`names`** of columns

```
df
```

```
##   Nums Lets
## 1    1    a
## 2    2    b
## 3    3    c
## 4    4    d
## 5    5    e
```

```
names(df)
```

```
## [1] ”Nums” ”Lets”
```

Subsetting Data: Data Frames II

- The nice thing about data frames is that instead of remembering the order of columns, we have the **`names`** of columns

```
df
```

```
##   Nums Lets
## 1    1    a
## 2    2    b
## 3    3    c
## 4    4    d
## 5    5    e
```

```
names(df)
```

```
## [1] ”Nums” ”Lets”
```

Subsetting Conditionally

- We often want to subset a `data.frame` based on a condition

Subsetting Conditionally

- We often want to subset a `data.frame` based on a condition
    - e.g. look only at **observations for which `Nums` are larger than 2**

Subsetting Conditionally

- We often want to subset a `data.frame` based on a condition
    - e.g. look only at **observations for which `Nums` are larger than 2**
- Can use original brackets `[ ]` to pick by **rows** (observations) for which `Num>2`

Subsetting Conditionally

- We often want to subset a `data.frame` based on a condition
    - e.g. look only at **observations for which `Nums` are larger than 2**
- Can use original brackets `[ ]` to pick by **rows** (observations) for which `Num>2`

- If we want **all** columns (variables)

```
df[df$Nums>2,]
```

```
##   Nums Lets
## 3    3    c
## 4    4    d
## 5    5    e
```

Subsetting Conditionally

- We often want to subset a `data.frame` based on a condition
  - e.g. look only at **observations for which `Nums` are larger than 2**
- Can use original brackets `[]` to pick by **rows** (observations) for which `Num>2`

- If we want **all** columns (variables)

```
df[df$Nums>2,]
```

```
##   Nums Lets
## 3    3    c
## 4    4    d
## 5    5    e
```

- If we only want column 1 ("`Nums`")

```
df[df$Nums>2,1]
```

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

Subsetting Conditionally

- We often want to subset a `data.frame` based on a condition
  - e.g. look only at **observations for which `Nums` are larger than 2**
- Can use original brackets `[]` to pick by **rows** (observations) for which `Num>2`

- If we want **all** columns (variables)

```
df[df$Nums>2,]
```

```
##   Nums Lets
## 3    3    c
## 4    4    d
## 5    5    e
```

- If we only want column 1 (`”Nums”`)

```
df[df$Nums>2,1]
```

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

- If we only want column 2 (`”Lets”`)

```
df[df$Nums>2,2]
```

```
## [1] ”c” ”d” ”e”
```

HOOD COLLEGE

Subsetting Conditionally: `subset()`

- One faster way that gets us away from `[]` is `subset(df, condition)`

Subsetting Conditionally: `subset()`

- One faster way that gets us away from `[]` is `subset(df, condition)`
  - Keeps only values of `df` for which condition is `TRUE`

Subsetting Conditionally: `subset()`

- One faster way that gets us away from `[]` is `subset(df, condition)`
    - Keeps only values of `df` for which condition is `TRUE`

```
subset(df, Nums>2)
```

```
##   Nums Lets
## 3    3    c
## 4    4    d
## 5    5    e
```

Subsetting Conditionally: `filter()` with `dplyr`

- `dplyr` makes this easier with `filter()`

```r
df %>%
  filter(Nums>2)
```

```
##   Nums Lets
## 1    3    c
## 2    4    d
## 3    5    e
```

Useful Conditionals

Condition	Description	Example(s)
`>`	Values greater than	`Num>2`
`>=`	Values greater than or equal to	`Num>=2`
`==`	Values equal to (put value in quotes if a character)	`Num==2`; `Let==”a”`
`!=`	Values are NOT equal to	`Num!=2`; `Let!=”a”`
`cond.1 & cond.2`	“AND”: BOTH conditions must be met	`Num>2 & Num<5`
`cond.1 | cond.2`	“OR”: Either one condition must be met	`Num>2 | Num<5`
`%in% c()`	Values are in a set of values defined in `c()`	`Num %in% c(1,2,3)`
`!%in% c()`	Values are NOT in defined set	`Num !%in% c(1,2,3)`

Dealing with Missing Data

Missing Data: NA

- If any observation is missing a value of a variable, it will show up as NA

```
x<-c(1,2,NA,4,5)
y<-c(”a”,NA,”c”,”d”,”e”)
df<-data.frame(x,y)

df
```

```
##     x    y
## 1  1    a
## 2  2 <NA>
## 3 NA    c
## 4  4    d
## 5  5    e
```

NAs Propagate...

- Missing data propagates and will ruin many functions you run on it

```
mean(df$x)
```

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

```
sd(df$x)
```

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

```
sum(df$x)
```

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

NA Strategy

- Several strategies to combat NAs

```
# with base R

df1<-df[!is.na(df$x),] # drop all observations for which there is NA for x
df1
```

```
##   x    y
## 1 1    a
## 2 2 <NA>
## 4 4    d
## 5 5    e
```

NA Strategy

- Several strategies to combat NAs

1. If looking at one variable:

```
# with base R

df1<-df[!is.na(df$x),] # drop all observations for which there is NA for x
df1
```

```
##   x    y
## 1 1    a
## 2 2 <NA>
## 4 4    d
## 5 5    e
```

NA Strategy

- Several strategies to combat NAs

1. If looking at one variable:
    - Keep only observations for which there are no NAs

```
# with base R

df1<-df[!is.na(df$x),] # drop all observations for which there is NA for x
df1
```

```
##   x    y
## 1 1    a
## 2 2 <NA>
## 4 4    d
## 5 5    e
```

NA Strategy

- Several strategies to combat NAs

1. If looking at one variable:
    - Keep only observations for which there are no NAs

```
# with base R

df1<-df[!is.na(df$x),] # drop all observations for which there is NA for x
df1
```

```
##   x    y
## 1 1    a
## 2 2 <NA>
## 4 4    d
## 5 5    e
```

NA Strategy II

2. Drop *all* observations that have some missing value across *any* variable with `na.omit(df)`

```
df2<-na.omit(df) # drop any row that has any NA value for any variable
df2
```

```
##   x y
## 1 1 a
## 4 4 d
## 5 5 e
```

NA Strategy II

2. Drop *all* observations that have some missing value across *any* variable with `na.omit(df)`
    - Often too extreme, may end up throwing out a lot of useful data!

```r
df2<-na.omit(df) # drop any row that has any NA value for any variable
df2
```

```
##   x y
## 1 1 a
## 4 4 d
## 5 5 e
```

NA Strategy III

3. Most functions have a NA option built in

```
mean(df$x, na.rm=TRUE)
```

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

```
sd(df$x, na.rm=TRUE)
```

```
## [1] 1.825742
```

```
sum(df$x, na.rm=TRUE)
```

```
## [1] 12
```

NA Strategy III

3. Most functions have a NA option built in
    - Add “,na.rm=TRUE” inside any function’s ( ) to simply *ignore* all observations with NAs

```
mean(df$x, na.rm=TRUE)
```

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

```
sd(df$x, na.rm=TRUE)
```

```
## [1] 1.825742
```

```
sum(df$x, na.rm=TRUE)
```

```
## [1] 12
```

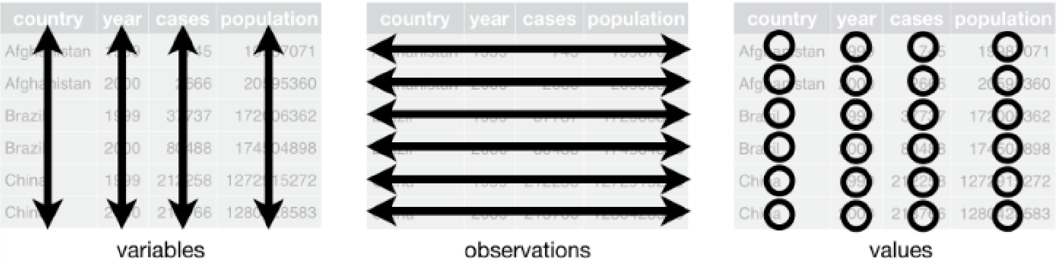
Merging Datasets

- `rbind()` adds observation(s)-(rows) for *all* existing variables (columns)

- **`rbind()`** adds observation(s)-(rows) for *all* existing variables (columns)
- **`cbind()`** adds variable(s)-(columns) for *all* existing observations (rows)

Tidy Data

Tidy Data


country
year
cases
population
variables
observations
values

Tidy Data

```
library(”knitr”)
FOTR<-read.csv(”../Data/The_Fellowship_Of_The_Ring.csv”)
TTT<-read.csv(”../Data/The_Two_Towers.csv”)
ROTK<-read.csv(”../Data/The_Return_Of_The_King.csv”)
```

Tidy Data II: Three Datasets

Film	Race	Female	Male
The Fellowship Of The Ring	Elf	1229	971
The Fellowship Of The Ring	Hobbit	14	3644
The Fellowship Of The Ring	Man	0	1995

Film	Race	Female	Male
The Two Towers	Elf	331	513
The Two Towers	Hobbit	0	2463
The Two Towers	Man	401	3589

Film	Race	Female	Male
The Return Of The King	Elf	183	510
The Return Of The King	Hobbit	2	2673
The Return Of The King	Man	268	2459

Tidying

```r
suppressPackageStartupMessages(library(”tidyverse”))
LOTR <- bind_rows(FOTR, TTT, ROTK)
```

```
## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector
```

Tidying II

```
str(LOTR)
```

```
## 'data.frame':    9 obs. of  4 variables:
##  $ Film  : chr  ”The Fellowship Of The Ring” ”The Fellowship Of The Ring” ”Th
##  $ Race  : Factor w/ 3 levels ”Elf”,”Hobbit”,..: 1 2 3 1 2 3 1 2 3
##  $ Female: int  1229 14 0 331 0 401 183 2 268
##  $ Male  : int  971 3644 1995 513 2463 3589 510 2673 2459
```

```
LOTR
```

```
##                         Film   Race Female Male
## 1 The Fellowship Of The Ring    Elf   1229  971
## 2 The Fellowship Of The Ring Hobbit     14 3644
## 3 The Fellowship Of The Ring    Man      0 1995
## 4             The Two Towers    Elf    331  513
## 5             The Two Towers Hobbit      0 2463
## 6             The Two Towers    Man    401 3589
## 7     The Return Of The King    Elf    183  510
## 8     The Return Of The King Hobbit      2 2673
## 9     The Return Of The King    Man    268 2459
```

Tidying III

```
LOTR_tidy <-
  gather(LOTR, key = 'Gender', value = 'Words', Female, Male)
LOTR_tidy
```

```
##                          Film   Race Gender Words
## 1  The Fellowship Of The Ring    Elf Female  1229
## 2  The Fellowship Of The Ring Hobbit Female    14
## 3  The Fellowship Of The Ring    Man Female     0
## 4              The Two Towers    Elf Female   331
## 5              The Two Towers Hobbit Female     0
## 6              The Two Towers    Man Female   401
## 7      The Return Of The King    Elf Female   183
## 8      The Return Of The King Hobbit Female     2
## 9      The Return Of The King    Man Female   268
## 10 The Fellowship Of The Ring    Elf   Male   971
## 11 The Fellowship Of The Ring Hobbit   Male  3644
## 12 The Fellowship Of The Ring    Man   Male  1995
## 13             The Two Towers    Elf   Male   513
## 14             The Two Towers Hobbit   Male  2463
## 15             The Two Towers    Man   Male  3589
```

Saving the Tidy Data

```
write.csv(LOTR_tidy,”../Data/LOTR_tidy.csv”)
```

Now We Can Work With This

- Tidy data works better for analysis

```
LOTR_tidy %>%
  count(Gender, Race, wt = Words)
```

```
## # A tibble: 6 x 3
##   Gender Race       n
##   <chr>  <fct>  <int>
## 1 Female Elf     1743
## 2 Female Hobbit    16
## 3 Female Man      669
## 4 Male   Elf     1994
## 5 Male   Hobbit  8780
## 6 Male   Man     8043
```

Now We Can Work With This

- Tidy data works better for analysis
- All `tidyverse` packages assume tidy data

```
LOTR_tidy %>%
  count(Gender, Race, wt = Words)
```

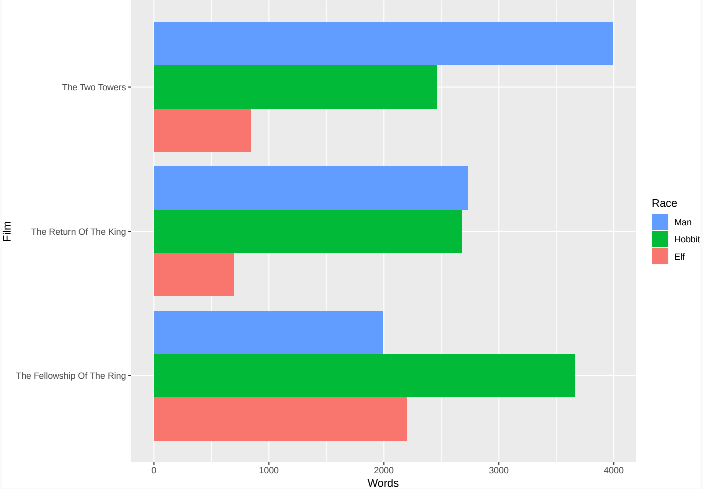
```
## # A tibble: 6 x 3
##   Gender Race       n
##   <chr>  <fct>  <int>
## 1 Female Elf     1743
## 2 Female Hobbit    16
## 3 Female Man      669
## 4 Male   Elf     1994
## 5 Male   Hobbit  8780
## 6 Male   Man     8043
```

```
by_race_film <- LOTR_tidy %>%
   group_by(Film, Race) %>%
   summarize(Words = sum(Words))

by_race_film
```

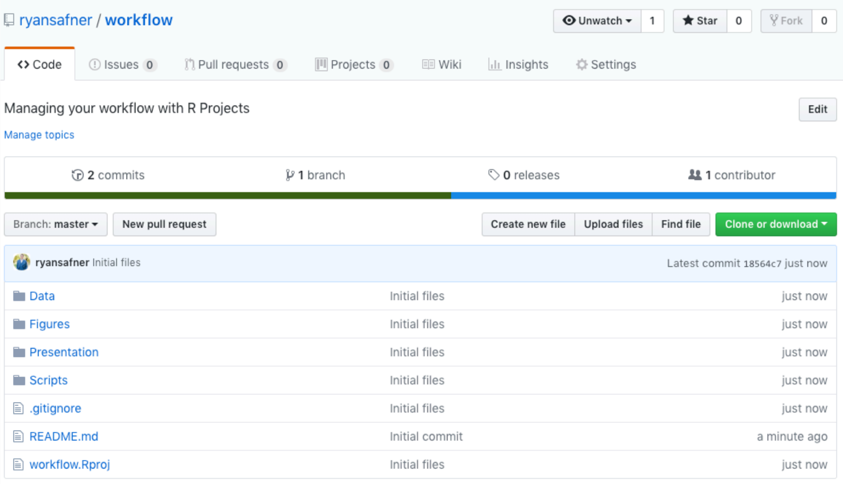
```
## # A tibble: 9 x 3
## # Groups:   Film [?]
##   Film                       Race   Words
##   <chr>                      <fct>  <int>
## 1 The Fellowship Of The Ring Elf     2200
## 2 The Fellowship Of The Ring Hobbit  3658
## 3 The Fellowship Of The Ring Man     1995
## 4 The Return Of The King     Elf      693
## 5 The Return Of The King     Hobbit  2675
## 6 The Return Of The King     Man     2727
```

```
p <- ggplot(by_race_film, aes(x = Film, y = Words, fill = Race))
p + geom_bar(stat = "identity", position = "dodge") +
  coord_flip() + guides(fill = guide_legend(reverse = TRUE))
```



Managing Your Workflow

USING RProj PROJECTS



ryansafner / workflow

Managing your workflow with R Projects

ryansafner Initial files Latest commit 18564c7 just now

Data	Initial files	just now
Figures	Initial files	just now
Presentation	Initial files	just now
Scripts	Initial files	just now
.gitignore	Initial files	just now
README.md	Initial commit	a minute ago
workflow.Rproj	Initial files	just now

Advanced: Automate All The Things!

Automate all the Things!

- You may find yourself in one of the following positions:

Automate all the Things!

- You may find yourself in one of the following positions:
    1. You have a very specific need that existing commands do not easily address

Automate all the Things!

- You may find yourself in one of the following positions:

    1. You have a very specific need that existing commands do not easily address
    2. You are running the same task many times on different objects

Automate all the Things!

- You may find yourself in one of the following positions:
  1. You have a very specific need that existing commands do not easily address
  2. You are running the same task many times on different objects
- If so, you can solve your needs with:

Automate all the Things!

- You may find yourself in one of the following positions:
  1. You have a very specific need that existing commands do not easily address
  2. You are running the same task many times on different objects
- If so, you can solve your needs with:
  1. Writing your own `R` `function`s

Automate all the Things!

- You may find yourself in one of the following positions:
    1. You have a very specific need that existing commands do not easily address
    2. You are running the same task many times on different objects
- If so, you can solve your needs with:
    1. Writing your own `R` `function`s
    2. Running a `for` loop

Automate all the Things!

- You may find yourself in one of the following positions:
    1. You have a very specific need that existing commands do not easily address
    2. You are running the same task many times on different objects
- If so, you can solve your needs with:
    1. Writing your own R `function`s
    2. Running a `for` loop
- Famous acronym in computer science: **DRY: Don't Repeat Yourself**

Automate all the Things!

- You may find yourself in one of the following positions:
  1. You have a very specific need that existing commands do not easily address
  2. You are running the same task many times on different objects
- If so, you can solve your needs with:
  1. Writing your own `R function`s
  2. Running a `for` loop
- Famous acronym in computer science: **DRY: Don't Repeat Yourself**
  - increases likelihood of error

Automate all the Things!

- You may find yourself in one of the following positions:

  1. You have a very specific need that existing commands do not easily address
  2. You are running the same task many times on different objects

- If so, you can solve your needs with:

  1. Writing your own `R function`s
  2. Running a `for` loop

- Famous acronym in computer science: **DRY: Don't Repeat Yourself**

  - increases likelihood of error
  - easier for readers to follow your intent

`functions` in R

- We've seen built in functions like `mean()`

```
my.function<-function(inputs){
  argument.using.inputs
}
```

functions in R

- We've seen built in functions like **mean()**
- You can write your own functions using the following syntax:

```r
my.function<-function(inputs){
  argument.using.inputs
}
```

`functions` in R II

- Let’s make our own `mean()` function called `my.mean()`:

```r
my.mean<-function(x){
  sum(x)/length(x)
}
```

`functions` in R III

- You can then use your function on any object

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

my.mean(a)
```

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

```
b<-c(2,4,6,8,10)

my.mean(b)
```

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

Another Simple Example

- You can even put another function as an input to a function

```
power<-function(exponent){
  function(x) x^exponent
}

# define other functions
square<-power(2)
cube<-power(3)

# run on examples

square(6)
```

```
## [1] 36
```

for Loops

- R will execute some statement **for** each **value** in a **sequence** of values:

```r
for (value in sequence){
  statement
}
```

forLOOPS

- Square all the numbers in the following vector `numbers.to.square`

```
numbers.to.square<-c(1,7.5,pi,3,-57,874,91)

for (i in numbers.to.square){
  print(i^2)
}
```

```
## [1] 1
## [1] 56.25
## [1] 9.869604
## [1] 9
## [1] 3249
## [1] 763876
## [1] 8281
```