R Workshop #1

Ryan Safner

11/27/2018

Subsetting Data

Dealing with Missing Data

Managing Your Workflow with `R Projects`

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 #]`

```r
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**:

```r
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]
```

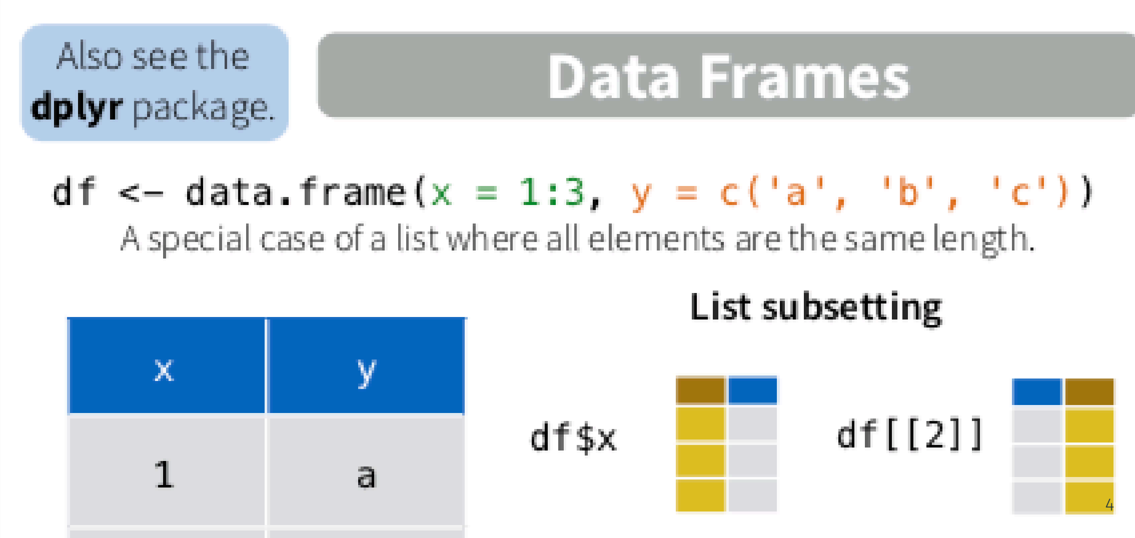
```
## [1] ”e” ”f”
```

- 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

List subsetting

`df$x`

`df[[2]]`

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)

```r
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"
```

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()`

```
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:

```r
# 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

```r
# 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 `NA`s

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

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

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

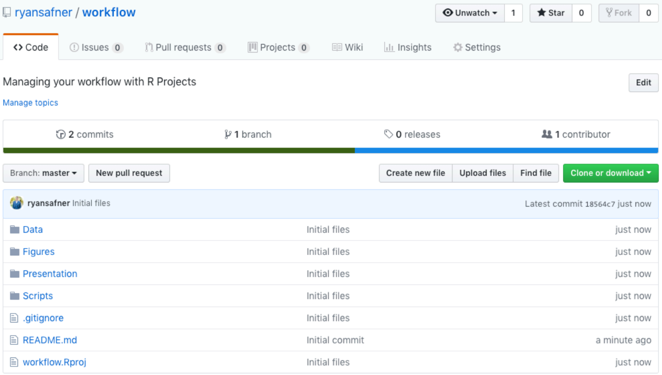
```
## [1] 1.825742
```

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

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

Managing Your Workflow with R `Projects`

Using `RProj` Projects



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

Go to github.com/ryansafner/workflow and follow the instructions!