Day 08: Data Wrangling (and more Cleaning)

Erin Rossiter

20 March, 2023

Announcements

Announcements

- PS07 due tomorrow
- PS08 posted tomorrow
  - will include some regex practice
  - second project update due 3/28 as well
- Midterm graded
  - good job!
- Final project
  - doesn't have to be in Rmd
  - other questions/comments?

Announcements

- PS07 due tomorrow
- PS08 posted tomorrow
  - will include some regex practice
  - second project update due 3/28 as well
- Midterm graded
  - good job!
- Final project
  - doesn't have to be in Rmd
  - other questions/comments?

Announcements

- PS07 due tomorrow
- PS08 posted tomorrow
  - will include some regex practice
  - second project update due 3/28 as well
- Midterm graded
  - good job!
- Final project
  - doesn't have to be in Rmd
  - other questions/comments?

Announcements

- PS07 due tomorrow
- PS08 posted tomorrow
  - will include some regex practice
  - second project update due 3/28 as well
- Midterm graded
  - good job!
- Final project
  - doesn't have to be in Rmd
  - other questions/comments?

Announcements

- PS07 due tomorrow
- PS08 posted tomorrow
  - will include some regex practice
  - second project update due 3/28 as well
- Midterm graded
  - good job!
- Final project
  - doesn't have to be in Rmd
  - other questions/comments?

Announcements

- PS07 due tomorrow
- PS08 posted tomorrow
  - will include some regex practice
  - second project update due 3/28 as well
- Midterm graded
  - good job!
- Final project
  - doesn't have to be in Rmd
  - other questions/comments?

Announcements

- PS07 due tomorrow
- PS08 posted tomorrow
  - will include some regex practice
  - second project update due 3/28 as well
- Midterm graded
  - good job!
- Final project
  - doesn't have to be in Rmd
  - other questions/comments?

Announcements

- PS07 due tomorrow
- PS08 posted tomorrow
  - will include some regex practice
  - second project update due 3/28 as well
- Midterm graded
  - good job!
- Final project
  - doesn't have to be in Rmd
  - other questions/comments?

Announcements

- PS07 due tomorrow
- PS08 posted tomorrow
  - will include some regex practice
  - second project update due 3/28 as well
- Midterm graded
  - good job!
- Final project
  - doesn't have to be in Rmd
  - other questions/comments?

Looking ahead

Looking ahead

- Next week:
  - more data wrangling (it didn't all fit today)
  - R packages
- Advanced topics for 3 weeks:
  - data viz
  - advanced relational data (i.e., SQL)
  - interfacing with other languages
  - other ideas? remote computing?
- Final week: project "presentations"
  - i.e., show and tell
  - no slides needed (unless you want)
  - big scraping project? walk us through steps, show final dataset
  - learn a new package? show us how you get it started and one-two cool functions and their outputs
  - everyone has **5 minutes** to share, I will cut you off
  - then ~5 minutes for Q&A

Looking ahead

- Next week:
  - more data wrangling (it didn't all fit today)
  - R packages
- Advanced topics for 3 weeks:
  - data viz
  - advanced relational data (i.e., SQL)
  - interfacing with other languages
  - other ideas? remote computing?
- Final week: project "presentations"
  - i.e., show and tell
  - no slides needed (unless you want)
  - big scraping project? walk us through steps, show final dataset
  - learn a new package? show us how you get it started and one-two cool functions and their outputs
  - everyone has **5 minutes** to share, I will cut you off
  - then ~5 minutes for Q&A

Looking ahead

- Next week:
  - more data wrangling (it didn't all fit today)
  - R packages
- Advanced topics for 3 weeks:
  - data viz
  - advanced relational data (i.e., SQL)
  - interfacing with other languages
  - other ideas? remote computing?
- Final week: project "presentations"
  - i.e., show and tell
  - no slides needed (unless you want)
  - big scraping project? walk us through steps, show final dataset
  - learn a new package? show us how you get it started and one-two cool functions and their outputs
  - everyone has **5 minutes** to share, I will cut you off
  - then ~5 minutes for Q&A

Looking ahead

- Next week:
  - more data wrangling (it didn't all fit today)
  - R packages
- Advanced topics for 3 weeks:
  - data viz
  - advanced relational data (i.e., SQL)
  - interfacing with other languages
  - other ideas? remote computing?
- Final week: project "presentations"
  - i.e., show and tell
  - no slides needed (unless you want)
  - big scraping project? walk us through steps, show final dataset
  - learn a new package? show us how you get it started and one-two cool functions and their outputs
  - everyone has **5 minutes** to share, I will cut you off
  - then ~5 minutes for Q&A

Looking ahead

- Next week:
  - more data wrangling (it didn't all fit today)
  - R packages
- Advanced topics for 3 weeks:
  - data viz
  - advanced relational data (i.e., SQL)
  - interfacing with other languages
  - other ideas? remote computing?
- Final week: project "presentations"
  - i.e., show and tell
  - no slides needed (unless you want)
  - big scraping project? walk us through steps, show final dataset
  - learn a new package? show us how you get it started and one-two cool functions and their outputs
  - everyone has **5 minutes** to share, I will cut you off
  - then ~5 minutes for Q&A

Looking ahead

- Next week:
  - more data wrangling (it didn't all fit today)
  - R packages
- Advanced topics for 3 weeks:
  - data viz
  - advanced relational data (i.e., SQL)
  - interfacing with other languages
  - other ideas? remote computing?
- Final week: project "presentations"
  - i.e., show and tell
  - no slides needed (unless you want)
  - big scraping project? walk us through steps, show final dataset
  - learn a new package? show us how you get it started and one-two cool functions and their outputs
  - everyone has **5 minutes** to share, I will cut you off
  - then ~5 minutes for Q&A

Looking ahead

- Next week:
  - more data wrangling (it didn't all fit today)
  - R packages
- Advanced topics for 3 weeks:
  - data viz
  - advanced relational data (i.e., SQL)
  - interfacing with other languages
  - other ideas? remote computing?
- Final week: project "presentations"
  - i.e., show and tell
  - no slides needed (unless you want)
  - big scraping project? walk us through steps, show final dataset
  - learn a new package? show us how you get it started and one-two cool functions and their outputs
  - everyone has **5 minutes** to share, I will cut you off
  - then ~5 minutes for Q&A

Looking ahead

- Next week:
  - more data wrangling (it didn't all fit today)
  - R packages
- Advanced topics for 3 weeks:
  - data viz
  - advanced relational data (i.e., SQL)
  - interfacing with other languages
  - other ideas? remote computing?
- Final week: project "presentations"
  - i.e., show and tell
  - no slides needed (unless you want)
  - big scraping project? walk us through steps, show final dataset
  - learn a new package? show us how you get it started and one-two cool functions and their outputs
  - everyone has **5 minutes** to share, I will cut you off
  - then ~5 minutes for Q&A

Looking ahead

- Next week:
  - more data wrangling (it didn't all fit today)
  - R packages
- Advanced topics for 3 weeks:
  - data viz
  - advanced relational data (i.e., SQL)
  - interfacing with other languages
  - other ideas? remote computing?
- Final week: project "presentations"
  - i.e., show and tell
  - no slides needed (unless you want)
  - big scraping project? walk us through steps, show final dataset
  - learn a new package? show us how you get it started and one-two cool functions and their outputs
  - everyone has **5 minutes** to share, I will cut you off
  - then ~5 minutes for Q&A

Looking ahead

- Next week:
  - more data wrangling (it didn't all fit today)
  - R packages
- Advanced topics for 3 weeks:
  - data viz
  - advanced relational data (i.e., SQL)
  - interfacing with other languages
  - other ideas? remote computing?
- Final week: project "presentations"
  - i.e., show and tell
  - no slides needed (unless you want)
  - big scraping project? walk us through steps, show final dataset
  - learn a new package? show us how you get it started and one-two cool functions and their outputs
  - everyone has **5 minutes** to share, I will cut you off
  - then ~5 minutes for Q&A

Looking ahead

- Next week:
  - more data wrangling (it didn't all fit today)
  - R packages
- Advanced topics for 3 weeks:
  - data viz
  - advanced relational data (i.e., SQL)
  - interfacing with other languages
  - other ideas? remote computing?
- Final week: project "presentations"
  - i.e., show and tell
  - no slides needed (unless you want)
  - big scraping project? walk us through steps, show final dataset
  - learn a new package? show us how you get it started and one-two cool functions and their outputs
  - everyone has **5 minutes** to share, I will cut you off
  - then ~5 minutes for Q&A

Looking ahead

- Next week:
  - more data wrangling (it didn't all fit today)
  - R packages
- Advanced topics for 3 weeks:
  - data viz
  - advanced relational data (i.e., SQL)
  - interfacing with other languages
  - other ideas? remote computing?
- Final week: project "presentations"
  - i.e., show and tell
  - no slides needed (unless you want)
  - big scraping project? walk us through steps, show final dataset
  - learn a new package? show us how you get it started and one-two cool functions and their outputs
  - everyone has **5 minutes** to share, I will cut you off
  - then ~5 minutes for Q&A

Looking ahead

- Next week:
  - more data wrangling (it didn't all fit today)
  - R packages
- Advanced topics for 3 weeks:
  - data viz
  - advanced relational data (i.e., SQL)
  - interfacing with other languages
  - other ideas? remote computing?
- Final week: project "presentations"
  - i.e., show and tell
  - no slides needed (unless you want)
  - big scraping project? walk us through steps, show final dataset
  - learn a new package? show us how you get it started and one-two cool functions and their outputs
  - everyone has **5 minutes** to share, I will cut you off
  - then ~5 minutes for Q&A

Looking ahead

- Next week:
  - more data wrangling (it didn't all fit today)
  - R packages
- Advanced topics for 3 weeks:
  - data viz
  - advanced relational data (i.e., SQL)
  - interfacing with other languages
  - other ideas? remote computing?
- Final week: project "presentations"
  - i.e., show and tell
  - no slides needed (unless you want)
  - big scraping project? walk us through steps, show final dataset
  - learn a new package? show us how you get it started and one-two cool functions and their outputs
  - everyone has **5 minutes** to share, I will cut you off
  - then ~5 minutes for Q&A

Looking ahead

- Next week:
  - more data wrangling (it didn't all fit today)
  - R packages
- Advanced topics for 3 weeks:
  - data viz
  - advanced relational data (i.e., SQL)
  - interfacing with other languages
  - other ideas? remote computing?
- Final week: project "presentations"
  - i.e., show and tell
  - no slides needed (unless you want)
  - big scraping project? walk us through steps, show final dataset
  - learn a new package? show us how you get it started and one-two cool functions and their outputs
  - everyone has **5 minutes** to share, I will cut you off
  - then ~5 minutes for Q&A

Today's plan

Today's plan

1. `apply` family
   - basic intuition
   - *some* of the variations
2. `plyr` package
   - basics
   - easy parallelization
3. Data wrangling with dplyr and tidyr
   - recoding data
   - subsetting
   - grouping

Today's plan

1. `apply` family
   - basic intuition
   - *some* of the variations
2. `plyr` package
   - basics
   - easy parallelization
3. Data wrangling with dplyr and tidyr
   - recoding data
   - subsetting
   - grouping

Today's plan

1. `apply` family
   - basic intuition
   - *some* of the variations
2. `plyr` package
   - basics
   - easy parallelization
3. Data wrangling with dplyr and tidyr
   - recoding data
   - subsetting
   - grouping

Today's plan

1. `apply` family
   - basic intuition
   - *some* of the variations
2. `plyr` package
   - basics
   - easy parallelization
3. Data wrangling with dplyr and tidyr
   - recoding data
   - subsetting
   - grouping

Today's plan

1. `apply` family
   - basic intuition
   - *some* of the variations
2. `plyr` package
   - basics
   - easy parallelization
3. Data wrangling with dplyr and tidyr
   - recoding data
   - subsetting
   - grouping

Today's plan

1. `apply` family
   - basic intuition
   - *some* of the variations
2. `plyr` package
   - basics
   - easy parallelization
3. Data wrangling with dplyr and tidyr
   - recoding data
   - subsetting
   - grouping

Today's plan

1. `apply` family
   - basic intuition
   - *some* of the variations
2. `plyr` package
   - basics
   - easy parallelization
3. Data wrangling with dplyr and tidyr
   - recoding data
   - subsetting
   - grouping

Today's plan

1. `apply` family
   - basic intuition
   - *some* of the variations
2. `plyr` package
   - basics
   - easy parallelization
3. Data wrangling with dplyr and tidyr
   - recoding data
   - subsetting
   - grouping

Today's plan

1. `apply` family
   - basic intuition
   - *some* of the variations
2. `plyr` package
   - basics
   - easy parallelization
3. Data wrangling with dplyr and tidyr
   - recoding data
   - subsetting
   - grouping

Today's plan

1. `apply` family
   - basic intuition
   - *some* of the variations
2. `plyr` package
   - basics
   - easy parallelization
3. Data wrangling with dplyr and tidyr
   - recoding data
   - subsetting
   - grouping

Base R ‘apply’ family

What does apply do?

- `apply()` is a fundamental function
  - for repetitive tasks
  - a loop substitute!
- it (and friends) are in base R and have sad documentation
- (better versions later today)
- yet you should know both so you can read others' code!

What does apply do?

- `apply()` is a fundamental function
  - for repetitive tasks
  - a loop substitute!
- it (and friends) are in base R and have sad documentation
- (better versions later today)
- yet you should know both so you can read others' code!

What does apply do?

- `apply()` is a fundamental function
  - for repetitive tasks
  - a loop substitute!
- it (and friends) are in base R and have sad documentation
- (better versions later today)
- yet you should know both so you can read others' code!

What does apply do?

- `apply()` is a fundamental function
  - for repetitive tasks
  - a loop substitute!
- it (and friends) are in base R and have sad documentation
- (better versions later today)
- yet you should know both so you can read others' code!

What does apply do?

- `apply()` is a fundamental function
  - for repetitive tasks
  - a loop substitute!
- it (and friends) are in base R and have sad documentation
- (better versions later today)
- yet you should know both so you can read others' code!

What does apply do?

- `apply()` is a fundamental function
  - for repetitive tasks
  - a loop substitute!
- it (and friends) are in base R and have sad documentation
- (better versions later today)
- yet you should know both so you can read others' code!

What does apply do?

- `apply()` is a fundamental function
  - for repetitive tasks
  - a loop substitute!
- it (and friends) are in base R and have sad documentation
- (better versions later today)
- yet you should know both so you can read others' code!

Basics of `apply`

The basic structure is like this (from help file)

```
apply(X, MARGIN, FUN, ...)
```

- `X` is an "array," so usually **matrices** (a 2-dimensional array)
- `MARGIN` controls how the matrix is analyzed. Should the function be executed on each **row** (margin=1) or each **column** (margin=2)?
- `FUN` is the **function** you want done on each row/column/whatever.
  - » *functions are objects, so they can be passed as arguments*
- `...` refers to any argument you want to pass onto `FUN`

Basics of `apply`

The basic structure is like this (from help file)

```
apply(X, MARGIN, FUN, ...)
```

- `X` is an "array," so usually **matrices** (a 2-dimensional array)
- `MARGIN` controls how the matrix is analyzed. Should the function be executed on each **row** (margin=1) or each **column** (margin=2)?
- `FUN` is the **function** you want done on each row/column/whatever.
  - » *functions are objects, so they can be passed as arguments*
- `...` refers to any argument you want to pass onto `FUN`

Basics of `apply`

The basic structure is like this (from help file)

```
apply(X, MARGIN, FUN, ...)
```

- `X` is an "array," so usually **matrices** (a 2-dimensional array)
- `MARGIN` controls how the matrix is analyzed. Should the function be executed on each **row** (margin=1) or each **column** (margin=2)?
- `FUN` is the **function** you want done on each row/column/whatever.
  - » *functions are objects, so they can be passed as arguments*
- `...` refers to any argument you want to pass onto `FUN`

Basics of `apply`

The basic structure is like this (from help file)

```
apply(X, MARGIN, FUN, ...)
```

- `X` is an "array," so usually **matrices** (a 2-dimensional array)
- `MARGIN` controls how the matrix is analyzed. Should the function be executed on each **row** (margin=1) or each **column** (margin=2)?
- `FUN` is the **function** you want done on each row/column/whatever.
  - » *functions are objects, so they can be passed as arguments*
- `...` refers to any argument you want to pass onto `FUN`

Basics of `apply`

The basic structure is like this (from help file)

```
apply(X, MARGIN, FUN, ...)
```

- `X` is an "array," so usually **matrices** (a 2-dimensional array)
- `MARGIN` controls how the matrix is analyzed. Should the function be executed on each **row** (margin=1) or each **column** (margin=2)?
- `FUN` is the **function** you want done on each row/column/whatever.
  - » *functions are objects, so they can be passed as arguments*
- `...` refers to any argument you want to pass onto `FUN`

Basics of `apply`

The basic structure is like this (from help file)

```
apply(X, MARGIN, FUN, ...)
```

- `X` is an "array," so usually **matrices** (a 2-dimensional array)
- `MARGIN` controls how the matrix is analyzed. Should the function be executed on each **row** (margin=1) or each **column** (margin=2)?
- `FUN` is the **function** you want done on each row/column/whatever.
  - » *functions are objects, so they can be passed as arguments*
- `...` refers to any argument you want to pass onto `FUN`

Example

Thing to remember: you are passing a function (ex: `sum`) as an argument to the `apply` function.

```
ex_mat <- matrix(rep(1:3, 3), ncol = 3)
ex_mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    2    2    2
## [3,]    3    3    3
```

```
# Sum each row
apply(ex_mat, 1, sum)
```

```
## [1] 3 6 9
```

```
# Sum each column
apply(ex_mat, 2, sum)
```

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

Example

Thing to remember: you are passing a function (ex: `sum`) as an argument to the `apply` function.

```
ex_mat <- matrix(rep(1:3, 3), ncol = 3)
ex_mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    2    2    2
## [3,]    3    3    3
```

```
# Sum each row
apply(ex_mat, 1, sum)
```

```
## [1] 3 6 9
```

```
# Sum each column
apply(ex_mat, 2, sum)
```

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

Variations of `apply`

Base R comes with many, many functions that have similar behavior. We'll cover two more:

- `lapply` and `sapply` (applies function to each element of a lists)
- `tapply` (applies function to each level of a factor)

There are others: `sweep`, `by`, `vapply`, `mapply`, `rapply`, `replicate`, `eapply`, `aggregate`, and more.

They all take in functions as arguments to, somehow, "apply" over your data, it just depends on what your data looks like.

Variations of `apply`

Base R comes with many, many functions that have similar behavior. We'll cover two more:

- `lapply` and `sapply` (applies function to each element of a lists)
- `tapply` (applies function to each level of a factor)

There are others: `sweep`, `by`, `vapply`, `mapply`, `rapply`, `replicate`, `eapply`, `aggregate`, and more.

They all take in functions as arguments to, somehow, "apply" over your data, it just depends on what your data looks like.

Variations of `apply`

Base R comes with many, many functions that have similar behavior. We'll cover two more:

- `lapply` and `sapply` (applies function to each element of a lists)
- `tapply` (applies function to each level of a factor)

There are others: `sweep`, `by`, `vapply`, `mapply`, `rapply`, `replicate`, `eapply`, `aggregate`, and more.

They all take in functions as arguments to, somehow, "apply" over your data, it just depends on what your data looks like.

Variations of `apply`

Base R comes with many, many functions that have similar behavior. We'll cover two more:

- `lapply` and `sapply` (applies function to each element of a lists)
- `tapply` (applies function to each level of a factor)

There are others: `sweep`, `by`, `vapply`, `mapply`, `rapply`, `replicate`, `eapply`, `aggregate`, and more.

They all take in functions as arguments to, somehow, "apply" over your data, it just depends on what your data looks like.

Variations of `apply`

Base R comes with many, many functions that have similar behavior. We'll cover two more:

- `lapply` and `sapply` (applies function to each element of a lists)
- `tapply` (applies function to each level of a factor)

There are others: `sweep`, `by`, `vapply`, `mapply`, `rapply`, `replicate`, `eapply`, `aggregate`, and more.

They all take in functions as arguments to, somehow, "apply" over your data, it just depends on what your data looks like.

`lapply()`

`lapply` is when your data input is a `list` or `data.frame` (recall they are like lists)

```
lapply(X, FUN, ...)
```

`lapply()`

`lapply` is when your data input is a `list` or `data.frame` (recall they are like lists)

```
lapply(X, FUN, ...)
```

Example

```
ex_list <- list(classname = "Programming",
                names = c("Erin", "Adriana", "Emily", "..."),
                meetings = paste0("class", 1:13))
lapply(X = ex_list, FUN = length) # length of each  element
```

```
## $classname
## [1] 1
##
## $names
## [1] 4
##
## $meetings
## [1] 13
```

Example

```
lapply(X = ex_list, FUN = is.character)
```

```
## $classname
## [1] TRUE
##
## $names
## [1] TRUE
##
## $meetings
## [1] TRUE
```

Understanding the output

```
output <- lapply(ex_list, length)
output
```

```
## $classname
## [1] 1
##
## $names
## [1] 4
##
## $meetings
## [1] 13
```

```
is.list(output) #output is a list
```

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

```
output_vec <- unlist(output) #need to unlist
output_vec
```

```
## classname     names  meetings
##         1         4        13
```

sapply()

sapply is a "simplified" version of lapply where it simplifies the outcome

```
output2 <- sapply(X = ex_list, FUN = length)

# unlists for you if it can
output2
```

```
## classname      names  meetings
##         1          4         13
```

```
# Not a list
is.vector(output2)
```

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

```
is.list(output2)
```

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

sapply()

sapply is a "simplified" version of lapply where it simplifies the outcome

```
output2 <- sapply(X = ex_list, FUN = length)

# unlists for you if it can
output2
```

```
## classname     names  meetings
##         1         4        13
```

```
# Not a list
is.vector(output2)
```

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

```
is.list(output2)
```

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

sapply() output

1. If the output is all single elements, the results will be a vector
2. If the output is all vectors of the same length, it will return a matrix.

```
ex_list2 <- list(x = sample(1:100, 10),
                 y = sample(1:100, 100))
sapply(ex_list2, summary)
```

```
##                x       y
## Min.     1.0   1.00
## 1st Qu. 17.5  25.75
## Median  41.5  50.50
## Mean    48.9  50.50
## 3rd Qu. 84.0  75.25
## Max.    98.0 100.00
```

sapply() output

1. If the output is all single elements, the results will be a vector
2. If the output is all vectors of the same length, it will return a matrix.

```
ex_list2 <- list(x = sample(1:100, 10),
                 y = sample(1:100, 100))
sapply(ex_list2, summary)
```

```
##                x       y
## Min.     1.0   1.00
## 1st Qu. 17.5  25.75
## Median  41.5  50.50
## Mean    48.9  50.50
## 3rd Qu. 84.0  75.25
## Max.    98.0 100.00
```

sapply() output

1. If the output is all single elements, the results will be a vector
2. If the output is all vectors of the same length, it will return a matrix.

```
ex_list2 <- list(x = sample(1:100, 10),
                 y = sample(1:100, 100))
sapply(ex_list2, summary)
```

```
##                x       y
## Min.      1.0   1.00
## 1st Qu. 17.5  25.75
## Median  41.5  50.50
## Mean    48.9  50.50
## 3rd Qu. 84.0  75.25
## Max.    98.0 100.00
```

tapply

```
tapply(X, INDEX, FUN, ...)
```

- useful for recoding tasks, summarizing data, etc.
- key is to understand that the "indices" are the **values of some other object** (usually a variable in your data)

```
data(cars)
# at its core, its just a table function
table(cars$speed)
```

```
##
##  4  7  8  9 10 11 12 13 14 15 16 17 18 19 20 22 23 24 25
##  2  2  1  1  3  2  4  4  4  3  2  3  4  3  5  1  1  4  1
```

```
tapply(cars$dist, cars$speed, FUN = length)
```

```
##  4  7  8  9 10 11 12 13 14 15 16 17 18 19 20 22 23 24 25
##  2  2  1  1  3  2  4  4  4  3  2  3  4  3  5  1  1  4  1
```

tapply

```
tapply(X, INDEX, FUN, ...)
```

- useful for recoding tasks, summarizing data, etc.
- key is to understand that the "indices" are the **values of some other object** (usually a variable in your data)

```
data(cars)
# at its core, its just a table function
table(cars$speed)
```

```
##
##  4  7  8  9 10 11 12 13 14 15 16 17 18 19 20 22 23 24 25
##  2  2  1  1  3  2  4  4  4  3  2  3  4  3  5  1  1  4  1
```

```
tapply(cars$dist, cars$speed, FUN = length)
```

```
##  4  7  8  9 10 11 12 13 14 15 16 17 18 19 20 22 23 24 25
##  2  2  1  1  3  2  4  4  4  3  2  3  4  3  5  1  1  4  1
```

tapply

```
tapply(X, INDEX, FUN, ...)
```

- useful for recoding tasks, summarizing data, etc.
- key is to understand that the "indices" are the **values of some other object** (usually a variable in your data)

```
data(cars)
# at its core, its just a table function
table(cars$speed)
```

```
##
##  4  7  8  9 10 11 12 13 14 15 16 17 18 19 20 22 23 24 25
##  2  2  1  1  3  2  4  4  4  3  2  3  4  3  5  1  1  4  1
```

```
tapply(cars$dist, cars$speed, FUN = length)
```

```
##  4  7  8  9 10 11 12 13 14 15 16 17 18 19 20 22 23 24 25
##  2  2  1  1  3  2  4  4  4  3  2  3  4  3  5  1  1  4  1
```

Another example

```
# we can apply functions to splices of the data
tapply(cars$dist, cars$speed, FUN = mean)
```

```
##        4        7        8        9       10       11        1
##  6.00000 13.00000 16.00000 10.00000 26.00000 22.50000 21.5000
##       14       15       16       17       18       19        2
## 50.50000 33.33333 36.00000 40.66667 64.50000 50.00000 50.4000
##       23       24       25
## 54.00000 93.75000 85.00000
```

plyr

The `plyr` package

- Base R apply is all quite confusing even to people who have done this a while
- The function names are not intuitive and arguments and outputs change weirdly between functions
- `plyr` package saves the day

```
library(plyr)
```

(Note: Borrowing heavily from:
http://www.r-bloggers.com/a-fast-intro-to-plyr-for-r)

The `plyr` package

- Base R apply is all quite confusing even to people who have done this a while
- The function names are not intuitive and arguments and outputs change weirdly between functions
- `plyr` package saves the day

```
library(plyr)
```

(Note: Borrowing heavily from:
http://www.r-bloggers.com/a-fast-intro-to-plyr-for-r)

The `plyr` package

- Base R apply is all quite confusing even to people who have done this a while
- The function names are not intuitive and arguments and outputs change weirdly between functions
- `plyr` package saves the day

```
library(plyr)
```

(Note: Borrowing heavily from:
http://www.r-bloggers.com/a-fast-intro-to-plyr-for-r)

plyr features

plyr makes things a bit easier by adding the following features:

1. Consistent naming protocols for the functions to know what you are putting in and taking out.
2. Easy to make parallel.
3. Built-in error recovery
4. Flexible and fairly intuitive handling of all basic data types.

plyr features

plyr makes things a bit easier by adding the following features:

1. Consistent naming protocols for the functions to know what you are putting in and taking out.
2. Easy to make parallel.
3. Built-in error recovery
4. Flexible and fairly intuitive handling of all basic data types.

plyr features

plyr makes things a bit easier by adding the following features:

1. Consistent naming protocols for the functions to know what you are putting in and taking out.
2. Easy to make parallel.
3. Built-in error recovery
4. Flexible and fairly intuitive handling of all basic data types.

plyr features

plyr makes things a bit easier by adding the following features:

1. Consistent naming protocols for the functions to know what you are putting in and taking out.
2. Easy to make parallel.
3. Built-in error recovery
4. Flexible and fairly intuitive handling of all basic data types.

plyr features

plyr makes things a bit easier by adding the following features:

1. Consistent naming protocols for the functions to know what you are putting in and taking out.
2. Easy to make parallel.
3. Built-in error recovery
4. Flexible and fairly intuitive handling of all basic data types.

plyr features

plyr makes things a bit easier by adding the following features:

1. Consistent naming protocols for the functions to know what you are putting in and taking out.
2. Easy to make parallel.
3. Built-in error recovery
4. Flexible and fairly intuitive handling of all basic data types.

Main functions

The main functions we want to use are: `a_ply`, `aaply`, `adply`, `alply`, `d_ply`, `daply`, `ddply`, `dlply`, `l_ply`, `laply`, `llply`, `m_ply`, `maply`, `mdply`, `mlply`

The first letter in each tells us what kind of input we are taking

- a=array
- d=data.frame
- l=list
- m=matrix

The second letter tells us what we want output

- a=array
- d=data.frame
- l=list
- m=matrix
- _=discard the results

Other than these letters, they all work basically the same.

Main functions

The main functions we want to use are: `a_ply`, `aaply`, `adply`, `alply`, `d_ply`, `daply`, `ddply`, `dlply`, `l_ply`, `laply`, `llply`, `m_ply`, `maply`, `mdply`, `mlply`

The first letter in each tells us what kind of input we are taking

- a=array
- d=data.frame
- l=list
- m=matrix

The second letter tells us what we want output

- a=array
- d=data.frame
- l=list
- m=matrix
- _=discard the results

Other than these letters, they all work basically the same.

Main functions

The main functions we want to use are: `a_ply`, `aaply`, `adply`, `alply`, `d_ply`, `daply`, `ddply`, `dlply`, `l_ply`, `laply`, `llply`, `m_ply`, `maply`, `mdply`, `mlply`

The first letter in each tells us what kind of input we are taking

- a=array
- d=data.frame
- l=list
- m=matrix

The second letter tells us what we want output

- a=array
- d=data.frame
- l=list
- m=matrix
- _=discard the results

Other than these letters, they all work basically the same.

Main functions

The main functions we want to use are: `a_ply`, `aaply`, `adply`, `alply`, `d_ply`, `daply`, `ddply`, `dlply`, `l_ply`, `laply`, `llply`, `m_ply`, `maply`, `mdply`, `mlply`

The first letter in each tells us what kind of input we are taking

- a=array
- d=data.frame
- l=list
- m=matrix

The second letter tells us what we want output

- a=array
- d=data.frame
- l=list
- m=matrix
- _=discard the results

Other than these letters, they all work basically the same.

Main functions

The main functions we want to use are: `a_ply`, `aaply`, `adply`, `alply`, `d_ply`, `daply`, `ddply`, `dlply`, `l_ply`, `laply`, `llply`, `m_ply`, `maply`, `mdply`, `mlply`

The first letter in each tells us what kind of input we are taking

- a=array
- d=data.frame
- l=list
- m=matrix

The second letter tells us what we want output

- a=array
- d=data.frame
- l=list
- m=matrix
- _=discard the results

Other than these letters, they all work basically the same.

Example

Anything you want to do is basically already set up

```
x <-list(1, 2, 3, 4, 5)
x
```

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

```
identity(x) # just a function that returns the argument
```

The outputs

Returns nothing

```
l_ply(x, identity)
```

The outputs

Returns a list

```
llply(x, identity)
```

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

The outputs

Returns a dataframe

```
ldply(x, identity)
```

```
##   V1
## 1  1
## 2  2
## 3  3
## 4  4
## 5  5
```

day08-applyfamily.R

day08-parallelize.R

day08-inclass-apply.Rmd

dplyr + tidyr

dplyr + tidyr

- `dplyr` is a handy package for changing data
- `tidyr` is a handy package for reshaping data
- In combination they offer a powerful way to quickly extract insight from your data
- A new language on top of the base R we've been learning
  - Should be able to pick this up quick given you know the fundamental logic underlying it
- More advanced features next week!

dplyr + tidyr

- `dplyr` is a handy package for changing data
- `tidyr` is a handy package for reshaping data
- In combination they offer a powerful way to quickly extract insight from your data
- A new language on top of the base R we've been learning
  - Should be able to pick this up quick given you know the fundamental logic underlying it
- More advanced features next week!

dplyr + tidyr

- `dplyr` is a handy package for changing data
- `tidyr` is a handy package for reshaping data
- In combination they offer a powerful way to quickly extract insight from your data
- A new language on top of the base R we've been learning
  - Should be able to pick this up quick given you know the fundamental logic underlying it
- More advanced features next week!

dplyr + tidyr

- `dplyr` is a handy package for changing data
- `tidyr` is a handy package for reshaping data
- In combination they offer a powerful way to quickly extract insight from your data
- A new language on top of the base R we've been learning
  - Should be able to pick this up quick given you know the fundamental logic underlying it
- More advanced features next week!

dplyr + tidyr

- `dplyr` is a handy package for changing data
- `tidyr` is a handy package for reshaping data
- In combination they offer a powerful way to quickly extract insight from your data
- A new language on top of the base R we've been learning
  - » Should be able to pick this up quick given you know the fundamental logic underlying it
- More advanced features next week!

dplyr + tidyr

- `dplyr` is a handy package for changing data
- `tidyr` is a handy package for reshaping data
- In combination they offer a powerful way to quickly extract insight from your data
- A new language on top of the base R we've been learning
  - » Should be able to pick this up quick given you know the fundamental logic underlying it
- More advanced features next week!

Using `dplyr`

- Subset data by rows: `filter`
- Reorder by rows: `arrange`
- Subset data by column: `select`
- Create new variables as a function of other variables: `mutate`
- Collapse values down (or extract statistic): `summarise`
- We can use `group_by` to make changes in the scope

Using `dplyr`

- Subset data by rows: `filter`
- Reorder by rows: `arrange`
- Subset data by column: `select`
- Create new variables as a function of other variables: `mutate`
- Collapse values down (or extract statistic): `summarise`
- We can use `group_by` to make changes in the scope

Using `dplyr`

- Subset data by rows: `filter`
- Reorder by rows: `arrange`
- Subset data by column: `select`
- Create new variables as a function of other variables: `mutate`
- Collapse values down (or extract statistic): `summarise`
- We can use `group_by` to make changes in the scope

Using `dplyr`

- Subset data by rows: `filter`
- Reorder by rows: `arrange`
- Subset data by column: `select`
- Create new variables as a function of other variables: `mutate`
- Collapse values down (or extract statistic): `summarise`
- We can use `group_by` to make changes in the scope

Using `dplyr`

- Subset data by rows: `filter`
- Reorder by rows: `arrange`
- Subset data by column: `select`
- Create new variables as a function of other variables: `mutate`
- Collapse values down (or extract statistic): `summarise`
- We can use `group_by` to make changes in the scope

Using `dplyr`

- Subset data by rows: `filter`
- Reorder by rows: `arrange`
- Subset data by column: `select`
- Create new variables as a function of other variables: `mutate`
- Collapse values down (or extract statistic): `summarise`
- We can use `group_by` to make changes in the scope

Using `dplyr`

- Subset data by rows: `filter`
- Reorder by rows: `arrange`
- Subset data by column: `select`
- Create new variables as a function of other variables: `mutate`
- Collapse values down (or extract statistic): `summarise`
- We can use `group_by` to make changes in the scope

day08-dplyr-tidyr.R

Piping

Piping

- The tidyverse includes a nice syntax for combining multiple commands so we don't have to create new objects all of the time.
- The %>% syntax allows us to pass on the results of one line to another
  - We already used this without indepth discussion when we did webscraping

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summa
##     summarize

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Rows: 16661 Columns: 33
```

Another example

```
basicPolls %>%
  group_by(candidate_name, state) %>%
  summarise(average_candidate = mean(pct), count = n()) %>%
  filter(count>10)
```

```
## # A tibble: 206 x 4
## # Groups:   candidate_name [44]
##    candidate_name state          average_candidate count
##    <chr>          <chr>                      <dbl> <int>
##  1 Amy Klobuchar  California                  1.74    40
##  2 Amy Klobuchar  Florida                     1.98    13
##  3 Amy Klobuchar  Iowa                        5.97    61
##  4 Amy Klobuchar  Nevada                      1.71    15
##  5 Amy Klobuchar  New Hampshire               5.42    84
##  6 Amy Klobuchar  Pennsylvania                1.53    11
##  7 Amy Klobuchar  South Carolina              1.27    34
##  8 Amy Klobuchar  Texas                       1.47    23
##  9 Amy Klobuchar  Wisconsin                   2.52    16
## 10 Amy Klobuchar  <NA>                        1.62   503
## # ... with 196 more rows
```

Another example

```
primaryPolls %>%
  group_by(candidate_name, state) %>%
  summarise(average_candidate = mean(pct), count = n()) %>%
  filter(count > 10) %>%
  mutate(average_prop = average_candidate/100) %>%
  select(average_prop, candidate_name, state, count)
```

```
## # A tibble: 206 x 4
## # Groups:   candidate_name [44]
##    average_prop candidate_name state          count
##           <dbl> <chr>          <chr>          <int>
##  1       0.0174 Amy Klobuchar  California        40
##  2       0.0198 Amy Klobuchar  Florida           13
##  3       0.0597 Amy Klobuchar  Iowa              61
##  4       0.0171 Amy Klobuchar  Nevada            15
##  5       0.0542 Amy Klobuchar  New Hampshire     84
##  6       0.0153 Amy Klobuchar  Pennsylvania      11
##  7       0.0127 Amy Klobuchar  South Carolina    34
##  8       0.0147 Amy Klobuchar  Texas             23
##  9       0.0252 Amy Klobuchar  Wisconsin         16
## 10       0.0162 Amy Klobuchar  <NA>             503
```

In class activity day08-inclass-dplyr.pdf