

# Practical R: Data Ingestion and Munging

Abhijit Dasgupta

Fall, 2019

# Where we left off

We learned how to

1. separate columns into multiple columns based on patterns or index
2. gather many columns into 2 columns, with one column having the header information and the other having the values
3. spread two columns into many columns, with one column providing the new column headers and the other the values.

# Tidying the weather data

```
library(tidyverse)
weather_data <- rio::import('data/weather.csv')
```

```
#>      id year month element d1  d2  d3 d4  d5
#> 1 MX17004 2010     1    tmax NA   NA   NA NA   NA
#> 2 MX17004 2010     1    tmin NA   NA   NA NA   NA
#> 3 MX17004 2010     2    tmax NA 27.3 24.1 NA   NA
#> 4 MX17004 2010     2    tmin NA 14.4 14.4 NA   NA
#> 5 MX17004 2010     3    tmax NA   NA   NA NA 32.1
#> 6 MX17004 2010     3    tmin NA   NA   NA NA 14.2
#>      d12 d13 d14 d15 d16 d17 d18 d19 d20 d21 d22 d23
#> 1  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA
#> 2  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA
#> 3  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 29.1
#> 4  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 10.5
#> 5  NA  NA  NA  NA 31.1  NA  NA  NA  NA  NA  NA  NA
#> 6  NA  NA  NA  NA 17.6  NA  NA  NA  NA  NA  NA  NA
#>      d29 d30 d31
#> 1  NA 27.8  NA
#> 2  NA 14.5  NA
#> 3  NA  NA  NA
#> 4  NA  NA  NA
#> 5  NA  NA  NA
#> 6  NA  NA  NA
```

1. Days are in separate columns
2. Temperatures for each day is in two rows, max and min
3. Don't worry about missing values. Just work on getting the shape right

# Tidying the weather data

```
weather1 <- tidyr::gather(weather_data, day, temp, -id)
head(weather1, 20)
```

```
#>      id year month element day temp
#> 1  MX17004 2010     1    tmax d1   NA
#> 2  MX17004 2010     1    tmin d1   NA
#> 3  MX17004 2010     2    tmax d1   NA
#> 4  MX17004 2010     2    tmin d1   NA
#> 5  MX17004 2010     3    tmax d1   NA
#> 6  MX17004 2010     3    tmin d1   NA
#> 7  MX17004 2010     4    tmax d1   NA
#> 8  MX17004 2010     4    tmin d1   NA
#> 9  MX17004 2010     5    tmax d1   NA
#> 10 MX17004 2010     5    tmin d1   NA
#> 11 MX17004 2010     6    tmax d1   NA
#> 12 MX17004 2010     6    tmin d1   NA
#> 13 MX17004 2010     7    tmax d1   NA
#> 14 MX17004 2010     7    tmin d1   NA
#> 15 MX17004 2010     8    tmax d1   NA
#> 16 MX17004 2010     8    tmin d1   NA
#> 17 MX17004 2010    10    tmax d1   NA
#> 18 MX17004 2010    10    tmin d1   NA
#> 19 MX17004 2010    11    tmax d1   NA
#> 20 MX17004 2010    11    tmin d1   NA
```

# Tidying the weather data

```
weather1 <- tidyr::gather(weather_data, day, temp, -id)
weather2 <- spread(weather1, element, temp)
head(weather2, 20)
```

```
#>      id year month day tmax tmin
#> 1  MX17004 2010     1  d1    NA   NA
#> 2  MX17004 2010     1 d10    NA   NA
#> 3  MX17004 2010     1 d11    NA   NA
#> 4  MX17004 2010     1 d12    NA   NA
#> 5  MX17004 2010     1 d13    NA   NA
#> 6  MX17004 2010     1 d14    NA   NA
#> 7  MX17004 2010     1 d15    NA   NA
#> 8  MX17004 2010     1 d16    NA   NA
#> 9  MX17004 2010     1 d17    NA   NA
#> 10 MX17004 2010     1 d18    NA   NA
#> 11 MX17004 2010     1 d19    NA   NA
#> 12 MX17004 2010     1  d2    NA   NA
#> 13 MX17004 2010     1 d20    NA   NA
#> 14 MX17004 2010     1 d21    NA   NA
#> 15 MX17004 2010     1 d22    NA   NA
#> 16 MX17004 2010     1 d23    NA   NA
#> 17 MX17004 2010     1 d24    NA   NA
#> 18 MX17004 2010     1 d25    NA   NA
#> 19 MX17004 2010     1 d26    NA   NA
#> 20 MX17004 2010     1 d27    NA   NA
```

# Tidying the weather data

```
weather1 <- tidyr::gather(weather_data, day, temp, -C)
weather2 <- spread(weather1, element, temp)
weather3 <- separate(weather2, col='day', into=c('sym', 'day'))
head(weather3)
```

This gets us into the right shape for the data.

There still is some work to do, but the format is tidy

```
#>      id year month symbol day tmax tmin
#> 1 MX17004 2010     1     d   1    NA   NA
#> 2 MX17004 2010     1     d  10    NA   NA
#> 3 MX17004 2010     1     d  11    NA   NA
#> 4 MX17004 2010     1     d  12    NA   NA
#> 5 MX17004 2010     1     d  13    NA   NA
#> 6 MX17004 2010     1     d  14    NA   NA
```

# Data transformation (dplyr)

The dplyr package gives us a few verbs for data manipulation

| Function  | Purpose  |
|-----------|--|
| select    | Select columns based on name or position       |
| mutate    | Create or change a column                      |
| filter    | Extract rows based on some criteria            |
| arrange   | Re-order rows based on values of variable(s)   |
| group_by  | Split a dataset by unique values of a variable |
| summarize | Create summary statistics based on columns     |

# select

You can select columns by name or position, of course.

You can select consecutive columns using `:` notation, e.g. `select(weather, d1:d31)`

You can also select columns based on some criteria, which are encapsulated in functions.

- `starts_with("___"), ends_with("___"), contains("_____")`
- `one_of("_____", "_____", "_____")`

There are others; see `help(starts_with)`.

These selection methods work in all tidyverse functions

Note that for these functions, the names of the columns don't need to be quoted. This is called *non-standard evaluation* and is a convenience



# select

```
weather1 <- tidyr::gather(weather_data, day, temp, d1)
head(weather1, 20)
```

```
#>      id year month element day temp
#> 1  MX17004 2010     1    tmax d1   NA
#> 2  MX17004 2010     1    tmin d1   NA
#> 3  MX17004 2010     2    tmax d1   NA
#> 4  MX17004 2010     2    tmin d1   NA
#> 5  MX17004 2010     3    tmax d1   NA
#> 6  MX17004 2010     3    tmin d1   NA
#> 7  MX17004 2010     4    tmax d1   NA
#> 8  MX17004 2010     4    tmin d1   NA
#> 9  MX17004 2010     5    tmax d1   NA
#> 10 MX17004 2010     5    tmin d1   NA
#> 11 MX17004 2010     6    tmax d1   NA
#> 12 MX17004 2010     6    tmin d1   NA
#> 13 MX17004 2010     7    tmax d1   NA
#> 14 MX17004 2010     7    tmin d1   NA
#> 15 MX17004 2010     8    tmax d1   NA
#> 16 MX17004 2010     8    tmin d1   NA
#> 17 MX17004 2010    10    tmax d1   NA
#> 18 MX17004 2010    10    tmin d1   NA
#> 19 MX17004 2010    11    tmax d1   NA
#> 20 MX17004 2010    11    tmin d1   NA
```

# select

```
weather1 <- tidyr::gather(weather_data, key='day', va  
                           starts_with('d'))  
head(weather1, 20)
```

```
#>      id year month element day temp  
#> 1  MX17004 2010     1    tmax d1   NA  
#> 2  MX17004 2010     1    tmin d1   NA  
#> 3  MX17004 2010     2    tmax d1   NA  
#> 4  MX17004 2010     2    tmin d1   NA  
#> 5  MX17004 2010     3    tmax d1   NA  
#> 6  MX17004 2010     3    tmin d1   NA  
#> 7  MX17004 2010     4    tmax d1   NA  
#> 8  MX17004 2010     4    tmin d1   NA  
#> 9  MX17004 2010     5    tmax d1   NA  
#> 10 MX17004 2010     5    tmin d1   NA  
#> 11 MX17004 2010     6    tmax d1   NA  
#> 12 MX17004 2010     6    tmin d1   NA  
#> 13 MX17004 2010     7    tmax d1   NA  
#> 14 MX17004 2010     7    tmin d1   NA  
#> 15 MX17004 2010     8    tmax d1   NA  
#> 16 MX17004 2010     8    tmin d1   NA  
#> 17 MX17004 2010    10    tmax d1   NA  
#> 18 MX17004 2010    10    tmin d1   NA  
#> 19 MX17004 2010    11    tmax d1   NA  
#> 20 MX17004 2010    11    tmin d1   NA
```

# mutate

```
weather4 <- mutate(weather3,
  num_day = as.numeric(day))
as_tibble(weather4)
```

```
#> # A tibble: 341 x 8
#>   id      year month symbol day    tmax  tmin n
#>   <chr>   <int> <int> <chr> <chr> <dbl> <dbl>
#> 1 MX17004 2010     1 d     1     NA    NA
#> 2 MX17004 2010     1 d    10     NA    NA
#> 3 MX17004 2010     1 d    11     NA    NA
#> 4 MX17004 2010     1 d    12     NA    NA
#> 5 MX17004 2010     1 d    13     NA    NA
#> 6 MX17004 2010     1 d    14     NA    NA
#> 7 MX17004 2010     1 d    15     NA    NA
#> 8 MX17004 2010     1 d    16     NA    NA
#> 9 MX17004 2010     1 d    17     NA    NA
#> 10 MX17004 2010     1 d    18     NA    NA
#> # ... with 331 more rows
```

# mutate

mutate can either transform a column in place or create a new column in a dataset

```
weather4 <- mutate(weather3, day = as.numeric(day))  
as_tibble(weather4)
```

```
#> # A tibble: 341 x 7  
#>   id      year month symbol  day  tmax  tmin  
#>   <chr>   <int> <int> <chr> <dbl> <dbl> <dbl>  
#> 1 MX17004 2010     1 d      1    NA    NA  
#> 2 MX17004 2010     1 d     10    NA    NA  
#> 3 MX17004 2010     1 d     11    NA    NA  
#> 4 MX17004 2010     1 d     12    NA    NA  
#> 5 MX17004 2010     1 d     13    NA    NA  
#> 6 MX17004 2010     1 d     14    NA    NA  
#> 7 MX17004 2010     1 d     15    NA    NA  
#> 8 MX17004 2010     1 d     16    NA    NA  
#> 9 MX17004 2010     1 d     17    NA    NA  
#> 10 MX17004 2010     1 d     18    NA    NA  
#> # ... with 331 more rows
```

# mutate

mutate can also be used to deal with missing values, by replacing them with a value, for example

```
mutate(weather4, tmax = replace_na(tmax, 0))
```

You wouldn't want to do exactly this, of course

```
#>      id year month symbol day tmax tmin
#> 1  MX17004 2010     1     d   1  0.0  NA
#> 2  MX17004 2010     1     d  10  0.0  NA
#> 3  MX17004 2010     1     d  11  0.0  NA
#> 4  MX17004 2010     1     d  12  0.0  NA
#> 5  MX17004 2010     1     d  13  0.0  NA
#> 6  MX17004 2010     1     d  14  0.0  NA
#> 7  MX17004 2010     1     d  15  0.0  NA
#> 8  MX17004 2010     1     d  16  0.0  NA
#> 9  MX17004 2010     1     d  17  0.0  NA
#> 10 MX17004 2010     1     d  18  0.0  NA
#> 11 MX17004 2010     1     d  19  0.0  NA
#> 12 MX17004 2010     1     d   2  0.0  NA
#> 13 MX17004 2010     1     d  20  0.0  NA
#> 14 MX17004 2010     1     d  21  0.0  NA
#> 15 MX17004 2010     1     d  22  0.0  NA
#> 16 MX17004 2010     1     d  23  0.0  NA
#> 17 MX17004 2010     1     d  24  0.0  NA
#> 18 MX17004 2010     1     d  25  0.0  NA
#> 19 MX17004 2010     1     d  26  0.0  NA
#> 20 MX17004 2010     1     d  27  0.0  NA
#> 21 MX17004 2010     1     d  28  0.0  NA
#> 22 MX17004 2010     1     d  29  0.0  NA
#> 23 MX17004 2010     1     d   3  0.0  NA
#> 24 MX17004 2010     1     d  30 27.8 14.5
```

# filter

`filter` extracts **rows** based on criteria

So if we wanted to just grab January data, we could use

```
january <- filter(weather4, month==1)  
head(january)
```

```
#>      id year month symbol day tmax tmin  
#> 1 MX17004 2010     1     d   1   NA   NA  
#> 2 MX17004 2010     1     d  10   NA   NA  
#> 3 MX17004 2010     1     d  11   NA   NA  
#> 4 MX17004 2010     1     d  12   NA   NA  
#> 5 MX17004 2010     1     d  13   NA   NA  
#> 6 MX17004 2010     1     d  14   NA   NA
```

# Filter

Some comparison operators for filtering

| Operator | Meaning                          |
|----------|----------------------------------|
| ==       | Equals                           |
| !=       | Not equals                       |
| > / <    | Greater / less than              |
| >= / <=  | Greater or equal / Less or equal |
| !        | Not                              |
| %in%     | In a set                         |

Combining comparisons

| Operator | Meaning |
|----------|---------|
| &        | And     |
|          | Or      |

# filter

Let's use the mpg dataset from the ggplot2 package

```
mpg1 <- filter(mpg,  
  (year==1999) &  
  (class %in% c('minivan','suv')))  
select(mpg1, manufacturer, cty, hwy, class, year)
```

```
#> # A tibble: 35 x 5  
#>   manufacturer cty hwy class year  
#>   <chr>      <int> <int> <chr> <int>  
#> 1 chevrolet    13    17 suv    1999  
#> 2 chevrolet    11    15 suv    1999  
#> 3 chevrolet    14    17 suv    1999  
#> 4 dodge       18    24 minivan 1999  
#> 5 dodge       17    24 minivan 1999  
#> 6 dodge       16    22 minivan 1999  
#> 7 dodge       16    22 minivan 1999  
#> 8 dodge       15    22 minivan 1999  
#> 9 dodge       15    21 minivan 1999  
#> 10 dodge      13    17 suv    1999  
#> # ... with 25 more rows
```



# filter

A common use of `filter` is to remove rows with missing values from your dataset

```
weather5 <- filter(weather4,
                    !is.na(tmax) & !is.na(tmin))
head(weather5, 20)
```

`is.na` is a *function* that tests whether a value is missing or not.

So `!is.na` is the opposite of that.

```
#>      id year month symbol day tmax tmin
#> 1  MX17004 2010     1     d  30 27.8 14.5
#> 2  MX17004 2010     2     d  11 29.7 13.4
#> 3  MX17004 2010     2     d   2 27.3 14.4
#> 4  MX17004 2010     2     d  23 29.9 10.7
#> 5  MX17004 2010     2     d   3 24.1 14.4
#> 6  MX17004 2010     3     d  10 34.5 16.8
#> 7  MX17004 2010     3     d  16 31.1 17.6
#> 8  MX17004 2010     3     d   5 32.1 14.2
#> 9  MX17004 2010     4     d  27 36.3 16.7
#> 10 MX17004 2010     5     d  27 33.2 18.2
#> 11 MX17004 2010     6     d  17 28.0 17.5
#> 12 MX17004 2010     6     d  29 30.1 18.0
#> 13 MX17004 2010     7     d  14 29.9 16.5
#> 14 MX17004 2010     7     d   3 28.6 17.5
#> 15 MX17004 2010     8     d  13 29.8 16.5
#> 16 MX17004 2010     8     d  23 26.4 15.0
#> 17 MX17004 2010     8     d  25 29.7 15.6
#> 18 MX17004 2010     8     d  29 28.0 15.3
#> 19 MX17004 2010     8     d  31 25.4 15.4
#> 20 MX17004 2010     8     d   5 29.6 15.8
```

# arrange

arrange reorders **rows** of a data set according to the values of one or more variables

```
arrange(weather5, day)
```

Not quite.

```
#>      id year month symbol day tmax tmin
#> 1  MX17004 2010    12     d   1 29.9 13.8
#> 2  MX17004 2010     2     d   2 27.3 14.4
#> 3  MX17004 2010    11     d   2 31.3 16.3
#> 4  MX17004 2010     2     d   3 24.1 14.4
#> 5  MX17004 2010     7     d   3 28.6 17.5
#> 6  MX17004 2010    11     d   4 27.2 12.0
#> 7  MX17004 2010     3     d   5 32.1 14.2
#> 8  MX17004 2010     8     d   5 29.6 15.8
#> 9  MX17004 2010    10     d   5 27.0 14.0
#> 10 MX17004 2010    11     d   5 26.3  7.9
#> 11 MX17004 2010    12     d   6 27.8 10.5
#> 12 MX17004 2010    10     d   7 28.1 12.9
#> 13 MX17004 2010     8     d   8 29.0 17.3
#> 14 MX17004 2010     3     d  10 34.5 16.8
#> 15 MX17004 2010     2     d  11 29.7 13.4
#> 16 MX17004 2010     8     d  13 29.8 16.5
#> 17 MX17004 2010     7     d  14 29.9 16.5
#> 18 MX17004 2010    10     d  14 29.5 13.0
#> 19 MX17004 2010    10     d  15 28.7 10.5
#> 20 MX17004 2010     3     d  16 31.1 17.6
#> 21 MX17004 2010     6     d  17 28.0 17.5
#> 22 MX17004 2010     2     d  23 29.9 10.7
#> 23 MX17004 2010     8     d  23 26.4 15.0
#> 24 MX17004 2010     8     d  25 29.7 15.6
```

# arrange

```
arrange(weather5, month, day)
```

```
#>      id year month symbol day tmax tmin
#> 1  MX17004 2010     1     d  30 27.8 14.5
#> 2  MX17004 2010     2     d   2 27.3 14.4
#> 3  MX17004 2010     2     d   3 24.1 14.4
#> 4  MX17004 2010     2     d  11 29.7 13.4
#> 5  MX17004 2010     2     d  23 29.9 10.7
#> 6  MX17004 2010     3     d   5 32.1 14.2
#> 7  MX17004 2010     3     d  10 34.5 16.8
#> 8  MX17004 2010     3     d  16 31.1 17.6
#> 9  MX17004 2010     4     d  27 36.3 16.7
#> 10 MX17004 2010     5     d  27 33.2 18.2
#> 11 MX17004 2010     6     d  17 28.0 17.5
#> 12 MX17004 2010     6     d  29 30.1 18.0
#> 13 MX17004 2010     7     d   3 28.6 17.5
#> 14 MX17004 2010     7     d  14 29.9 16.5
#> 15 MX17004 2010     8     d   5 29.6 15.8
#> 16 MX17004 2010     8     d   8 29.0 17.3
#> 17 MX17004 2010     8     d  13 29.8 16.5
#> 18 MX17004 2010     8     d  23 26.4 15.0
#> 19 MX17004 2010     8     d  25 29.7 15.6
#> 20 MX17004 2010     8     d  29 28.0 15.3
#> 21 MX17004 2010     8     d  31 25.4 15.4
#> 22 MX17004 2010    10     d   5 27.0 14.0
#> 23 MX17004 2010    10     d   7 28.1 12.9
#> 24 MX17004 2010    10     d  14 29.5 13.0
#> 25 MX17004 2010    10     d  15 28.7 10.5
#> 26 MX17004 2010    10     d  28 31.2 15.0
#> 27 MX17004 2010    11     d   2 31.3 16.3
```

# arrange

1. I use arrange sparingly in my workflow
  - For spiffying up final presentation tables
  - If order is **really** important
2. Sorting data is one of the most computationally expensive operations you can do
  - It can crash your computer for big data

# Cluttering up our workspace

We've done a bit, but lets see all the objects we've created

```
ls()
```

```
#>  [1] "dat"          "f"            "fname"        "january"  
#>  [5] "mpg1"         "outdir"       "pdfname"      "weather_data"  
#>  [9] "weather1"     "weather2"     "weather3"     "weather4"  
#> [13] "weather5"
```

We see a lot of intermediate datasets we've created, that we aren't going to really use anymore

# **Workflow pipes in the tidyverse**

# Intermediate data sets

Recall how we cleaned the weather dataset yesterday

```
weather <- as_tibble(weather)
weather1 <- tidyr::gather(weather, key='day', value = 'temp', starts_with('d'))
weather2 <- spread(weather1, element, temp)
weather3 <- separate(weather2, day, c('symbol', 'day'), sep = 1)
weather4 <- select(weather3, -symbol)
weather5 <- mutate(weather4, day = as.numeric(day),
                    tmax = replace_na(tmax, 0),
                    tmin = replace_na(tmin, 0))
weather6 <- arrange(weather5, year, month, day)
```

This required us to create and keep track of several intermediate datasets

These datasets are essentially temporary datasets which do not hold the final result

What we did is a series of sequential steps to process the data

# The tidyverse pipe

The pipe operator `%>%` was first introduced in the `magrittr` package.

The idea behind the pipe is to take the result of one function and insert that as an input of another function

In the tidyverse, we start with a tibble, and every intermediate result is also a tibble



# The tidyverse pipe

1. The result of each function needs to be a tibble
2. You can omit the input for the data in the code for the function on the right side of the pipe operator, since we know that it's the output from the function to the left of the operator

# The tidyverse pipe

| Old School                                   | Piping school                                  |
|--|--|
| <code>mutate(mpg, differ = hwy-city))</code> | <code>mpg %&gt;% mutate(differ=hwy-cty)</code> |
| <code>select(mpg, cyl, cty, hwy)</code>      | <code>mpg %&gt;% select(cyl, cty, hyw)</code>  |

There is a handy shortcut in RStudio to type the pipe operator. It is Ctrl-Shift-M on Windows/Linux and Cmd-Shift-M on Mac.

# Reading a pipe

```
mpg %>%
  mutate(differ = hwy - cty) %>%
  group_by(cyl) %>%
  summarize(difference = mean(differ)) %>%
  ungroup()
```

```
#> # A tibble: 4 x 2
#>   cyl difference
#>   <int>      <dbl>
#> 1     4      7.79
#> 2     5      8.25
#> 3     6      6.61
#> 4     8      5.06
```

You can verbalize the pipe operator as **then**

# Reading a pipe

Since the end result of a pipe is a tibble, we can assign a name to store it

```
summary_mpg <- mpg %>%  
  mutate(differ = hwy - cty) %>%  
  group_by(cyl) %>%  
  summarize(difference = mean(differ)) %>%  
  ungroup()  
summary_mpg
```

```
#> # A tibble: 4 x 2  
#>   cyl difference  
#>   <int>     <dbl>  
#> 1     4     7.79  
#> 2     5     8.25  
#> 3     6     6.61  
#> 4     8     5.06
```

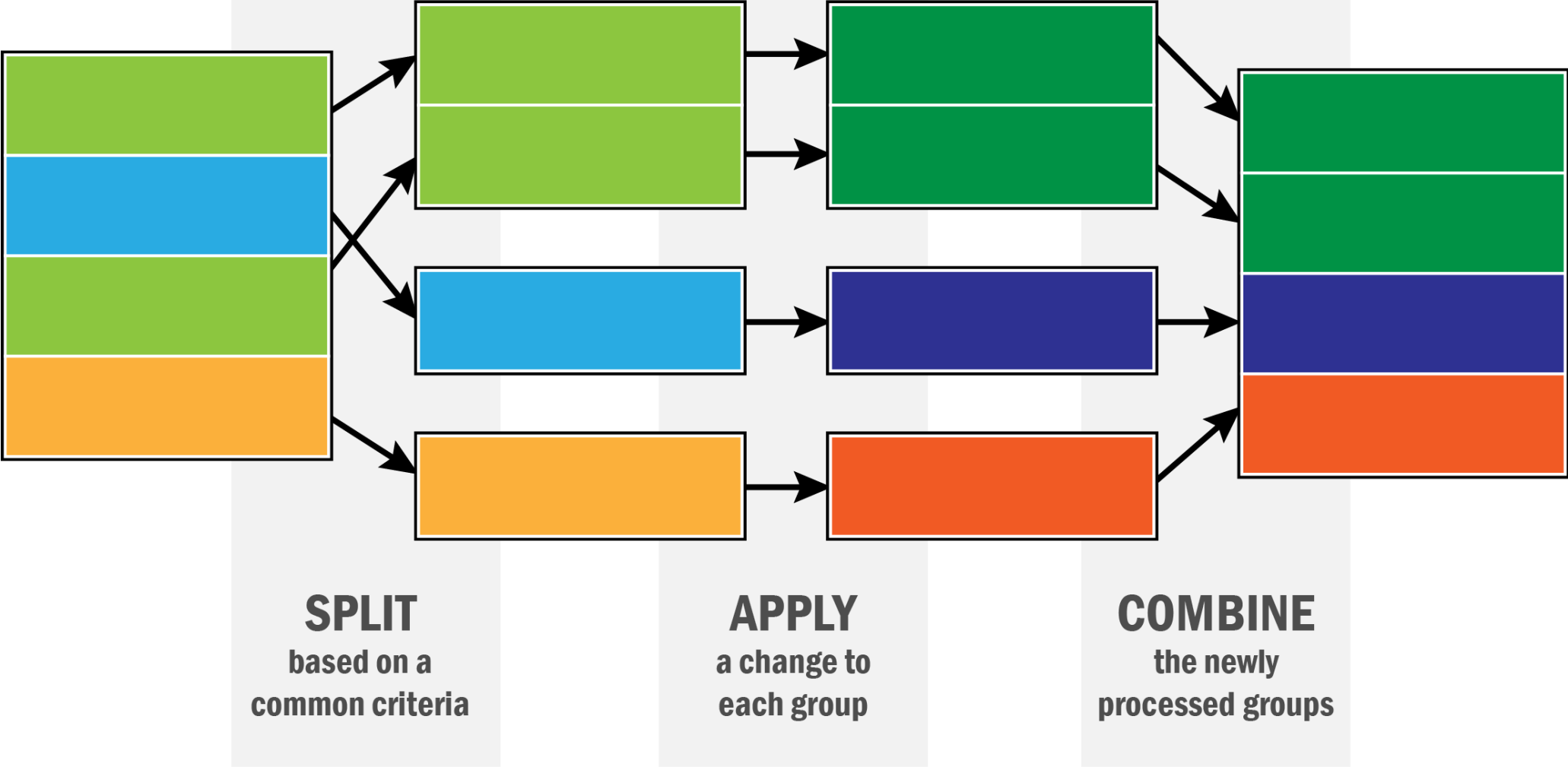
# Exercise

Transform the process of tidying the weather data into a pipe format

10:00

# group\_by and summarize

The `group_by` and `summarize` functions work together, in a principle called  
split-apply-combine



{

# Split-apply-combine

Let's find the average city mileage from the mpg dataset by vehicle class

```
mpg %>%  
  group_by(class) %>%  
  summarize(avgMPG = mean(cty))
```

```
#> # A tibble: 7 x 2  
#>   class      avgMPG  
#>   <chr>      <dbl>  
#> 1 2seater    15.4  
#> 2 compact   20.1  
#> 3 midsize   18.8  
#> 4 minivan   15.8  
#> 5 pickup    13  
#> 6 subcompact 20.4  
#> 7 suv       13.5
```

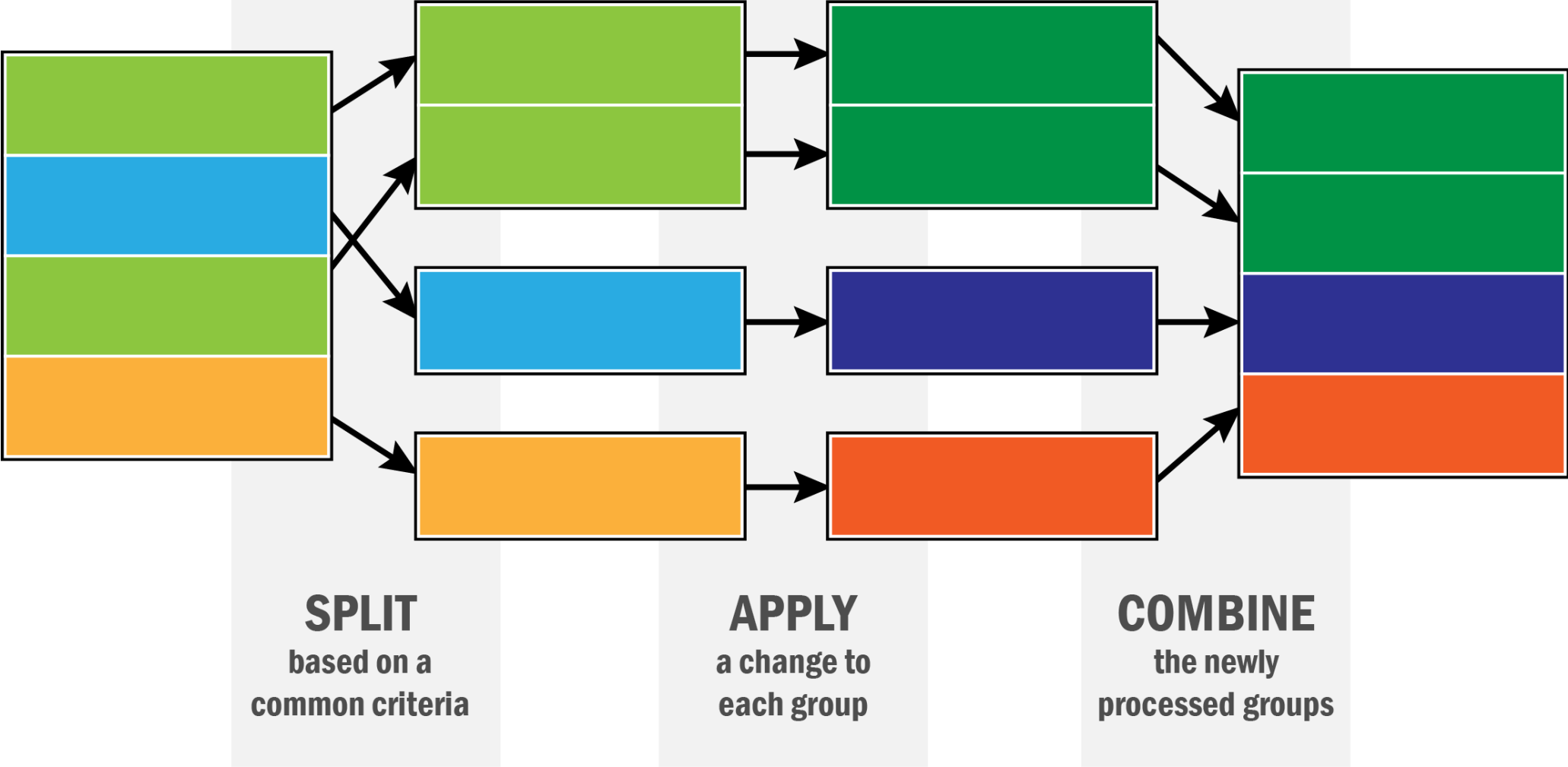


# Split-apply-combine

Let's find the average city mileage from the mpg dataset by vehicle class

```
mpg %>%  
  group_by(class) %>%  
  summarize(avgMPG = mean(cty)) %>%  
  knitr::kable(format='html')
```

| class      | avgMPG   |
|------------|----------|
| 2seater    | 15.40000 |
| compact    | 20.12766 |
| midsize    | 18.75610 |
| minivan    | 15.81818 |
| pickup     | 13.00000 |
| subcompact | 20.37143 |
| suv        | 13.50000 |



# Split-apply-combine

We can compute the monthly average minimum and maximum temperatures from the weather dataset

```
weather5 %>%  
  arrange(month, day) %>%  
  group_by(month) %>%  
  summarize(avgMin = mean(tmin, na.rm=T), avgMax = mean(tmax, na.rm=T))
```

```
#> # A tibble: 11 x 3  
#>   month avgMin avgMax  
#>   <int> <dbl> <dbl>  
#> 1     1  14.5  27.8  
#> 2     2  13.2  27.8  
#> 3     3  16.2  32.6  
#> 4     4  16.7  36.3  
#> 5     5  18.2  33.2  
#> 6     6  17.8  29.0  
#> 7     7  17    29.2  
#> 8     8  15.8  28.3  
#> 9    10  13.1  28.9  
#> 10    11  12.5  28.1  
#> 11    12  12.2  28.8
```

# Split-apply-combine

The split-apply-combine method is also useful in mutate.

In the weather dataset, let's impute the monthly averages for the missing values

```
weather4 %>% # weather4 still has missing values
  arrange(month, day) %>%
  group_by(month) %>%
  mutate(tmax = replace_na(tmax, mean(tmax, na.rm=T)),
         tmin = replace_na(tmin, mean(tmin, na.rm=T)))
```

```
#> # A tibble: 341 x 7
#> # Groups:   month [11]
#>   id      year month symbol  day  tmax  tmin
#>   <chr>   <int> <int> <chr> <dbl> <dbl> <dbl>
#> 1 MX17004  2010     1 d      1  27.8  14.5
#> 2 MX17004  2010     1 d      2  27.8  14.5
#> 3 MX17004  2010     1 d      3  27.8  14.5
#> 4 MX17004  2010     1 d      4  27.8  14.5
#> 5 MX17004  2010     1 d      5  27.8  14.5
#> 6 MX17004  2010     1 d      6  27.8  14.5
#> 7 MX17004  2010     1 d      7  27.8  14.5
#> 8 MX17004  2010     1 d      8  27.8  14.5
#> 9 MX17004  2010     1 d      9  27.8  14.5
#> 10 MX17004 2010     1 d     10  27.8  14.5
#> # ... with 331 more rows
```

# Split-apply-combine

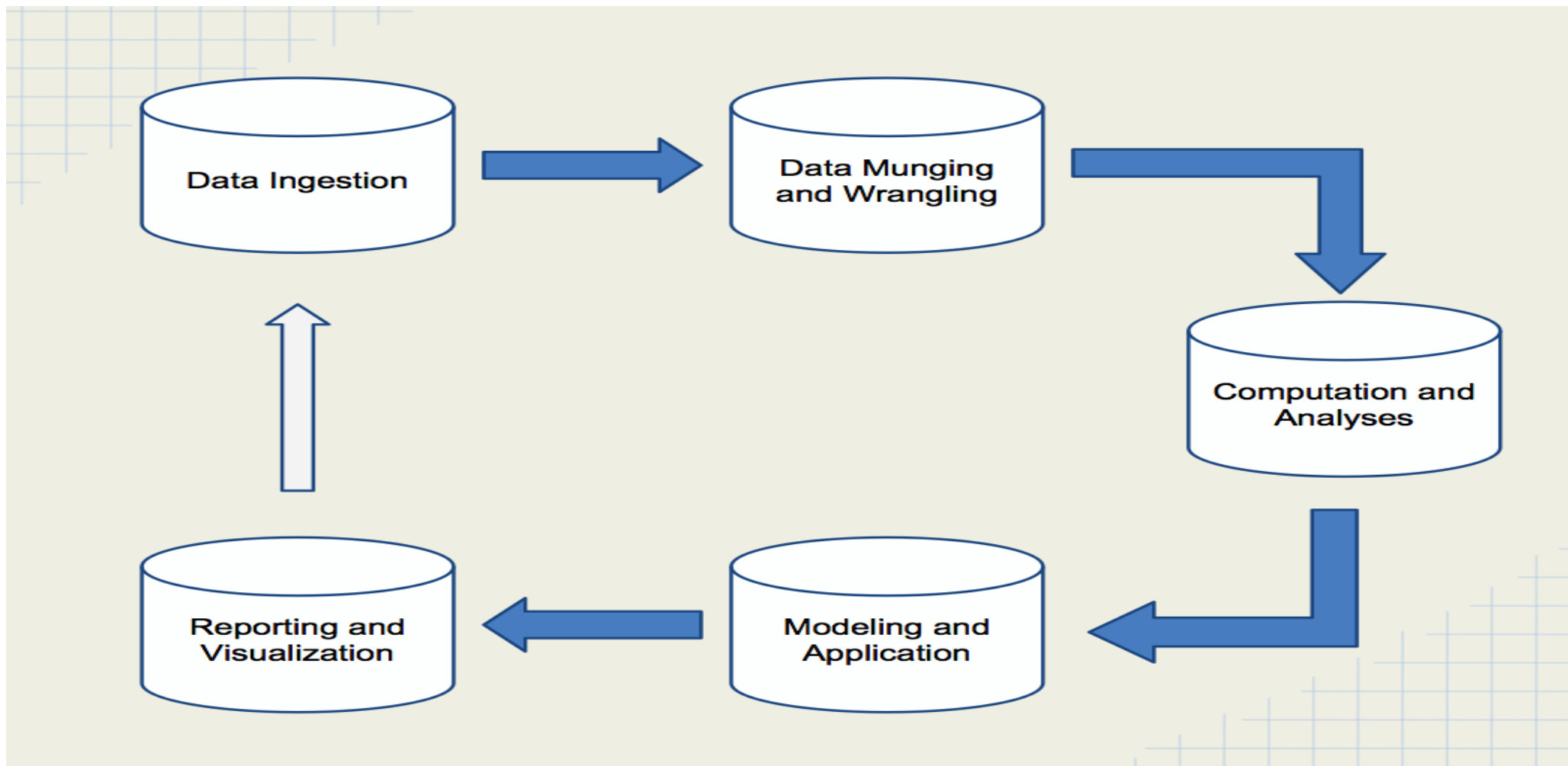
We can even use this principle for some filtering

```
weather5 %>%
  arrange(month, day) %>%
  group_by(month) %>%
  filter(tmax >= mean(tmax, na.rm=T)) %>%
  ungroup()
```

```
#> # A tibble: 16 x 7
#>   id      year month symbol  day  tmax  tmin
#>   <chr>   <int> <int> <chr> <dbl> <dbl> <dbl>
#> 1 MX17004  2010     1 d     30  27.8  14.5
#> 2 MX17004  2010     2 d     11  29.7  13.4
#> 3 MX17004  2010     2 d     23  29.9  10.7
#> 4 MX17004  2010     3 d     10  34.5  16.8
#> 5 MX17004  2010     4 d     27  36.3  16.7
#> 6 MX17004  2010     5 d     27  33.2  18.2
#> 7 MX17004  2010     6 d     29  30.1  18
#> 8 MX17004  2010     7 d     14  29.9  16.5
#> 9 MX17004  2010     8 d      5  29.6  15.8
#> 10 MX17004 2010     8 d      8  29    17.3
#> 11 MX17004 2010     8 d     13  29.8  16.5
#> 12 MX17004 2010     8 d     25  29.7  15.6
#> 13 MX17004 2010    10 d     14  29.5  13
#> 14 MX17004 2010    10 d     28  31.2  15
#> 15 MX17004 2010    11 d      2  31.3  16.3
#> 16 MX17004 2010    12 d      1  29.9  13.8
```

# Organizing data projects

# The data science pipeline



# RStudio Projects

- RStudio allows you to create [Projects](#)
- These encapsulate individual projects



# Use a template to organize your projects

Before you even get the data

- Set up a folder structure where
  - you know what goes where
  - you can have canned scripts/packages set things up
- Make sure it's the same structure every time
- Next time you visit the project, you don't have to go into desperate search mode

# MY STRUCTURE

| Name              |  |
|-------------------|--|
| .DS_Store         |  |
| background        |  |
| data              |  |
| .DS_Store         |  |
| raw               |  |
| rda               |  |
| docs              |  |
| graphs            |  |
| lib               |  |
| .DS_Store         |  |
| R                 |  |
| src               |  |
| tests             |  |
| packages.R        |  |
| reload.R          |  |
| scripts           |  |
| python            |  |
| Report.Rmd        |  |
| DataAcquisition.R |  |
| DataMunging.R     |  |
| Figures.R         |  |
| Modeling.R        |  |

- Background materials
- Raw data (storage, not to be touched)
- Intermediate and final R data sets
- Generated documents (docx, html, pdf)
- Graphs (pdf, png, tiff)
- Custom R functions (all of them, without exception)
- Custom C/C++ functions
- Unit tests
- List of R packages for the project
- Automated loading of functions and packages in a separate environment
- Scripts for file management and conversion

# Storing data

1. Keep one copy of the raw data in the format you received it
2. Read in the data into R (see Lecture 7) -> `DataAcquisition.R`
3. Save a copy of this data in RDSformat (use endings `.rds`). We'll see how to do this in a few slides

# Start working with the data

Summaries:

- `summary`
- `dplyr::summarise`
- `mean, sd, range`

Exploratory graphs

- `ggplot`
- `plot`

Maybe call this file `DataExploration.R`

# Data munging

- Reshaping data
- Aggregating data
- Split-apply-combine

Maybe call this file `DataMunging.R`

# Modeling

- Hypothesis testing
- Linear models
- Logistic regression
- Whatever model you need to run

This process requires a lot of exploration and trial-and-error, so it gets messy I'll usually create several files that look at different models, but once I'm done, my "final" models go in `Modeling.R`

■ Coming soon

# Packages to be used

You can use several packages in a particular project

It's good practice to load them first, and know what they are

- Makes sure packages are installed
- Makes sure package dependencies are met
- Makes sure package conflicts are known and fixed

# Packages to be used

```
reqpackages <- c('dplyr', 'reshape2', 'readxl',  
                'ggplot2', 'stringr', 'ggthemes')  
for(pkg in reqpackages){  
  if(!(pkg %in% row.names(installed.packages()))){  
    install.packages(pkg, repos='http://cran.r-project.org')  
  }  
}  
  
library(dplyr)  
library(reshape2)  
library(readxl)  
library(stringr)  
library(ggplot2)  
library(ggthemes)
```

This goes in packages.R



# Creating a pipeline

Now you can ensure that your analyses are reproducible by creating a pipeline where code is run sequentially in a particular order

```
source('packages.R')  
source('DataAcquisition.R')  
source('DataExploration.R')  
source('DataMunging.R')  
source('Modeling.R')
```

Essentially, I'm doing modular programming

- Separate code by function
- Makes it easier to debug
- Try to write code for a particular function once

# Saving your work

You would often like to store intermediate datasets, and final datasets, so that you can access them quickly.

There are several ways of saving even large datasets so that they can be quickly accessed.

| Function  | Package | Example   | Re-loading the data                                |
|-----------|---------|---|--|
| saveRDS   | base    | <code>saveRDS(weather, file = 'weather.rds')</code> | <code>weather &lt;- readRDS('weather.rds')</code>  |
| write_fst | fst     | <code>write_fst(weather, file='weather.fst')</code> | <code>weather &lt;- read_fst('weather.fst')</code> |