Tips and Tricks for Working with Real-world Data

Christopher R. Barbour

Wednesday, Septemer 28th, 2016

Format of Non-classroom data

- When going through coursework, a false sense of comfort with datasets can occur.
- Typically in practice, the form that data comes to you is not what you would like for analysis:
  - Individuals who do not perform analysis regularly do not understand how it should be formatted.
  - Data scraped from websites can come in non-standard formats, where a fair bit of processing is needed to get to a data frame.
  - Variables are coded in improper ways (missing data errors, improper continuous/categorical labeling, miscoded when importing into R).
  - Actual structure of the data does not correspond to the question of interest that we would like to address (long format versus wide, hierarchical structure of datasets, etc.).

Concept of Tidy Data

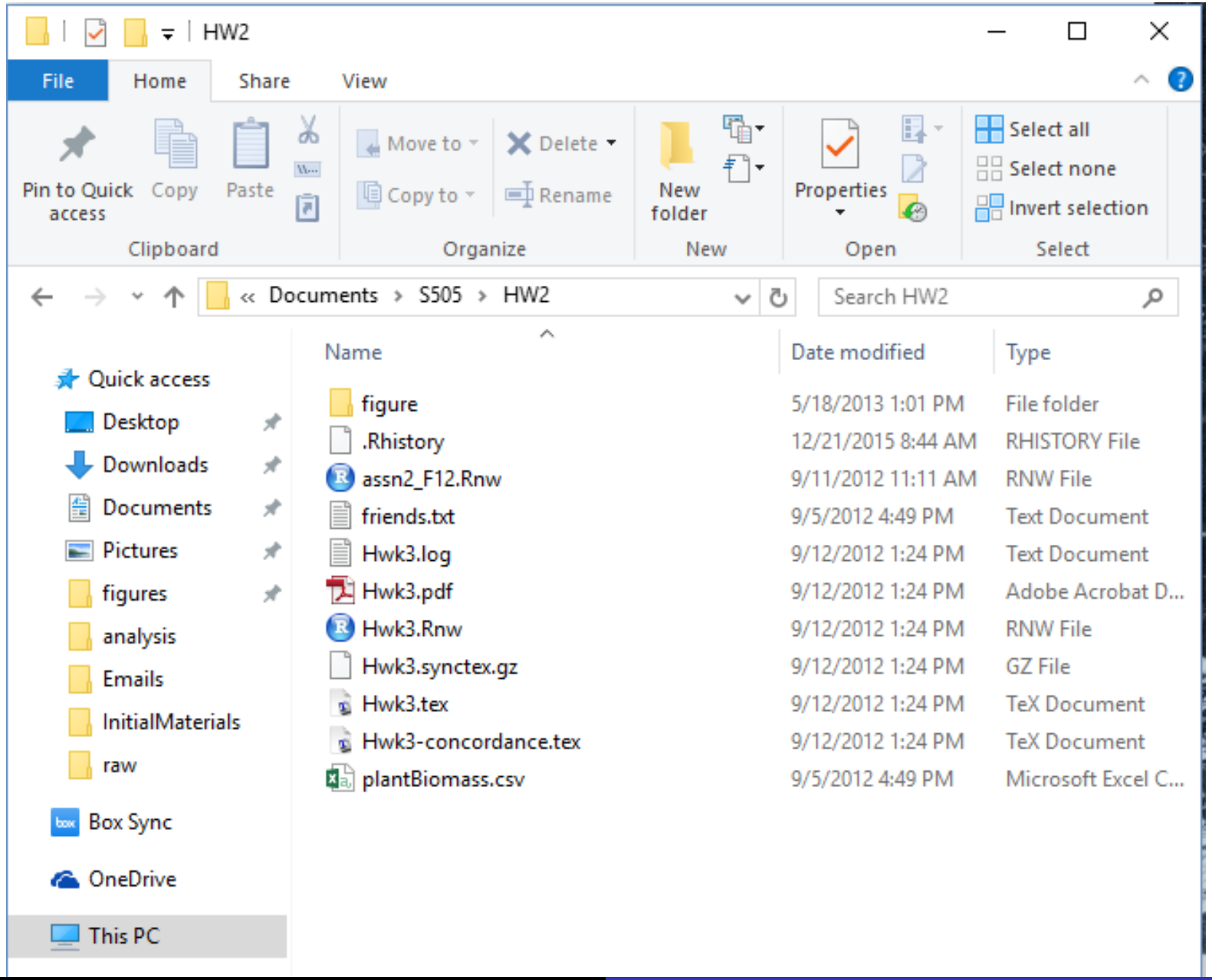
- The concept of "tidy data" was formally introduced in 2014 by a paper written in the Journal of Statistical Software by Hadley Wichkam.
- Main points correspond to:
  - each row represents an observation (lowest level)
  - each column represents a variable.
  - each observational unit is represented by a table.
- Since then, a number of tools (**tidyr**, **dplyr**, and **broom** R packages for example) have been developed for working with data frames that allow for concise, easily readable code.
- Not necessarily novel methodology (or even code in some cases), but a useful set of tools for working with data frames.

Concept of Tidy Data

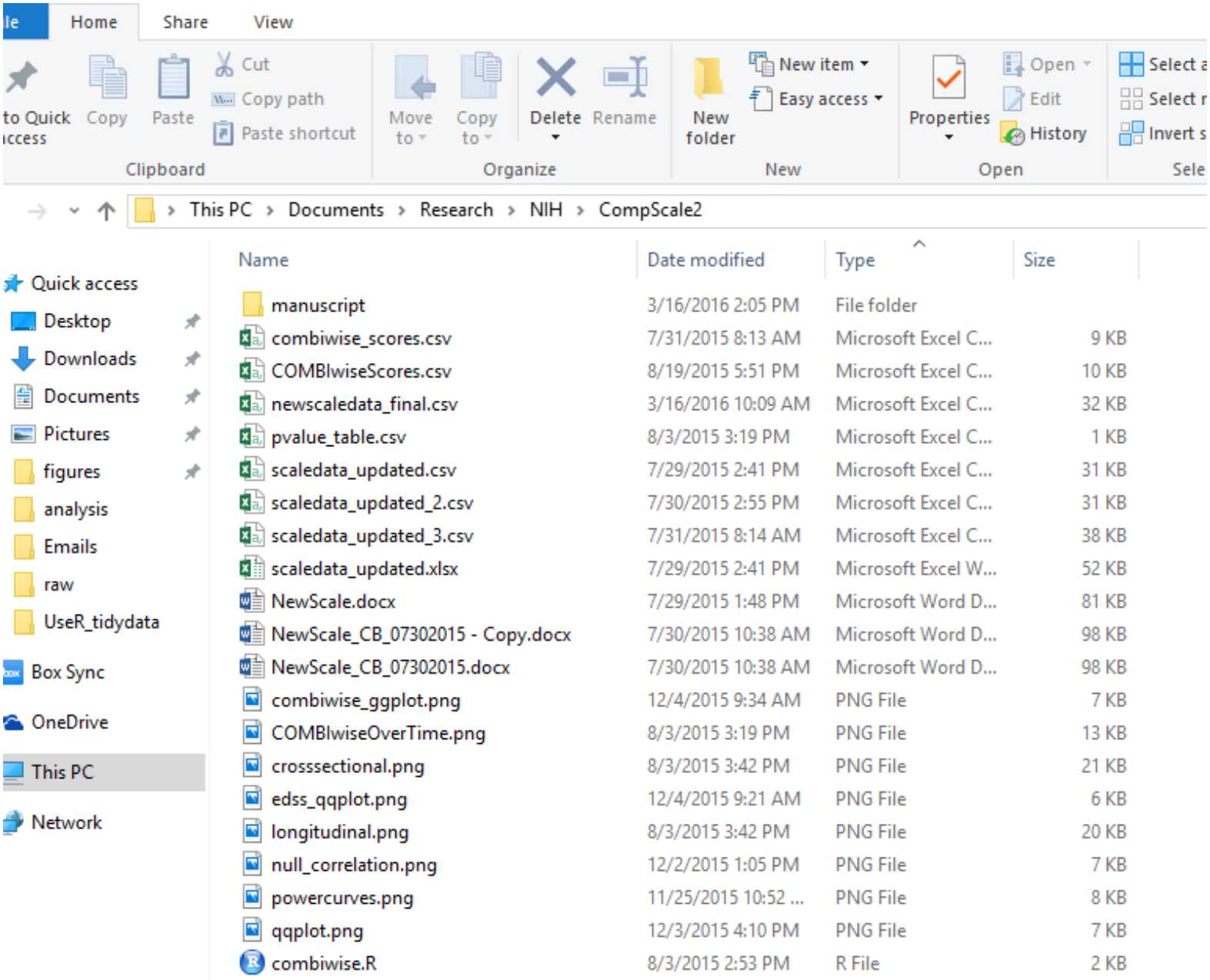
Why care?

- Eventually, you will have to work with data that is not provided to you in a perfect format.
- Unless you are a moderate/strong R programmer, you will need to invest in a set of tools to allow you to quickly locate and reformat different paces of your analysis.
- Working with demanding researchers/supervisors, a lot of pressure can be put on you to produce many different analysis/results under different conditions in a short time frame.
- Having a sloppy work-space and/or file structure can drive you crazy!

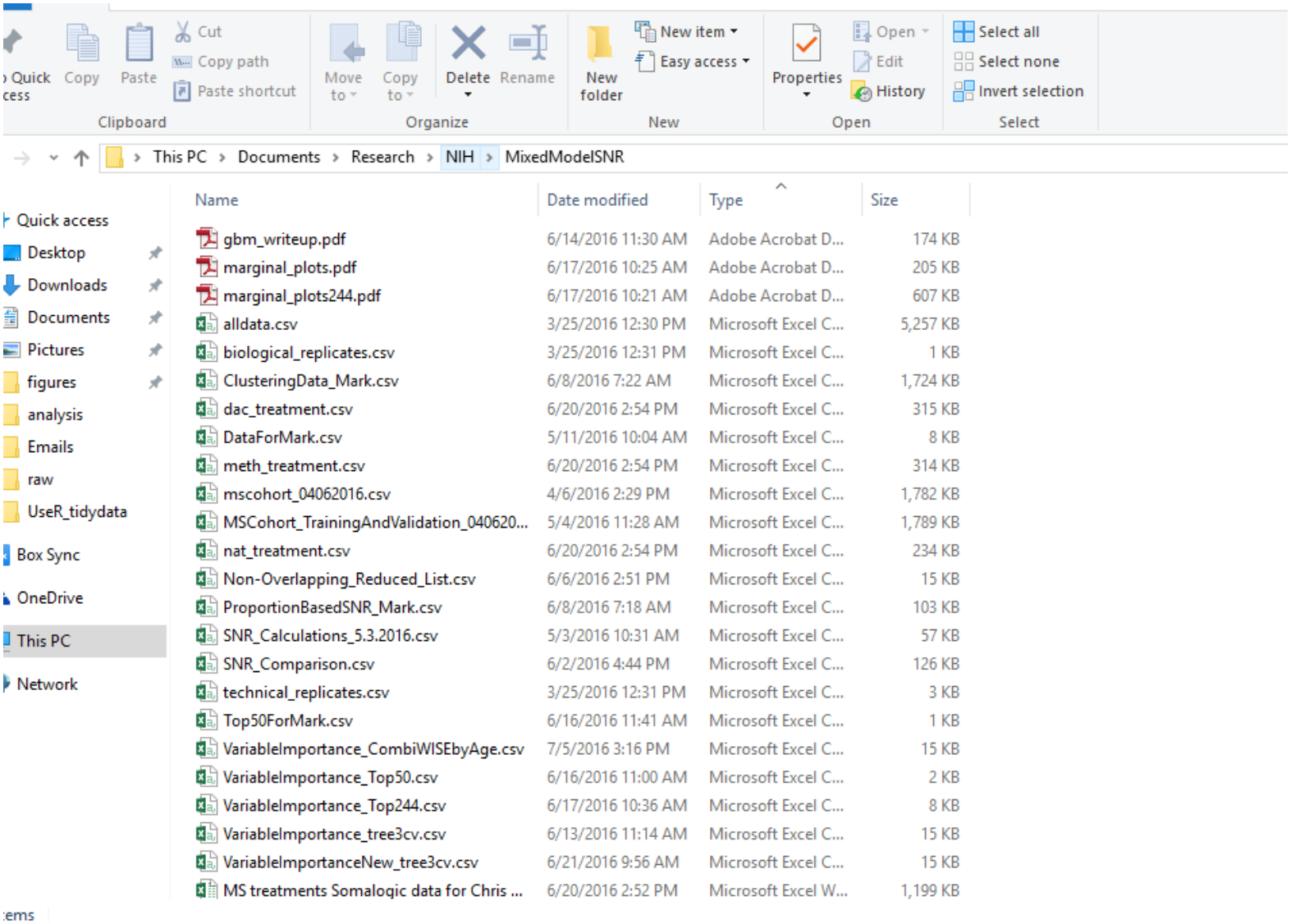
Easy to Work With



Slightly more difficult



AAAHHHHHHHH!!!!!



Personal Strategy

- figures
  - exploratory_results
  - final
- scripts
  - raw
    - packages.R
    - 0-load-data.R
    - 1-EDA.R
    - 2-modeling.R
  - final
- writeup
  - raw
  - final
- data
  - raw
  - processed
  - exploratory_results

Motivating Examples

- Land Resources: Researchers have collected coverage information on various plant functional groups along transects setup within sites (within locations). Measurements were collected on both sides of the transect, with one side of the transect having had pesticides sprayed for specific weeds.
- Public Health: Researchers have collected longitudinal data on patients with moderate to severe multiple sclerosis. The response is a proxy measurement for disease severity, and researchers are interested if a set of baseline covariates can predict disease progression over time.

Using dplyr and tidyr packages

- What **does** the data look like?

Using tbl_df()

- Turns a data frame (data.frame object) into a tbl (or tibble) object.
- Acts exactly as a data.frame in most instances, but prints much nicer in your R terminal.
- While data.frames will print the whole dataset (up to the "max.print"), tbl's will only print a smaller portion of information.

The select and rename Functions.

- The **select** function allows you to select a subset of variables (and rename them) from a data frame.
  - **select(data, x1, x2)** outputs a subset of data with columns x1, x2.
  - **select(data, -x3)** outputs a subset of data with all columns except x3.
  - **select(data, x4, everything())** outputs the data with all columns, with x4 in the first column.
  - **select(data, var1=x1, var2=x2)** outputs the same as bullet one, but with renamed variables.
  - **select(data, x1:x7)** outputs all variables between x1 and x7.
- The **rename** function works like bullet 4 above, but it keeps all of the variables.

The select and rename functions.

- So the following code takes the raw data, turns it into a tibble, removes the tapeside variable, and renames the rep variable to be transect.
- Passing this code to another individual may be difficult to read.

```
rename(select(tbl_df(pc_raw),-tapeside),transect=rep)
```

The piping operator

- The piping operator, **%>%**, is a useful tool for improving code readability.
  - **x %>% f** is equivalent to **f(x)**
  - **x %>% f(...)** is equivalent to **f(x, ...)**
  - **x %>% f(y, .)** is equivalent to **f(y,x)**
- When using pipes, readers (and yourself) can more simply follow along with the steps of your analysis.
- It also can help you quickly locate specific sections of pre-processing that you need to make changes too.

The piping operator

```
pc <- pc_raw %>%
  tbl_df() %>%
  select(-tapeside) %>%
  rename(transect=rep)
```

Reshaping data

- Depending on the type of analysis that we want to perform, the current format of the data may or may not be appropriate.
  - If we wanted to cluster the different points along transects based on the composition of functional groups, than the current format would be easier to work with.
  - If we wanted to set up a model that allows for different means for the different functional groups, than we would want to switch the data from a "wide" format to a "long" format.

```
## # A tibble: 3,916 x 5
##    litter  rock  soil    af    ag
##     <dbl> <dbl> <dbl> <dbl> <dbl>
## 1      72    10     6   0.1   0.2
## 2      72    10     6   0.1   1.1
## 3      80     0    14   0.1   0.1
## 4      80     0    14   0.1   0.1
## 5      65     5    23   0.1   0.2
## 6      65     5    23   1.0   1.1
```

Using the gather function

- **gather(data, key, value, . . .)**
  - "key"" specifies the name of the variable that the column names will be grouped into.
  - "value" specifies the name of the variable that the column values will be put into.
  - ". . ." is the specification of the columns that will be gathered (can use the **select** syntax).

```
pc_long <- pc %>%
  gather(functional_group, percent_cover,litter:target)
```

Using the spread function

- Sometimes we want to go the other way. For example, we may want the measurements for treated and untreated at the same transect point to be in the same row.
- **spread(data, key, value)**
  - "key" is the name of the current column which will represent column headings.
  - "value" is the name of the current column whose value will populate the cells.

```
pc_diff <- pc_long %>%
  spread(herbicide,percent_cover) %>%
  rename(treated_side=Yes, untreated_side=No)
```

Subsetting rows of a dataframe, the filter function.

- Sometimes, we are only interested in a subset of observations for a specific analysis.
- For example, the researchers were initially interested in the "canopy cover" for this analysis.
- **filter(data,condition)** returns a dataset where the logical condition specified is TRUE.

```
pc_diff <- pc_diff %>%
  filter(covertype=="c") %>%
  select(-covertype)
```

Creating new variables, the mutate function.

- **mutate(data, . . .)** is a function that can be used to create new variables. The output of mutate is a tibble that includes the created variables.
- The ". . . " can be a series of any "window" functions, which take in a vector of values and return an equal size vector of values (uses recycling?).
- **mutate_each** is a function that can be used to apply a window function to every column of a dataframe.
- **transmute** is the same as mutate, but drops original columns used in the mutation

```
pc_diff <- pc_diff %>%
  mutate(difference=treated_side-untreated_side,
         year=as.factor(year))
```

Sorting observations by specific variables using arrange

- Sometimes, we would like our observations to appear in a particular order for viewing purposes.
- For example, we may want all of the values for a specific functional group at a specific site to be grouped together when we view our data.
- The **arrange(data, . . .)** function allows us to sort our observations in either ascending or descending order for a hierarchy of variables.
- In our example, we could sort first by year, than by site, and finally by functional group.

Sorting observations by specific variables using arrange

```
pc_diff <- pc_diff %>%
  arrange(desc(year),site,functional_group)
```

Summarizing your data using the summarise function.

- **summarise(data, . . .)** is a handy way to display summary information on one or multiple variables in your data.
- The ". . . " can be a sequence of any summary functions, or functions that input a vector of values and output a single value.
- **summarise_each** can apply a summary function (or functions) to each column in the dataset.
- VERY powerful in conjunction with **group_by** statements.

Summarizing your data using the summarise function.

```
pc_diff %>%
  summarise(mean_difference=mean(difference,na.rm=TRUE),
            sd_difference=sd(difference,na.rm=TRUE))
```

```
## # A tibble: 1 x 2
##   mean_difference sd_difference
##             <dbl>         <dbl>
## 1      -0.2858008      7.139296
```

Creating a group structure, the group_by function.

- Many times, it is useful to perform data processing/summarization by a grouping structure in your data, which can be composed of one or more variables.
- For instance, we may want to aggregate the data to the transect level and consider the average difference within each transect to be the response variable.
- The **group_by(data,...)** in conjunction with **summarise** is an incredibly powerful tool to quickly and efficiently perform these kind of tasks.
- This grouping structure can also be applied to creating new variables with **mutate**, where the window functions that are being used are applied *within* each group. (If a summary function is given, than it is applied in each group and the output is recycled).

Creating a group structure, the group_by function.

```
pc_diff %>%
  group_by(targetweed) %>%
  summarise(mean_difference=mean(difference,na.rm=TRUE),
            sd_difference=sd(difference,na.rm=TRUE)) %>%
  mutate(effect_size=mean_difference/sd_difference)
```

```
## # A tibble: 3 x 4
##   targetweed mean_difference sd_difference effect_size
##        <fctr>           <dbl>         <dbl>       <dbl>
## 1       CEST      -0.2175450      6.898102 -0.03153694
## 2       LIDA      -0.2543111      7.329896 -0.03469506
## 3       PORE      -0.3788894      7.236437 -0.05235855
```

Creating a group structure, the group_by function.

```
pc_transect <- pc_diff %>%
  select(-untreated_side,-treated_side, -tapemeter) %>%
  group_by(year,location,targetweed,site,transect,functiona
  summarise_each(funs(mean)) %>%
  ungroup
```

Putting it all together

```
pc_diff <- pc_raw %>%
  tbl_df() %>%
  select(-tapeside) %>%
  rename(transect=rep) %>%
  gather(functional_group, percent_cover,litter:target) %>%
  spread(herbicide,percent_cover) %>%
  rename(treated_side=Yes, untreated_side=No) %>%
  mutate(difference=treated_side-untreated_side,
         year=as.factor(year))
pc_transect <- pc_diff %>%
  select(-untreated_side,-treated_side, -tapemeter) %>%
  group_by(year,location,targetweed,site,
           transect,functional_group,covertype) %>%
  summarise_each(funs(mean)) %>%
  ungroup
```

Using tibbles with the purrr package

- **tibbles** are a relatively new type of data.frame; For a more detailed discussion, checkout Hadley Wichkam's keynote address at User2016 (to find, google "Channel 9 useR2016"").
- The main benefit of using **tibbles** is that they allow for non-standard columns that data.frames do not easily allow for (columns that are datasets, lists, linear models, etc.)
- Very powerful tool!
- In the public health example, we want to model the progression of disease severity over time using baseline covariates. Lets pull off individual SLR slopes and some baseline covariates using **tibbles**.

Using tibbles with the purrr package

```
## # A tibble: 1,431 x 6
##    patient       date therapy     y    x1    x2
##      <chr>     <fctr>  <fctr> <dbl> <dbl> <dbl>
## 1        1 06/23/2010       A  0.70  0.95  0.38
## 2        1 03/23/2011       A  0.83  1.02  0.47
## 3        1 07/29/2011       A  0.66  1.05  0.31
## 4        1 01/12/2012       B  0.47  1.09  0.13
## 5        1 06/21/2012       B  0.46  1.13  0.10
## 6        1 12/20/2012       B  0.45  1.18  0.08
## 7        1 05/22/2013       B  0.38  1.21  0.01
## 8        1 12/05/2013       C  0.16  1.26 -0.20
## 9        1 06/25/2014       C  0.14  1.31 -0.23
## 10       2 09/29/2008       A -0.75  0.21 -0.81
## # ... with 1,421 more rows
```

Using tibbles with the purrr package

```
clindat <- clindat %>%
  group_by(patient) %>%
  mutate(date=mdy(date),
         time=as.numeric((date-min(date))/365)) %>%
  ungroup()
```

Using tibbles with the purrr package

```
clindat %>%
  group_by(patient) %>%
  mutate(baseline_x1=x1[1], baseline_x2=x2[1]) %>%
  group_by(baseline_x1, baseline_x2,add=TRUE) %>%
  nest() %>%
  mutate(unique_therapies =
           map(data,~unique(.$therapy_group))) %>%
  mutate(model=
           map(data,~lm(y~time,data=.))) %>%
  mutate(slope=
           sapply(model, function(mod){coef(mod)[2]}))
```

Punchline

- Data rarely come to you in a perfect format.
- Tools from **dplyr** and **tidyr** can help you re-format data in proper ways for different types of analysis using easy to read (understand) code.
- Whether you adopt the use of these tools, or work more using base R functions, knowing how to clean and reformat data is a very important skill to acquire for working outside of the classroom.
- I have done data cleaning and reformatting using both **dplyr** and base R (probably not the best), and personally, I have found working with **dplyr** to be cleaner to work with.

Thank you!!!

Excercise

- Working with the 2000 Olympic diving results. Feel free to try base R as well as the tools presented today and see which you prefer. Download **dplyr**, **tidyr**, and **purrr** if you have not already.

1. For every individual dive, calculate the mean score from all of the judges and add it into your dataset.
2. Ignore the Judge names and Judge country for this problem (will help to remove them). Create a new dataset where each of the 7 scores for a single dive are on the same row. Arrange the order of the rows in the dataset in a way that you think is meaningful.
3. If a country has more than 5 divers in the preliminary round, examine the diver to diver variability within that country using mixed-models (the **lme** function in the **nlme** package, or the **lmer** function in the **lme4** package can perform these, talk to a stats neighbor for help).