Tidyverse Tutorial

Daljit Singh (singhdj2@ksu.edu)

5/24/2018

Why tidyverse?

The aim of this tutorial is to make you familiar with the verbs that are part of a very neat R meta-package called *tidyverse*. Of late I have become a heavy user of these functions. My views are biased but I think everyone can benefit from using tidy workflows in their analysis. It not only has neat data wrangling functions, the ggplot2 makes it a great graphics package as well. At the core of these functions is the tidy data philosophy which views the tidy data as essenatially a long format unique data rows. You can read more about Hadely Wickham, the creator of *tidyverse* package. He is a prolific writer and a frequent lurker on reddit, stackoverflow, quora etc. The main tidyverse packages are dplyr,tidyr,stringr, and ggplot2. You must have heard about some of these packages before. For the sake of time, I will try to stick with the most useful (in my opinion) verbs in this tutorial. So lets go...

Load packages

I like to use package manager 'pacman' as it saves you the extra work of downloading the package if it doesn't already exist in your environment. The 'if loop' below is an example how you do it without pacman. You may want to check out some other interesting pacman functions as well.

```
# download pacman
if (!require(pacman)){
  install.packages('pacman')
}
```

```
## Loading required package: pacman
```

```
# load packages
library(pacman)
p_load(tidyverse,lme4,corrr)
```

Load data file

Phenotype data description

We will use this sample dataset from my field experiments in South Asia. These experiments are part of CIMMYT's advanced yield trials in South Asia. These trials are laid out in an alpha-lattice design. But we will treat them as RCBD for the sake of simplicity. The dataframe has 'plot_id', Canopy Temp ('CT'), 'NDVI', and Grain Yield (GRYLD) data columns. The CT and NDVI were collected at multiple time-points during the wheat season. Each date of CT and NDVI data will be treated as separate trait for this tutorial.

```
myFile <- '/Users/singhdj2/Dropbox/1.Research_and_Education/Kansas_State_University_2014-/Proffessional_
dat <- read.csv(myFile, header = T, stringsAsFactors = F)
```

```
# check data class
class(dat)
```

```
## [1] "data.frame"
```

```
# change to tibble format for convenience
dat <- as.tibble(dat)
# lets take a glimpse at our data
glimpse(dat)
```

```
## Observations: 1,200
## Variables: 12
## $ plot_id   <chr> "18-LDH-BMZ-EHT-10001", "18-LDH-BMZ-EHT-10002", "18-...
## $ CT.101    <dbl> 18.7, 18.0, 18.3, 18.2, 18.3, 17.8, 17.8, 18.0, 18.1...
## $ CT.121    <dbl> 22.5, 21.6, 22.7, 21.4, 21.4, 20.5, 21.8, 21.9, 23.4...
## $ CT.136    <dbl> 24.0, 24.5, 22.6, 23.6, 26.2, 25.5, 25.7, 24.0, 26.7...
## $ CT.156    <dbl> 31.2, 27.0, 29.2, 29.8, 29.9, 30.9, 29.2, 28.8, 30.9...
## $ CT.77     <dbl> 14.8, 15.1, 13.5, 14.7, 14.8, 14.3, 14.3, 14.8, 15.2...
## $ GRYLD.NA  <dbl> 8.372, 7.159, 8.068, 6.914, 6.592, 6.835, 6.748, 8.0...
## $ NDVIG.101 <dbl> 0.84, 0.86, 0.83, 0.83, 0.80, 0.80, 0.81, 0.78, 0.82...
## $ NDVIG.120 <dbl> 0.81, 0.86, 0.84, 0.82, 0.82, 0.84, 0.82, 0.80, 0.81...
## $ NDVIG.135 <dbl> 0.73, 0.78, 0.78, 0.71, 0.73, 0.74, 0.74, 0.75, 0.69...
## $ NDVIG.154 <dbl> 0.44, 0.59, 0.56, 0.37, 0.45, 0.42, 0.56, 0.47, 0.26...
## $ NDVIG.78  <dbl> 0.81, 0.82, 0.78, 0.74, 0.72, 0.74, 0.78, 0.70, 0.75...
```

The *glimpse* and *tibble* functions are part of tidyverse. *glimpse* is nice in that it gives us both headshot and a structure of data variables. *Tibble* has its own benefits which we will see as we move along this tutorial. You will also notice that our data is in a wide format, which is a common format in reasearch experiments. If you are familiar with structured databases like SQL then think of this as opposite of database format. Here the data variables (i.e. GRYLD, CT and NDVI) are arranged as columns as opposed to long format where each row corresponds to unique data points.

Concept of pipes

By far the most convenient feature of *tidyverse* is the ability to pipe your code. If you are familiar with Unix pipes, you could easily relate to the idea. I will give a pertinent exmple. Lets say I want to execute the lines above (data loading section) in a single line of code. We can do this with pipes easily:

```
read.csv(myFile,header=T,stringsAsFactors = F) %>%
  as.tibble() %>%
  glimpse()
```

```
## Observations: 1,200
## Variables: 12
## $ plot_id   <chr> "18-LDH-BMZ-EHT-10001", "18-LDH-BMZ-EHT-10002", "18-...
## $ CT.101    <dbl> 18.7, 18.0, 18.3, 18.2, 18.3, 17.8, 17.8, 18.0, 18.1...
## $ CT.121    <dbl> 22.5, 21.6, 22.7, 21.4, 21.4, 20.5, 21.8, 21.9, 23.4...
## $ CT.136    <dbl> 24.0, 24.5, 22.6, 23.6, 26.2, 25.5, 25.7, 24.0, 26.7...
## $ CT.156    <dbl> 31.2, 27.0, 29.2, 29.8, 29.9, 30.9, 29.2, 28.8, 30.9...
## $ CT.77     <dbl> 14.8, 15.1, 13.5, 14.7, 14.8, 14.3, 14.3, 14.8, 15.2...
## $ GRYLD.NA  <dbl> 8.372, 7.159, 8.068, 6.914, 6.592, 6.835, 6.748, 8.0...
## $ NDVIG.101 <dbl> 0.84, 0.86, 0.83, 0.83, 0.80, 0.80, 0.81, 0.78, 0.82...
## $ NDVIG.120 <dbl> 0.81, 0.86, 0.84, 0.82, 0.82, 0.84, 0.82, 0.80, 0.81...
## $ NDVIG.135 <dbl> 0.73, 0.78, 0.78, 0.71, 0.73, 0.74, 0.74, 0.75, 0.69...
## $ NDVIG.154 <dbl> 0.44, 0.59, 0.56, 0.37, 0.45, 0.42, 0.56, 0.47, 0.26...
## $ NDVIG.78  <dbl> 0.81, 0.82, 0.78, 0.74, 0.72, 0.74, 0.78, 0.70, 0.75...
```

We can capture this output in a new object like this:

```
dat.tib <- read.csv(myFile,header=T,stringsAsFactors = F) %>%
  as.tibble() %>%
  glimpse()
```

```
## Observations: 1,200
## Variables: 12
## $ plot_id   <chr> "18-LDH-BMZ-EHT-10001", "18-LDH-BMZ-EHT-10002", "18-...
## $ CT.101    <dbl> 18.7, 18.0, 18.3, 18.2, 18.3, 17.8, 17.8, 18.0, 18.1...
## $ CT.121    <dbl> 22.5, 21.6, 22.7, 21.4, 21.4, 20.5, 21.8, 21.9, 23.4...
## $ CT.136    <dbl> 24.0, 24.5, 22.6, 23.6, 26.2, 25.5, 25.7, 24.0, 26.7...
## $ CT.156    <dbl> 31.2, 27.0, 29.2, 29.8, 29.9, 30.9, 29.2, 28.8, 30.9...
## $ CT.77     <dbl> 14.8, 15.1, 13.5, 14.7, 14.8, 14.3, 14.3, 14.8, 15.2...
## $ GRYLD.NA  <dbl> 8.372, 7.159, 8.068, 6.914, 6.592, 6.835, 6.748, 8.0...
## $ NDVIG.101 <dbl> 0.84, 0.86, 0.83, 0.83, 0.80, 0.80, 0.81, 0.78, 0.82...
## $ NDVIG.120 <dbl> 0.81, 0.86, 0.84, 0.82, 0.82, 0.84, 0.82, 0.80, 0.81...
## $ NDVIG.135 <dbl> 0.73, 0.78, 0.78, 0.71, 0.73, 0.74, 0.74, 0.75, 0.69...
## $ NDVIG.154 <dbl> 0.44, 0.59, 0.56, 0.37, 0.45, 0.42, 0.56, 0.47, 0.26...
## $ NDVIG.78  <dbl> 0.81, 0.82, 0.78, 0.74, 0.72, 0.74, 0.78, 0.70, 0.75...
```

Notice that with pipes one can easily stack things together while keeping the code clean, concise, and easy to read.

Selecting variable columns

Tidyverse::dplyr allows you to select the desirable data columns in many different (and very intuitive) ways: number, bare variable names, range of variable columns etc.

```
# give me the plot_id and column names starting with CT
dat.tib %>%
  select(plot_id,starts_with('CT'))
```

```
## # A tibble: 1,200 x 6
##    plot_id              CT.101 CT.121 CT.136 CT.156 CT.77
##    <chr>                 <dbl>  <dbl>  <dbl>  <dbl> <dbl>
##  1 18-LDH-BMZ-EHT-10001   18.7   22.5   24.0   31.2  14.8
##  2 18-LDH-BMZ-EHT-10002   18.0   21.6   24.5   27.0  15.1
##  3 18-LDH-BMZ-EHT-10003   18.3   22.7   22.6   29.2  13.5
##  4 18-LDH-BMZ-EHT-10004   18.2   21.4   23.6   29.8  14.7
##  5 18-LDH-BMZ-EHT-10005   18.3   21.4   26.2   29.9  14.8
##  6 18-LDH-BMZ-EHT-10006   17.8   20.5   25.5   30.9  14.3
##  7 18-LDH-BMZ-EHT-10007   17.8   21.8   25.7   29.2  14.3
##  8 18-LDH-BMZ-EHT-10008   18.0   21.9   24.0   28.8  14.8
##  9 18-LDH-BMZ-EHT-10009   18.1   23.4   26.7   30.9  15.2
## 10 18-LDH-BMZ-EHT-1001    16.1   22.8   26.4   27.6  16.4
## # ... with 1,190 more rows
```

```
# give me columns in a certain range with names
dat.tib %>%
  select(plot_id:CT.136)
```

```
## # A tibble: 1,200 x 4
##    plot_id              CT.101 CT.121 CT.136
##    <chr>                 <dbl>  <dbl>  <dbl>
##  1 18-LDH-BMZ-EHT-10001   18.7   22.5   24.0
##  2 18-LDH-BMZ-EHT-10002   18.0   21.6   24.5
```

```
##  3 18-LDH-BMZ-EHT-10003   18.3   22.7   22.6
##  4 18-LDH-BMZ-EHT-10004   18.2   21.4   23.6
##  5 18-LDH-BMZ-EHT-10005   18.3   21.4   26.2
##  6 18-LDH-BMZ-EHT-10006   17.8   20.5   25.5
##  7 18-LDH-BMZ-EHT-10007   17.8   21.8   25.7
##  8 18-LDH-BMZ-EHT-10008   18.0   21.9   24.0
##  9 18-LDH-BMZ-EHT-10009   18.1   23.4   26.7
## 10 18-LDH-BMZ-EHT-1001    16.1   22.8   26.4
## # ... with 1,190 more rows
```

```
# you can select with contains argument (similar to MySQL).
dat.tib %>%
  select(plot_id,contains('CT'))
```

```
## # A tibble: 1,200 x 6
##    plot_id              CT.101 CT.121 CT.136 CT.156 CT.77
##    <chr>                 <dbl>  <dbl>  <dbl>  <dbl> <dbl>
##  1 18-LDH-BMZ-EHT-10001   18.7   22.5   24.0   31.2  14.8
##  2 18-LDH-BMZ-EHT-10002   18.0   21.6   24.5   27.0  15.1
##  3 18-LDH-BMZ-EHT-10003   18.3   22.7   22.6   29.2  13.5
##  4 18-LDH-BMZ-EHT-10004   18.2   21.4   23.6   29.8  14.7
##  5 18-LDH-BMZ-EHT-10005   18.3   21.4   26.2   29.9  14.8
##  6 18-LDH-BMZ-EHT-10006   17.8   20.5   25.5   30.9  14.3
##  7 18-LDH-BMZ-EHT-10007   17.8   21.8   25.7   29.2  14.3
##  8 18-LDH-BMZ-EHT-10008   18.0   21.9   24.0   28.8  14.8
##  9 18-LDH-BMZ-EHT-10009   18.1   23.4   26.7   30.9  15.2
## 10 18-LDH-BMZ-EHT-1001    16.1   22.8   26.4   27.6  16.4
## # ... with 1,190 more rows
```

```
# deselect/drop columns with minus sign
dat.tib %>%
  select(plot_id,-contains('CT'),starts_with('NDVI'))
```

```
## # A tibble: 1,200 x 6
##    plot_id              NDVIG.101 NDVIG.120 NDVIG.135 NDVIG.154 NDVIG.78
##    <chr>                    <dbl>     <dbl>     <dbl>     <dbl>    <dbl>
##  1 18-LDH-BMZ-EHT-10001     0.840     0.810     0.730     0.440    0.810
##  2 18-LDH-BMZ-EHT-10002     0.860     0.860     0.780     0.590    0.820
##  3 18-LDH-BMZ-EHT-10003     0.830     0.840     0.780     0.560    0.780
##  4 18-LDH-BMZ-EHT-10004     0.830     0.820     0.710     0.370    0.740
##  5 18-LDH-BMZ-EHT-10005     0.800     0.820     0.730     0.450    0.720
##  6 18-LDH-BMZ-EHT-10006     0.800     0.840     0.740     0.420    0.740
##  7 18-LDH-BMZ-EHT-10007     0.810     0.820     0.740     0.560    0.780
##  8 18-LDH-BMZ-EHT-10008     0.780     0.800     0.750     0.470    0.700
##  9 18-LDH-BMZ-EHT-10009     0.820     0.810     0.690     0.260    0.750
## 10 18-LDH-BMZ-EHT-1001      0.780     0.770     0.760     0.430    0.720
## # ... with 1,190 more rows
```

```
# also the sequence of columns in select() is preserved in output, meaning you can use select to rearra
dat.tib %>%
  select(plot_id,CT.77,CT.101,CT.121)
```

```
## # A tibble: 1,200 x 4
##    plot_id              CT.77 CT.101 CT.121
##    <chr>                <dbl>  <dbl>  <dbl>
##  1 18-LDH-BMZ-EHT-10001  14.8   18.7   22.5
```

```
##  2 18-LDH-BMZ-EHT-10002  15.1   18.0   21.6
##  3 18-LDH-BMZ-EHT-10003  13.5   18.3   22.7
##  4 18-LDH-BMZ-EHT-10004  14.7   18.2   21.4
##  5 18-LDH-BMZ-EHT-10005  14.8   18.3   21.4
##  6 18-LDH-BMZ-EHT-10006  14.3   17.8   20.5
##  7 18-LDH-BMZ-EHT-10007  14.3   17.8   21.8
##  8 18-LDH-BMZ-EHT-10008  14.8   18.0   21.9
##  9 18-LDH-BMZ-EHT-10009  15.2   18.1   23.4
## 10 18-LDH-BMZ-EHT-1001   16.4   16.1   22.8
## # ... with 1,190 more rows
```

There are many more other options within *select* that you can explore further by yourself.

Filtering data rows

We have seen how we can manipulate the columns with *select*. Just as the *select* does the columns, *filter* allows you to play with the data rows.

```
# get data rows with more than 0.85 NDVI values on 101 DAS
dat.tib %>%
  select(plot_id,NDVIG.101) %>%
  filter(NDVIG.101 > 0.85)        # can also use >, <, ==, !=, is.na...
```

```
## # A tibble: 32 x 2
##    plot_id              NDVIG.101
##    <chr>                    <dbl>
##  1 18-LDH-BMZ-EHT-10002     0.860
##  2 18-LDH-BMZ-EHT-10018     0.860
##  3 18-LDH-BMZ-EHT-10038     0.860
##  4 18-LDH-BMZ-EHT-10058     0.860
##  5 18-LDH-BMZ-EHT-10082     0.860
##  6 18-LDH-BMZ-EHT-3103      0.870
##  7 18-LDH-BMZ-EHT-4020      0.860
##  8 18-LDH-BMZ-EHT-4022      0.860
##  9 18-LDH-BMZ-EHT-4094      0.860
## 10 18-LDH-BMZ-EHT-4100      0.860
## # ... with 22 more rows
```

filter+stringr: easy manipulattion of strings

```
dat.tib %>%
  select(plot_id,NDVIG.101) %>%
  filter(str_detect(plot_id,'10001'))
```

```
## # A tibble: 1 x 2
##   plot_id              NDVIG.101
##   <chr>                    <dbl>
## 1 18-LDH-BMZ-EHT-10001     0.840
```

```
# there are other str_* functions you may want to check out...
```

Adding new variables with mutate

Anoter handy verb is *mutate* that, as its name suggests, mutates the columns. *Mutate* adds new variables and keeps existing columns as such. Different variations of mutate exist i.e. *mutate_if*, *mutate_all*,* mutate_at*.

```
dat.tib %>%
  mutate(my_plot_id_column_wao_cool=plot_id) %>% #create a new column...
  select(plot_id,contains('my_plot_id'))
```

```
## # A tibble: 1,200 x 2
##    plot_id              my_plot_id_column_wao_cool
##    <chr>                <chr>
##  1 18-LDH-BMZ-EHT-10001 18-LDH-BMZ-EHT-10001
##  2 18-LDH-BMZ-EHT-10002 18-LDH-BMZ-EHT-10002
##  3 18-LDH-BMZ-EHT-10003 18-LDH-BMZ-EHT-10003
##  4 18-LDH-BMZ-EHT-10004 18-LDH-BMZ-EHT-10004
##  5 18-LDH-BMZ-EHT-10005 18-LDH-BMZ-EHT-10005
##  6 18-LDH-BMZ-EHT-10006 18-LDH-BMZ-EHT-10006
##  7 18-LDH-BMZ-EHT-10007 18-LDH-BMZ-EHT-10007
##  8 18-LDH-BMZ-EHT-10008 18-LDH-BMZ-EHT-10008
##  9 18-LDH-BMZ-EHT-10009 18-LDH-BMZ-EHT-10009
## 10 18-LDH-BMZ-EHT-1001  18-LDH-BMZ-EHT-1001
## # ... with 1,190 more rows
```

Mutate can take multiple arguments

```
dat.tib %>%
  mutate(CT.101.scaled= as.vector(scale(CT.101)),
         CT.101.add=CT.101+100,
         CT.101.div=CT.101-100) %>%
  select(contains('CT.101'))
```

```
## # A tibble: 1,200 x 4
##    CT.101 CT.101.scaled CT.101.add CT.101.div
##     <dbl>         <dbl>      <dbl>      <dbl>
##  1   18.7        0.616        119.      -81.3
##  2   18.0       -0.536        118.      -82.0
##  3   18.3       -0.0424       118.      -81.7
##  4   18.2       -0.207        118.      -81.8
##  5   18.3       -0.0424       118.      -81.7
##  6   17.8       -0.865        118.      -82.2
##  7   17.8       -0.865        118.      -82.2
##  8   18.0       -0.536        118.      -82.0
##  9   18.1       -0.372        118.      -81.9
## 10   16.1       -3.66         116.      -83.9
## # ... with 1,190 more rows
```

Notice the original columns are preserved with mutate.

Advance mutate

We often ancounter situations where we have to change multiple column attributes from numeric, character, vector etc. . . Mutate_if can be handy in that stuation

```
dat.tib %>%
  mutate_if(is.numeric,as.character) %>%
  select(contains('CT')) %>%
  glimpse()
```

```
## Observations: 1,200
## Variables: 5
## $ CT.101 <chr> "18.7", "18", "18.3", "18.2", "18.3", "17.8", "17.8", "...
## $ CT.121 <chr> "22.5", "21.6", "22.7", "21.4", "21.4", "20.5", "21.8",...
## $ CT.136 <chr> "24", "24.5", "22.6", "23.6", "26.2", "25.5", "25.7", "...
## $ CT.156 <chr> "31.2", "27", "29.2", "29.8", "29.9", "30.9", "29.2", "...
## $ CT.77  <chr> "14.8", "15.1", "13.5", "14.7", "14.8", "14.3", "14.3",...
```

```
# I would encourage you to also explore other mutate_* family of functions.
```

Correlations with tidyverse

```
dat.tib.corr <- dat.tib %>%
  select(-plot_id) %>%
  corrr::correlate() %>%       # create correlation matrix
  #shave() %>%                 # shave off upper triangle
  fashion()                    # nicely formatted corr table
  #stretch() %>%               # stretch in long format
#View(dat.tib.corr)
```

Reshape data with tidyr

We consider the long format data with unique data rows as tidy (roughly speaking). The *tidyr* package (part of *tidyverse* suite) makes it easier to manipulate data shape.

Lets check out *gather* and *spread* verbs from *tidyr* package

since our original data is in wide format. First, we will try to gather it into long format

```
dat.tib %>%
  gather(key='trait.DAS',value='phenotype_value',CT.101:NDVIG.78)
```

```
## # A tibble: 13,200 x 3
##    plot_id              trait.DAS phenotype_value
##    <chr>                <chr>               <dbl>
##  1 18-LDH-BMZ-EHT-10001 CT.101               18.7
##  2 18-LDH-BMZ-EHT-10002 CT.101               18.0
##  3 18-LDH-BMZ-EHT-10003 CT.101               18.3
##  4 18-LDH-BMZ-EHT-10004 CT.101               18.2
##  5 18-LDH-BMZ-EHT-10005 CT.101               18.3
##  6 18-LDH-BMZ-EHT-10006 CT.101               17.8
##  7 18-LDH-BMZ-EHT-10007 CT.101               17.8
##  8 18-LDH-BMZ-EHT-10008 CT.101               18.0
##  9 18-LDH-BMZ-EHT-10009 CT.101               18.1
## 10 18-LDH-BMZ-EHT-1001  CT.101               16.1
## # ... with 13,190 more rows
```

Similarly we can spread the data back to original format with spread

```
dat.tib %>%
  gather(key='trait.DAS',value='phenotype_value',CT.101:NDVIG.78) %>%
  spread(key = trait.DAS,value = phenotype_value)
```

```
## # A tibble: 1,200 x 12
##    plot_id  CT.101 CT.121 CT.136 CT.156 CT.77 GRYLD.NA NDVIG.101 NDVIG.120
##    <chr>     <dbl>  <dbl>  <dbl>  <dbl> <dbl>    <dbl>     <dbl>     <dbl>
##  1 18-LDH-~   18.7   22.5   24.0   31.2  14.8     8.37     0.840     0.810
##  2 18-LDH-~   18.0   21.6   24.5   27.0  15.1     7.16     0.860     0.860
##  3 18-LDH-~   18.3   22.7   22.6   29.2  13.5     8.07     0.830     0.840
##  4 18-LDH-~   18.2   21.4   23.6   29.8  14.7     6.91     0.830     0.820
##  5 18-LDH-~   18.3   21.4   26.2   29.9  14.8     6.59     0.800     0.820
##  6 18-LDH-~   17.8   20.5   25.5   30.9  14.3     6.84     0.800     0.840
##  7 18-LDH-~   17.8   21.8   25.7   29.2  14.3     6.75     0.810     0.820
##  8 18-LDH-~   18.0   21.9   24.0   28.8  14.8     8.02     0.780     0.800
##  9 18-LDH-~   18.1   23.4   26.7   30.9  15.2     8.58     0.820     0.810
## 10 18-LDH-~   16.1   22.8   26.4   27.6  16.4     6.38     0.780     0.770
## # ... with 1,190 more rows, and 3 more variables: NDVIG.135 <dbl>,
## #   NDVIG.154 <dbl>, NDVIG.78 <dbl>
```

Lets wrap this section with capturing the ouput in a data object

```
dat.tib.long <- dat.tib %>%
  gather(key='trait.DAS',value='phenotype_value',CT.101:NDVIG.78)
```

Split-aggregate data rows

The group_by is a great tool to perform split-style actions on data columns, especially when combined with summarize. Lets divide our data by trait.DAS column and perform summary function on phenotype_value column.

```
dat.tib.long %>%
  group_by(trait.DAS) %>%   #split by trait.DAS
  summarize(phen.val.mean=mean(phenotype_value,na.rm = T)) #summary means per trait.DAS
```

```
## # A tibble: 11 x 2
##    trait.DAS phen.val.mean
##    <chr>             <dbl>
##  1 CT.101             18.3
##  2 CT.121             22.4
##  3 CT.136             24.5
##  4 CT.156             29.4
##  5 CT.77              15.1
##  6 GRYLD.NA            7.58
##  7 NDVIG.101           0.828
##  8 NDVIG.120           0.827
##  9 NDVIG.135           0.743
## 10 NDVIG.154           0.451
## 11 NDVIG.78            0.782
```

Now let's put *select*, *mutate*, *group_by* and *summarize* together. BTW just like *mutate_*, *summarize* also has summarize_at, summarize_all functions.

```
dat.tib.long %>%
  filter(!is.na(phenotype_value)) %>%
  group_by(trait.DAS) %>%
  summarize(phen.val.mean=mean(phenotype_value),
            num.entries=n())
```

```
## # A tibble: 11 x 3
```

```
##    trait.DAS phen.val.mean num.entries
##    <chr>             <dbl>       <int>
##  1 CT.101           18.3          1199
##  2 CT.121           22.4          1200
##  3 CT.136           24.5          1200
##  4 CT.156           29.4          1200
##  5 CT.77            15.1          1200
##  6 GRYLD.NA          7.58         1200
##  7 NDVIG.101         0.828        1200
##  8 NDVIG.120         0.827        1200
##  9 NDVIG.135         0.743        1200
## 10 NDVIG.154         0.451        1200
## 11 NDVIG.78          0.782        1200
```

Summarize also takes other summary functions like min, max, standard deviation etc. And most importantly we can apply our own custom functions with summarize. There is only thing to remember when you supply your custom function to *summarize*: output should be able to take a vector and creates single element vector.

Apply a custom coefficient of variation function

```
# my CV function
myCVFunc <- function(x){sd(x)/mean(x)*100}
# supply our custom function to summarize
dat.tib.long %>%
  filter(!is.na(phenotype_value)) %>%
  group_by(trait.DAS) %>%
  summarize(phen.val.mean=mean(phenotype_value),
            phen.val.CV=myCVFunc(phenotype_value),
            num.entries=n())
```

```
## # A tibble: 11 x 4
##    trait.DAS phen.val.mean phen.val.CV num.entries
##    <chr>             <dbl>       <dbl>       <int>
##  1 CT.101           18.3         3.32         1199
##  2 CT.121           22.4         5.02         1200
##  3 CT.136           24.5         4.33         1200
##  4 CT.156           29.4         4.65         1200
##  5 CT.77            15.1         6.49         1200
##  6 GRYLD.NA          7.58        7.86         1200
##  7 NDVIG.101         0.828       2.33         1200
##  8 NDVIG.120         0.827       2.26         1200
##  9 NDVIG.135         0.743       3.98         1200
## 10 NDVIG.154         0.451      19.9          1200
## 11 NDVIG.78          0.782       3.53         1200
```

Other useful data manipulation functions

I encourage you to also check *rename, arrange, separate, unite, distinct* verbs for advanced data wrangling....

Combining two datasets with joins

More often with breeding datasets we need to combine multiple files from field experiments. For example, there are separate plot-layout and phenotype data files. We end up joining multiple files originating from field with excel. But with these functions you can join dataframes with very large number of variable columns very easy!

Left-join example

Load plot attribute data from file

```
## Observations: 1,200
## Variables: 24
## $ plot_id       <chr> "18-LDH-BMZ-EHT-10001", "18-LDH-BMZ-EHT-10002", ...
## $ iyear         <int> 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, ...
## $ ilocation     <chr> "LDH", "LDH", "LDH", "LDH", "LDH", "LDH", "LDH",...
## $ itrial        <chr> "BMZ", "BMZ", "BMZ", "BMZ", "BMZ", "BMZ", "BMZ",...
## $ icondition    <chr> "EHT", "EHT", "EHT", "EHT", "EHT", "EHT", "EHT",...
## $ plot_no       <int> 10001, 10002, 10003, 10004, 10005, 10006, 10007,...
## $ trial         <chr> "SABWGPYT10", "SABWGPYT10", "SABWGPYT10", "SABWG...
## $ seed_source   <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ planting_date <chr> "10/23/2017", "10/23/2017", "10/23/2017", "10/23...
## $ site          <chr> "Ludhiana", "Ludhiana", "Ludhiana", "Ludhiana", ...
## $ year          <int> 2018, 2018, 2018, 2018, 2018, 2018, 2018, 2018, ...
## $ location      <chr> "Ludhiana", "Ludhiana", "Ludhiana", "Ludhiana", ...
## $ cycle         <chr> "SA17-18", "SA17-18", "SA17-18", "SA17-18", "SA1...
## $ conditions    <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ rep           <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ block         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ subblock      <int> 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 3, 3, 3, 3, 3, ...
## $ col           <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 10, 10, 9, 8, 7, 6...
## $ row           <int> 109, 109, 109, 109, 109, 109, 109, 109, 109, 1, ...
## $ entry         <int> 10001, 10002, 10003, 10004, 10005, 10006, 10007,...
## $ purpose       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ gid           <int> 7627605, 304660, 7627645, 7627648, 7627651, 7627...
## $ tid           <int> 46213, 46213, 46213, 46213, 46213, 46213, 46213,...
## $ occ           <int> 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, ...
```

Join plots and phenotype data together

```
dat.joined <- dat.plots %>%
  left_join(.,dat.tib.long,by='plot_id')
glimpse(dat.joined)
```

```
## Observations: 13,200
## Variables: 26
## $ plot_id         <chr> "18-LDH-BMZ-EHT-10001", "18-LDH-BMZ-EHT-10001"...
## $ iyear           <int> 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18...
## $ ilocation       <chr> "LDH", "LDH", "LDH", "LDH", "LDH", "LDH", "LDH...
## $ itrial          <chr> "BMZ", "BMZ", "BMZ", "BMZ", "BMZ", "BMZ", "BMZ...
## $ icondition      <chr> "EHT", "EHT", "EHT", "EHT", "EHT", "EHT", "EHT...
## $ plot_no         <int> 10001, 10001, 10001, 10001, 10001, 10001, 1000...
## $ trial           <chr> "SABWGPYT10", "SABWGPYT10", "SABWGPYT10", "SAB...
## $ seed_source     <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA...
## $ planting_date   <chr> "10/23/2017", "10/23/2017", "10/23/2017", "10/...
```

```
## $ site            <chr> "Ludhiana", "Ludhiana", "Ludhiana", "Ludhiana"...
## $ year            <int> 2018, 2018, 2018, 2018, 2018, 2018, 2018, 2018...
## $ location        <chr> "Ludhiana", "Ludhiana", "Ludhiana", "Ludhiana"...
## $ cycle           <chr> "SA17-18", "SA17-18", "SA17-18", "SA17-18", "S...
## $ conditions      <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA...
## $ rep             <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ block           <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA...
## $ subblock        <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ col             <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2...
## $ row             <int> 109, 109, 109, 109, 109, 109, 109, 109, 109, 1...
## $ entry           <int> 10001, 10001, 10001, 10001, 10001, 10001, 1000...
## $ purpose         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA...
## $ gid             <int> 7627605, 7627605, 7627605, 7627605, 7627605, 7...
## $ tid             <int> 46213, 46213, 46213, 46213, 46213, 46213, 4621...
## $ occ             <int> 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4...
## $ trait.DAS       <chr> "CT.101", "CT.121", "CT.136", "CT.156", "CT.77...
## $ phenotype_value <dbl> 18.700, 22.500, 24.000, 31.200, 14.800, 8.372,...
```

Get rid of extra columns and change column attributes

```
dat.joined <- dat.joined %>%
  select(plot_id,trial,rep,subblock,col,row,entry,
         gid,trait.DAS,phenotype_value) %>%
  mutate_at(vars(rep:gid), as.character)
glimpse(dat.joined)
```

```
## Observations: 13,200
## Variables: 10
## $ plot_id         <chr> "18-LDH-BMZ-EHT-10001", "18-LDH-BMZ-EHT-10001"...
## $ trial           <chr> "SABWGPYT10", "SABWGPYT10", "SABWGPYT10", "SAB...
## $ rep             <chr> "1", "1", "1", "1", "1", "1", "1", "1", "1", "...
## $ subblock        <chr> "1", "1", "1", "1", "1", "1", "1", "1", "1", "...
## $ col             <chr> "1", "1", "1", "1", "1", "1", "1", "1", "1", "...
## $ row             <chr> "109", "109", "109", "109", "109", "109", "109...
## $ entry           <chr> "10001", "10001", "10001", "10001", "10001", "...
## $ gid             <chr> "7627605", "7627605", "7627605", "7627605", "7...
## $ trait.DAS       <chr> "CT.101", "CT.121", "CT.136", "CT.156", "CT.77...
## $ phenotype_value <dbl> 18.700, 22.500, 24.000, 31.200, 14.800, 8.372,...
```

Calculating heritability the 'tidy' way

Generally, it is a good practice to keep your function definitions and source calls at the beginning of code file. But for the sake of our tutorial workflow I am keeping it here.

```
# function to calc heritability per each trait (trial as main factor)
calcH2Trialfunc <- function(dat) {
  if (sum(!is.na(dat$phenotype_value))/dim(dat)[1] > 0.9) {
      v = data.frame(VarCorr(lmer(phenotype_value ~ 0 + (1|gid) + (1|trial:rep) + (1|trial:rep:subblock)
      data.frame(H2=round(v[1,4]/(v[1,4]+(v[4,4]/2)),2), vG=v[1,4], vE=v[4,4])
  } else {
      data.frame(H2=NA, vG=NA, vE=NA)
  }
}
# This is the basic mixed model for calculating broad-sense heritability on an entry-mean basis in my e
```

Now lets apply the heritability/repeatability function by grouping with trait.DAS

```
dat.h2 <- dat.joined %>%
  group_by(trait.DAS) %>%
  do(calcH2Trialfunc(.)) %>%  # 'do' is summarize equivalent but for dataframes
  ungroup()                   # ungroup the dataframe output
# lets take a look at our heritability data
dat.h2
```

```
## # A tibble: 11 x 4
##    trait.DAS     H2       vG       vE
##    <chr>      <dbl>    <dbl>    <dbl>
##  1 CT.101    0.0700 0.00695  0.192
##  2 CT.121    0.110  0.0287   0.480
##  3 CT.136    0.0600 0.0295   0.884
##  4 CT.156    0.530  0.464    0.818
##  5 CT.77     0.     0.       0.584
##  6 GRYLD.NA  0.570  0.118    0.181
##  7 NDVIG.101 0.560  0.000102 0.000159
##  8 NDVIG.120 0.650  0.000145 0.000154
##  9 NDVIG.135 0.800  0.000509 0.000252
## 10 NDVIG.154 0.790  0.00434  0.00227
## 11 NDVIG.78  0.440  0.000134 0.000348
```

Our heritability data has: first column 'trait.DAS' (our grouping variable); second column heritability; third and fourth the genotypic and residual variances, respectively.

Plotting the H2 data with ggplot2

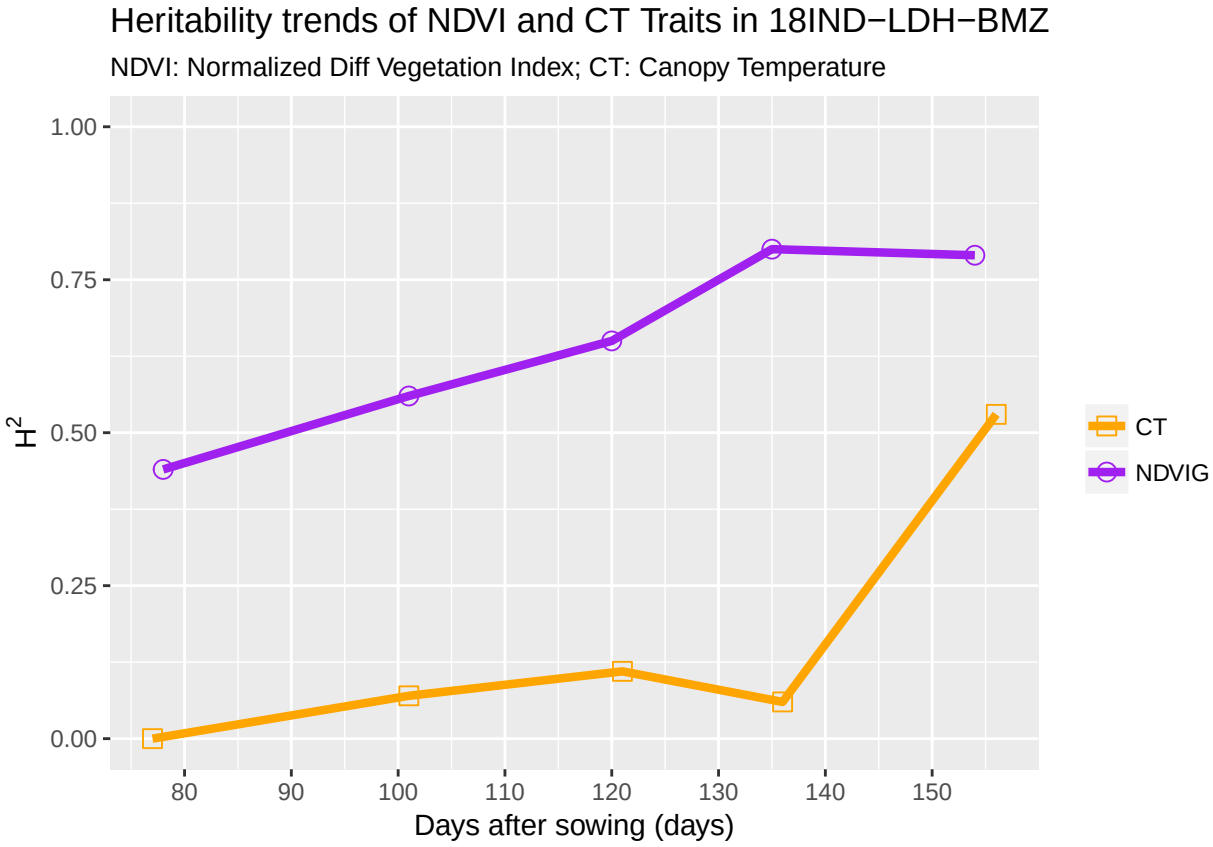
Split columns

```
dat.h2.tidy <- dat.h2 %>%
  separate(trait.DAS,into=c('trait_id','DAS'),remove=F)
# take a look at data
dat.h2.tidy
```

```
## # A tibble: 11 x 6
##    trait.DAS trait_id DAS       H2       vG       vE
##    <chr>     <chr>    <chr>  <dbl>    <dbl>    <dbl>
##  1 CT.101    CT       101   0.0700 0.00695  0.192
##  2 CT.121    CT       121   0.110  0.0287   0.480
##  3 CT.136    CT       136   0.0600 0.0295   0.884
##  4 CT.156    CT       156   0.530  0.464    0.818
##  5 CT.77     CT       77    0.     0.       0.584
##  6 GRYLD.NA  GRYLD    NA    0.570  0.118    0.181
##  7 NDVIG.101 NDVIG    101   0.560  0.000102 0.000159
##  8 NDVIG.120 NDVIG    120   0.650  0.000145 0.000154
##  9 NDVIG.135 NDVIG    135   0.800  0.000509 0.000252
## 10 NDVIG.154 NDVIG    154   0.790  0.00434  0.00227
## 11 NDVIG.78  NDVIG    78    0.440  0.000134 0.000348
```

Now we can proceed with plotting

```
dat.h2.tidy %>%
  filter(trait_id %in% c('CT','NDVIG')) %>%   #keep only NDVI & CT for plotting
  ggplot(aes(y=H2, x=as.numeric(DAS),
             colour = trait_id,
             linetype=trait_id,
             shape=factor(trait_id)),
         data = .) +
  geom_line(size = 1.5) +
  ylim(0.00,1.00) +
  scale_x_continuous(breaks = seq(60,160,10)) +
  labs(title = "Heritability trends of NDVI and CT Traits in 18IND-LDH-BMZ",subtitle= 'NDVI: Normalized
  labs(x = "Days after sowing (days)", y = expression(H^2)) +
  scale_colour_manual("",values=c("orange","purple")) +
  scale_linetype_manual("", values=c(1,1)) +
  scale_shape_manual("", values=c(0:1,2)) +
  geom_point(size = 3)
```



This concludes our session on tidy data analyis. In coming sessions we will explore other data modeling and management related topics. Please feel free to ask if you have any questions.

```
sessionInfo()
```

```
## R version 3.4.0 (2017-04-21)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS  10.13.4
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] bindrcpp_0.2    corrr_0.2.1     lme4_1.1-13     Matrix_1.2-10
##  [5] forcats_0.3.0   stringr_1.2.0   dplyr_0.7.4     purrr_0.2.4
##  [9] readr_1.1.1     tidyr_0.8.0     tibble_1.4.2    ggplot2_2.2.1
## [13] tidyverse_1.2.1 pacman_0.4.6
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_0.2.4 reshape2_1.4.2   splines_3.4.0    haven_1.1.1
##  [5] lattice_0.20-35  colorspace_1.3-2 htmltools_0.3.6  yaml_2.1.14
##  [9] utf8_1.1.3       rlang_0.2.0      nloptr_1.0.4     pillar_1.2.1
## [13] foreign_0.8-68   glue_1.2.0       modelr_0.1.1     readxl_1.0.0
## [17] bindr_0.1        plyr_1.8.4       munsell_0.4.3    gtable_0.2.0
## [21] cellranger_1.1.0 rvest_0.3.2      psych_1.8.3.3    evaluate_0.10
## [25] labeling_0.3     knitr_1.20       parallel_3.4.0   broom_0.4.4
## [29] Rcpp_0.12.16     scales_0.4.1     backports_1.1.0  jsonlite_1.5
## [33] mnormt_1.5-5     hms_0.4.2        digest_0.6.12    stringi_1.1.5
## [37] grid_3.4.0       rprojroot_1.2    cli_1.0.0        tools_3.4.0
## [41] magrittr_1.5     lazyeval_0.2.0   crayon_1.3.4     pkgconfig_2.0.1
## [45] MASS_7.3-47      xml2_1.2.0       lubridate_1.7.3  minqa_1.2.4
## [49] assertthat_0.2.0 rmarkdown_1.9    httr_1.3.1       rstudioapi_0.7
## [53] R6_2.2.2         nlme_3.1-131     compiler_3.4.0
```