# Visualizing Statistics and Regressions from a Spreadsheet using R

## Contents



This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

These notebooks are typically this is designed to create a pleasing viewing environment of data analysis that allows you to include figures, text, links, etc. so that your work is better understood and can be reproduced and used with confidence.

The source code for this R notebook (Rmd suffixed files), when stored as web pages (html files), can be downloaded by clicking the button at the top of the page.

If viewing the source code in R Studio, try executing each R "chunk" by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

**Warning. Typos are *Legion*!**

## 1. Introduction

When you're in MATH 381 (Intro to Probability and Stats) you'll get a taste of R. R is an open-source statistical package build off of an earlier generation of commercial.

The goal here is to demonstrate cracking open an excel spreadsheet in R and calculate some basic stats, create various plots to view the statistics, and finally, do some linear and multivariate regression

Another goal here is to show off some of R's features. R is a very powerful tool. When translating "powerful" from computereese to any frustrated human dialect, that means "steep learning curve." It's also a community-supported environment. When translating "powerful" from computereese to any overscheduled human dialect, that means "there are LOTS of people donating packages and libraries to R." Some have evolved to be a standard in the community. Others are highly specialized for a given discipline (but have one or two items that people outside their user communities find handy.)

But don't let that intimidate you. Once you learn one language you can slowly pick up more. Also with this demo we aren't going to get to to be an R guru in a day.

If you want a good stepping off point to learn R I'd recommend some of the resources at Data Camp which have some free starter tutorials for R.

# 2. Loading the Libraries

To work with R we will first have to load some libraries. This is like in C where you have the #include statement to do things like raise things to powers and stuff like that.

Some of these libraries or "packages" come with R. Others will have to be installed. Here are the ones we are using for this exercise.

Also in this exercise, we're going to use the tidyverse set of packages. Tidyverse is a set of co-developed tools for data science in R. This is the new big thing in R and is widely used so we are just going to jump in here. SD Mines has a course beyond Engineering Stats, MATH 443/543 (Data Analysis) that leverages this set of packages.

- Install Us First
  - tidyverse : Set of commonly-used Data Science packages for R that it can install and load all at once. In the long-run you probably also want to install the tidyverse package suite anyway. For this exercise this will include…
    - ggplot2 : Create Elegant Data Visualizations Using the Grammar of Graphics
    - tibble : Simple Data Frames
    - tidyr : Tools for shepherding data in data frames.
    - readr : Read Rectangular Text Data
    - purr : Functional Programming Tools
    - dplyr : A grammar of data manipulation
    - stringr : Simple, Consistent Wrappers for Common String Operations
    - forcats : Tools for Working with Categorical Variables (Factors)
  - readxl : also part of the tidyverse package suite for reading traditional excel spreadsheets.

  - moderndive : Tidyverse-Friendly Introductory Linear Regression
- This should come with R's core install, if not install 'em.
  - MASS : Has a lot of resources for regression.
- This doesn't come with R's core install so install that one…
  - moments : This has a load of good stuff for data analysis and plotting, more than you will need here, but get it anyway.
- This is a nice contributed library that lets us make pretty statistics tables. It was written for ecological applications but it's still pretty handy for looking at concrete
  - pastecs: Package for Analysis of Space-Time Ecological Series
- Another nice contributed library that makes matrices of correlation coefficients look pretty (and graphically informative).
  - corrplot Visualization of a Correlation Matrix

- While not officially needed for this activity but I'll demonstrate how units can be used in R in this example
  - udunits2 Provides simple bindings to Unidata's udunits library for unit conversions (will be demonstrating but not explicity needing it here)
  - units Provides Measurement Units for R Vectors

```
# Tidyverse Handling Libraries

suppressMessages(library(package = "tidyverse"))   # main tidyverse suite
library(package = "readxl")     # Read Excel Files
library(package = "moderndive") # regression support

# Statistics Libraries

library(package = "moments")   # Moments, cumulants, skewness, kurtosis and related tests
library(package = "MASS", warn.conflicts=FALSE)      # Support Functions and Datasets for Venables &

# Extra Graphics Libraries

library(package = "corrplot")  # Visualization of a Correlation Matrix
```

```
## corrplot 0.92 loaded
```

```
# Data Processing Libraries

library(package = "pastecs")   # Package for Analysis of Space-Time Ecological Series
```

```
##
## Attaching package: 'pastecs'

## The following objects are masked from 'package:dplyr':
##
##     first, last

## The following object is masked from 'package:tidyr':
##
##     extract
```

```
library(package = "udunits2")  # Unit Conversion Support
```

```
## udunits system database read from /Users/wjc/Library/R/arm64/4.2/library/udunits2/share/udunits2.xml
```

```
library(package = "units")     # Measurement Units for R Vectors
```

```
## udunits database from /Users/wjc/Library/R/arm64/4.2/library/udunits2/share/udunits2.xml
```

## 3. Cracking a Spreadsheet

The spreadsheet example below is a more complicated than what you hopefully have.

The original data set is from a set of papers on Concrete by I-Cheng Yeh

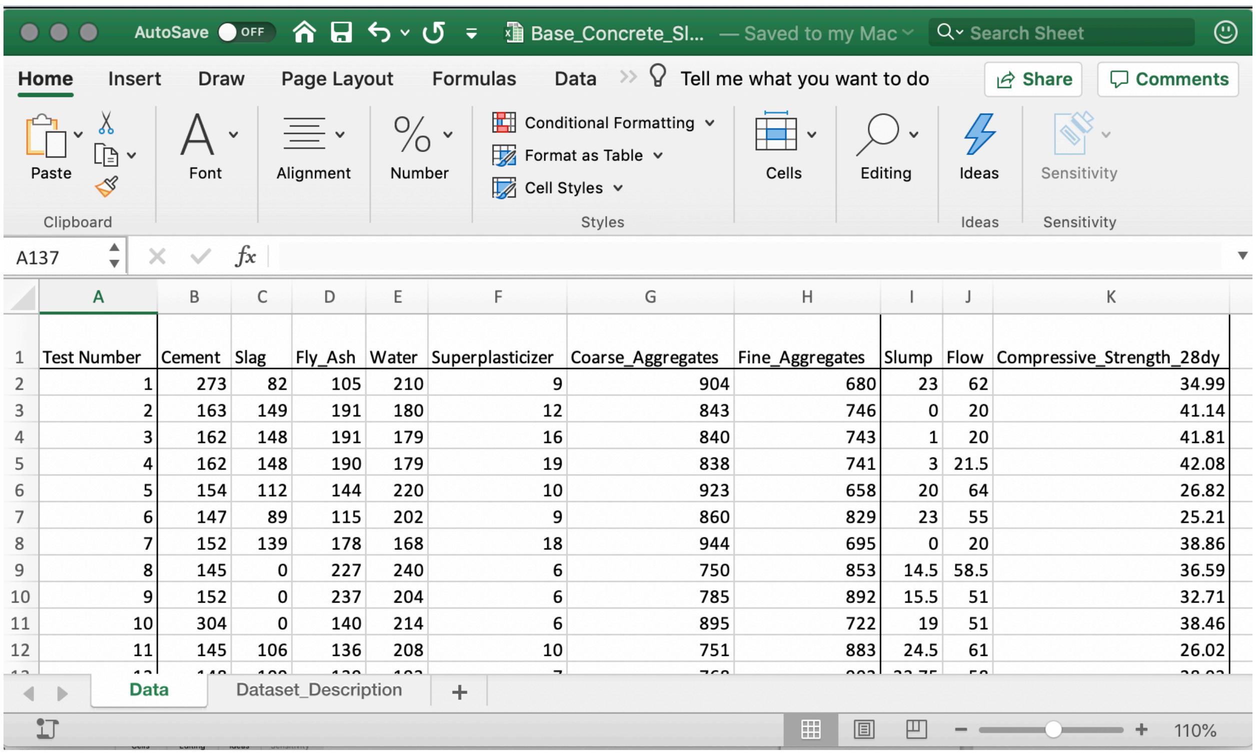
- Yeh, I-Cheng, "Modeling slump of concrete with fly ash and superplasticizer," *Computers and Concrete*, **5**(6), 559-572, 2008. doi: 10.12989/cac.2008.5.6.559.
- Yeh, I-Cheng, "Simulation of concrete slump using neural networks," *Construction Materials*, **162**(1), 11-18, 2009. doi: 10.1680/coma.2009.162.1.11

- Yeh, I-Cheng, "Prediction of workability of concrete using design of experiments for mixtures," *Computers and Concrete*, **5**(1), 1-20, 2008. doi: 10.12989/cac.2008.5.1.001
- Yeh, I-Cheng, "Modeling slump flow of concrete using second-order regressions and artificial neural networks," *Cement and Concrete Composites*, **29**(6), 474-480, 2007. doi: 10.1016/j.cemconcomp.2007.02.001
- Yeh, I-Cheng, "Exploring concrete slump model using artificial neural networks," *ASCE J. of Computing in Civil Engineering*, **20**(3), 217-221, 2006. doi: 10.1061/(ASCE)0887-3801(2006)20:3(217)

and is kept at the UC-Irvine Machine Learning Repository.

It can be found here at http://kyrill.ias.sdsmt.edu/cee_284/Base_Concrete_Slump_Test_for_R.xlsx

The relevant page and screenshot is below. For drama-free R import you are probably best off keeping a page on your spreadsheet file that is very simple, with numbers going down, and a single line for Row-1 with the headers of each column. If you want to get fancy on other pages that you'd turn in as tables in reports, you can do that on another spreadsheet page.



| Test Number | Cement | Slag | Fly_Ash | Water | Superplasticizer | Coarse_Aggregates | Fine_Aggregates | Slump | Flow | Compressive_Strength_28dy |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 273 | 82 | 105 | 210 | 9 | 904 | 680 | 23 | 62 | 34.99 |
| 2 | 163 | 149 | 191 | 180 | 12 | 843 | 746 | 0 | 20 | 41.14 |
| 3 | 162 | 148 | 191 | 179 | 16 | 840 | 743 | 1 | 20 | 41.81 |
| 4 | 162 | 148 | 190 | 179 | 19 | 838 | 741 | 3 | 21.5 | 42.08 |
| 5 | 154 | 112 | 144 | 220 | 10 | 923 | 658 | 20 | 64 | 26.82 |
| 6 | 147 | 89 | 115 | 202 | 9 | 860 | 829 | 23 | 55 | 25.21 |
| 7 | 152 | 139 | 178 | 168 | 18 | 944 | 695 | 0 | 20 | 38.86 |
| 8 | 145 | 0 | 227 | 240 | 6 | 750 | 853 | 14.5 | 58.5 | 36.59 |
| 9 | 152 | 0 | 237 | 204 | 6 | 785 | 892 | 15.5 | 51 | 32.71 |
| 10 | 304 | 0 | 140 | 214 | 6 | 895 | 722 | 19 | 51 | 38.46 |
| 11 | 145 | 106 | 136 | 208 | 10 | 751 | 883 | 24.5 | 61 | 26.02 |

Figure 1: Concrete Spreadsheet Screenshot

To crack open the spreadsheet we will want to use the read_excel function.

You can read the spreadsheet from a local drive or from a website.

```
# you will need the full path to the file you are using (either online or locally on your disk)

# The if else block should query your machine to determine which operating system.
#  if you are not bi-platform, you likely don't need this.

if(.Platform$OS.type == "windows") {
  # Windows
  spreadsheet_name      = "%HOMEPATH%/Downloads/Base_Concrete_Slump_Test_for_R.xlsx"
} else {
```

```
    # Unix (Linux, MacOS, Solaris)
    spreadsheet_name     = "~/Downloads/Base_Concrete_Slump_Test_for_R.xlsx"
  }


  # I am keeping a copy of these spreadsheet at the URL below.  It can be downloaded automatically
  #   and then loaded.  We can also discretely delete it when done.

      spreadsheet_url = "http://kyrill.ias.sdsmt.edu/wjc/eduresources/Base_Concrete_Slump_Test_for_R.xls

      download.file(url      =   spreadsheet_url, # URL location
                    destfile = spreadsheet_name) # local downloaded location

      remove(spreadsheet_url) # clean up variables

  # this command will read the file

  concrete = read_excel(path      = spreadsheet_name,  # remove spreadsheet location
                        sheet     = "Data",            # page of spreadsheet
                        col_names = TRUE)              # first row are the column headers


  # clean up your hard drive!  Don't be like me!

  if(.Platform$OS.type == "windows") {
    # Windows
    system(str_c("DEL   ",
                 spreadsheet_name,
                 sep=""))
    } else {
    # Unix (Linux, MacOS, Solaris)
    system(str_c("rm -v  ",
                 spreadsheet_name,
                 sep=""))
      }

  remove(spreadsheet_name) # clean up variables
```

With the data read in we can now look at the table of the data. This looks much nicer when working in R Notebooks instead of Plain Ordinary R.

```
# Print data frame
colnames(concrete)[1] = "Test_Number"
print(concrete)
```

```
## # A tibble: 103 x 11
##    Test_Number Cement  Slag Fly_Ash Water Superpla~1 Coars~2 Fine_~3 Slump  Flow
##          <dbl>  <dbl> <dbl>   <dbl> <dbl>      <dbl>   <dbl>   <dbl> <dbl> <dbl>
##  1          14    354     0       0   234          6     959     691  17    54
##  2          42    154   141     181   234         11     797     683  23    65
##  3           4    162   148     190   179         19     838     741   3    21.5
##  4          76    149   109     139   193          6     892     780  23.5  58.5
##  5          71    276    90     116   180          9     870     768   0    20
##  6          31    321     0     164   190          5     870     774  24    60
```

```
##  7           59     143    131      168    217           6      891      672  25     69
##  8           47     280     92      118    207           9      883      679  25.5   64
##  9           41     145    177      227    209          11      752      715   2.5   20
## 10           32     349      0      178    230           6      785      721  20     68.5
## # ... with 93 more rows, 1 more variable: Compressive_Strength_28dy <dbl>, and
## #   abbreviated variable names 1: Superplasticizer, 2: Coarse_Aggregates,
## #   3: Fine_Aggregates
```

**Extra: Units (not part of this exercise but it's a nifty tangent)**

*Dang. I like units. I don't see any. I'm anal and have learned that adding as much descriptive data early on in processing your data set will make people (and most importantly, yourself) not hate you at a later date. So I am adding them here with the set_units function. This will add units as an attribute.

Units don't work with everything and you should probably keep a copy of your original un-unitted data frame.

```
# first we clone our data frame

concrete_units = concrete

concrete_units$Cement                    = set_units(x     = concrete_units$Cement,
                                                     value = "kg m-3")

concrete_units$Slag                      = set_units(x     = concrete_units$Slag,
                                                     value = "kg m-3")

concrete_units$Fly_Ash                   = set_units(x     = concrete_units$Fly_Ash,
                                                     value = "kg m-3")

concrete_units$Water                     = set_units(x     = concrete_units$Water,
                                                     value = "kg m-3")

concrete_units$Superplasticizer          = set_units(x     = concrete_units$Superplasticizer,
                                                     value = "kg m-3")

concrete_units$Coarse_Aggregates         = set_units(x     = concrete_units$Coarse_Aggregates,
                                                     value = "kg m-3")

concrete_units$Fine_Aggregates           = set_units(x     = concrete_units$Fine_Aggregates,
                                                     value = "kg m-3")

concrete_units$Slump                     = set_units(x     = concrete_units$Slump,
                                                     value = "cm")

concrete_units$Flow                      = set_units(x     = concrete_units$Flow,
                                                     value = "cm")

concrete_units$Compressive_Strength_28dy = set_units(x     = concrete_units$Compressive_Strength_28dy,
                                                     value = "MPa")

print(concrete_units)
```

```
## # A tibble: 103 x 11
##    Test_Number   Cement   Slag Fly_Ash Water Super~1 Coars~2 Fine_~3 Slump Flow
##          <dbl> [kg/m^3] [kg/m~ [kg/m^~ [kg/~ [kg/m^~ [kg/m^~ [kg/m^~  [cm] [cm]
```

```
## 1           14      354      0       0    234        6      959      691  17    54
## 2           42      154    141     181    234       11      797      683  23    65
## 3            4      162    148     190    179       19      838      741   3    21.5
## 4           76      149    109     139    193        6      892      780  23.5 58.5
## 5           71      276     90     116    180        9      870      768   0    20
## 6           31      321      0     164    190        5      870      774  24    60
## 7           59      143    131     168    217        6      891      672  25    69
## 8           47      280     92     118    207        9      883      679  25.5 64
## 9           41      145    177     227    209       11      752      715   2.5 20
## 10          32      349      0     178    230        6      785      721  20    68.5
## # ... with 93 more rows, 1 more variable: Compressive_Strength_28dy [MPa], and
## #   abbreviated variable names 1: Superplasticizer, 2: Coarse_Aggregates,
## #   3: Fine_Aggregates
```

If you click in the Global Environment Box, those units aren't arbitrary strings. They are listed as numerators, denominators and also the way in which squares, etc., are archived are explicit.

Better Still, the same command of set_units when applied to a variable that already has units will convert it. This is nice when moving between SI units, USCS units. [If you are going to be cheeky and try the Furlong/Firkin/Fortnight system (FFF), sorry to disappoint, that while the udunits2 package in R recognizes all three units, it recognizes firkins as a volume measure (which is really is) and not the mass measure based on density of water.]

Example here:

```r
# a little unit play!

strength_in_psi = set_units(x     = concrete_units$Compressive_Strength_28dy,
                            value = "psi")

print(concrete_units$Compressive_Strength_28dy[1])
```

```
## 33.91 [MPa]
```

```r
print(strength_in_psi[1])
```

```
## 4918.23 [psi]
```

```r
# Ok now I'm being silly but so were the package developers.
# Blame them.
# (Once again, I can't do official FFF units)

cement_in_slug_per_cu3 = set_units(x     = concrete_units$Cement,
                                   value = "slugs/furlongs^3")

print(concrete_units$Cement[1])
```

```
## 354 [kg/m^3]
```

```r
print(cement_in_slug_per_cu3[1])
```

```
## 197474579 [slugs/furlongs^3]
```

```r
# cleaning-up our horseplay..

remove(strength_in_psi)
remove(cement_in_slug_per_cu3)

remove(concrete_units)
```

Caveat! As useful as this can be, know this: Not all R functions play nice with units or other "attributes" in data frames Some of the plotting routines and linear regression routines below will work with this.

If you need your units and want to minimize "messy" code in R when it conflicts any given function. You can later strip out units by using the as.numeric() function

# 4. Some Basic Statistics and Traditional Single Variable Plots

Lets start with some basic statistics and plotting of them.

## 4.1. The "classic" stats

Let's get the mom-and-apple-pie stats for Concrete That second argument allows you to deal with missing data.

```
# statistics for cement


print(str_c("    Mean Cement : ",
            mean(x     = concrete$Cement, # variable to crunch
                 na.rm =            TRUE) # ignore msissing data
            ))
```

```
## [1] "    Mean Cement : 229.894174757282"
```

```
print(str_c("   Stdev Cement : ",
            sd(x     = concrete$Cement, # variable to crunch
               na.rm =            TRUE) # ignore msissing data
            ))
```

```
## [1] "   Stdev Cement : 78.8772300268858"
```

```
print(str_c("Skewness Cement : ",
            skewness(x     = concrete$Cement, # variable to crunch
                     na.rm =            TRUE) # ignore msissing data
            ))
```

```
## [1] "Skewness Cement : 0.143018080025135"
```

```
print(str_c("Kurtosis Cement : ",
            kurtosis(x     = concrete$Cement, # variable to crunch
                     na.rm =            TRUE) # ignore msissing data
            ))
```

```
## [1] "Kurtosis Cement : 1.33448397363582"
```

OK this is a little clunky. It would be nice if someone somewhere made a support library for R that will make nice tables of statistics.

In this case Vive La France! A team from French Research Institute for Exploitation of the Sea thought the same question and as is often the case for the R community not only drafted a set of tools to do this, *and* made it public.

Here we ware using their stat.desc function.

This will hopefully give people wanting to make basic tables maximum satisfaction with minimal effort.

```
# Plot a statistics table -- all the classics nice and handy and pretty.

options(digits=2) # this simply set the decimal count in the table to be created below
```

```r
                    # this particular function creates the table in scientific notation

concrete_statistics = stat.desc(x     = concrete,  # data frame
                                basic =     TRUE,  # includes counts and extremes
                                desc =      TRUE,  # include classic stats (mean etc)
                                norm =      TRUE,  # include normal dist stats (skewness etc)
                                p     =     0.95)  # use 95% confidence limits


print(concrete_statistics)
```

```
##              Test_Number   Cement     Slag  Fly_Ash    Water Superplasticizer
## nbr.val          1.0e+02  1.0e+02  1.0e+02  1.0e+02  1.0e+02          1.0e+02
## nbr.null         0.0e+00  0.0e+00  2.6e+01  2.0e+01  0.0e+00          0.0e+00
## nbr.na           0.0e+00  0.0e+00  0.0e+00  0.0e+00  0.0e+00          0.0e+00
## min              1.0e+00  1.4e+02  0.0e+00  0.0e+00  1.6e+02          4.4e+00
## max              1.0e+02  3.7e+02  1.9e+02  2.6e+02  2.4e+02          1.9e+01
## range            1.0e+02  2.4e+02  1.9e+02  2.6e+02  8.0e+01          1.5e+01
## sum              5.4e+03  2.4e+04  8.0e+03  1.5e+04  2.0e+04          8.8e+02
## median           5.2e+01  2.5e+02  1.0e+02  1.6e+02  2.0e+02          8.0e+00
## mean             5.2e+01  2.3e+02  7.8e+01  1.5e+02  2.0e+02          8.5e+00
## SE.mean          2.9e+00  7.8e+00  6.0e+00  8.4e+00  2.0e+00          2.8e-01
## CI.mean.0.95     5.8e+00  1.5e+01  1.2e+01  1.7e+01  3.9e+00          5.5e-01
## var              8.9e+02  6.2e+03  3.7e+03  7.3e+03  4.1e+02          7.9e+00
## std.dev          3.0e+01  7.9e+01  6.0e+01  8.5e+01  2.0e+01          2.8e+00
## coef.var         5.7e-01  3.4e-01  7.8e-01  5.7e-01  1.0e-01          3.3e-01
## skewness         0.0e+00  1.4e-01 -1.9e-01 -6.6e-01  2.6e-01          1.1e+00
## skew.2SE         0.0e+00  3.0e-01 -3.9e-01 -1.4e+00  5.4e-01          2.3e+00
## kurtosis        -1.2e+00 -1.7e+00 -1.4e+00 -8.0e-01 -8.5e-01          1.6e+00
## kurt.2SE        -1.3e+00 -1.8e+00 -1.5e+00 -8.5e-01 -9.1e-01          1.7e+00
## normtest.W       9.5e-01  8.4e-01  8.6e-01  8.6e-01  9.7e-01          9.0e-01
## normtest.p       1.4e-03  2.9e-09  2.1e-08  1.6e-08  1.2e-02          1.4e-06
##              Coarse_Aggregates Fine_Aggregates    Slump     Flow
## nbr.val                1.0e+02         1.0e+02  1.0e+02  1.0e+02
## nbr.null               0.0e+00         0.0e+00  1.1e+01  0.0e+00
## nbr.na                 0.0e+00         0.0e+00  0.0e+00  0.0e+00
## min                    7.1e+02         6.4e+02  0.0e+00  2.0e+01
## max                    1.0e+03         9.0e+02  2.9e+01  7.8e+01
## range                  3.4e+02         2.6e+02  2.9e+01  5.8e+01
## sum                    9.1e+04         7.6e+04  1.9e+03  5.1e+03
## median                 8.8e+02         7.4e+02  2.2e+01  5.4e+01
## mean                   8.8e+02         7.4e+02  1.8e+01  5.0e+01
## SE.mean                8.7e+00         6.2e+00  8.6e-01  1.7e+00
## CI.mean.0.95           1.7e+01         1.2e+01  1.7e+00  3.4e+00
## var                    7.8e+03         4.0e+03  7.7e+01  3.1e+02
## std.dev                8.8e+01         6.3e+01  8.8e+00  1.8e+01
## coef.var               1.0e-01         8.6e-02  4.8e-01  3.5e-01
## skewness               1.2e-01         2.6e-01 -1.1e+00 -5.1e-01
## skew.2SE               2.5e-01         5.4e-01 -2.3e+00 -1.1e+00
## kurtosis              -8.8e-01        -6.9e-01 -2.0e-01 -9.5e-01
## kurt.2SE              -9.3e-01        -7.3e-01 -2.1e-01 -1.0e+00
## normtest.W             9.7e-01         9.7e-01  8.1e-01  9.1e-01
## normtest.p             2.9e-02         1.5e-02  4.4e-10  2.0e-06
##              Compressive_Strength_28dy
```

```
## nbr.val                        1.0e+02
## nbr.null                       0.0e+00
## nbr.na                         0.0e+00
## min                            1.7e+01
## max                            5.9e+01
## range                          4.1e+01
## sum                            3.7e+03
## median                         3.6e+01
## mean                           3.6e+01
## SE.mean                        7.7e-01
## CI.mean.0.95                   1.5e+00
## var                            6.1e+01
## std.dev                        7.8e+00
## coef.var                       2.2e-01
## skewness                       1.9e-01
## skew.2SE                       3.9e-01
## kurtosis                       7.5e-02
## kurt.2SE                       7.9e-02
## normtest.W                     9.9e-01
## normtest.p                     4.8e-01
```

### 4.2. Reorganizing Your Data to Handle Multiple Variables at Once

To leverage some of R's more nifty features we will need to reorganize our data from a "spreadsheet style" format to what some people have called a "long form" table so that the column headers of our concrete traits become a single column with the values in the columns placed all into a single column similar to the graphic below.

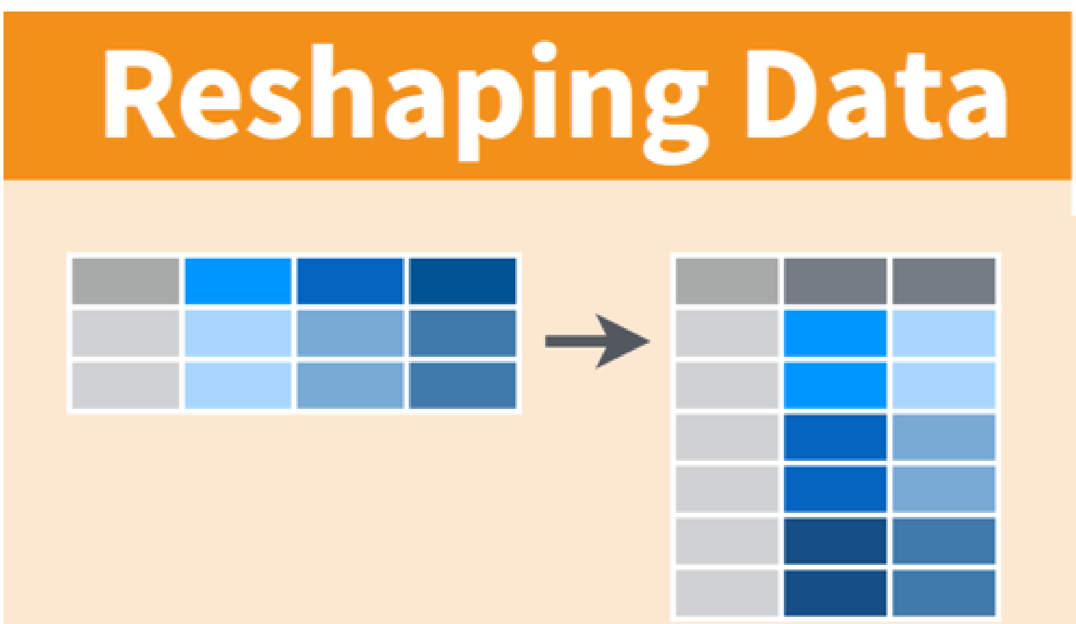



Figure 2: Example of the Gather Function

This is done with the function gather()

```
# Gathering our components into a single column.
```

```r
# We just want the names of our components here so we get everything past
# the first column (which is the experiment name)

column_names  = colnames(concrete[2:ncol(concrete)])

tbl_df(column_names) # tbl_df makes it look pretty when printed
```

```
## Warning: `tbl_df()` was deprecated in dplyr 1.0.0.
## i Please use `tibble::as_tibble()` instead.
```

```
## # A tibble: 10 x 1
##    value
##    <chr>
##  1 Cement
##  2 Slag
##  3 Fly_Ash
##  4 Water
##  5 Superplasticizer
##  6 Coarse_Aggregates
##  7 Fine_Aggregates
##  8 Slump
##  9 Flow
## 10 Compressive_Strength_28dy
```

```r
# the gather command will group everything. in the column name group

concrete_tidy = gather(data  =    concrete, # your data frame
                       key   = "Parameter", # column name for your former columns
                       value =     "Value", # column name for your data
                       column_names       ) # the list for the columns to "gather"
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(column_names)
##
##   # Now:
##   data %>% select(all_of(column_names))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
```

```r
# this will let us sort future plots in the same order as our plots.

concrete_tidy$Parameter = factor(x      = concrete_tidy$Parameter,
                                 levels = column_names)

# we can also split things between our dependant variables and independant variables.


concrete_independent = subset(x      = concrete_tidy,
                              subset = (Parameter != "Slump") &
                                       (Parameter != "Flow")  &
                                       (Parameter != "Compressive_Strength_28dy")
                              )
```

```r
concrete_dependent = subset(x      = concrete_tidy,
                            subset = (Parameter == "Slump") |
                                     (Parameter == "Flow")  |
                                     (Parameter == "Compressive_Strength_28dy")
                            )




print(concrete_tidy)
```

```
## # A tibble: 1,030 x 3
##    Test_Number Parameter Value
##          <dbl> <fct>     <dbl>
##  1          14 Cement      354
##  2          42 Cement      154
##  3           4 Cement      162
##  4          76 Cement      149
##  5          71 Cement      276
##  6          31 Cement      321
##  7          59 Cement      143
##  8          47 Cement      280
##  9          41 Cement      145
## 10          32 Cement      349
## # ... with 1,020 more rows
```

```r
print(concrete_independent)
```

```
## # A tibble: 721 x 3
##    Test_Number Parameter Value
##          <dbl> <fct>     <dbl>
##  1          14 Cement      354
##  2          42 Cement      154
##  3           4 Cement      162
##  4          76 Cement      149
##  5          71 Cement      276
##  6          31 Cement      321
##  7          59 Cement      143
##  8          47 Cement      280
##  9          41 Cement      145
## 10          32 Cement      349
## # ... with 711 more rows
```

```r
print(concrete_dependent)
```

```
## # A tibble: 309 x 3
##    Test_Number Parameter Value
##          <dbl> <fct>     <dbl>
##  1          14 Slump      17
##  2          42 Slump      23
##  3           4 Slump       3
##  4          76 Slump      23.5
##  5          71 Slump       0
##  6          31 Slump      24
##  7          59 Slump      25
```

```
## 8           47 Slump      25.5
## 9           41 Slump       2.5
## 10          32 Slump      20
## # ... with 299 more rows
```

# 5. Plotting Graphics using Tidyverse Resources

R has a few ways to do the basic histograms, Boxplots and other distribution plots.

There are a number of spiffy ways to plot these statistical plots in R. We're just using one here…

## 5.1. SLOOOOWWWWLLLLLYYY Making a Simple Plot (Histogram Edition)

Now I'm going to do this one tiny step at a time until we get to a viable product. (This is how I work through cryptic procedures so I can see what each little additional mystery thingie does.)

Graphing is invoked by the ggplot2 command.. which has a heluvalot under its hood! For me all that detail was what had me a little shy to adopt this way of printing data.

Tidyverse uses what is sometimes called the "grammar of graphics" method… to make a long story longer, the GoG presents separate commands to do separate things rather bundle stuff in a single graphing function. Sometimes it makes a lot of sense… other times it may be confusion. (Hence me demonstrating making a graph this one tiny step at a time!

First thing we are going to do is open a plotting space with the command ggplot()

```
# invoke the ggplot plotting environmnent.

ggplot()
```

Wow. We have a… big square of… grey. All it's doing is setting up our plot environment… so let's do some more…

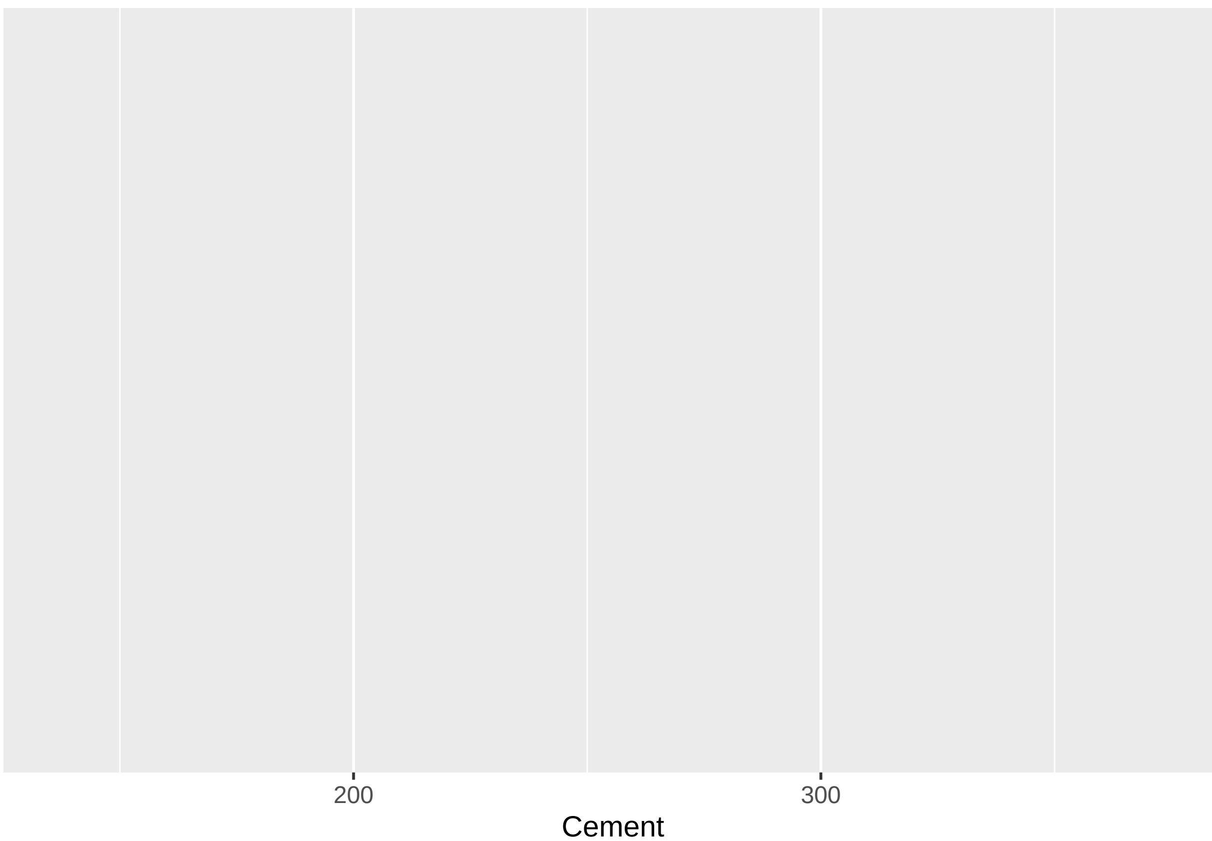
If we want to do a histogram we are going to have to tell it what we want to print and where to get the stuff

When we add things to a plot command in Tidyverse we "add" to the steps incrementally.

This involves a "mapping" function called "aes" (short for aesthetics)

here, we are working with the data frame "concrete" and are working on the variable Cement which we are tossing onto the x axis because that's where the bins of cement go!

```
ggplot(data = concrete) +    # EDIT:  invoke graphics environment using a given dataframe

  aes(x    = Cement)         # NEW: select variable to print... You can get really fancy here later
```



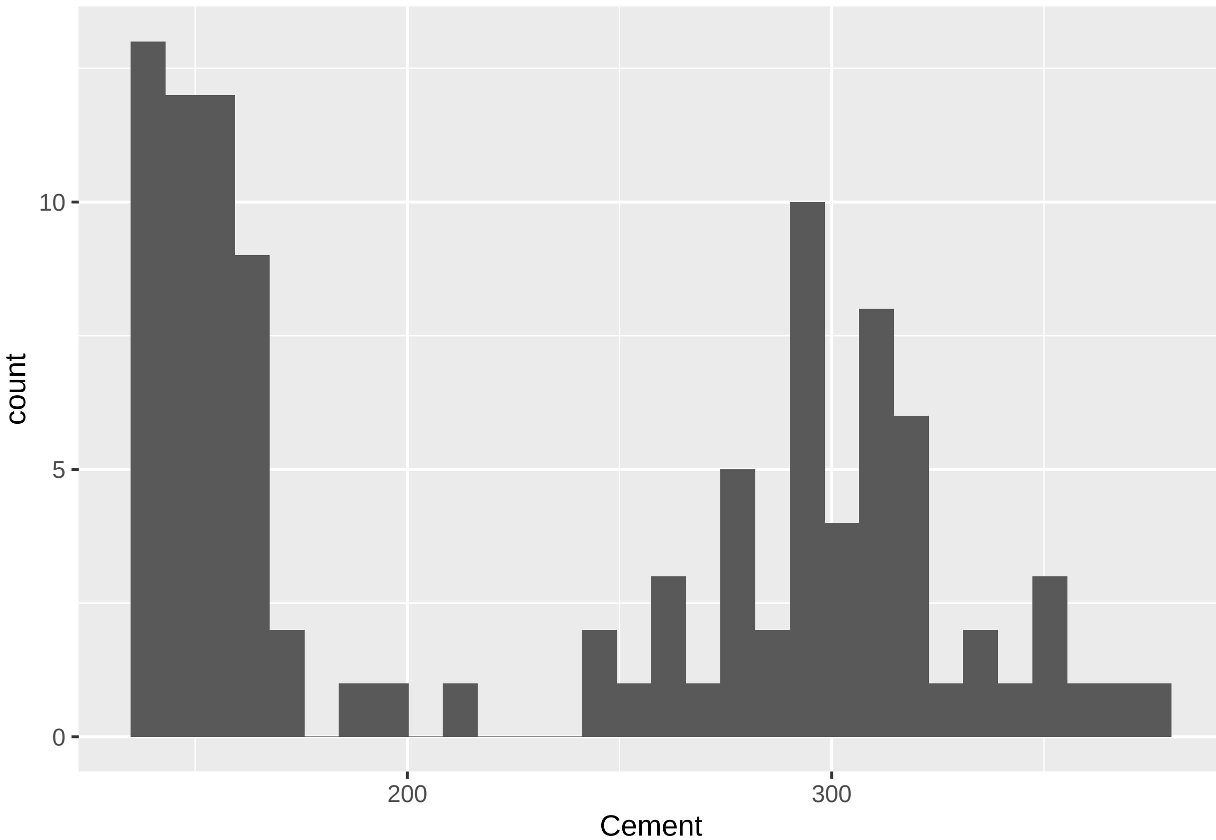

OK now we have something that looks like we may have the making of the graph. If you don't like grey outlines and white grids, no worries, we can change that shortly.

OK.. we are now ready to make a histogram…

Here we will use one of the gglot2's "geom_*" (draw stuff) resources. The default should work for us here.

```
ggplot(data = concrete) +    # invoke graphics environment using a given dataframe

  aes(x = Cement)   +        # select variable to print... You can get really fancy here later

  geom_histogram()           # NEW: insert histogram
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




(you may have gotten a warning about using the bin=X, you can adjust it.)

Now quickly before moving on… I am not keen on the grey background with white lines.

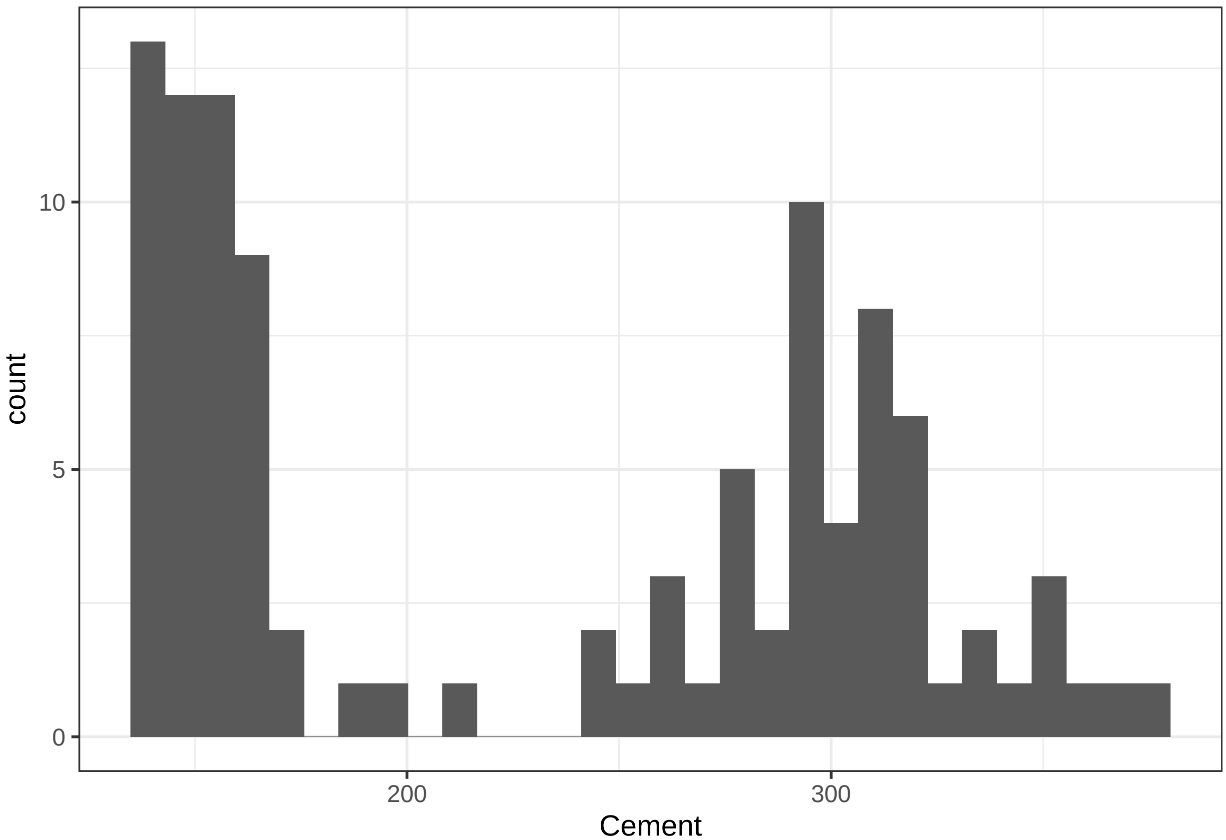
There are a number of out-of-the-box "themes" for ggplot2.

I'm partial to theme_bw() and theme_light() but try the ones that you prefer or stick with the default, theme_gray().

These plots shown here are mine. You should fidget about so they are *yours* and so you can adapt to this new way of working with data.

```r
ggplot(data = concrete) + # invoke graphics environment using a given dataframe

  theme_bw() +            # NEW: changing the plotting theme

  aes(x = Cement) +       # select variable to print... You can get really fancy here later

  geom_histogram()        # insert histogram (including controlling number of bins)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

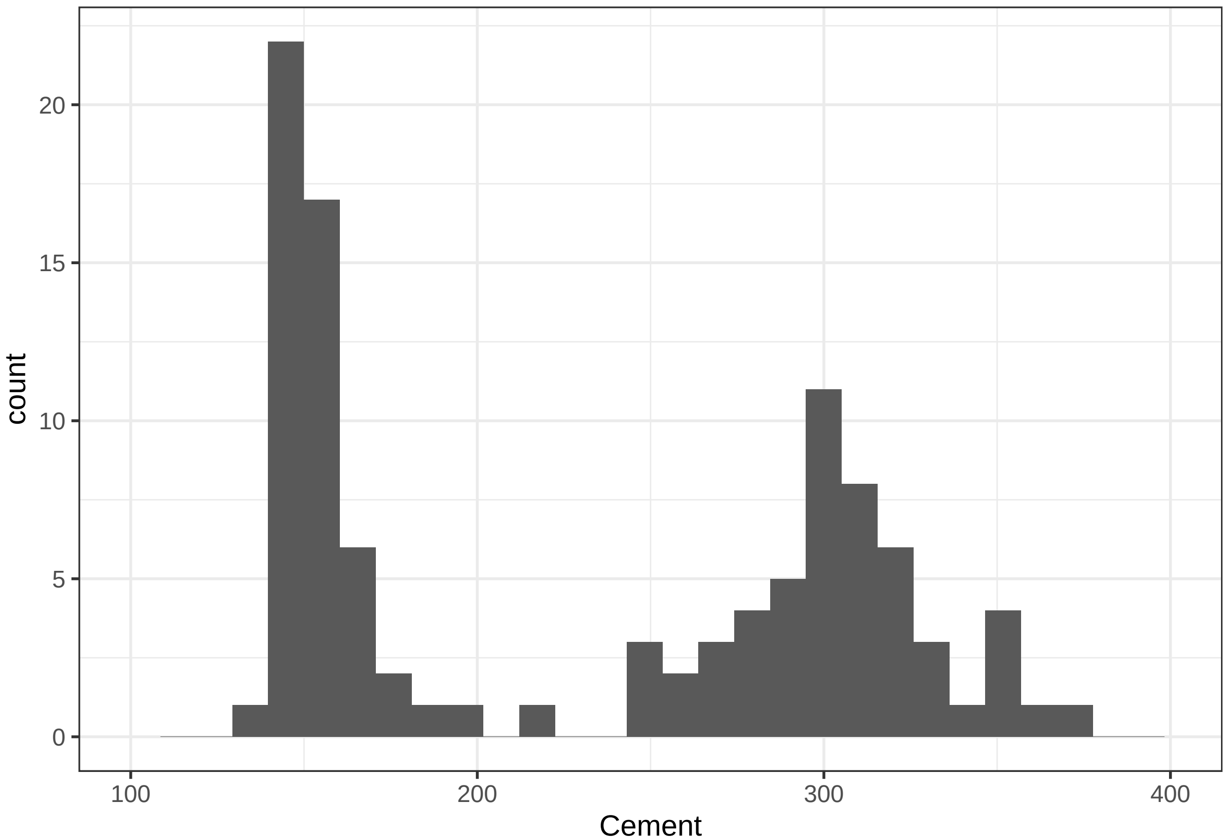



My OCD hates axes where the labels don't envelop all of the data…

We can fix that with xlim() or ylim()

```
ggplot(data = concrete) +      # invoke graphics environment using a given dataframe
  theme_bw() +                 # changing the plotting theme
  aes(x = Cement) +            # select variable to print... You can get really fancy here later
  xlim( 100, 400 ) +           # NEW: adding x-axis limits
  geom_histogram()             # insert histogram
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 2 rows containing missing values (`geom_bar()`).
```

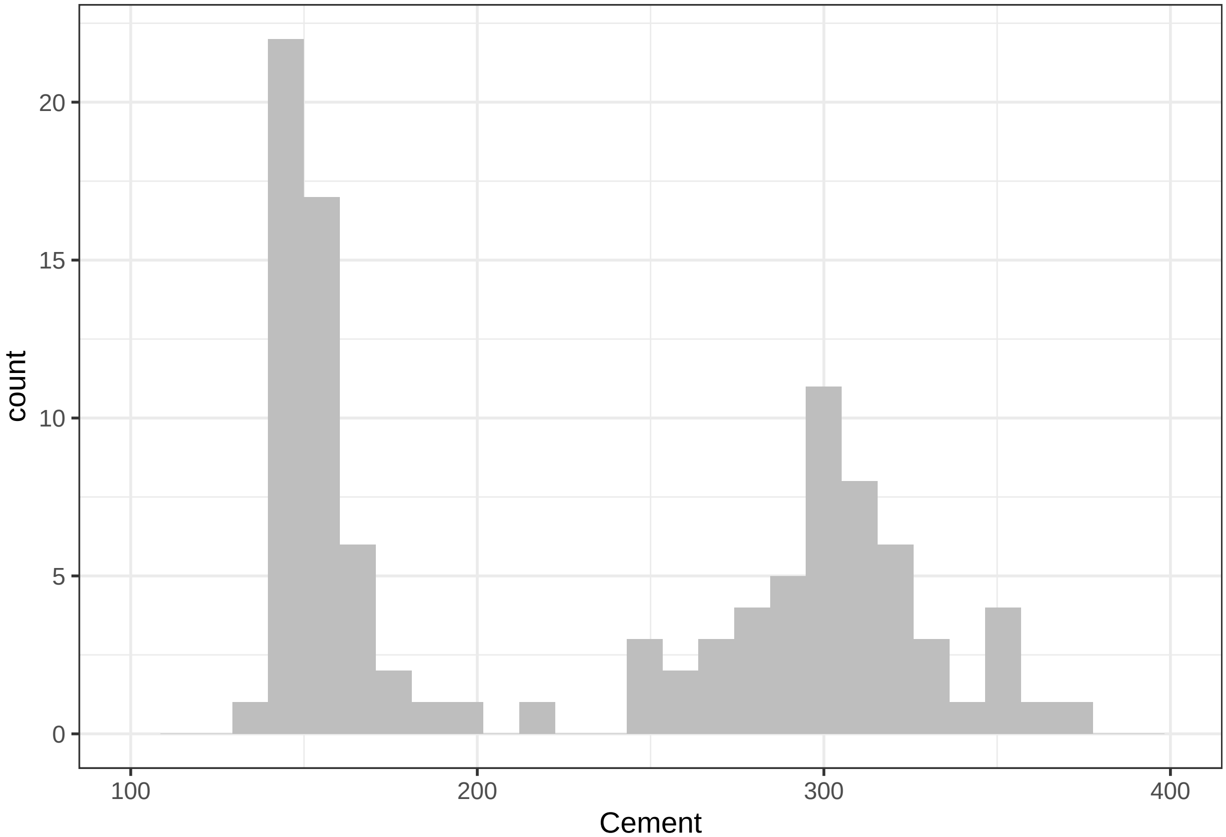



How about changing the color of the fill in the bars…

You really don't want to know about all the colors you can use.

```
ggplot(data = concrete) +      # invoke graphics environment using a given dataframe
  theme_bw() +                 # changing the plotting theme
  aes(x = Cement) +            # select variable to print... You can get really fancy here later
  xlim( 100, 400 ) +           # NEW: adding x-axis limits
  geom_histogram(fill="gray") # EDIT: insert histogram (with a single chosen color)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 2 rows containing missing values (`geom_bar()`).
```



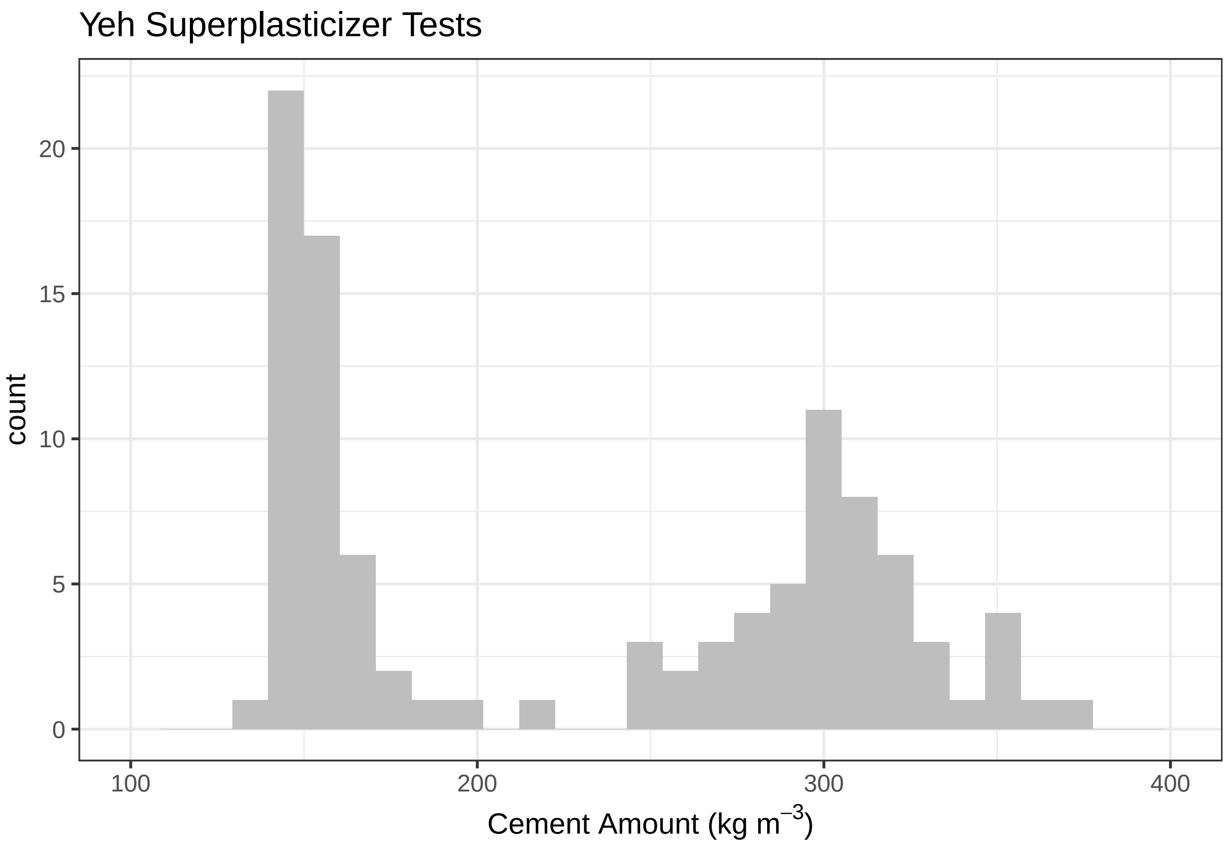

Want to customize the labels and titles so we can have units?

You can add custom labels and titles! (https://www.nceas.ucsb.edu/~frazier/RSpatialGuides/colorPaletteCheatsheet.pdf)

For the superscripting in the x-axis label, I am using the expression() tool in R.

```
ggplot(data = concrete) +      # invoke graphics environment using a given dataframe

  theme_bw() +                 # changing the plotting theme

  aes(x = Cement) +            # select variable to print... You can get really fancy here later

  xlim( 100, 400 ) +           # adding x-axis limits

  ggtitle("Yeh Superplasticizer Tests") +          # NEW : Custom Title

  xlab(expression('Cement Amount (kg m'^-3*")")) + # NEW : Custom Axis Label

  geom_histogram(fill="gray") # insert histogram (with a single chosen color)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 2 rows containing missing values (`geom_bar()`).
```

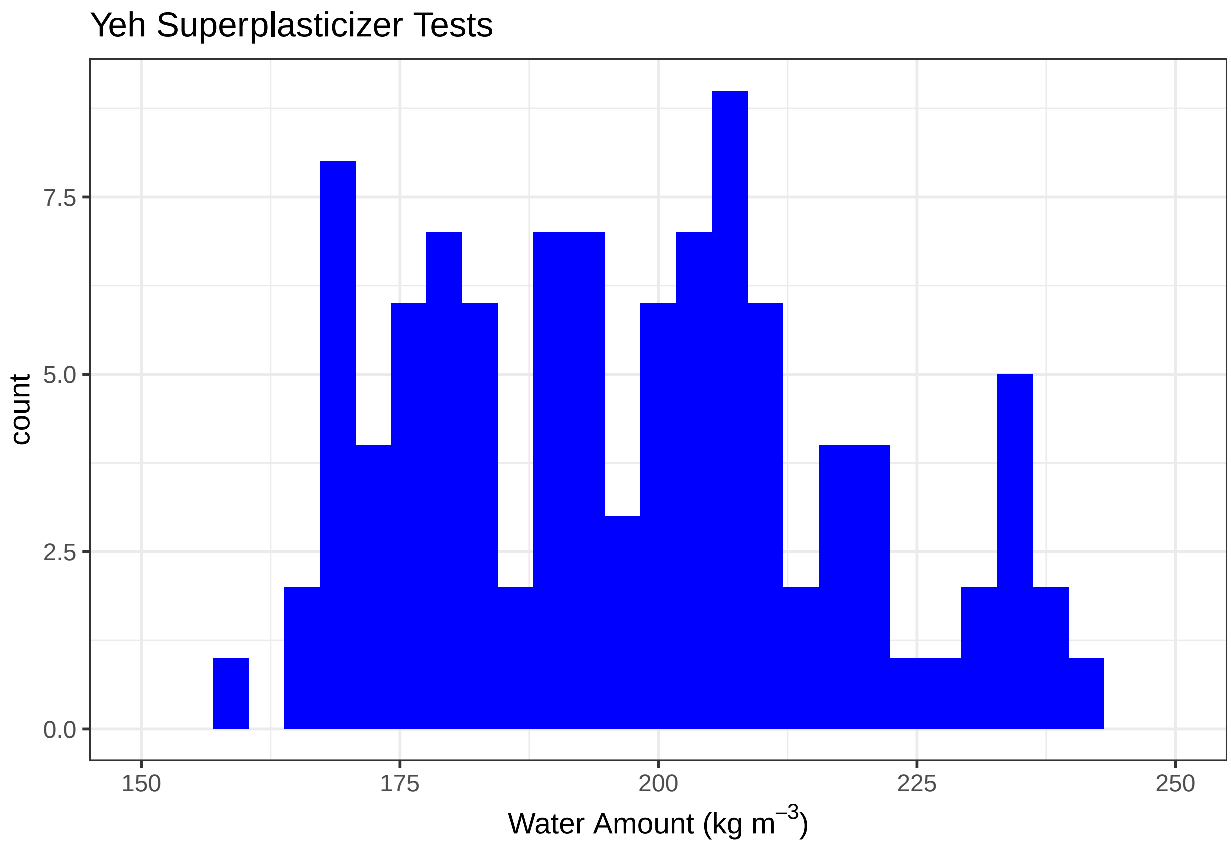



And I could keep tweaking this graph all day, but good enough is good enough so this is a good place to stop...

We also can plot a few other fields with some trial and error..

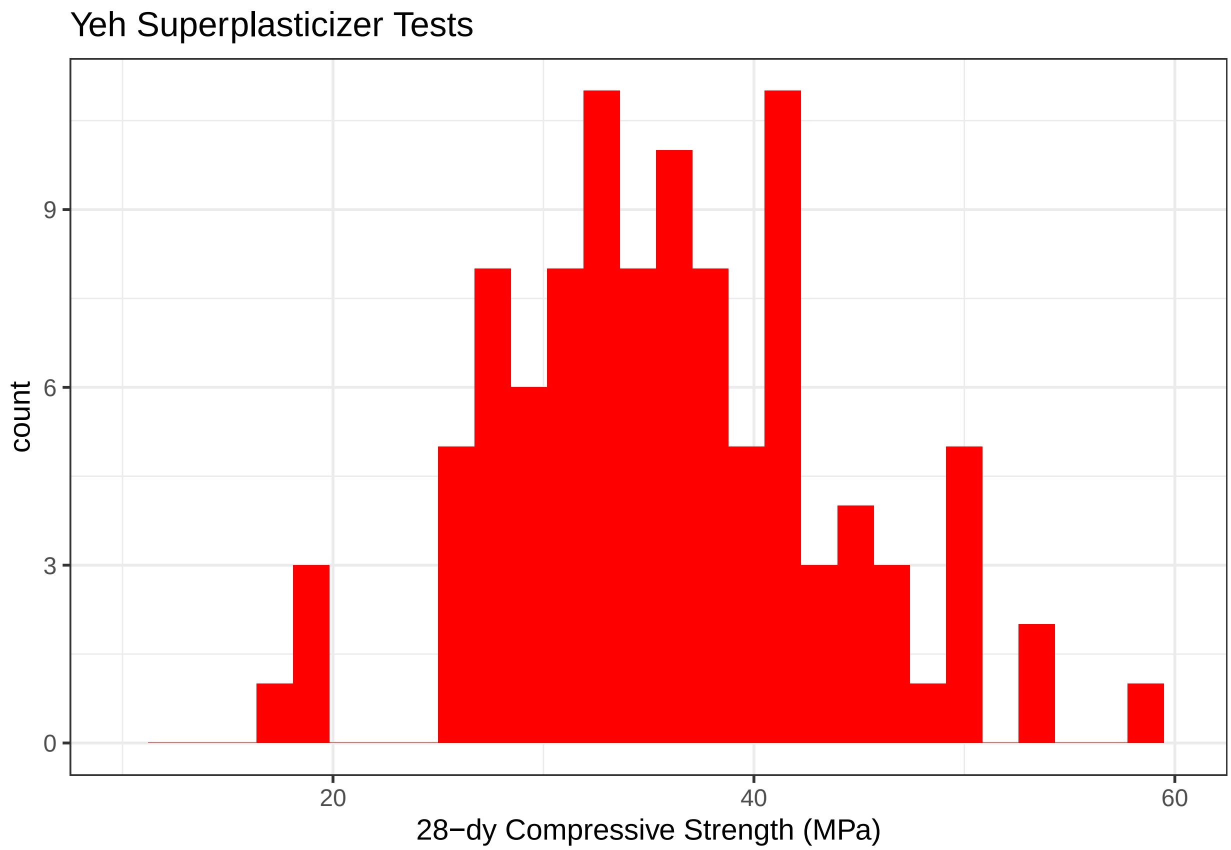
```
# Histogram of Water

ggplot(data = concrete) +      # invoke graphics environment using a given dataframe

  theme_bw() +                 # changing the plotting theme

  aes(x = Water) +            # select variable to print... You can get really fancy here later

  xlim( 150, 250 ) +           # adding x-axis limits

  ggtitle("Yeh Superplasticizer Tests") + #Custom Title

  xlab(expression('Water Amount (kg m'^-3*")")) + # NEW : Custom Axis Label note use of superscripts fr

  geom_histogram(fill="blue") # insert histogram (with a single chosen color)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 1 rows containing missing values (`geom_bar()`).
```




```
# Histogram of Strength

ggplot(data = concrete) +      # invoke graphics environment using a given dataframe

  theme_bw() +                 # changing the plotting theme

  aes(x = Compressive_Strength_28dy) + # select variable to print... You can get really fancy here late

  xlim( 10, 60 ) +             # adding x-axis limits

  ggtitle("Yeh Superplasticizer Tests") + #Custom Title

  xlab("28-dy Compressive Strength (MPa)") + # NEW : Custom Axis Label

  geom_histogram(fill="red") # insert histogram (with a single chosen color)
```

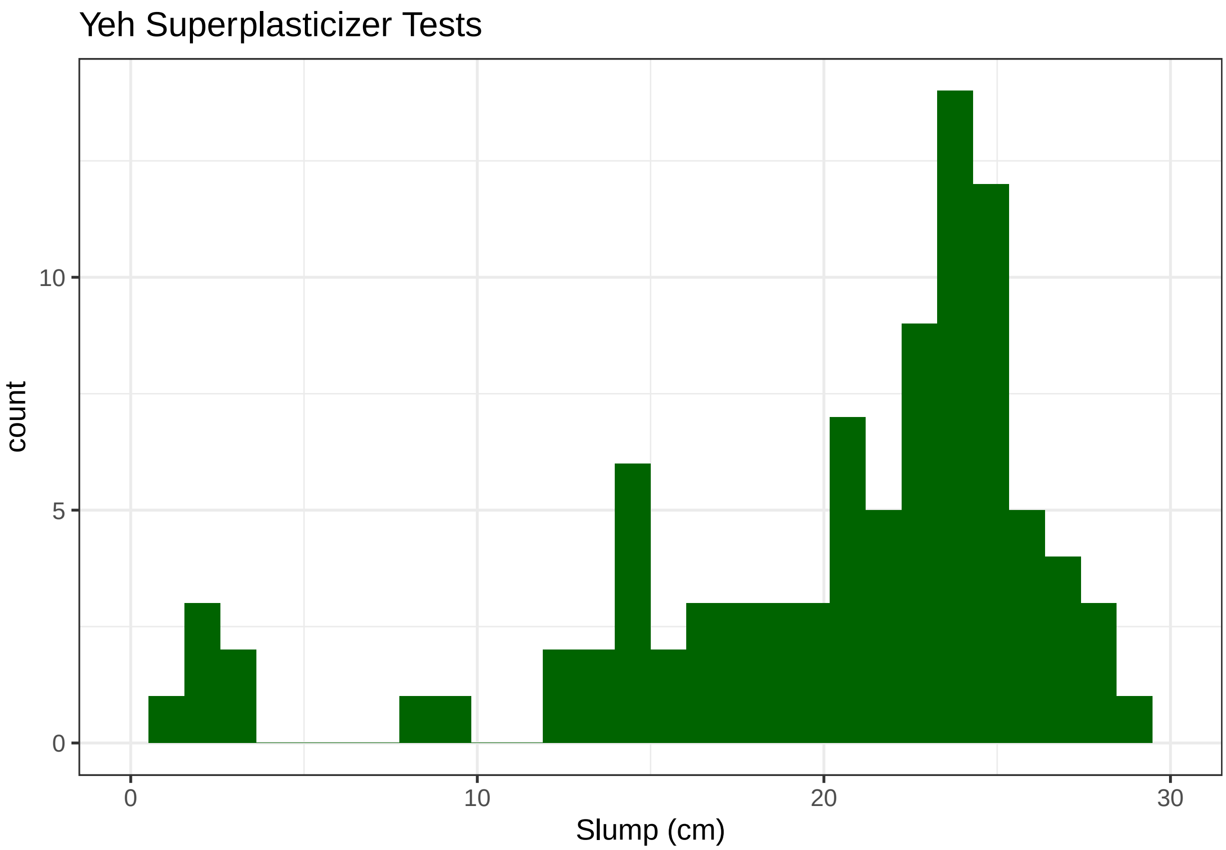
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 2 rows containing missing values (`geom_bar()`).
```




(And from our Intro to Stats Lecture...)

```
# Histogram of Strength

ggplot(data = concrete) +      # invoke graphics environment using a given dataframe

  theme_bw() +                 # changing the plotting theme

  aes(x = Slump) + # select variable to print... You can get really fancy here later

  xlim( 0, 30 ) +           # adding x-axis limits

  ggtitle("Yeh Superplasticizer Tests") + #Custom Title

  xlab("Slump (cm)") + # NEW : Custom Axis Label

  geom_histogram(fill="darkgreen") # insert histogram (with a single chosen color)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 2 rows containing missing values (`geom_bar()`).
```



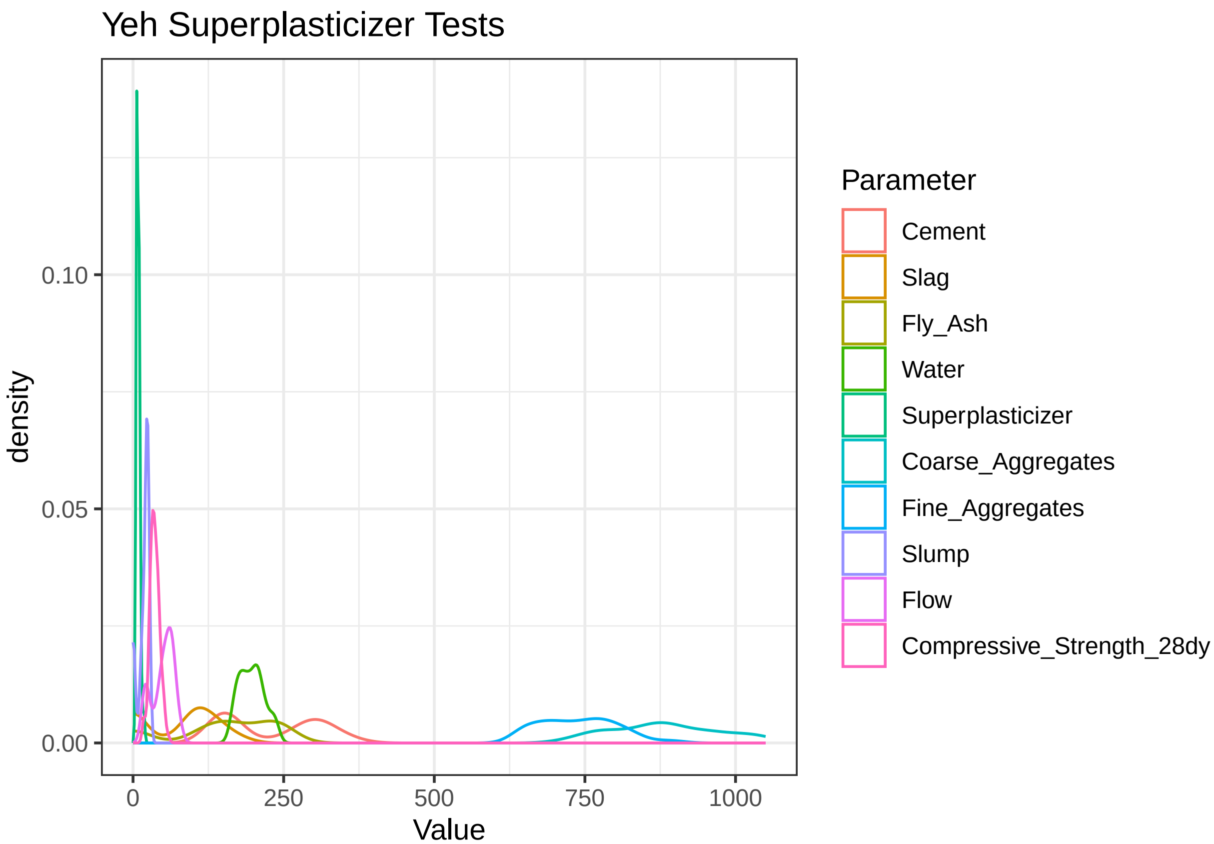

### 5.2 Distribution Plot [not so good an] Example

There are some other plots that we can use to describe our data.

Here to play with them we will take a quick step back and address that "tidy"'ed (should that say "tidied"?) dataframe "concrete_tidy"

We can now use all the parameters in the "tidy" (long) data frame to print by specific traits.

```
ggplot(data = concrete_tidy) +             # invoke graphics environment using a given dataframe

  theme_bw() +                             # changing the plotting theme

  aes(x      = Value,                      # map x-axis value
      color  = Parameter) +                # map colors for different quality

  ggtitle("Yeh Superplasticizer Tests") + # Custom Title

  xlab("Value") +                          #  Custom Axis Label

  geom_density()                           # insert crete a relative density plot
```

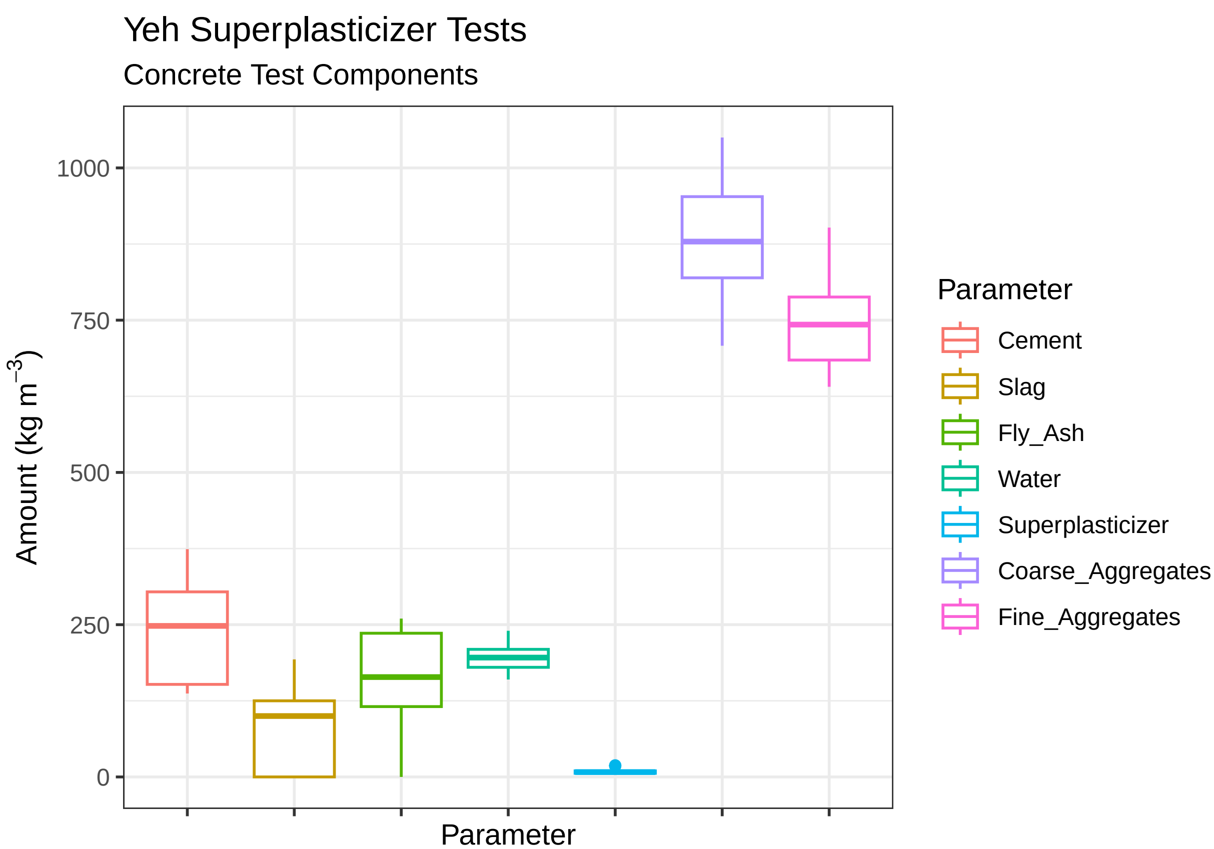



In the past, I've gotten good results with this but in this case, I think it's too messy in part due to the disparity in the dynamic range of our parameters.

### 5.3. Box-Whisker Plot Example

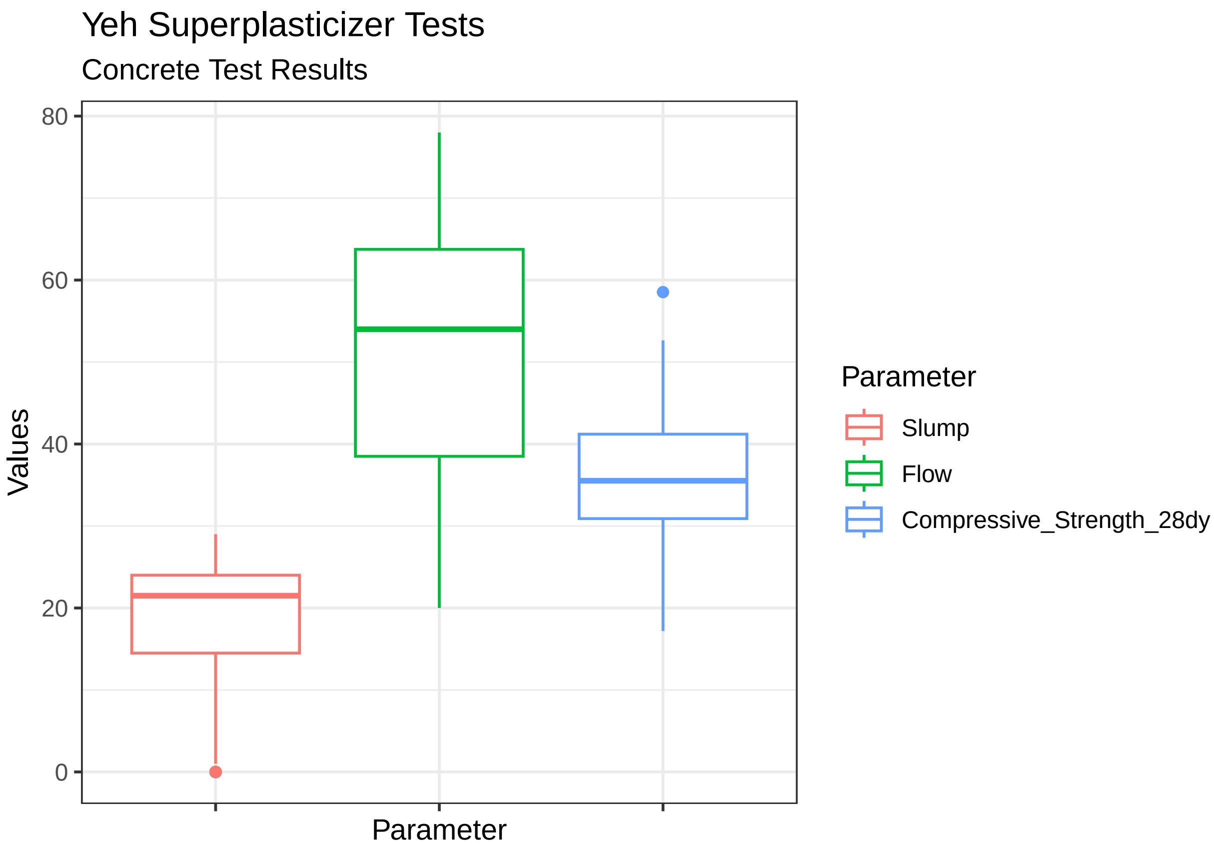
How about leveraging a box whisker? (I'm using only the independent variables this time.)

```
ggplot(data = concrete_independent) +      # EDIT Changing dataframe

  theme_bw( ) +                            # changing the plotting theme

  theme(axis.text.x = element_blank()) +   # adding an extra trait to the x-axis
                                           # to not print labels on the x-axis
                                           # (the labels overlap and doesn't look
                                           # pretty...)

  aes(y      = Value,                     # map y-axis value
      x      = Parameter,                 # map x-axis value
      color  = Parameter) +               # map colors for different quality

  ggtitle(label    = "Yeh Superplasticizer Tests",
          subtitle = "Concrete Test Components") + # Custom Title

  ylab(expression('Amount (kg m'^-3*")")) + # EDIT : Changing Custom Axis Label

  geom_boxplot()                          # insert crete a relative density plot
```




What about our dependent variables? We can start by changing the data frame...

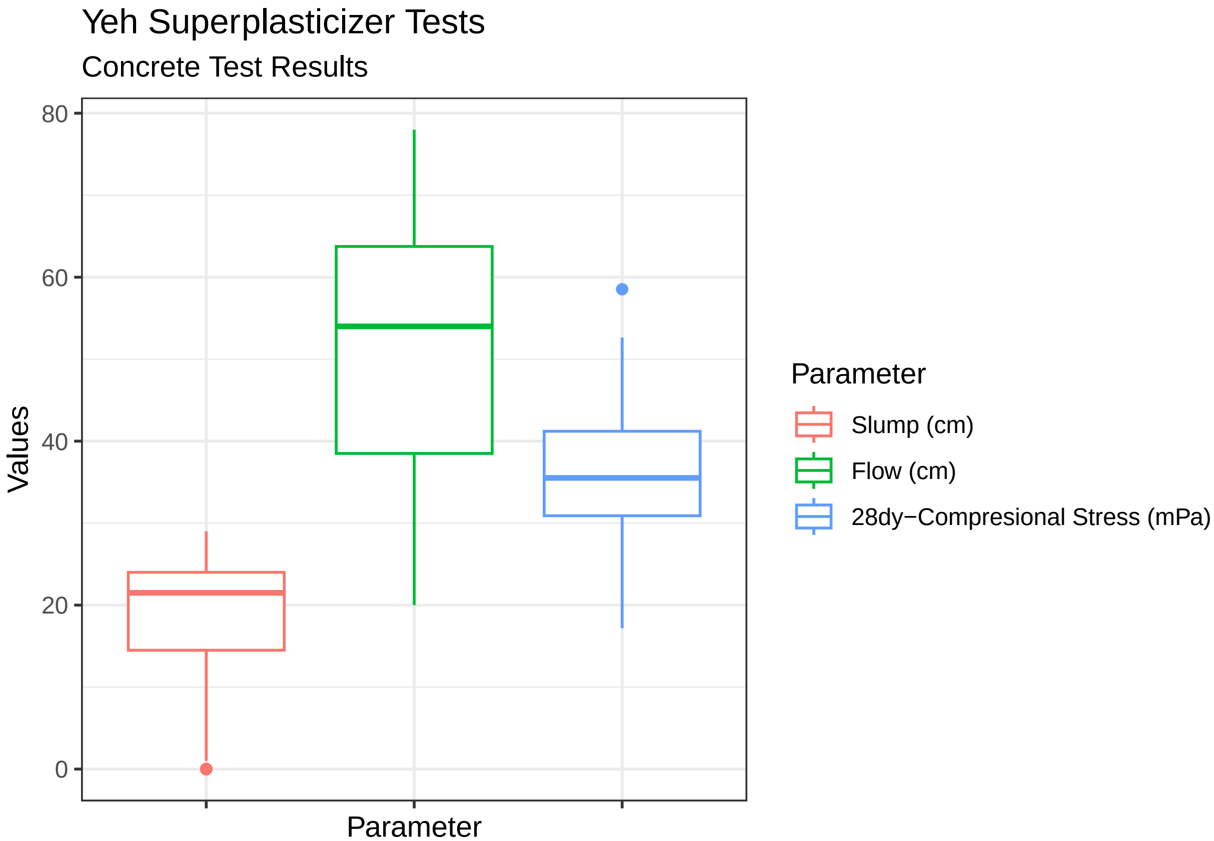
```
ggplot(data = concrete_dependent) +      # EDIT Changing dataframe

  theme_bw( ) +                           # changing the plotting theme

  theme(axis.text.x = element_blank()) +  # adding an extra trait to the x-axis
                                          # to not print labels on the x-axis
                                          # (the labels overlap and doesn't look
                                          # pretty...)

  aes(y       = Value,                   # map y-axis value
      x       = Parameter,               # map x-axis value
      color   = Parameter) +             # map colors for different quality

  ggtitle(label    = "Yeh Superplasticizer Tests",
          subtitle = "Concrete Test Results") + # Custom Title

  ylab("Values") +

  geom_boxplot()                         # insert crete a relative density plot
```




Want units? That's a little tougher here since the units differ by parameter. We can force the values to into new names though.

```
ggplot(data = concrete_dependent) +       # EDIT Changing dataframe

  theme_bw( ) +                            # changing the plotting theme

  theme(axis.text.x = element_blank()) +   # adding an extra trait to the x-axis
                                           # to not print labels on the x-axis
                                           # (the labels overlap and doesn't look
                                           # pretty...)

  aes(y      = Value,                     # map y-axis value
      x      = Parameter,                 # map x-axis value
      color  = Parameter) +               # map colors for different quality

  ggtitle(label    = "Yeh Superplasticizer Tests",
          subtitle = "Concrete Test Results") + # Custom Title

  ylab("Values") +

  # NEW: It says scale color but "color" is how we are distinguishing
  #      out boxplots (as seen in the mapping/aes command)
  #      we can then use the same plot order above to rewrite the labels
  #      (likewise we could change the plot order and of coruse the colors.)
  scale_color_discrete(labels = c("Slump (cm)",
                                  "Flow (cm)",
                                  "28dy-Compresional Stress (mPa)")) +
```

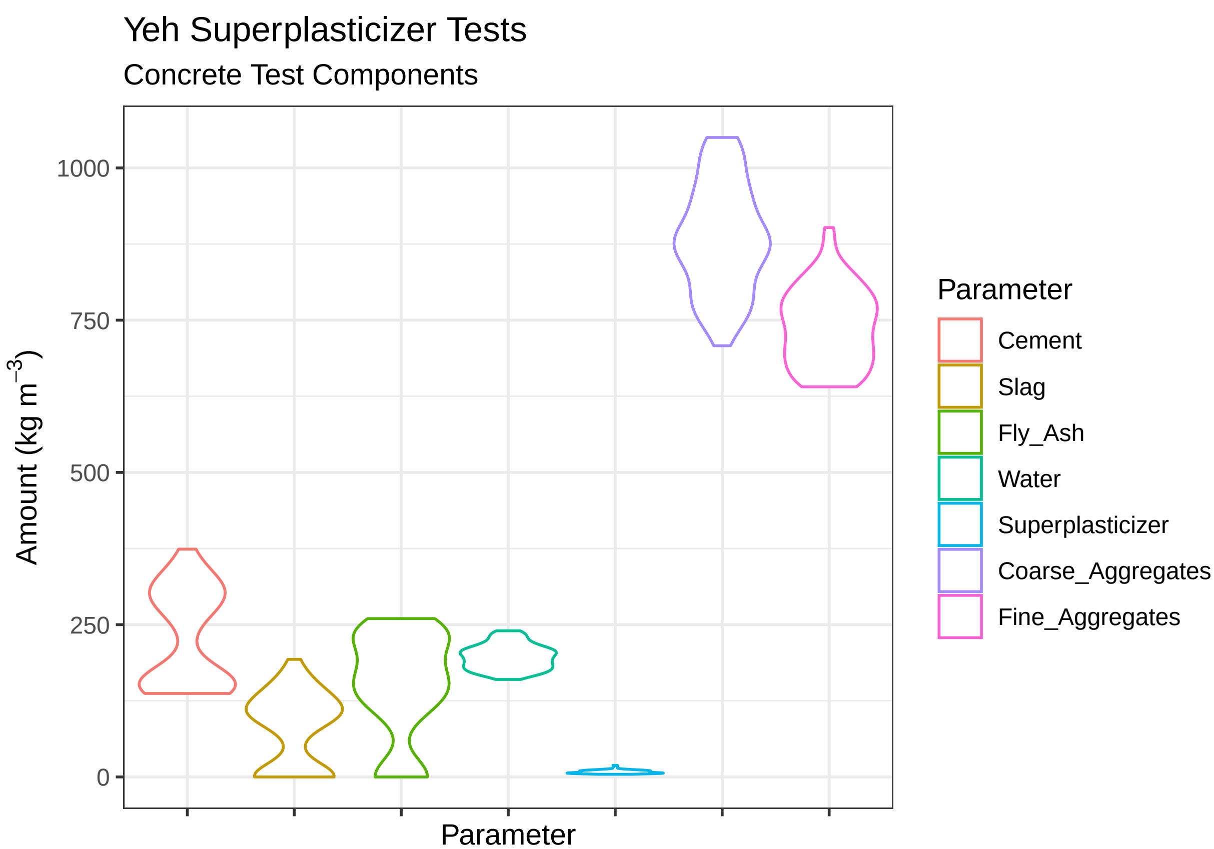
```
geom_boxplot() # insert crete a relative density plot
```




### 5.4. Violin Plot Example

How about leveraging a "violin" plot? A violin plot's width swells in areas with more observations and contracts with sparser data so it is like looking at a probability distribution.

```
ggplot(data = concrete_independent) +       # EDIT Changing dataframe

  theme_bw( ) +                             # changing the plotting theme

  theme(axis.text.x = element_blank()) +    # adding an extra trait to the x-axis
                                            # to not print labels on the x-axis
                                            # (the labels overlap and doesn't look
                                            # pretty...)

  aes(y      = Value,                      # map y-axis value
      x      = Parameter,                  # map x-axis value
      color  = Parameter) +                # map colors for different quality

  ggtitle(label    = "Yeh Superplasticizer Tests",
          subtitle = "Concrete Test Components") + # Custom Title

  ylab(expression('Amount (kg m'^-3*")")) + #  Changing Custom Axis Label

  geom_violin(scale="width") # EDIT: change to a violin plot
```

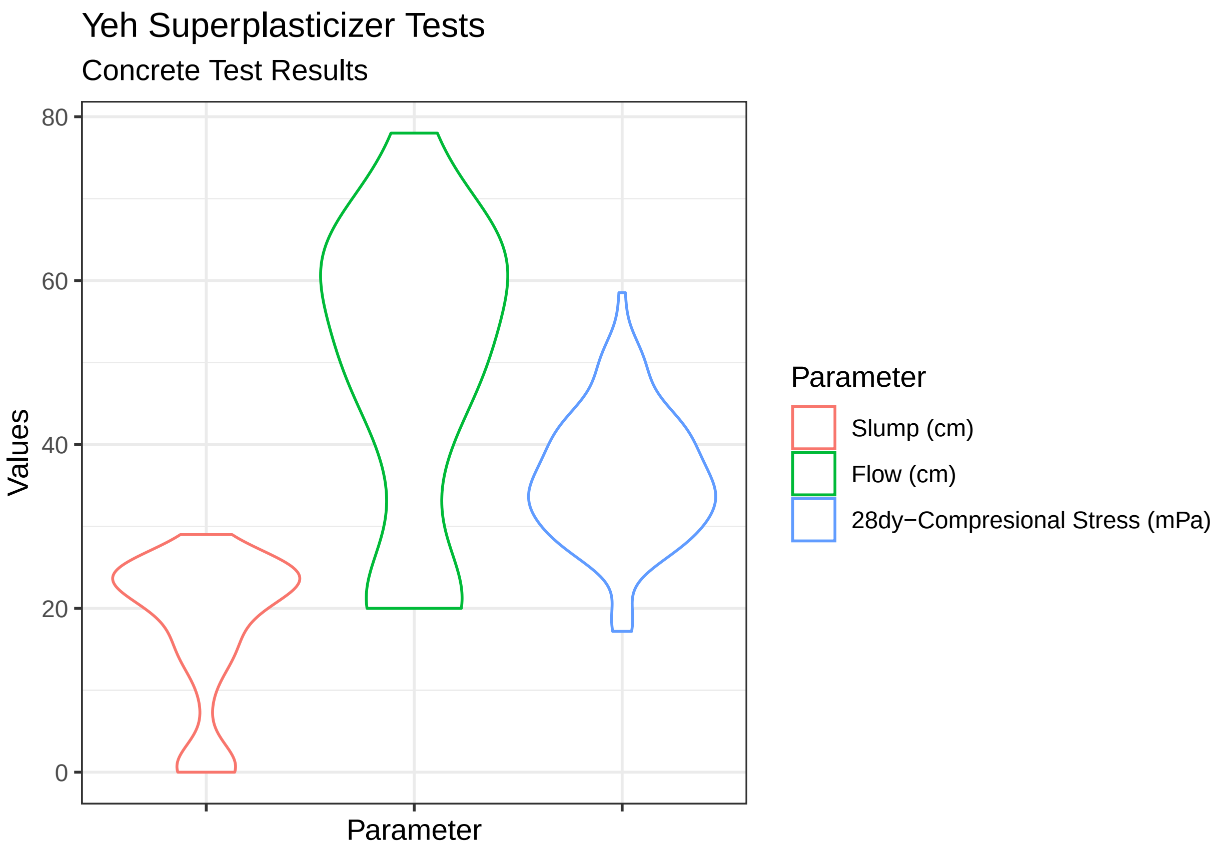



```
                                          #   the width argument
                                          # gives every plot the same width
```

and...

```
ggplot(data = concrete_dependent) +       # EDIT Changing dataframe

  theme_bw( ) +                             # changing the plotting theme

  theme(axis.text.x = element_blank()) +    # adding an extra trait to the x-axis
                                            # to not print labels on the x-axis
                                            # (the labels overlap and doesn't look
                                            # pretty...)

  aes(y       = Value,                     # map y-axis value
      x       = Parameter,                 # map x-axis value
      color   = Parameter) +               # map colors for different quality

  ggtitle(label     = "Yeh Superplasticizer Tests",
          subtitle = "Concrete Test Results") + # Custom Title

  ylab("Values") +

  # NEW: It says scale color but "color" is how we are distinguishing
  #       out boxplots (as seen in the mapping/aes command)
  #       we can then use the same plot order above to rewrite the labels
  #       (likewise we could change the plot order and of coruse the colors.)
  scale_color_discrete(labels = c("Slump (cm)",
```

```
                                     "Flow (cm)",
                                     "28dy-Compresional Stress (mPa)")) +


  geom_violin(scale="width") # EDIT: change to a violin plot
```




```
                                # the width argument
                                # gives every plot the same width
```

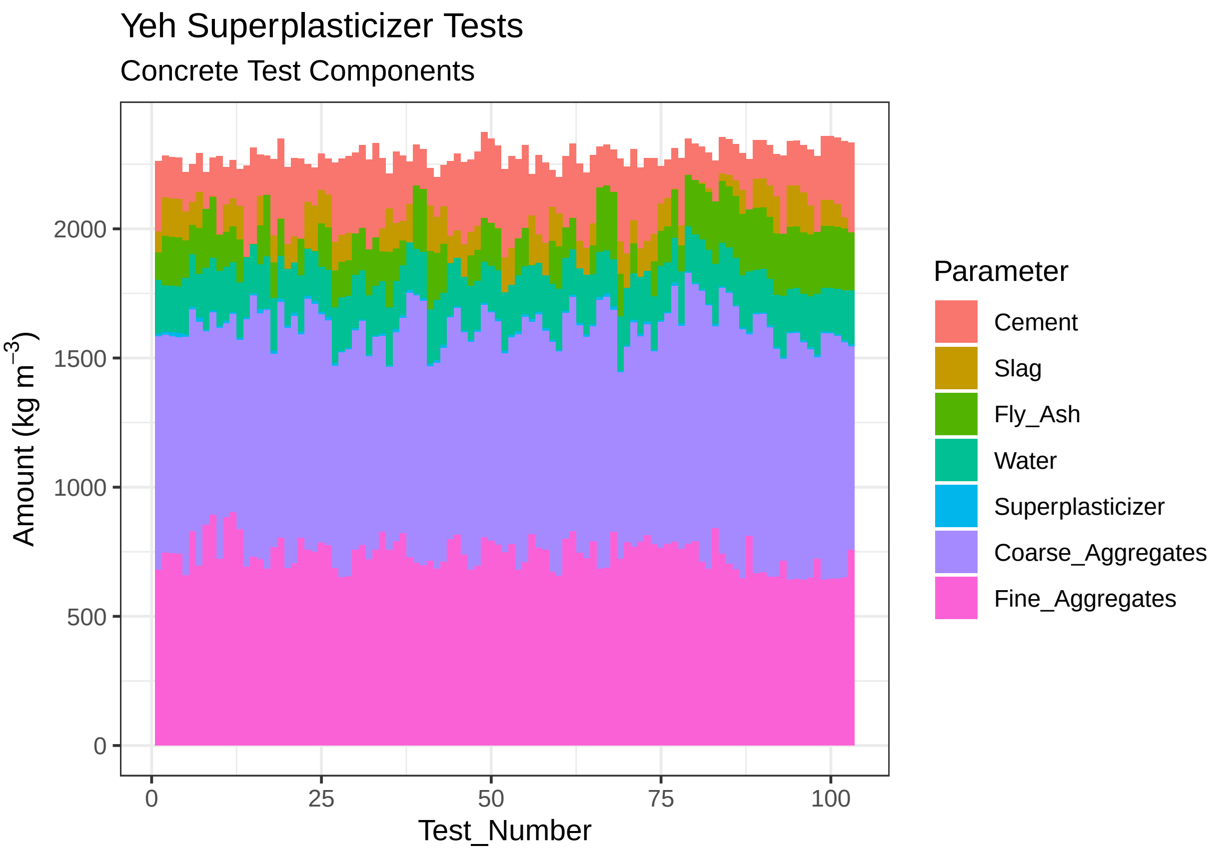
This is basically the above "density" plot but "looking down" as with a box plot. Also here we are trimming the plot so that when we leave the range of any of the data points, the "violins" are truncated.

### 5.5. Stacked Column or Bar Plot Example

We also can do bar plots or stacked column plots. The one produced here shows the combined components by test unit.

```
ggplot(data = concrete_independent) +      # EDIT Changing dataframe

  theme_bw( ) +                            # changing the plotting theme


  aes(x      = Test_Number,
      y      = Value,
      fill   = Parameter) +              # map colors for different quality

  ggtitle(label    = "Yeh Superplasticizer Tests",
          subtitle = "Concrete Test Components") + # Custom Title
```

```r
  ylab(expression('Amount (kg m'^-3*")")) + #  Changing Custom Axis Label

  geom_col(position = "stack",  # new, create a stacekd column graph
           width    = 1.0    )  # with no space between columns
```




# 6. Correlation of Variables

## 6.1. Correlating and then Fitting Cement to Compressive Strength

Let's start by doing a "simple"" plot . In this case since I already know the answer because the spreadsheet also has a table of how well our independent variables correlate against the dependent variables (e.g., Slump, Flow, or in our case Strength). The Cement correlates the best against Compressive Strength (OK, truth be told, it correlates the least badly).

We can actually do this with a correlate function, cor()…

To grab a value in the table "concrete" we call the data frame (concrete) and the variable name (Cement or Water vs Compressive_Strength_28dy), separating the frame and variable names by a $ sign.

```r
print("Cement vs Compressive Strength Correlation, r")
```

```
## [1] "Cement vs Compressive Strength Correlation, r"
```

```r
cor(x = concrete$Cement,                    # the x-value
    y = concrete$Compressive_Strength_28dy, # the y-value
    method = "pearson"                      # method of correlation
    )
```

```
## [1] 0.45
```

or if you like to do everything at once…

```
# calculate all correlation values against each other

correlation_matrix = cor(x      = concrete, # using our dataframe to correlate evything
                         method = "pearson" )

tbl_df(correlation_matrix)
```

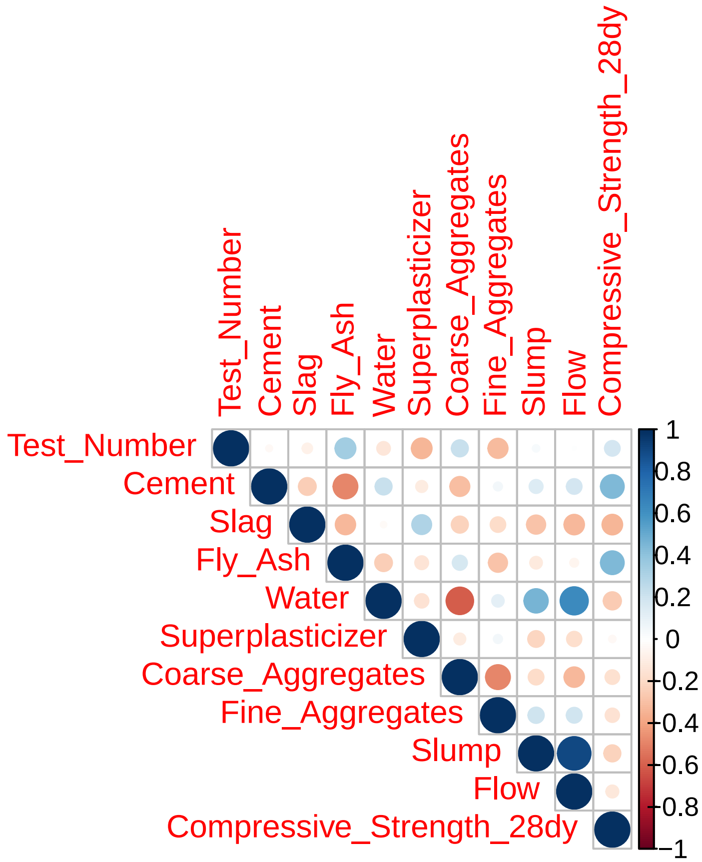
```
## Warning: `tbl_df()` was deprecated in dplyr 1.0.0.
## i Please use `tibble::as_tibble()` instead.
```

```
## # A tibble: 11 x 11
##    Test_Number  Cement    Slag Fly_Ash   Water Superpl~1 Coars~2 Fine_~3   Slump
##          <dbl>   <dbl>   <dbl>   <dbl>   <dbl>     <dbl>   <dbl>   <dbl>   <dbl>
##  1     1        -0.0316 -0.0798  0.341  -0.138    -0.335    0.222 -0.314   0.0374
##  2    -0.0316    1      -0.244  -0.487   0.221    -0.106   -0.310  0.0570  0.146
##  3    -0.0798   -0.244   1      -0.323  -0.0268    0.307   -0.224 -0.184  -0.284
##  4     0.341    -0.487  -0.323   1      -0.241    -0.144    0.173 -0.283  -0.119
##  5    -0.138     0.221  -0.0268 -0.241   1        -0.155   -0.602  0.115   0.467
##  6    -0.335    -0.106   0.307  -0.144  -0.155     1       -0.104  0.0583 -0.213
##  7     0.222    -0.310  -0.224   0.173  -0.602    -0.104    1     -0.489  -0.188
##  8    -0.314     0.0570 -0.184  -0.283   0.115     0.0583  -0.489  1       0.202
##  9     0.0374    0.146  -0.284  -0.119   0.467    -0.213   -0.188  0.202   1
## 10     0.00866   0.186  -0.327  -0.0554  0.632    -0.176   -0.326  0.190   0.906
## 11     0.186     0.446  -0.332   0.444  -0.254    -0.0379  -0.161 -0.154  -0.223
## # ... with 2 more variables: Flow <dbl>, Compressive_Strength_28dy <dbl>, and
## #   abbreviated variable names 1: Superplasticizer, 2: Coarse_Aggregates,
## #   3: Fine_Aggregates
```

Lots of numbers… not all that insightful on their own…

You also can graph the look-n-feel of what all of the different correlations are… (it works best with a much smaller number of variables)

```
  # draw a coorelation graphic...

  corrplot(corr    = correlation_matrix,
           type    = "upper")
```

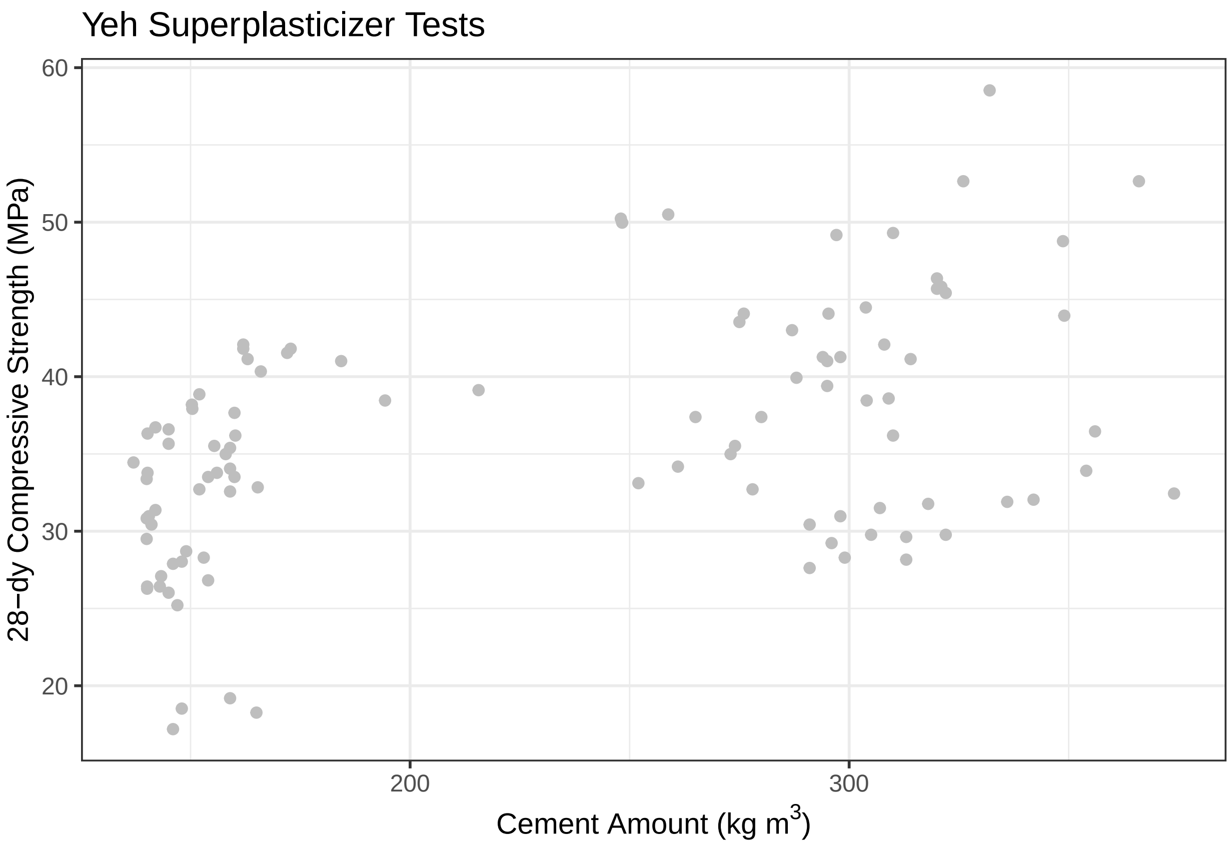



We can now see for example that cement, slag, and fly ash amounts have a nominal but not thrilling correlation to compression strength while water has a good correlation with the resulting slump values. One thing that this does *not* show is how well these parameters play with other parameters. As we'll see when all of our independent values are working together we'll discover that cement and water, followed by fly ash and coarse aggregates will, together, contribute the most of our independent parameters in calculating the compressive strength.

## 6.2. Scatter Plot Example

But for now, let's plot plot the Cement amount against Compressive Strength

```
# Making a simple X-Y scatterplot.

ggplot(data = concrete) +                # invoke graphics environment using a given dataframe

  theme_bw( ) +                           # changing the plotting theme

  aes(x      = Cement,                      # x-value
      y      = Compressive_Strength_28dy) +  # y-value

  ggtitle("Yeh Superplasticizer Tests") +     # Custom Title

  xlab(expression('Cement Amount (kg m'^3*")")) +    # x-label
  ylab("28-dy Compressive Strength (MPa)")      +    # y-label

  geom_point(colour="grey")   # EDIT: plot points the color keyword part was
```

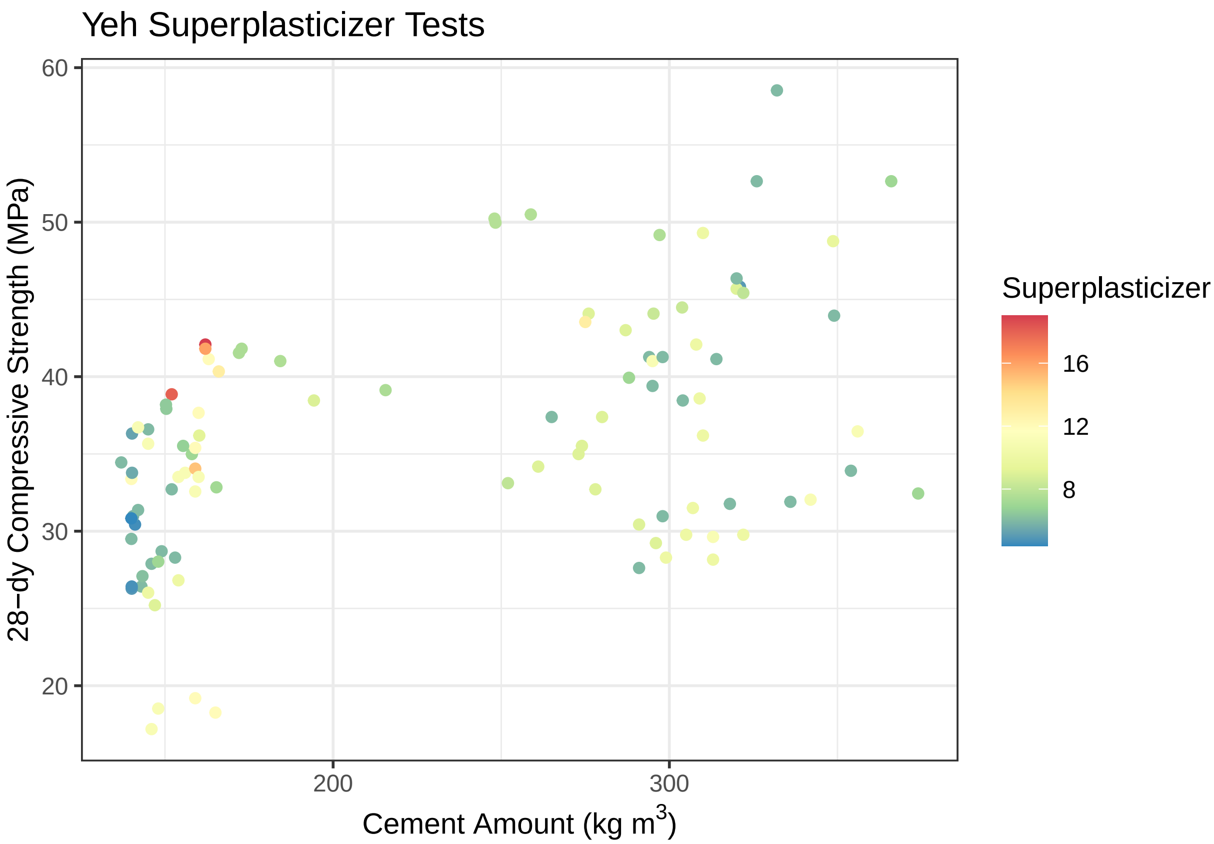



```
#       writen by an anglophile!
```

Here's a cute trick: Could we color those dots by a variable?

Sure!

```
# Making a simple X-Y scatterplot now coloured by another parameter

ggplot(data = concrete) +                # invoke graphics environment using a given dataframe

  theme_bw( ) +                           # changing the plotting theme

  aes(x       = Cement,                      # x-value
      y       = Compressive_Strength_28dy,   # y-value
      color   = Superplasticizer)          +  # ADD: we can color by a variable!

  ggtitle("Yeh Superplasticizer Tests") +    # Custom Title

  xlab(expression('Cement Amount (kg m'^3*")")) +    # x-label
  ylab("28-dy Compressive Strength (MPa)")      +    # y-label

  geom_point() +  # plot points
  scale_color_distiller(palette = "Spectral") # NEW: pick a custom "colour" palate.
```

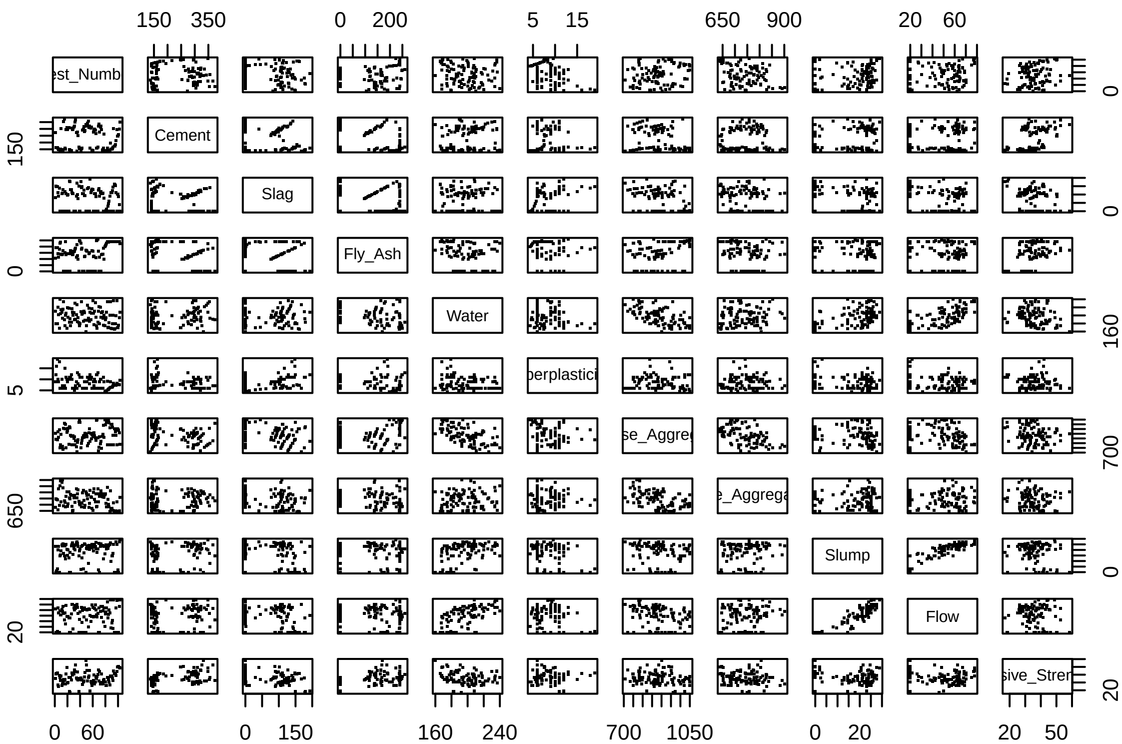



Love overkill without any distinct numerical score and look at how everything in your data set correlates with every other variables...?

Try pairs()

(I like the corrplot function better!)

```
# way too many tiny plots!

pairs(x   = concrete, # do everything in the dataframe
      pch = ".")       # plot dots (the default is circles)
```




(Obviously the more variables in your dataframe the messier it gets!)

## 6.3. Creating our linear model and "calibrating" it

We weren't all that thrilled with the correlation between these components and strength but let's go ahead and demonstrate a regression.

But let's move on and create a regression model from this.

Here we will use the lm() (linear model) function from the MASS package.

For the regression formula

$\hat{y}(x) = \alpha_0 + \alpha_1\ x$

or

$\widehat{Strength}(concrete) = \alpha_0 + \alpha_1\ concrete$

the "prototype" (formula) for the function is written as …

"Y ~ X" (with the y-intercept implicit in the formula… you don't put it in but it'll be there when you're done.)

The above syntax is works like this….

Dependent Variable [~ is a function of ] Independent Variable [and any other parameter you need gets added with a plus]

If this were a $\hat{y}(x) = \alpha_0 + \alpha_0\ x^3$, then the prototype for the function would be y ~ x^3

This will hopefully make more sense as we continue!

*(lm and similar linear regression functions don't play well with units.)*

```
linear_model.S_v_c =  lm(formula = Compressive_Strength_28dy ~ Cement, # your formula y ~ x
                         data    = concrete)                          # the data frame
```

Let's see what we have… This summary command will provide the details of the lm() function's important results

For us we want to see the Y-Intercept [the (Intercept) under "Estimate"] and the slope that goes with our independent value ("Concrete" under "Estimate")

The Standard Error of the Estimate is there (Residual Standard Error) as is the Coefficient of Determination (Multiple R-squared)

We'll talk about a few of the other features when we do the larger multivariate regression

```
summary(object = linear_model.S_v_c)
```

```
##
## Call:
## lm(formula = Compressive_Strength_28dy ~ Cement, data = concrete)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.134  -5.313   0.832   5.155  17.968
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 25.85676    2.15022      12  < 2e-16 ***
## Cement       0.04429    0.00885       5  2.4e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7 on 101 degrees of freedom
## Multiple R-squared:  0.199,  Adjusted R-squared:  0.191
## F-statistic:   25 on 1 and 101 DF,  p-value: 2.38e-06
```

In the above output, the asterisk identify the most significant independent variables. Here it's trivial even though this is a terrible relationship between cement and strength. Later we will use all of our available independent variables and the use of these asterisks will become more important.

Want to plot it?

Good news?

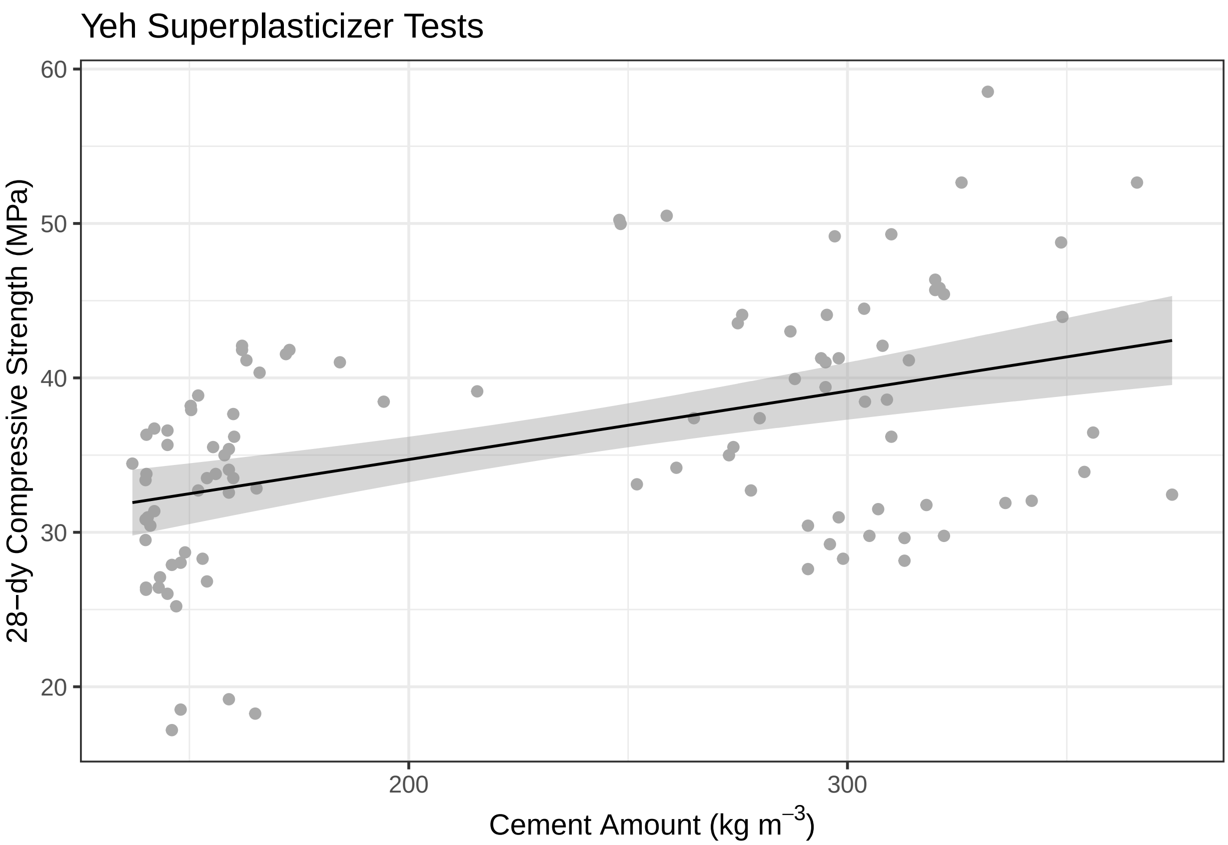
Like Excel, you have some automated features to give you quick satisfaction and happiness. More still, it will give you confidence limits.

For this we use an extension to the graphics package called geom_smooth()

```
# Making a simple X-Y scatterplot and adding a regression to it

ggplot(data = concrete) +                # invoke graphics environment using a given dataframe

  theme_bw( ) +                           # changing the plotting theme

  aes(x      = Cement,                       # x-value
      y      = Compressive_Strength_28dy) +  # y-value

  ggtitle("Yeh Superplasticizer Tests") +    # Custom Title

  xlab(expression('Cement Amount (kg m'^-3*")")) +    # x-label
  ylab("28-dy Compressive Strength (MPa)")      +    # y-label
```

```
  geom_point(colour="darkgrey") +  # plot points
  geom_smooth(method  = "lm",     # use a simple linar model
              formula = y ~ x,    # lm-style formula
              se      = TRUE,     # splay Confidence Intervals
              level   = 0.95,     # Confidene Level to Map Out
              colour  = "black", # regression line color
              size    = 0.5)      # line thickness
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
```




The line here looks like a positive correlation between the cement amount and the resulting strength.

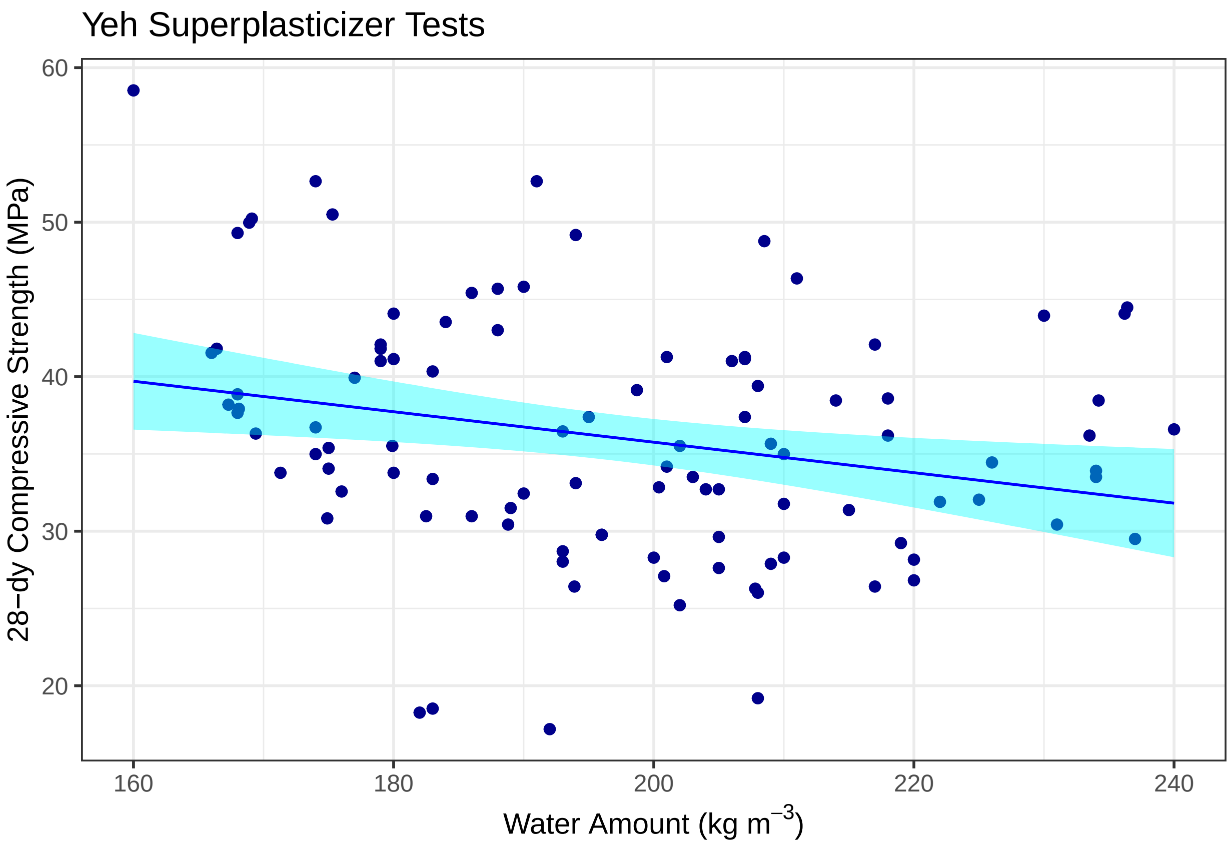
Let's try water:

```
# getting the linear model


linear_model.S_v_w =  lm(formula = Compressive_Strength_28dy ~ Water, # your formula y ~ x
                         data     = concrete   )                           # the data frame

summary(linear_model.S_v_w)
```

```
##
## Call:
## lm(formula = Compressive_Strength_28dy ~ Water, data = concrete)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -19.359  -5.451  -0.986   4.690  18.825
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  55.4824     7.3978    7.50  2.5e-11 ***
## Water        -0.0986     0.0373   -2.64   0.0096 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.6 on 101 degrees of freedom
## Multiple R-squared:  0.0646, Adjusted R-squared:  0.0554
## F-statistic: 6.98 on 1 and 101 DF,  p-value: 0.00956
```

```
# Making a simple X-Y scatterplot and adding a regression to it

ggplot(data = concrete) +                # invoke graphics environment using a given dataframe

  theme_bw( ) +                          # changing the plotting theme

  aes(x      = Water,                     # x-value
      y      = Compressive_Strength_28dy) +  # y-value

  ggtitle("Yeh Superplasticizer Tests") +    # Custom Title

  xlab(expression('Water Amount (kg m'^-3*")")) +  # x-label
  ylab("28-dy Compressive Strength (MPa)")      +   # y-label

  geom_point(colour="darkblue") +  # plot points

  geom_smooth(method  = "lm",    # use a simple linar model
              formula = y ~ x,   # lm-style formula
              se      = TRUE,    # splay Confidence Intervals
              level   = 0.95,    # Confidene Level to Map Out
              colour  = "blue",  # regression line color
              fill    = "cyan",  # NEW: fill for confidence limits
              size    = 0.5)     # line thickness
```




Looking up back the tables none of the variables

## 7. Multivariate Linear Regression

And now we're going to do something about that!

We're now going to use not just one independent variable… but all 7 of them!

The good news is that it follows the same form as the simple linear regression. This time we string along all of our independent variables with in our formula prototype.

Our formula now has multiple independent values but still follows the same style of solution…

$\hat{y}(\mathbf{x}) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_2 x_3 + \ldots + \alpha_n x_n$

```
linear_model.S_v_all <- lm(data     = concrete,                            # your data frame
                           formula = Compressive_Strength_28dy ~ Cement +  # your formula
                                                                 Slag +
                                                                 Fly_Ash +
                                                                 Water +
                                                                 Superplasticizer +
                                                                 Fine_Aggregates +
                                                                 Coarse_Aggregates)
```

And here are these results…

```
summary(object = linear_model.S_v_all)
```

```
##
## Call:
```

```
## lm(formula = Compressive_Strength_28dy ~ Cement + Slag + Fly_Ash +
##     Water + Superplasticizer + Fine_Aggregates + Coarse_Aggregates,
##     data = concrete)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -5.841 -1.706 -0.283  1.299  7.942
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       139.7815    71.1013    1.97   0.0522 .
## Cement              0.0614     0.0228    2.69   0.0084 **
## Slag               -0.0297     0.0318   -0.94   0.3520
## Fly_Ash             0.0505     0.0232    2.18   0.0316 *
## Water              -0.2327     0.0717   -3.25   0.0016 **
## Superplasticizer    0.1031     0.1346    0.77   0.4453
## Fine_Aggregates    -0.0391     0.0288   -1.36   0.1783
## Coarse_Aggregates  -0.0556     0.0274   -2.03   0.0455 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.6 on 95 degrees of freedom
## Multiple R-squared:  0.897,  Adjusted R-squared:  0.889
## F-statistic:  118 on 7 and 95 DF,  p-value: <2e-16
```

Our regression coefficients are still here under the "Estimate" column as are our Standard Error of our Estimate and our Coeff of Determination.

Also we can now take a good look at those asterisks at the end of line with the parameter coefficients. These can explain which independent variables do the heaviest lifting in our regression. The more asterisks, the more important the dependent variable is to the larger multivariate regression. Here, we can see that the Cement and Water are doing most of the "work" in fitting our suite of independent variables to our dependent variable of Compressive Strength.

Finally there is the P parameter for which the smaller it is, the better we can say that the relationship that we've made with our regression represents our dependent variable.

Now… on to looking at our results.

Here is where viewing the results of the regression is tricky.

We have 7 independent variables but we'd like to see the impact of the fit if all 7 variables on our strength

When I do this I like to plot the true y value against my regression y(x1,x2,x3,..)

So to do this I will take the fitted values of y and plot them against the original values of y

Getting the fitted values is easy.

I'm using the get_regression_points function which adds the modeled "y-hat" value to the dataframe of all of the other values get_regression_points() function.

The fitted version is the dependent variable w/ a "_hat"" at the end

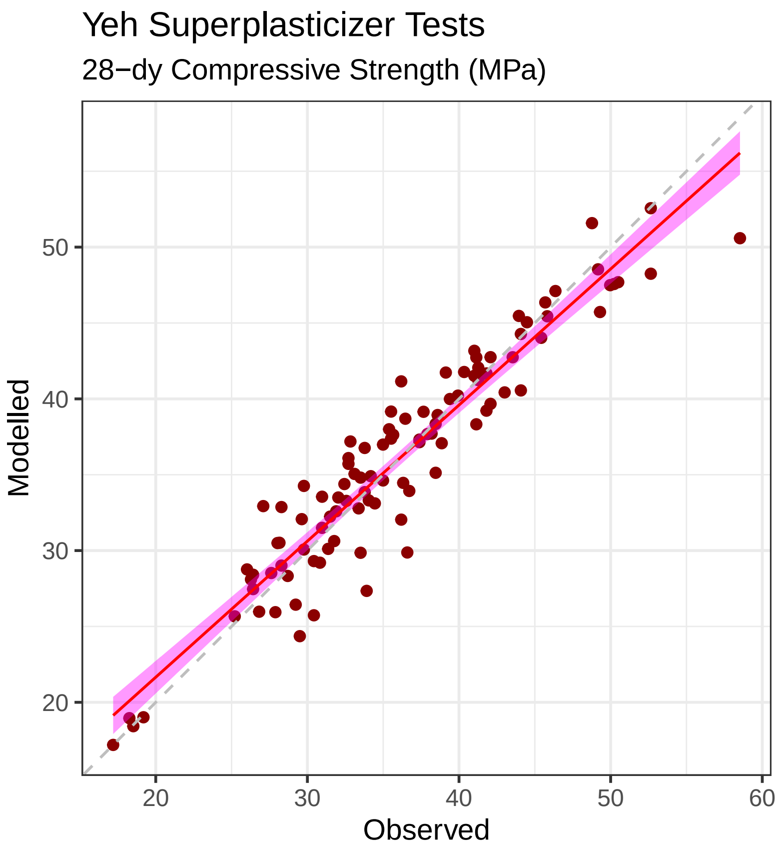
```r
fitted.S_v_all = get_regression_points(model = linear_model.S_v_all)

print(fitted.S_v_all)
```

```
## # A tibble: 103 x 11
##       ID Compressiv~1 Cement  Slag Fly_Ash Water Super~2 Fine_~3 Coars~4 Compr~5
```

```
##    <int>        <dbl>  <dbl> <dbl>   <dbl> <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##   1     1         33.9    354     0       0   234       6     691     959    27.3
##   2     2         33.5    154   141     181   234      11     683     797    29.9
##   3     3         42.1    162   148     190   179      19     741     838    39.7
##   4     4         28.7    149   109     139   193       6     780     892    28.3
##   5     5         44.1    276    90     116   180       9     768     870    40.6
##   6     6         45.8    321     0     164   190       5     774     870    45.4
##   7     7         26.4    143   131     168   217       6     672     891    27.5
##   8     8         37.4    280    92     118   207       9     679     883    37.3
##   9     9         35.7    145   177     227   209      11     715     752    37.6
## 10    10         44.0    349     0     178   230       6     721     785    45.5
## # ... with 93 more rows, 1 more variable: residual <dbl>, and abbreviated
## #   variable names 1: Compressive_Strength_28dy, 2: Superplasticizer,
## #   3: Fine_Aggregates, 4: Coarse_Aggregates, 5: Compressive_Strength_28dy_hat
```

And finally we can plot our actual vs modeled values. (I'm adding a trend line)

```r
# Making a simple X-Y scatterplot and adding a regression to it

ggplot(data = fitted.S_v_all) +           # invoke graphics environment using a given dataframe

  theme_bw( ) +                           # changing the plotting theme

  aes(x      = Compressive_Strength_28dy,    # x-value
      y      = Compressive_Strength_28dy_hat) +  # y-value

  ggtitle("Yeh Superplasticizer Tests",
          subtitle = "28-dy Compressive Strength (MPa)") +    # EDITED: Custom Title now with a subtitl

  ylab("Modelled")     + # y-label
  xlab("Observed")     + # x-label

  geom_point(colour="darkred") +  # plot points

  geom_smooth(method  = "lm",       # use a simple linar model
              formula = y ~ x,      # lm-style formula
              se      = TRUE,       # display Confidence Intervals
              level   = 0.95,       # Confidene Level to Map Out
              colour  = "red",      # regression line color
              fill    = "magenta", # fill for confidence limits
              size    = 0.5)  +     # line thickness

  geom_abline(slope     = 1,        # NEW: add a very simple line
              intercept = 0,        #  (for a 1:1 reference)
              color     = "grey",
              linetype  = "dashed") +

  coord_fixed(ratio = 1)            # NEW: make the aspect ratio
```




```
#   (I like my plots square)
```

And here we have a nice plot showing our true vs predicted values.

## 8. Regression Quality Metrics

And to close things off, we can do some general error metrics that may be useful..

First, the Mean Squared Error (MSE) or Bias… (if we are too high or too low)

$BIAS = MSE = \frac{1}{N}\sum_{i=1}^{n}[\hat{y}(\vec{x_i}) - y_i] = \overline{[\hat{y}(\vec{x_i}) - y_i]}$

```
# Calculate Bias (MSE)

bias = mean(fitted.S_v_all$Compressive_Strength_28dy_hat -
                fitted.S_v_all$Compressive_Strength_28dy)

print(str_c(" Mean Squared Error (MSE) or Bias: ", bias))
```

```
## [1] " Mean Squared Error (MSE) or Bias: 2.91262135922143e-05"
```

For a linear or multivariate regression the average of our residuals (the difference between each observation and prediction) *should* be zero.

The root mean squared error (RMSE) is shown here. It shouldn't be zero since the residuals are squared before summing them up. We technically should use the standard error of the estimate, but RMSE remains a common error metric. We can always do both. The standard error of the estimate takes into account the degrees of freedom which which now includes all of the independent variables (p). We can get the standard error of the estimate from our

$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{n}[\hat{y}(\overrightarrow{x_i}) - y_i]^2} = \sqrt{\overline{[\hat{y}(\overrightarrow{x_i}) - y_i]^2}}$

$s_e$ or $s_{y/x} = \sqrt{\frac{1}{N-p-1}\sum_{i=1}^{n}[\hat{y}(\overrightarrow{x_i}) - y_i]^2}$

```
# Calculate RMSE

rmse = sqrt(mean( (fitted.S_v_all$Compressive_Strength_28dy_hat -
                   fitted.S_v_all$Compressive_Strength_28dy)^2)  )

print(str_c("     Root Mean Squared Error (RMSE): ",
            rmse))
```

```
## [1] "     Root Mean Squared Error (RMSE): 2.50527978593714"
```

```
print(str_c("Standard Error of the Estimate (se): ",
            summary(linear_model.S_v_all)$sigma))  # you have to dig for this one!
```

```
## [1] "Standard Error of the Estimate (se): 2.60865763395229"
```

And finally our correlation coefficient (which is basically our coefficient of determination before the “R” is “squared”)

```
# Get The Unadjusted Correlation Coefficient

r = cor(x = fitted.S_v_all$Compressive_Strength_28dy,     # the x-value
        y = fitted.S_v_all$Compressive_Strength_28dy_hat, # the y-value
        method = "pearson"                                # method of correlation
        )

print(str_c("                        correlation coefficient (r): ", r))
```

```
## [1] "                       correlation coefficient (r): 0.94701611900088"
```

```
print(str_c("                  coefficient of determination (r²): ", r^2,
                                                                " ",
                               summary(linear_model.S_v_all)$r.squared))
```

```
## [1] "                 coefficient of determination (r²): 0.89683952964749 0.89683760981401"
```

```
print(str_c("adjusted coefficient of determination (Adjusted r²): ",
             summary(linear_model.S_v_all)$adj.r.squared))
```

```
## [1] "adjusted coefficient of determination (Adjusted r²): 0.889236170537147"
```

# 9. Closing

And with that, we’re done… Once again, this exercise demonstrates a lot of tricks just to show how you can use R for various statistics. You may not use all of them in your encouters with R for linear or multivariate regression or even at all, but you may be able to cannibalize some of the tricks here for other applications.